Basics of Cloud Computing 2018/19 spring

Cus

- Main (https://courses.cs.ut.ee/2019/cloud/spring/Main/HomePage)
- Lectures (https://courses.cs.ut.ee/2019/cloud/spring/Main/Lectures)
- Practicals (https://courses.cs.ut.ee/2019/cloud/spring/Main/Practicals)
- Results (https://courses.cs.ut.ee/2019/cloud/spring/Main/Results)
- Submit Homework (https://courses.cs.ut.ee/2019/cloud/spring/Main/Homework)

Practice 7 - Google App Engine continued

In this practice session we will continue with Google App engine. You will learn how to store arbitrary records in the Google data store service and extend the last week solution by creating another web page in your application where users can leave and read messages.

References

Referred documents and web sites contain supportive information for the practice.

- Python 2.7 tutorial (http://docs.python.org/2/tutorial/)
- Google App Engine tutorial for Python 2.7 (https://developers.google.com/appengine/docs/python/gettingstartedpython27/introduction)
- WebApp2 Tutorial (https://webapp2.readthedocs.io/en/latest/tutorials/quickstart.html)

Exercise 7.1. Creating additional Google App pages

- Lets create a new sub-page under your Google App engine application
  - NB! Make sure you DO NOT overwrite your current main page as this would remove your previous lab submission when you deploy your application later!
- Create a new python file (named `second.py`) under your Google app engine project folder
  - Inside the new python file, create a new class for displaying the page content
    - For example: `class SecondMainPage(webapp2.RequestHandler):`
    - Also add a `get(self)` method and display some text on the web page. `get(self)` method corresponds to HTTP GET request.
  - Import the webapp2 module: `import webapp2`
- In the original python file, import the classes from the newly created file:
  - `from second import *`
- In the original python file, Create a new entry under `application = webapp2.WSGIApplication` for the newly created page
  - It should link `/secondexercise` to `SecondMainPage` class
- Deploy the application and check that everything is working correctly and `/secondexercise` page is accessible.
- From now on, continue adding new content to the `second.py` file while only changing the original file to modify the `application = webapp2.WSGIApplication` entries.

Exercise 7.2. Using the Google NDB Datastore service

We will now implement a message board functionality for the `/secondexercise` page and you will learn how to store data (In our case it will be messages) in the Google Datastore. You should notice that you do not need to set up a database or database schemas, all you need to do is to use the Google ndb API and the PaaS takes care of everything else for you.

- Create a new NDB object class **Message**
  - Check https://cloud.google.com/appengine/docs/standard/python/ndb/creating-entity-models (https://cloud.google.com/appengine/docs/standard/python/ndb/creating-entity-models) to see how it is done.
  - It must have the following fields
    - message_from - name of the sender, String type
    - message_text - message content, String type
    - message_time - time of the message, DateTime format
      - DateTime should have an option: `auto_now=True`
      - You can check the supported data types from here: https://cloud.google.com/appengine/docs/standard/python/ndb/entity-property-reference (https://cloud.google.com/appengine/docs/standard/python/ndb/entity-property-reference)
- Create a new form for entering messages on the **/secondexercise** page
  - `<form action="/message" name ="myform" method="post">`
  - `From: <input type="text" name="from" /> <br />`
  - `Message: <input type="text" name="text" />`
  - `<input type="submit" value="Submit" name="submit">`
  - `</form>`
- Create a class **HandleMessage** for processing the message form and link it to sub-page **/message**
- Add a `def post(self):` method to **HandleMessage** class
  - `post(self)` method corresponds to HTTP POST request.
  - You can access the entered form data from inside the post method like this:
    - `m_from = self.request.get('from')`
  - Create a new Message object using the entered data and enter it to database:
    - `message = Message(message_from=m_from, message_text=m_text)`
    - `message.put()`
  - output a confirmation about the entered message and create a HTML link back to the https://<your_google_app_id>.appspot.com/secondexercise (https://<your_google_app_id>.appspot.com/secondexercise) page

Exercise 7.3. Displaying data from NDB Datastore

We also need to display the messages which have been submitted by the users.

- Modify the **/secondexercise** page to display the latest 10 Messages from NDB
  - Check https://developers.google.com/appengine/docs/python/ndb/queries (https://developers.google.com/appengine/docs/python/ndb/queries) on how to query NDB.
  - Create a query for all data from Message's
  - Sort by Message.message_time in Descending order
  - Fetch 10 entries
  - create a cycle over all the results and print them out one at a time

- Use cgi library to escape any unneeded html characters
  - `import cgi`
  - For example: `self.response.out.write(cgi.escape(message.message_text))`

Exercise 7.4. Add user identification and information to the Messages

Lets simplify creating messages by fetching the user name directly from the Google App Engine user service, like we did in the last lab.

- Use the Google **user** API for log in and authentication, just like in last week lab.
- Verify that user authentication is working. User should not be able to send or view messages if they have not logged in.
- Remove the "From" field from the message form and replace it with the username or full name from the google user service (`user.nickname()`).
- Display the name of the user in the list of posted messages.

Bonus exercise (2p)

The goal of the bonus task is to add a delete message functionality to your page and change user authentication so that only user who submitted the message can access and use the delete message functionality.

- Generate delete buttons next to each message that belongs to you. Should only be shown if your user is logged in. Clicking the button should delete the message from Datastore.
- Create a class/page for deleting a messages (i.e. `/delete`). Should not work wrong when no user is logged in. The page should also leave a notice that `access is denied` and have a link to `Log In` .
- When submitting the lab solution, make sure to leave a easily noticeable comment that you have completed the bonus task.

Deliverables

- Deploy the latest version of the application in App Engine
  - NB! Verify that your previous lab assignment is also still working in appspot.com
- Provide the full web link to your GoogleAppEngine second exercise page
- Upload the source code of your full Google App.

Task	lab 7
Current submission	TAR (/course-2065/submissions/get/7/B91541_tar/B91541_4309_1) (20:46 29.03.2019)
	If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File no file selected
Comment	

https://emmanuellab6.appspot.com/secondexercise

attempted delete, but did not complete it

Submit

Institute of Computer Science (http://cs.ut.ee/) | Faculty of Science and Technology (http://reaalteadused.ut.ee/) | University of Tartu (http://www.ut.ee/)

In case of technical problems or questions write to: ati.error@ut.ee (mailto:ati.error@ut.ee)

The courses of the Institute of Computer Science are supported by following programs:

iktp

European Union
Regional Development Fund

Investing in your future

European Union
European Social Fund

Investing
in your future

tiigriülikool
Programm IKT toetuseks kõrghariduses

Study IT in .ee
sponsored by Skype™