RoomFinder

Webanwendungen mit Python

Hochschule
Augsburg University of
Applied Sciences

Fakultät für Informatik

Felix Kampfer

Betreuer: Dipl.-Inf. (FH), Dipl.-Designer (FH) Erich Seifert, MA

Wintersemester 2017/2018

Einleitung

Motivation

Während eines Semester kommt es öfters vor, dass Studenten kurzzeitig einen Raum benötigen um ungestört arbeiten zu können. Da die meisten Räume der Hochschule Augsburg keinen ausgedruckten Belegungsplan aufweisen, müssen Studenten, die einen solchen Arbeitsplatz suchen, entweder mehrere Türen öffnen (und dabei möglicherweise dutzende von Insassen stören und sich ungewollte Aufmerksamkeit holen) oder auf die WebUntis Plattform über viele Klicks Räume einzeln auswählen, bis sie einen passenden gefunden haben.
Ziel des Projekts ist es, Studenten eine schnellere und elegantere Lösung anzubieten.

Anforderungen an die Anwendung

Die Anwendung "RoomFinder" soll es ermöglichen, freie Räume an der Hochschule Augsburg zu finden. Dabei sollen die freien Räume durch einen grünen Hintergrund in der Übersicht der Hochschule Augsburg gekennzeichnet werden. Optional soll zudem angezeigt werden, wie lange der jeweilige Raum noch frei ist. Belegte Räume hingegen sollen durch einen roten Hintergrund erkennbar sein und anzeigen, wie lange diese noch belegt sind. Als Ermittlungsgrundlage dient die Zeit des Aufrufs der Website.
Um Auskünfte über freie Räume zu erhalten, soll der Benutzer das gewünschte Gebäude auswählen können. Eventuell sollte davor eine Auswahl über den gewünschten Campus (Campus am Brunnenlech oder Campus am Roten Tor) so wie das gewünschte Stockwerk getroffen werden.
Die benötigten Daten sollen über Webuntis ermittelt werden. Da sich diese Informationen nur selten (geschätzt einmal pro Tag) ändern, sollen diese Informationen in einer Datenbank zwischengespeichert werden. Somit soll auch sichergestellt werden, dass die Anwendung nicht bei jedem Aufruf die Daten von Webuntis neu herunterladen und verarbeiten muss..

Begutachtung

Die Webanwendung ist unter der URL http://roomfinder.informatik.hs-augsburg.de/ zu erreichen.

Entwicklung

Ansätze und Implementierung der Benutzereingabe

Zwei Ansätze: Konfigurierbar vs. Minimalistisch

Ideen für Auswahlmöglichkeiten für den Benutzer und die Implementierung der UI beliefen sich auf verschiedene Varianten: Konfigurierbar und Minimalistisch.
Der Konfigurierbare Ansatz würde es dem Benutzer ermöglichen, mittels HTML-Formular ein oder mehrere Gebäude, ein oder mehrere Stockwerke, den Tag und die Uhrzeit festzulegen, für die er eine Auskunft erzielt. Teil des Konfigurierbaren Ansatzes war auch das Einbauen von Logik zum Überprüfen der Benutzereingaben (hat der Benutzer überhaupt eine Uhrzeit eingegeben?) und “Quality of Life” Funktionen wie die Möglichkeit zum Invertieren von den bereits ausgewählten Gebäuden.
Der minimalistische Ansatz würde dem Benutzer lediglich die Möglichkeit geben, ein Gebäude auszuwählen, ohne einzelne Stockwerke oder auch nur die Uhrzeit für die Anfrage einzugeben. Dieser Ansatz hätte den Vorteil, dass der Benutzer schneller die wohl am häufigsten verwendete Funktion ausführen könnte: Das Suchen nach freien Räumen im hier und jetzt.

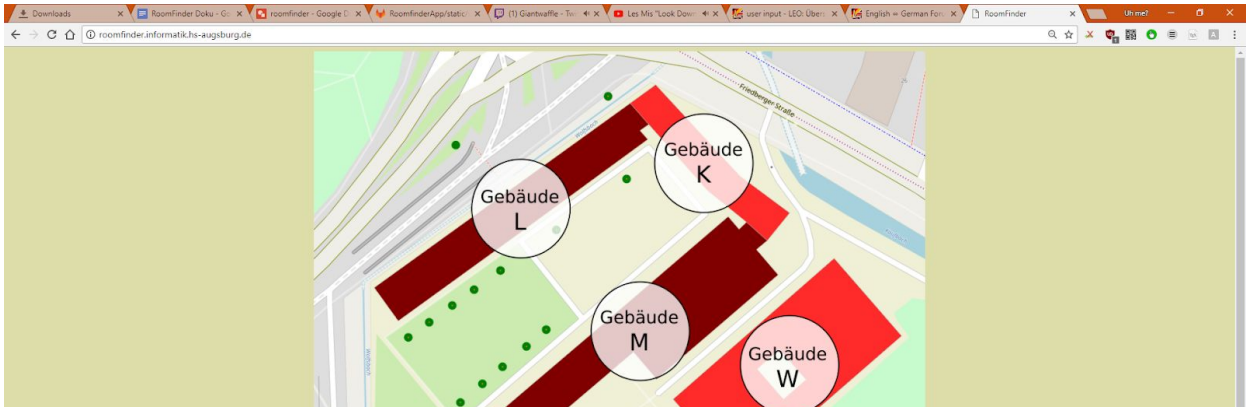
Prototyp der “konfigurierbaren” Benutzereingabe

Die Implementierung des “Konfigurierbaren” Ansatzes wurde wie im folgenden Bild zu sehen ist gehandhabt, überwiegend durch den ehemaligen Projektpartner Daniel Hörmann.

Um dem Benutzer Arbeit zu ersparen wurden außerdem Ansätze in Richtung Session Management geplant, um so die vorherige Eingabe des Benutzers bei seinem nächsten Besuch bereits für ihn ausgewählt zu haben.

Prototyp der “minimalistischen” Benutzereingabe

In der letzten Phase der Entwicklung wurde das komplizierte Formular durch zwei Grafiken ersetzt, basierend auf exportierten und bearbeiteten Karten-Daten von OpenStreetMaps. Dieser Ansatz soll das Abfragen von Verfügbarkeitsdaten sowohl für Desktop und Mobile-Benutzer vereinfachen.



Um zukünftige Weiterentwicklungen in Richtung Interaktivität nicht zu verhindern, wurden weiterhin SVG Grafiken verwendet um, ähnlich wie bei Image Maps, per Auswahl einer Region im Bild den Browser auf eine neue Seite zu navigieren.

Ein Problem bei diesem Schritt war der nicht ganz verwirrungsfreie Charakter von Objekten, welche in Webseiten eingebettet sind. Um die digital beschrifteten Bereiche (Pfade) der SVGs ansprechen zu können, musste man nämlich auf das sog. “contentDocument” der außenliegenden “Object” Tags zugreifen, um dann auf das vollständige Laden der Grafiken zu warten. Erst dann war es möglich, die weißen Kreise im Bild anklickbar zu machen. Rechts ist der Code, der die SVG Kreise anklickbar gemacht hat.

```
var buildings = ["A", "B", "C", "D", "E", "F", "G", "H", "J", "K",
var rotesTor = document.getElementById("rotesTorSVG");
var brunnenlech = document.getElementById("brunnenlechSVG");

function makeCirclesClickable(svgElement) {
    // get the inner DOM of the svg
    var svgDoc = svgElement;
    for (var buildingIndex in buildings) {
        var circleName = "circle"+buildings[buildingIndex];
        // get the inner element by id
        var circle = svgDoc.getElementById(circleName);
        if (circle !== null) {
            circle.addEventListener("mousedown", function() {
                window.location.href="/building/"+this.id.slice(-1)
            }, false);
        }
    }
}

// It's important to add a load event listener to the object,
// as it will load the svg doc asynchronously
rotesTor.addEventListener("load",function(){
    makeCirclesClickable(rotesTor.contentDocument);
}, false);
brunnenlech.addEventListener("load",function(){
    makeCirclesClickable(brunnenlech.contentDocument);
}, false);
```

Entgültige Fassung der Benutzereingabe

Als die Webseite zum ersten mal auf einem mobilen Gerät getestet wurde, ist schnell klar geworden dass die Verwendung der OpenStreetMap SVG Grafiken ein Problem darstellt: Obwohl die die beiden SVG Grafiken insgesamt nur 1,7 Megabyte groß waren, hat es trotzdem länger als zwanzig Sekunden gedauert, bis die Startseite vollständig geladen und die Kreise

gemalt worden sind. Somit wurden die SVG Grafiken zu PNG Bildern konvertiert, und die Verlinkung mithilfe von statischen Image Maps realisiert. Die Benutzereingabe lädt jetzt auf allen getesten Geräten innerhalb weniger Sekunden und entspricht einem akzeptablen Zustand.

Ansätze und Implementierung der Datenabfrage

Erste WebUntis Ansätze

Um Auskünfte über die Verfügbarkeit von Räumen geben zu können, galt es als selbstverständlich, dass man von den Daten von Untis/WebUntis Gebrauch machen müsste, auch wenn Details zur Anbindung noch gefehlt hatten. Erste Recherchen in diese Richtung haben sich an den GET-Requests orientiert, welche beim Auswählen von bestimmten Räumen auf der offiziellen WebUntis-Seite (https://melpomene.webuntis.com/WebUntis/?school=HS-Augsburg) erfolgen. Beim Auswählen von Raum B3.02 werden beispielsweise folgende Parameter mittels GET-Request übermittelt:

Parameter	Erklärung	Wert
elementType	Welche Art von Element (Lehrer, Veranstaltung, Raum) angefragt wird	4
elementId	Einzigartes ID für den Lehrer/Veranstaltung/Raum	95
date	Datum	2018-01-10
formatId	Wie die Information zurückgegeben werden soll, z.B. JSON / iCAL	7
filter.buildingId	Gebäude-ID (Untis-intern)	-1
filter.roomGroupId	Gebäudegruppe (wird evtl nicht verwendet)	-1
filter.departmentId	Fakultät-ID (Untis-intern)	6

Leider entspricht WebUntis einer proprietären Software, mangels öffentlich zugänglicher Dokumentation, und somit ließe sich die Funktionsweise nur schwer entziffern. Problematisch war vor allem der Umgang mit dem Rückgabewert. Für die obige Anfrage hat die dazugehörige Response (im JSON Format) mehr als zwei tausend Zeilen, welche durch die extreme Verschachtelung von Daten (Verweise auf Verweise mittels nicht dokumentierten IDs) noch schwieriger gemacht wird.

Als alternative Möglichkeit zu den GET-Requests wurde außerdem das Herunterladen der ICS-Kalender Dateien untersucht, welche auf der WebUntis Seite zu jedem Raum vorhanden sind. Diese Dateien müssten dann allerdings erst mittels selbstgeschriebenem Parser ausgelesen werden, was einem Mehraufwand entsprechen würde. Außerdem müsste man diese Datei bei jedem der 237 Räumen abrufen. Hier ein Beispiel für eine Vorlesung in B3.02 im ICS-Format:

```
BEGIN:VEVENT
DTSTAMP:20180110T063505Z
DTSTART:20180109T070000Z
DTEND:20180109T083000Z
UID:1369020-64438-
SUMMARY:M-BM 3A Mess- und Regelungstechnik 1 V
LOCATION:B3.02
END:VEVENT
```

Andere Ansätze zum Umgang mit Daten von WebUntis lassen sich in Alexander Kulows Bachelorarbeit “Evaluation der Funktionalität des Amazon Kindle als Alternative zu etablierten Einplatinencomputern am Beispiel eines Systems zur energieeffizienten Anzeige von Vorlesungsplänen vor den Hörsälen” bzw. Marvin Reiters Bachelorarbeit “HSApp: Cross-Platform Mobile App Development mit Xamarin am Beispiel einer Hochschul-App“ nachlesen.

WebUntis Traumlösung

Die regelrechte Traumlösung für den Umgang mit WebUntis-Daten entstand durch den Kontakt zu einem anderen Studenten an der Hochschule mit ähnlichen Zielen, welcher dem Projekt Zugangsdaten bereitstellen konnte. Mit diesen Zugangsdaten lässt sich die inoffizielle aber einfach zu bedienende Python-API für WebUntis verwenden. Die Anleitung zur Python-API enthält den Hinweis “You need an account for the API. Many schools just make the timetable world-accessible though, preventing any use of the API. If you happen to be at such a school, you’re a pitiful bastard”, weswegen dieser Ansatz ohne Zugangsdaten zuerst keine Lösung dargestellt hatte.

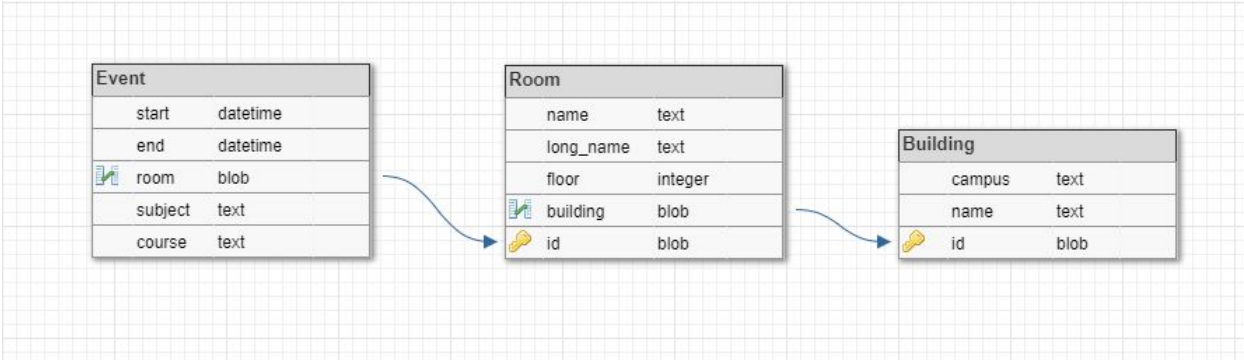
Mit der Python API lassen sich wie im folgenden Beispiel alle Termine für die nächste Woche für alle Räume der Hochschule abrufen (sofern man sich in der Session bereits eingeloggt hat):

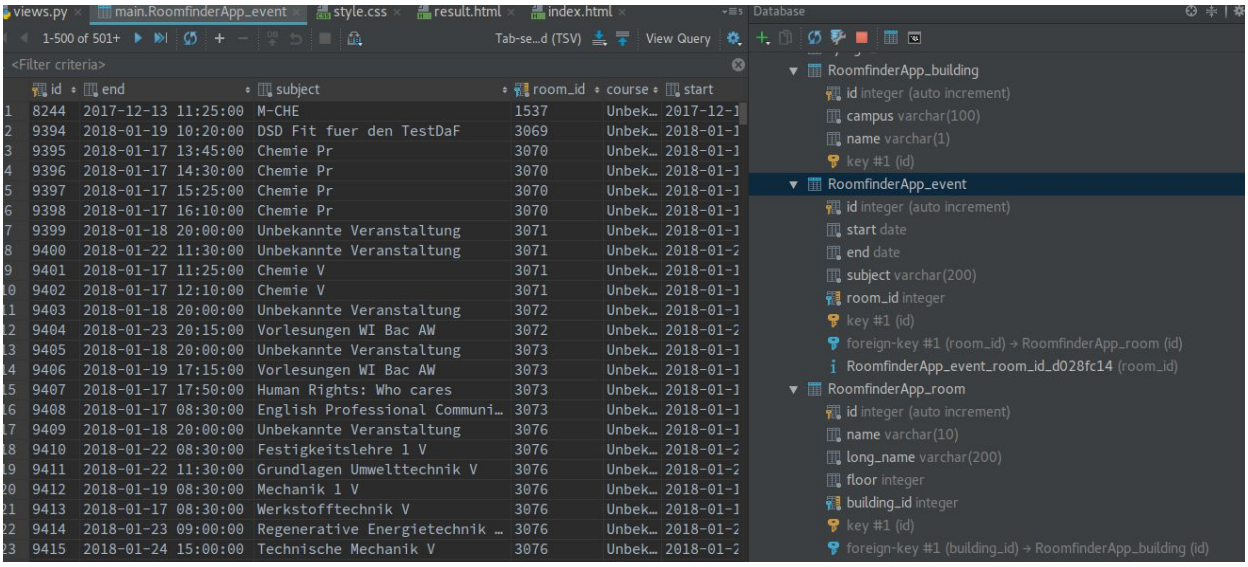
```
all_rooms = s.rooms()
for room in all_rooms:
    untis_events = s.timetable (room=untis_room, start=[...].now(), end=[...]+timedelta(days=7))
    # [...] entspricht einem Aufruf der Standardklasse datetime bzw. datetime.now()
```

Ein bisschen müssen die Daten allerdings noch sauber gemacht werden. Manche Untis-Räume gibt es nämlich beispielsweise doppelt (z.B. K1.06), andere sind keine Räume (wie z.B. der Raum “Abfrage-Ansicht”), und dann gibt es auch noch Räume die nicht auf dem Campus existieren (wie z.B. P3.04). Allen Veranstaltungen ohne Namen werden dann noch Dummy-Werte zugeschrieben, und alle Zeitangaben werden noch zur Zeitzone 'Europe/Berlin' konvertiert, da Untis diese ohne Zeitzone speichert und das Paket datetime ohne Zeitzonen-Angaben nicht zurechtkommt.

Zwischenspeichern der Daten

Theoretisch würden sich nach der Auswahl eines Gebäudes die angeforderten Raumbelegungen sofort von WebUntis abrufen und an den Benutzer ausgeben lassen. Leider wird durch die Python-API jeder Raum einzeln nacheinander abgerufen, wobei jede Abfrage etwas länger als eine Sekunde dauert. Um den Benutzer nicht all zu lange warten zu lassen, hat man sich dafür entschieden, die Daten in einer mit Django integrierten Sqlite3 Datenbank zwischenzuspeichern. Momentan müssen die Rauminformationen durch den Besuch von http://roomfinder.informatik.hs-augsburg.de/update noch manuell aktualisiert werden. Dieser Prozess dauert etwas länger als drei Minuten. Die Datenbank wurde wie im folgenden Bild aufgebaut, wobei die IDs von SQLite intern gehandhabt werden. Ein Gebäude hat somit mehrere Räume, und jeder Raum hat mehrere Veranstaltungen.





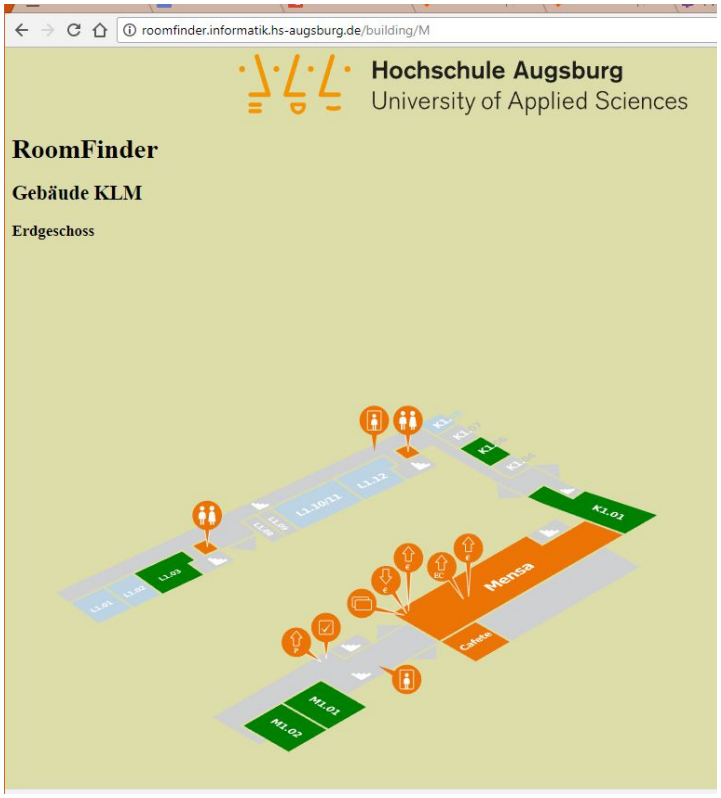
Ausgabe von Daten

Damit der Benutzer der Applikation nicht nur Text sieht, sondern grafische Auskünfte über die Verfügbarkeit von Räumen erlangt, war es auch nötig sich Grafiken der Gebäudepläne der Hochschule zu beschaffen. Als Notlösung hätte sich möglicherweise das Fotografieren von Evakuierungsplänen angeboten, aber die Qualität und v.a. die digitale Aufbereitung der Daten hätten stark dagegen gesprochen.
Zum Glück wurde dank freundlicher Unterstützung des OHA-Teams der Hochschule, welches jedes Semester einen Orientierungshelfer/Kalender für neue Studenten anbietet, ein uneingeschränkten Zugang zu den Gebäudeplan-Grafiken der Hochschule in .eps, .ai, .svg und .pdf Formaten gewährleistet. Da man die Grafiken noch etwas bearbeiten wollte, und kein Zugang zu Adobe Produkten bestand, hat man sich für die Bearbeitung der SVG-Dateien mittels Atom und Inkscape entschieden. Mit diesen Werkzeugen konnte man sich in die dutzenden Stockwerk-Plänen durcharbeiten um so jeden einzelnen Raum digital beschriften zu können.
Waren die Räume einmal beschriftet, kann man ganz einfach im Browser mittels Javascript die individuellen Räume der Hochschule auswählen und je nach Verfügbarkeit einfärben, wie im folgenden Codeausschnitt zu sehen ist:

```
for (var room in roomData) {
    var roomGraphic = document.getElementById(roomData[room].name);
    if (roomData[room]["free"]) {
        roomGraphic.setAttribute("fill","green");
    } else {
        roomGraphic.setAttribute("fill","red");
    }
}
```

Ein ursprüngliches Problem mit den Gebäudegrafiken war, dass manche von ihnen nicht korrekt im Browser dargestellt wurden (die Räume wurden teilweise mit einer undurchsichtigen Ebene überdeckt). Als Lösungsansatz wurden die betreffenden SVGs mit Atom, Inkscape und auch einem Online-Optimierungstool neu exportiert (und wurden von den jeweiligen Editoren auch korrekt angezeigt), allerdings ohne Erfolg.
Die Lösung bestand daraus, die SVG-Dateien innerhalb von Objekt-Tags unterzubringen, anstatt alle SVG Grafiken innerhalb des gleichen HTML-Elements zu platzieren. Dank Djangos Template Sprache kann man mit wenigen Zeilen Code alle relevanten Ebenenpläne für ein Gebäude anzeigen. Wenn die Ebenenpläne beispielsweise J1.svg, J2.svg, usw. heißen, kann man sie wie in diesem Codeausschnitt zu sehen ist in einem HTML Dokument einbauen:

```
<object id="{{ building }}{{ floor }}"
        class="floorGraphic"
        data="{% static 'roomplans/' %}{{ building }}{{ floor }}.svg"
        type="image/svg+xml"></object>
```



Links ist die nach der Verfügbarkeit gefärbte grafische Ausgabe zu sehen. Die orange gefärbten Räume und Symbole entstammen den Originalgrafiken des OHA-Teams. Da sie tendenziell besondere Räume kennzeichnen, dessen Reservierung nicht möglich ist, hat man sie nicht verändert.

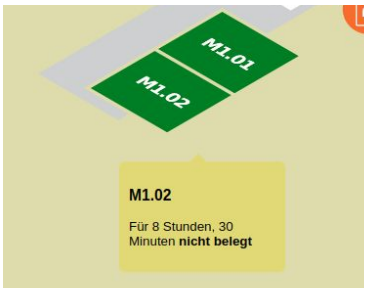
Beispiel-Durchlauf

Im Folgenden wird die Funktionsweise der RoomFinder Applikation anhand einer beispielhaften Benutzeranfrage näher erläutert.

1. Der Benutzer klickt auf das Gebäude M und navigiert so zu http://roomfinder.informatik.hs-augsburg.de/building/M
2. Mittels des URL-Patterns “url(r'^building/(?P<building>\w+)$', views.result)“ ruft der Server die Methode “request” auf mit “M” als Wert für den Parameter “building“
3. Als nächstes ermittelt der Server die für das Gebäude dazugehörigen Stockwerke
4. Weil die Gebäude K und L zwar separat von M abgespeichert sind, aber Teil des gleichen Gebäudeplans sind, werden sie zur weiteren Verarbeitung zum “buildings” Objekt hinzugefügt
5. Nun werden alle Räume ermittelt, die in den jeweiligen Gebäuden vorhanden sind
6. Für jeden Raum wird jetzt die aktuelle Verfügbarkeit festgestellt, und die Zeitdauer bis sich diese Verfügbarkeit ändert. Außerdem wird für belegte Räume die Veranstaltung (der Kurs) und die Klasse ermittelt. Um die Verfügbarkeit herauszufinden, wird für jeden Raum jede Veranstaltung wie folgt analysiert:

```
duration_until_occupied = datetime.timedelta(days=20)
duration_until_available = datetime.timedelta(minutes=0)
#...
for event in Event.objects.filter(room=room):
  if now < event.start:
    # before the event
    if duration_until_occupied > event.start - now:
      duration_until_occupied = event.start - now
  elif event.start <= now <= event.end:
    # during the event
    availability = False
    course = event.course
    subject = event.subject
    if duration_until_available < event.end - now:
      duration_until_available = event.end - now
time_until_change = (duration_until_occupied if availability else duration_until_available)
room_info.append(RoomView(room.name, course, subject, availability, time_until_change))
```

7. Die ermittelten Informationen werden zu JSON konvertiert und in das resultierende HTML-Dokument eingebaut. Außerdem wird die time_until_change von einem Python-internen timedelta Objekt zu einem deutschen Text konvertiert.
8. Der Browser zeigt das fertige HTML Dokument an, mit farblich markierter Verfügbarkeit. Der Benutzer kann außerdem mit der Maus über einen Raum fahren und einen Tooltip genießen, welcher Informationen über den Raum und seine Verfügbarkeit ausgibt:



Ein bestehendes Problem bei der Anzeige des Tooltips ist dass er je nach Auflösung und Zoom-Level des Browsers eine falsche Positionierung aufweist. Auf mobilen Geräten lässt sich der Tooltip daher praktisch nicht verwenden.

Fazit

Trotz mancher Ärgerlichkeiten hat die Webanwendung sein Ziel vollständig erfüllt und kann Studenten im Studiumsalltag durchaus helfen. Zu klären bleibt eventuell noch, ob das kostenlose Angebot dieser Funktionalität gegen WebUntis Nutzungsrechte verstößt.