

# **Dokumentation**

## zum Programmentwurf „Anwesenheitsliste“

im Rahmen der Vorlesung Programmieren II

bei Prof. Dr. Helmut Neemann

an der Dualen Hochschule Baden-Württemberg Mosbach

Autoren

5736465, 5703004

22.12.2021

# Inhaltsverzeichnis

1 Architekturdokumentation.....	3
1.1 Grundlegender Aufbau.....	3
1.1.1 timeutil.....	3
1.1.2 journal.....	4
1.1.3 convert.....	4
1.1.4 web.....	5
1.2 Token- und Session-Management.....	6
1.3 Login- und Logout-Prozess.....	7
1.4 Formular abschicken.....	7
1.5 Strukturierung der Anwendungen.....	8
2 Anwenderdokumentation.....	9
2.1 Webservice.....	9
2.1.1 QR-Code-Service.....	9
2.1.2 Login-/Logout-Service.....	10
2.2 Analyzer-Tool.....	12
2.2.1 locations.....	13
2.2.2 contacts.....	14
2.2.3 attendances.....	14
3 Dokumentation des Betriebs.....	14
3.1 Kompilieren der Anwendungen.....	15
3.2 Definition verfügbarer Orte.....	15
3.3 Betrieb der Webanwendung.....	15
3.4 Betrieb des Kommandozeilentools.....	17
4 Aufgabenverteilung.....	17

# 1 Architekturdokumentation

Das Ziel des Projekts ist es, eine Webanwendung umzusetzen, die mittels QR-Codes Anwesenheitslisten für bestimmte Orte und Personen erstellt. Diese Listen können wiederum mit einem Kommandozeilentools ausgewertet werden. Die genauen Anforderungen können in einer separaten Datei eingesehen werden.

Im Folgenden wird die Architektur des Programmentwurf genauer beleuchtet.

## 1.1 Grundlegender Aufbau

Die grundlegende Ordnerstruktur enthält die beiden Hauptprogramme *analyzer* und *service* im *cmd*-Ordner, die separat voneinander kompiliert und ausgeführt werden können.

Die Programme greifen hauptsächlich auf eine gemeinsame Codebasis, die im Ordner *internal* zu finden ist, zurück. Hier werden einzelne Packages themenbasiert definiert, die im Folgenden näher erläutert werden.

### 1.1.1 timeutil

Das Package *timeutil* abstrahiert das *time*-Package der Go-Standardbibliothek, um eigene Formate für Zeitstempel (*Timestamp*) und Daten (*Date*) bereitzustellen. Abbildung 1 zeigt ein Klassendiagramm für die Typen, die in diesem Package bereitgestellt werden.

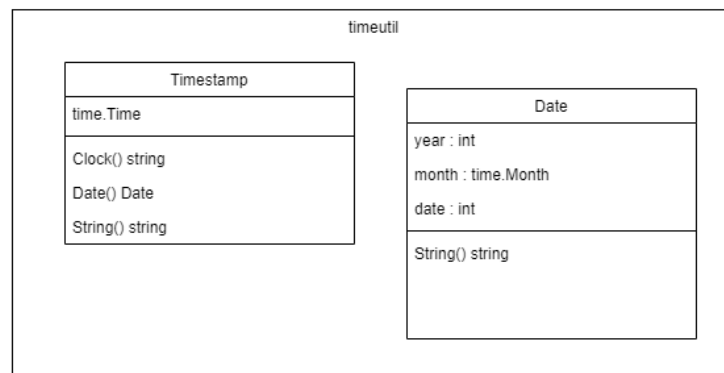


Abbildung 1: Klassendiagramm für das Package *timeutil*

Die Funktionen innerhalb dieses Packages können somit beliebig erweitert werden, um ggf. weitere Anforderungen mit niedrigem Aufwand implementieren zu können. So bietet das Package z.B. die Möglichkeit, ein Datum aus einem Zeitstempel zu extrahieren, Daten aus Zeichenketten zu lesen oder in Zeichenketten zu schreiben. Dabei wird stets die Kompatibilität mit der Go-Standardbibliothek eingehalten.

### 1.1.2 journal

Im *journal*-Package werden alle Strukturen und Funktionalitäten definiert, um Journaldateien zu pflegen. Die Informationen und Daten eines Besuchers werden in der Struktur *Person* gespeichert. *Address* repräsentiert die Wohnadresse eines Besuchers. Die Struktur *Contact* repräsentiert die Zusammenkunft zweier Person mit Angabe der Person (*Person*), dem Ort (*Location*), der Start- (*Start*) und der Endzeit (*End*) des Kontakts, sowie der Dauer (*Duration*) des Kontakts. Mit der Struktur *ContactList* wird eine Liste von beliebigen Kontakten erstellt. Die Struktur *JournalEntry* repräsentiert einen Eintrag einer Journaldatei. Diese werden dann in dem Attribut *Entries* der *Journal*-Struktur abgelegt. Ein *Event* ist eine Liste von Aktionen, die eintreten können. Diese beschränken sich momentan auf den *Login* und *Logout* einer Person von einem Ort (*Location*). *AttendanceEntry* ist dabei eine Struktur, in der alle Kontakte zu einer Person zusammen mit der Login-, wie auch Logoutzeit, gepflegt werden können. Die *AttendanceList* ist analog zur *ContactList* eine Liste für die *AttendanceEntries*. Eine Übersicht über alle Datenstrukturen in diesem Package mitsamt ihren Attributen findet sich in Abbildung 2.

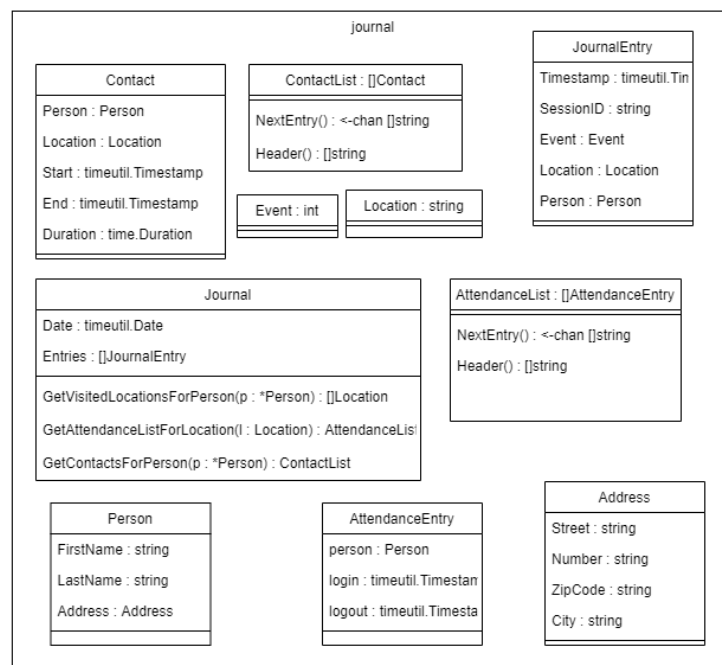


Abbildung 2: Klassendiagramm für das Package *journal*

### 1.1.3 convert

Das Package *convert* stellt ein Interface *Converter* bereit. Alle Strukturen, die dieses Interface implementieren, stellen somit die Funktionen *Header* und *NextEntry* bereit, mit denen eine Listenstruktur in beliebige Formate konvertiert werden können. Die Funktion *ToCSV* kann eine Datenstruktur schließlich in ein CSV-Format bringen, das anschließend weiterverarbeitet oder ausgegeben werden

kann. Mit diesem Interface besteht allerdings auch die Möglichkeit, flexible Ausgaben zu gestalten. So könnte beispielsweise eine Ausgabe in der Kommandozeile eine Struktur in ein anderes Format konvertieren, als wenn sie in eine Datei ausgegeben wird. Da diese Anforderung nicht explizit gefordert ist, wurde auf eine Implementierung verzichtet. Die Architektur der Bibliothek bietet aber mit dem *Converter*-Interface eine einfache und schnelle Möglichkeit, eine Struktur auch in andere Formate zu überführen. Abbildung 3 zeigt die schematische Darstellung des Interfaces.

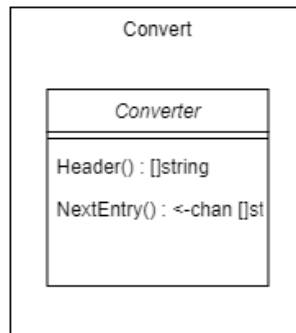


Abbildung 3: Klassendiagramm für das Package convert

#### 1.1.4 web

Das *web*-Package stellt schließlich die Funktionalitäten und Typen für Zugangstoken, die Sessionverwaltung, Usercookies und die Verwaltung der Tokens für verschiedene Orte bereit.

Ein *AccessToken* stellt ein Zugangstoken für einen Ort dar. Dieser wird mittels der privaten Struktur *jsonAccessToken* mit JSON-konformen Datentypen repräsentiert, sodass er von einer Web-API abgerufen werden kann.

Da mehrere *AccessTokens* gleichzeitig für einen Ort existieren können, werden alle Tokens mit einem *Valid*-Attribut versehen. Dieses Attribut wird mit jedem Ablauf der *Expire-Time* eines Tokens verringert. Ist der Wert kleiner als 0, so ist das Token nicht mehr valide. Alle validen Tokens werden in der Struktur *ValidTokens* gespeichert, die auf die Map-Struktur des *sync*-Packages der Go-Standardbibliothek zurückgreift. Damit wird verhindert, dass *Dataraces* aufgrund von gleichzeitigem Schreib- und Lesezugriff auftreten. Zusätzlich bietet die Struktur einige Funktionalitäten, um die Arbeit mit den Tokens zu erleichtern.

Die Struktur *UserCookie* enthält den Cookie, der im Browser eines Benutzers gespeichert wird, um später auf dessen Daten zugreifen zu können. Der Cookie ist dabei an die Struktur eines JSON-Web-Tokens angelehnt. Er speichert alle User-Daten im JSON-Format. Die Daten werden mit Base64 kodiert. Um die Integrität der Daten zu gewährleisten, wird in dem Cookie ein Hash-Wert abgespeichert, der die JSON-Daten des Benutzers, zusammen mit einem privaten Schlüssel des Servers hashet. So ist sichergestellt, dass die Daten nicht manuell verändert werden können.

Die Datenstruktur *Locations* hält schließlich einen Slice von Orten, die in der XML-Datei definiert wurden und verwaltet die entsprechenden *AccessTokens* dafür.

Um die mehrfache An- und Abmeldung eines Benutzers eindeutig identifizieren zu können, werden auf dem Server *Sessions* gespeichert. Diese Datenstruktur hält sowohl die *Person*, die sich angemeldet hat, als auch die *Location*, an der die Person sich angemeldet hat, zusammen mit einer eindeutigen *SessionID*. Für die Generierung von Ids steht ein eigener Generator bereit, der zufällige, eindeutige (hängt von der Wahl der Länge der Ids ab) Ids definieren kann.

Eine Übersicht über alle Strukturen und deren Attribute im *web*-Package bietet Abbildung 4.

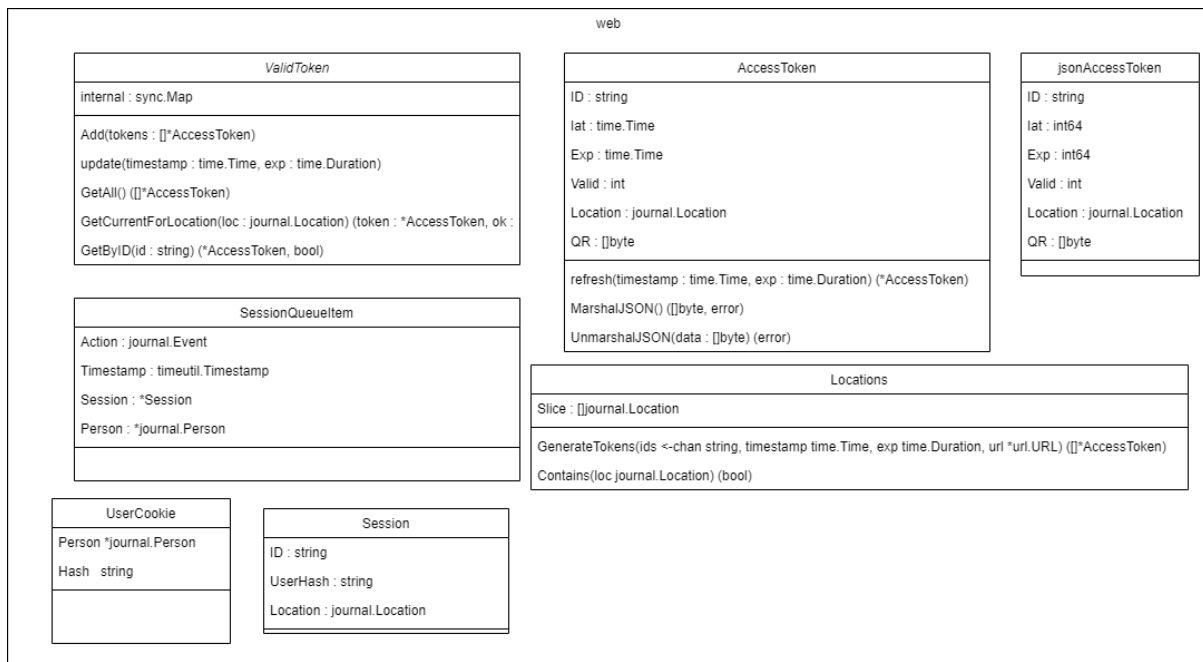


Abbildung 4: Klassendiagramm für das Package *web*

## 1.2 Token- und Session-Management

Das Verwalten von *AccessTokens* und *Sessions* funktioniert analog zueinander. Die Verwaltung wird jeweils in eine separate Goroutine ausgelagert. Innerhalb einer Goroutine wird schließlich ein Event-Loop gestartet, der auf verschiedene Ereignisse hört, die über Channels gesendet werden. So läuft im Token-Manager beispielsweise ein Timer ab, der in regelmäßigen Abständen eine Liste an validen *AccessTokens* verwaltet: Er fügt neue hinzu, aktualisiert bestehende und löscht selbstständig Tokens, die nach Ablauf der Zeit ungültig geworden sind.

Der Session-Manager hört auf sogenannte *SessionQueueItems*. Diese Items werden mit jedem Login oder Logout in eine Queue geleitet (Channel mit Buffer). Die Goroutine des Session-Managers arbeitet diese Queue nacheinander ab und schreibt entsprechende Aktionen in eine Journal-Datei. Durch diese Architektur wird zum einen sichergestellt, dass mehrere Anfragen gleichzeitig gestellt werden

können, ohne dass die Webanwendung plötzlich nicht mehr reagiert, wenn das Schreiben einer Aktion in die entsprechende Journaldatei länger dauert, zum anderen werden durch die sequenzielle Abarbeitung der Ereignisse Aktionen in der richtigen Reihenfolge – nämlich in der des Eintretens – in die Journaldatei geschrieben. Kurze Wartezeiten seitens eines Benutzers sind dann zu erwarten, wenn die Queue vollgelaufen ist. Diese Parameter lässt sich aber je nach Auslastung des Servers und geplantem Einsatz konfigurieren und damit sehr gut skalieren.

### 1.3 Login- und Logout-Prozess

Im Folgenden soll kurz der Prozess des Logins bzw. Logouts schematisch dargestellt werden, auf dessen Basis die Anwendungslogik implementiert wurde.

Wird der Link eines QR-Codes aufgerufen, wird zunächst geprüft, ob der eingegebene *AccessToken* noch gültig ist. Bei Ungültigkeit wird ein Fehler angezeigt, dass der Benutzer zu der Aktion der An- bzw. Abmeldung nicht berechtigt ist. Ist der Token gültig, so wird der *user-Cookie* vom Client abgefragt. Wenn dieser nicht vorhanden oder ungültig ist, so muss davon ausgegangen werden, dass der Benutzer sich zum ersten Mal anmeldet. Es wird ein Anmeldeformular mit leeren Feldern angezeigt. Ist der Cookie jedoch vorhanden und auch valide, so kann zunächst nach einer Session dieses Benutzers gesucht werden. Ist bereits eine Session vorhanden, so hat der User sich zuvor angemeldet und kann nun abgemeldet werden. Es wird eine Logout-Seite ausgegeben. Ist noch keine Session vorhanden, so will der Benutzer sich anmelden. Es wird das entsprechende Anmeldeformular angezeigt, vorausgefüllt mit den Daten des Benutzers aus dem Cookie. Eine schematische Darstellung dieses Prozesses zeigt Abbildung 5.

### 1.4 Formular abschicken

Der Login-Prozess ist zweigeteilt. Zunächst wird überprüft, ob das Anmeldeformular angezeigt werden muss und ob bereits Daten vorhanden sind. Der zweite Schritt der Anmeldung ist das tatsächliche Abschicken des Formulars. Dies wird mittels eines *POST*-Requests umgesetzt, bei dem alle eingegebenen Daten mitgesendet werden.

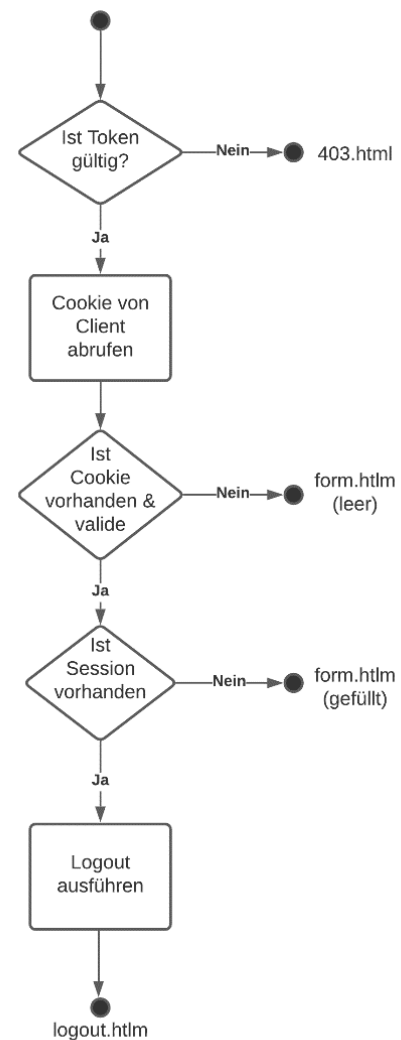


Abbildung 5: An-/Abmeldeprozess

Zunächst wird im Backend das Formular auf Vollständigkeit und Gültigkeit überprüft. Ist beispielsweise der *AccessToken* nicht mehr verfügbar, so werden entsprechende Fehler angezeigt. Sind die Daten alle valide, wird dem Benutzer ein Cookie mit diesen Daten ausgestellt, um ihn später anhand dieses Cookies wieder identifizieren zu können. Anschließend wird ermittelt, ob bereits eine Session für diesen Benutzer existiert. Dieser Fall kann auftreten, wenn sich ein Benutzer angemeldet, aber nicht abgemeldet hat. Ist bereits eine Session vorhanden, so wird der Benutzer zunächst von dieser abgemeldet. Das verhindert, dass ein Benutzer an zwei Sessions (und damit zwei Orten) gleichzeitig sein kann, bzw. ein Benutzer, hat er einmal vergessen, sich an einem Ort abzumelden, für immer an diesem Ort angemeldet bleibt. Zuletzt wird eine neue Session für die aktuelle Anmeldung erzeugt. Mit dem Anlegen- bzw. Schließen einer Session wird automatisch vom Session-Manager ein neuer Journal-Eintrag erstellt. Dieser Prozess wird in Abbildung schematisch dargestellt.

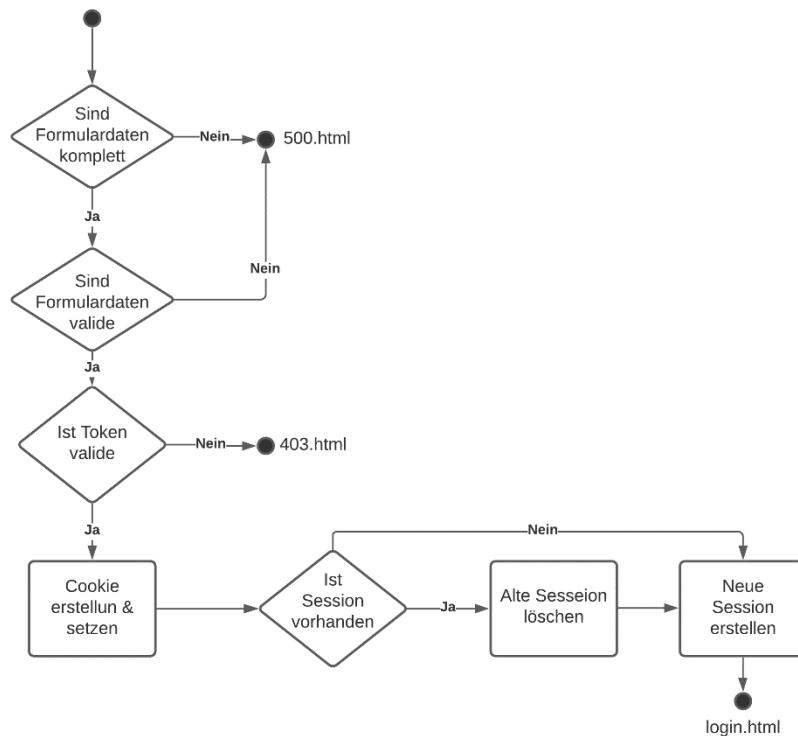


Abbildung 6: Anmeldeprozess nach Abschicken des Formulars

## 1.5 Strukturierung der Anwendungen

Um alle Anforderungen an das Programm umzusetzen, werden die Anwendungen in zwei Hauptprogramme aufgeteilt: einem CLI-Tool (*analyzer*) und einem Webservice (*service*). Die beiden Programme können unabhängig voneinander kompiliert und weiterentwickelt werden.

Der Webservice wiederum bietet zwei separate Webserver, die auf zwei unterschiedlichen Ports laufen. Das ist zum einen der QR-Code-Service, mit dem QR-Codes für verschiedene Locations ange-



zeigt werden können – diese Anwendung kann schließlich auf einem Monitor angezeigt werden, da sich die QR-Codes automatisch aktualisieren –, zum anderen dem Login-/bzw. Logout-Service, der mithilfe des Links eines QR-Codes erreicht werden kann. Hier können sich Benutzer beispielsweise von ihrem Smartphone aus an- bzw. abmelden, indem der QR-Code gescannt wird und die entsprechenden Daten eingegeben werden.

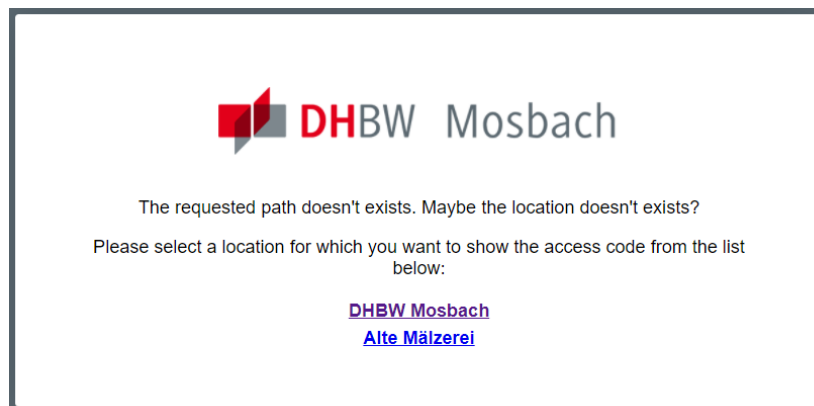
## 2 Anwenderdokumentation

### 2.1 Webservice

Die Webanwendung ist in zwei Teile unterteilt: dem QR-Code-Service und dem Login- bzw. Logout-Service. In den folgenden Beschreibungen werden die Standardkonfigurationen zur besseren Darstellung verwendet. Werden die Konfigurationen beim Starten der Webservices verändert, so müssen auch die hier gezeigten Pfade entsprechend angepasst werden.

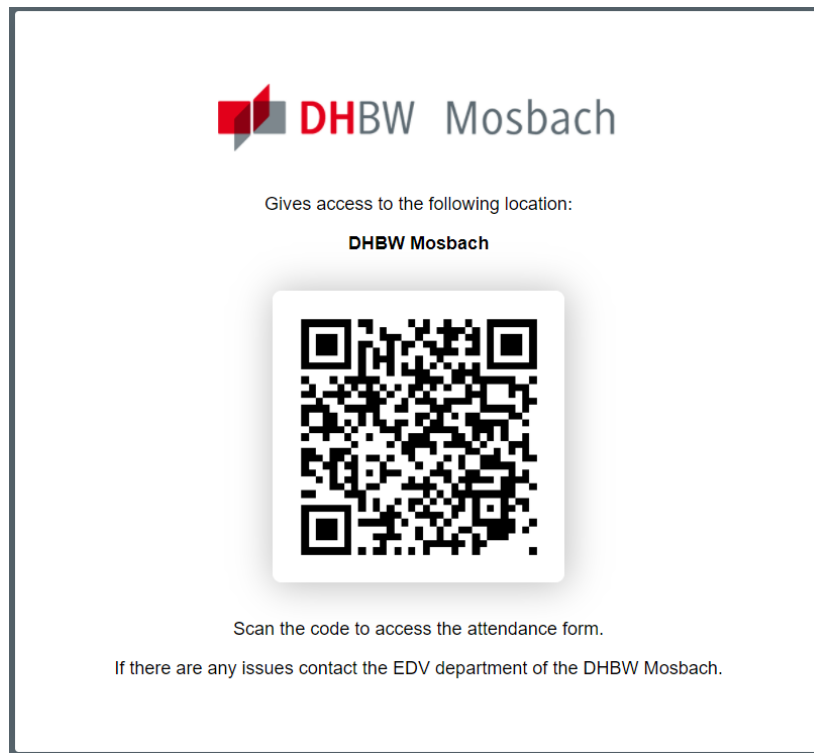
#### 2.1.1 QR-Code-Service

Um einen QR-Code anzuzeigen, rufen Sie die entsprechende URL, hier <https://localhost:4443>, auf.



Wird beim Aufrufen der URL noch keine Location spezifiziert, erhalten Sie eine Übersicht, über alle verfügbaren Locations, für die QR-Codes generiert werden können. Von hier aus kann die entsprechende Location ausgewählt werden. Alternativ kann der Ort auch direkt mit dem URL-Aufruf mitgegeben werden, z.B. <https://localhost:4443/DHBW%20Mosbach>.

In jedem Fall erhalten Sie die entsprechende Seite für den Ort, wie die folgende Abbildung zeigt. Der QR-Code wird nun automatisch in regelmäßigen Zeitabständen aktualisiert.



### **Hinweis:**

Der aktuelle Token für einen Ort kann auch direkt über die API als JSON-String über beispielsweise <https://localhost:4443/api/tokens?location=DHBW%20Mosbach> aufgerufen werden.

Soll eine Liste aller aktuellen Tokens abgerufen werden, kann <https://localhost:4443/api/tokens> aufgerufen werden.

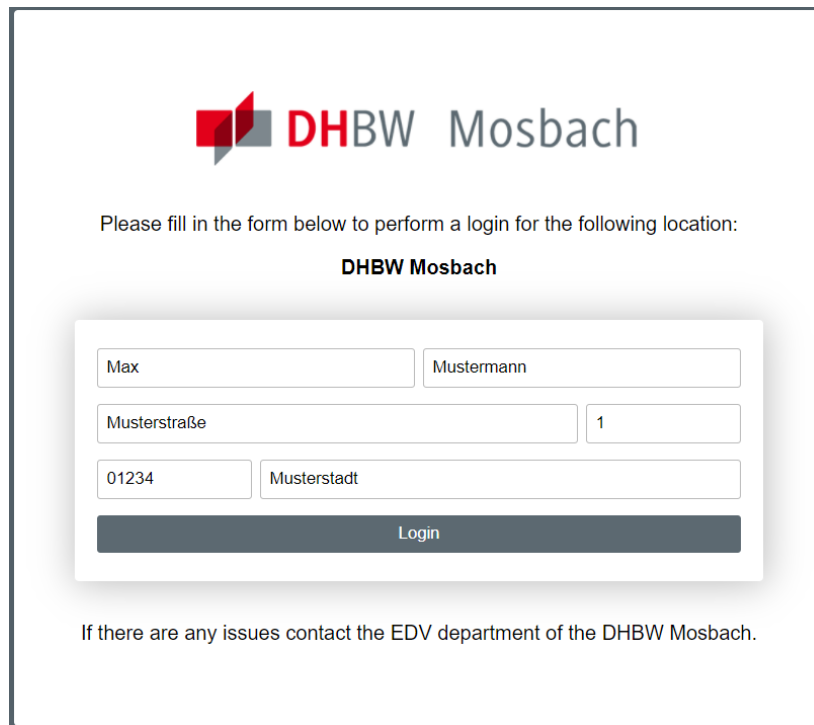
Wird ein Ort aufgerufen, der nicht existiert, so erscheint eine Fehlermeldung (404 Not Found).

Wird der QR-Code eingescannt, so wird die Login-Seite aufgerufen.

### **2.1.2 Login-/Logout-Service**

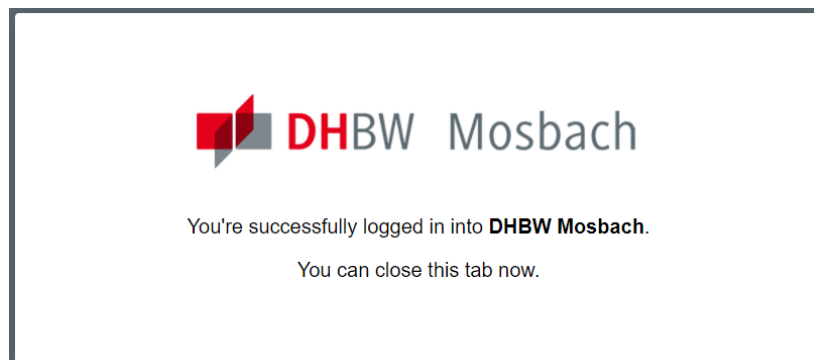
Soll die Login-Seite manuell aufgerufen werden, so kann beispielsweise <https://localhost:4444/access?token=<token>> aufgerufen werden, wobei `<token>` durch einen validen *AccessToken* ersetzt werden muss.

Beim Betreten oder Verlassen eines Ortes wird der QR-Code am Eingang eingescannt. Mit dem Link gelangt man zur entsprechenden Login- bzw. Logout-Seite. Hier müssen die Kontaktdaten eingegeben werden. Wurde zuvor schon einmal ein Formular ausgefüllt, so werden die Felder beim erneuten Scannen eines QR-Codes bereits vorausgefüllt sein, wie die folgende Abbildung zeigt.

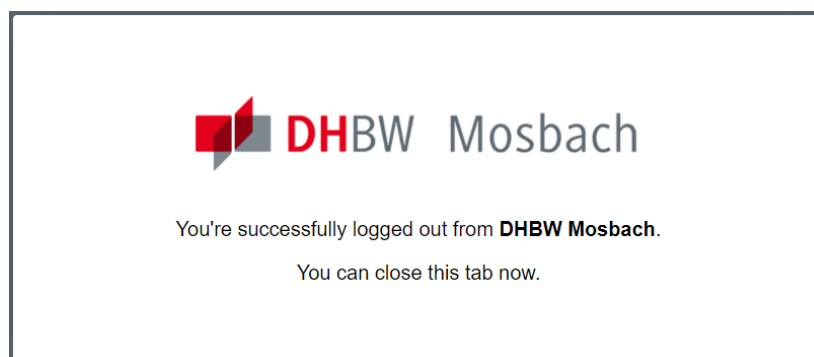


The screenshot shows a login interface for DHBW Mosbach. At the top is the logo, which consists of a red square with a white 'D' and the text 'DHBW Mosbach'. Below the logo, the text 'Please fill in the form below to perform a login for the following location:' is displayed, followed by 'DHBW Mosbach'. The form itself is a white box with a shadow, containing several input fields: 'Max' (first name), 'Mustermann' (last name), 'Musterstraße' (street), '1' (house number), '01234' (postal code), and 'Musterstadt' (city). A dark grey 'Login' button is at the bottom of the form. Below the form, a message states: 'If there are any issues contact the EDV department of the DHBW Mosbach.'

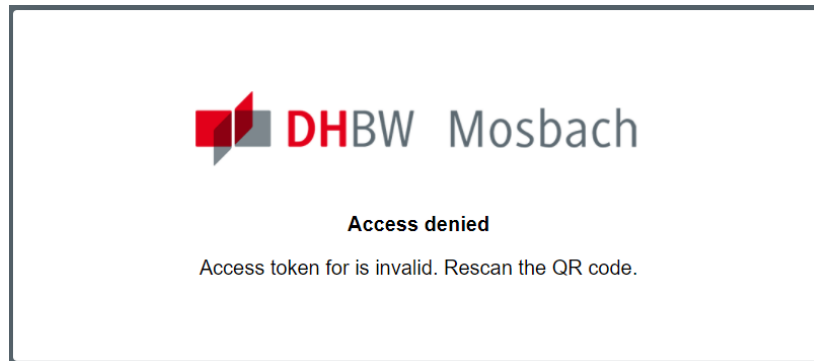
Mit einem Klicken auf den *Login*-Button wird der entsprechende Benutzer eingeloggt und erhält eine Erfolgsmeldung:



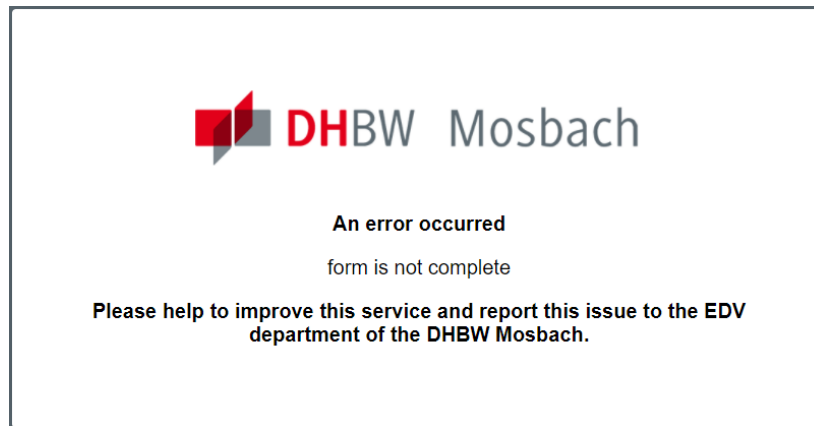
Wird der QR-Code erneut eingescannt, so wird der Benutzer automatisch abgemeldet. Ist der Logout erfolgreich, so wird eine entsprechende Erfolgsmeldung angezeigt:



Falls der *AccessToken* nicht gültig sein sollte oder während des An- bzw. Abmeldeprozesses seine Gültigkeit verliert, so wird eine entsprechende Fehlermeldung (403 Unauthorized) angezeigt. Der Login bzw. Logout war dann nicht erfolgreich und muss ggf. erneut durchgeführt werden. Um einen neuen Access-Token zu erhalten, muss der QR-Code erneut eingescannt werden.



Wurde das Formular zur Anmeldung nicht korrekt ausgefüllt oder bei der Übertragung der Daten werden Fehler festgestellt, so erscheint ebenfalls eine Fehlermeldung (500 Internal Error).



## 2.2 Analyzer-Tool

Mit dem Kommandozeilentool lassen sich die von der Webserver geschriebenen Journaldateien auslesen und auswerten. Dass das Tool entsprechende Journaldateien finden kann, muss sich auf derselben Ordner Ebene eine *data*-Ordner befinden. In diesem Ordner wird nach Journaldateien gesucht.

Ein Aufruf des Analyzer-Tools folgt folgendem Schema:

```
analyzer [command] <date>
```

Dabei ist wird mit *<date>* das Datum der zu analysierenden Journaldatei in der Form *Jahr/Monat/Tag* angegeben. Mit dem *-h* Flag kann auch eine Übersicht über verfügbare Commands gegeben werden.

```
Usage:
  analyzer [command] <date>

  <date> is of form YYYY/mm/dd and specifies for which date a
  journal file should be load.

Commands:
  locations    Print locations for a specific person.
  contacts     Print all contacts for a specific person.
  attendances  Create an attendance list for a specific location.
```

Es stehen momentan drei verschiedene Commands zur Verfügung, die wiederum weitere Parameter benötigen. Im Folgenden sollen diese Commands näher erläutert werden.

### 2.2.1 locations

Mit dem *locations*-Command kann eine Liste von Orten angegeben werden, die eine Person besucht hat. Es wird mit dem Parameter *-person* aufgerufen. Ein Aufruf könnte demnach wie folgt aussehen:

```
analyzer locations -person=Max 2021/11/13
```

Mit diesem Aufruf würde eine Liste von Orten, die Max am 13. November 2021 besucht hat, erstellt werden. Hier gibt es noch einige Dinge zu beachten:

- Wenn die Journaldatei nicht existiert, wird ein Fehler ausgegeben
- Eine Person kann mit allen Attributen, die sie beschreibt, also *Name*, *Vorname* oder sogar Adressdaten wie *Straße*, *Hausnummer*, *Postleitzahl* oder *Stadt* gesucht werden. Werden mehrere Personen mit denselben Attributen gefunden, so wird eine Liste dieser Personen ausgegeben und es müssen mehr Attribute, durch Kommata getrennt angegeben werden, um die Person eindeutig zu identifizieren.

#### Beispiel:

```
analyzer locations -person=Musterstadt 2021/11/13
```

Dieser Aufruf würde alle Personen, die in Musterstadt wohnen filtern (oder die ein beliebiges Attribut haben, das Musterstadt heißt). Gibt es mehrere Personen in dieser Journaldatei, die in Musterstadt wohnen und gefunden werden, so muss die Suche spezifiziert werden, beispielsweise mit:

```
analyzer locations -person=Musterstadt,Max 2021/11/13
```

Nun werden nur noch alle Personen gefiltert, die ein Attribut *Musterstadt* und *Max* haben. Sobald die Suche eindeutig ist, wird die gewünschte Liste ausgegeben.

### 2.2.2 contacts

Das *contacts*-Command wird ebenfalls mit dem Parameter *-person* aufgerufen. Es gelten dieselben Bedingungen wie zuvor. Mit diesem Command kann eine Kontaktliste aller Personen an diesem Tag erstellt werden, die sich am selben Ort zur selben Zeit wie die Person aufgehalten haben.

Die Ausgabe ist im CSV-Format und wird standardmäßig zunächst nur auf der Kommandozeile ausgegeben. Mit dem optionalen Parameter *-w* kann eine Datei definiert werden, in die stattdessen die Ausgabe geschrieben werden soll. Diese Datei kann später in anderen Programmen verarbeitet werden. Ein Aufruf könnte beispielsweise folgendermaßen aussehen:

```
analyzer contacts -person=Max -w=contacts.csv 2021/11/13
```

### 2.2.3 attendances

Mit dem *attendances*-Command kann eine Anwesenheitsliste aller Personen an diesem Tag erstellt werden. Sie wird mit dem Parameter *-location* aufgerufen und definiert, für welchen Ort eine Kontaktliste erstellt werden soll. Die Ausgabe erfolgt ebenfalls im CSV-Format und kann optional mit dem *-w* Parameter in eine Datei umgelenkt werden. Ein Aufruf könnte beispielsweise folgendermaßen aussehen:

```
analyzer attendances -location="DHBW Mosbach" -w=attendacnes.csv \
2021/11/13
```

Bei der Ausgabe könnte es vorkommen, dass ein Login- oder Logoutzeitpunkt fehlt, dann hat sich ein Benutzer zwar ein- oder ausgeloggt aber die entsprechende andere Aktion wurde an diesem Tag nicht ausgeführt. Eventuell findet sich ein Logout- oder Logineintrag am nachfolgenden bzw. vorherigen Tag in einer Journaldatei.

## 3 Dokumentation des Betriebs

Der folgende Abschnitt beschreibt, welche Schritte vollzogen werden müssen, um die Webanwendung erfolgreich auf einem eigenen System zu betreiben. Es wird dabei vorausgesetzt, dass eine entsprechende Go-Installation (Version 1.12 oder höher) bereits erfolgt ist. Eine detaillierte Anleitung zur Installation von Go auf verschiedenen Betriebssystemen findet sich unter <https://go.dev/doc/install>.

### 3.1 Kompilieren der Anwendungen

Die beiden Anwendungen – sowohl der Webservice, als auch das Analyzer-Tool – liegen als Quelldateien vor und können für jedes gängige Betriebssystem kompiliert werden. Dazu kann aus dem Root-Verzeichnis heraus folgender Befehl ausgeführt werden: `go build ./cmd/service` bzw. `go build ./cmd/analyzer`. Damit werden die entsprechenden ausführbaren Dateien erzeugt.

### 3.2 Definition verfügbarer Orte

Der Webservice stellt QR-Codes für verschiedene Orte (*Locations*) bereit, die beim Starten des Services aus einer XML-Datei gelesen werden. Eine Beispiel-Datei, die bereits mit der Anwendung im *assets*-Ordner ausgeliefert wird, zeigt, wie die Struktur der XML-Datei gewählt werden muss, dass sie korrekt interpretiert werden kann:

```
<?xml version="1.0" encoding="utf-8" ?>
<Locations>
  <Location>DHBW Mosbach</Location>
  <Location>Alte Mälzerei</Location>
</Locations>
```

Alle Orte werden zwischen einem *Locations*-Tag definiert. Dabei wird ein neuer Ort jeweils mit einem neuen *Location*-Tag hinzugefügt. Es können beliebig viele neue Orte definiert werden.

### 3.3 Betrieb der Webanwendung

Um die Anwendung zu starten, müssen drei Pflichtparameter übergeben werden. Diese sind:

`-locations <Path>`

Dieser Parameter definiert den Pfad zur XML-Datei, von der die verfügbaren Orte eingelesen werden sollen. Eine Beispieldatei befindet sich im *assets*-Ordner.

`-cert <Path>`

Der Webservice kommuniziert über eine verschlüsselte TLS-Verbindung. Hierfür wird ein gültiges SSL-Zertifikat benötigt. Das SSL-Zertifikat muss dabei entsprechend für die Domain ausgestellt werden, unter der der Service später erreichbar sein wird. Zu Testzwecken und in der Standardkonfiguration wird der Server unter *localhost* gestartet. Ein entsprechendes self-signed SSL-Zertifikat befindet sich ebenfalls im *assets*-Ordner unter *assets/cert.pem*. Mit diesem Parameter muss der Pfad zur Zertifikatsdatei angegeben werden.

`-key <Path>`

Zusätzlich zum Zertifikat wird der dazugehörige private Schlüssel benötigt. Dieser wird i.d.R. zusammen mit dem Zertifikat generiert. Eine Beispieldatei für das Zertifikat für *localhost* ist ebenfalls unter *assets/key.pem* verfügbar. Dieser Parameter gibt den Pfad zum privaten Schlüssel des Zertifikats an.

Soll ein lokaler Webserver zu Testzwecken gestartet werden, kann mit der Standardkonfiguration der folgende Aufruf ausgeführt werden:

```
service -locations=./assets/locations.xml \  
        -cert=./assets/cert.pem -key=./assets/key.pem
```

Zusätzlich zu den Pflichtparametern kann der Webservice durch optionale Parameter konfiguriert werden. Dazu gehören die Verfallszeit eines Einlass-Tokens in Sekunden (*-expire*), der Port, auf dem der Login-Service angezeigt werden soll (*-login-port*), die tatsächliche URL, unter der der Login-Service von außen erreichbar ist (*-login-url*), und der Port, auf dem der QR-Code-Service gestartet werden soll (*-qr-port*).

Eine Übersicht über die Standardwerte einzelner Parameter kann mit dem *-h* Flag eingesehen werden. Die Ausgabe wird im Folgenden ebenfalls gezeigt:

```
-cert path  
    The path to the SSL/TLS certificate file  
-expire int  
    The expire duration for an access token in seconds  
    (default 60)  
-key path  
    The path to the SSL/TLS key file  
-locations path  
    The locations XML file path in the file system  
-login-port int  
    The port the login service should running on  
    (default 4444)  
-login-url url  
    The url encoded in the QR code for the login page  
    (default https://localhost:4444/access)  
-qr-port int  
    The port the QR code service should running on  
    (default 4443)
```

Zuletzt soll an dieser Stelle noch ein Hinweis darauf gegeben werden, dass der private Schlüssel des Servers, der zum Prüfen der Integrität von Cookie-Daten genutzt wird, momentan hart im Sourceco-



de kodiert wird. Soll ein anderer Schlüssel gewählt werden, muss dieser vor dem Kompilieren im Sourcecode geändert werden. Für Testinstanzen kann dieser Umstand vernachlässigt werden, in Produktivsystemen sollte der Schlüssel aber in jedem Fall geändert werden, um die Integrität gesendeter Daten an den Server sicherzustellen.

In Zukunft kann der private Schlüssel beispielsweise über *Environment*-Variablen geladen werden. Für den Testbetrieb wird davon ausgegangen, dass der Benutzer die Cookie-Daten nicht manuell ändert, bzw. mithilfe des private Serverschlüssels ein eigenes Cookie generiert.

### 3.4 Betrieb des Kommandozeilentools

Die einzelnen Funktionen des Kommandozeilentools werden bereits unter Punkt 2.2 genauer erläutert.

Hier sei noch einmal der Hinweis gegeben, dass sich sowohl die Executables *analyzer* und *service* in einem Ordner auf der selben Ebene ausgeführt werden müssen, auf der es ebenfalls einen Ordner *data* gibt, bzw. vom *service* während des Betriebs automatisch angelegt wird, sodass das Analyzer-Tool auch die generierten Journal-Dateien findet.

## 4 Aufgabenverteilung

### 5736465:

Ich habe mich hauptsächlich mit der Webanwendung beschäftigt. Dafür habe ich aus dem *internal*-Package zum großen Teil das *web*-Package und einen kleinen Teil des *journal*-Package implementiert. Im Package *cmd* habe ich einen großen Teil vom *service* implementiert. In der Dokumentation habe ich die Architekturdokumentation und die Anwenderdokumentation geschrieben.

### 5703004:

Ich habe einen Großteil des *journal*-Packages und das *timeutil*-Package implementiert. Außerdem wurde das *convert*-Package inklusive aller daraus hervorgehenden Funktionen implementiert. Im *web*-Package wurde ein Großteil der *Sessions* von mir implementiert, ebenso der ID-Generator für die Generierung (einzigartiger) Ids. Im *cmd*-Package wurde der *analyzer* zu großen Teilen von mir implementiert, inklusive des Schreibens der Journaldateien aus dem *service*. Von der Dokumentation wurde hauptsächlich die Dokumentation des Betriebs geschrieben, in anderen Teilen wurden lediglich einzelne Punkte ergänzt. Für die Dokumentation des Sourcecodes bin ich hauptverantwortlich.

**5736465 und 5703004:**

Allgemein wurden Tests für einzelne Funktionen vom jeweils Implementierenden geschrieben, wobei an einigen Stellen das Schreiben des Tests und die Implementierung der Funktion auch aufgeteilt wurde.

Vor allem die Planungs- und Designphase wurde gemeinsam durchlaufen, sodass hier eine klare Trennung der Ideengebungen nicht möglich ist. Die meisten Funktionen wurden außerdem vom jeweils anderen Studierenden geprüft und reviewed, sodass grobe Programmierfehler ausgeschlossen werden können.