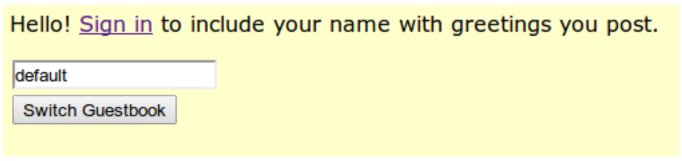
Adding Application Code and UI

In this part of the tutorial, we'll create an app that integrates with Google Accounts so users can sign in using their Google accounts. In App Engine, the integration with Google accounts is achieved via the App Engine Users service. We'll use this to personalize our application's greeting.

This is what the app will look like, in this part of the tutorial (we'll add more later):


Hello! Sign in to include your name with greetings you post.
default
Switch Guestbook

This application consists of these main logical "parts":

- A JSP page that the user interacts with to make requests to the app.
- A servlet named GuestbookServlet.java that prompts the user to log in and then displays a personalized greeting.

> **Note:** This part of the tutorial is an "intermediate" version of the app we'll be building. It doesn't have the datastore logic and UI yet. So, if you look in the GitHub **View Project** links we provide, be sure to go to **phase1**, not to **final**, within the project.

Creating the UI using JSP

To create the UI:

1. In `guestbook/src/main/webapp`, create a file named `guestbook.jsp` with the following contents:

   View File View Project Download P

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.google.appengine.api.users.User" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>
<%@ page import="java.util.List" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<html>
<head>
    <link type="text/css" rel="stylesheet" href="/stylesheets/main.css"/>
</head>

<body>

<%
    String guestbookName = request.getParameter("guestbookName");
    if (guestbookName == null) {
        guestbookName = "default";
    }
    pageContext.setAttribute("guestbookName", guestbookName);
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
        pageContext.setAttribute("user", user);
%>

<p>Hello, ${fn:escapeXml(user.nickname)}! (You can
    <a href="<%= userService.createLogoutURL(request.getRequestURI()) %>">sign out</a>.)</p>
<%
} else {
%>
<p>Hello!
    <a href="<%= userService.createLoginURL(request.getRequestURI()) %>">Sign in</a>
    to include your name with greetings you post.</p>
<%
    }
%>


<form action="/guestbook.jsp" method="get">
    <div><input type="text" name="guestbookName" value="${fn:escapeXml(guestbookName)}"/></di
    <div><input type="submit" value="Switch Guestbook"/></div>
</form>

</body>
</html>
```

   Notice the imports for the App Engine `Users` service. Also, by default, any file in `webapp/` or in a subdirectory other than `WEB-INF/` that has the s file suffix `.jsp` is automatically mapped to a URL path consisting of the path to the `.jsp` file, including the filename. This JSP will be mapped automatically to the URL `/guestbook.jsp`.

2. In `guestbook/src/main/webapp`, create a directory named `stylesheets`, and create a file named `main.css` with the following contents:

   View File View Project Download P

```
body {
    font-family: Verdana, Helvetica, sans-serif;
    background-color: #FFFFCC;
}
```

3. Proceed to `web.xml` configuration, described next.

Configuring `web.xml`

To configure the `web.xml` file:

1. In `guestbook/src/main/webapp/WEB-INF`, open `web.xml` in a text editor, and replace the contents of the file with the following:

View File View Project Download P

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app PUBLIC
 "-//Oracle Corporation//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>guestbook</servlet-name>
        <servlet-class>com.example.guestbook.GuestbookServlet</servlet-class>
    </servlet>
    <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>
```

This configuration maps the servlet to its serving location and specifies the JSP file you created to be the application home page. For more information about the `web.xml` file and how to use it, see the Deployment Descriptor page.

2. Proceed to servlet creation, described next.

Creating the servlet `GuestbookServlet.java`

App Engine Java applications use the Java Servlet API to interact with the web server. An HTTP servlet is an application class that can process and respond to web requests. This class extends either the javax.servlet.GenericServlet class or the javax.servlet.http.HttpServlet class.

To create the servlet:

1. In `guestbook/src/main/java`, create the subdirectories `com/example/guestbook` by invoking the following command (in a Linux or Mac OSX terminal window):

```
mkdir -p com/example/guestbook
```

2. In `guestbook/src/main/java/com/example/guestbook`, create a file named `GuestbookServlet.java`.

3. Add the following contents to the file:

View File View Project Download P

```
package com.example.guestbook;

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.Properties;

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

public class GuestbookServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws IOException {
    if (req.getParameter("testing") == null) {
      resp.setContentType("text/plain");
      resp.getWriter().println("Hello, this is a testing servlet. \n\n");
      Properties p = System.getProperties();
      p.list(resp.getWriter());

    } else {
      UserService userService = UserServiceFactory.getUserService();
      User currentUser = userService.getCurrentUser();

      if (currentUser != null) {
        resp.setContentType("text/plain");
        resp.getWriter().println("Hello, " + currentUser.getNickname());
      } else {
        resp.sendRedirect(userService.createLoginURL(req.getRequestURI()));
      }
    }
  }
}
```

The servlet checks whether the user is logged on. If the user is logged on, the servlet displays a personalized greeting; otherwise the user is redirected to the logon page.

> **Note:** The development server knows how to simulate the Google Accounts sign-in facility. So when this app runs on your local machine, the redirect goes to a page where you can enter any email address to simulate an account sign-in. However, when the app runs on production App Engine, the redirect goes to the actual Google Accounts screen.

4. You project should look like this:

```
guestbook/
├── eclipse-launch-profiles
│   ├── DevAppServer.launch
│   └── UpdateApplication.launch
├── nbactions.xml
├── pom.xml
├── README.md
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── example
    │   │           └── guestbook
    │   │               └── GuestbookServlet.java
    │   └── webapp
    │       ├── guestbook.jsp
    │       ├── stylesheets
    │       │   └── main.css
    │       └── WEB-INF
    │           ├── appengine-web.xml
    │           ├── logging.properties
    │           └── web.xml
    └── test
```

Your app is now ready to build and test the app locally on the development server, which is described next.

Building and testing the app

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 on your local machine by invoking this command from `/guestbook`:

   ```
   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 that is running on the same machine as your terminal window, visit `localhost:8080` to access the app on your local machine. If prompted, click **Sign in**.

   > **Note:** If you get a runtime error when you first visit `localhost:8080`, referring to a restricted class, for example, `java.lang.NoClassDefFoundError: java.nio.charset.StandardCharsets is a restricted class.`, check the settings for `guestbook/pom.xml`. The App Engine version must be set to the most recent App Engine SDK version: 1.9.17. If you get a `403` error, check the `web.xml` to make sure the file is configured correctly as described above under "Configuring `web.xml`".

4. In the login form supply an email or accept the `test@example.com` dummy email and click **Log In**. (When run locally, there is no validity check.)

5. Observe that the greeting now displays your email address.

6. You have successfully created a simple Java App Engine app. You are ready to do something a bit more useful next, adding the ability to accept and store user posts in a database.

Storing User-Supplied Data in Datastore >>