



# Codelab: Authenticating Users

*Updated: March 20, 2015*

In the previous codelab, you downloaded and deployed the Conference Central application. All pages are currently accessible to everybody -- anybody can view them. In this exercise, you will restrict access to several pages so that only logged in users can access those pages. You will also restrict access to the developers page to users who are registered developers for the application.

## Contents

[What You Will Learn](#)

[What You Will Do](#)

[Get Ready](#)

[Configure Access Controls for the Pages in the Application](#)

[Java -- Configure the Access Control for the Relevant Pages](#)

[Python -- Configure Access for the Relevant Pages](#)

[Test Application Locally](#)

[Run the Application on the Development Server](#)

[Java Instructions](#)

[Python Instructions](#)

[Testing the App](#)

[Use the UserService API to Personalize the Web Pages](#)

[Update the Navigation Bar to Detect if the User is Logged In](#)

[Java -- Detect if the User is Logged In](#)

[Python -- Detect if the User is Logged In](#)

[Optional Exercise: Update the Home Page to Display Information about the User](#)

[Java -- Update the Home Page to show info about the user](#)

[Python -- Update the Home Page to Show info about the User](#)

[Optional: Deploy the Application to App Engine](#)

[Summary](#)

## What You Will Learn

The goal of this exercise is to help you learn how to set up access controls for pages in your application.

You will also learn how to use the UserService API to find out if the user is logged in, to retrieve information about the user if they are logged in, and to allow users to login and logout.



## What You Will Do

In this exercise, you will:

1. Work with the Conference Central App that you setup in the previous exercise and configure **web.xml** for Java or **app.yaml** for Python to specify who can view which pages
  - Home Page -- accessible by everyone
  - List Conference Page -- accessible by everyone
  - Schedule Conference Page -- accessible by anyone who has logged in with a Google Account
  - User Profile Page -- accessible by anyone who has logged in with a Google Account
  - Developer Page -- accessible only by logged in users who are also registered developers for the application
2. Write code that invokes the User Service to detect if the user is logged in, and if so, to personalize the page to the user.
3. Write code that uses the User Service to allow the user to login and logout.

## Get Ready

For this new exercise, do **one** of the following:

- Continue working on your existing Conference Central application, which you have in the previous exercise.

**OR**

- Download and install the zip containing the Conference Central application:
  - Java zip: [ConferenceApp 1 Java completed.zip](#)
  - Python zip: [ConferenceApp 1 Python completed.zip](#)

You can also clone the projects directly from GitHub:

```
git clone
```

<https://github.com/GoogleCloudPlatformTraining/cp300-gae-conference-app-complete-1-java.git>

**OR**

```
git clone
```

<https://github.com/GoogleCloudPlatformTraining/cp300-gae-conference-app-complete-1-python.git>

If you start with one of these zip files remember to reset the **Application ID**.



## Configure Access Controls for the Pages in the Application

Your application has the following pages:

Page	Access Criteria
Home Page	open to everyone
List Conferences Page	open to everyone
Schedule Conference Page	restricted to users who are logged in with a Google account
User Profile Page	restricted to users who are logged in with a Google account
Developer Page	restricted to "administrators," (that is, developers listed in the Permissions page in the App Engine Admin Console)

In this exercise, you will configure the access requirements for each of the pages.

By default, all pages are accessible by everybody, so there is nothing extra you need to do to configure the Home Page and the List Conferences Page to be viewable by everyone. However, to restrict access to the Schedule Conference Page, the User Profile Page, and the Developers Page, you need to configure access by editing the application's configuration file.

To set up access controls for pages in your application, all you have to do is:

- specify `<security-constraint>` directives in `web.xml` for Java applications
- specify the login setting for the page handler for Python applications

If the appropriate access controls are set up, then if a user tries to go to a page that has restricted access, the user will be required to login. You do not need to write the code to handle the login interaction -- it happens automatically.

### ➡ Java -- Configure the Access Control for the Relevant Pages

#### 1. Configure Access Control for the User Profile Page.

In `web.xml`, add a `<security-constraint>` directive for the User Profile Page JSP. Add this after the configuration for `userprofile.jsp`.

Recall that the `jsp` for the User Profile Page is mapped to the `/userprofile` URI.



Specify the `<role-name>` as `*`, which restricts access to users who logged in with a Google Account.

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/userprofile</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>*</role-name>
  </auth-constraint>
</security-constraint>
```

### 2. Configure the access control for the Schedule Conference Page (/scheduleconference).

In `web.xml`, configure the access for the Schedule Conference Page just as you did for the User Profile Page, to ensure that a user is logged in before they can access the page. You can simply add another `<url-pattern>` to the existing `<web-resource-collection>` section as follows:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/userprofile</url-pattern>
    <url-pattern>/scheduleconference</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>*</role-name>
  </auth-constraint>
</security-constraint>
```

### 3. Configure the access control for the developer page

In `web.xml`, add a `<security-constraint>` directive for `/developer`.

Specify the `<role-name>` as **admin**, which indicates that access to any urls that match the `<url-pattern>` is restricted to users who logged in with a Google Account and who are also registered developers for the application.

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/developer</url-pattern>
  </web-resource-collection>
  <auth-constraint>
```



```
<role-name>admin</role-name>
</auth-constraint>
</security-constraint>
```

### ➡ Python -- Configure Access for the Relevant Pages

1. Configure Access Control for the User Profile Page to require users to be logged in to view the page.

In app.yaml, add a login setting to the URL script handler for /userprofile.

```
- url: /userprofile
  script: main.app
  login: required
```

2. Configure the access control for the Schedule Conference Page (/scheduleconference).

In app.yaml, configure the access for the Schedule Conference Page just as you did for the User Profile Page, to ensure that a user is logged in before they can access the page.

3. Configure the access control for the developer page (/developer) to restrict access to administrators.

```
- url: /developer
  script: main.app
  login: admin
```

Save the file.

**Note:** For Python, make sure that the mapping to:

```
- url: /.* script: main.app
```

is the last handler mapping in app.yaml, otherwise the access control does not work.

## Test Application Locally

Now it's time to deploy and test the application.



In a development (local) deployment of the application, you can simulate logging in as a registered developer for the application by checking the "Sign as Administrator" checkbox in the login dialog box.

## Run the Application on the Development Server

### Java Instructions

To build and run the Java App Engine project, execute the following Maven commands from the command line / terminal. These commands should be executed from the

**cp300-gae-conference-app-complete-1-java** folder i.e. where your pom.xml file is present:

1. To build the project

```
mvn package
```

2. To run the application in the local devserver :

```
mvn appengine:devserver
```

### Python Instructions

To run the Python App, go to **cp300-gae-conference-app-complete1-python** folder and give the followign command at the command line / terminal:

```
dev_appserver.py .
```

### Testing the App

1. To test the application, open a new **Incognito Window** in Chrome, and go to to the home page at `http://localhost:portnumber/`.

2. Try going to the Schedule Conference Page (**Create Conference**) and the User Profile Page.

The first time you attempt to go to any of those pages, you will be required to login.

## Use the UserService API to Personalize the Web Pages

So far you have created several pages, and specified access controls for some of the pages. You have seen that you can require users to login to access pages simply by specifying login constraints in the application's configuration file. Next you will use the User Service API to retrieve information about the current user.

App Engine's UserService provides an API for:

- Checking if the current user is logged in
- Generating URLs to allow users to sign in and out
- Retrieving the logged in user's email address



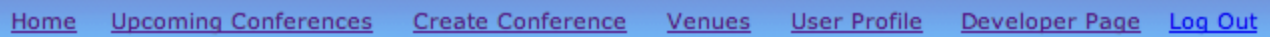
Next, you will:

- Use the UserService API to personalize the pages to the logged in user
- Write code to allow the user to login and logout

## Update the Navigation Bar to Detect if the User is Logged In

If the user is logged in, the navigation bar should display a link to logout. If the user is not logged in, the navigation bar should display a link to login.

If, and only if, the user is logged in as an admin (registered developer for the application), the navigation bar should display the link to the Developer Page, as shown here:



### ➡ Java -- Detect if the User is Logged In

You will change the code to display the link to the Developer Page only if the user is logged in as an administrator.

#### 1. Open header.jsp.

Notice the following import statements at the top of the file:

```
<%@ page import="java.security.Principal" %>
<%@ page import="com.google.appengine.api.users.UserService" %>
<%@ page import="com.google.appengine.api.users.UserServiceFactory" %>
```

#### 2. Take a look at the code for header.jsp. It includes several TODOs. Please paste the code given below:

```
<%
    // Get a handle to the UserService
    UserService userService = UserServiceFactory.getUserService();

    // Get the logged in user
    String requestUri = request.getRequestURI();
    Principal userPrincipal = request.getUserPrincipal();

    // If the user is NOT logged in,
    // display a "login" link, with css style class="nav-item"
    if (userPrincipal == null) {
        String loginLink = userService.createLoginURL(requestUri); %>
        <span="nav-item"> <a href="<%= loginLink %>">Sign In</a></span></p> <%
    }
}
```



```
// If the user is logged in as an ADMIN,  
// display a link to the developer page (/developer)  
else {  
    if (userService.isUserAdmin()) { %>  
        <span class="nav-item"><a href="/developer">Developer  
Page</a></span> <%  
    }  
  
    // If the user is logged in,  
    // display a "sign out" link, with css style class="nav-item"  
    String logoutLink = userService.createLogoutURL(requestUri);  
    %> <span="nav-item"><a href="<%= logoutLink %>">Log Out</a></span></p>  
<%  
    }  
    %>
```

### 3. Run the application locally.

The header bar should show a link for logging in or logging out as appropriate.

4. If you don't see a link to the Developer Page in the navigation bar, logout then login again, this time checking the **Sign in as Administrator** checkbox in the login dialog box.

Not logged in

Email:

☒ Sign in as Administrator

In the development server, you simulate logging in as a developer for the application by signing in as an Administrator.

5. Go to the Developer Page. (The link in the navigation bar is only visible if you are logged in as an Administrator).

**Note:** The Developer page has some predefined content, which you will use in later exercises.

6. While you are viewing one of the Create Conference Page, the User Profile Page, or the Developer Page, logout.





You will be immediately requested to login again, because you are viewing a page that requires you to be logged in.

### ➡ Python -- Detect if the User is Logged In

1. Edit `base.template` and review the links to the Developer Page, login and logout url after the link to the User Profile Page.

Each link is enclosed in a `<span class="nav-item"> </span>` tags

For an admin (`is_admin`) the anchor is `"/developer"` and the text is Developer Page

If the user exists the anchor is `{{ logout_url }}` otherwise it is `{{ login_url }}`

The if statements use simple jinja2 syntax for the if statement `{% if condition %}`, `{% else %}`, and `{% endif %}`.

The code contains template variables. You define a dictionary that contains the values and pass in the dictionary as an argument to the `jinja2.Template.render()` function.

2. Since `base` template is extended by all other pages, you will create a corresponding base handler, `_BaseHandler`, that will be extended by other handlers in `main.py`.

The basehandler will use the `google.appengine.api.users` package to get the basic information about the logged in user and to get the login and logout url. This handler gets this information in the `initialize()` method and also creates a jinja2 template dictionary for other sub-classed handlers to use.

The basic structure of the `_BaseHandler` is already in `main.py`. All that remains is to fill in some of the blanks.

1. In `main.py`, first make sure `users` is imported from `google.appengine.api`:  
(HINT: Look for: `from google.appengine.api import users`)
2. Uncomment the `_BaseHandler` class (remove the triple quotes around it).
3. In `initialize` in `_BaseHandler` do the following
  1. Use `users.get_current_user()` to initialize `self.user`
  2. If the user exists:
    - Use `self.user` as the value for `'user'` in `self.template_values`
    - Use `users.is_current_user_admin()` as value for `'is_admin'` in `self.template_values`
    - Use `users.create_logout_url('/')` to initialize `'logout_url'` in `self.template_values`
  - If the user does not exist:
    - Use `users.create_login_url(self.request.url)` to initialize `'login_url'`



All of the code for class `_BaseHandler` can be found the file `class_BaseHandler.txt` in `Code_Samples`.

3. Change all the other classes in `main.py` to extend from `_BaseHandler` instead of `webapp2.RequestHandler`.

4. In class `HomePage`, edit method `get(self)` to specify the template dictionary, `self.template_values`, as a parameter for the `template.render()` function so the `homepage.template` can be rendered with the correct values.

The complete class `Homepage` code is available in the file `class_HomePage.txt` in `Code_Samples`.

5. Run the application locally and go to the home page.

The header bar should show a link for logging in or logging out as appropriate.

6. If you don't see a link to the Developer Page in the navigation bar, logout then login again, this time checking the **Sign in as Administrator** checkbox in the login dialog box.

Not logged in

Email:

☒ Sign in as Administrator

7. Go to the Developer Page. (The link in the navigation bar is only visible if you are logged in as an Administrator).

**Note:** The Developer page has some predefined content, which you will use in later exercises.

8. If you were watching carefully, you should see that the logout option on the menu bar disappears when you go to any page except Home. Think! What is being passed on the `HomePage` that is not being passed on all of the rest of the pages? Fix the problem and run the application locally again.

9. While you are viewing one of the Create Conference Page, the User Profile Page, or the Developer Page, logout.

You will be immediately requested to login again, because you are viewing a page that requires you to be logged in.



## Optional Exercise: Update the Home Page to Display Information about the User

To practice using the User Service API, update the Home Page to:

- Greet the user by their email if they are logged in.
- Display a login or logout link, as appropriate.
- If the user is logged in as an Administrator, display a link to the Developer Page.

### ➡ Java -- Update the Home Page to show info about the user

1. Write the code yourself to update the Home Page to greet the user by their login email, and to display either a login or logout link as appropriate, as well as to display a link to the Developer Page only if the user is an developer for the application.

Here are some hints to help you:

1. When users log in to the Conference Central application, they use their Google Accounts email as their login name.
2. Recall that header.jsp defines the `userPrincipal` variable as follows:

```
Principal userPrincipal = request.getUserPrincipal();
```

You can get the `mainEmail` (the login email) as follows:

```
if (userPrincipal != null) {  
    String mainEmail = userPrincipal.getName();  
}
```

### ➡ Python -- Update the Home Page to Show info about the User

1. Write the code yourself to update the Home Page template to greet the user by their login email, and to display either a login or logout link as appropriate, as well as to display a link to the Developer Page only if the user is an developer for the application. Update the Home Page template (`home.template`).

In `main.py`, the `_BaseHandler()` has already added all the required variables to the template dictionary. HINT: a user's email address can be obtained from `user.email()`.

The code necessary to modify the home template is in `home.template_update.txt` in `Code_Samples`



## Optional: Deploy the Application to App Engine

Follow Java or Python instructions for deploying your application as given below:

### Java Instructions

The project that you have downloaded is a Maven project. Follow the next steps to deploy the previous application.

1. Go to the command line / terminal and to the root directory of the project i.e. **cp300-gae-conference-app-complete-1-java**.
2. Build the project by giving the following command:  
  

```
mvn package
```
3. Go to src\main\webapp\WEB-INF folder. Open the appengine-web.xml file in an editor and provide the value for your application id in the <application> tag. Save the file.
4. Go back to the root folder i.e. **cp300-gae-conference-app-complete-1-java** and give the following command:

```
mvn appengine:update
```

You will be prompted for an authorization code in the terminal window and your web browser will launch with a consent screen which you must accept in order to be authorized. Follow the prompts to copy any codes from the browser to the command line. For more information refer to the [App Engine Maven Guide](#).

### Python Instructions

1. Go to the command line / terminal and to the root directory of the project i.e. **cp300-gae-conference-app-complete-1-python**.
2. Provide your application id in the app.yaml file. Save the file.
3. Deploy your application by giving the following command

```
appcfg.py update .
```

To test the application, open a new **Incognito Window** in Chrome, and go to the home page at `http://<yourappid>.appspot.com/`

Again, try accessing the Schedule Conference Page or the User Profile Page.

**Note:** See **Viewing and Adding Developers to Your Application** in [Tasks for Working with](#)



[App Engine Applications](#) to learn how to view the developers for your application.

### Summary

In this exercise, you learned how to extend your application to restrict access to some of the pages, to use the UserService to detect if the user is logged in or not and to provide links for the user to login and logout.