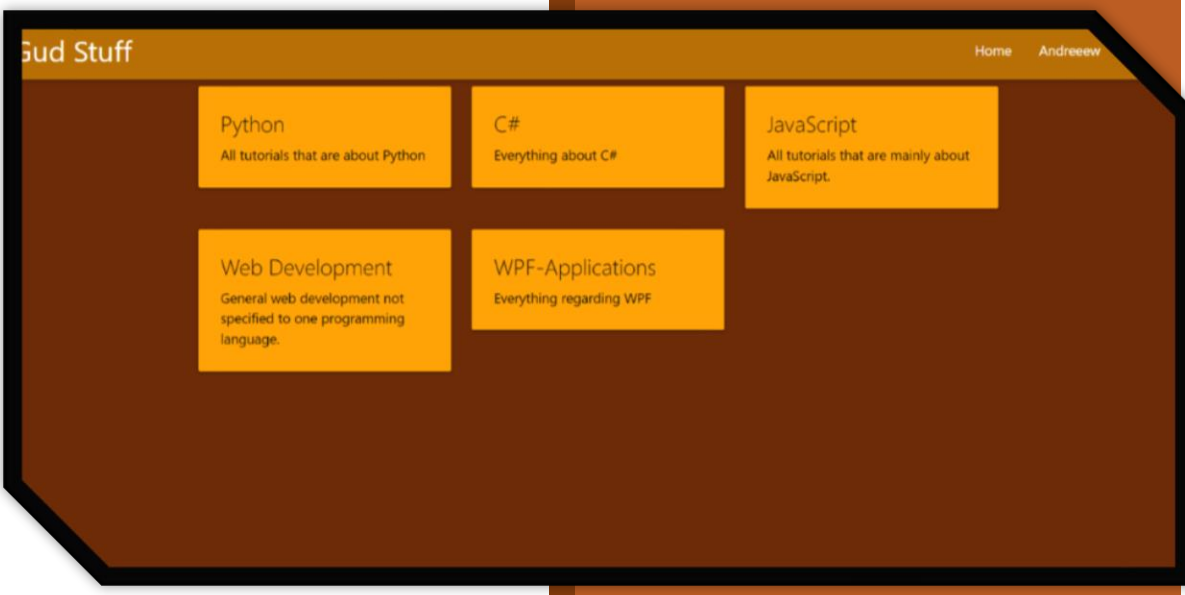
Gud Stuff

Dokumentation



André Möll, Colin Wüst

M151

Gud Stuff

Inhaltsverzeichnis

1. Einleitung 2
2. Projektbeschrieb 2
3. Zieldefinierung 2
4. Meilensteine 2
5. Arbeitsjournal 3
6. Angriffsarten 4
   6.1 Cross-Site-Scripting 4
   6.2 SQL-Injection 4
   6.3 Cross-Site-Request-Forgery 4
   6.4 Session Forging / Hijacking 5
   6.5 E-Mail-Header-Injection 5
   6.6 Directory-Traversal 5
   6.7 Exposed-Error-Messages 5
   6.8 Man-In-The-Middle 6
   6.9 Shell-Injection 6
   6.10 Formulare 6
   6.11 Denial of Service 6
7. Wahl des Frameworks 7
8. Datenbank 7
9. Sessions / Transaction 8
10. Seite 8
11. Rückblick 10
   11.1 Commits 10
12. Fazit 10

1. Einleitung

In diesem Modul geht es um die Anbindung von Datenbanken an Webanwendungen. Hier, in diesem spezifischen Projekt, soll der Fokus vor allem auf dem Schutz der Website / DB vor verschiedensten Arten von Angriffen stehen. Dafür mussten wir in Gruppen eine Webanwendung erstellen, welche vor allem gegen Angriffsarten, welcher wir vorher zusammen im Unterricht behandelt hatten, geschützt sind. Diese reichen von SQL-Injections bis hin zu Cross-Site-Scripting. Nachfolgend wird genauer darauf eingegangen.

2. Projektbeschrieb

Das Startdatum beziehungsweise die Informationen zum Projekt haben wir am 02.05 bekommen. In Gruppen von 1-3 Personen sollen wir eine Webanwendung realisieren, die einen Datenbank-anschluss besitzt und gegen Angriffe geschützt ist. Es geht dabei primär um den Schutz vor den verschiedenen Angriffsvarianten, wobei der tatsächliche Inhalt der Webseite zweitrangig ist. Unsere Gruppe, bestehend aus Colin Wüst und André Möll, hat sich für eine Tutorialseite entschieden. Auf ihr soll der Enduser zwischen verschiedenen Kategorien navigieren können und sich dann ein entsprechendes Tutorial anschauen können. Als Framework haben wir dabei Django verwendet. Das Abgabedatum ist der 20.06.2022.

3. Zieldefinierung

Unser Ziel besteht darin, eine Tutorialseite zu realisieren. Dazu lassen sich folgende Ziele definieren:

- Tutorial kann erstellt und angezeigt werden
- Benutzer kann erstellt und angezeigt werden
- Homeseite mit Kategorien & Unterseiten mit Tutorials
- Fertigstellung vor Abgabetermin

4. Meilensteine

Unsere Meilensteine sind folgende:

- Planung abgeschlossen
- Frontend steht
- Datenbankverbindung besteht
- Benutzer- & Tutorialverwaltung funktioniert
- Dokumentation fertig => Projektabschluss

5. Arbeitsjournal

1. Woche (02.05, Beide):

In der ersten Woche wurde uns das Projekt vorgestellt und wir haben uns Gedanken gemacht, was wir für eine Webanwendung machen und wie wir die verschiedenen Schutzarten einbauen.

2. Woche (09.05, Beide):

Praxisprüfung & Wir haben uns für eine Tutorialseite entschieden und mit der Planung begonnen sowie erste UI-Entscheidungen gefällt. Ausserdem wurde die Wahl von Django als Framwork getätigt.

3. Woche (16.05, Colin):

Selbständige Weiterarbeit am Projekt, zudem ist André ab heute in Berlin. Colin hat sich heute genauer mit den Möglichkeiten von Django, verschiedene Angriffsarten abzuwehren, auseinandergesetzt.

4. Woche (23.05, Colin):

Die Datenbank-Models wurden mit Django erstellt und erste Tutorial-Daten getestet. Es besteht mittlerweile ein provisorisches UI einer Homepage, die noch lokal ist.

5. Woche (30.05, Colin):

Heute wurde das Register- sowie das Login-Formular angefangen. Das Login-Formular funktioniert soweit, bei der Verlinkung des Registrierungsformulars scheint jedoch noch etwas nicht ganz zu stimmen.

6. Woche (06.06, Beide):

Feiertag & Wir haben heute Modelle für die Kategorien und die Tutorials selber erstellt, sowie einen Code-Editor eingebunden.

7. Woche (13.06, Beide):

André ist nun zurück, nach einer Lagebesprechung wurde die Dokumentation fast fertiggestellt und das Register-Formular wurde ebenfalls repariert. Das UI ist fertig und es fehlen nur noch wenige Feinschliffe.

8. Woche (20.06, Beide)

Projektabgabe

6. Angriffsarten

Für Viele der unten genannten Arten ist das Framework Django mitverantwortlich. An dieser Stelle verweisen wir für genauere Informationen gerne auch auf die Django-Dokumentation.

6.1 Cross-Site-Scripting

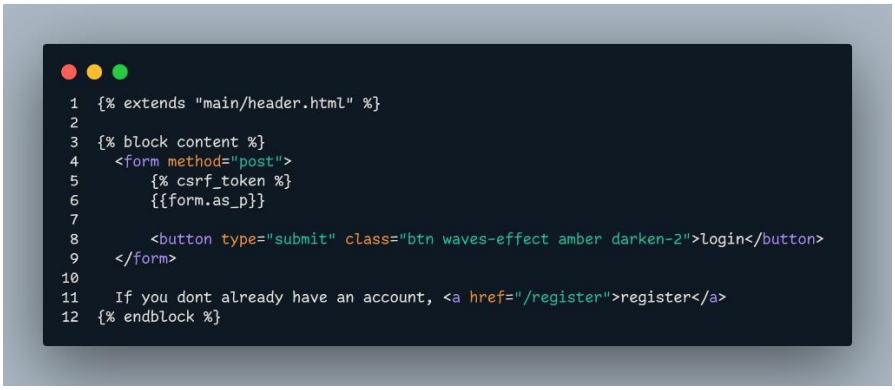
Wir nutzen für alle Seiten und Unterseiten auch die dynamisch angezeigten Templates im Hintergrund. Im Template wird genau festgelegt, was angezeigt wird und man kann nicht zum Beispiel im Link noch HTML-Tags oder Ähnliches einfügen. Zusätzlich werden die Links zu allen Unterseiten durch 'Slugs' generiert. Ein 'Slug' ist ein einzigartiger Identifikator, welcher Teil eines Links darstellt. Er dient zur Identifizierung einer Ressource und ist lesbarer für Menschen als beispielsweise ein reguläre ID. Wenn man also beispielsweise http://website/tutorialXYZ eingibt, zeigt dieser Link auf das TutorialXYZ, in dem angezeigt wird, was wir im Template festgelegt haben. Wenn man dort noch HTML-Tags einfügt, wird der Slug ungültig und man wird stattdessen auf eine Error-Page weitergeleitet. So sind alle Seiten gehandelt und sicher.

6.2 SQL-Injection

Es gibt keine Stelle auf der Weboberfläche, wo eine Usereingabe direkt für ein SQL-Statement verwendet wird. Auf unserer Website bestehen nur Modell-Klassen im Backend, die eine Tabelle im SQL abbilden, welche der Enduser abfragt. Die Umwandlung beziehungsweise Übersetzung von Modell zu SQL wird direkt von Django, unserem verwendeten Framework, gemacht. Django benutzt dabei Querry Parameterization, wobei die Parameter, welche vom Nutzer kommen und damit ein potenzielles Sicherheitsrisiko sind, separat vom SQL-Querry definiert sind. Die Parameter werden dabei vom Datenbanktreiber escaped. Die models.py-Klasse (Siehe Foto unten) ist die einzige Klasse, in der wir verschiedene Felder haben, die aber auch automatisch von Django in Tabellen umgewandelt werden. Und neue Objekte dieser Klasse werden dann im SQL gespeichert, aber nie über ein SQL-Statement von uns beziehungsweise dem Enduser sondern immer über Django, was sicherer ist. Somit ist die Seite sehr sicher vor SQL Injections.

6.3 Cross-Site-Request-Forgery

Für die Sicherung gegen Cross-Site-Request-Forgery ist ebenfalls Django zuständig: In das Registrierungs-Formular und das Login-Formular haben wir noch ein {% csrf_token %} eingefügt. Das ist eine Funktion von Django, die CSRF komplett verhindert. Das Token überprüft dabei, ob die Request von der eigenen Seite kommt oder nicht. Wenn nicht, wird der Angreifer blockiert und eine Warnung angezeigt. Dies macht Cross-Site-Request-Forgery praktisch unmöglich.



```
{% extends "main/header.html" %}

{% block content %}
  <form method="post">
      {% csrf_token %}
      {{form.as_p}}

      <button type="submit" class="btn waves-effect amber darken-2">login</button>
  </form>

  If you dont already have an account, <a href="/register">register</a>
{% endblock %}
```

Cross-Site-Request-Forgery-Token in Action

6.4 Session Forging / Hijacking

Sessions sind nie in der URL enthalten, was schon ein wichtiger Punkt gegen einen Session-Forging-Angriff ist. Die Session selbst ist nicht direkt in einem Cookie gespeichert, sondern nur in einer ID. Dieselbe ID der Session ist mit Session Data, die im Backend gespeichert ist, verknüpft. Im Cookie selbst befindet sich daher nur diese Session ID, während alles andere in der Datenbank vom Backend gespeichert wird. Man kann sich nie vollständig davor schützen, aber so ist man schon sehr sicher dagegen.

6.5 E-Mail-Header-Injection

Wir haben in unserer Website keine E-Mail-Formulare verwendet, jedoch würde Django auch hier davor schützen: Die eingebauten Mail-Funktionen (in django.core.mail) erlauben einfach keine Zeilenumbrüche in jeglichen Feldern, welche gebraucht werden, um die Headers (Absender, Empfänger und Betreff) zu konstruieren. Sollte ein Angreifer trotzdem versuchen, mit der django.core.mail.send_mail-Funktion und einem Betreff, welcher Zeilenumbrüche oder Ähnliche bösartige Inhalte beinhaltet, eine Mail zu senden, wird er stattdessen eine BadHeaderError-Exception erhalten.

6.6 Directory-Traversal

Da wir in unserer Webanwendung weder Daten aus Files laden, noch welche in Files schreiben, ist das Directory Traversal entsprechend in dieser Form überhaupt nicht möglich.

6.7 Exposed-Error-Messages

Der Debug-Mode ist deaktiviert. Dadurch werden Error-Messages gar nicht in der Konsole angezeigt und man kann sich keinen bösartigen Vorteil verschaffen, in dem man diese interpretiert.

6.8 Man-In-The-Middle

Für eine Man-In-The-Middle Attacke braucht es eine Verbindung beziehungsweise die Seite muss gehostet werden. Wenn wir die Webseite also tatsächlich veröffentlichen würden (also nicht nur in einer Entwicklungsumgebung), kann man sich vor allem mit SSL davor schützen beziehungsweise auch HTTP**S**. Bei unserem momentanen Hoster ist jedoch der Port 80 von etwas Anderem besetzt, weshalb es unmöglich ist, SSL dort laufen zu lassen. Bei einem Release würden wir dies dann mit einem Gratis- beziehungsweise bezahltem Hoster machen.

6.9 Shell-Injection

Bei einer Shell Injection wird etwas Schädliches in einer Shell ausgeführt. Grundsätzlich wird bei uns Nichts in der Shell gemacht, beziehungsweise kann generell nur der Adminuser Dinge erstellen, wodurch es höchstens für Phishing oder Ähnliches anfällig wird.

6.10 Formulare

Bei den Formularen wird beispielsweise der Username einfach als Text ausgegeben, nie als ausführbarer Code oder Ähnliches. Auch HTML-Tags (im Namen) werden nur als Plain-Text angezeigt. Zudem existieren Vorgaben für Namenslänge, Passwortkomplexität und E-Mail. Das Passwort wird ausserdem nur in einem Hash gespeichert.

```python
def AccountRequest(request):
    # check if user is logged in
    if request.user.is_authenticated:
        if request.method == 'POST':
            form = AccountForm(request.POST, instance=request.user)
            if form.is_valid():
                form.save()
                messages.success(request, "Account updated successfully")
                return redirect('main:homepage')

        form = AccountForm()
        return render(request, 'main/account.html', {'form': form})
    else:
        messages.error(request, "You are not logged in")
        return redirect("main:login")
```

```python
class UserForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

    def save(self, commit=True):
        user = super(UserForm, self).save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
        return user
```

6.11 Denial of Service

Der Windows-Server, auf dem die ganze Website gehostet ist, ist bereits seitens Windows zu einem gewissen Level von DDOS-Angriffen geschützt. Wenn wir die Webanwendung tatsächlich publizieren würden, würden wir ausserdem noch Cloudflare einsetzen, um einen zusätzlichen Schutz davor zu schaffen.

7. Wahl des Frameworks

Wir haben uns hier entschieden, ein Framework einzusetzen. Dabei fiel die Wahl auf Django, ein Python-Framework, welches Open Source ist. Da Colin in seiner Firma praktisch nur mit Python arbeitet und er dadurch bereits ein wenig Erfahrung mit Django hat, fiel die Wahl leicht. Ein wichtiger Grund war ausserdem, dass André während einem grossen Teil des Projektes abwesend und mit einem Berufspraktikum im Ausland beschäftigt war, weshalb hier eher auf die Präferenzen von Colin eingegangen wurde. Dazu findet sich im Rückblick noch eine genauere Ausführung. Zudem regelt Django schon einige Angriffsarten selbst oder vereinfacht die Angehensweise zumindest etwas.

8. Datenbank

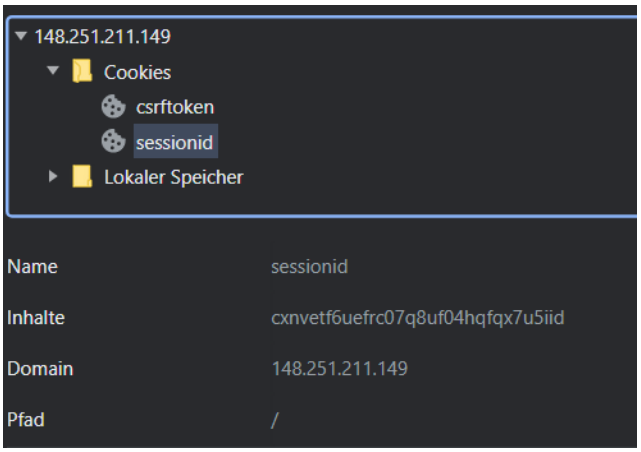
Die Datenbank ist bereits in Django eingebaut und Django kümmert sich damit auch um die Verbindung zur Datenbank. Wir haben es ausserdem so konfiguriert, dass grundsätzlich nie eigene SQL-Statements gemacht werden, sondern stattdessen alles über die models.py-Klasse direkt mittels Django in die Datenbank synchronisiert wird. Jedes Mal, wenn zum Beispiel ein Tutorial erstellt wird, wird dieses automatisch auch in der Datenbank gespeichert.

```python
class TutorialCategory(models.Model):
    tutorialCategory = models.CharField(max_length=200)
    categorySummary = models.CharField(max_length=400)
    categorySlug = models.CharField(max_length=200, default=1)

    class Meta:
        verbose_name_plural = "Categories"

    def __str__(self):
        return self.tutorialCategory


class TutorialSeries(models.Model):
    tutorialSeries = models.CharField(max_length=200)
    tutorialCategory = models.ForeignKey(TutorialCategory, null=True, default=None, verbose_name="Category", on_delete=models.SET_DEFAULT)
    tutorialSeriesSummary = models.CharField(max_length=400)

    class Meta:
        verbose_name_plural = "Series"

    def __str__(self):
        return self.tutorialSeries



class Tutorial(models.Model):
    tutorialTitle = models.CharField(max_length=200)
    tutorialContent = models.TextField()
    tutorialPublished = models.DateTimeField("date published", default=datetime.now())
    tutorialSeries = models.ForeignKey(TutorialSeries, null=True, default=1, verbose_name="Series" ,on_delete=models.SET_DEFAULT)
    tutorialSlug = models.CharField(max_length=200, default=1)

    def __str__(self):
        return self.tutorialTitle
```

Models.py-Klasse

9. Sessions / Transaction

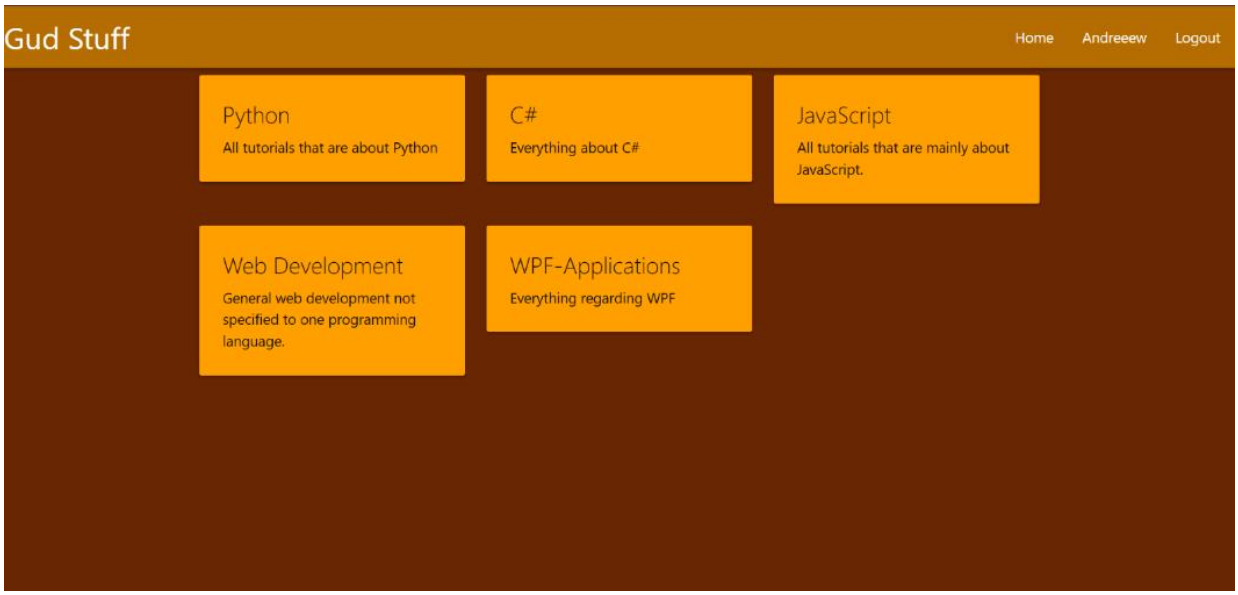
Sessions werden auch automatisch von Django übernommen. Dadurch war es sehr einfach zum Aufsetzten und ist sehr sicher. Die Sicherheit besteht hauptsächlich daraus, dass für den User nur ein SessionID Cookie sichtbar ist, in dem die Session-ID gespeichert ist. Der Rest ist in der Datenbank gespeichert, die von der Userseite natürlich gesichert ist. Die Sicherheit ist also von der Datenbank abhängig, die aber auf dem Server läuft und sowieso sehr gut gesichert ist.



SessionID im Cookie

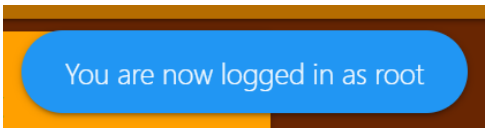
10. Seite

Die Seite ist auf einem Windows-Server gehostet und erreichbar auf http://148.251.211.149:8000/ - Bei Aufruf des Links kommt man auf folgende HomePage:



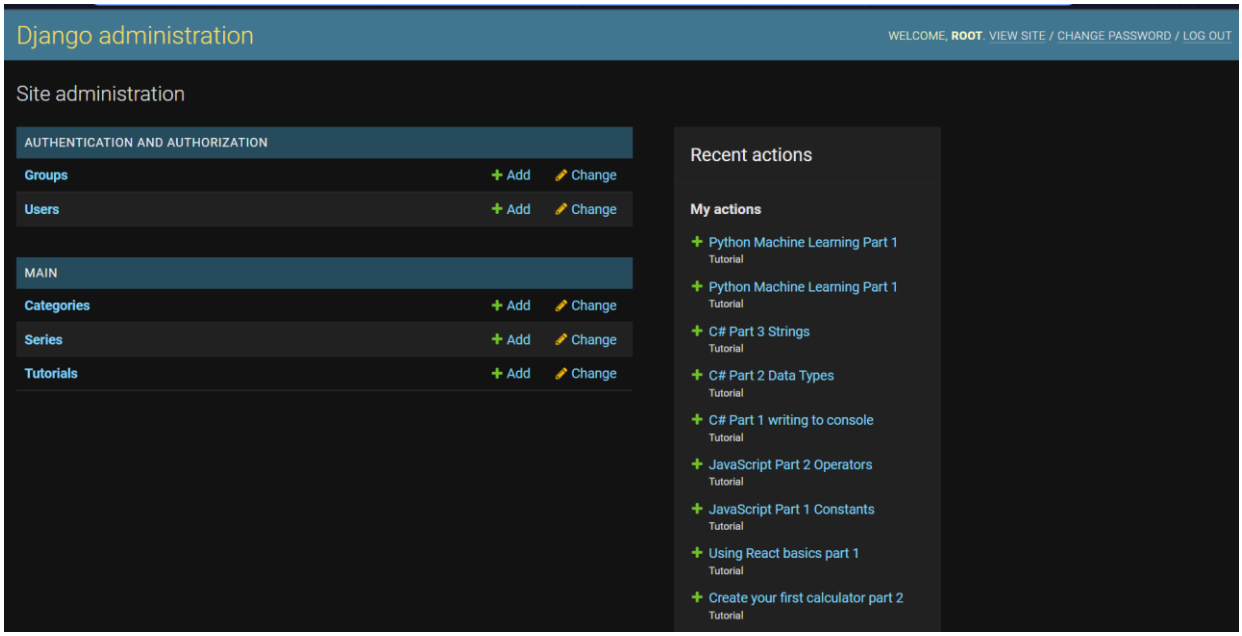
Von dort aus kann man zu den einzelnen Unterkategorien hin-navigieren und man erhält eine Auswahl an Tutorials. Zwischen den einzelnen Tutorials lässt sich auch hin und her navigieren. Es kann ein User registriert, eingeloggt und bearbeitet werden mit dem Kontextmenü oben rechts.

Eines unser persönlichen Lieblingsfeatures ist das Popup, das beim Login und Logout erscheint:



Es gibt ausserdem ein Admin-Panel, welches über http://148.251.211.149:8000/admin/login verfügbar ist - die Nutzerdaten für den Adminaccount sind root : root.

Dort lassen sich dann die einzelnen Tutorials bearbeiten, Neue hinzufügen und anzeigen. Das Ganze sieht dann in etwa so aus:



11. Rückblick

Anhand des Arbeitsjournals lässt sich eventuell schon etwas abschätzen, dass das Projekt nicht ganz so planmässig verlief, wie wir uns das ursprünglich vorgestellt hatten. Dies hat einen guten Grund: Wie bereits oben erwähnt, war André in Berlin für ein Berufspraktikum, welches knapp einen Monat andauerte und es ihm während dieser Zeit kaum möglich machte, tatkräftig am Projekt mitzuwirken. Da das gesamte Angebot für Berlin sehr spontan war (1 Woche vor Abflug), konnte er sich auch nicht wirklich vorbereiten beziehungsweise hatte nicht die Chance, für dieses Projekt vorzuarbeiten. Schlussendlich musste deshalb Colin einen Grossteil der Programmier-Arbeit leisten, während André kleinere Dinge gemacht hat und vor allem dann am Ende noch einen grossen Teil der Dokumentation geschrieben hat. Generell wurden leider mehr Dinge als geplant gegen Ende gemacht, was auf die Abwesenheit von André zurückzuführen ist. Colin hat jedoch trotzdem auch während der Abwesenheit fleissig weitergearbeitet (mit Unterstützung von André, so weit möglich) und konnte André dann bei seiner Rückkehr gut erklären, wie die Einzelteile zusammen funktionieren und noch genauere Angaben zu beispielsweise dem Schutz geben. Alles in allem war es ein sehr schlechtes Timing mit dem Praktikum und wir haben uns einfach versucht, so gut wie möglich daran anzupassen.

Was dagegen gut funktioniert hat war einerseits der Aufbau des User Interfaces und die Datenbankanbindung. Auch die Wahl von Django als Framework stellte sich als gut heraus, da uns dieses Framework an einigen Stellen gut unterstützen konnte. Django machte vieles sehr einfach, in dem fast alles vom Framework gemacht wird, solange man das Framework korrekt nutzt. Wenn man sich an bestimmte Prinzipien beim Aufbau der Seite gehalten hat, wird einem danach sehr viel Arbeit erleichtert. Dadurch war Django optimal für uns, da wir noch nie zuvor so etwas gemacht hatten und so relativ einfach und unkompliziert eine sichere Seite erstellen konnten.

11.1 Commits

Die stetigen Commits in unserem Repository sind grösstenteils ausgeblieben, dies liegt einerseits an der Abwesenheit von André, andererseits daran, dass es ehrlich gesagt einfach etwas untergegangen ist.

12. Fazit

Alles in Allem ist das Projekt doch (unserer Meinung nach) erfolgreich herausgekommen, obwohl wir mit einigen Herausforderungen, inklusive Abwesenheit des halben Teams, zu kämpfen hatten. Dies ist nicht zuletzt der Verwendung eines Frameworks zu verdanken, dass uns beim Schutz einiger der oben erwähnten Angriffsarten tatkräftig unterstütze.