Java

+1 39

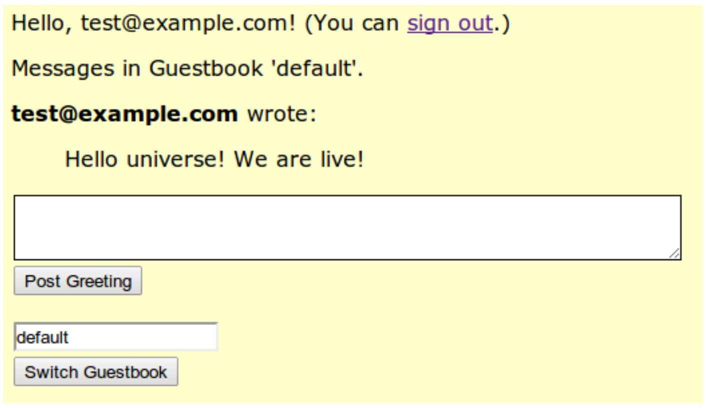
Write Feedback

In this part of the tutorial, we'll extend the simple application created earlier by adding UI that allows the user to POST greetings and display previously posted greetings. To support the UI, we'll add a new servlet that handles the POST interaction with the database to store the data.

The database we are using is App Engine Datastore, which is a NoSQL, schema-less database available to your app without any further sign-up or activation. Because Datastore is schema-less, you don't need to define objects in the database in advance before you start writing to Datastore. You simply import the required Datastore classes via your `import` statements and write your data. (For a complete description of this powerful storage system, visit the App Engine Datastore pages.)

> **Note:** This tutorial uses the low-level Datastore API, for the sake of simplicity. App Engine Datastore also includes implementations of the Java Data Objects (JDO) and Java Persistence API (JPA) interfaces, if you wish to use those. You could alternatively use Objectify or Slim3.

This is what the app will look like after we add the UI and support for Datastore:


Hello, test@example.com! (You can sign out.)

Messages in Guestbook 'default'.

test@example.com wrote:

Hello universe! We are live!

We'll make these changes to our existing application :

- Edit the JSP page to allow users to post greetings to the datastore and to display all the greetings currently stored.
- Create a new servlet named SignGuestbookServlet.java that handles the interactions with Datastore.
- Add required entries in `web.xml` so requests can be routed to the new servlet.

Note: This part of the tutorial is the completed, or "final", version of the app we'll be building, including the datastore logic and UI. So, if you look in the GitHub **View Project** links we provide, be sure to go to **final**, not to **phase1**, within the project.

To add the new UI pieces:

1. In `guestbook/src/main/webapp`, open `guestbook.jsp` in an editor to add the following imports to the imports section:

   View File View Project Download P

```
<%@ page import="com.google.appengine.api.datastore.DatastoreService" %>
<%@ page import="com.google.appengine.api.datastore.DatastoreServiceFactory" %>
<%@ page import="com.google.appengine.api.datastore.Entity" %>
<%@ page import="com.google.appengine.api.datastore.FetchOptions" %>
<%@ page import="com.google.appengine.api.datastore.Key" %>
<%@ page import="com.google.appengine.api.datastore.KeyFactory" %>
<%@ page import="com.google.appengine.api.datastore.Query" %>
```

2. Just above the line `<form action="/guestbook.jsp" method="get">` add the following code:

View File View Project Download P

```
<%
    DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
    Key guestbookKey = KeyFactory.createKey("Guestbook", guestbookName);
    // Run an ancestor query to ensure we see the most up-to-date
    // view of the Greetings belonging to the selected Guestbook.
    Query query = new Query("Greeting", guestbookKey).addSort("date", Query.SortDirection.DES
    List<Entity> greetings = datastore.prepare(query).asList(FetchOptions.Builder.withLimit(
    if (greetings.isEmpty()) {
%>
<p>Guestbook '${fn:escapeXml(guestbookName)}' has no messages.</p>
<%
    } else {
%>
<p>Messages in Guestbook '${fn:escapeXml(guestbookName)}'.</p>
<%
        for (Entity greeting : greetings) {
            pageContext.setAttribute("greeting_content",
                    greeting.getProperty("content"));
            String author;
            if (greeting.getProperty("author_email") == null) {
                author = "An anonymous person";
            } else {
                author = (String)greeting.getProperty("author_email");
                String author_id = (String)greeting.getProperty("author_id");
                if (user != null && user.getUserId().equals(author_id)) {
                    author += " (You)";
                }
            }
            pageContext.setAttribute("greeting_user", author);
%>
<p><b>${fn:escapeXml(greeting_user)}</b> wrote:</p>
<blockquote>${fn:escapeXml(greeting_content)}</blockquote>
<%
        }
    }
%>

<form action="/sign" method="post">
    <div><textarea name="content" rows="3" cols="60"></textarea></div>
    <div><input type="submit" value="Post Greeting"/></div>
    <input type="hidden" name="guestbookName" value="${fn:escapeXml(guestbookName)}"/>
</form>
```

The query-related code results in a query when the page is loaded; Datastore is searched for all greetings currently stored for this app in Datastore, and lists them in the UI.

The form-related part of this code sends a POST request containing a new greeting from the user. That POST is handled by the post handler servlet `SignGuestbookServlet.java`, which you will create in the next procedure.

SignGuestbookServlet.java

To create the servlet:

1. In `guestbook/src/main/java/com/example/guestbook`, create a file named `SignGuestbookServlet.java`.
2. Add the following contents to the file:

View File View Project Download P

```java
package com.example.guestbook;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import java.io.IOException;
import java.util.Date;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SignGuestbookServlet extends HttpServlet {
  @Override
  public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws IOException {
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();

    String guestbookName = req.getParameter("guestbookName");
    Key guestbookKey = KeyFactory.createKey("Guestbook", guestbookName);
    String content = req.getParameter("content");
    Date date = new Date();
    Entity greeting = new Entity("Greeting", guestbookKey);
    if (user != null) {
      greeting.setProperty("author_id", user.getUserId());
      greeting.setProperty("author_email", user.getEmail());
    }
    greeting.setProperty("date", date);
    greeting.setProperty("content", content);

    DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
    datastore.put(greeting);

    resp.sendRedirect("/guestbook.jsp?guestbookName=" + guestbookName);
  }
}
```

This servlet is the POST handler for the greetings posted by users. It takes the the greeting (`content`) from the incoming request and stores it in Datastore as an entity called `Greeting`, and stores properties along with the Greeting, such as the e-mail address and the id of the user who posted the greeting and the date it was posted. (For more information about entities in App Engine, see Entities, Properties, and Keys).

3. Open `guestbook/src/main/webapp/WEB-INF/web.xml` and ensure the new servlets have URLs mapped. The final version should look like this:

View File View Project Download P

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/java
    <servlet>
        <servlet-name>sign</servlet-name>
        <servlet-class>com.example.guestbook.SignGuestbookServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>guestbook</servlet-name>
        <servlet-class>com.example.guestbook.GuestbookServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sign</servlet-name>
        <url-pattern>/sign</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>guestbook</servlet-name>
        <url-pattern>/guestbook</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>guestbook.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

4. You are ready to build and test the app locally on the development server, which is described next.

To build and test the app:

1. Change directory to `guestbook`, and invoke the command:

   ```
   mvn clean install
   ```

   Wait for the build to complete.

2. Run the app in the development server by invoking this command:

   ```
   mvn appengine:devserver
   ```

   Wait for the success message, which looks something like this:

   ```
   INFO] INFO: The admin console is running at http://localhost:8080/_ah/admin
   [INFO] Aug 18, 2014 5:35:04 PM com.google.appengine.tools.development.DevAppServerImpl
   [INFO] INFO: Dev App Server is now running
   ```

3. In a browser running on the same machine as your terminal window, visit `localhost:8080` to access the app. If you aren't already signed-in, click **Sign in**, supply an email, and click **Log In**.

4. Supply a text message in the textbox and click **Post Greeting**. Observe that the greeting is now displayed in the greetings list under your email name.

When you run the development server and exercise your app, indexes required for operation in production App Engine are automatically generated in `guestbook/target/guestbook-1.0-SNAPSHOT/WEB-INF/appengine-generated/datastore-indexes-auto.xml`. You'll also notice the file `local_db.bin`at that same location: this is local storage for the development server to persist your app data between development server sessions. The file `datastore-indexes-auto.xml` is automatically uploaded along with your app; `local_db.bin` is not uploaded.

Important: Be aware that running `mvn clean install` clears the `datastore-indexes-auto.xml` file; if you run it on your app prior to uploading to production App Engine, you won't get required indexes and you'll experience a runtime error.

For complete information about indexes, see Datastore Indexes.

Uploading Your Application >>