



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**

English Version

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)

Contents

List of Figures	III
List of Tables	V
List of Listings	VII
Acronyms	IX
Glossary	XIII
1 Introduction	3
2 Basics	5
2.1 Internet Reference Model	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 Chosen Security Procedures	8
2.4.1 API-Keys	8
2.4.2 OAuth 2.0	9
3 Problem Definition	13
3.1 Mandatory Requirements	13
3.2 Optional Requirements	13
4 Analysis	15
4.1 Data management	15
4.2 Target system	15
4.2.1 Operating System	15
4.2.2 Log-in options	15
4.2.3 Possible frameworks	16
4.3 IT Grundschutz Kompendium	17
4.3.1 APP.3.1	18
4.4 OWASP Top 10	22

5	Realization	27
5.1	Overall structure	27
5.1.1	Authorization server	27
5.1.2	Web service	28
5.1.3	Tests	30
5.1.4	Demo Anwendung	30
5.2	Safety Aspects	31
5.3	Usage	32
6	Results and Conclusions	33
7	Summary and Outlook	35
	Bibliography	37
A	Figures	39
B	Tables	47
C	Listings	51
C.1	Authorization Server	51
C.2	Webservice	53
C.3	WebserviceTests	57
C.4	Demo Application	61

List of Figures

2.1	OAuth 2.0 Abstract Protocol Flow	10
2.2	OAuth 2.0 Refreshing an Expired Access Token	12
A.1	Layer model	39
A.2	Comparison of request times	40
A.3	Sequence diagram of a request	41
A.4	Pseudo diagram of the relevant database tables	42
A.5	HTTP methods p.1	43
A.6	HTTP methods p.2	44
A.7	HTTP methods p.3	45

List of Tables

B.1	Cross reference table APP.3.1	47
B.2	.NET Core 2.1 vs. .NET Framework	48
B.3	IT security requirements table	49

List of Listings

C.1	Program.cs in Authorization Server	51
C.2	Startup.cs in Authorization Server	51
C.3	Config.cs in Authorization Server	52
C.4	Program.cs in Webservice	53
C.5	ConfigureServices in Webservice	53
C.6	Configure in Webservice	54
C.7	DataNotFoundException in Webservice	55
C.8	ApiControllerBase.cs in Webservice	56
C.9	CountriesController.cs in Webservice	56
C.10	AuthorizedClientFixture.cs in WebserviceTests	57
C.11	CountriesSQLTest.cs in WebserviceTests	58
C.12	CountriesWebTestAuthorized.cs in WebserviceTests	59
C.13	CountriesWebTestUnauthorized.cs in WebserviceTests	60
C.14	Application.cs in demo application	61
C.15	Request token in demo application	62
C.16	Requesting data in demo application	63

Acronyms

API	Application Programming Interface <i>Glossar:</i> API , XV , 8 , 9 , 13 , 28 , 40
ARINC	Aeronautical Radio Incorporated <i>Glossar:</i> AR-INC , 29
BSI	Federal Office for Information Security (ger.: Bundesamt für Sicherheit in der Informationstechnik) 13 , 17 , 19 , 21
CRUD	Create Read Update Delete 6 , 13 , 14 , 35
DLL	Dynamic Link Library <i>Glossar:</i> DLL , 15
FMS	Flight Management System <i>Glossar:</i> FMS , 3 , 33
FTP	File Transfer Protocol 6 , 39
GUI	Graphical User Interface 16 , 30
HATEOAS	Hypermedia as the Engine of Application State 13
HTML	Hypertext Markup Language <i>Glossar:</i> HTML ,
HTTP	Hypertext Transfer Protocol III , XV , 3 , 6 , 7 , 8 , 14 , 18 , 28 , 29 , 30 , 32 , 39 , 41 , 43 , 44 , 45
HTTPS	Hypertext Transfer Protocol Secure 8 , 18 , 27 , 39
ICAO	International Civil Aviation Organization <i>Glossar:</i> ICAO , 30
ICMP	Internet Control Message Protocol 5
IEEE	Institute of Electrical and Electronic Engineers <i>Glossar:</i> IEEE , 5 ,
IETF	Internet Engineering Task Force <i>Glossar:</i> IETF , 7 , 9

IIS	Internet Information Services <i>Glossar:</i> IIS , 17
IP	Internet Protocol 5 , 25 , 39
ISO	International Standards Organisation 5 , 39
JDK	Java Development Kit <i>Glossar:</i> JDK , 24
JSON	JavaScript Object Notation <i>Glossar:</i> JSON , 29 , 30 , 32
JWT	JSON Web Token <i>Glossar:</i> JWT , 23 , 32 , 41
LFID	Flight Information Database (ger.: Luftfahrt Informations Datenbank) 3 , 13 , 15 , 32 , 35 , 41
LINQ	Language Integrated Query <i>Glossar:</i> LINQ , 16 , 29 , 31 , 49
MAC	Message Authentication Code <i>Glossar:</i> MAC , 8
ORM	Object Relational Mapping <i>Glossar:</i> ORM , 23
OSI	Open Systems Interconnection 5 , 39
OWASP	Open Web Application Security Project 3 , 4 , 21 , 22 , 23
REST	Representational State Transfer 3 , 8 , 13 , 18 , 48
SID	Standard Instrument Departure <i>Glossar:</i> SID ,
SMTP	Simple Mail Transfer Protocol 6 , 39
SQL	Structured Query Language <i>Glossar:</i> SQL , XIV , XV , 15 , 16 , 23 , 28 , 30 , 31 , 32 , 33 , 40 , 41
SSL	Secure Sockets Layer 7
SSMS	SQL Server Management Studio <i>Glossar:</i> SSMS , 15
SSRF	Server-Side-Request-Forgery 25
STAR	Standard Terminal Arrival Routes <i>Glossar:</i> STAR ,
TCP	Transmission Control Protocol 5 , 6 , 39
TLS	Transport Layer Security 3 , 7

UDP	User Datagram Protocol 5
URI	Uniform Resource Identifier <i>Glossar:</i> URI , XV , 6 , 25
URL	Uniform Resource Locator <i>Glossar:</i> URL , XV , 22 , 25 , 30 , 32
URN	Uniform Resource Name <i>Glossar:</i> URN , XV ,
WAF	Web Application Firewall <i>Glossar:</i> WAF , 21
WPF	Windows Presentation Foundation <i>Glossar:</i> WPF , 17
ZSimNav	Zentrum für Simulations- und Navigationsunterstützung Fliegende Waffensysteme der Bundeswehr 3

Glossary

Aeronautical Radio Incorporated (ARINC) A – especially in aviation – well-known company, which publishes standards for protocols and data formats

Application Programming Interface (API) Set of commands, protocols and functions, which developers can use to create software to perform general operations. This programming interface thus provides the basis for communication between applications.

Authentication A user submits evidence which is supposed to confirm the user's identity (Assertion of an identity)

Authentification Represents the verification of the claimed [Authentication](#) (verification of identity).

Authorisation After successful [Authentication](#), special rights are granted to the user (granting or denying rights).

Brute Force Often used in the context of computer science, to describe a problem solution where no efficient algorithms are known, but rather all possible solutions are simply tried out

CI/CD pipeline Stands for “Contiguous Integration/Contiguous Delivery” pipeline, which includes several steps that need to be performed to deliver a new software version.

Cipher Suite Collection of cryptographic algorithms, which are used for the encryption of messages

Constructor Injection Methodology used in [Frameworks](#) to create classes or attributes, which in turn are used by the calling class

Cookie Small text files that contain user data stored by the browser

Dynamic Link Library (DLL) Dynamic program library, which is mainly executable under Windows

Flight Management System (FMS) electronic aid for pilots for control and navigation

Framework Provides a reusable structure, which can be used for the development of various programs.

Hypertext Markup Language (HTML) Language for displaying content via a web browser

Institute of Electrical and Electronic Engineers (IEEE) Worldwide professional association of engineers, scientists and technicians, which is mainly known for standardizations and other scientific publications

International Civil Aviation Organization (ICAO) Organization, which has the aim to promote international aviation

Internet Information Services (IIS) Service platform from Microsoft for the provision of data, documents and functions over Internet protocols

Internet Engineering Task Force (IETF) Organization for the improvement and further development of the functioning of the Internet

ISO/OSI reference model Model for the network protocols as a representation via a layered architecture, see also [figure A.1](#) on page 39

Java Development Kit (JDK) Composition of Java compiler, Java debugger and some other development tools, which enable the development of Java applications

JavaScript Object Notation (JSON) Machine-readable language for representing objects. With JavaScript this can be used among other things for an instantiation of new objects.

JSON Web Token (JWT) Token that contains claims about the user. It is presented in an encoded form and can be decoded and validated by applications.

Language Integrated Query (LINQ) Language construct in C# where queries resemble [SQL](#) commands.

Man in the middle attack Attack scenario in which a third party places a system between two communication partners, to be able to intercept sensitive data

Message Authentication Code (MAC) Code for integrity check of a message

Object Relational Mapping (ORM) Technique to store objects from object-oriented programming languages in relational databases

OpenAPI Specification Open specification of an [API](#), which can be used to generate client code in different languages.

Salted Hash Procedure Used as an additional safeguard, e.g. when storing passwords. In this procedure, a randomly generated character string is added to the plaintext before further processing.

Singleton Design pattern where there is only one object of a class, which is called a singleton

SQL Server Management Studio (SSMS) Microsoft program for [SQL](#) database management

SSL certificate Data set, which is issued by a certification authority to companies and / or operators of websites. This contains numerous data, including the name of the issuer, the period of validity, a serial number or even the fingerprint of the encryption.

Standard Instrument Departure (SID) Fixed routes, which an aircraft can fly after takeoff, in order to reach an aeronautical route

Standard Terminal Arrival Routes (STAR) Specified routes, which an aircraft can fly to reach a destination.

Structured Query Language (SQL) Language for defining the structure of a database, as well as queries of the contained data

Uniform Resource Name (URN) Addressing objects without specifying a protocol (Unique and constant reference - name of the resource).

Uniform Resource Identifier (URI) Unique addressing of abstract and physical resources on the Internet
[URI](#): [URLs](#) \cup [URNs](#)

Uniform Resource Locator (URL) Addressing of information objects with definition the access protocol (location of the resource)

Web Application Firewall (WAF) Procedure to protect [Web Services](#) from attacks via [HTTP](#). This creates an additional layer that checks all requests for malicious code.

Web Service Software system, which realizes the machine-to-machine communication via a network¹.

Windows Presentation Foundation (WPF) [Framework](#) for the simple creation of a user interface, which has been available in .NET Core since version 3 and is part of the .NET Framework.

¹Cf. David Booth et al. *Web Services Architecture*. Feb. 2004. URL: <https://www.w3.org/TR/ws-arch/> (visited on 07/06/2022).

Gender Statement

On grounds of better readability there will be no simultaneous usage of the forms male, female and diverse (m/f/d). All phrases are counting equally for all genders.

1 Introduction

A flight simulator contains different components where most of them need to be supplied with special processed data. Here the emphasis is on the simulation of communication and navigation devices and [flight management systems \(FMS\)](#). The logistics for this is realized by the [Flight Information Database \(ger.: Luftfahrt Informations Datenbank\) \(LFID\)](#) system which manages those files in a database. This database itself is managed centrally at [Zentrum für Simulations- und Navigationsunterstützung Fliegende Waffensysteme der Bundeswehr \(ZSimNav\)](#) and can be used by connected simulators. The simulators themselves currently use a specialized format which is outdated since it cannot manage the greater data volume of today. So the idea came up to deliver a navigation and communication data service inside the simulator so that the different components retrieve the data from there. This service shall be web-based and use the [Representational State Transfer \(REST\)](#) architecture. Moreover it must satisfy the requirements of the Bundeswehr regarding its security. A big part of this is the [Authentication](#) and the used transport protocols.

Starting with the chapter [Basics](#) a general description of important terms will be given. Firstly the [Internet Reference Model](#) will be explained following the [Hypertext Transfer Protocol \(HTTP\)](#) and consequently the [Transport Layer Security \(TLS\)](#). Finally two security procedures will be introduced.

The [Problem Definition](#) contains mandatory and optional requirements which were determined for the product at the beginning of this thesis.

Afterwards, the [Analysis](#) of the problem is separated into four segments and then analysed. The [Data management](#) basically contains how and where the data is being made available. The [Target system](#) is mostly analysed for the compatible [Frameworks](#). To ensure transport security the “IT Grundschatz Kompendium”¹ will be discussed followed by the top 10 of the [Open Web Application Security Project \(OWASP\)](#).

The [Realization](#) includes explanations on how the [Web Services](#) are implemented and what was being used to realize the necessary security requirements.

In the chapter [Results and Conclusions](#) all results of this thesis are listed and conclusions are touched on how this product could be improved and developed in the future.

The closing chapter [Summary and Outlook](#) sums up everything and gives an outlook.

The mayor aim of this thesis will be to draft and produce a [REST](#) based [Web Ser-](#)

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

[vice](#) which fulfills the challenges of the Bundeswehr regarding IT security as well as the requirements of the target system (simulator) on which this product needs to run. Additionally the [OWASP](#) top 10 will be brought up, analyzed and considered. In the end the reader should be able to making a decision which security procedure is used on different scenarios as well as (slightly) understand how a [Web Service](#) should be created and implemented with [.NET Core 2.1](#).

2 Basics

2.1 Internet Reference Model

In the following the “basis for the internet” will be explained shortly to be able to classify the following chapters into the context as a whole.

Generally speaking there are existing two models: At first there was the [ISO/OSI reference model](#) which is provided since 1977 by the [International Standards Organisation \(ISO\)](#) for the communication in open networks ([Open Systems Interconnection \(OSI\)](#)). Later on a simplified and newer version was supplied, the [TCP/IP](#) reference model where the name comes from the [Transmission Control Protocol \(TCP\)](#) and from the [Internet Protocol \(IP\)](#). Those so called layered models define a stack of layers which are trying to abstract tasks, decoupling them and thereby simplifying them. As you can see in figure [A.1](#) on page [39](#) those different layers of the two models are compared and examples are given which protocols fit inside which layer.

If you now start counting through the layers from bottom to top, the first layer in the [ISO/OSI](#) reference model is the Physical Layer, which together with the second layer, Data Link Layer, makes up the “host-to-network” Layer in the [TCP/IP](#) reference model. This layer is responsible for ensuring the physical transmission. Examples of this are primarily the Ethernet (cable) and the WLAN 802.11 according to the [IEEE](#) standard, whereby simple error detection procedures are additionally located in it.

The next layer contains the [IP](#) and is thus responsible for addressing and transmitting the packets. However, the order of the packets does not necessarily have to be adhered to. It is also possible for transmission errors to occur, but these cannot be detected by this layer. The Internet or network layer also provides the [Internet Control Message Protocol \(ICMP\)](#), which makes the dispatch of control information – for example change of routing, or status inquiries – possible.

In both models, this is followed by the Transport Layer. This is responsible for communication between two partners and contains two protocols. The [TCP](#) is reliable and connection-oriented, which is why it is used for critical and security-relevant transmissions. The [User Datagram Protocol \(UDP\)](#) is connectionless and unreliable, but therefore faster, which is why it can be used for the transmission of large data streams – such as video streams – whereby the absence of individual smaller packets is not noticeable. The three original layers (Session, Presentation and Application) in the [ISO/OSI](#) reference model have been combined into one layer, the Processing Layer, in the [TCP/IP](#)

reference model. This area is used to provide higher level protocols and application software. These include the [File Transfer Protocol \(FTP\)](#), the [Simple Mail Transfer Protocol \(SMTP\)](#) and also the [HTTP](#).

2.2 HTTP

The first version of the [HTTP](#) – [HTTP/1.0](#)¹ – was published in 1996. Since this time the protocol is primarily used for smaller [Web Services](#) because it is fast, lightweight and simple to implement. To improve the loading time in 1999 there was “[HTTP/1.1](#)”² published and standardized. Now there was a header field called “Connection: keep-alive” which can be used by the client to indicate that the underlying [TCP](#) connection should be kept alive. The version “[HTTP/2.0](#)”³ sped up the transfer again. This version is downward compatible since 2015 and can now be used after an initial request by the client using the header field “Connection: Upgrade”. The added feature is called “multiplexing”. The data transfer is split into streams which allows parallel transmission. With the use of unique IDs the data can be reconstructed.

Each action has a specific method defined by [HTTP](#). For example, each operation described by [Create Read Update Delete \(CRUD\)](#) has an equivalent [HTTP](#) method. The [HTTP](#) methods, and where possible the [CRUD](#) counterparts, are listed in the following:

- **GET** requests a resource which is defined inside the request-[URI](#). (**READ**)
- **HEAD** similar to GET but delivers just the head without the body.
- **POST** transfers data to the server. (**CREATE**)
- **PUT** wants to override data on the server with own data. (**UPDATE**)
- **DELETE** deletes data on the server. (**DELETE**)
- **CONNECT** opens a tunnel to the server for bidirectional communication.
- **OPTIONS** requests an access path from a resource and the further potential communication.
- **TRACE** for testing. The request is sent back from the server.

¹Cf. Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. May 1996. DOI: [10.17487/RFC1945](#). URL: [https://rfc-editor.org/rfc/rfc1945.txt](#).

²Cf. Roy T. Fielding and Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. June 2014. DOI: [10.17487/RFC7230](#). URL: [https://rfc-editor.org/rfc/rfc7230.txt](#).

³Cf. Mike Belshe, Roberto Peon, and Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 2015. DOI: [10.17487/RFC7540](#). URL: [https://rfc-editor.org/rfc/rfc7540.txt](#).

2.3 TLS/SSL

In order to make the transmission of data safe and secure for example through [HTTP](#) and to protect against access by third parties, the [TLS](#) protocol is used. [Secure Sockets Layer \(SSL\)](#) is often used synonymously, but officially it was renamed to [TLS](#) by “The TLS Protocol Version 1.0”⁴(1999) published by the [Internet Engineering Task Force \(IETF\)](#). However, this version is now considered invalid and should therefore no longer be used. The versions currently in use are 1.2⁵ and 1.3⁶. Basically, two internal protocols are used for security, which are explained below.

2.3.1 TLS Handshake Protocol

All [TLS](#) handshakes use an asymmetric encryption method. The protocol goes through the following steps:

1. The client sends a “Hello” message to the server and thus initiates the handshake. The message consists of the supported [TLS](#) version, the supported [Cipher Suites](#) and a random arrangement of bytes which are referred to as “Client Random”.
2. The response from the server also known as the “Server Hello” contains the [SSL certificate](#), the selected [Cipher Suite](#) and a “Server Random” which is itself a byte sequence, too.
3. The client then checks the [SSL certificate](#) at the certification authority.
4. The “Premaster Secret” and a randomly generated byte sequence is encrypted using the public key contained in the [SSL certificate](#). This message is sent to the server and can only be decrypted it with the server’s private key.
5. The server now decrypts the “Premaster Secret” whereupon both partners generate a session key from “Client Random”, “Server Random” and “Premaster Secret”.
6. Now the client and server each send a “Done” message which both are encrypted with the session key.
7. The handshake is complete and further communication is continued with the session key.

⁴Cf. Christopher Allen and 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>.

⁵Cf. Eric Rescorla and 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>.

⁶Cf. 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>.

2.3.2 TLS Record Protocol

The record protocol is used for the actual data transmission and is independent from the [TLS Handshake Protocol](#). First, the data, that is to be sent, is divided into manageable blocks which can optionally be compressed. Each message is then assigned a message authentication code ([Message Authentication Code \(MAC\)](#)) which is used for integrity checking. Finally this data will be encrypted with the negotiated key inside the [TLS Handshake Protocol](#). If the data is now fully encrypted and sent, the receiving side traverses the protocol backwards one more time to be able to show the data again.

2.4 Chosen Security Procedures

For the identification of the communication partners and thereby also the superior coverage of [Application Programming Interface \(API\)s](#) it is necessary to understand the difference between the terms [Authentication](#), [Authentification](#) and [Authorisation](#). Talking about [Authentication](#) it is meant that the user proves his identity to the service/system. For this step the proofs are usually id cards or in the digital world for example passwords, patterns or fingerprints. The following step, the [Authentification](#), those verification documents are going to be verified or falsified by an independent authority. Finally this verification leads to the last step the [Authorisation](#) where the system/service grants or denies access rights to the user according to the outcome of the [Authentification](#).

Hereinafter two procedures will be described that could be used as security. Aspects of these will be categorized based on the vocabulary just explained.

2.4.1 API-Keys

The [API](#) keys are suitable for easy [Authorisation](#) of a client and are often used by programs that do not require direct access to sensitive data. For this purpose, a common key is deposited with both partners, with which the communication partner can be identified. This key should be uniquely assigned and in the case of [REST](#) and [HTTP](#) it is handed over either via the request query in a POST request or as a header or [Cookie](#) in a GET request. Based on the key, the [Web Service](#) checks the approved or blocked resources and evaluates the request based on the [Authorisation](#). This procedure is weak against [Man in the middle attacks](#). This combined with other factors means that using [Hypertext Transfer Protocol Secure \(HTTPS\)](#) is required to prevent the key from being intercepted and used.

2.4.2 OAuth 2.0

To create an open standard for [API](#) access delegation, an initiative was started in November 2006 with the aim of developing the OAuth protocol in version 1.0. An open standard was desired because different companies like for example Twitter had already developed their own methods. The best of it should be combined in order to simplify this delegation. The delegation of [Authentication](#) is gaining necessity as more and more [Web Services](#) and web applications come into existence, many of them requiring user administration. OAuth assists with this by only transferring certain rights to the used application. The version OAuth 1.0 was therefore standardized four years later by the [IETF](#)⁷. However, since this version was difficult to implement and some implementations even had security gaps, version 2.0 was released in October 2012⁸. This also had some other advantages and made version 1.0 almost completely obsolete because, among other things, there is no compatibility between those two. Hereinafter only OAuth 2.0 will be discussed.

2.4.2.1 Roles

OAuth 2.0 defines four different roles, which besides other things revolutionizes the classic client-server [Authentication](#) in such a way that the client is now divided into two roles.

Resource Owner

If it is a person, this role is also called “end-user”. This represents an entity that has the ability to grant a third party access to its protected resources.

Resource Server

The Resource Server provides and hosts (protected) data. Additionally it is also able to accept or deny [resource servers](#).

Client

The Term “client” does not require any special characteristics and describes a (web)application which is able to request resources at the [client](#) on behalf of the “end-user”.

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

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

Authorization Server

In order to get [Authorisation](#) tokens for the [authorization server](#), the authorization server is utilized. This can create, send and validate [Token](#).

2.4.2.2 Token

[OAuth 2.0](#) uses tokens in order to protect resources from undesired access. The most common tokens used are called “Bearer-Tokens”.

Access Token

These tokens are used to obtain encrypted credentials of a [access token](#), provide specific spaces and the duration of access for a [access token](#). It is represented as a string and is usually opaque to the [access token](#). In addition, depending on the security guidelines of the server, they can be vary regarding representation and encryption.

Refresh Token

Refresh tokens are optional and – as the name suggests – for renewing the already issued permission. The [refresh token](#) only uses this token in exchange with the [refresh token](#) to get a new [refresh token](#) . This can additionally strengthen the security of this protocol if the lifetime of the [refresh token](#) is short and the [refresh token](#) must frequently request a new [refresh token](#) via the [refresh token](#).

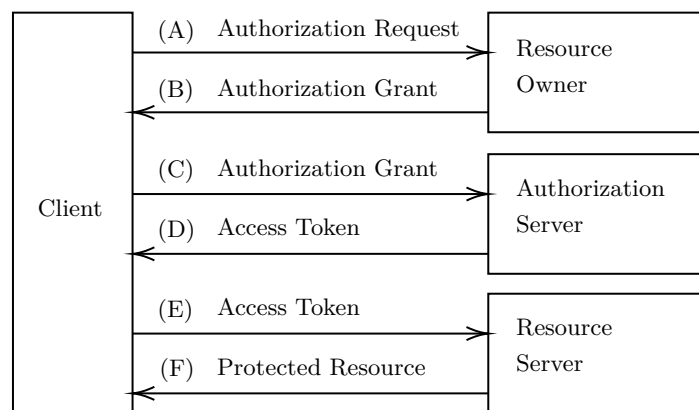


Figure 2.1: OAuth 2.0 Abstract Protocol Flow

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

2.4.2.3 Abstract Protocol Flow

The figure 2.1 on the facing page shows the general abstract flow of an [Authorisation](#) with OAuth 2.0 and is shown as follows:

- (A) The [refresh token](#) requests [Authorisation](#) from the [refresh token](#). This request can be made directly to the [refresh token](#) or even better indirectly through the [refresh token](#) as an intermediary.
- (B) The [refresh token](#) responds with an [authorization permit](#) which can be one of four allowed types, each of which depends on the supported types of the [refresh token](#).
- (C) The [refresh token](#) requests an [refresh token](#) in combination with the previously received [authorization permit](#) to the [refresh token](#).
- (D) The [refresh token](#) authenticates the [refresh token](#), validates the [authorization permit](#) and issues an [refresh token](#).
- (E) The [refresh token](#) requests the resource from the [refresh token](#) and authenticates itself with the [refresh token](#).
- (F) The [refresh token](#) validates the [Token](#) in conjunction with the [refresh token](#) and grants access to the requested resource.

2.4.2.4 Requesting a Refresh Token

In order to be able to reapply for the previously mentioned [refresh token](#), as mostly a period of validity is set, a new [refresh token](#) is requested together with the [refresh token](#), as shown in figure 2.2 on the next page. The procedure is as follows:

- (A) As with the first request for an [refresh token](#), the [refresh token](#) sends an [authorization request](#) to the [refresh token](#).
- (B) The [refresh token](#) responds with both an [refresh token](#) as well as a [refresh token](#).
- (C) The [refresh token](#) requests the protected data from the [refresh token](#) in combination with the [refresh token](#).
- (D) If the [refresh token](#) is valid, the request is answered with the requested data.
- (E) Steps (C) and (D) are repeated until the lifetime of the [refresh token](#) has expired. If the [refresh token](#) knows this, it jumps directly to step (G), otherwise a new request is made.
- (F) Since the [refresh token](#) is now invalid, the [refresh token](#) returns an “Invalid Token Error”.
- (G) The [refresh token](#) requests a new [refresh token](#) from the [refresh token](#) by showing the [refresh token](#).

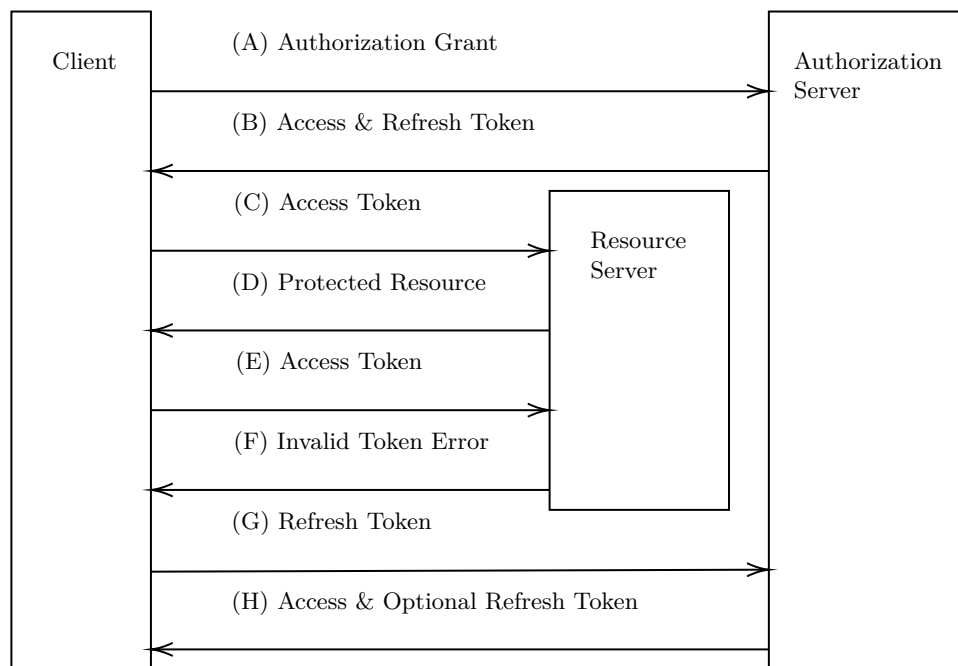


Figure 2.2: OAuth 2.0 Refreshing an Expired Access Token

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

- (H) If the validation of the [refresh token](#) was successful, the [refresh token](#) grants the [refresh token](#) a new [refresh token](#) and optionally a new [refresh token](#).

3 Problem Definition

This thesis shall prove besides other important things that a secure communication between client and [Web Service](#) together with a [Authentication](#) makes sense and is necessary, too. The [Mandatory Requirements](#) are therefore strongly limited other than the [Optional Requirements](#) because theoretically there is much more to be shown and implemented according to the complexity of this topic.

3.1 Mandatory Requirements

This [Web Service](#) has to meet the it security requirements of the [Federal Office for Information Security](#) (ger.: [Bundesamt für Sicherheit in der Informationstechnik](#)) (BSI) and therefore the regulations from the “IT-Grundschutz-Kompendium”¹. There has to be a [REST](#) interface with methods for working with the underlying database. The product in whole is supposed to be fast and reliable and should be able to process numerous requests per second. As the description of a [REST](#) based [Web Service](#) states the [API](#) has to be self discoverable (keyword: [Hypermedia as the Engine of Application State \(HATEOAS\)](#)) and – as far as possible – reflect all [CRUD](#) operations. For testing the [Web Service](#) and displaying the results graphically, besides other things for showing the integration inside the simulator, there has to be a graphical user interface which preferably looks like a real cockpit. Finally all programmes need to be implemented in C# and obviously have to run on the system.

3.2 Optional Requirements

The [LFID](#) system provides data that can be interpreted differently by different programs. This connection to the database is currently still done for each individual program via direct database access and is also not currently used directly in the flight simulator. The solution is to change all programs and flight simulator components, which need this data, in such a way that these must [authorize](#) themselves with the [Web Service](#), in order to be able to request data. The programs that only need read access can then be assigned a different access level through [Authorisation](#) than programs that are actually [authorized](#) to change data. Here it would play naturally also directly with the fact that for each

¹Schildt, *IT-Grundschutz-Kompendium*.

entity all [HTTP](#) methods must be made available, in order to be able to represent all [CRUD](#) operations.

In addition the [Authorization server](#) is so far only a command line program, as well as the [Web Service](#), which could possibly be supported by a graphical user interface, especially for the setup of the rights distribution.

4 Analysis

As far as now the [LFID](#) system with its provided data is not used directly inside a simulator yet. The [Web Service](#) can thus be formed relatively free except the attention to the following points.

4.1 Data management

As the name suggests, the data to be queried from the [LFID](#) system is provided in a database. This is represented as a [Structured Query Language \(SQL\)](#) database and is managed with the [SQL Server Management Studio \(SSMS\)](#). At present administrator access is always set up, which has full access to all data and operations. The database tables discussed here are exemplarily shown in figure [A.4](#) on page [42](#).

4.2 Target system

This [Web Service](#) is to be built primarily for a flight simulator, in which a simulation component has to request the required data from the web service.

4.2.1 Operating System

A stripped-down Windows 10 is used as the operating system, which has all the necessary services and programs installed and all the necessary ports enabled. Thus, the ready-built [Dynamic Link Library \(DLL\)](#)s can be started and used without further ado, since the database was installed beforehand, among other things. As already mentioned, several programs (or services) depend on the data, whereby each service has its own database access and therefore all rights. The installation could be simplified by deploying a container with the database, the [Web Service](#) and the [Authorization server](#) in addition to releasing the ports from it so that all methods can be reached.

4.2.2 Log-in options

The simulator environment is booted once, which also starts all services and programs. The pilots, the flight instructors and any spectators then have access to this environment, which means that a login by a special user during the start-up phase can only be realized with great effort. The security of the [Web Service](#) will therefore have to take place

through the identification of the programs, since in addition also no password input is possible.

4.2.3 Possible frameworks

If possible, [Frameworks](#) that simplify the entire process should be used to standardize the implementation so that, at best, additional programming in different languages is not necessary. Since as few other packages as necessary should be installed on the target system, the programming language C# is an obvious choice. Due to the conditions of older [operating systems](#), the two most current possible C# [Frameworks](#) are presented below.

4.2.3.1 .NET Framework

This [Framework](#) is the most used and oldest concept in Windows to build applications. The Nu-Get package manager can be used to integrate external code, which makes it easy to implement database connectivity, among other things, in a uniform manner. However, microservices are not supported here, making .NET Framework more suitable for programming [Graphical User Interfaces \(GUIs\)](#). In addition, a compiled .NET Framework project is not executable on its own, which is why, since Windows Vista, .NET Framework is installed together with the Windows Updates.

4.2.3.2 .NET Core 2.1

.NET Core 2.1 was released by Microsoft on 05/30/2018 for use in Visual Studio 2017 version 15.7. This provides a [Framework](#) that functions as an Open Source project for Windows, macOS and Linux. Developers can use it to program cross-platform applications and, with the NuGet package manager, include and use a range of existing code, whether this comes from Microsoft or from other developers in this community. For example, code for creating [Web Services](#) and [OAuth 2.0 authentication services](#) can be found there, among other things. For working with an [SQL](#) database, the most suitable package is the “Entity Framework Core” package from Microsoft. which, among other things, simplifies and standardizes the use and connection also with [Language Integrated Query \(LINQ\)](#).

4.2.3.3 Comparison of both frameworks

So both previous [Frameworks](#) are theoretically possible and usable. However, there are some aspects to be considered, which simplify the decision for a [Framework](#). For both

Web Services (data and authentication service), **.NET Core 2.1** was now used, as the creation and hosting is easier here. In **.NET Framework**, for example, it is difficult to create self-hosting **Web Services** because Microsoft provides **Internet Information Services (IIS)** to handle such things. However, if one wants to create a self-hosting program, as is the case in this work, this must be realized in **.NET Core 2.1**. On the other hand, it is difficult to create graphical applications in **.NET Core 2.1**. It is therefore easier and almost more intuitive to fall back on the concepts of **.NET Framework**, such as **Windows Presentation Foundation (WPF)** or Windows Forms. The demo program for testing the two **Web Services** is therefore implemented in **.NET Framework**. Finally, it is worth mentioning that Microsoft introduced the “**.NET Standard**” as a link between the two **Frameworks** to enable the use of common libraries. This is also used here, since the elements of the database are obviously the same in both programs, the **Web service** and the demo program. Furthermore, in table **B.2** on page **48**, some differences are listed, which also have contributed to the previous decision.

4.3 IT Grundschutz Kompendium

The IT-Grundschutz-Kompendium is a document published by the **BSI**, which identifies security risks in the area of information security and presents preventive measures against them. In general, this document defines three basic values of information security¹ that must be protected:

Loss of availability:

A software system works only in the presence of certain data. If this data is not available, or is only available to a limited extent, this can lead to the impairment of simple tasks or even to the complete shutdown of work processes.

Loss of confidentiality of information:

Every company must handle personal data of customers and users confidentially so that no damage is done to privacy. This includes, among other things, data on sales, marketing or research.

Loss of integrity (correctness of information):

In order to be able to avoid faulty orders, etc., it is essential to ensure, that all stored data is correct and unaltered. The so-called “digital identity” is also playing an increasingly important role, which means that the threat of false declarations or identity forgery is growing.

¹Vgl. Schildt, *IT-Grundschutz-Kompendium*.

The document is divided into building blocks to eliminate these and other risks, each of which is divided into:

1. Description
2. Hazard situation
3. Requirements
4. Further information
5. Annex: cross reference table for elementary hazards

4.3.1 APP.3.1 Web applications and web services

In the following, the module “APP.3.1 Web applications and [Web Services](#)”, which is relevant for this work, is analyzed according to this pattern.

4.3.1.1 Description

Here, a [Web Service](#) is once again delineated and defined in the introduction. Based on this module, a [Web Service](#) is an application that uses [HTTP](#) or [HTTPS](#) to provide resources. In the vast majority of cases, this service is not controlled directly by a user, but is accessed by other applications. This module also deals exclusively with [Web Services](#) with a [REST](#) interface².

4.3.1.2 Hazard situation

The hazard situation of this building block is subdivided once again. In the following relevant hazards are addressed³. Die Gefährdungslage dieses Bausteins ist noch einmal unterteilt.

Insufficient logging of safety-relevant events

The logging of security-relevant events is necessary in order to later determine the causes of a particular event and thus to be able to trace vulnerabilities, critical errors, or unauthorized changes.

Disclosure of security-related information in web applications and web services

When delivering [Web Services](#), care must be taken not to disclose any security-relevant data, i.e., information about programs, systems, or versions used.

²Cf. Schildt, *IT-Grundschatz-Kompendium*, APP.3.1 S. 1.

³Cf. *ibid.*, APP.3.2 S. 1.

Abuse of a web application through automated use

A [Web Service](#) can be misused by automated queries and thus potentially disclose user data. This could allow attackers to collect valid usernames and then use them to gain access to the application by trying passwords multiple times.

Insufficient authentication

Most often, role profiles are created to give users the possibility to reach certain resources which cannot be reached by other profiles. If the [authentication](#) of the users is insufficient, an attacker could gain easy access and in the worst case even access the user data, if they are also accessible via this role.

4.3.1.3 Requirements

The requirements for [Web Services](#) are again divided into three areas and, in addition, responsibilities should always exist so that an office is always “basically responsible” and optionally other responsibilities can also be assigned.

Basic requirements

Above all, the basic requirements MUST be met according to the [BSI](#)⁴.

APP.3.1.A1 Authentication

[Web Services](#) MUST always be designed in such a way, that blocked resources can only be accessed via [Authentication](#). The method for this SHOULD be logged, where the business must also set a maximum limit for failed login attempts⁵.

APP.3.1.A4 Controlled inclusion of files and content

This requirement does not apply to this [Web Service](#) because no data is uploaded.

APP.3.1.A7 Protection against unauthorized automated use

It HAS TO BE be guaranteed, that the [Web Service](#) is protected against [unauthorized](#), automated use, while the behavior of the actions on authorized users HAS also TO BE taken into account⁶.

⁴Cf. [ibid.](#), APP.3.1 S. 3.

⁵Cf. [ibid.](#), APP.3.1 S. 3.

⁶Cf. [ibid.](#), APP.3.1 S. 3.

APP.3.1.A14 Protection of confidential data

Confidential data, such as credentials, **MUST** be secured on the server side using [Salted Hash Procedure](#) to protect them from [unauthorized](#) access.

Standard requirements

Standard requirements **SHOULD** basically be fulfilled together with the Basic requirements⁷.

APP.3.1.A8 System Architecture [Procurement Agency]

Security guidelines and concepts **SHOULD** already be considered during the planning of a [Web Services](#)⁸.

APP.3.1.A9 Procurement of web applications and web services

As a supplement to the other requirements, the IT-Grundschutzkompendium provides a list which additional properties **SOLLTEN** be considered when procuring a [Web Services](#)⁹:

- secure input validation and output encoding
- secure session management
- secure cryptographic procedures
- secure authentication methods
- secure procedures for server-side storage of access data
- suitable authorization management
- sufficient logging capabilities
- regular security updates by the software developer
- protection mechanisms against widespread attacks on web applications and web services
- access to the source code of the web application or web service

APP.3.1.A11 Secure connection of background systems

Background systems on which functions and data are outsourced, **SHOULD** be accessible only via defined interfaces. In addition, communication for cross-network and cross-site applications **SHOULD** be [authenticated](#) and encrypted¹⁰.

⁷Cf. Schildt, *IT-Grundschutz-Kompendium*, APP.3.2 S. 4.

⁸Cf. *ibid.*, APP.3.1 S. 4.

⁹Cf. *ibid.*, APP.3.1 S. 4.

¹⁰Cf. *ibid.*, APP.3.1 S. 4.

APP.3.1.A12 Secure configuration

Access or requests to resources and methods of a [Web Services](#) SHOULD be restricted in such a way, so that communication is only possible via predefined paths. All other methods and resources that are not required SHOULD be disabled.

APP.3.1.A21 Secure HTTP configuration for web applications

The following response headers SHOULD always be set¹¹:

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

Moreover, these SHOULD always be made as restrictive as possible.

APP.3.1.A22 Penetration testing and revision

Penetration testing and auditing SHOULD be performed on a regular basis to check for security issues and breaches. The results SHOULD be logged¹².

Requirements with increased need for protection These requirements must be met if the protection requirement exceeds the protection level corresponding to the state of the art. This is determined by means of an individual analysis.¹³

APP.3.1.A20 Use of Web Application Firewalls

A [Web Application Firewall \(WAF\)](#) SHOULD be used, to increase the security level. The configuration must also be adjusted after each update¹⁴.

4.3.1.4 Further information

In this chapter, reference is made to the [OWASP](#), which is also addressed and explained in section 4.4 on the following page. In addition, the document “Cryptographic Procedures: Recommendations and Key Lengths: BSI TR-02102” is proposed by [BSI](#), which provides information on the use of cryptographic procedures¹⁵.

¹¹Cf. [ibid.](#), APP.3.1 S. 5.

¹²Cf. [ibid.](#), APP.3.1 S. 5.

¹³Cf. [ibid.](#), APP.3.1 S. 5.

¹⁴Cf. [ibid.](#), APP.3.1 S. 5.

¹⁵Cf. [ibid.](#), APP.3.4 S.6.

4.3.1.5 Appendix: Cross-reference table for elementary hazards

A cross reference table (see table B.1 on page 47) is shown here, which elementary hazards¹⁶ can be mitigated or even eliminated by which requirements. The following elementary hazards are dealt with:

G 0.14 Spying out information (espionage)

G 0.15 Wiretapping

G 0.18 Misplanning or lack of adaptation

G 0.19 Disclosure of information requiring protection

G 0.21 Manipulation of hardware or software

G 0.28 Software vulnerabilities or errors

G 0.30 Unauthorized use or administration of devices and systems

G 0.31 Incorrect use or administration of devices and systems

G 0.43 Import of messages

G 0.46 Loss of integrity of information requiring protection

Furthermore, there is a column “CIA” which deals with the basic values of information security as defined on page 17.

4.4 OWASP Top 10

The OWASP is a public community that publishes top-10 lists¹⁷ of the most exploited security vulnerabilities at three-year intervals. These lists are intended to encourage organizations to deploy secure web applications and protect themselves against these threats. The most recent version at this time is the 2021 list. In the following the contained vulnerabilities are enumerated and it is explained how developers can protect applications against them.

A01:2021 - Broken Access Control

The so-called access control errors are gaining a lot in importance compared to 2017, as they have been moved from 5th to 1st place.

If users of an application get more rights than actually intended, this inevitably leads to the release of data, for which they are not authorized. Attackers could achieve this by bypassing access controls, for example, by changing the Uniform Resource Locator

¹⁶see Schildt, *IT-Grundschrift-Kompendium*, Elementary hazards S. 1ff

¹⁷Cf. 2021 OWASPTop10Team. *OWASP Top 10:2021*. 2021. URL: <https://owasp.org/Top10/> (visited on 07/06/2022).

(URL), manipulating the metadata sent along, or reusing or modifying the [JSON Web Token \(JWT\)](#).

Rights, roles and rules must therefore be clearly defined during development, how users can [authenticate](#) themselves and which rights may be assigned to which users. This should also be ensured by logging the events and invalidation of the [JWTs](#).

A02:2021 - Cryptographic Failures

Previously known as “Sensitive Data Exposure” or “Loss of Sensitive Data Confidentiality”, the “cryptographic errors” move to second place on the list.

As a rule, these involve [Man in the middle attacks](#), which aim to break not the encryption itself, but to use the data of the transmission to compromise it. Furthermore, the use of insecure protocols, older encryption methods or no bindingly enforced encryption, the vulnerability of the application can be increased.

This threat can be prevented by avoiding unnecessary storage of sensitive data, disabling the cache for this data, or by using secure transport protocols.

A03:2021 - Injection

This threat has been downgraded from rank 1 to 3.

If the web application or service manages a database connection, “Injection” is a popular attack strategy. Attackers attempt to disclose, compromise or delete data by deliberately exploiting [SQL](#) queries. For example, a successful injection can also be used to impersonate an administrator account, which would of course be fatal.

Correct use of an [Object Relational Mapping \(ORM\) Frameworks](#) can mitigate this threat. In addition, input data should always be checked for disallowed letters, as well as characters and length.

A04:2021 - Insecure Design

The new category “Insecure Design” refers to errors in the architecture and design of applications.

Here, security vulnerabilities of external [Frameworks](#) and design patterns are exploited. The [OWASP](#) thereby attempts to make developers of frameworks aware of security vulnerabilities and to promote secure design patterns.

This can be prevented by using [Frameworks](#) that have been tested multiple times, as well as the development of secure design patterns.

A05:2021 - Security Misconfiguration

Compared to 2017, this category has moved up one place. It includes vulnerabilities in the configuration of firewalls, [Web Services](#) and web applications.

Attackers could try to gain access to the system via standard passwords, or find the compiled classes via folder path listings, then decompile them, in order to find over “Reverse engineering” a safety gap.

Developers can minimize or eliminate such a security hole by uninstalling unnecessary features, [Frameworks](#) or components and delivering a “slimmed down” application. Furthermore, outdated security gaps can be closed via regular updates of the system/[Framework](#) used.

A06:2021 - Vulnerable and Outdated Components

This category is the second most important, as its importance has increased by 3 levels. The use of components with known vulnerabilities is recorded.

A current example of this is the vulnerability of Java’s logging [Framework](#) “Log4j”, which became known on December 10, 2021. The internal structure of the language allows exploits to be made and the host computer to be controlled.

Companies should therefore always be aware of this, which external libraries are used how they work and whether they are up to date. In the case of a logging [Framework](#) or other basic components, it is therefore sometimes advantageous to either write the code itself or to use classes (like e.g. `java.util.logging`) directly from the [Java Development Kit \(JDK\)](#).

A07:2021 - Identification and Authentication Failures

[Authentication](#) errors have lost a lot of their importance.

Attacks on [Authentication](#) are well researched, which means that there are already many scripts that use dictionaries as a data basis for [Brute Force](#) attacks to find out user names and passwords. [Web Services](#) with non-expiring session tokens are also particularly at risk, which can then be used for false [Authentication](#).

If possible, the following topics should therefore be implemented in order to provide authentication with a higher security standard. To minimize the risk of automated attacks, multi-factor [Authentication](#) should be implemented and, in the course of this, all passwords for a newly created user account should also be checked against a list of the 10,000 most popular passwords. In addition, no standard users and/or administrative users should be present in the delivery state, and the same error message should always be returned even if logon attempts fail, so that the user names cannot be guessed.

A08:2021 - Software and Data Integrity Failures

The second new category relates to vulnerabilities in software updates, critical data and [CI/CD pipelines](#) without integrity checks. This includes the outdated category from 2017 “A08:2017 - Insecure Deserialization”.

For example, the deserialization process can be used to execute malicious code on the [Web Service](#) using faulty routines. Furthermore, external code can be installed and executed on an existing system if new packages are not checked and automatically reloaded. Digital signatures or similar mechanisms for validating the packages are therefore required to rule out such risks. In addition, it should be ensured - especially in the context of web applications - that package managers (such as Maven, Gradle or npm) only load code from trustworthy sources.

A09:2021 - Security Logging and Monitoring Failures

The category “A10:2017 - Insufficient Logging & Monitoring” at that time mainly includes insufficient logging of login attempts or incorrect visibility of logging information. Attackers can therefore attempt to guess credentials automatically and with a large number of tests. Furthermore, logs could be tapped and interpreted.

It is therefore imperative to log login attempts and draw the right conclusions from them, such as blocking the [IP](#) address or similar. Furthermore, care must be taken where the logs are written, as this information must not be accessible to outside persons.

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

[Server-Side-Request-Forgery \(SSRF\)](#) is the third new threat in 2021 and refers to the control of the [Web Service](#) via requests that are manipulated by the attacker in such a way that the application executes different code than planned or even discloses protected data.

This can be exploited, for example, by replacing [URL](#) requests with 127.0.0.1 or “localhost” and thus forwarding them to the own lookup interface.

Protective mechanisms against this are rare, since, among other things, not even libraries agree on how to interpret certain [Uniform Resource Identifier \(URI\)s](#). However, depending on the application, either the whitelist or the blacklist approach could be used for validating the [URLs](#).

5 Realization

As stated in the chapters before the [Web Service](#) as well as the [Authorization server](#) are both implemented in [.NET Core 2.1](#) whereas the demo program and all other applications using the [Web Service](#) are implemented in [.NET Framework](#).

5.1 Overall structure

Meanwhile, the developed software system consists of three different components, which must work well together for smooth operation. These programs are needed to perform the [Authentication](#), to query data and finally to evaluate them. In the following the code of the individual components is analyzed and explained, whereby only relevant lines/methods/calls are dealt with. The self-explanatory code fragments remain mostly unmentioned to be able to highlight the important lines/calls, so that the code would be executable nevertheless exactly the same. Additionally, it should be mentioned that [.NET Core 2.1](#) by design requires two classes to create a [Web Service](#). The class `Program.cs` contains the `Main` method, which internally creates the class `Startup.cs`.

5.1.1 Authorization server

The authorization server is the core of the implemented [OAuth 2.0 Authorisation](#). As seen in listing [C.1](#) on page [51](#), a new [Web Service](#) is created here that listens on port 5003 (see line 15) and uses [HTTPS](#). Line 14 specifies that the `Startup` class should be used, which is shown in listing [C.2](#) on page [51](#). This is then created by [Constructor Injection](#) and gets an object of type `IConfiguration`. The real important method for the authorization server is `ConfigureServices`. This gets a collection of type `IServiceCollection` and adds functions and properties to it via some method calls so that [Authorisation](#) can take place. In lines 12 and 13, the predefined scopes and the resulting clients are added via the methods `Config.GetApiResources` and `Config.GetClients`. In listing [C.3](#) on page [52](#), these same methods are present. Lines 5 and 6 show that two “scopes” are created, one with read-only privileges, the other with full privileges. These are then in turn used in the `GetClients` method, in which two clients are created. The first client (line 12–20) can again only read and therefore gets exactly this “scope” assigned, while the other (121–291–29) is again allowed to do everything. Furthermore in `Startup` still the method `Configure` exists, which is responsible above all for representing this program as identity server and to use [HTTPS](#). Additionally it is decided whether developer specific error

codes should be displayed or not, which depends on whether the call of the program takes place in “Debug” or “Release” mode.

5.1.2 Web service

The actual program uses almost the same code as the [Authorization server](#) in listing C.4 on page 53, since both programs represent a [Web Service](#). Only the port on which this [Web Service](#) listens differs (see line 15) to be able to use both at the same time. Otherwise, [Constructor Injection](#) is also used here and thus the class `Startup` is created, which differs in the methods `ConfigureServices` und `Configure`. For better clarity, `ConfigureServices` is therefore shown in listing C.5 on page 53 and `Configure` in listing C.6 on page 54. Both methods configure the [Web Service](#) to be created and are called internally when the program is started.

5.1.2.1 ConfigureServices

In line 7, a [Singleton](#) of an `AutoMapper` class is created and added to the services, which will later ensure that the objects from the database are assigned to so-called “shallow objects”. This makes sense, because not all attributes from the database have to be presented in these objects. Line 13 also adds [Singletons](#) that take care of [Authorisation](#) and [API](#) management, i.e. where which methods are located. Lines 15–34 further process the [Authorisation](#) and link, among other things, the access authorizations and the [Authorization server](#) with this [Web Service](#). For this, attributes are defined, which can be assigned as options to the [HTTP](#) methods, in order to restrict the rights. For easier development and graphical view of the usable [HTTP](#) methods, Swagger¹ is used and set up in the following lines. Another advantage of this is that an XML file is automatically created by Swagger, formatted in the [OpenAPI Specification 3.0](#)². Finally, the [SQL](#) database is added and accessed via a `DbContext` by reading the connection string from a configuration file.

5.1.2.2 Configure

Here, among other configurations, a self-written `ExceptionHandler` is added to the application, which is addressed if an exception is thrown during the execution of any [HTTP](#)

¹Cf. SmartBear. *Swagger UI*. 2022. URL: <https://swagger.io/tools/swagger-ui/> (visited on 07/06/2022).

²Cf. Darrel Miller et al. *OpenAPI Specification V3.0.0*. July 2017. URL: <https://spec.openapis.org/oas/v3.0.0> (visited on 07/06/2022).

method. The internal `switch-case` construct distinguishes between an `ArgumentOutOfRangeException`, a `DataNotFoundException`³ and all other exceptions. In each case, the error messages are converted into [JavaScript Object Notation \(JSON\)](#) and returned together with a corresponding `HTTP` statuscode. The last lines then only set some attributes of the program.

5.1.2.3 ApiControllerBase

This is the parent class, which is implemented by all other controller classes and provides some useful attributes. The annotations above the class name from li2–62-6 are also inherited. Via `[ApiController]`, the class is marked as a controller and can now implement `HTTP` methods, all of which return a file formatted in `JSON` via `[Produces("application/json")]`. To be able to call these methods afterwards, `[Route("api/controller")]` specifies that the address starts with `api/<Name des Controllers>/`. [Authorisation](#) is controlled in lines 5–6 by setting schemas and policy, which can still be overridden for individual methods, however. This class also works with [Constructor Injection](#) and therefore gets some objects that are used to set the class attributes to be able to continue working with it. The passed `IActionDescriptorCollectionProvider` contains a previously generated list of all available methods and links. As mentioned before, it makes sense not to reveal all attributes of the database object, for which the object of type `IMapper` is responsible. Furthermore, each controller can query data from the database with the `lfidContext` and the `logger` is of course there for logging.

5.1.2.4 Controller

Listing C.9 on page 56 shows an example of a controller that implements [ApiControllerBase](#). It shows how a `GET` and a `HEAD` route can be created and how the database query works.

Nothing happens in the constructor except that the super constructor is called again by [Constructor Injection](#). The real work happens in the `GetShadowCountries` method, where this represents a `GET` and a `HEAD` method by the annotations in lines 7 and 8. Internally, a set of all countries is queried here via the `LfidContext` object. If the resulting object is empty, a `DataNotFoundException`⁴ is thrown and otherwise we continue with it. Lines 12–15 represent a [LINQ](#) query that sorts the returned set first by “CustareaCode”⁵

³Cf. listing C.7 on page 55

⁴S. listing C.7 on page 55

⁵The `CustareaCode` represents a code specified in [Aeronautical Radio Incorporated \(ARINC\)](#) format for the corresponding region.

and then by “IcaoCode”⁶. The elements are then grouped by “IcaoCode” and finally mapped into a `CountryModel`. By calling `return Ok(...)`, the resulting set is returned as a formatted **JSON** object. The return object is a `Task of ActionResult` of `IEnumerable of CountryModel`, among other things to make the method to be asynchronous.

5.1.3 Tests

The tests listed here exist primarily to show the differences in request times as documented in figure A.2 on page 40. The `AuthorizedClientFixture` class shows how a client can **authorize** and connect to the **Web Service**. Line 7–11 in listing C.10 on page 57 set the necessary information for the **authentication process**, such as the **URL** of the **Authorization Server**, the `clientId`, the `scope` and the `clientSecret`. These parameters are then used for the `RequestTokenToAuthorizationServer` function, which requests the **refresh token** and returns it if successful. **HTTP** header fields are used for the request. Upon return from this method, the **refresh token** is parsed from **JSON** format and passed to the client, which can now set a header field called “Bearer” with the **refresh token** as its value. This client is then used to send **authorized** requests to the **Webservice**. In the case shown in listing C.12 on page 59, all existing countries are requested. The request itself takes place in the `GetCountryModels` method in lines 28–36, using an asynchronous call to the given URI to request the resource. This list of countries is then deserialized and returned as a `List` object. The same method is found in listing C.13 on page 60, with the only distinction that this client is not **authorized**. The last test (see listing C.11 on page 58) shows how a program can connect directly to the database and receive data via an **SQL** query.

5.1.4 Demo Anwendung

This application is used to emulate the simulator component, which will later query the data. The code for the **GUI** is not mentioned here, because it is not really relevant for this work and complicates the explanation unnecessarily. The class `MainWindow`⁷ provides in the core also only an **authorized** client, as already explained in listing C.10 on page 57. How and over which routes the requests are sent is shown in listing C.16 on page 63. However, these are almost the same methods that are used in the **Tests**.

⁶The **International Civil Aviation Organization (ICAO)** code is used to uniquely identify airports. Since the first two letters describe the country, they are also listed here in the table of countries.

⁷S. listing C.14 on page 61

5.2 Safety Aspects

For a better overview of the realized and treated safety aspects, everything covered in the following is briefly summarized again in table B.3 on page 49. To maintain the structure, the risks of the [OWASP Top 10](#) are assigned to some of the requirements of the [IT Grundschutz Kompendium](#) and – if possible – processed chronologically.

Threat [A01:2021](#) is mitigated by the fact that this [Web Service](#) does not have any classic users, but each [authorized](#) program is assigned its own rights and areas which may be released and accessed. This fulfills the basic requirement [APP.3.1.A1](#) in particular, along with others.

The use of [TLS/SSL](#) counteracts [A02:2021](#), as this means that the transmitted data is generally encrypted and is also not sent at all if the communication partner is not [authorized](#). This can meet the [standard requirements APP.3.1.A8](#), [APP.3.1.A9](#) and [APP.3.1.A11](#).

Since [.NET Core 2.1](#) is used with a [Framework](#) to create a [Web Service](#), each route automatically asks if the given parameter matches the passing value. In addition, the implementations of the methods are designed with [LINQ](#), which means that Microsoft's Entity Framework is used internally for the database queries, which means that the passed values are checked even more and virtually no [SQL](#) injection is possible anymore. Hereby [APP.3.1.A14](#), [APP.3.1.A9](#) and [APP.3.1.A11](#) is fulfilled and counteracts [A03:2021](#).

The fourth threat, [A04:2021](#), relates primarily to [APP.3.1.A8](#) und [APP.3.1.A9](#), the procurement of external material, since foreign code in particular must also be checked for security. This program uses therefore also only packages, which are provided either directly by Microsoft or have very many users and positive entries regarding security. In addition, for each package or [Framework](#), care is taken to ensure that the latest patch of the respective version is used, no version conflicts arise and necessary updates are carried out. This also directly counteracts [A06:2021](#) and [A08:2021](#). Unfortunately, however, this cannot be guaranteed for [.NET Core](#) with regard to the [Framework](#) conditions of the [Operating System](#). Since the [Web Service](#) should and has been designed to be as open as necessary, but as restrictive as possible, [A05:2021](#) and [A07:2021](#) will be addressed. Specifically, [APP.3.1.A1](#), [APP.3.1.A7](#), [APP.3.1.A14](#), [APP.3.1.A9](#) and [APP.3.1.A12](#) can be assigned to this threat. From the beginning, only [authorized](#) client methods were released and no default passwords or user IDs were used, with an additional layer of security provided by the implemented [OAuth 2.0](#) standard, as well as the expiration of the [refresh token](#).

Logging files should be created during each session to record actions on the [Web Service](#),

which is also required by [APP.3.1.A1](#). For example, in order to be able to detect failed login attempts and classify them according to how dangerous they are, this information must be stored securely and, in particular, it must not be passed on to the user. This thwarts [A09:2021](#) if, among other things, when an [authentication attempt](#) fails, it is stored and only the message that the attempt was unsuccessful is returned to the user. Additionally, logging will satisfy [APP.3.1.A7](#). Here, all actions of the [Web Service](#) are stored in a file that cannot be viewed from the outside, so that everything can be traced. The last threat to be addressed, [A10:2021](#), is aimed more at a threat to the user and not to the service providing it. Nevertheless, requirement [APP.3.1.A12](#) can once again be addressed and implemented, since a great deal can be achieved with a restrictive release of methods, which is also reflected in the implementation.

5.3 Usage

If a client is registered with the [Authorization server](#) and has been assigned rights, areas and an ID, the operation always works as shown in figure [A.3](#) on page [41](#). First the client must request a [JWT](#) from the [Authorization server](#), which sends it back as a response after successful validation. After that the actual request of the data follows. The figures [HTTP methods p.1](#), [HTTP methods p.2](#) and [HTTP methods p.3](#) show the currently available [URLs](#) of the [Web service](#), all of which - except for `/api/Countries/fullAnonymous`⁸ - are only accessible via prior [Authorisation](#). The client has to send a request to one of these methods together with the [JWT](#) in the header, which can then be processed by the [Web service](#). The [Web service](#) first has the token validated by the [Authorization server](#), in order to be able to query the data in the database of the [LFID](#) system via an [SQL](#) query if the request is successful. If the data was successfully read from the database, it is sent back to the client as a [JSON](#) object.

⁸The route `/api/Countries/fullAnonymous` was introduced for testing purposes, mainly to be able to create figure [A.2](#) on page [40](#).

6 Results and Conclusions

The design of the [Authentication](#) was – as seen in the [Authorization server](#) – successful and feasible, as well as the provision of some methods for querying the data. Basically, the [Mandatory Requirements](#) could be met. The demo program queries in the background exactly the data in a structured way, which is required by a real [Flight Management System \(FMS\)](#). The user interface remained rather secondary, since different [FMSs](#) with different displays exist. The only important thing here is that the requested data is presented in cascade.

According to the [Comparison of request times](#), the web-based request with an [Authorisation](#) takes about $5ms \pm 2ms$. Thus, the duration hardly differs from the [unauthorized](#) request, from which it can be concluded that an [Authorisation](#) does not have a negative effect on the response times. Obviously, the direct [SQL](#) query is much faster, especially after the first query, but this is exactly what is not desired. Except for the [Web Service](#), no other program should be allowed to send direct [SQL](#) statements to the database. If only the [Web Service](#) has direct access to the database, the entire system will be more stable and more efficient due to the uniformity and the elimination of the error source of multiple simultaneous queries.

7 Summary and Outlook

Especially for the provision of new data and formats of the [LFID](#) system, a [Web Service](#) had to be designed and conceived in such a way that it could communicate securely with the clients and run both reliably and quickly. For this purpose, the foundation of web-based communication was created with [HTTP](#), which can send data encrypted with [TLS/SSL](#). However, this is not yet sufficient for secure communication, for which [OAuth 2.0](#) was introduced afterwards. This protocol is therefore used for the [Authorisation](#) of the clients at the [Web Service](#) by outsourcing the [Authentication](#) to another [Web Service](#). In order to be able to implement all this, the target system was then analyzed, in particular which languages would be useful and feasible for the implementation. The decision was made to use [C#](#) and [.NET Core 2.1](#) for the [Web Service](#) and [.NET Framework](#) for the demo program. The [IT Grundschrift Kompendium](#) was then referred to for the correct design of this application, although due to the sheer size of this document, only the [APP.3.1](#) module was addressed. This was then split and analyzed for each sub-item. As also described in [Further information](#), this was followed by [OWASP Top 10 Threats](#), which is a slightly more general definition of security risks on the Internet, but still works well in conjunction with the [IT Grundschrift Kompendium](#). This can also be seen in the [IT security requirements table](#), where one or more requirements have been added to each threat. The description of the source code can be found in the [Realization](#), where, among other things, much has been said about the attributes of the [.NET Core 2.1 Framework](#) that are used.

To follow up on this thesis and make the [Web Service](#) more widely usable, one could – as already described in the [Optional Requirements](#) – in any case create routes to all database entities. These should reflect all [CRUD](#) operations and also be secured in such a way that there is a difference between read-only and processing clients. Furthermore, a user interface for the [Authorization server](#) should be implemented, which enables the administrator of the applications to create roles and groups and thus also distribute rights to clients. A web page for the [Web Service](#) should also exist so that the administrator can make certain settings with the program. This includes, for example, saving and loading a database backup, importing new data into the database and reading out logging files. Most importantly, the administrator of the system should be able to set the database password and thus connect to the database. In addition, one of the next steps must be to upgrade the programs to the latest version of [.NET](#) (currently [.NET 6.0](#)) to be able to fix security gaps and known errors.

Bibliography

- Allen, Christopher and 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, and Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 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, and David Orchard. *Web Services Architecture*. Feb. 2004. URL: <https://www.w3.org/TR/ws-arch/> (visited on 07/06/2022).
- Fielding, Roy T. and Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. June 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. Oct. 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, and Uri Sarid. *OpenAPI Specification V3.0.0*. July 2017. URL: <https://spec.openapis.org/oas/v3.0.0> (visited on 07/06/2022).
- Nielsen, Henrik, Roy T. Fielding, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. May 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/> (visited on 07/06/2022).
- 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 and 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/> (visited on 07/06/2022).

Srivastava, Nimit. *Differences between .NET Core and .NET framework*. June 2022.
URL: <https://www.geeksforgeeks.org/differences-between-net-core-and-net-framework/#:~:text=Application%20Models-,.,windows%20forms%20and%20WPF%20applications.> (visited on 07/06/2022).

A Figures

Figure A.1: *Layer model*

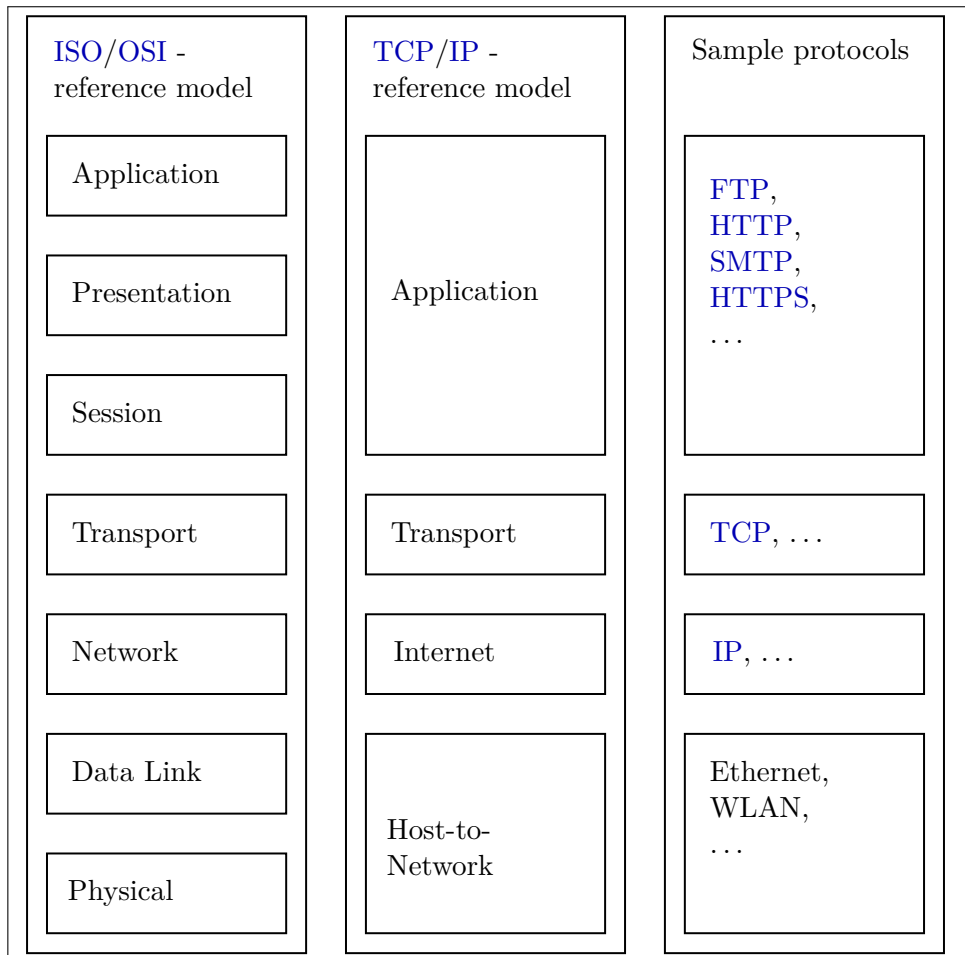


Figure A.2: *Comparison of request times*

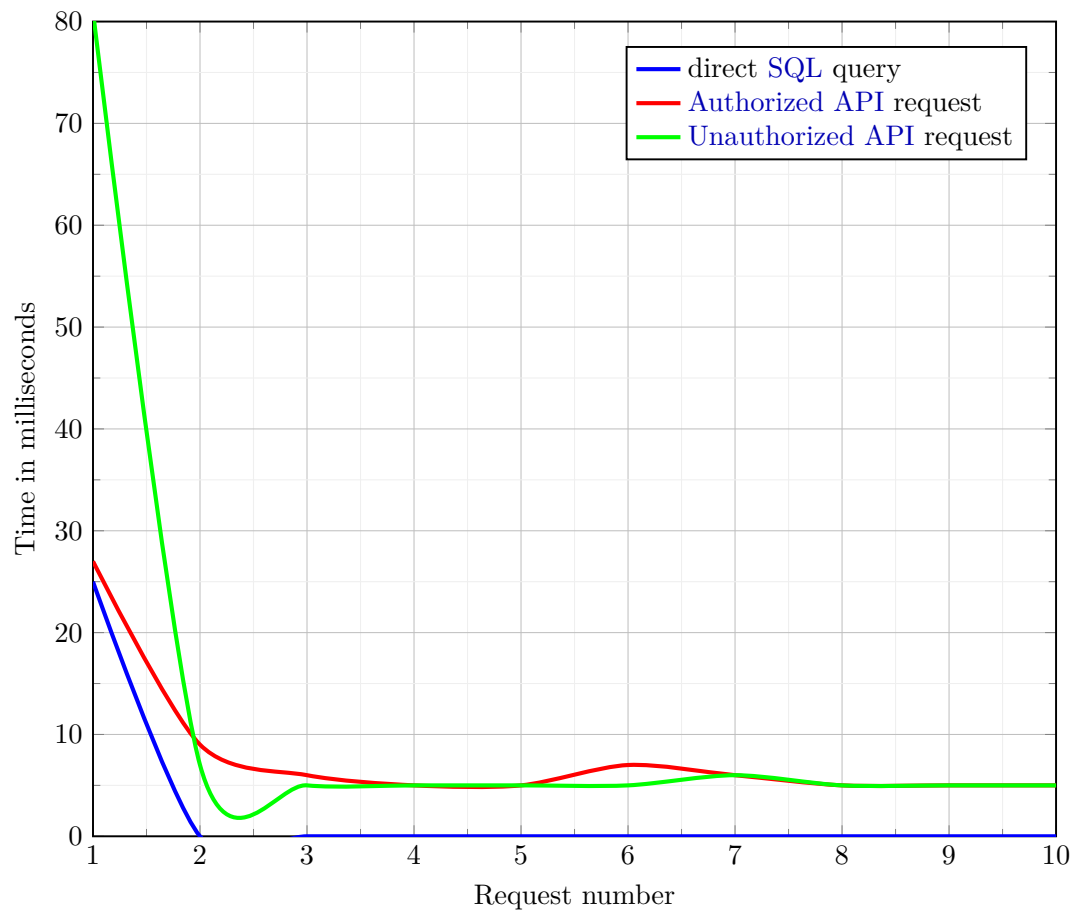


Figure A.3: Sequence diagram of a request

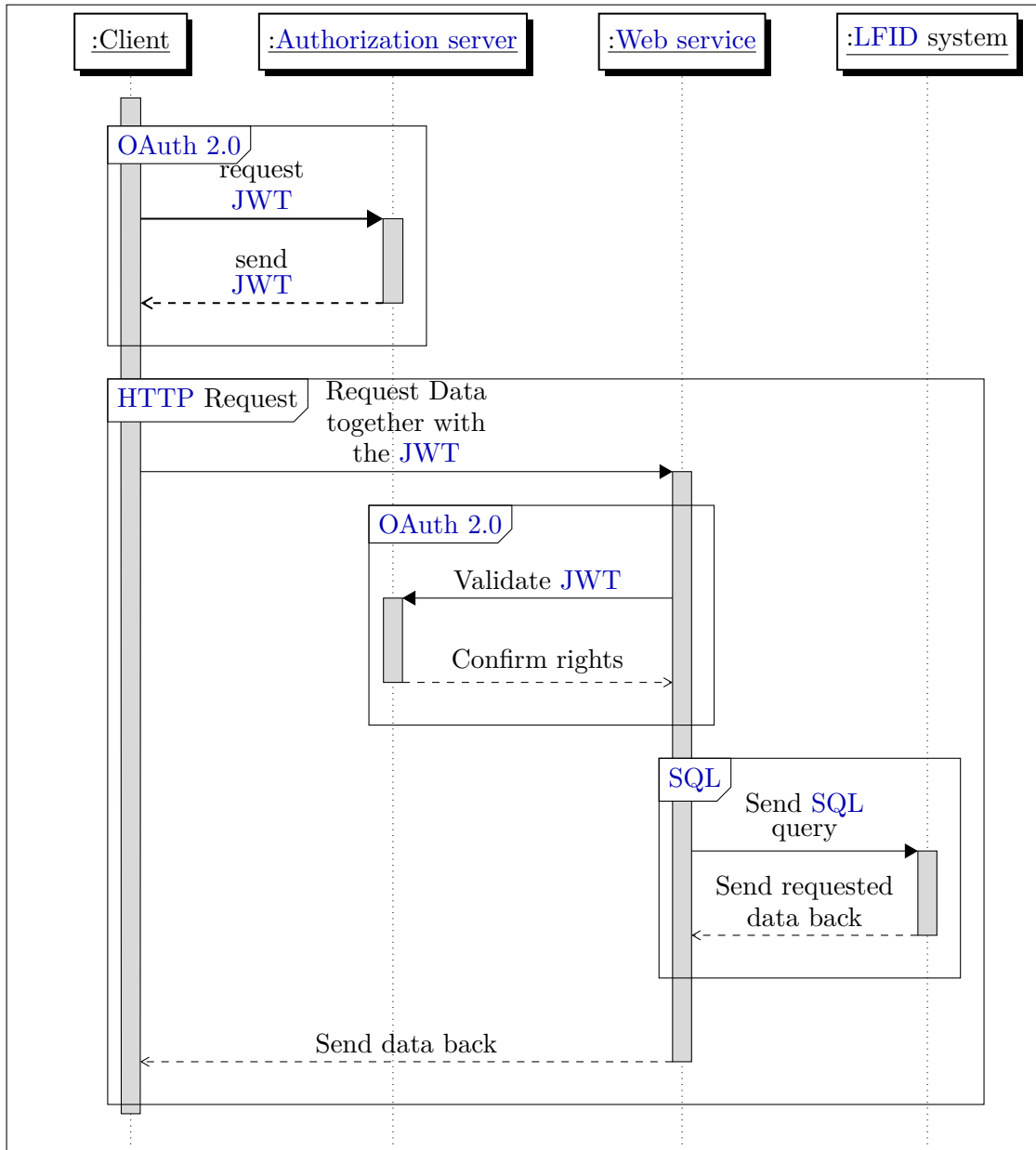


Figure A.4: Pseudo diagram of the relevant database tables

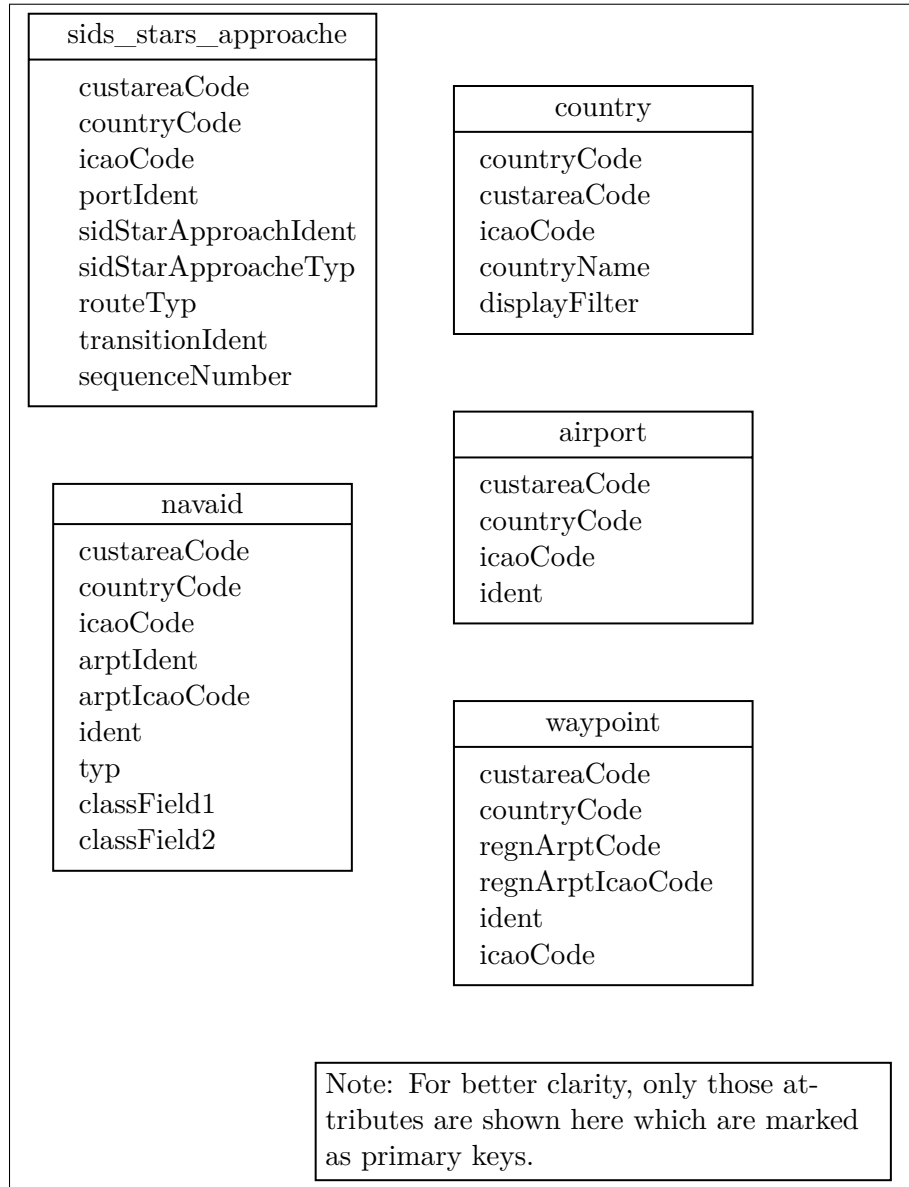



Figure A.5: HTTP methods p.1



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"	

Figure A.6: *HTTP methods p.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"	🔒
Countries ▼			
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		🔒
HateoasHome ▼			
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"	🔒
Navaid s ▼			
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		🔒

Figure A.7: *HTTP methods p.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 Tables

Table B.1: Cross reference table for elementary hazards 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 0.14	G 0.15	G 0.18	G 0.19	G 0.23	G 0.28	G 0.30	G 0.31	G 0.43	G 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				

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

Quelle: Cf. Nimit Srivastava. Differences between .NET Core and .NET framework. June 2022. URL: <https://www.geeksforgeeks.org/differences-between-net-core-and-net-framework/#:~:text=Application%20Models-,,windows%20forms%20and%20WPF%20applications.> (visited on 07/06/2022)

	.NET Core 2.1	.NET Framework
Open Source	Yes	Some components only
Cross-Platform	Yes, runs on Windows, Linux and macOS	No, runs only on Windows
Application Models	Does not support desktop application development, focuses more on web application development	Used for desktop application development, but can also be used for web applications
Installation	All dependencies are packed into the package to be delivered so that it can be used across platforms	Here only the project specific package is packed, all others will be delivered by the installation in the operating system
Support for micro-services and REST services	Yes, supports both	Supports only the implementation of REST services
Performance and scalability	More efficient and scalable	Not so effective compared to .NET Core 2.1

Table B.3: *IT security requirements table*

OWASP Top 10	IT Grundschutz Kompendium	Realisierung
A01:2021	APP.3.1.A1, APP.3.1.A14, APP.3.1.A9	APP.3.1.A7, APP.3.1.A8,
A02:2021	APP.3.1.A8, APP.3.1.A11	APP.3.1.A9,
A03:2021	APP.3.1.A14, APP.3.1.A11	APP.3.1.A9,
A04:2021, A06:2021, A08:2021	APP.3.1.A8, APP.3.1.A9	
A05:2021, A07:2021	APP.3.1.A1, APP.3.1.A14, APP.3.1.A12	APP.3.1.A7, APP.3.1.A9,
A09:2021	APP.3.1.A1, APP.3.1.A7	
A10:2021	APP.3.1.A12	

C Listings

C.1 Authorization Server

Listing C.1: *Program.cs in Authorization Server*

```
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 Authorization Server*

```
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 Authorization Server*

```

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 Application

Listing C.14: *Application.cs* in demo application

```
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: *Request token in demo application*

```
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: Requesting data in demo application

```

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 }

```

