Google App Engine: assignment 2

Deploying web-based distributed applications on a cloud platform

---

Practical arrangements

There will be 3 exercise sessions on Google App Engine:

1. Tuesday 24 November 2020:
   - Assignment 1: Introduction to Google App Engine (GAE) and GAE datastore.
   - Submit your solution on Toledo before Friday 27 November 2020, 19:00.
2. Tuesday 1 December 2020:
   - Assignment 2 *(this assignment)*: Extend Google App Engine application with task queues and worker processes.
   - No submission.
3. Tuesday 8 December 2020:
   - Assignment 2 *(this assignment)*: Extend Google App Engine application with task queues and worker processes (continued).
   - **Submit** *(1) the report* and *(2) the code* on Toledo **before Friday 11 December 2020, 19:00**.

In total, **each** student must submit their solutions to the two GAE assignments to Toledo.

Submission: Before Friday 11 December 2020 at 19:00, **each** student must submit their results to the Toledo website. To do so, enter the base directory of your project and zip your solution using the following command:

```
mvn assembly:single
```

Please do not use a regular zip application to pack your entire project folder, as this would include the entire Google App Engine SDKs (about 40 MB). *Make sure that your report (i.e. it should be placed in the base directory of your project), and additional libraries used in your solution are also included in the zip file.* Submit the zip file to the Toledo website (under Assignments) before Friday, 19:00.

Important:

- This session must be carried out in groups of *two* people. Team up with the same partner you collaborated with in the previous sessions.
- When leaving, make sure your server is no longer running.
- Retain a copy of your work for yourself, so you can review it before the exam.

Good luck!

1 Introduction

Goal: In the previous session, we started with deploying our car rental application on top of Google App Engine. As a first step, we adapted the data entities in order to make them compliant with the Google Cloud Datastore provided by Google App Engine (GAE) [1]. In this session, we focus on more cloud-related requirements: high availability and performance scalability. By using indirect communication techniques offered by GAE, we will redesign the application to be more loosely-coupled in order to address these requirements.

2 The Car Rental Application

2.1 Rental agency

No changes are made to how data is loaded on server startup and persisted in the GAE datastore.

2.2 Extra requirements for cloud services

Cloud computing is an increasingly popular paradigm for software developers and various types of software users because of the *economies of scale*. Simply said, this means that the more users a software application serves, the lower its relative costs are. This also provides new challenges: the parallel use of a cloud application by many people may render the direct use of transactions impractical, e.g. imagine thousands of transactions being queued, making the user wait and letting the system seem very slow (poor user experience).

Using Indirect Communication. An approach to avoid such a delay is to decouple the part of the application that communicates with the client, i.e. the front end, from the part that is involved in the bottleneck activity of that application, i.e. the back end. Thereby, availability and performance requirements become valid for the front end only, allowing the back end to introduce additional delay for its processing.

3 Assignment

First, we setup and test an extended version of the car rental application. If your previous assignment is properly implemented, this should work out of the box; however, remember that you may still need to **implement `confirmQuotes`** in the `CarRentalModel` if you had not already done so. Next, we design and implement a loosely-coupled version of the application, taking the additional requirements for cloud environments into account.

3.1 Loosely-coupled Back and Front End

Recall from the previous GAE session that we had a reservation system that was based on the Java EE implementation. We will now adapt this system step by step to better utilize the scalability opportunities offered by the underlying cloud environment.

Design: Make a design of the distributed car rental agency that uses the Task Queues provided by the App Engine SDK [2]. Therefore, divide your application into a front- and a back-end part in order to loosely couple these parts using indirect communication (as explained above).

Implementation: Implement your design. Please use **Push Queues** [3, 4] (no Pull Queues) and **Instance-Workers** (no Backend processes) or **`DeferredTask`**s [5]. For the implementation of your worker, we provided an additional but empty Servlet (`ds.gae.Worker`) that is bound to the URL `http://localhost:8080/worker`.

Important: Think about how to realize a backchannel to the caller, i.e. how to inform the client whether a quote confirmation has executed successfully or not. The latter case may be even more important to report back, as an omission of positive feedback can be ambiguous and may also suggest that the task at hand has not been processed yet. You may use any (free and non-deprecated) service provided by the GAE SDK to realize this backchannel. Depending on your solution, this may require adding or modifying Java Servlets or JSPs. In case you want to return some information to the car renter on a web page, you may use the given JSP `confirmQuotesReply.jsp` (optional). For your solution, please bear in mind that **backend-processing** (and thereby the delay of the feedback) may be of long duration, e.g. **up to hours and days**.

Hints:

- Overusing indirect communication may also degrade the user experience, as in most cases prompt response is both feasible and more desirable.
- For feedback, do not rely on the same browser (session) that made the reservation.
- If actually invoking the backchannel is infeasible (e.g. a valid recipient is required), you may create a dummy implementation of your backchannel (e.g. writing to the console what would happen).

3.2 Reporting

We would like you to report on your design and the decisions you have made[1], and on your understanding and implementation of distributed concepts. Provide your answers in English, and keep them short, concise and to the point: a few lines is often enough. You are encouraged to use the LaTeX template on Toledo. Don't forget to put your names in the report, and store the report (as one PDF called `report-gae.pdf`) in the main directory of your solution (next to `pom.xml`), such that the final zip-creation process includes these reports.

Compile a report that answers the following questions:

1. Which hosts/systems execute which processes, i.e. how are the remote objects **distributed over hosts**, if run in a real deployment on the App Engine platform (not a lab deployment where everything runs on the same machine)? Clearly outline which parts belong to the front- and back-end. Create a component/deployment diagram to illustrate this: highlight where the client and server are.
2. At which step of the workflow for booking a car reservation (create quote, collect quotes, confirm quotes) would the **indirect communication** between objects or components kick in? Describe the steps that cause the indirect communication, and what happens afterwards.
3. Which kind of **data** is passed **between the front- and back-end** when confirming quotes? Does it make sense to persist data and only pass references to that data?
4. How have you implemented your **backchannel**? What triggers the backchannel to be used, how does the client use the backchannel and what information is sent?
5. How does your solution to **indirect communication** improve **scalability**?
6. Workers in GAE's Task Queues are by default set to run in **parallel**. While parallelism is usually a desirable property of a cloud service, as it enables scalability and thus faster overall processing, it may also endanger the application's state consistency. Assume a scenario in which two different clients try to confirm a couple of tentative reservations, i.e. their quotes are queued to be processed by the back end. Both include a tentative reservation for the last available car of a certain car type, so that, assuming correct behaviour of the car rental application, it should fail to confirm the quotes for one of them.
   (a) Is there a scenario, unrelated to your specific solution, in which the code to confirm the quotes is executed multiple times in parallel, resulting in a positive confirmation to both clients' quotes?
   (b) If so, can you name and illustrate one (or more) possibilities to prevent this bogus behaviour?
   (c) In case your solution to the previous question limits parallelism, would a different design of the indirect communication channels help to increase parallelism? For this question, you may

[1] With design decisions we mean design choices, possible alternatives and trade-offs.

assume that a client will have quotes belonging to one car rental company only.

7. How does using a **NoSQL** database affect **scalability** and **availability**?
8. How have you structured your **data model** (i.e. the entities and their relationships)? Compared to a relational database, what sort of **query limitations** have you faced when using the Cloud Datastore?
9. What is the most critical difference between `confirmQuote` and `confirmQuotes` from the viewpoint of **transaction management**? Explain an alternative to the solution that you provide to achieve **all-or-nothing semantics** of `confirmQuotes` which is essentially different from the viewpoint of transaction management. Given the underlying distributed storage layer, what are the implications of each solution for performance and data consistency?

4 Getting up and running

4.1 Preliminaries

Extend your car rental application from the last session:

1. Download `ds_gae2_src.zip` from Toledo. In this ZIP file, we provide additional Servlets and JSPs to enable car renters to create and confirm quotes in the different car rental companies. These additional files use the `ds.gae.CarRentalModel` class that you implemented in the previous session.
2. Unzip `ds_gae2_src.zip` in your project of the previous session. For that, first close the IDE, unzip the files, and reopen the project and immediately clean/build your project.
3. Before continuing, ensure that the function `confirmQuotes(List<Quote>)` in `ds.gae.CarRentalModel` is implemented using the datastore and confirms either all quotes or none (when one of the cars is not available anymore). For your implementation, you do not have to provide strong consistency *guarantees*, and may assume the absence of system/network failures.
4. Run the application by deploying your GAE project (see the first GAE assignment) and open `http://localhost:8080` in your browser. **Please make sure that the application works properly before going on to the next task**, e.g. create at least a quote at each car rental company and request to confirm all quotes at once. The tab `Bookings` shows all reservations that succeeded. The last tab corresponds to the persistence test that was used during the previous session. To check its reservations, login as "test.user@kuleuven.be".

4.2 Practical Information

- Data can be passed from a Servlet to a JSP by using a HTTP session: attributes that are set by the Servlet can be read by the JSP. For further information, we refer to the previous assignment.
- To inspect the data in the datastore or the tasks that are currently waiting in the Task Queue, start the application and go to `http://localhost:8080/_ah/admin`.
- For creating diagrams you can use Visual Paradigm in the **computer labs**:

  ```
  /localhost/packages/visual_paradigm/bin/Visual_Paradigm
  ```

  You can also install Visual Paradigm on your own PC: `https://ap.visual-paradigm.com/kuleuven`.

Important:

- Check if you have enough storage space in your home directory via `quota` and `du | sort -n`. **Lack of storage space results in strange errors!**
- If Eclipse does not shut down cleanly, there may be lingering server processes. You may get `java.io.IOException: Failed to bind to /0.0.0.0:8080`. Use a task manager to kill old `java` processes running `com.google.appengine.tools.development.DevAppServerMain`.

References

[1] Google, Inc. *Google App Engine*. https://cloud.google.com/appengine/docs/java/.

[2] Google Inc. *Task Queue Java API*. https://cloud.google.com/appengine/docs/standard/java/taskqueue/.

[3] Google Inc. *Using Push Queues*. https://cloud.google.com/appengine/docs/standard/java/taskqueue/push/.

[4] Google Inc. *Java Task Queue Configuration*. https://cloud.google.com/appengine/docs/standard/java/taskqueue/push/creating-push-queues.

[5] Google Inc. *DeferredTask Java Interface (JavaDoc)* . https://cloud.google.com/appengine/docs/standard/java/javadoc/com/google/appengine/api/taskqueue/DeferredTask.