Getting Started with App Engine: Part 3

This guide is intended to help you get started working with AppEngine. If you know just a little bit of Python, HTML, and CSS, you should be able to work through these examples and get a live web application up and running pretty easily.

This chapter focuses on working with the App Engine users module. You should be familiar with Part (creating an App Engine application with simple outputs) before you start this material. Part 2 is not vital for this chapter, but it wouldn't hurt to take a look at it.

Table of Contents

Getting Started with App Engine: Part 3

Introduction

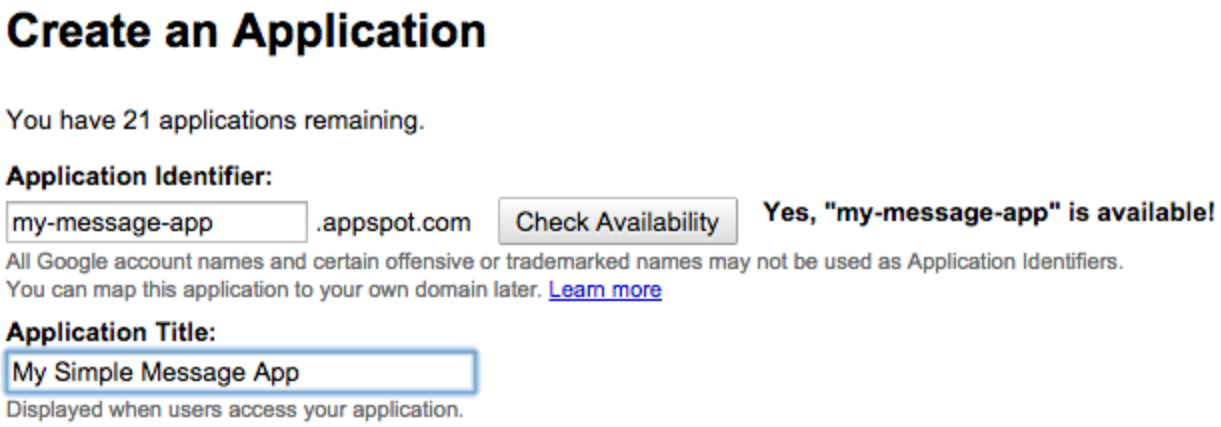
In the last 2 chapters, we reviewed how to deploy App Engine applications and how to process basic form data. We can do a lot with just those capabilities, but to go much futher we're going to want to store data. In order to do that, we're probably going to want to identify users who are saving the data.

App Engine provides a users service to allow you to easily authenticate users without the security headaches or implementation details typically required to build a full-featured user management system. We can use this service to easily create log in / log out capabilities for our application, which will be useful if we want to associate data with our users.

Creating an Application for Users

We'll start by going to http://appspot.com to create a new application ID for this exercise. I'm going to call this one "my-message-app" - you can call yours whatever you like (within the application name constraints, of course).

Your screen should look something like this:



After registration, you should see a screenshot that looks something like this:

Application Registered Successfully

The application will use **my-message-app** as an identifier. This identifier belongs in your application's configuration as well. Note that this identifier cannot be changed. Learn more

The application uses the **High Replication** storage scheme. Learn more

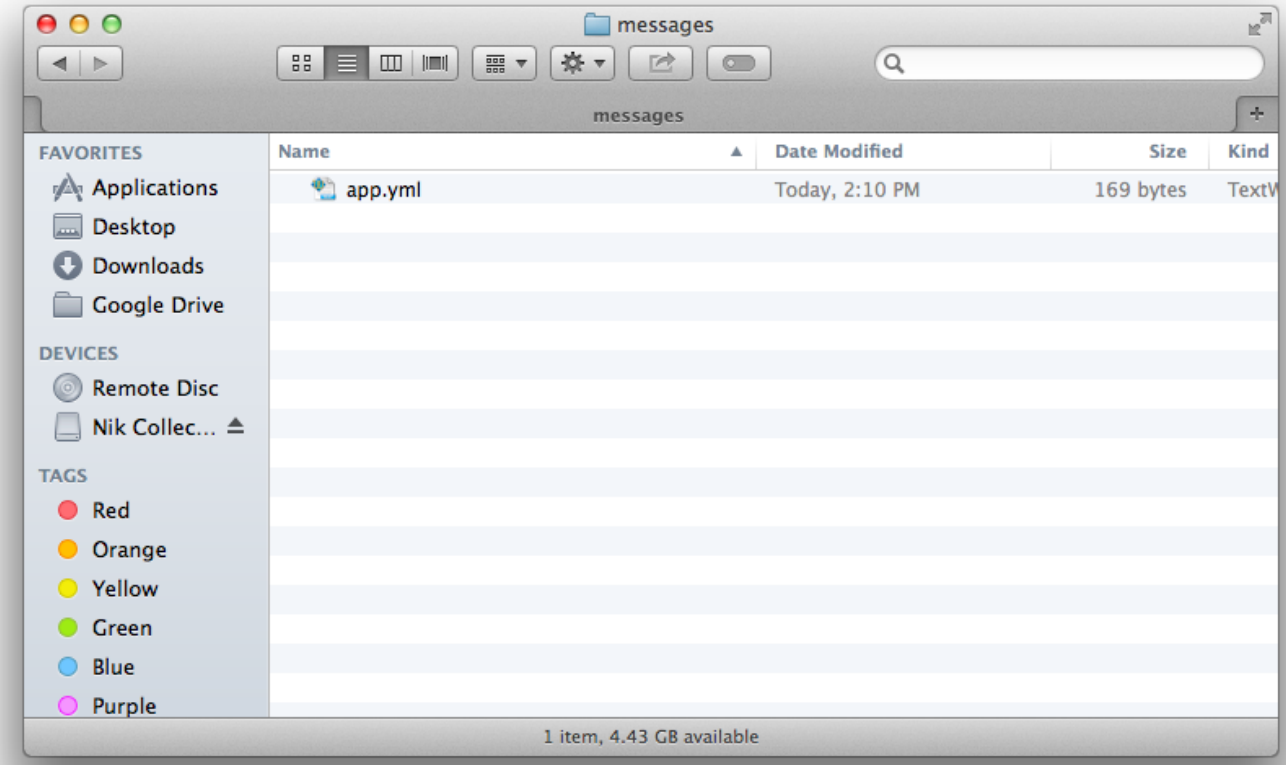
If you use Google authentication for your application, **My Simple Message App** will be displayed on Sign In pages when users access your application.

Choose an option below:

- View the dashboard for My Simple Message App.
- Use appcfg to upload and deploy your application code.
- Add administrators to collaborate on this application.

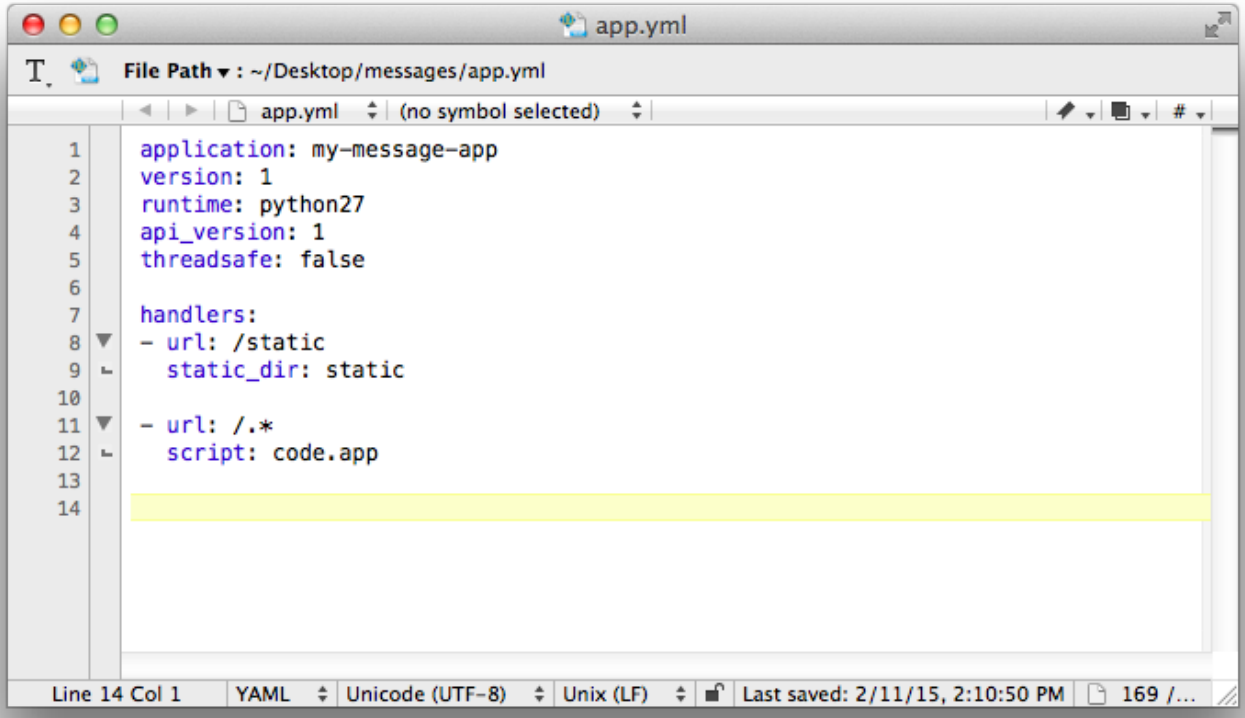
Getting Started with App Engine: Part 3

Now that we've created an App Engine application ID, we'll start creating the files. This time we'll start from scratch. I've created a folder below called "messages" with a single app.yaml file:



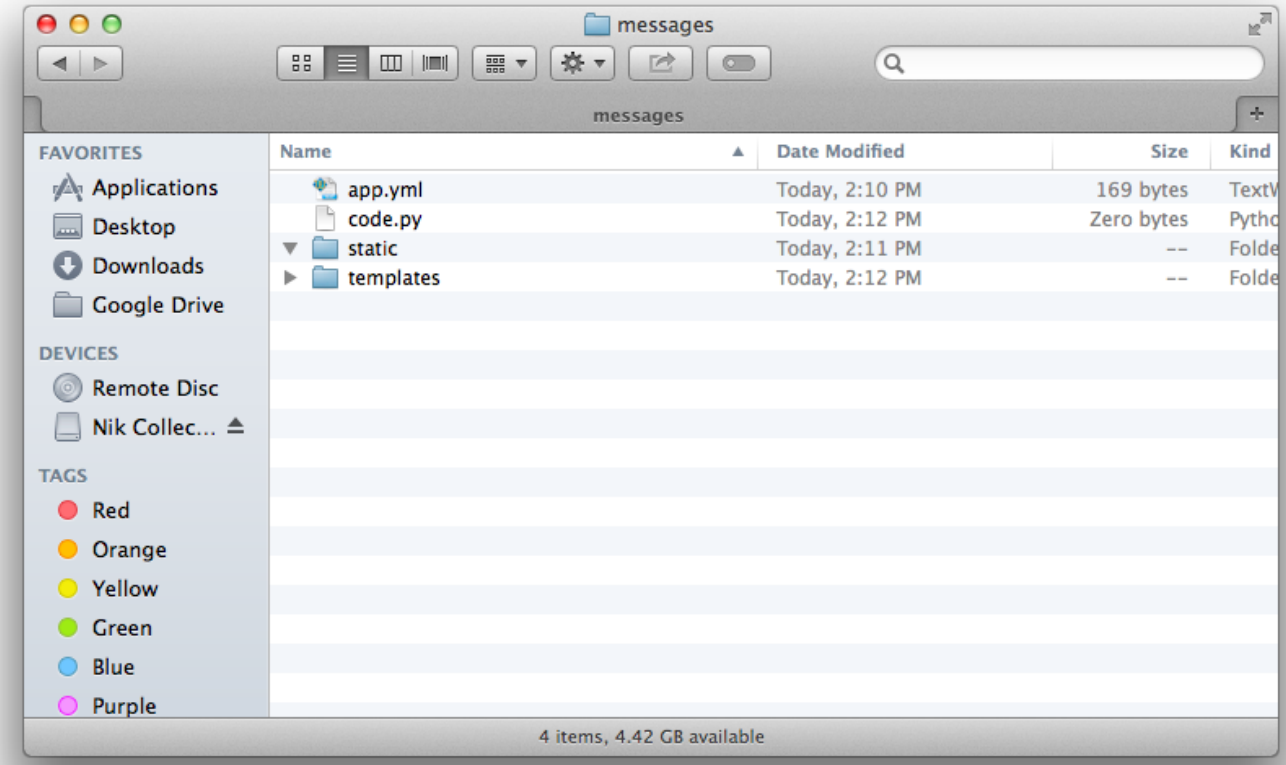
Getting Started with App Engine: Part 3

Your app.yaml file should be similar to the ones we used in Part 1 and Part 2. Fill in your application ID where I have “my-message-app” on line 1 below.

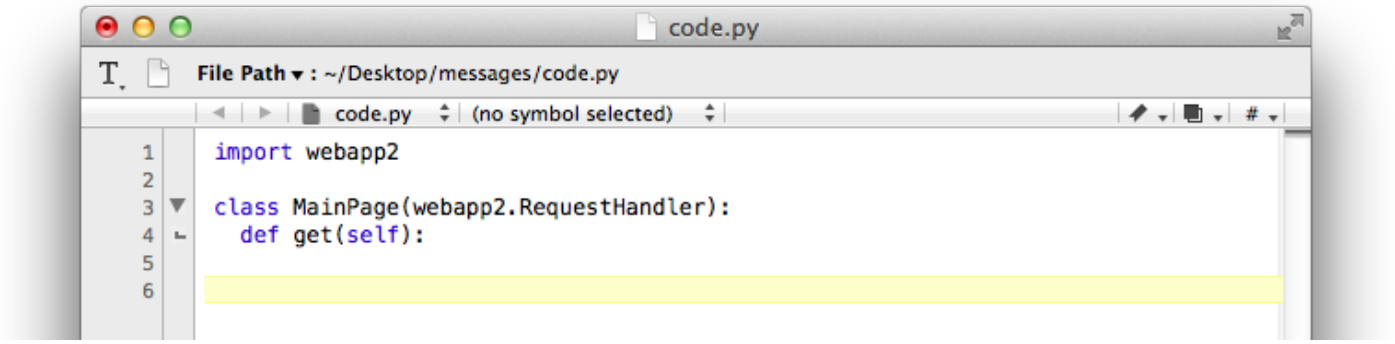


Getting Started with App Engine: Part 3

You should probably also create a “static” and a “templates” folder, though we won’t use those just yet. I created a code.py file below that we can use to build our application.



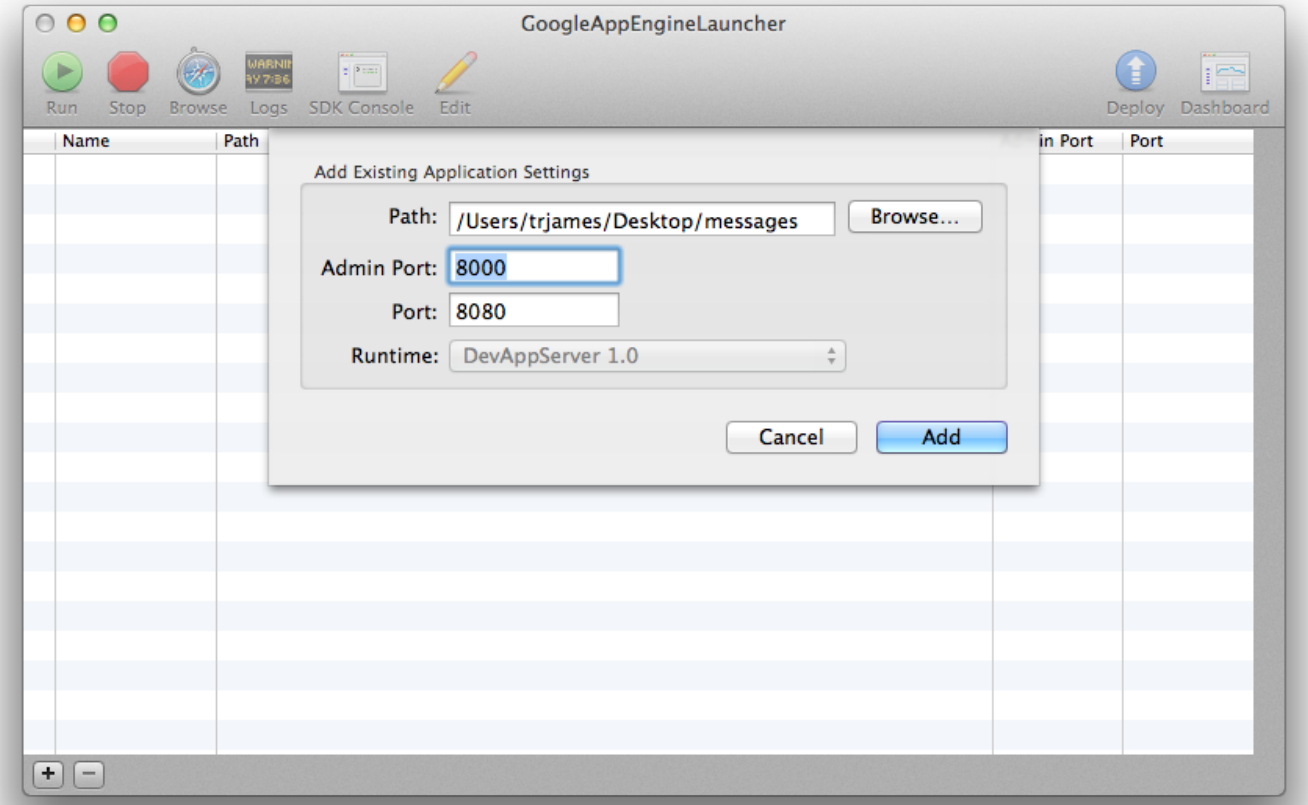
Let's start by importing webapp2 (on line 1 below). We'll declare our RequestHandler class on line 3, and a get method on line 4.



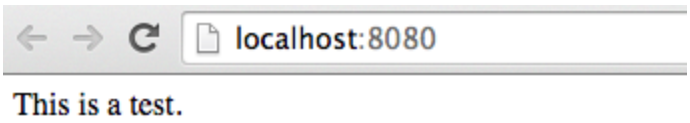
This gets us started, but we'll probably also want some test code and to create our application, so add a simple output on line 5, and add your app declaration on lines 8-10 as in the screenshot below.

Getting Started with App Engine: Part 3

If we've done everything correctly so far, we should be able to test very easily. We'll start by opening the App Engine SDK and choosing File => Add Existing Application from the menu. We should see a dialog like the one below:

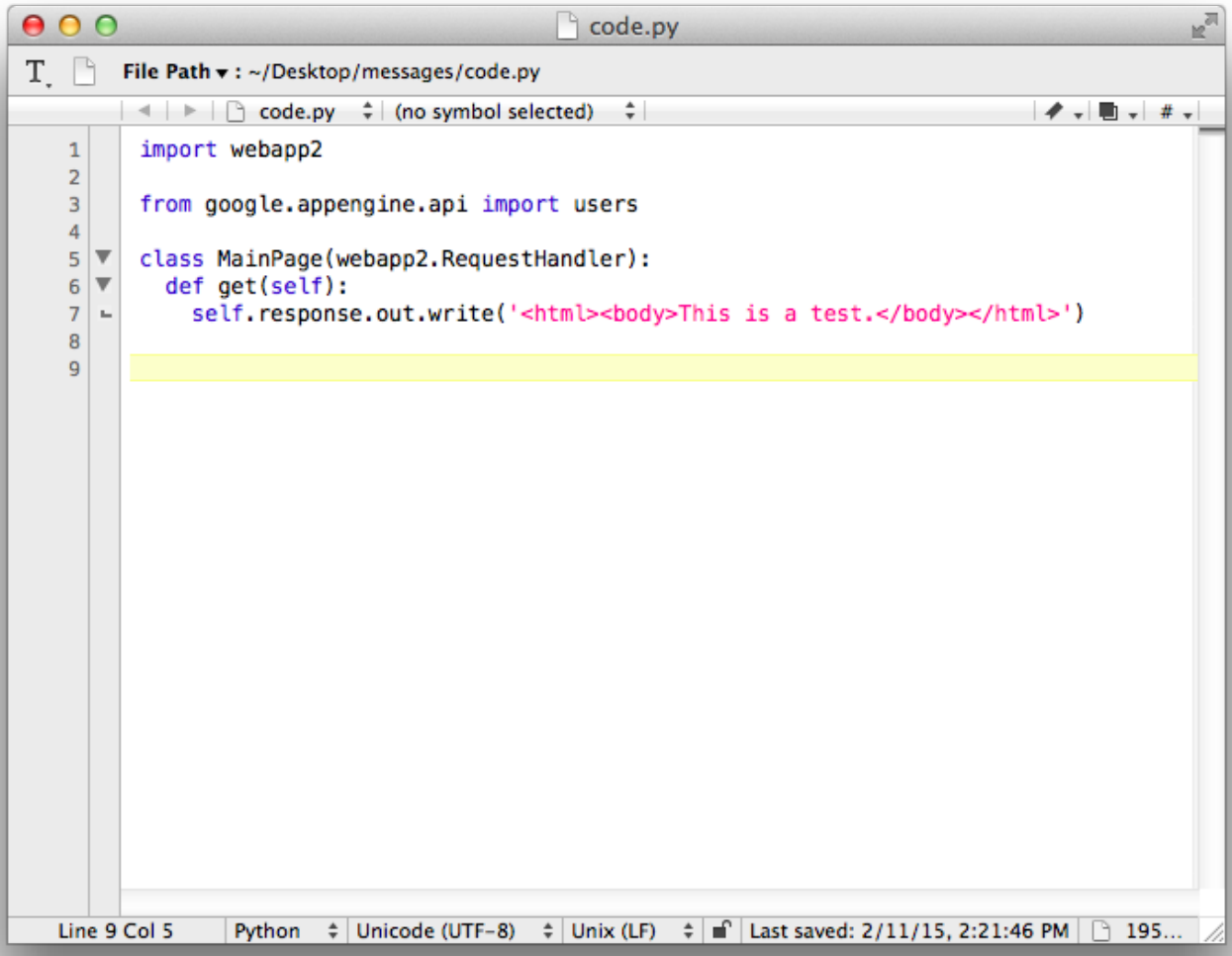


I created my "messages" folder on my Desktop so I clicked "Browse" then selected the Desktop folder; then I selected the "messages" folder and clicked the "Add" button. If everything worked, you should be able to run your application and test it in a browser - it should look something like this:

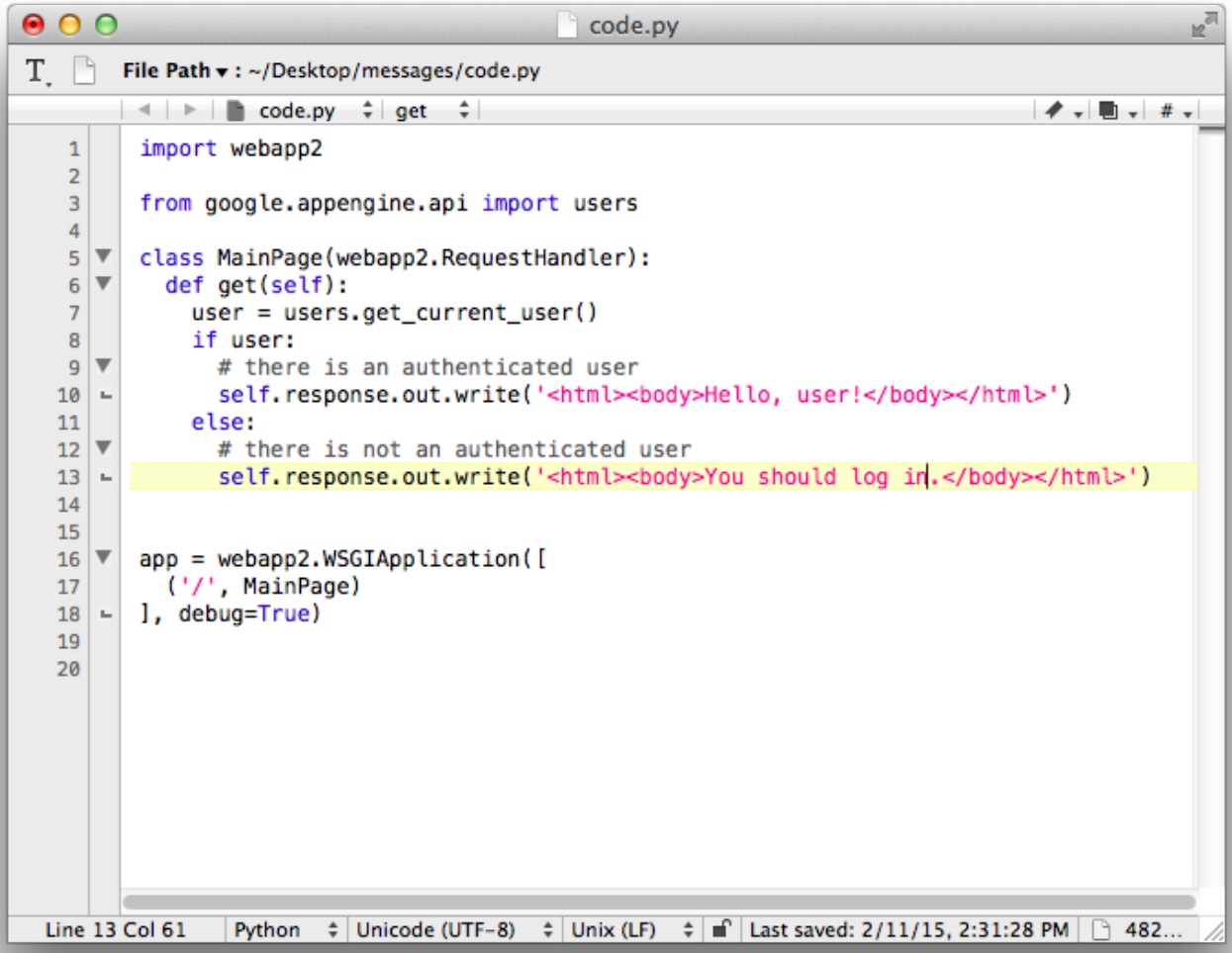


Determining if an Authenticated User Exists

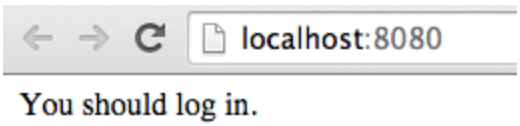
Now that we have a basic project up and running, we'll start working with users. The first thing we'll want to do is import the module for users from the App Engine API. You can see this below on line 3:



That will allow us to use App Engine's users module to retrieve basic user details. Determining if a user has logged in is really easy in App Engine; all we have to do is make a call to users.get_current_user() like in the screenshot below on line 7:



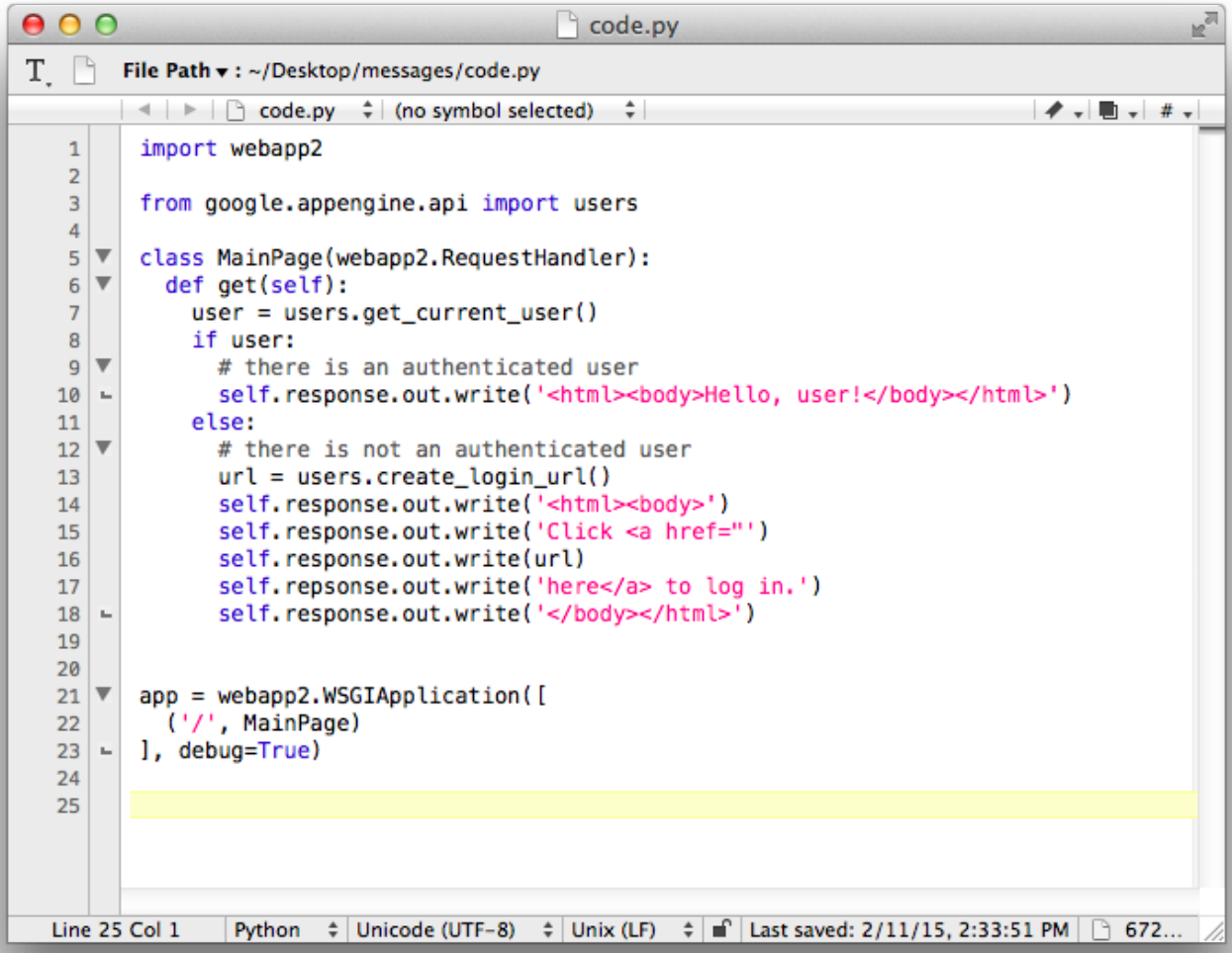
Note that we're also checking to see if the user is defined; if there is no user, users.get_current_user() will return "None" which evaluates to false, so we can easily take 2 different actions here, as in the if statement on line 8. If we have an authenticated user, we will greet them; if we don't, we'll tell them to sign in. If you test this code, you should see something like this:



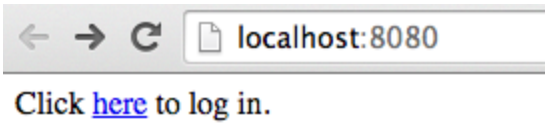
Allowing a User to Log In

So now we have a web application that can detect a user, but it doesn't actually allow a user to log in (it's a very exclusive club). Fortunately the users module provides us an easy way to do this as well. First we'll write some code that allows a user to log in. We start by retrieving a login URL; the users module will give us a URL that the user can visit to sign in by using the users.create_login_url() function. You can see this below on line 13.

We can embed this in a link within our HTML code; we'll change the simple HTML code from above, and on lines 14-18 we will create an HTML response that includes a link to this URL.



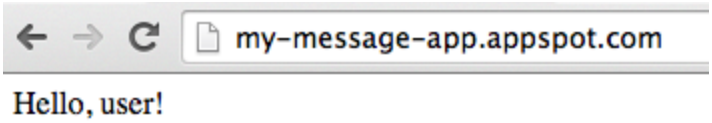
If this works properly, we should be able to log in by clicking the link in this screen:



If you're running from the SDK, clicking that link will take you to a page where you can specify the email address you want to use. This is simply for testing purposes - you can use any email address you like.

If you're running in the live App Engine environment, clicking that link will take you to the official Google sign-in page. You'll need to authenticate as a real user with your Google account on that page.

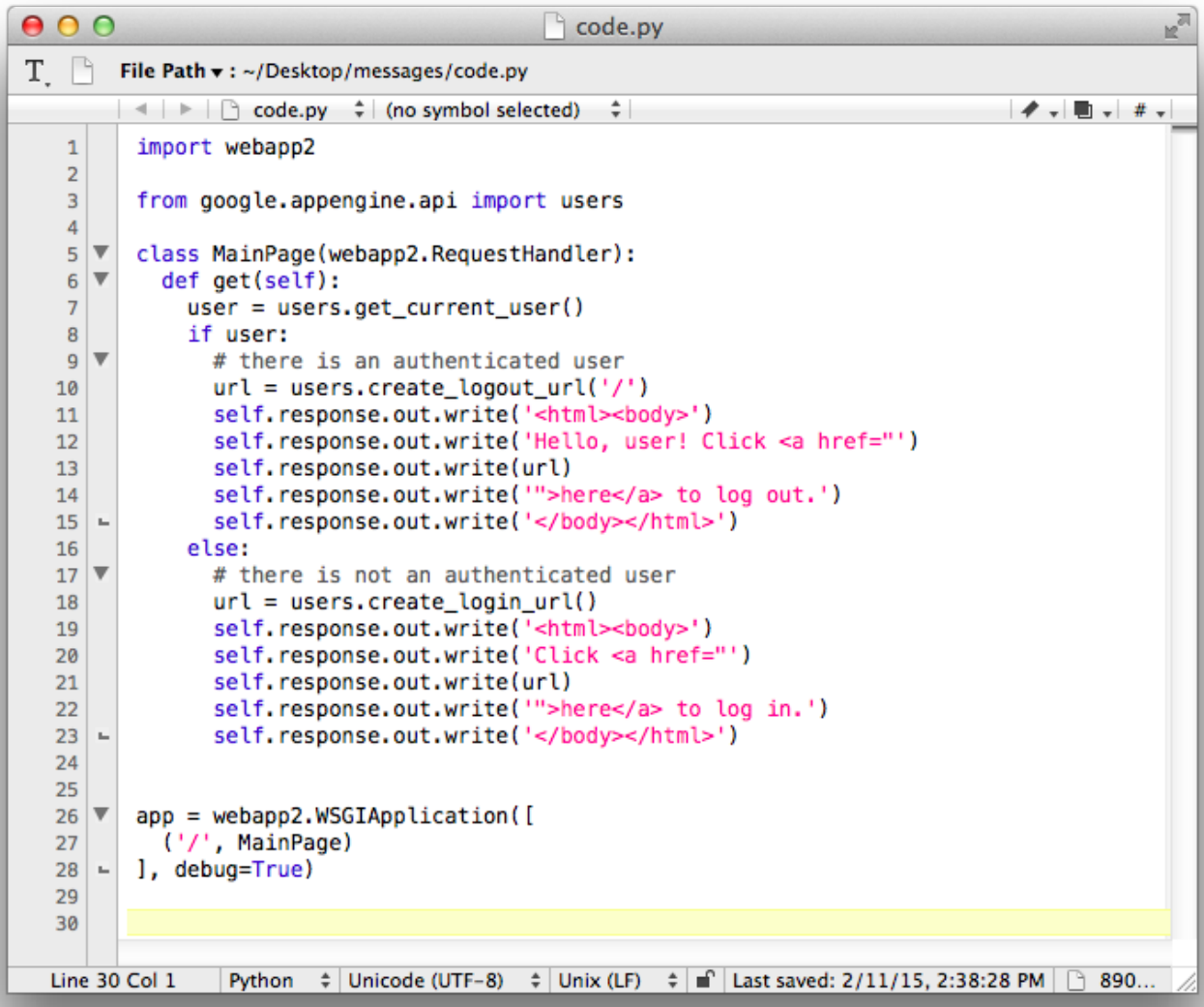
After you sign in, you should see a screen like this:



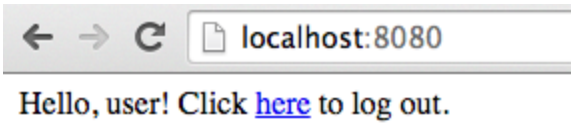
Allowing a User to Log Out

Now that we've allowed a user to log in, there's no way to get out (it's like a bad nightmare)! Fortunately this is easy as well. We can user a similar feature of the users module to build a logout URL.

On line 10 below, we create a logout URL with the users.create_logout_url() function. Note that this function takes a parameter; this is the path where your user will be redirected after he / she visits that URL. We'll create some HTML on lines 11-15 that includes the link to log out.



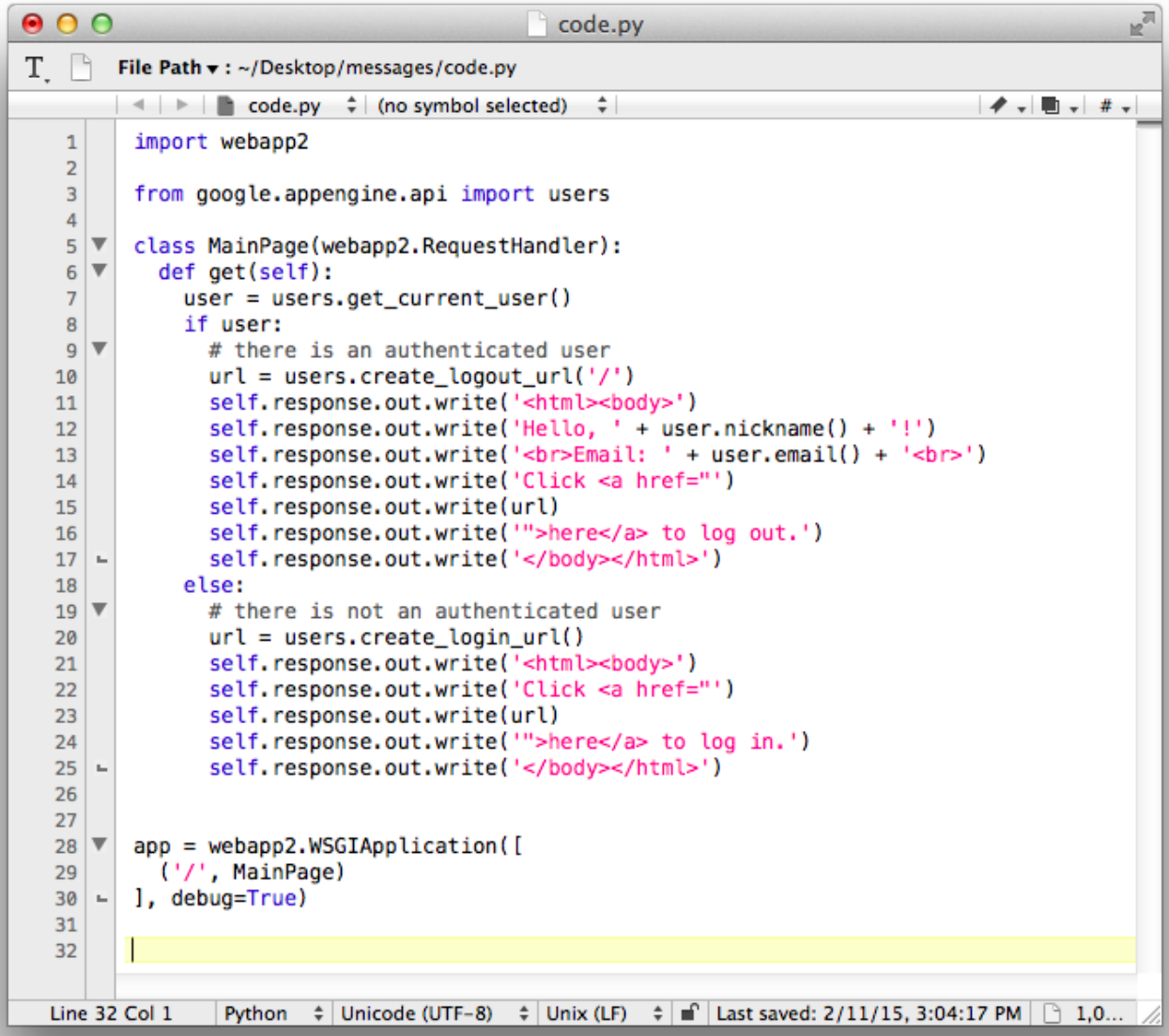
If everything works out, you should see a screen like this one:



Clicking that link will take you back to the original page; you can now log in or log out very easily. It's not a great application, but it will show you how simple authentication works.

There are two other methods available when you have a user object that you might want to use: user.email() and user.nickname(). These allow you to retrieve basic details about the user.

A few slight modifications to our application will allow us to use these; you can see on their use on line 12 and 13 below.



That's it for Part 3 of this tutorial; we'll build on this code in Part 4.