Google AppEngine lab

TLC/ST course

The purpose of this lab is to discover a Platform as a Service (PaaS) solution and see what can be their benefits for developpers. We choosed Google AppEngine for this lab, but other PaaS solutions exist that have some slightly different features, like Heroku, CleverCloud or Openshift.

You will need to access the Google Cloud Platform console to create and deploy your application at this URL: `https://console.cloud.google.com/`. You can use your existing Google Account (if you have a GMail address for example) or create a new one otherwise. We will use the Google Cloud Platform free tier, however you will probably still be required to enter a credit card. At least, your credit card will **not** be automatically charged if you exceed a limit. If you have any difficulty, please contact the lecturer of this course.

---

1 Discovering Google AppEngine

This part will not be assessed

You will start your journey here: `https://cloud.google.com/appengine/docs/`. You must start by choosing a language, a version and an environment to access the corresponding documentation. You are free to use any couple of language and version supported by Google AppEngine in this lab. We will use the standard environment. Now you should follow the Quickstart guide and deploy your first application.

Google AppEngine doesn't allow you to write files on disk, but you will need to persist some data in this lab. To do so, we will use Google Datastore. The Google Datastore Getting Started guide[1] will teach you how to integrate it to your project. You can enhance your previous Hello World application to log every requests in the datastore and display the log on a page. We also recommend the reading of the Google Datastore Concepts documentation[2] which might give you useful insights.

2 Develop your Fitness Application Backend

Your company decided to develop the next trending fitness application which will let people record their performances while they are running, similar to figure 1. To do so, the software architect chose to create a mobile application that send its data to a backend hosted in the "cloud". You will only be in charge of the backend of your service. You don't have to worry about authentication or authorization. You must provide an HTTP API, like a HTTP REST endpoint[3], that will be used by the mobile application to store and search a user and its friends runs. When your users will record their runs, your application will regularly save user's position (latitude and longitude), the associated timestamps (the date and the time), user's username and finally a unique identifier that identify the specific run. We identify these data as a `record`. All `records` with the same run identifiers are simply identified as a `run`. Of course, you must store the

[1] `https://cloud.google.com/datastore/docs/datastore-api-tutorial`
[2] `https://cloud.google.com/datastore/docs/concepts/overview`
[3] `https://en.wikipedia.org/wiki/Representational_state_transfer`

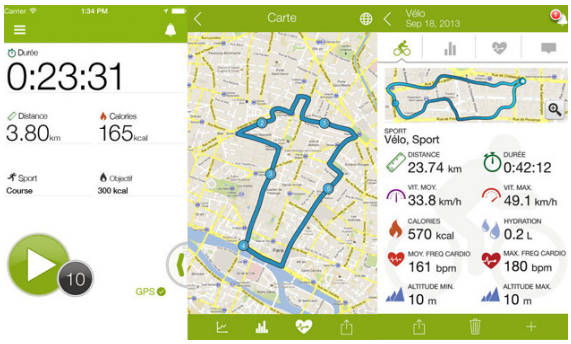

Figure 1: An example of a fitness application

data somewhere to survive a reboot of your backend. You decide to start simple by providing only these three main functionnalities:

Bulk Add To prevent data loss, each created `record` is directly sent to the server. However, sometimes network will not be available, that's why we must be able to send multiple records in one request when network comes back.

Search `records` are public, letting anyone to search them. A user must be able to search `records` by selecting any combination of user, run identifier, before and/or after a timestamp and inside a defined rectangular area (eg. between 48.8400, 2.1000 and 49.2147, 2.4201). To do so, you will need to add indexes on your data. Verify that your implementation works with many different combinations of requests. *From this functionnality, it must be possible to retrieve every `records` of a `run` to display it to your end user or to retrieve every `records` in a specific zone at a specific date to see if a place is crowded.*

Bulk Delete A user must be able to delete multiple `runs` at once (we don't want to delete `records` separately as it would change the meaning of the associated `run`). The bulk delete operation is the less straightforward one. If you don't see how to implement this one, look at the data returned by the search operation corresponding to the objects you want to delete and see what information you can extract to trigger a deletion.

As you will benchmark your application later, try to be clever when using Datastore: you can set indexes, do bulk add and delete, etc. If you don't want to code the whole application, you can use one of the skeleton available here: `https://gitlab.inria.fr/qdufour/tlc-student/tree/master/lab1`.

To test your API, if you choosed to build a HTTP REST endpoint, you can use curl[4] in command line or any graphical REST client, like Postman[5]. Here is an example to operate a bulk add with curl (but with only one record):

```
curl \
  --header "Content-Type: application/json" \
  --request POST \
  --data '[{"id":9,"lat":48.8601,"long":2.3507,"user":"lea","timestamp":1543775727}]' \
  https://tlcgae.appspot.com/api/run
```

Another example with curl to do a search (only records for `lea` between `1543775726` and `1543775729`):

```
curl "https://tlcgae.appspot.com/api/run?user=lea&timestamp=1543775726,1543775729"
```

And finally an example to delete many runs by identifier (delete runs 6, 7 and 9):

[4] `https://curl.haxx.se/docs/manpage.html`
[5] `https://www.getpostman.com/apps`

```
curl \
  --request DELETE \
  https://tlcgae.appspot.com/api/run/6,7,9
```

3 Measure the Scalability of your Backend on Google AppEngine

In this task, you are asked to investigate the scalability of the Google AppEngine (as far as you are able to under the free quotas imposed by GAE). To do so, you will need to submit requests to your backend to perform bulk add, search, and bulk delete functions that you have implemented previously. You then need to measure and analyse the round-trip latency for receiving responses from the backend as the number of requests per time unit increases, and as the records data size increases.

Latencies and throughputs are likely to vary between each individual request. You should therefore take appropriate steps to derive average values and examine the spread of the performance data under comparable conditions (request frequency, size of the data store, request type).

Many tools exist to help you benchmarking HTTP servers[6]. If you don't find anyone that fit your needs, feel free to build your own thanks to existing HTTP client libraries in the language of your choice. Be sure that your test will not exhibit limitation of your testing tools or setup (like a poor testing software, limited hardware or network).

You can work alone or on team of 2 people (or 3 people with the lecturer's agreement). Based on your work, write a short report of 900 words maximum highlighting the results of your investigation and making recommendations to a client as to whether the Google AppEngine is a suitable technology for very large scale applications for fitness applications. Your discussion is not restricted to the scalability of the Google AppEngine. For instance, you might report the limitations on the current programming interfaces, discuss why such limitations can prevent the deployment of real applications in the Google AppEngine, or suggest further improvements to the Google AppEngine platform. Be sure to include the URL at which your GAE application is available in your report.

Please submit your report by email along with your code and experimental data electronically no later than Sunday February 10th, 2019 to `quentin.dufour@inria.fr`.

A Marking Scheme

Correctness / quality of application			5 points (25%)
Richness of the features your application offers			5 points (25%)
Report	Overall presentation (clarity, structure)	1 point (5%)	10 points (50%)
	Design and its motivation	3 point (15%)	
	Quality of evaluation	2 point (10%)	
	Depth of analysis	4 point (20%)	
Total			**20 points (100%)**

Penalty for being late: -20% (-4) for each day late, 0 if more than three days late.

B Potential Issues

Use of JSP in Google App Engine To use JSP in the Google App Engine, you need to include the JDK in your build path instead of JRE. Otherwise you will have a compilation error. If you use Python or Java

[6] `https://github.com/denji/awesome-http-benchmark`

Servlet for this exercise, you should not encounter this problem.

If you use eclipse, you can go to Window -> Preferences -> Java -> Installed JREs, and add the folder that contains your JDK to installed JREs. You can then remove the JRE library from the build path of your project, and then add JDK instead.

Unauthenticated error with Google Cloud Datastore You must create a service account, download its keys (stored in a JSON file) and inject the path of this file in an appropriate environment variable. You will find the full documentation here: `https://cloud.google.com/datastore/docs/reference/libraries`

Application doesn't exist with Google Cloud Datastore If you are sure that your configuration is correct, check on the Google Cloud administration page that Google Datastore is activated for your project.

— the end —