Inhalt

Version	Datum, Zeit	Anmerkungen
0.1	13.10.2020, 08:03	Ergänzung Query-String-Funktionalität, erste Ergänzungen allgemein
0.2	15.10.2020, 09:50	Ergänzung Datentypen in Datenbank-Diagramm, Änderung am Programm wurde vorgenommen: config.json-Datei und -funktionalität wurde hinzugefügt und dokumentiert
0.3	19.10.2020, 16:10	Neue Funktionalitäten hinzugefügt und dokumentiert: -Request zum Ändern des Passworts -Passwortrichtlinien konfigurierbar über config.json
0.4	22.10.2020, 16:10	Funktionalitäten erweitert und dokumentiert: -Autorisierung eines Veranstalters löscht alle nicht-autorisierten Veranstalter-Einträge mit derselben E-Mail -Login-Request gibt alle Berechtigungen an
0.5	25.11.2020, 14:00	Tokens hinzugefügt, Konfigurationsdatei um Token- und Emailverifizierungsparameter erweitert
0.6	09.12.2020, 14:10	Veraltete Information, welche Tokens nicht in Betracht zieht aktualisiert, Kalender anhand von Id abfragen, statt Namen
0.7	10.12.2020, 08:30	Veranstalter anlegen hinzugefügt und automatische Verifizierung
0.8	15.12.2020, 13:30	Funktionalitäten erweitert und dokumentiert: -Request zum Ändern der Veranstalter-Email hinzugefügt Aktualisierung der Konfig-Datei -Email-System freigegeben (kann deaktiviert werden)
0.9	16.12.2020, 15:15	Verlegung von Requests: GET veranstaltungen/<kalenderId> -> GET kalender/<id>/veranstaltungen; POST veranstaltungen/<kalenderId> -> POST kalender/<kalenderId>/veranstaltungen; GET veranstaltungen/<kalenderId>/<uid> -> GET veranstaltungen/<uid>; PUT veranstaltungen/<kalenderId>/<uid> -> PUT veranstaltungen/<uid>; DELETE veranstaltungen/<kalenderId>/<uid> -> DELETE veranstaltungen/<uid>; Korrekturen, Anpassung von Berechtigungen
0.10	17.12.2020, 19:00	Konfigurations-Datei: Parameter-Beschreibungen hinzugefügt, nicht implementierte Veranstaltungsgruppen-Funktionalität erläutert, Datenbank-Diagramm aktualisiert
0.11	23.12.2020, 12:00	Einige Response-Körper ergänzt

Einführung

Diese Dokumentation befasst sich mit der Funktionalität der „AisBuchung_Api“. Dieses ist während der Praktikumszeit von Martti Rasilainen vom 30.6.2020 bis 07.08.2020 entwickelt worden. Sie ist noch nicht im vollendeten Zustand und es besteht keine Garantie, dass an der Api von derselben Person oder überhaupt weitergearbeitet wird.
Da nun vom 5.10.2020 bis zum 23.12.2020 das Frontend des Buchungssystems entwickelt wird, wird auch die API weiterentwickelt. In diesem Zeitraum wird auch die Dokumentation weitergeführt. Fehlende Details sowie Änderungen am Programm werden dabei mit aufgeführt.

Konfigurationsdatei:

In der Datei config.json können einige Änderungen am Verhalten der Api vorgenommen werden.
Sollte die Datei noch nicht existieren, wird sie automatisch erstellt.
Hier ist der Inhalt der Datei mit Standardkonfigurationen:

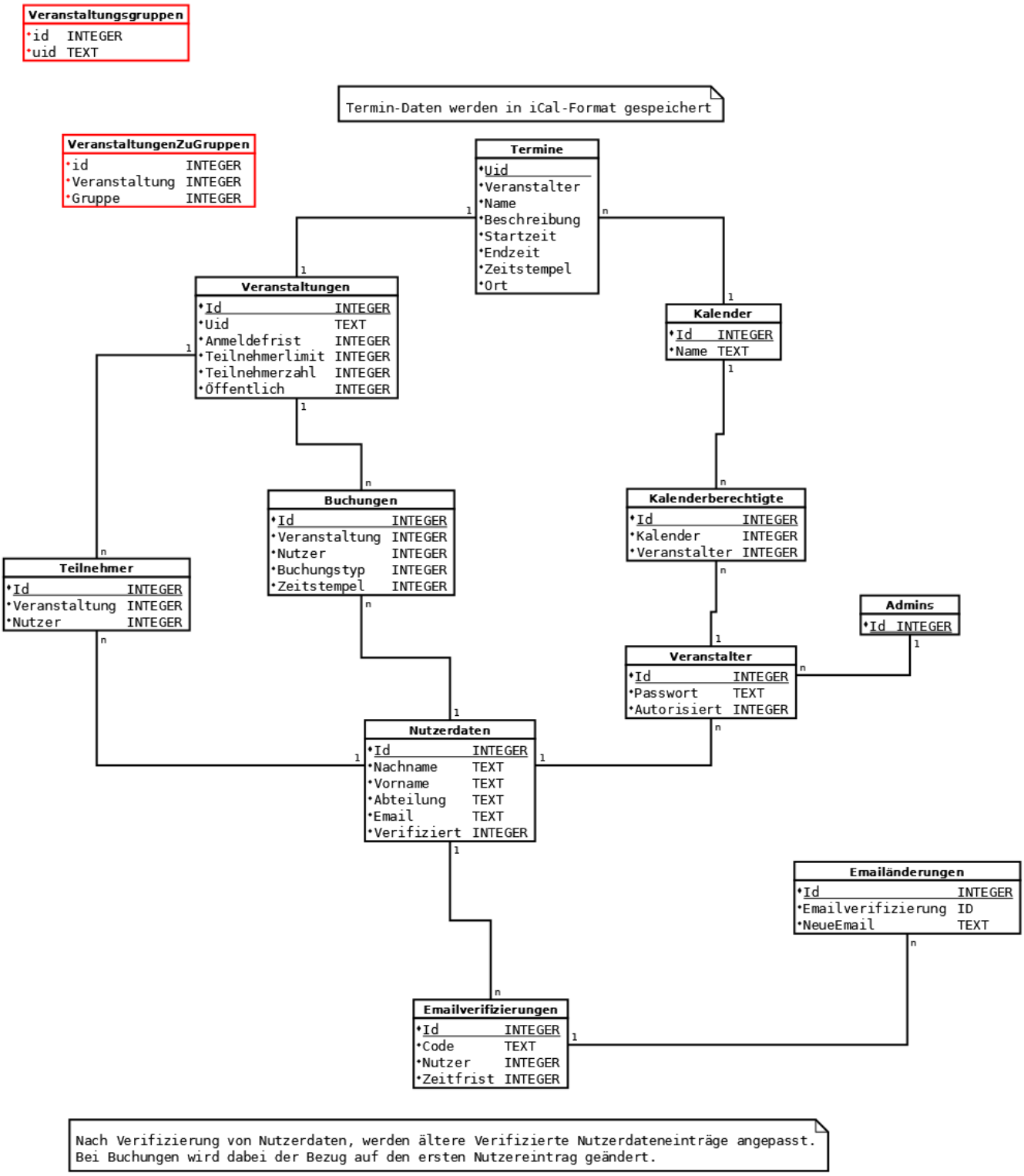
```
{
	"aufbewahrungsfrist": 14,
	"datenbereinigungInterval": 0.25,
	"uidLänge": 8,
	"emailVerifizierung": {
		"emailWirdGesendet": false,
		"verifizierungsfrist": 0.5,
		"verifizierungslink": "frontend.de/verifizieren/",
		"emailAdresse": "mail@bbw.de",
		"emailPasswort": "r23crm0evfiw1",
		"emailHost": "smtp.mail.yahoo.com",
		"emailPort": 587,
		"codeLänge": 8,
		"automatischeVerifizierung": false,
		"adminsKönnenVerifizieren": false
	},
	"debugKonfigurationen": {
		"alleHabenDebugRechte": false,
		"adminsHabenDebugRechte": false,
		"debugBerechtigungErlaubtAlles": false
		"debugKonsoleIstAktiv": false
	}
	"passwortRichtlinien": {
		"erfordertZiffer": true,
		"erfordertGroßbuchstaben": true,
		"erfordertKleinbuchstaben": true,
		"erfordertSonderzeichen": true,
		"mindestlänge": 8
	}
	"tokenKonfiguration": {
		"tokenDauer": 0.5,
		"tokenSchlüssel": "YrtHYvdsZ5v74cn5"
	}
}
```

Gezielte Änderungen an dieser Datei können nicht über die API vorgenommen werden.

Beschreibung aller Konfigurationsparameter:
aufbewahrungsfrist: Gibt an, wie lange Buchungsdaten nach Ablauf der Anmeldefrist gespeichert werden. Angabe in Tagen.
datenbereinigungInterval: Gibt an, in welchen Zeitabständen das Programm alle unnötigen Daten löscht. Angabe in Tagen.
uidLänge: Gibt die Länge der Uid einer Veranstaltung bei Erstellung an. Reicht von 4 bis 32.
emailverifizierung.emailWirdGesendet: Gibt an, ob eine e-Mail zur Verifizierung gesendet wird oder nicht. Dies ist standardmäßig false, da die restlichen Parameter ohnehin angegeben werden müssen.
emailverifizierung.verifizierungsfrist: Gibt an, wie viel Zeit der Nutzer hat um seine e-Mail zu verifizieren. Angabe in Tagen.
emailverifizierung.verifizierungslink: Gibt den Pfad zur Verifizierung über das Frontend an, welcher über die e-Mail-Adresse mitgesendet wird.
emailverifizierung.emailAdresse Gibt die e-Mail-Adresse an, welche das Frontend zum Senden der e-Mail verwendet.
emailverifizierung.emailPasswort: Gibt das Passwort für die Nutzung der e-Mail-Adresse an.
emailverifizierung.emailHost: Gibt den Host der e-Mail-Adresse an.
emailverifizierung.emailPort: Gibt den Port an, welcher benutzt wird.
emailverifizierung.codeLänge: Gibt die Länge des Guid-Codes an, welcher verwendet wird. Reicht von 4 bis 32.
emailverifizierung.automatischeVerifizierung: Gibt an, ob ein Nutzer automatisch verifiziert wird, ohne sich per e-Mail verifizieren zu müssen.
emailverifizierung.emailPort: Gibt an, ob Admins einen Veranstalter automatisch verifizieren können, wenn sie diesen autorisieren.
debugKonfiguration.alleHabenDebugRechte: Gibt an, ob Requests, welche Debug-Berechtigungen erfordern allen Nutzern zur Verfügung stehen.
debugKonfiguration.adminsHabenDebugRechte: Gibt an, ob Requests, welche Debug-Berechtigungen erfordern allen Admins zur Verfügung stehen.
debugKonfiguration.adminsHabenDebugRechte: Gibt an, ob Veranstalter, welche Debug-Requests verwenden können, alle Requests verwenden können.
debugKonfiguration.debugKonsoleIstAktiv: Gibt an, ob die Konsole Debug-Nachrichten ausgibt. **Dies ist derzeit noch nicht implementiert.**
passwortRichtlinien.erfordertZiffer: Gibt an, ob ein Passwort eine Ziffer enthalten muss.
passwortRichtlinien.erfordertGroßbuchstaben: Gibt an, ob ein Passwort einen Großbuchstaben enthalten muss.
passwortRichtlinien.erfordertKleinbuchstaben: Gibt an, ob ein Passwort einen Kleinbuchstaben enthalten muss.
passwortRichtlinien.erfordertSonderzeichen: Gibt an, ob ein Passwort ein Sonderzeichen enthalten muss.
passwortRichtlinien.mindestlänge: Gibt an, wie viele Zeichen ein Passwort mindestens enthalten muss.
tokenKonfiguration.tokenDauer: Gibt an, wie lange ein Token gültig ist. Angabe in Tagen.
tokenKonfiguration.tokenSchlüssel: Gibt den Schlüssel an, welcher zum Erstellen und Verifizieren von Tokens verwendet wird. Er muss mindestens 16 Zeichen beinhalten.

Datenbank

Datenbank-Diagramm:



Entitäten mit roten Rahmen sind nicht implementiert, da sie zu einer Funktionalität gehören, welche derzeit noch nicht implementiert wird.

Veranstaltungsgruppen-Funktionalität (nicht implementiert)

Veranstaltungsgruppen sind eine Funktionalität, welche in einem Gespräch mit dem Auftraggeber Herrn Zenker erwähnt wurden. Diese Funktionalität wird allerdings bis zum Projektende am 23.12.2020 nicht mehr implementiert. Veranstaltungsgruppen sollen mehrere Veranstaltungen zusammenfassen, sodass Nutzer sich an allen dazugehörigen Veranstaltungen über nur einen Link, welcher nicht die Uid einer Veranstaltung sondern die Uid einer Veranstaltungsgruppe beinhaltet.

Berechtigungen:

Login-Daten werden über die Request-Körper gesendet.
Jede Controller-Methode, welche Berechtigungen erfordert, ruft zunächst die zugehörige Methode aus dem AuthModel auf, welche die gesendeten Login-Daten überprüft. Diese befinden sich in den Objekten sämtlicher Klassen, welche von der abstrakten LoginData-Klasse vererben.

Admin-Berechtigung:

Die Admin-Berechtigung gewährt automatisch alle Berechtigung (mit Ausnahme von der Debug-Berechtigung). Administratoren werden in der Admins-Tabelle der Datenbank aufgeführt.

Veranstalter-Berechtigung:

Veranstalter, welche autorisiert und deren Nutzerdaten verifiziert sind erhalten die Veranstalter-Berechtigung. Sie werden in der Veranstalter-Tabelle der Datenbank aufgeführt. Ihr Autorisiert-Wert muss 1 betragen.

Kalender-Berechtigung:

Kalenderberechtigungen werden individuell für jeden Kalender aufgeführt. Diese werden in der Kalenderberechtigte-Tabelle der Datenbank aufgeführt. Kalenderberechtigte und Admins können Änderungen an einem Kalender vornehmen.

Account-Berechtigung:

Über die Account-Berechtigung verfügen nur der Inhaber an dessen Daten Veränderungen vorgenommen werden sollen und alle Administratoren.

Debug-Berechtigung:

Es gibt keine Logik, die unterscheidet, ob Debug-Berechtigungen gegeben sind oder nicht. Derzeit wird bei der Methode `AuthModel.CheckIfDebugPermissions(loginData)` immer `true` zurückgegeben. Methoden, die diese Berechtigung verlangen sollen im Betrieb nicht aufgerufen werden. Sie dienen zum Testen, Warten oder ihre Model-Methoden sollen auf andere Art und Weise aufgerufen werden. Diese Berechtigung kann nur Admins erteilt werden, wenn in der Konfigurationsdatei „adminsHabenDebugRechte“ den Wert `true` besitzt.

Alle Berechtigungen:

Die Methode `AuthModel.CheckIfDebugPermissionsTakePriority(loginData)` gibt zurück, ob Debug-Berechtigungen, dem Anwender alle Berechtigungen bei allen Requests gewährt. Dies ist der Fall, wenn in der Konfigurationsdatei „debugBerechtigungErlaubtAlles“ den Wert `true` besitzt. Sind also „adminsHabenDebugRechte“ und „debugBerechtigungErlaubtAlles“ beide `true` bedeutet dies, dass Admins alle Requests erfolgreich ausführen können.

Http-Requests:

Die erforderlichen Berechtigungen werden in allen Requests angegeben. Diese sind nicht fest in Stein gesetzt. Änderungen an ihnen könnten noch vorgenommen werden.

Login-Requests:

POST login

Login-Daten sind bei allen Requests, welche Berechtigungen erfordern.
Sie sind im Request-Körper enthalten. Dabei kann es sich entweder um E-Mail und Passwort handeln oder um einen Token.
Request-Körper mit Login-Daten:

```
{
	"ml": "<email>",
	"pw": "<passwort>"
}
```

Request-Körper mit Beispiel-Token:

```
{
	"token": "<token>"
}
```

Bei diesem Request werden bei erfolgreicher Autorisierung die Nutzerdaten zurückgegeben, die mit diesen Login-Daten verbunden sind. Auf Anfrage der Frontend-Entwickler werden auch die Berechtigungen des Nutzers angezeigt. Sollte die Authentifizierung bei diesem Request über Login-Daten statt Token stattfinden, wird zusätzlich ein Token erstellt und bei der Response angegeben.
Response-Inhalt bei Statuscode 200 mit Token:

```
{
	"id": <id>,
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"email": "<email>",
	"abteilung": "<abteilung>",
	"verifiziert": <0/1>,
	"autorisiert": <0/1>,
	"veranstalterRechte": <0/1>,
	"adminRechte": <0/1>,
	"debugRechte": <0/1>,
	"alleRechte": <0/1>,
	"token": "<token>"
}
```

Sollten die Login-Daten keinem autorisierten Veranstalter entsprechen, wird der Statuscode 401 zurückgegeben. Der Response-Inhalt enthält jedoch auch dann noch die Rechte des Nutzers.
Response-Inhalt bei Statuscode 401:

```
{
	"veranstalterRechte": <0/1>,
	"adminRechte": <0/1>,
	"debugRechte": <0/1>,
	"alleRechte": <0/1>
}
```

Erfordert Veranstalter-Berechtigung für Statuscode 200.

Admins-Requests:

-

Die Id von Veranstaltern ist identisch mit der Id ihrer Nutzerdaten.

-

GET admins:

Gibt eine Liste mit allen Administratoren und deren Nutzerdaten zurück.
Response-Inhalt:

```
{
	"admins": [
		{<admin (siehe GET admins/<id>)>},
		…
	]
}
```

Erfordert Veranstalter-Berechtigung.

POST admins:

Verleiht einem vorhandenen Veranstalter Admin-Rechte.
Request-Körper:

```
{
	"veranstalter": <id>
}
```

Erfordert Admin-Berechtigung.

GET admins/<id>

Gibt Nutzerdaten eines Administratoren zurück.
Response-Inhalt:

```
{
	"id": <nutzerId>,
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"email": "<email>",
	"abteilung": "<abteilung>",
}
```

Erfordert Veranstalter-Berechtigung.

DELETE admins/<id>

Entzieht Veranstalter Administrator-Rechte.
Erfordert Administrator-Berechtigung.

Buchungen-Requests:

Diese Requests dienen dem Erstellen und Verwalten von Buchungen.

GET buchungen/<uid>

Gibt Buchungen mit Nutzerdaten der Veranstaltung mit der uid zurück.
Erfordert Kalender-Berechtigung.

POST buchungen/<uid>

Buchung zu einer Veranstaltung hinzufügen. Legt Nutzerdaten als neuen unverifizierten Eintrag an. Dadurch wird auch ein neuer Emailverifizierungscode angelegt.
Request-Körper:

```
{
	"buchungstyp": <0/1>,
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"abteilung": "<abteilung>",
	"email": "<email>"
}
```

Buchungstyp 0 für Anmeldung, Buchungstyp 1 für Abmeldung.
Eine Buchung kann nur durchgeführt werden, wenn die Anmeldefrist der Veranstaltung nicht abgelaufen ist und die Teilnehmerzahl kleiner als das Teilnehmerlimit ist oder das Teilnehmerlimit kleiner als 0 ist (siehe Methode `BuchungenModel.CheckIfEventCanBeBooked(eventId)`).
Erfordert keine Berechtigungen.

GET buchungen/<uid>/<id>

Gibt spezifische Buchung zurück.
Erfordert Debug-Berechtigung.

POST buchungen/<uid>/verarbeiten

Verarbeitet alle Buchungen zu einer Veranstaltung.
Derzeit können Buchungen nur über Debug-Requests verarbeitet werden. Die Model-Methoden sind nicht als Teil anderer Requests eingebunden und erfolgen nicht automatisiert.
Erfordert Debug-Berechtigung.

POST buchungen/<uid>/<id>/verarbeiten

Verarbeitet eine spezifische Buchung.
Bei Buchungstyp 0 (Anmeldung) wird der Teilnehmer der Veranstaltung hinzugefügt.
Bei Buchungstyp 1 (Abmeldung) wird der Teilnehmer der Veranstaltung entfernt.
Erfordert Debug-Berechtigung.

POST buchungen/bereinigen

Buchungs-Einträge von Veranstaltungen, deren Anmeldefrist überschritten wurde, werden entfernt.
Anmerkung: Aufgrund des Coronavirus wurde eine Änderung vorgenommen. Derzeit werden Teilnehmer-Einträge erst 14 Tage nach Ablauf des Veranstaltungsdatums gelöscht. Diese Aufbewahrungsfrist wird in der config.json-Datei in Tagen angegeben.
Erfordert Debug-Berechtigung.

Daten-Requests:

Diese Requests werden zur allgemeinen Verwaltung von Daten. Es ist nicht vorgesehen, dass diese Requests im Betrieb ausgeführt werden können. Die Durchführung der Model-Methoden des Bereinigen- und Sichern-Requests sollte automatisiert werden.

POST daten/bereinigen

Führt alle Model-Methoden aus, welche durch
buchungen/bereinigen,
emailverifizierungen/bereinigen,
teilnehmer/bereinigen und
nutzer/bereinigen
aufgerufen werden.
Diese entfernen jeweils unnötige Daten.
Erfordert Debug-Berechtigung.

POST daten/leeren

Die Datenbank- und Kalender-Datei werden in ein neues Verzeichnis unter „Backup\<zeit>“ bewegt.
Erfordert Debug-Berechtigung.

POST daten/sichern

Die Datenbank- und Kalender-Datei werden in ein neues Verzeichnis unter „Backup\<zeit>“ kopiert.
Erfordert Debug-Berechtigung.

Emailverifizierungen-Requests:

Diese Requests stehen zur Verwaltung von Verifizierungscodes zur Verfügung.

GET emailverifizierungen

Gibt alle Emailverifizierungseinträge zurück.
Response-Inhalt:

```
{
	"emailverifizierungen": [
		{<emailverifizierung>},
		…
	]
}
```

Aufbau Emailverifizierung:

```
{
	"id": <id>,
	"code": "<code>",
	"nutzer": <nutzerId>,
	"zeitfrist": <YYYYMMDDhhmm>,
}
```

Erfordert Debug-Berechtigung.

POST emailverifizierungen/<code>/bestätigen

Nimmt Bestätigung des Codes zur Kenntnis. Diesbezügliche Nutzerdaten werden verifiziert.
Sollte es sich um einen Veranstalter handeln und es gibt noch keine Veranstalter, wird dieser automatisch autorisiert.
Sollte kein Administrator vorhanden sein, werden diesem automatisch Administratorrechte verleiht.
Bei Nutzerdaten für eine Buchung wird diese verarbeitet.
Wenn dieser Code zur Änderung einer e-Mail-Adresse generiert wurde, wird diese Änderung am Veranstalter durchgeführt.
Der Code sowie weitere daran gebundene Anfragen werden von der Datenbank gelöscht.
Erfordert keine Berechtigungen.

DELETE emailverifizierungen/<code>

Entfernt Emailverifizierungseintrag mit dem Code.
Erfordert Debug-Berechtigung.

POST emailverifizierungen/<code>/versenden

Versendet Email, welche zur Verifizierung dient.
Das E-Mail-System ist bislang nicht getestet worden, da das Netzwerk jegliche Versuche blockiert.
Erfordert Debug-Berechtigung.

POST emailverifizierungen/bereinigen

Entfernt Emailverifizierungen, deren Zeitfrist abgelaufen ist.
Erfordert Debug-Berechtigung.

Kalender-Requests:

GET kalender

Ruft alle vorhandenen Kalender ab.
Response-Inhalt:

```
{
	"kalender": [
		{
			"id": <id>,
			"name": "<name>"
		}
	]
}
```

Der Inhalt ist derzeit sehr mager. Es wird in Betracht gezogen, den Namen des ersten Veranstalters aufzulisten, welcher in der Kalenderberechtigten-Tabelle zu diesem Kalender eingetragen ist.
Erfordert keine Berechtigungen.

POST kalender

Legt neuen Kalender an.
Request-Körper:

```
{
	"name": "<name>"
}
```

Wenn der Kalender erfolgreich hinzugefügt wurde, antwortet das Backend mit Code 200 und der Id des angelegten Kalenders im Response-Inhalt.
Response-Körper:

```
{
	"id": <id>,
}
```

Erfordert Veranstalter-Berechtigungen

GET kalender/<id>

Ruft Eigenschaften eines spezifischen Kalenders ab.
Reponse-Körper:

```
{
	"id": <id>,
	"name": "<name>"
}
```

Erfordert keine Berechtigungen.

PUT kalender/<id>

Aktualisiert Kalendereigenschaften.
Request-Körper:

```
{
	"name": "<name>"
}
```

Erfordert Kalender-Berechtigung.

DELETE kalender/<id>

Entfernt vorhandenen Kalender.
Erfordert Kalender-Berechtigung.

GET kalender/<id>/veranstalter

Gibt Veranstalter zurück, welche die Kalenderberechtigungen zu dem spezifischen Kalender besitzen.
Erfordert keine Berechtigungen.

GET kalender/<id>/veranstaltungen

Gibt alle Veranstaltungen des Kalenders zurück.
Erfordert Veranstalter-Berechtigung.

POST kalender/<id>/veranstaltungen

Fügt einem Kalender eine Veranstaltung hinzu.
Request-Körper:

```
{
	"name": "<name>",
	"beschreibung": "<beschreibung>",
	"datum": <YYYYMMDD>,
	"startzeit": <hhmm>,
	"endzeit": <hhmm>,
	"ort": "<ort>",
	"teilnehmerlimit": <anzahl>,
	"anmeldefrist": <YYYYMMDDhhmm>,
	"öffentlich": <0/1>
}
```

Die Datums- und Zeitangaben müssen als Integer im Request-Körper vorhanden sein, wobei Anmeldefrist ein Long sein muss. Dabei muss bei einem Integer die Vorgabe eingehalten werden, dass die erste Ziffer keine 0 sein darf.
Der „öffentlich“-Parameter erfüllt derzeit keinen Zweck. Theoretisch kann ein Frontend dennoch Gebrauch von diesem machen und per QueryString-Funktion spezifisch nach öffentlichen Veranstaltungen fragen.
Response-Körper:

```
{
	"uid": "<uid>"
}
```

Erfordert Kalender-Berechtigung.

POST kalender/<id>/veranstalter

Verleiht einem vorhandenen Veranstalter Berechtigungen zu dem spezifischen Kalender.
Request-Körper:

```
{
	"veranstalter": <id>
}
```

Erfordert Kalender-Berechtigung.

DELETE kalender/<id>/veranstalter/<id>

Entzieht Veranstalter die Kalenderberechtigung.
Erfordert Kalender-Berechtigung.

Nutzer-Requests:

Nutzer-Requests sind zur Verwaltung von Nutzerdaten zu gebrauchen. Dabei spielt es keine Rolle, ob es sich um Teilnehmer- oder Veranstalter-daten handelt. Diese Requests sind nur zu Debug-Zwecken zu Verwenden. Derzeit ist nicht vorgesehen, dass sie im Betrieb verwendet werden können.

GET nutzer

Ruft alle vorhandenen Nutzerdaten ab.
Response-Inhalt:

```
{
	"nutzer": [
		{<nutzer (siehe GET nutzer/<id>)},
		…
	]
}
```

Erfordert Debug-Berechtigung.

GET nutzer/<id>

Ruft spezifische Nutzerdaten ab.
Response-Inhalt:

```
{
	"id": <id>,
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"email": "<email>",
	"abteilung": "<abteilung>",
	"verifiziert": <0/1>
}
```

Erfordert Debug-Berechtigung.

PUT nutzer/<id>

Ändert vorhandene Nutzerdaten.
Request-Körper:

```
{
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"abteilung": "<abteilung>",
	"email": "<email>"
}
```

Erfordert Debug-Berechtigung.

POST nutzer/<id>/verifizieren

Setzt die Verifizierung von einem Nutzerdaten-Eintrag zu 1.
Erfordert Debug-Berechtigung.

POST nutzer/bereinigen

Entfernt Nutzerdaten, auf die weder Buchungen, Teilnehmer noch Veranstalter Bezug nehmen.
Erfordert Debug-Berechtigung.

Teilnehmer-Requests:

Diese Requests dienen zur Verwaltung vorhandener Teilnehmer. Diese können nicht direkt hinzugefügt werden. Teilnehmer müssen sich zunächst über Buchungs-Requests anmelden.

GET teilnehmer

Gibt alle Teilnehmereinträge zurück, unabhängig von der Veranstaltung.
Erfordert Debug-Berechtigung.

POST teilnehmer/bereinigen

Entfernt alle Teilnehmer-Einträge zu Veranstaltungen, welche nicht mehr vorhanden oder abgelaufen sind.
Anmerkung: Aufgrund des Coronavirus wurde eine Änderung vorgenommen. Derzeit werden Teilnehmer-Einträge erst 14 Tage nach Ablauf des Veranstaltungsdatums gelöscht. Diese Aufbewahrungsfrist wird in der config.json-Datei in Tagen angegeben.
Erfordert Debug-Berechtigung.

GET teilnehmer/<uid>

Gibt alle Teilnehmereinträge einer Veranstaltung zurück.
Response-Körper:

```
{
	"teilnehmer": [
		{
			"id": <id>,
			"vorname": "<vorname>",
			"nachname": "<nachname>",
			"abteilung": "<abteilung>",
			"email": "<email>"
		},
		{…}
	]
}
```

Erfordert Kalender-Berechtigung.

POST teilnehmer/<uid>

Fügt einen Teilnehmereintrag hinzu.
Request-Körper:

```
{
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"abteilung": "<abteilung>",
	"email": "<email>"
}
```

Erfordert Debug-Berechtigung.

DELETE teilnehmer/<uid>/<id>

Entfernt Teilnehmer einer Veranstaltung.
Erfordert Kalender-Berechtigung.

Veranstalter-Requests:

Die Id von Veranstaltern ist identisch mit der Id ihrer Nutzerdaten.

GET veranstalter

Gibt alle Veranstalter-Einträge zurück.
Response-Inhalt:

```
{
	"veranstalter": [
		{<veranstalter (siehe GET veranstalter/<id>)},
		…
	]
}
```

Um ausschließlich einen Array mit autorisierten Veranstalter zu erhalten, kann die QueryString-Funktionalität verwendet werden: `veranstalter?autorisiert=1`.
Erfordert Veranstalter-Berechtigung.

POST veranstalter

Registriert unautorisierten Veranstalter und legt unverifizierte Nutzerdaten an.
Legt Nutzerdaten als neuen unverifizierten Eintrag an. Dadurch wird auch ein neuer Emailverifizierungscode angelegt.
Request-Körper:

```
{
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"abteilung": "<abteilung>",
	"email": "<email>",
	"passwort": "<passwort>"
}
```

Erwarteter Rückgabecode ist 200. 201 wird nicht zurückgegeben, da vor Verifizierung und Autorisierung, der Nutzer noch keine Veranstalter-Berechtigung hat und die URI des Veranstalters nutzlos ist.
Falls das Passwort nicht den Richtlinien in der Konfigurationsdatei entspricht oder die E-Mail-Adresse bereits vergeben ist, wird Statuscode-400 zurückgegeben mit folgendem Response-Inhalt:

```
{
	"errorMessage": "<Fehlermeldung>"
}
```

Erfordert keine Berechtigungen.

POST veranstalter/anlegen

Registriert verifizierten und autorisierten Veranstalter.
Erfordert Admin-Berechtigungen.

GET veranstalter/<id>

Gibt Daten eines Veranstalter-Eintrags zurück.
Response-Inhalt:

```
{
	"id": <nutzerId>,
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"email": "<email>",
	"abteilung": "<abteilung>",
	"verifiziert": <0/1>,
	"autorisiert": <0/1>
```

```
}
```
Erfordert Veranstalter-Berechtigung.

PUT veranstalter/<id>
Ändert Nutzerdaten eines Veranstalters.
Request-Körper:
```
{
	"vorname": "<vorname>",
	"nachname": "<nachname>",
	"abteilung": "<abteilung>",
}
```
Die e-Mail-Adresse kann nur über POST veranstalter/<id>/email geändert werden.
Erfordert Account-Berechtigung.

DELETE veranstalter/<id>
Entfernt Veranstalter-Eintrag.
Derzeit können nur über diesen Request Veranstalter-Einträge entfernt werden, welche möglicherweise durch unerwünschte Personen geschaffen wurden.
Erfordert Account-Berechtigung.

GET veranstalter/<id>/kalender
Gibt Kalender zurück zu denen der Veranstalter berechtigt ist.
Response-Inhalt:
```
{
	"kalender": [
		{<kalender>},
		…
	]
}
```

Erfordert Veranstalter-Berechtigung.

POST veranstalter/<id>/autorisieren
Setzt die Autorisierung von einem Veranstalter-Eintrag zu 1 um Legitimität zu bestätigen.
Diesem Veranstalter können nun Berechtigungen gewährt werden.
Der erste Veranstalter, dessen Nutzerdaten verifiziert werden, wird automatisch autorisiert.
Der erste autorisierte Veranstalter wird zudem zum ersten Admin.
Sollten Veranstalter-Einträge vorhanden sein, die nicht autorisiert werden und dieselbe E-Mail-Adresse enthalten, werden diese entfernt.
Erfordert Veranstalter-Berechtigung.

POST veranstalter/<id>/passwort
Über diesen Request kann das Passwort eines Veranstalters geändert werden.
Request-Körper:
```
{
	"altesPasswort": "<altesPasswort>",
	"neuesPasswort": "<neuesPasswort>",
}
```
Auch dieser Request verwendet „ml“ und „pw“ oder den „token“ als Login-Daten und überprüft auf Berechtigungen. Handelt es sich bei der eingeloggten Person um den Veranstalter dessen Passwort geändert werden soll, wird mit „altesPasswort“ explizit noch einmal nach dem Passwort gefragt. Stimmt dieses mit dem der Datenbank überein, kann das Passwort geändert werden. Handelt es sich bei der eingeloggten Person um einen Admin, spielt „altesPasswort“ bei der Bearbeitung des Requests keine Rolle.

Damit die Änderung durchgeführt werden kann, muss „neuesPasswort“ den Passwort-Richtlinien entsprechen.
Die Passwortrichtlinien können in der Konfigurations-Datei config.json angepasst werden. Standardmäßig sind mindestens ein Sonderzeichen, ein Groß- und ein Kleinbuchstaben, eine Ziffer und eine Mindestlänge von acht Zeichen erforderlich. Sollte das neue Passwort nicht den Richtlinien entsprechen, wird Statuscode 400 zurückgegeben mit einer Fehlernachricht welche das nicht erfüllte Kriterium beschreibt.
Ebenfalls wird eine Fehlernachricht zurückgegeben, wenn bereits ein autorisierter Veranstalter mit derselben E-Mail-Adresse vorhanden ist.
Response-Inhalt bei einem Bad Request.

```
{
	"errorMessage": "<Fehlermeldung>"
}
```

Erfordert Account-Berechtigung mit Angabe des alten Passworts oder Admin-Berechtigung ohne Angabe des alten Passworts.

POST veranstalter/<id>/email

Über diesen Request kann die e-Mail-Adresse eines Veranstalters geändert werden.
Request-Körper:

```
{
	"neueEmail": "<e-Mail-Adresse>"
}
```

Existiert bereits ein Veranstalter mit der neuen Adresse, wird ein BadRequest zurückgegeben.
Erfordert Account-Berechtigung mit Bestätigung über neue e-Mail-Adresse. Erfordert Admin-Berechtigung für direkte Änderung der Adresse.

Veranstaltungen-Requests:

Jede Veranstaltung hat eine Unique Id (Uid). Diese wird mithilfe von Guid generiert. Dabei wird jedoch nicht zwangsläufig die gesamte Länge der Uid übernommen. In der Konfigurationsdatei kann mit „uidLänge“ diese Länge angepasst werden. Ihr Standardwert ist 8. Ihr Mindestwert beträgt 4 und ihr Höchstwert 36. Beispiel für Uid mit Länge 8: "bfbd8ece". Beispiel für Uid mit maximaler Länge (36): "67a2087b-fd6a-4a1e-aa9e-b0357c501da3".

GET veranstaltungen

Gibt alle Veranstaltungen zurück.
Erfordert Veranstalter-Berechtigungen.

GET veranstaltungen/<uid>

Gibt eine spezifische Veranstaltung zurück.
Response-Inhalt:

```
{
	"uid": "<uid>",
	"name": "<name>",
	"beschreibung": "<beschreibung>",
	"ort": "<ort>",
	"datum": <YYYYMMDD>,
	"startzeit": <hhmm>,
	"endzeit": <hhmm>,
	"jahr": <YYYY>,
	"monat": <MM>,
	"tag": <DD>,
	"teilnehmerzahl": <anzahl>,
	"teilnehmerlimit": <anzahl oder niedriger als 0 für unbegrenzt>,
	"anmeldefrist": <YYYYMMDDhhmm>,
	"öffentlich": <0/1>
}
```

Datums- und Zeitangaben werden als Integer zurückgegeben. Hierbei ist zu beachten, dass die Anzahl an Ziffern somit nicht fix ist. Beispiel: 08:30 Uhr -> 830.
Erfordert keine Berechtigung.

PUT veranstaltungen/<uid>

Aktualisiert Daten der Veranstaltung.
Request-Körper wie bei POST.
Erfordert Kalender-Berechtigung.

DELETE veranstaltungen/<uid>

Entfernt Veranstaltung.
Erfordert Kalender-Berechtigung.

Query-String-Funktionalität

Einige Get-Requests machen von einer Methode Gebrauch, welche Response-Inhalte filtern können. Die Methode ist hierbei `Json.QueryJsonData(…)` aus der JsonSerializer.dll. Sie filtert anhand des Query-Strings in der Url. Die Syntax ist hierbei: „var1=val1;var2=val2“. Somit werden nur Objekte aufgelistet, wo die Werte der Variablen den Anforderungen im Query entsprechen.
Es kann auch nach Größer-Gleich und Kleiner-Gleich gefiltert werden, wenn es sich um numerische Werte handelt. Die Syntax hierzu ist:
„var1={"lt": 100};var2={"gt": 30};var3={"lt": 100, "gt": 30}“.
Somit können z. B. Kalendereinträge nach dem Datum gefiltert werden.