

Cloud Computing und SaaS

Exercise – GAE

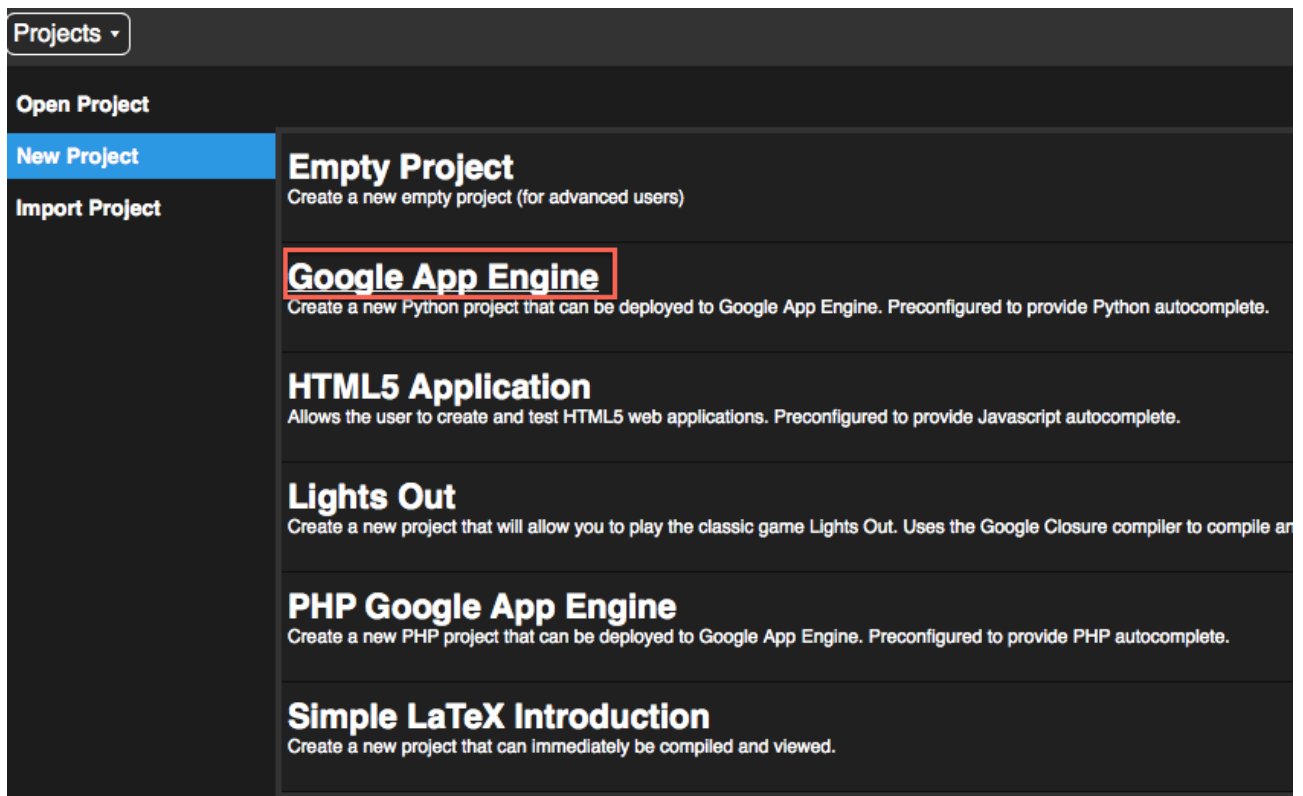
Hannes Knopf

Gewählte Entwicklungsmethode

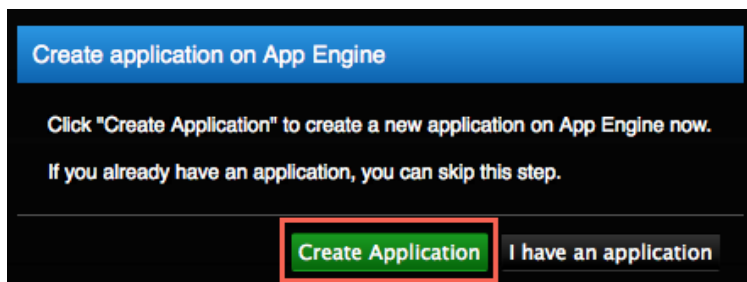
Python mit der DevTable IDE

Neues Projekt anlegen

New Project => Google App Engine



Beim Erstellen bekommt man auch gleich die Möglichkeit das Projekt mit einer Applikation auf der Google App Engine zu erstellen bzw. zu verknüpfen.



Mit 'Create Application' kommt man auf die *Google App Engine* Seite.

Create an Application

You have 9 applications remaining.

Application Identifier:

fhwncloudtodo .appspot.com

Check Availability

All Google account names and certain offensive or trademarked names may not be used as Application Identifiers. You can map this application to your own domain later. [Learn more](#)

Application Title:

Displayed when users access your application.

Authentication Options (Advanced): [Learn more](#)

Google App Engine provides an API for authenticating your users, including **Google Accounts**, **Google Apps**, and **OpenID**. If you choose to use you'll need to specify now what type of users can sign in to your application:

☒ **Open to all Google Accounts users (default)**

If your application uses authentication, anyone with a valid Google Account may sign in.

☐ **Restricted to the following [Google Apps](#) domain:**

e.g. foo.com

If your application uses authentication, **only members of this Google Apps domain may sign in**. If your organization uses Google Apps, use an HR tracking tool) that is only accessible to accounts on your Google Apps domain. This option cannot be changed once it has been set.

☐ **(Experimental) Open to all users with an OpenID Provider**

If your application uses authentication, anyone who has an account with an OpenID Provider may sign in.

Create Application

Cancel

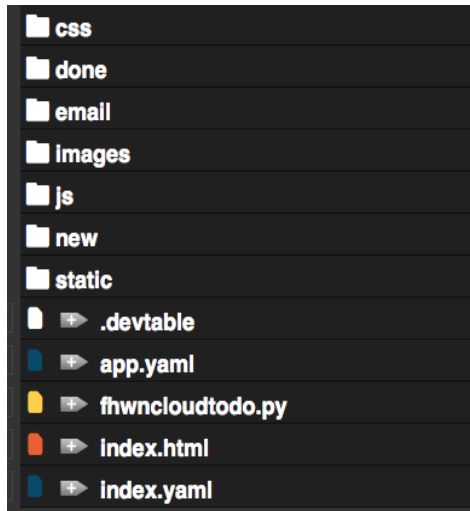
Deployed man das Projekt, so wird die Applikation unter *fhwncloudtodo.appspot.com* (wie oben mit dem Application Identifier angegeben) erreichbar sein.

Kopieren vom Tutorial

<http://www.vogella.com/tutorials/GoogleAppEngine/article.html>

Beim Kopieren muss man auf Python-Einrückungsfehler aufpassen und diese korrigieren.

Nach dem Kopieren vom Tutorial entsteht folgende Ordner-Struktur:



Das app.yaml folgender Maßen anpassen:

```
application: fhwncloudtodo
version: 1
runtime: python27
threadsafe: true
api_version: 1

handlers:
- url: /css
  static_dir: css
- url: /js
  static_dir: js
- url: /images
  static_dir: images
- url: /*
  script: fhwncloudtodo.app

- url: /static
  static_dir: static

libraries:
- name: webapp2
  version: latest
```

Hinzufügen von 'importance'

In der Klasse *TodoModel* - Hinzufügen des Attributes *'importance'*

```
class TodoModel(db.Model):
    author = db.UserProperty(required=True)
    shortDescription = db.StringProperty(required=True)
    longDescription = db.StringProperty(multiline=True)
    url = db.StringProperty()
    created = db.DateTimeProperty(auto_now_add=True)
    updated = db.DateTimeProperty(auto_now=True)
    dueDate = db.StringProperty(required=True)
    finished = db.BooleanProperty()
    importance = db.StringProperty()
```

In der Klasse *New* – Beim Erstellen eines neuen ToDo's wird eine Instanz von einem *TodoModel* angelegt. Auch hier muss natürlich das Attribut *'importance'* mit dem eingegeben/übertragenen Wert gesetzt werden.

```
class New(webapp2.RequestHandler):
    def post(self):
        user = users.get_current_user()
        if user:
            testurl = self.request.get('url')
            if not testurl.startswith("http://") and testurl:
                testurl = "http://" + testurl
            todo = TodoModel(author = users.get_current_user(),
                             shortDescription = self.request.get('shortDescription'),
                             importance = self.request.get('importance'),
                             longDescription = self.request.get('longDescription'),
                             dueDate = self.request.get('dueDate'),
                             url = testurl,
                             finished = False)
            todo.put()
            self.redirect('/')
        else:
            self.redirect('/')
```

Im *index.html* - In der Table der offenen ToDo's, muss natürlich der Header der Table um *'Importance'* erweitert werden. Zusätzlich muss für jedes ToDo, welches in einer Table Row dargestellt wird, auch der Wert des Attributes *'importance'* angezeigt werden.

```
<table>
  <tr>
    <th>Short description </th>
    <th><a href="/?sort={{sort}}">Importance</a></th>
    <th>Due Date</th>
    <th>Long Description</th>
    <th>URL</th>
    <th>Created</th>
    <th>Updated</th>
    <th>Done</th>
    <th>Send Email reminder</th>
  </tr>

  {% for todo in todos %}
  <tr>
    <td>
      {{todo.shortDescription}}
    </td>
    <td>
      {{todo.importance}}
    </td>
```

Im *index.html* – Im Formular für das Erstellen eines neuen ToDo's, muss eine weitere Table Row für das Attribut 'importance' angelegt werden. Dies ist eine ComboBox mit 3 Auswahlmöglichkeiten.

```
<form action="/new" method="post">
  <table>
    <tr>
      <td><label for="shortDescription">Summary</label></td>
      <td><input type="text" name="shortDescription" id="shortDescriptions" size="80"/></td>
    </tr>
    <tr>
      <td><label for="importance">Importance</label></td>
      <td><select name="importance">
        <option value="1">1</option>
        <option value="2">2</option>
        <option value="3">3</option>
      </select></td>
    </tr>
  </table>
</form>
```

Sortieren nach 'importance'

Anfangs wollte ich die Table mit einer jQuery Library (Tablesorter 2.0) sortierbar machen. Dies hat aber auf die schnelle nicht funktioniert, somit hab ich gegoogelt: https://developers.google.com/appengine/docs/python/datastore/queries#Python_query_interface

In diesem Link steht eine *GqlQuery* mit einem *Order By*, und in einem Absatz darüber steht auch etwas von *Indexes* anlegen. Da mir auch ein Index-Fehler beim Deployen bzw. Testen entgegen kam, welche folgende indexes forderte.

```
indexes:
- kind: TodoModel
  properties:
  - name: author
  - name: finished
  - name: importance
```

```
indexes:
- kind: TodoModel
  properties:
  - name: author
  - name: finished
  - name: importance
  direction: desc
```

Die Indexes angelegt in einer *index.yaml* Datei, ergeben folgende Einträge im *Data Store Indexes* der Applikation in der *Google App Engine*

Entity and Indexes	Status	Index Entry Count	Index Storage
TodoModel			
author ▲ , finished ▲ , importance ▲	Serving		
author ▲ , finished ▲ , importance ▼	Serving		

Für das Sortieren wird im GET-Request die Variable *'sort'* abgefragt. Je nach dem welchen Wert die Variable hat wird aufsteigend oder absteigend sortiert. *'sort'* wird in den values übertragen, damit man im HTML Code im *<a> Tag* mitschicken kann, nach welchem Wert man als nächstes sortiert.

```
class MainPage(webapp2.RequestHandler):
    def get(self):
        user = users.get_current_user()
        url = users.create_login_url(self.request.uri)
        url_linktext = 'Login'

        if user:
            url = users.create_logout_url(self.request.uri)
            url_linktext = 'Logout'

        # GQL is similar to SQL
        order = ""
        if self.request.get('sort') == "":
            sort = "asc"
        elif self.request.get('sort') == "asc":
            order = "ORDER BY importance ASC"
            sort = "desc"
        elif self.request.get('sort') == "desc":
            order = "ORDER BY importance DESC"
            sort = "asc"

        todos = db.GqlQuery("SELECT * FROM TodoModel WHERE author = :author and finished=false " + order,
                             author=users.get_current_user())

        doneTodos = TodoModel.gql("WHERE author = :author and finished=true",
                                   author=users.get_current_user())

        values = {
            'todos': todos,
            'sort': sort,
            'numbertodos': todos.count(),
            'doneTodos': doneTodos,
            'numberDoneTodos': doneTodos.count(),
            'user': user,
            'url': url,
            'url_linktext': url_linktext,
        }
        self.response.out.write(template.render('index.html', values))
```


Hinzufügen von History

Ändern der Klasse *Done* – Wenn auf ein ToDo eine 'Done' Aktion ausgeführt wird, soll diese 'finished' gesetzt werden.

```
class Done(webapp2.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if user:
            raw_id = self.request.get('id')
            id = int(raw_id)
            todo = TodoModel.get_by_id(id)
            todo.finished = True
            todo.put()
            self.redirect('/')

```

In der Klasse *MainPage* – Abfragen und Bereitstellen der 'finished' ToDo's.

```
doneTodos = TodoModel.gql("WHERE author = :author and finished=true",
    author=users.get_current_user())

values = {
    'todos': todos,
    'sort': sort,
    'numbertodos' : todos.count(),
    'doneTodos': doneTodos,
    'numberDoneTodos' : doneTodos.count(),
    'user': user,
    'url': url,
    'url_linktext': url_linktext,
}
```

Im *index.html* – Hier muss natürlich ähnlich der Table der offenen ToDo's eine Table erstellt werden. Die Daten bekommt sie von den in der Klasse *MainPage* gesetzten values 'doneTodos' und 'numberDoneTodos'.

```
{% if numberDoneTodos %}
History - You have a total number of {{numberDoneTodos}} Todos done.

<table>
  <tr>
    <th>Short description </th>
    <th>Importance</th>
    <th>Due Date</th>
    <th>Long Description</th>
    <th>URL</th>
    <th>Created</th>
    <th>Finished</th>
  </tr>

  {% for todo in doneTodos %}
  <tr>
    <td>
      {{todo.shortDescription}}
    </td>
    <td>
      {{todo.importance}}
    </td>

```


Ergebnis



Todos

You have a total number of 2 Todos.

Short description	Importance	Due Date	Long Description	URL	Created	Updated	Done	Send Email reminder
exercise 2	1	21.04.2014			17.02.2014	17.02.2014	Done	Email
exercise 3	3	21.03.2014			17.02.2014	17.02.2014	Done	Email

Create new Todo?

Summary

Importance

Due Date

Description

URL

Create

History - You have a total number of 1 Todos done.

Short description	Importance	Due Date	Long Description	URL	Created	Finished
exercise 1	2	21.02.2014			17.02.2014	17.02.2014

So sieht das Ergebnis aus. Oben die Table der offenen ToDo's, mit der Möglichkeit der Sortierung nach 'Importance'. Darunter das Formular zum Erstellen eines neuen ToDo's, mit dem neuen Attribut 'Importance'. Und ganz unten die History. In diese Tabelle kommen diejenigen ToDo's, welche aus der obigen Tabelle mit der Aktion 'Done' auf 'finished' gesetzt wurden.