Distributed Systems: Google App Engine

Pieter Robberechts, Xavier Goás Aguililla

Friday, December 12, 2014

GAE Exercise 3.2

The creation and collection of quotes isn't a lot of work. They are not stored in the database and can be easily put in a collection. In addition, the result of creating a quote should be immediately visible to the user. The most work-intensive part is checking whether a list of quotes can be persisted as reservations in the database. This task is performed by a background worker to avoid delay in the frontend. To do so we serialize the list of quotes and send it to the default push queue. Next, the backend worker will process the tasks in this queue and notify the user of success or failure. We store this notification in the database. The user can view his notifications at any time on the *notifications* page.

Alternatively we could have persisted all quotes that need to be confirmed, and then pass references to the back end. However these quotes are no longer needed after they're confirmed (or cancelled) and would waste storage space.

GAE Exercise 3.3

Google App Engine allows us to process several tasks simultaneously. So, if a couple of tentative reservations are waiting in the queue to be processed, the code to confirm the quotes could be executed in parallel, possibly resulting in an illegal state if we do not take precautions.

To avoid this scenario we implemented the confirmation process as a transaction. It was a bit tricky to implement this as GAE requires Datastore operations in a transaction to operate on entities in the same entity group if the transaction is a single group transaction, or on entities in a maximum of five entity groups if the transaction is a cross-group (XG) transaction.

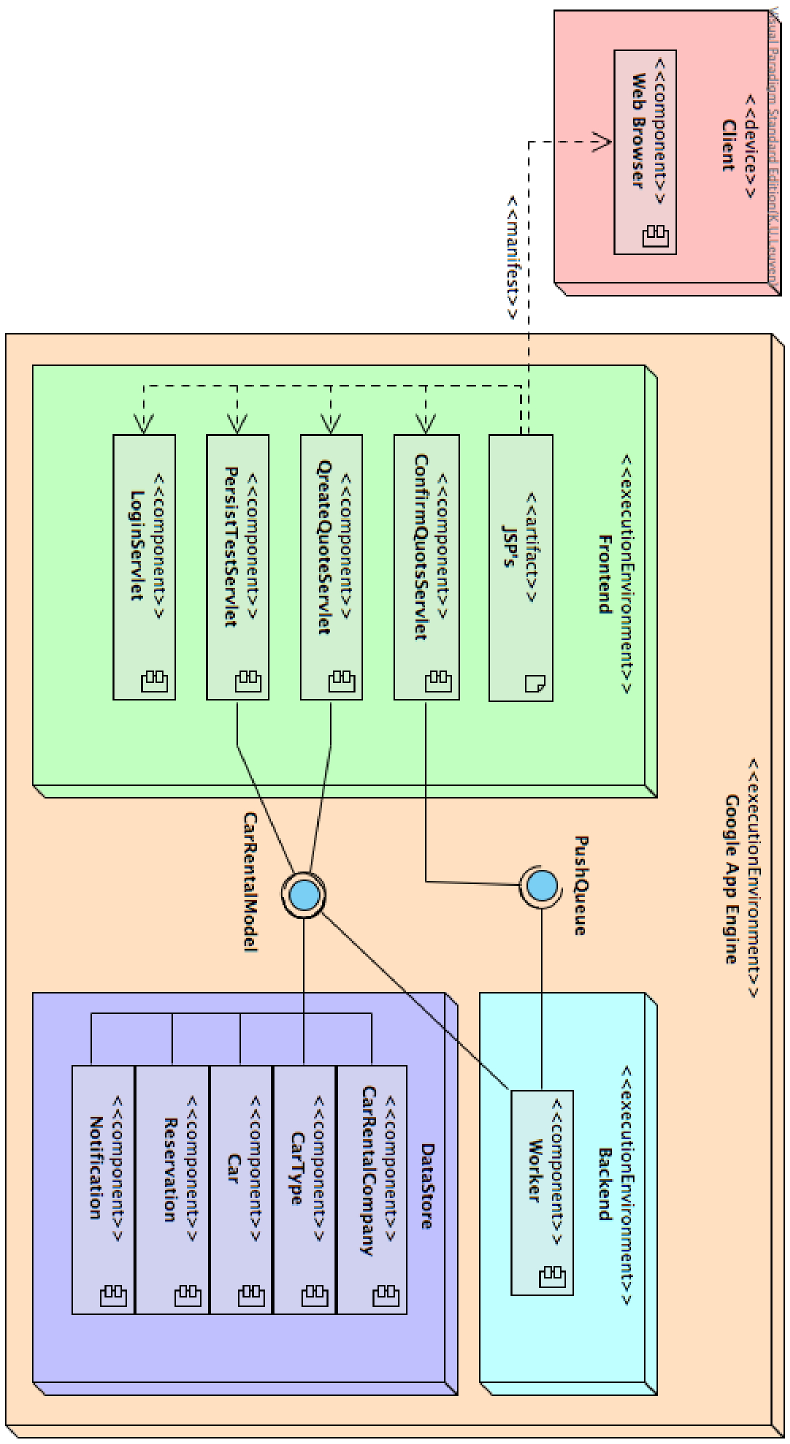
This isn't really a problem as we only have two entity groups (Dockx and Hertz), but to make our app scalable we use a different transaction for each set of reservations at a distinct company.

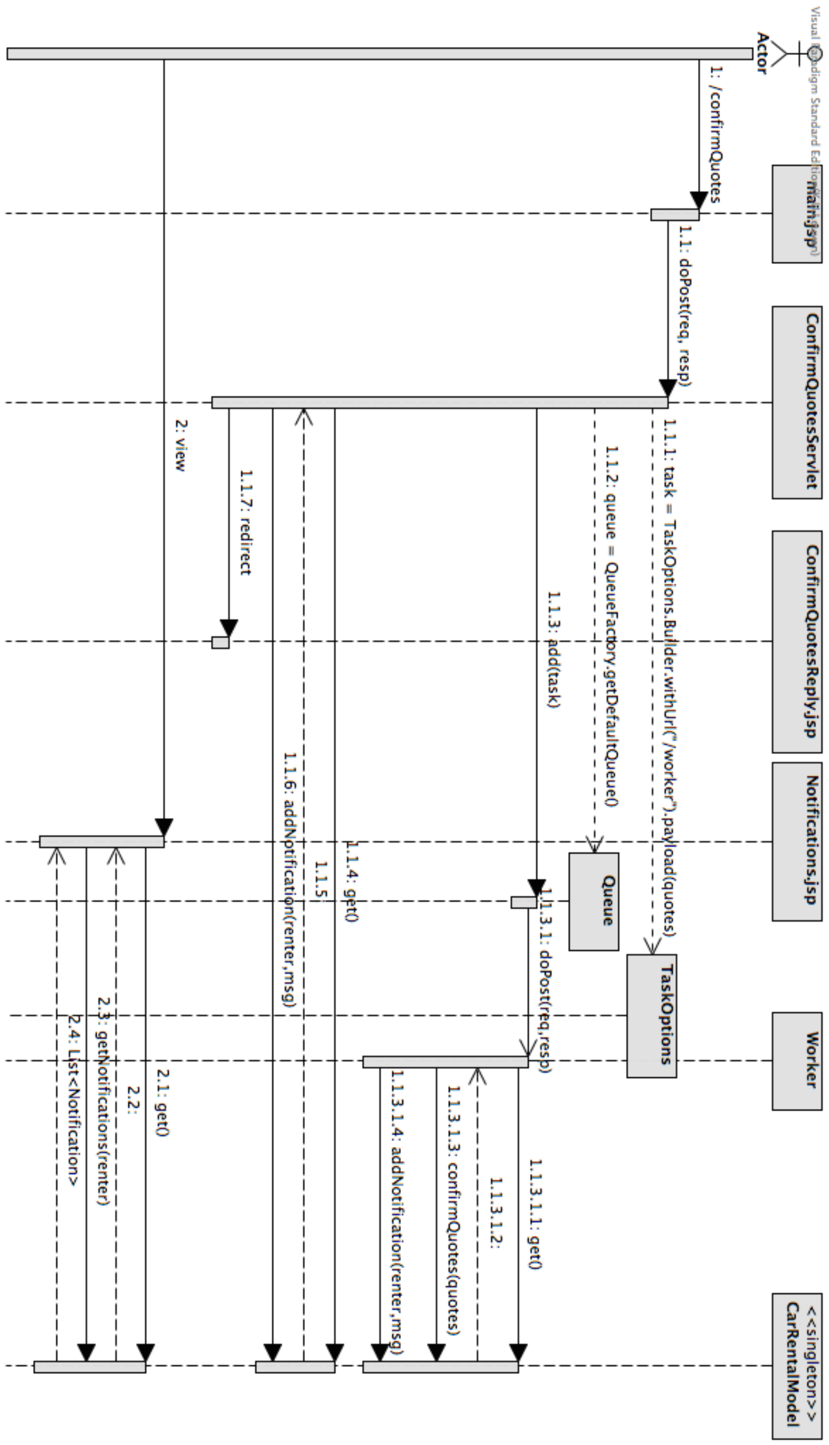
This limits parellellism in the sense that two sets of reservations for different companies can't be handled in parallel, only sequentially, even though they can't conflict; any set of quotes is regarded as possibly conflicting with another.

To improve this, we could, for instance, exploit task queues and have separate workers confirming quotes for each company, at the cost of creating extra tasks in the queue. We opted to keep our solution a little more simple.

Application URL

Our application is deployed at `https://dockxandhertz.appspot.com`.


Visual Paradigm Standard Edition(K.U.Leuven)
<<device>>
Client
<<component>>
Web Browser
<<manifest>>
<<executionEnvironment>>
Google App Engine
<<executionEnvironment>>
Frontend
<<artifact>>
JSP's
<<component>>
ConfirmQuotsServlet
<<component>>
QreateQuoteServlet
<<component>>
PersistTestServlet
<<component>>
LoginServlet
PushQueue
CarRentalModel
<<executionEnvironment>>
Backend
<<component>>
Worker
DataStore
<<component>>
CarRentalCompany
<<component>>
CarType
<<component>>
Car
<<component>>
Reservation
<<component>>
Notification


Visual Paradigm Standard Edition
Actor
main.jsp
ConfirmQuotesServlet
ConfirmQuotesReply.jsp
Notifications.jsp
Queue
TaskOptions
Worker
<<singleton>> CarRentalModel
1: /confirmQuotes
1.1: doPost(req, resp)
1.1.1: task = TaskOptions.Builder.withUrl("/worker").payload(quotes)
1.1.2: queue = QueueFactory.getDefaultQueue()
1.1.3: add(task)
1.1.3.1: doPost(req,resp)
1.1.3.1.1: get()
1.1.3.1.2:
1.1.3.1.3: confirmQuotes(quotes)
1.1.3.1.4: addNotification(renter,msg)
1.1.4: get()
1.1.5:
1.1.6: addNotification(renter,msg)
1.1.7: redirect
2: view
2.1: get()
2.2:
2.3: getNotifications(renter)
2.4: List<Notification>