Dokumentation

Sprechstunden Internet Verwaltung

cdeinlein, tdubovik, spahl, ndiadowitz

Inhaltsverzeichnis

1 Vorwort

In dieser Studienarbeit geht es um die Erstellung einer Website zur Sprechstundenverwaltung an der Hochschule Hof. Als Projektname wurde `Schiv` gewählt das eine Abkürzung von (S)pre(ch)stunden (I)nternet V(erwaltung) ist. Für die Implementierung dürfen folgende Hilfsmittel verwendet werden:

- HTML 5
- CSS 3
- Javascript (Angular JS, Bootstrap, Ajax, jQuery)
- PHP (Laravel)
- MySQL

2 Funktionale Anforderungen

Eine Liste von Funktionalen Anforderungen die während der Gespräche mit dem Auftraggeber beschlossen wurden:

1. Konten sind nur registrierbar mit E-Mail-Adressen (`@hof-university.de`) der Hochschule Hof.
2. Zur Registrierung wird nur die E-Mail-Adresse und ein Passwort benötigt.
3. Konten werden nach der Registrierung als „Studenten“ behandelt.
4. Eine Liste der aktiven Dozenten wird über die Schnittstelle der iOS-Stundenplan-App[1] abgefragt. Hierbei muss aus den Namen des Dozenten die E-Mail-Adresse abgeleitet werden, da die diese nicht direkt abfragbar ist. Für jeden Dozenten wird ein deaktiviertes Konto angelegt.
5. Konten können nur „Dozent“ werden, wenn bereits ein deaktiviertes Konto diesen Typs angelegt ist.
6. Jedem Konto sind ein oder mehrere Fakultäten zugeordnet.
7. Aktivierung eines Kontos erfolgt über eine E-Mail die nach der Registrierung mit einem Aktivierungs-Link geschickt wird.
8. Die Anwendung soll mindestens auf normalen Desktop-PCs mit den weitverbreitetsten Browsern laufen.
9. Es werden generell keine Datensätze gelöscht, sondern nur weich gelöscht („unsichtbar gemacht“).
10. Am Ende eines Semesters werden Studenten aus dem System gelöscht und müssen sich dann wieder erneut registrieren
11. Aktivierungslink wird über einen GET abgesetzt, d.h. der Aktivierungslink muss im Frontend implementiert sein.

[1] https://github.com/HochschuleHofStundenplanapp/iOS-App/wiki/Schnittstellen-zum-Server

3 Style-Guide

3.1 Farbschema

(TODO)

Es wird sich am Farbschema der Hochschule Hof orientiert.

3.2 Typographie

(TODO)

Als Schriftart wird `Roboto`[1] verwendet, da diese klar lesbar ist.

3.3 Layout

(TODO)

Beim Layout muss zwischen Student und Dozent unterschieden werden. Das Layout der Seite soll möglichst einfach und übersichtlich sein.

3.4 Navigation

Es wird nur eine Navigationleiste am oberen Rand geben, die immer sichtbar ist. Alle Funktionen sind darüber mit wenigen Klicks erreichbar.

[1] https://fonts.google.com/specimen/Roboto?selection.family=Roboto

4 Datenbankmodel

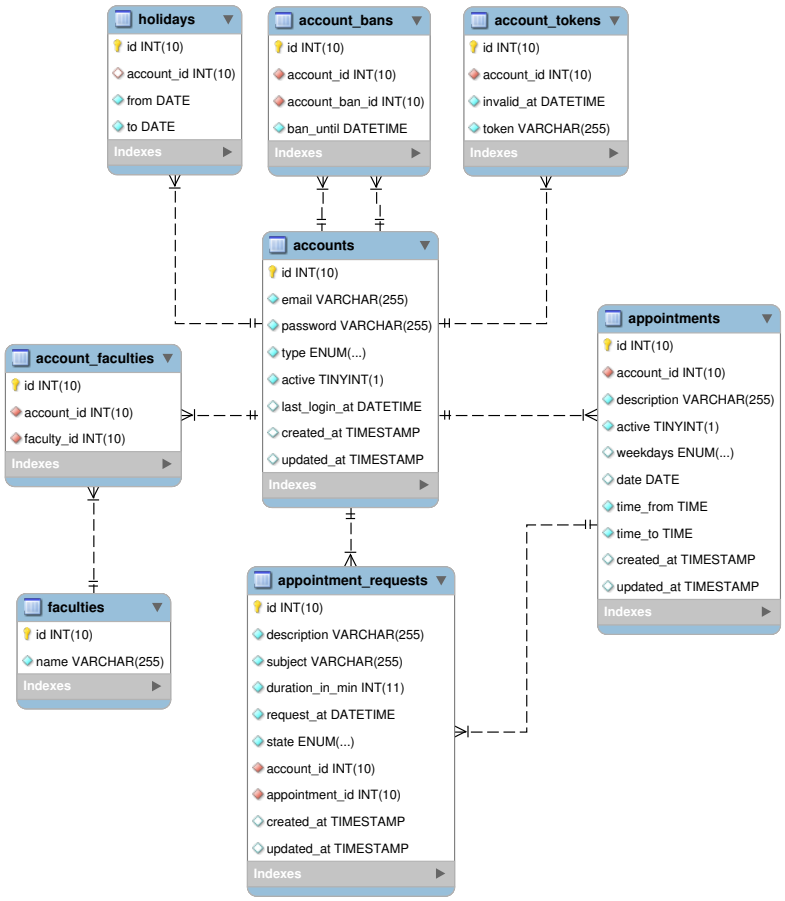

Abbildung 4.1: Datenbankmodel von schiv (Erstellt mit mysql-workbench)

5 REST-Schnittstelle

5.1 Überblick

Tabelle 5.1: Zeigt den Aufbau der REST-Schnittstelle

Beschreibung	[A|D] Method:Url (Body)
Login	put:login (email, password)
Logout	put:logout (email, password)
Reset Passwort	put:reset (email)
Registrieren	post:register (email, password)
	put:register (token)
Suche	get:docents
Informationen bzw. Termine von Dozenten	get:docents/{docent_id}
Terminanfragen abrufen	**D** get:appointment_requests
Termin Einschreibung	**A** post:appointment_requests (...)
Termine annehmen bzw. ablehen	**D** put:appointment_requests (id, state)
	A delete:appointment_requests/{id}
Termine Dozenten	**D** get:appointments
	D get:appointments/{count}
	D get:appointments/{from}/{to}
	D post:appointments (...)
	D delete:appointments/{appointment_id}
Einstellungen	**A** get:settings
	A put:settings (email, password)
Ausgeschlossene Konten	**D** get:account_bans
	D post:account_bans (account_ban_id)
	D delete:account_bans/{id}

Anmerkungen:

- `(...)` bedeutet das alle Felder, wie im Model definiert, verwendet werden können.
- `get:appointments/{from}/{to}`: `from` und `to` müssen im Datumsformat `YYYY-MM-DD HH:II:SS` angegeben werden.
- **A**: Zugriff nur wenn man als Student oder Dozent eingeloggt.
- **D**: Zugriff nur wenn man als Dozent eingeloggt.

5.2 Details

5.2.1 Allgemein

Die Nachrichten sind im JSON-Format. Bei Erfolg kommt der HTTP-Status **200** zurück, im Fehlerfall kann **401**, **404**, **422** oder **500** zurückgegeben werden:

- **200**: Es kommen die angeforderten Daten zurück oder eine leere Nachricht falls die kein Rückgabewert nötig ist.
- **401**: Falls der Login fehlschlägt, der Benutzer nicht eingeloggt ist oder der Benutzer ein Dozent sein um auf die Route zugreifen zu können. Als Antwort kommt eine Fehlermeldung mit dem Grund zurück. Beispiel:

```
{"message":"login unsuccessful"}
{"message":"account isn't of type 'Docent'"}
{"message":"login required"}
```

- **404**: Die angefragten Objekte sind nicht in der Datenbank vorhanden. Als Antwort kommt eine leere Nachricht zurück. Beispiel:

```
[]
```

- **422**: Die Anfrage enthält nicht alle nötigen Felder, oder ein Feld ist falsch formatiert. Die Antwort enthält als Attribute die Feldnamen bei denen ein Fehler festgestellt wurde. Als Wert der Attribute wird ein Array mit Fehlermeldungen zurückgegeben. Beispiel:

  Anfrage an `post:register`:

```
{"email": "alice","password": "bob"}
```

  Antwort von `post:register`:

```
{"email":["validation.required"],"password":["validation.min.string"]}
```

- **500**: Ein allgemeiner Fehler (Syntaxfehler oder nicht behandelter Fehler). Als Antwort wird eine Fehlermeldung im Attribut `message` zurückgegeben. *Hinweis:* Ist die Anwendung im Debugmodus (`app.debug == true`), dann wird die Meldung aus der auslösenden Exception zurückgegeben. Es kann im Log unter `storage/logs/laravel.log` die genaue Fehlermeldung nachgeschaut werden. Beispiel:

```
{"message": "fatal error"}
```

5.2.2 Register, Login, Logout und Reset Password

Der Login funktioniert über eine Session, die serverseitig gespeichert wird. Auf Clientseite muss in einem Cookie die geschickte Session-Id mitgeführt werden. Da die Anwendung sowieso in einem Browser läuft, kümmert sich der Browser um die Cookie Verwaltung. Auf dem Server

kümmert sich Laravel um die Session, die für jede Verbindung eindeutig ist und sein muss. Der Webserver muss später mit über das HTTPS-Protkoll angesprochen werden, sonst könnte ein Angreifer die Session „hijacken“.

Register: Beim Registrieren wird ein `post:register` geschickt mit E-Mail und Passwort des Benutzers. Wenn dies erfolgreich ist, dann wird eine E-Mail an den Benutzer mit einem Aktivierungslink geschickt (Momentan: wird einfach zum Testen der Token mit in der Antwort zurückgeben). Beispiel:

```
{"email":"alice@wonder.land","password":"rabbithole"}
```

Momentane Antwort:

```
{"token":"twon-ha"}
```

Login: Wenn der Login nicht erfolgreich ist wird ein **401** zurückgeben, sonst ein **200** mit leerer Nachricht. Beispiel: siehe **Register**.

Logout: Löscht die Session auf dem Server und loggt den Benutzer aus. Falls ein nicht angemeldeter Benutzer versucht sich auszuloggen wird ein **401** zurückgeliefert.

Reset Password: Benötigt nur die E-Mail-Adresse und schickt eine E-Mail an den Benutzer mit einem neu generierten Passwort und einem Aktivierungslink. (Momentan: wird einfach zum Testen der Token und das neue Passwort mit in der Antwort `{"token":"<token>","password":"<password>"}` zurückgeben). Nach einem Klick auf den Aktivierungslink ist das Konto des Benutzer wieder aktiviert. Einloggen kann sich der Benutzer dann mit dem generierten Passwort. Das Passwort ist permant und dem Benutzer ist es freigestellt es wieder zu ändern. Beispiel:

```
{"email":"alice@wonder.land"}
```

Momentane Antwort:

```
{"token":"twon-ha","password":"rabbithole2"}
```

Laut Vorgabe (siehe Kapitel 2) muss der Aktivierungslink im Frontend implementiert sein. Da der Link über ein `GET` abgesetzt wird, aber das Backend ein `PUT` benötigt. Das Frontend muss dann ein `put:register` mit dem Token in der Nachricht absetzen, um das Konto wieder zu aktivieren.

5.2.3 Suche

(TODO)

5.2.4 Termine

(TODO)

5.2.5 Einstellungen

(TODO)

5.2.6 Konten Sperren

(TODO)

6 Implementierung

6.1 Allgemein

Die REST-Schnittstelle kann manuell über `test.html` ausprobiert bzw. getestet werden. Implementierungsdetails für die REST-Schnittstelle werden bereits in Abschnitt 5.2 beschrieben.

7 Testfälle

7.1 Backend

Es sind für jeden Controller Testfälle im Verzeichnis `tests/Feature` angelegt.

7.2 Fontend

(TODO)

8 Benutzerhandbuch

8.1 Installation für Entwicklung und zum Ausprobieren (unter Ubuntu 17.04)

```
$ sudo apt update
$ sudo apt install git composer unzip php php-mbstring php-xml
$ sudo apt install php-sqlite3 sqlitebrowser
$ cd $HOME
$ git clone https://github.com/studentcstn/schiv
$ cd schiv
$ composer install
$ cp .env.sqlite .env
$ touch /tmp/db
$ php artisan migrate:refresh --seed
$ php artisan serve
$ php artisan retrieve:docents <username> <password>
$ firefox http://localhost:8000
```

8.2 Dozenten aktualisieren

Am Ende des Semester können die Dozenten-Konten aktualisiert werden. Dazu dient der Befehl `php artisan retrieve:docents`. Es werden alle vorhanden Dozenten-Kontos deaktiviert und das Passwort zurückgesetzt. Nur Dozenten die noch Zugriff auf ihr E-Mail-Konto haben können sich erneut registrieren und anmelden. Die Zugangsdaten für die Schnittstelle der iOS-Stundenplan-App müssen in der `.env`-Datei eingetragen werden (Schlüssel: `IOSAPP_USERNAME` und `IOSAPP_PASSWORD`).

Zur Vereinfachung kann man das Task-Scheduling aktivieren, wie in dieser Anleitung beschrieben https://laravel.com/docs/5.4/scheduling. Der Task würde am 15. März bzw. 31. September jedes Jahr ausgeführt.

Zum Testen kann noch die Option `--from-cache` verwendet werden. Hier werden die Daten nur einmal vom Server geholt und dann in einer lokalen Datei `/tmp/docents` zwischengespeichert.