# Dokumentation

**Autoren:**

Anna Lopatkina, Ella Hirche, Fabian Wolf

## Aufgabe und Zielstellung

**Aufgabe des Systems**:

Das Hauptziel unseres Gesamtsystems ist die Darstellung und Verwaltung von Studiengängen und dem zugehörigen Kursangebot im Kontext eines einzelnen Nutzers.
Der einzelne Nutzer soll also, seiner Rolle (zum Beispiel Student, Admin) entsprechend, die Möglichkeiten haben mit dem System auf verschiedene Art und Weise zu interagieren.
Daraus ergeben sich zwei Teilziele.
Zum einen müssen die Studiengänge samt ihres Lehrangebotes zur Verfügung gestellt und bearbeitbar sein. Zum Anderen müssen Nutzer mit ihren entsprechenden Nutzungsrechten und personalisierten Anzeigen verwaltbar und nutzbar gemacht werden.

**Grobe Struktur des Systems**:

Um die genannten beiden Teilziele zu erreichen, werden zwei verschiedene Services (Dienste) zur Verfügung gestellt. Der erste Service bietet die Schnittstelle zur Nutzerverwaltung, unabhängig vom Lehrangebot an. Der zweite Service bietet eine Schnittstelle zur Speicherung, Ausgabe und Verwaltung von Studiengängen einer Universität mit ihren zugehörigen Lehrangeboten, unabhängig vom Nutzer des Services, an.
Die Aufgabe des Clients ist es damit, die isolierten Services miteinander zu verbinden um das Gesamtziel des Systems zu erreichen.

**Anwendungsumfeld**:

Der entwickelte Service soll im universitären Umfeld Anwendung finden. Die Zielgruppe umfasst sowohl die Organisatorische Einheit der Universität (Mitarbeiter von Lehrstühlen/Prüfungsamt), als auch die Studierenden. Um für beide Hauptnutzergruppen die relevanten Funktionen zu Verfügung zu stellen, gibt es eine entsprechende Nutzerverwaltung. Es ist vorstellbar, dass sich die Zielgruppe mit neu eingepflegten Funktionen erweitert, was auch wieder durch die (isolierte) Nutzerverwaltung ermöglicht wird.

## Das Team

Das Entwicklerteam besteht aus Fabian Wolf, Ella Hirche und Ania Lopatkina.

**Verantwortlichkeiten**:

An vielen Stellen hatten wir keine ganz klare Aufgabenverteilung, sondern haben uns gegenseitig unter die Arme gegriffen. Das Konzept haben wir zum Beispiel von vornherein im Team entwickelt. Den Client hat Fabian komplett alleine entwickelt, für die Services waren Ella und Ania Schwerpunktmäßig zuständig, Fabian hat sich im späteren Verlauf allerdings um eine Umstrukturierung gekümmert. Die zugehörigen API Dokumentationen haben entsprechend auch Ella und Ania gemacht, die textuelle Dokumentation hat Schwerpunktmäßig Ella durchgeführt. Fabian hatte in dem Projekt eine besondere Rolle, weil er sich um die fachliche Leitung gekümmert hat.

# Eingrenzung des Problemraums

## Stakeholder-Modell

In der folgenden Tabelle werden die Interessengruppen dargestellt, die bei dem Entwurf und der Umsetzung der Services und des Clients eine Rolle spielen. Die Services und der Client werden hier als Gesamtsystem betrachtet, welches am Ende zur Verfügung steht.

| Interessengruppe | Beschreibung | Interesse |
|---|---|---|
| Management der Universität | Die Verantwortlichen für den Betrieb der Universität | Geringe Kosten und Verwaltungsaufwand |
| Lehrstuhlverantwortliche / Prüfungsamt | Die Verantwortlichen für das Einpflegen von Studiengängen | Anforderung an den Client zur Studiengangverwaltung: Benutzerfreundlichkeit - muss simpel zu bedienen sein |
| Administrator | Die Person, die für Wartung und Verwaltung des Systems verantwortlich ist | Service zur Nutzerverwaltung muss übersichtlich sein; |
| Entwickler | Personengruppe, die für die Implementierung und Wartung dieses und zukünftiger Systeme im Rahmen der Universität verantwortlich sind | Einfache Wartbarkeit, Einfache Erweiterbarkeit, gute Isolierung verschiedener Dienste, Wiederverwendbarkeit |
| Studierende | Client-Nutzer, die ihren Studiengang und Noten einsehen möchten | Ansprechende und einfach zu bedienende Benutzeroberfläche |

## Glossar (Domänenmodell)

In der folgenden Tabellen sind die wichtigsten Begriffe des Systems beschrieben. Das Glossar soll die Verständlichkeit der beschriebenen Aufgaben, Funktionen und Umsetzung unterstützen.

| Begriff/Klasse | Beschreibung |
| --- | --- |
| Admin | Eine bestimmte Rolle, die vom Nutzer (User) eingenommen werden kann. Es wird vor der ersten Nutzung ein Admin-Account durch das System initialisiert. |
| Benutzer-Dienst (User-Service) | Dienst, der die Verwaltung und Speicherung von Nutzern über eine Schnittstelle zur Verfügung stellt |
| Klient (Client) | Webanwendung die die beiden Dienste nutzt, um einen personalisierten Zugriff auf Studiengangdaten zu ermöglichen |
| Lehrveranstaltung (Lecture) | Kleinste Organisationseinheit der Lehre eines Studienganges; Eine oder meherere Lehrveranstaltungen bilden ein Modul. Auf jede Lehrveranstaltung kann einzeln eine Note vergeben werden. |
| Modul (Module) | Ein Modul ist eine Organisationseinheit eines Studiums, welche ein oder mehrere Lehrveranstaltung zusammenfasst. Auf jedes Modul gibt es eine Gesamtnote. |
| ModulesLectures | Datenbank die die Relation (Many-to-Many) zwischen Lehrveranstaltungen und Modulen beschreibt. Es ist jeweils eine Lehrveranstaltung einem Modul über die jeweiligen Primärschlüssel (IDs) zugeordnet. Dabei können sowohl Lehrveranstaltungen als auch Module mehrfach zugeordnet werden. |
| Nutzer (User) | Ein Nutzer (User) ist eine Instanz die auf das System mittels Nutzeraccount zugreifen kann. Ein Nutzer wird mittels Registrierung zur Laufzeit angelegt. Die Verwaltung eines Nutzers wird durch den Nutzer-Dienst (User-Service) zur Verfügung gestellt. |
| Nutzer-Token (User-Token) | Der Nutzer-Token ist ein Token, der bei der Registrierung angelegt wird und bei jeder Anmeldung eines Nutzers verwendet wird. |
| Rolle (Role) | Es gibt verschiedene Rollen, die Nutzer haben können. Insbesondere gibt es die Rolle Admin und die Rolle Studierender. Bestimmte Nutzunsrechte werden bestimmten Rollen zugeteilt. |
| Studiengang (Study) | Ein Studiengang ist eine Organisationseinheit einer Universität, welche Module zusammenfasst, die abgelegt werden müssen, um ein bestimmten Abschluss zu erreichen. |
| Studiengang-Dienst (Study-Service) | Dienst, der den Zugriff, die Speicherung und Verwaltung der Studiengänge mit ihren Kursdaten zur Verfügung stellt |

| Begriff/Klasse | Beschreibung |
| --- | --- |
| Studierender | Eine bestimmte Rolle, die vom Nutzer (User) eingenommen werden kann. Nutzer, die die Rolle Studierender haben, können zur Laufzeit von jedem über die Registrierung angelegt werden. |
| StudiesModules | Datenbank die die Relation (Many-to-Many) zwischen Modulen und Studiengängen beschreibt. Es ist jeweils ein Module einem Studiengang über die jeweiligen Primärschlüssel (IDs) zugeordnet. Dabei können sowohl Module als auch Studiengänge mehrfach zugeordnet werden. |
| Studiengang-Token (Study-Token) | Der Studiengang-Token ist ein Token, der von authorisierten Rollen (insbesondere dem Admin) generiert werden kann und von ihnen zum ändern und hinzufügen von Studiengängen und deren Lerhveranstaltungen genutzt wird. |
| Token | Ein Token ist ein Schlüssel, der neben dem Passwort für den Zugang zu bestimmten Bereichen im System von einem Nutzer vorzuweisen ist. |

# Anforderungen

## Funktionale Anforderungen

In der folgenden Tabelle sind die funktionellen Anforderungen beschrieben, die an den Client gestellt wurden. In der letzten Spalte sind außerdem die daraus resultierenden Anforderungen an den entsprechenden Dienst aufgeführt. Diese sind nicht zwangsläufig funktional.

| **Funktionale Anforderung an den Client** | **Beschreibung** | **Anforderung an die Dienste** |
|---|---|---|
| Registrierung Nutzer | • Es soll für jeden unangemeldeten Anwender möglich sein, sich über den Client als neuer Nutzer zu registrieren | • Der Nutzer-Dienst muss Registrierungsdaten entgegennehmen und persistent in Form eines neuen Nutzers speichern können</li/ul> |
| Anmeldung Nutzer | • Es soll für einen Anwender mit gültigen Zugangsdaten möglich sein, sich als Nutzer anzumelden | • Der Nutzer-Dienst muss auf Anfrage Anmeldedaten mit persistent vorliegenden Nutzern abgleichen und verifizieren können |
| Anmeldung Admin | • Es soll für einen Anwender mit Administratorrechten schon zum Anwendungsstart möglich sein, sich mit gültigen Zugangsdaten als Admin anzumelden | • Der Nutzer-Dienst muss auf Anfrage Anmeldedaten mit persistent vorliegenden Admindaten abgleichen und verifizieren können |
| Profil darstellen und bearbeiten | • Es soll für jeden angemeldeten Nutzer möglich sein, sein Profil detailliert anzeigen zu lassen<br>• In der gleichen Ansicht soll es möglich sein, das Profil des zugehörigen eigenen Nutzers zu bearbeiten | • Der Nuter-Dienst soll auf Anfrage Profildaten für einen Nutzer zur Verfügung stellen<br>• Der Nutzer-Dienst soll af Anfrage geänderte Profildaten persistent speichern |

| **Funktionale Anforderung an den Client** | **Beschreibung** | **Anforderung an die Dienste** |
|---|---|---|
| Module darstellen | • Ein angemeldeter Nutzer soll sich eine Übersicht über die Modudle anzeigen lassen können, die zu seinem Studiengang gehören<br>• Diese Darstellung soll sowohl in Listenform, als auch in erweiterter detaillierter Listenform möglich sein, in der auch einzelne Lehrveranstaltungen dargestellt werden | • Der Studiengang-Dienst soll auf Anfrage alle Module und Lehrveranstaltungen eines Studiengangs ausgeben |
| Hinzufügen Studiengang | • Es muss für einen Admin möglich sein einen Studiengang hinzuzufügen | • Der Studiengang-Dienst muss auf Anfrage einen neuen Studiengang persistent hinterlegen<br>• Der Nutzer-Dienst muss auf Anfrage Admin-Zugangsdaten verifizieren können |
| Detailansicht Studiengang | • Es muss für einen Admin möglich sein, einen Studiengang in Detailansicht mit seinen Modulen anzeigen zu lassen | • Der Studiengang-Dienst muss auf Anfrage Studiengangdaten (Eigenschaften, Module und Lehrveranstaltungen) eines existierenden Studienganges ausgeben |
| Änderung Studiengang | • Es muss für einen Admin möglich sein , einen Studiengang zu ändern | • Der Studiengang-Dienst muss auf Anfrage einen gegebenen Studiengang persistent ändern |

| **Funktionale Anforderung an den Client** | **Beschreibung** | **Anforderung an die Dienste** |
|---|---|---|
| Hinzufügen von Modul zu Studiengang | • Es muss für einen Admin möglich sein , ein Modul zu einem Studiengang hinzuzufügen | • Der Studiengang-Dienst muss auf Anfrage persistent ein neues Modul speichern und zu einem gegebenen Studiengang hinzufügen |
| Bearbeitung Modul | • Es muss für einen Admin möglich sein , ein Modul zu ändern | • Der Studiengang-Dienst muss auf Anfrage ein gegebenes Modul persistent ändern |
| Hinzufügen von Lehrveranstaltung zu Modul | • Es muss für einen Admin möglich sein , eine Lehrveranstaltung zu einem Modul hinzuzufügen | • Der Studiengang-Dienst muss auf Anfrage persistent eine neue Lehrveranstaltung speichern und zu einem gegebenen Modul hinzufügen |
| Darstellung Nutzer | • Es muss für den Admin möglich sein, sich einen Überblick über alle Nutzer im System anzeigen zu lassen | • Der Nutzer-Dienst muss auf Anfrage alle Nutzer des Systems ausgeben<br>• Der Nutzer-Dienst muss auf Anfrage Admin-Zugangsdaten verifizieren können |
| Hinzufügen Rolle | • Es muss für den Admin möglich sein, eine noch nicht existierende (Nutzer-)Rolle hinzuzufügen | • Der Nutzer-Dienst muss auf Anfrage Admin-Zugangsdaten verifizieren können<br>• Der Nuzer-Dienst muss auf Anfrage eine neue Rolle persisent anlegen, wenn sie noch nicht existiert |

| Funktionale Anforderung an den Client | Beschreibung | Anforderung an die Dienste |
|---|---|---|
| Generierung Studiengang-Token | • Der Admin muss mit gültigem Token-Generierungs-Zugang in der Lage sein, einen Studiengang-Token zu generieren | • Der Nutzer-Dienst muss auf Anfrage Token-Generierungs-Zugangsdaten verifizieren können |

### Nicht funktionale Anforderungen

| Anforderung | Beschreibung |
|---|---|
| Sicherheit | Die Kommunikation mit Nutzern des Client soll mittels HTTPS/TLS gesichert sein. |

### Semi-Funktionale Anforderungen

| Anforderung | Beschreibung |
|---|---|
| Sicherheit | Der Client soll mit dem Nutzerservice durch eine tokenbasierte Authentifizierung kommunizieren. Darüber hinaus soll der administrative Zugriff auf den Studyservice mit einem Token, welches durch einmalige Eingabe eines Passwortes generiert wird, erfolgen. |

## Umsetzung und Entwicklungsentscheidungen

Das System besteht aus zwei Diensten (Services): dem Nutzer-Dienst (User-Service) und dem Studiengang-Dienst (Study-service); Außerdem gibt es einen Webclient, der diese Dienste nutzt um die Speicherung, Verwaltung und Ausgabe von Studiengängen personalisiert für Nutzer zur Verfügung zu stellen.

Es gibt mehrere Gründe für diese Struktur. Der Client ist als Webclient realisiert, damit ein einfacher, intuitiver und mobiler Zugriff für alle Nutzergruppen erfolgen kann. Mit dem Client kann der Anwender, der im System als Nutzer agiert, direkt interagieren. Der Client ist dafür verantwortlich die beiden Dienste so zu nutzen, dass nur authorisierte Anwender die entsprechenden Funktionen nutzen können. Er muss also unter der direkten Aufsicht der Universität stehen.

Es gibt zwei verschiedene Dienste, um die Nutzerverwaltung von der Verwaltung der Studiengänge zu isolieren. Beide Systeme sind also unabhängig voneinander erweiterbar und außerhalb unseres ganz speziellen Anwendungskontextes wiederverwendbar. So ist zum Beispiel vorstellbar, dass der gleiche Service zur Nutzerverwaltung auch an anderer Stelle innerhalb der Universität verwendet wird.

## Technisches

Sowohl für den Client, als auch für die Services wurde das Framework Flask, welches auf Python basiert, verwendet. Der Hauptgrund für diese Entscheidung war die gute Erweiterbarkeit des Frameworks, welche wir sowohl für den Webclient, als auch für die Services genutzt haben. So nutzten wir clientseitig zum Beispiel die Handhabung von Sessions und serviceseitig SQLAlchemy Toolkit für SQLite.
Außerdem fiel die Entscheidung für Flask mit der Entscheidung für einen REST Stil. Der Client sollte aktiv die beiden Dienste nutzen und zusammenführen. Außerdem sollte unsere Anwendung ressourcenorientiert sein, weil sowohl die Nutzerverwaltung, als auch die Studiengangverwaltung auf konzeptionellen Listen basiert.

## Umsetzung des Clients

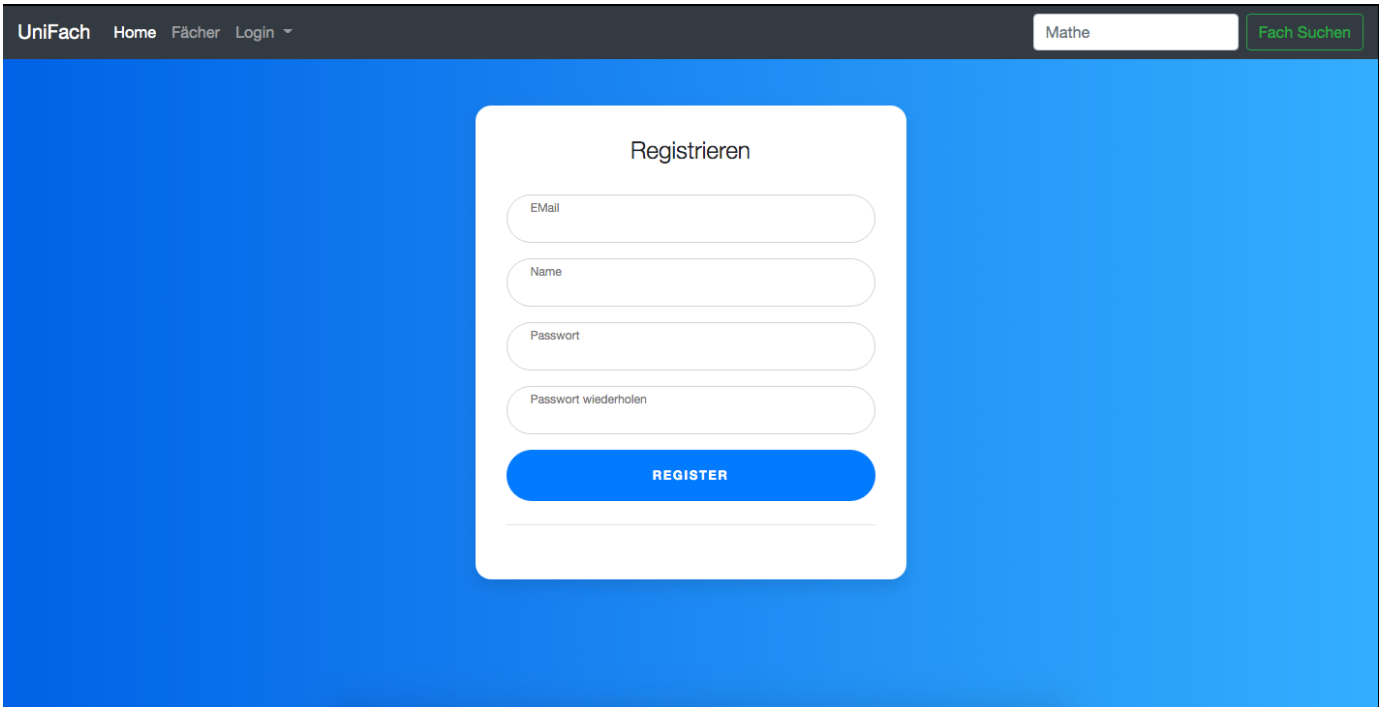
### Bedienung des Clients

- Siehe GUI
- Standardlogin Admin: admin@fwao.de (mailto:admin@fwao.de) / admin (konfigurierbar, siehe Abschnitt *Konfiguration*)

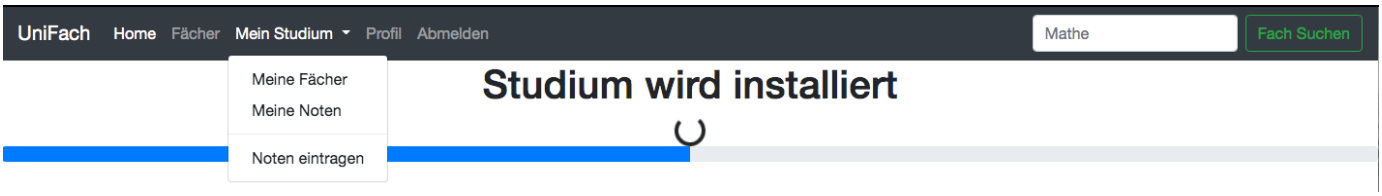
**GUI des Clients**:
Nachfolgend ist die GUI des Clients dargestellt und führt durch die verschiedenen Möglichkeiten, das Gesamtsystem zu nutzen.

**Die Navigationsleiste, wenn kein Nutzer angemeldet ist:**

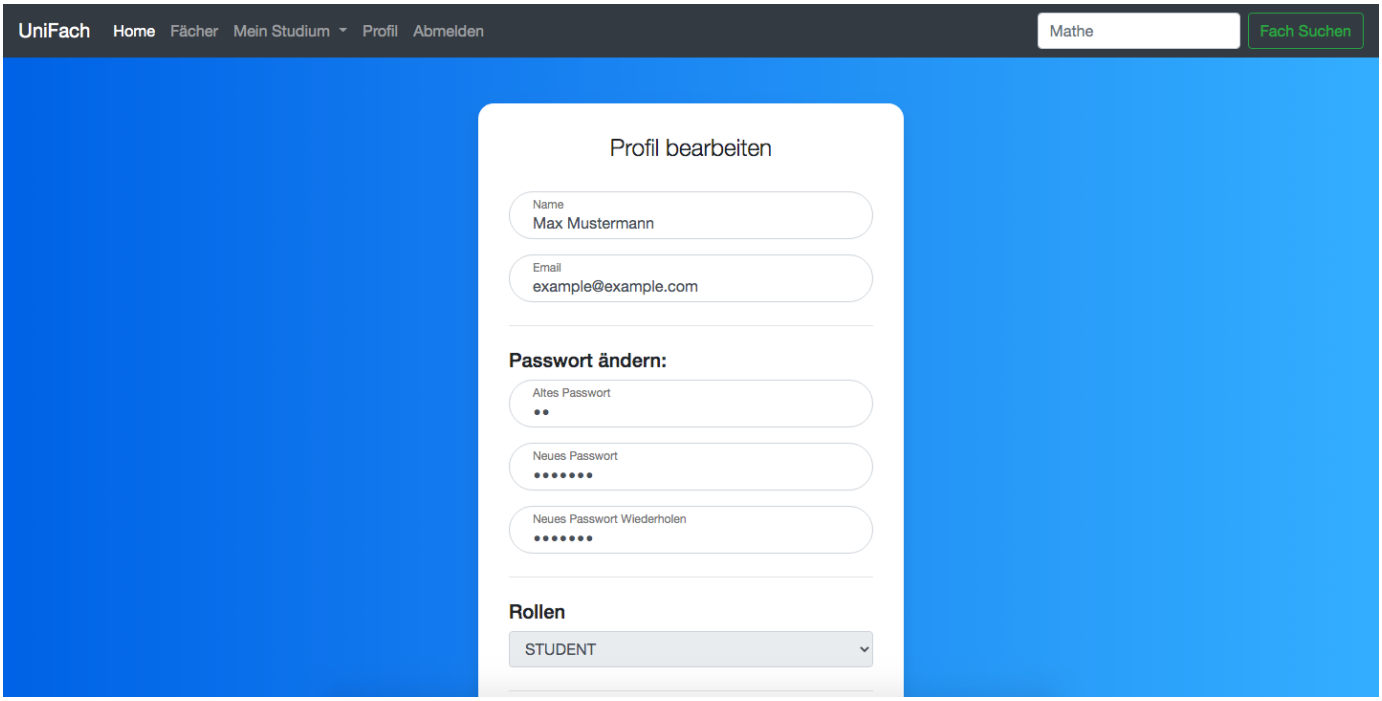

**Die Registrierung eines neuen Nutzers; Auf diese Seite kann auch unangemeldet zugegriffen werden:**

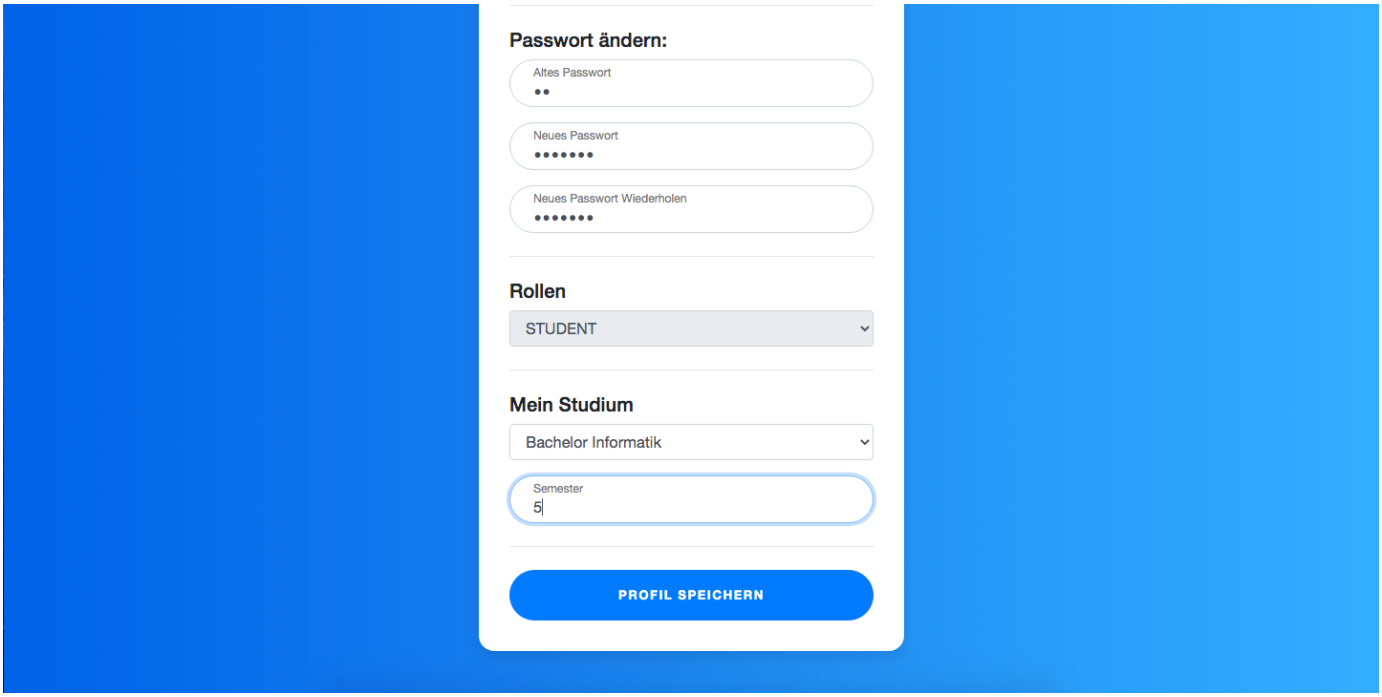



Die Navigationsleiste, wenn ein Nutzer, der nicht der Admin ist, angemeldet ist:

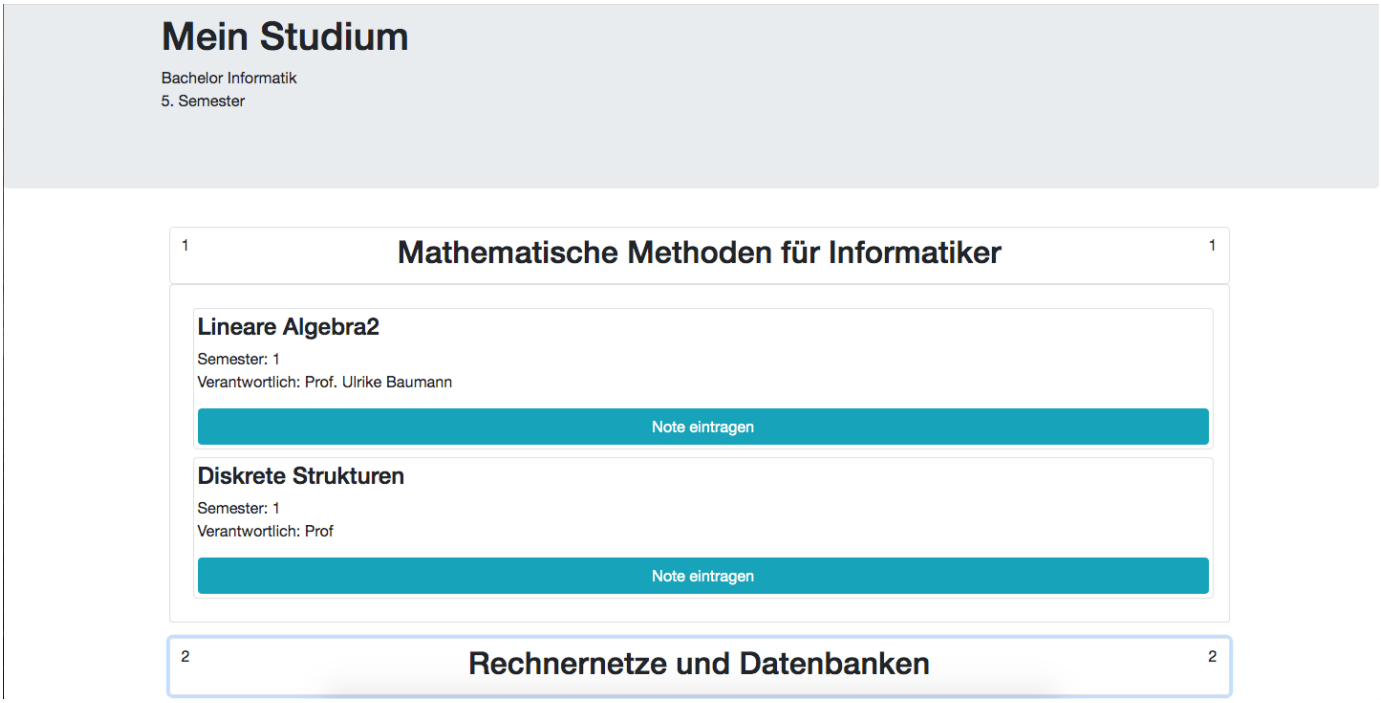



Der (normale) Nutzer, kann sein Profil, mit Ausnahme seiner Rollenzugehörigkeit bearbeiten:

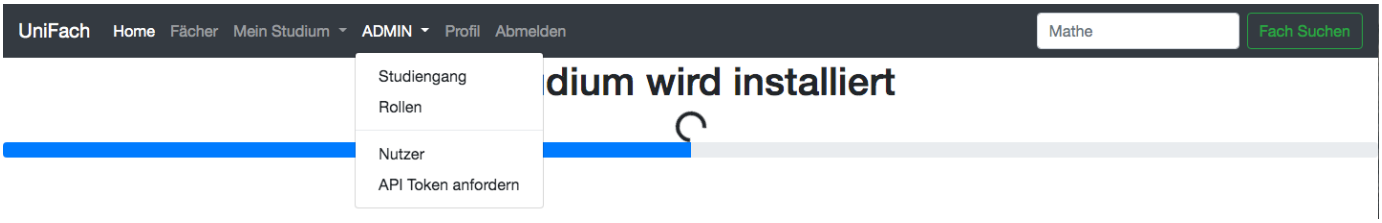




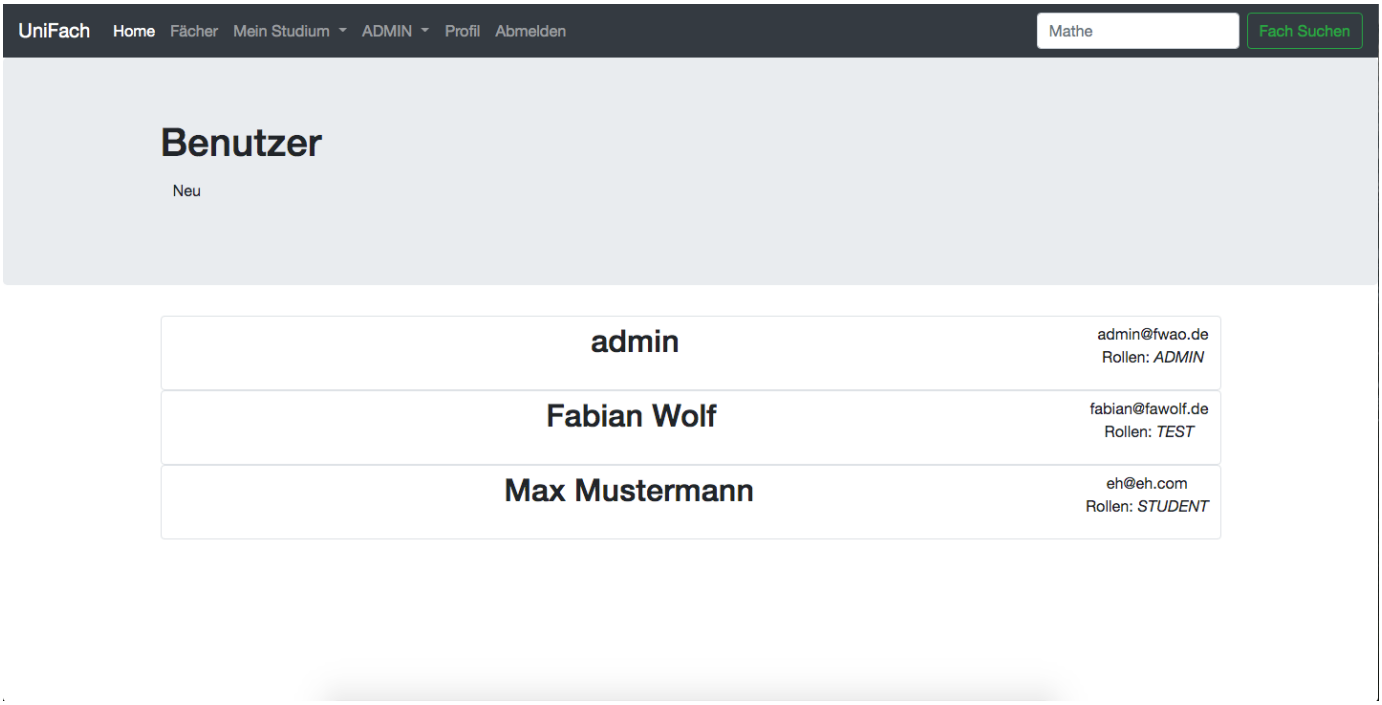

Die (detaillierte) Studienübersicht des angemeldeten Nutzers:



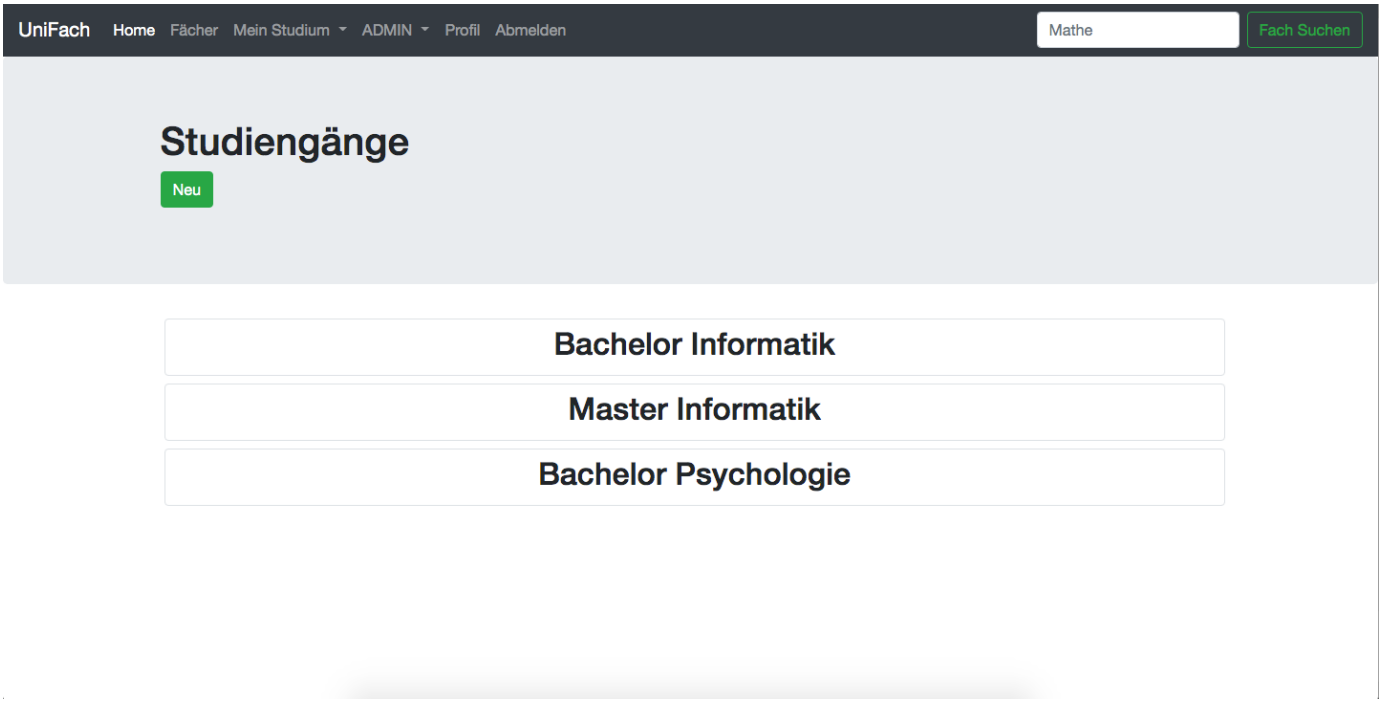

Die Navigationsleiste, wenn der Admin angemeldet ist:



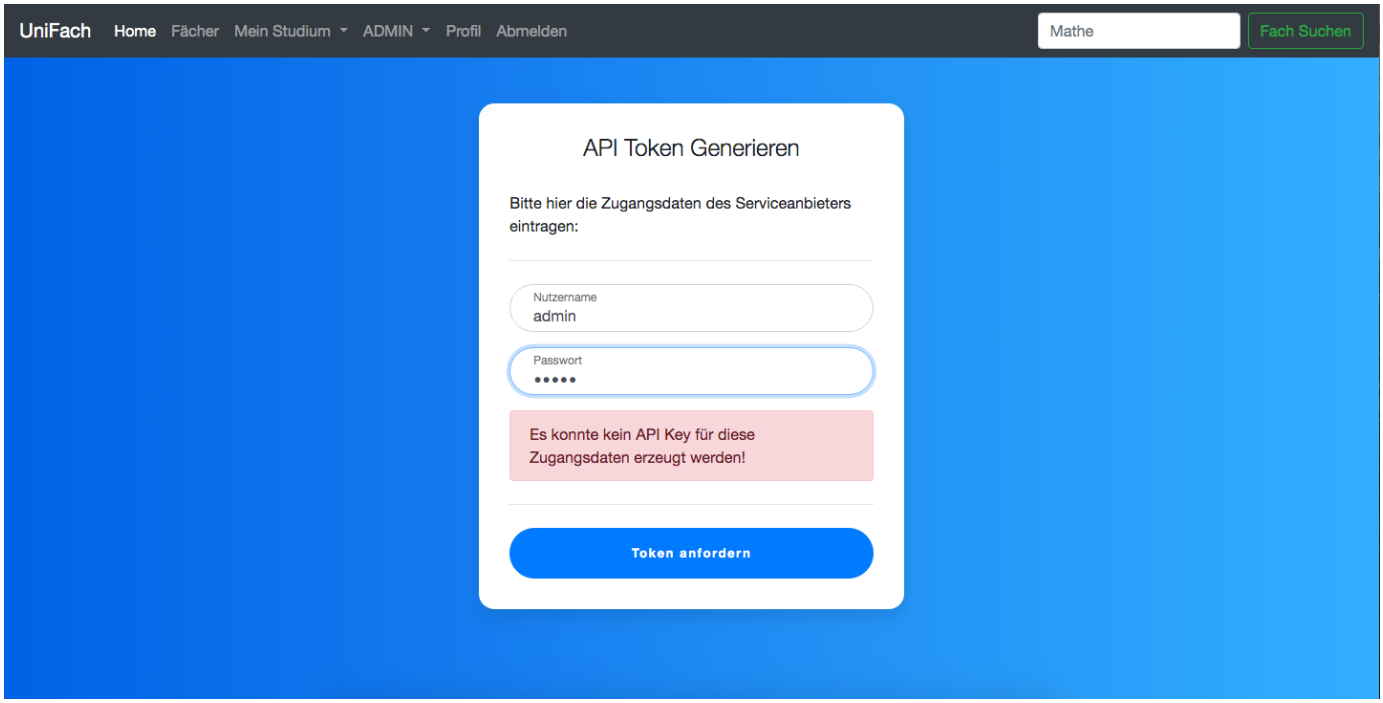

Der Admin, kann sich sämtliche Nutzer des Systems anzeigen lassen:




Der Admin kann sich die die Studiengänge in einer Übersicht anzeigen lassen:

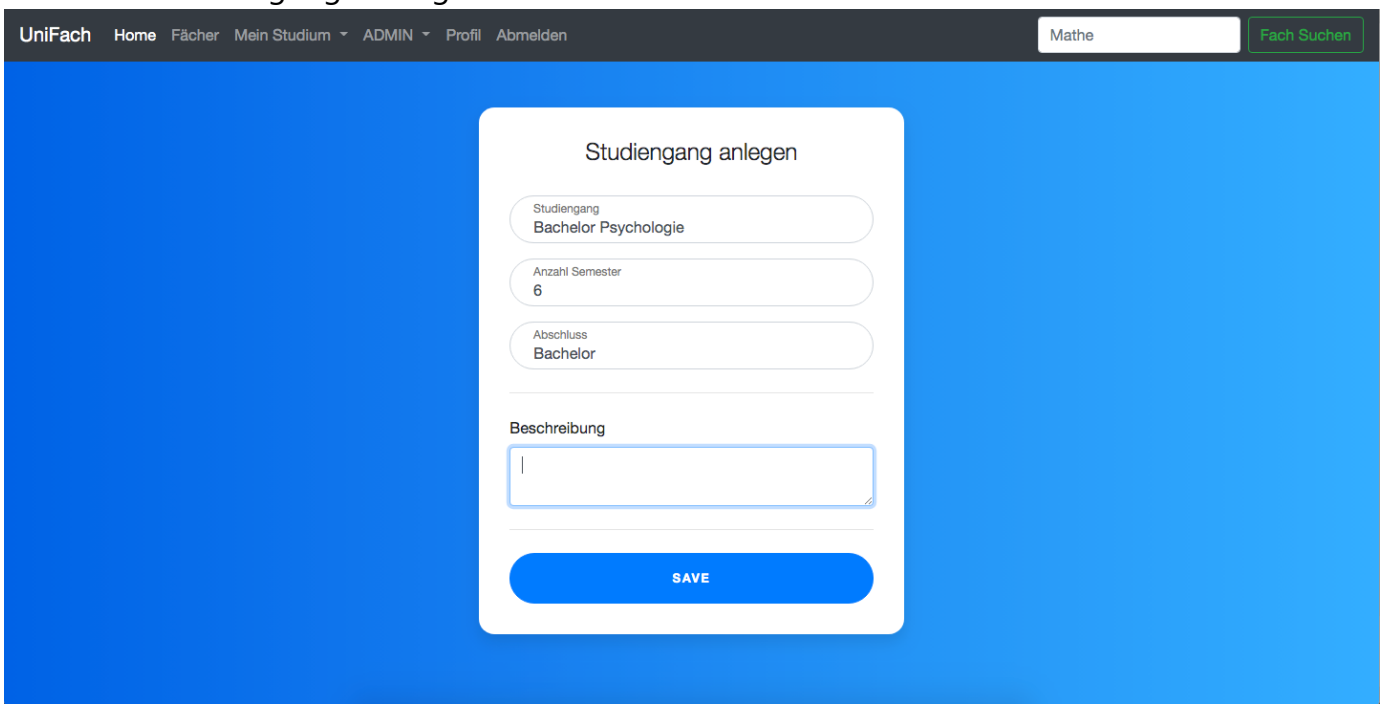



Um neue Studiengänge anlegen zu können, muss der Admin einen Studiengang-Token generieren (Hier ist dargestellt, was passiert, wenn die Anmeldedaten die falschen sind):

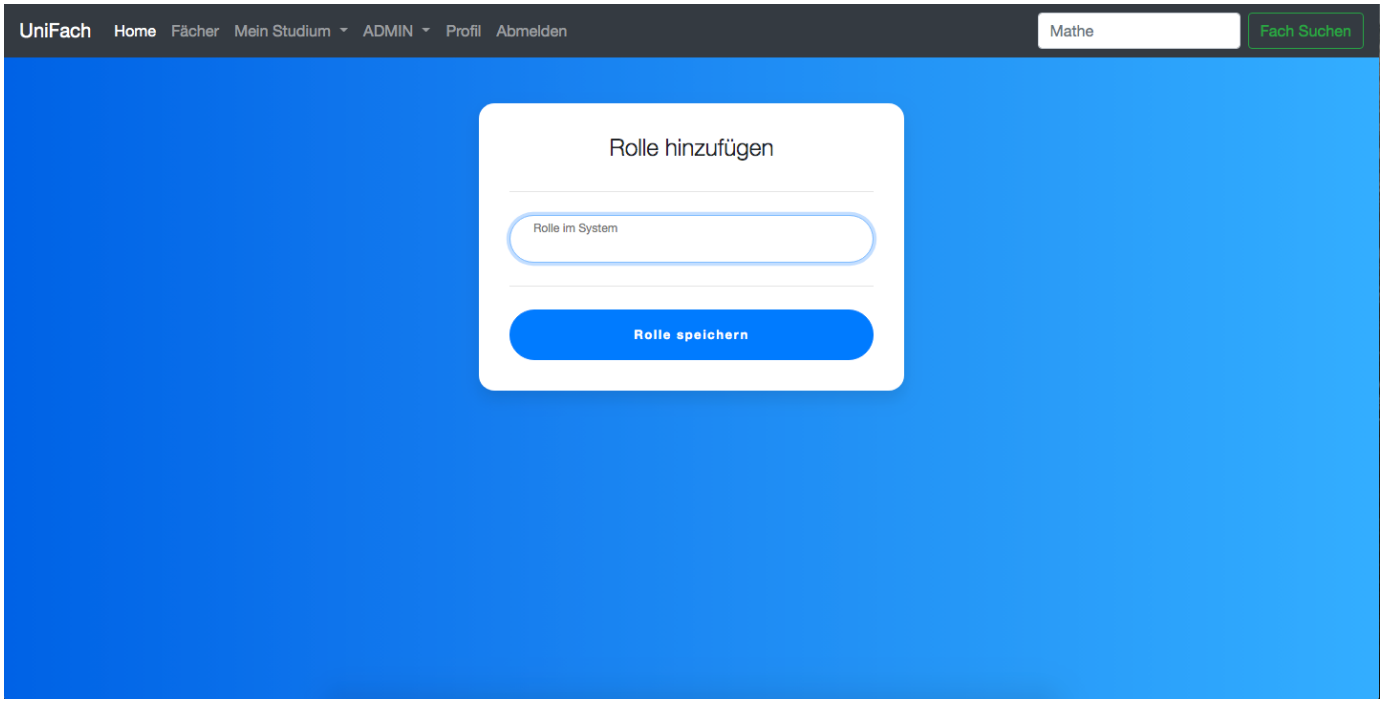



**Wenn ein Token für den Admin erstellt oder überprüft wurde, kann ein neuer Studiengang angelegt werden:**

Und neue Studiengänge anlegen:




**Auch neue (Nutzer)-Rollen können mit dem Token angelegt werden:**




## Umsetzung der Services

Die Dienste sind jeweils ähnlich strukturiert. Sie besitzen jeweils eine Klasse, die Anfragen an den Dienst handhabt und entsprechende Antworten generiert und eine, die die Relationen der jeweiligen Datenbank beschreibt.
Beide Services speichern ihre persistenten Daten mittels dem ToolKit SQLAlchely, welches jeweils mit einer SQLite Datenbank interagiert. Auf den Relationen der Datenbanken können wiederum verschiedene definierte Methoden angewendet werden, um die eingehenden Anfragen zu bearbeiten.

### Umsetzung Study Service

Der Study-Service speichert die folgenden Relationen persistent in einer Datenbank:

- ModulesLectures
  - Gibt an welche Lehrveranstaltungen zu welchen Modulen gehören
  - Bildet eine Many-to-Many Beziehung ab
- StudiesModules
  - Gibt an, welche Module zu welchen Studiengängen gehören
  - Bildet eine Many-to-Many Beziehung ab
- Study
  - Enthält jeden Studiengang mit seinen Attributen
  - Jeder Studiengang wird mittels des Primärschlüssels study_id identifiziert
  - steht in Relation mit Module
- Module
  - Enthält jedes Modul mit seinen Attributen
  - Jedes Modul wird mittels des Primärschlüssels module_id identifiziert

    - Steht in Relation mit Lecture
- Lecture
    - Enthält alle Lehrveranstaltungen mit ihren Attributen
    - Jede Lehrveranstaltung wird mittels des Primärschlüssels lecture_id identifiziert
- Token
    - Enthält alle Token, die dem Besitzer erlauben die Studiengänge entsprechend der Admin-Rechte zu verwalten

**Umsetzung User Service**

In der Initialisierung des Dienstes wird der Admin-Account angelegt (siehe Konfiguration). Der User Service speichert die folgenden Relationen persistent in einer Datenbank:

- Study
    - Enthält eine Kurzversion des Studienganges.
- Role
    - Enthält alle möglichen Rollen, die ein Nutzer annehmen kann
    - Jede Rolle wird mittels des Primärschlüssels id identifiziert
- UserRoles
    - Gibt an, welche User zu welchen Roles gehören
    - Bildet eine Many-to-Many Beziehung ab
- Token
    - Token, welches bei Login des Nutzers generiert wird und zur Authentifizierung der Requests auf diesen Service genutzt wird.
- SToken
    - Speichert Token eines berechtigten Nutzers, welches für bearbeitende Zugriffe auf den Study Service notwendig ist.
- UserToken
    - Gibt an, welche Token zu welchen Usern gehören
    - Bildet eine Many-to-Many Beziehung ab
- User
    - Enthält alle User mit ihren Attributen, die dem System bekannt sind
    - Steht in Beziehung zu Role
    - Besitzt jeweils einen Token und einen SToken

# Konfiguration

## Client

Der Webclient besitzt eine *config.py* *(http://config.py)*:

```
service_ip = "IP-Adresse-Studyservice"
userservice_ip = "IP-Adresse-Userservice"
service_port = "Port-Studyservice"
userservice_port = "Port-Userservice"

api_version = "api"
```

Hier können die notwendigen Informationen zur Erreichbarkeit der beiden Services eingetragen werden.

### Study Service

*config.py (http://config.py)*:

```
token_username = "admin"
token_password = "superadmin"
```

Derzeit (noch) statisches Passwort zur Ausstellung eines API Tokens für den Study Service. Für diesen Use Case wäre eine Lösung mit Einmalpasswörtern, oder alternativ eine eigene Administratorverwaltung sinnvoll.

### User Service

*config.py (http://config.py)*:

```
admin_username = "admin"
admin_email = "admin@fwao.de"
admin_password = "admin"
```

Bei der Erstellung der Datenbank wird ein Administratorkonto angelegt, dessen Zugangsdaten sich hier einstellen lassen.

## API Dokumentation

Befindet sich in separaten Dokumenten **output-study.pdf** und **output-user.pdf**.
Die Schnittstelle wurde über Swagger Hub beschrieben und dann in das PDF Format umgewandelt. Dabei sind einige Details verloren gegangen, deshalb sind im Repository außerdem die Originale zu finden.

## Sicherheitsmechanismen

### Authentification UserService

Zur Authentifizierung der Nutzer, die im Userservice verwaltet werden, kommt ein Tokenbasiertes System zum Einsatz. Hierzu werden zunächst beim Login die E-Mail Adresse, sowie das Passwort vom Client an den Userservice übertragen. Dieser generiert bei Übereinstimmung ein Token zur Autorisierung der nachfolgenden Anforderungen an den Userservice, welches an den Client gesendet und dort in einer durch Python Flask verwalteten Session gespeichert wird.

Zum Einsatz kommt hier jedoch nur eine primitive Implementierung von token-based authentication, welche nicht aktuellen Sicherheitsanforderungen entspricht. Dazu fehlt beispielsweise eine Angabe zur Gültigkeit, wie es bei JWT der Fall ist. Grundsätzlich müsste hier einfach die Implementierung des Token ausgetauscht werden.

### Authentification Study Service

Um Zugang zum Study Service zu erlangen, wird ein entsprechendes Passwort benötigt, welches außerhalb der definierten Schnittstellen vom Servicebetreiber des Study Services an einen berechtigten Nutzer des Client übergeben werden kann und auch im Userservice gespeichert wird.
Mithilfe von diesem kann beim Study Service ebenfalls ein einfaches Token angefordert und generiert werden, welches dann den Zugriff auf einige modifizierende API Endpunkte des Study Services erlaubt.

### HTTPS / TLS

Derzeit nur zur Kommunikation zwischen Client und Service kommt HTTPS/TLS zum Einsatz. Mehr dazu im Abschnitt Docker & Deployment.

## Docker & Deployment

Der Client und beide Microservices wurden als Docker Image unter Nutzung von Dockerhub auf die Clound Instanz übertragen. Das Bauen der Images funktioniert bei allen drei Programmen aufgrund der sehr ähnlichen Anforderungen gleich.
Als Python Umgebung und zur Installation der benötigten Python Packages kommt pipenv zum Einsatz, welches eine einfache Installation des Pyhton Web Servers gunicorn ermöglicht. Letzterer kommt zum Einsatz, da der Client selbst (und wenn nötig auch die Services) nach außen TLS verwendet. Gunicorn kann mit Verweis auf generierte TLS Zertifikate genutzt werden, um mittels https mit den Endgeräten kommunizieren zu können.

### TLS Zertifikat

Auf der Cloud Instanz wurde mithilfe von certbot ein Lets Encrypt Zertifikat generiert, welches derzeit beim Webclient zum Einsatz kommt.
Das funktioniert, da bei einer vorhandenen Domain ein DNS A Record einer Subdomain auf die IP Adresse der Cloud Instanz estellt wurde. Damit entfallen hier die Sicherheitswarnungen, die bei selbst zertifizierten Zertifikaten angezeigt werden.

## Docker auf Cloud Instanz:

**- pull Images:**

Die Images wurden auf Dockerhub hochgeladen: fwao/webclient, fwao/service, fwao/userservice

**- Docker Netzwerk Bridge anlegen**

```
docker network create --driver bridge name-des-netzwerkes
```

**- Webclient Container:**

```
docker run -d --network name-des-netzwerkes --name webclient -p 443:443 --mount type=bind,source=/home/debian/certs,target=/certs fwao/webclient
```

Dabei werden die Lets Encrypt Zertifikate als externer Speicher in den Container mit eingebunden und können von gunicorn verwendet werden. Die Zertifikate selbst gehören jedoch niemals in ein Docker Image, welches darüber hinaus noch auf öffentlich zugänglichen Plattformen zur Verfügung steht. Diese Anforderung wird mit dem mount type=bind erfüllt.

**- Container für beide Services erstellen:**

Das ist einfacher, da für die Demonstration die APIs der Services nicht aus dem Internet direkt erreichbar sind, sondern nur der Client an sich.

```
docker run -d --network name-des-netzwerkes --name userservice fwao/userservice
```