Flask-Stormpath Documentation

Release 0.4.6

Stormpath, Inc.

September 06, 2016

Contents

Flask-Stormpath is an extension for Flask that makes it *incredibly* simple to add users and user data to your application. It aims to completely abstract away all user registration, login, authentication, and authorization problems, and make building secure websites painless. And the best part? **You don't even need a database!**

Note: Unfortunately, this library will NOT work on Google App Engine due to the requests module not working :(

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started with Flask-Stormpath. If you're a new Flask-Stormpath user, start here!

1.1 About

Not sure if Flask-Stormpath is for you? Read along, and I'll help you decide whether or not Flask-Stormpath is a good fit for your project!

1.1.1 What is Stormpath?

Stormpath is an API service for creating and managing user accounts. Stormpath allows you to do things like:

- Create user accounts.
- Edit user accounts.
- Store user data with each account.
- Create groups and roles.
- Assign users various permissions (groups, roles, etc.).
- Handle complex authentication and authorization patterns.
- Log users in via social login with Facebook and Google OAuth.
- Cache user information for quick access.
- Scale your application as you get more users.
- Securely store your users and user data in a central location.

In the backend, what Stormpath does is provide a simple REST API for storing user accounts. For instance, if you wanted to create a new user account given an email address and password, you could send Stormpath an `HTTP POST` request and Stormpath would create a new user account for you, and store it securely on Stormpath's cloud service.

In addition to allowing you to create users, Stormpath also allows you to store custom data with each user account. Let's say you want to store a user's birthday – you can send Stormpath an `HTTP POST` request to the user's account and store *any* variable JSON data (birthdays, images, movies, links, etc.). This information is encrypted in transit and at rest, ensuring your user data is secure.

1.1.2 Who Should Use Stormpath

Stormpath is a great service, but it's not for everyone!

Firstly, Stormpath is only oriented around website users right now. This means that if you're building an API only application, Stormpath won't be a good fit.

If you're building a website, however (where you expect to register and log in users), keep reading!

You might want to use Stormpath if:

- You want to make user creation, management, and security as simple as possible (you can get started with Flask-Stormpath with only one line of code *excluding settings*)!
- User security is a top priority. We're known for our security.
- Scaling your userbase is a potential problem (Stormpath handles scaling your users transparently).
- You need to store custom user data along with your user's basic information (email, password).
- You would like to have automatic email verification for new user accounts.
- You would like to configure and customize password strength rules.
- You'd like to keep your user data separate from your other applications to increase platform stability / availability.
- You are building a service oriented application, in which multiple independent services need access to the same user data.
- You are a big organization who would like to use Stormpath, but need to host it yourself (Stormpath has an on-premise system you can use internally).

Basically, Stormpath is a great match for applications of any size where security, speed, and simplicity are top priorities.

You might **NOT** want to use Stormpath if:

- You are building an application that does not need user accounts.
- Your application is meant for internal-only usage.
- You aren't worried about user data / security much.
- You aren't worried about application availability / redundancy.
- You want to roll your own custom user authentication.

If you don't need Stormpath, you might want to check out Flask-Login (which Flask-Stormpath uses behind the scenes to handle sessions).

Want to use Stormpath? OK, great! Let's get started!

1.2 Setup

This section covers the basic setup you need to perform in order to get started with Flask-Stormpath.

1.2.1 Create a Stormpath Account

Now that you've decided to use Stormpath, the first thing you'll want to use is create a new Stormpath account: https://api.stormpath.com/register

1.2.2 Create an API Key Pair

Once you've created a new account, create a new API key pair by logging into your dashboard and clicking the "Create an API Key" button. This will generate a new API key for you, and prompt you to download your keypair.

Note: Please keep the API key pair file you just downloaded safe! These two keys allow you to make Stormpath API requests, and should be properly protected, backed up, etc.

Once you've downloaded your *apiKey.properties* file, save it in your home directory in a file named *~/.stormpath/apiKey.properties*. To ensure no other users on your system can access the file, you'll also want to change the file's permissions. You can do this by running:

```
$ chmod go-rwx ~/.stormpath/apiKey.properties
```

1.2.3 Create a Stormpath Application

Next, you'll want to create a new Stormpath Application.

Stormpath allows you to provision any number of "Applications". An "Application" is just Stormpath's term for a project.

Let's say you want to build a few separate websites. One site named "dronewars.com", and another named "carswap.com". In this case, you'd want to create two separate Stormpath Applications, one named "dronewars" and another named "carswap". Each Stormpath Application should represent a real life application of some sort.

The general rule is that you should create one Application per website (or project). Since we're just getting set up, you'll want to create a single Application.

To do this, click the "Applications" tab in the Stormpath dashboard, then click "Register an Application" and follow the on-screen instructions.

Note: Use the default options when creating an Application, this way you'll be able to create users in your new Application without issue.

Now that you've created an Application, you're ready to plug Flask-Stormpath into your project!

1.2.4 Bonus: Create a Directory

As you might have noticed, Stormpath also has something called "Directories". When you created an Application in the previous step, Stormpath automatically created a new Directory for you.

You can think of Directories in Stormpath as buckets of user accounts. Every user account that you create on Stormpath will belong to a Directory. Directories hold unique groups of users.

In most situations, your Stormpath Application will have a single Directory associated with it which is where all of your users will go. In some situations, however, you might want to create additional Directories (or share a Directory between multiple Applications).

Let's say you have two separate websites (*and therefore, two separate Stormpath Applications*): "dronewars.com" and "carswap.com". If you wanted both of your websites to share the same user accounts (*so that if a user signs up on dronewars.com they can use their same login on carswap.com*), you could accomplish that by having a single Directory, and mapping it to both of your Stormpath Applications.

Directories are useful for sharing and segmenting users in more complex authentication scenarios.

You can read more about Stormpath Directories here: http://docs.stormpath.com/rest/product-guide/#directories

Note: Stormpath has multiple types of "Directories". There are: "Cloud Directories", "Mirror Directories", "Facebook Directories" and "Google Directories".

Cloud Directories hold *typical* user accounts.

Facebook and Google Directories allow you to automatically create Stormpath user accounts for both Facebook and Google users (using social login). Social login will be covered in detail later on.

Mirror Directories are used for syncing with Active Directory and LDAP services (most people don't ever need these).

1.2.5 Install the Package

Now that you've got a Stormpath account all setup and ready to go, all that's left to do before we can dive into the code is install the Flask-Stormpath package from PyPI.

To install Flask-Stormpath, you'll need pip. You can install the latest version of Flask-Stormpath by running:

```
$ pip install Flask-Stormpath
```

If you'd like to upgrade to the latest version of Flask-Stormpath (*maybe you have an old version installed*), you can run:

```
$ pip install -U Flask-Stormpath
```

To force an upgrade.

Note: Flask-Stormpath is currently *only* compatible with Python 2.7. We will be adding Python 3 support in the near future.

1.3 Quickstart

Now that we've got all the prerequisites out of the way, let's take a look at some code! Integrating Flask-Stormpath into an application can take as little as **1 minute**!

1.3.1 Initialize Flask-Stormpath

To initialize Flask-Stormpath, you need to create a `StormpathManager` and provide some Flask settings.

You can do this in one of two ways:

1. Pass your Flask app into the `StormpathManager` directly:

```
from flask.ext.stormpath import StormpathManager

app = Flask(__name__)
stormpath_manager = StormpathManager(app)
```

2. Lazily initialize your `StormpathManager` (this is useful if you have a factory function creating your Flask application):

```
    from flask.ext.stormpath import StormpathManager

    stormpath_manager = StormpathManager()

    # some code which creates your app
    stormpath_manager.init_app(app)
```

The `StormpathManager` is what initializes Stormpath, grabs configuration information, and manages sessions / user state. It is the base of all Flask-Stormpath functionality.

1.3.2 Specify Your Credentials

Now that you have your manager configured, you need to supply some basic configuration variables to make things work:

```
app.config['SECRET_KEY'] = 'someprivatestringhere'
app.config['STORMPATH_API_KEY_FILE'] = '/path/to/apiKey.properties'
app.config['STORMPATH_APPLICATION'] = 'myapp'
```

Or, if you prefer to use environment variables to specify your credentials, you can do that easily as well:

```
from os import environ

app.config['SECRET_KEY'] = 'someprivatestringhere'
app.config['STORMPATH_API_KEY_ID'] = environ.get('STORMPATH_API_KEY_ID')
app.config['STORMPATH_API_KEY_SECRET'] = environ.get('STORMPATH_API_KEY_SECRET')
app.config['STORMPATH_APPLICATION'] = environ.get('STORMPATH_APPLICATION')
```

Note: The `STORMPATH_API_KEY_ID` and `STORMPATH_API_KEY_SECRET` variables can be found in the `apiKey.properties` file you downloaded in the previous step.

The `STORMPATH_APPLICATION` variable should be the name of your Stormpath application created in the Setup docs. “dronewars”, for instance.

The `SECRET_KEY` variable should be a random string – this is used by Flask internally for securing sessions – make sure this isn’t easily guessable!

Note: **Please don’t hardcode your API key information into your source code!** To keep your credentials safe and secret, we recommend storing these credentials in environment variables or keeping your `apiKey.properties` file out of version control.

1.3.3 Testing It Out

If you followed the two steps above, you will now have fully functional registration, login, and logout functionality active on your site!

Don’t believe me? Test it out! Start up your Flask web server now, and I’ll walk you through the basics:

- Navigate to `/register`. You will see a registration page. Go ahead and enter some information. You should be able to create a user account. Once you’ve created a user account, you’ll be automatically logged in, then redirected back to the root URL (`/`, by default).

- Navigate to `/logout`. You will now be logged out of your account, then redirected back to the root URL (`/`, by default).
- Navigate to `/login`. You will see a login page. You can now re-enter your user credentials and log into the site again.

Wasn't that easy?!

Note: You probably noticed that you couldn't register a user account without specifying a sufficiently strong password. This is because, by default, Stormpath enforces certain password strength rules on your Stormpath Directories.

If you'd like to change these password strength rules (or disable them), you can do so easily by visiting the Stormpath dashboard, navigating to your user Directory, then changing the "Password Strength Policy".

1.4 Product Guide

This product guide covers more advanced Flask-Stormpath usage. You can selectively jump around from topic-to-topic to discover all the neat features that Flask-Stormpath provides!

1.4.1 Enforce User Authentication

Now that we've seen how easy it is to register, login, and logout users in your Flask app, let's see how simple it is to restrict views to logged-in users only.

Let's say you have a simple view which should only be accessible to users who have logged in. Below is a code sample which shows how easy it is to restrict access to your view:

```
from flask.ext.stormpath import login_required

@app.route('/secret')
@login_required
def secret():
    return 'secret information here'
```

The *`login_required()`* decorator makes it really easy to enforce user authentication on your views.

If you try to visit the `/secret` URL and you're not logged in, you'll be redirected to: `/login?next=%2Fsecret`. If you then enter your credentials and log in – you'll be immediately redirected back to the page you were trying to access: `/secret`.

Note: If you have `TESTING` set to True in your Flask settings, this decorator will *NOT* enforce authentication. This is done to simplify unit testing.

1.4.2 Enforce User Authorization

Stormpath supports extremely complex authorization rules. This section aims to provide a basic introduction to Flask-Stormpath's authorization enforcement (this topic is covered in-depth later on).

The main authorization resource in Stormpath is the `Group`. A Stormpath Group is a named resource (*admins, developers, paid users, free users, etc.*) which can be assigned to any number of user accounts.

Let's say you're building a site that has three tiers of users: free users, paid users, and admins. In this case, you'd want to create three Stormpath Groups: `free users`, `paid users`, and `admins`.

Let's quickly take a look at how we can create and assign a Group to a *User*:

```
>>> directory = stormpath_manager.application.default_account_store_mapping.account_store

>>> free_users = directory.groups.create({'name': 'free users'})
>>> paid_users = directory.groups.create({'name': 'paid users'})
>>> admins = directory.groups.create({'name': 'admins'})

>>> # Put the current user into the 'Free Users' group.
>>> user.add_group(free_users)
```

Now that we've created our Groups, and also added our *User* to the "free users" group – let's see how we can enforce different types of authorization on our *User* using the *groups_required()* decorator:

```
from flask.ext.stormpath import groups_required

@app.route('/admins')
@groups_required(['admins'])
def admins_only():
    """A top-secret view only accessible to admins."""
    pass
```

If the *User()* tries to visit `/admins`, they'll get redirected to the login page and won't be able to access the view.

What if we wanted to build a view only accessible to users who are both free users and admins? In this case we could just list both required Groups:

```
@app.route('/free_and_admins')
@groups_required(['free users', 'admins'])
def free_users_and_admins_only():
    """Only free users and admins can access this view."""
    pass
```

Now that you've seen how you can require a *User()* to be a member of multiple Groups, let's take a look at how you can enforce selective Group membership:

```
@app.route('/any_user')
@groups_required(['free users', 'paid users', 'admins'], all=False)
def any_user():
    """A view accessible to any user, but only if they're logged in."""
```

The view above lists three Groups, and sets the `all` parameter to `False` – signifying that a *User* must be a member of **at least one** of the list Groups in order to gain access.

Note: If you have `TESTING` set to True in your Flask settings, this decorator will *NOT* enforce authentication. This is done to simplify unit testing.

1.4.3 Restrict Session Duration / Expiration

Another thing people commonly want to do is restrict how long a user can be logged in without activity before being forced to log into their account again.

As of the latest Flask-Stormpath release, this is now possible!

You can easily change the default session / cookie expiration by modifying the STORMPATH_COOKIE_DURATION setting:

```
from datetime import timedelta

app.config['STORMPATH_COOKIE_DURATION'] = timedelta(minutes=30)
```

By default, sessions / cookies will not expire for a year (out of convenience).

1.4.4 Access User Data

Let's take a quick look at how we can access user data from a custom view.

Let's say we've defined a simple view that should simply display a user's email address. We can make use of the magical *user* context variable to do this:

```
from flask.ext.stormpath import login_required, user

@app.route('/email')
@login_required
def name():
    return user.email
```

The *user* context allows you to directly interact with the current *User* model. This means you can perform *any* action on the *User* model directly.

For more information on what you can do with a *User* model, please see the Python SDK documentation: http://docs.stormpath.com/python/product-guide/#accounts

Let's say you want to change a user's given_name (*first name*). You could easily accomplish this with the following code:

```
>>> user.given_name = 'Randall'
>>> user.save()
```

As you can see above, you can directly modify *User* attributes, then persist any changes by running user.save().

1.4.5 Working With Custom User Data

In addition to managing basic user fields, Stomrpath also allows you to store up to 10MB of JSON information with each user account!

Instead of defining a database table for users, and another database table for user profile information – with Stormpath, you don't need either!

Let's take a look at how easy it is to store custom data on a *User* model:

```
>>> user.custom_data['somefield'] = 'somevalue'
>>> user.custom_data['anotherfield'] = {'json': 'data'}
>>> user.custom_data['woot'] = 10.202223
>>> user.save()

>>> user.custom_data['woot']
10.202223

>>> del user.custom_data['woot']
>>> user.save()
>>> user.custom_data['woot']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'woot'
```

As you can see above – storing custom information on a `User` account is extremely simple!

1.4.6 Customize Redirect Logic

As you might have already noticed by playing around with the registration and login pages – when you first register or log into an account, you'll be immediately redirected to the URL `/`.

This is actually a configurable setting – you can easily modify this default redirect URL by adding the following config setting:

```
app.config['STORMPATH_REDIRECT_URL'] = '/dashboard'
```

You can also redirect users to different URL after they register by adding this config setting:

```
app.config['STORMPATH_REGISTRATION_REDIRECT_URL'] = '/thank-you'
```

If this setting is not set, users will be redirected to `STORMPATH_REDIRECT_URL` after registration.

This allows you to build nicer apps as you can do stuff like redirect newly registered users to a tutorial, dashboard, or something similar.

Note: If a user visits a page which has restricted access, they'll be redirected to the login page. Once the user logs in, they'll be immediately redirected back to whatever page they were initially trying to access (this behavior overrides the `STORMPATH_REDIRECT_URL` setting).

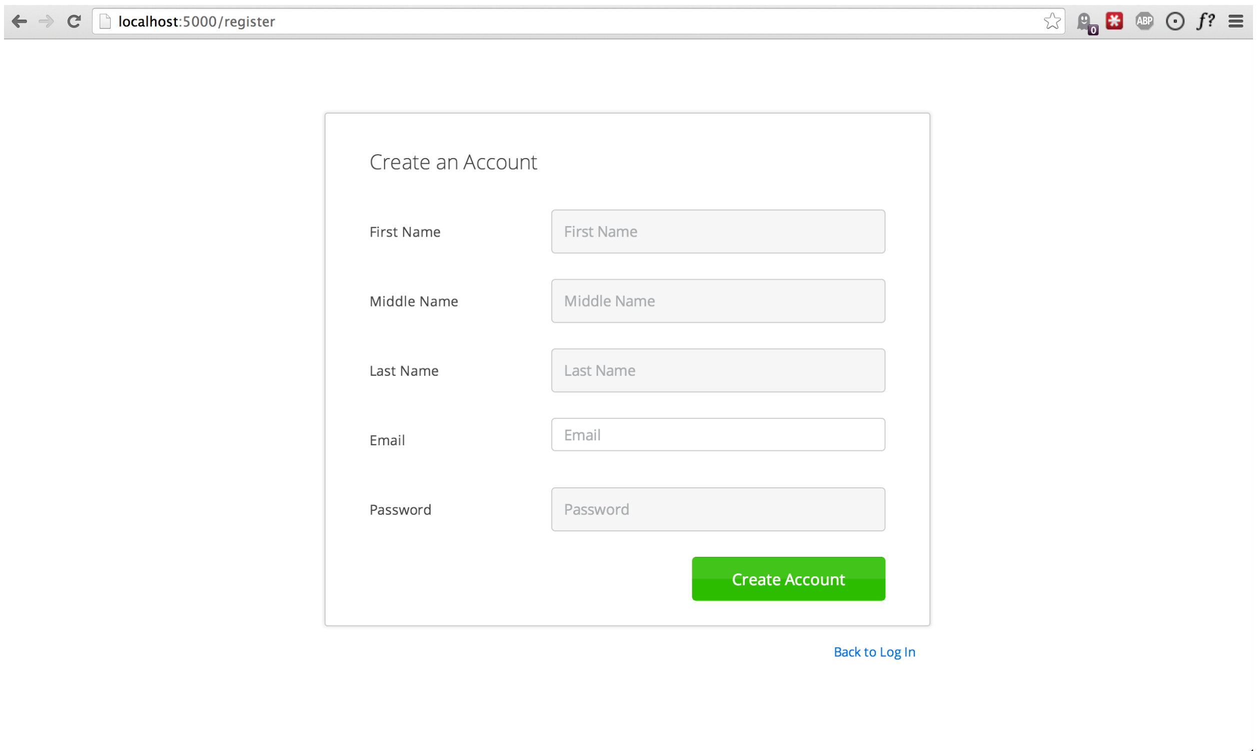
1.4.7 Customize User Registration Fields

In many cases you might want to change the fields you collect when a user registers. Let's customize the fields we ask for when a user registers!

Every user you register ends up getting stored in Stormpath as an Account object. Accounts in Stormpath have several fields you can set:

- username
- email (**required**)
- password (**required**)
- given_name (**required**) also known as "first name"
- middle_name
- surname (**required**) also known as "last name"

By default, the built-in registration view that Flask-Stormpath ships with gets you a registration page that looks like this:


localhost:5000/register

Create an Account

First Name: First Name

Middle Name: Middle Name

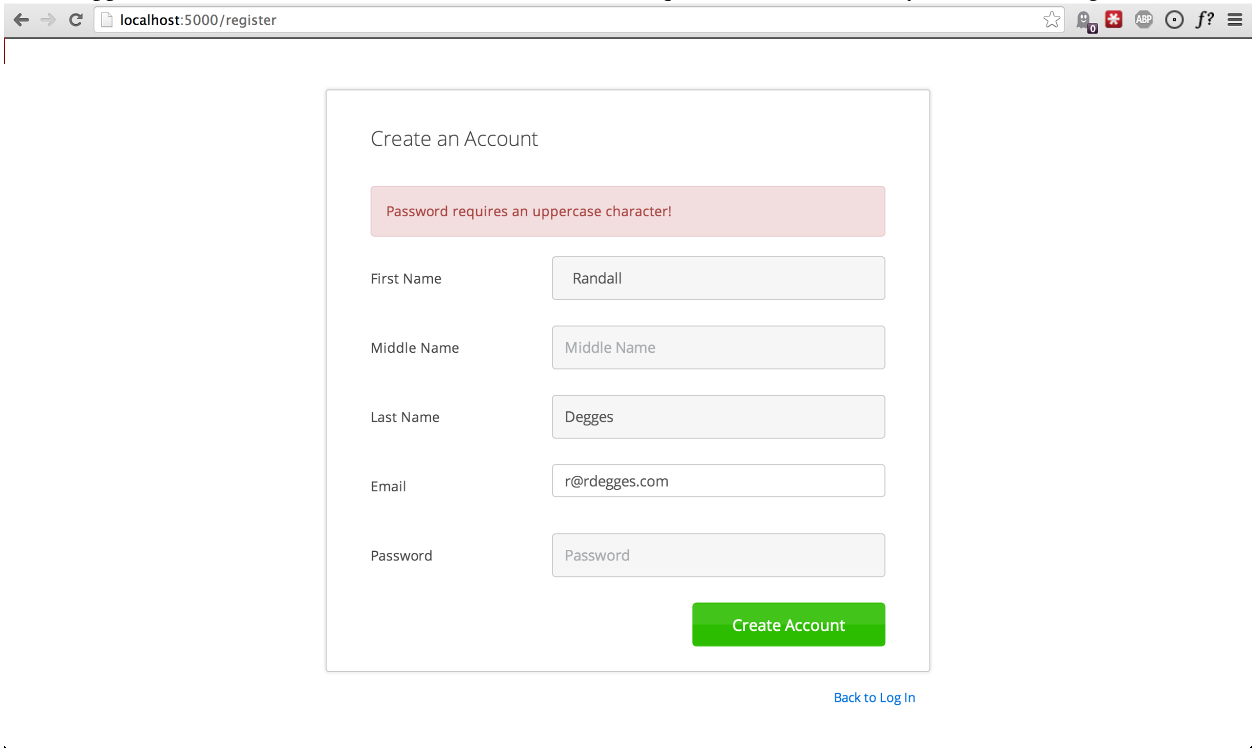
Last Name: Last Name

Email: Email

Password: Password

As you can see, it includes the `given_name`, `middle_name`, `surname`, `email`, and `password` fields by default. All of these fields are required, with the exception of `middle_name`.

What happens if a user enters an invalid value – or leaves a required field blank? They'll see something like this:


localhost:5000/register

Create an Account

Password requires an uppercase character!

First Name: Randall

Middle Name: Middle Name

Last Name: Degges

Email: r@rdegges.com

Password: Password

But what if you want to force the user to enter a value for middle name? Doing so is easy! Flask-Stormpath is **highly customizable**, and allows you to easily control which fields are accepted, and which fields are required.

To require a user to enter a middle name field, set the following value in your Flask app config:

```
app.config['STORMPATH_REQUIRE_MIDDLE_NAME'] = True
```

Now go ahead and give it a try – if you attempt to create a new user and don't specify a middle name, you'll see an error!

But what if you wanted to only accept `email` and `password`? By using the `STORMPATH_ENABLE_*` and `STORMPATH_REQUIRE_*` settings in your Flask app, you can completely customize which fields are accepted (*and required*)! Now, remove the `STORMPATH_REQUIRE_MIDDLE_NAME` setting and add the following in its place:

```
app.config['STORMPATH_ENABLE_GIVEN_NAME'] = False
app.config['STORMPATH_ENABLE_MIDDLE_NAME'] = False
app.config['STORMPATH_ENABLE_SURNAME'] = False
```

If you refresh the registration page, you'll now see a form that only accepts `email` and `password`! Not bad, right?

Note: If you explicitly disable the `given_name` and `surname` fields as shown above, those fields will automatically receive the value `'Anonymous'` (as they are required by Stormpath).

We're currently working to make these fields optional on Stormpath's side.

Want to keep everything as default, except make first and last name optional for the user? All you'd have to do is:

```
app.config['STORMPATH_REQUIRE_GIVEN_NAME'] = False
app.config['STORMPATH_REQUIRE_SURNAME'] = False
```

Lastly, it's also simple to add in a `username` field (either required or optional). Just like the examples above, you can use the `ENABLE` and `REQUIRE` settings to control the registration behavior:

```
app.config['STORMPATH_ENABLE_USERNAME'] = True
app.config['STORMPATH_REQUIRE_USERNAME'] = False
```

And that's it!

1.4.8 Customize User Login Fields

If you visit your login page (`/login`), you will see (*by default*), two input boxes: one for `email` and one for `password`.

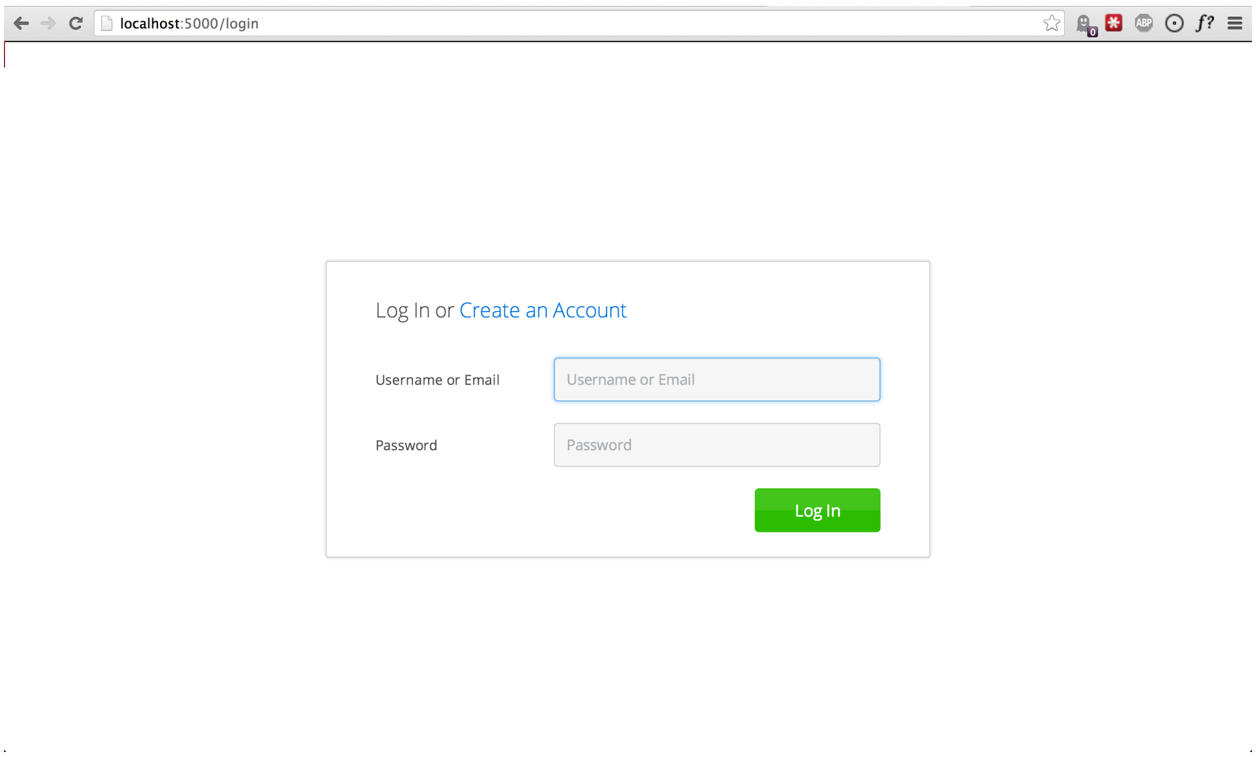
While this is fine for most purposes, sometimes you might want to let users log in with a `username` **or** `email` (especially if your site collects `username` during registration).

Doing this is simple: by enabling the `STORMPATH_ENABLE_USERNAME` setting you'll not only make the `username` field available on the registration page, but also on the login page (so users can log in by entering either their `username` or `email` and `password`).

To enable `username` support, just set the following config variable:

```
app.config['STORMPATH_ENABLE_USERNAME'] = True
```

You should now see the following on your login page:



Note: In the example above we didn't set the `STORMPATH_REQUIRE_USERNAME` field to `True` – if we did, this would ensure that when a new user registers for the site, they **must** pick a `username`.

The `STORMPATH_REQUIRE_USERNAME` field has no effect on the login page.

1.4.9 Customize User Registration, Login, and Logout Routes

By default, Flask-Stormpath automatically enables three separate views and routes:

- `/register` - the registration view
- `/login` - the login view
- `/logout` - the logout view

Customizing the built-in URL routes is quite simple. There are several config variables you can change to control these URL mappings. To change them, just modify your app's config.

- `STORMPATH_REGISTRATION_URL` – default: `/register`
- `STORMPATH_LOGIN_URL` – default: `/login`
- `STORMPATH_LOGOUT_URL` – default: `/logout`

If you were to modify your config such that:

```
app.config['STORMPATH_REGISTRATION_URL'] = '/welcome'
```

Then visit `/welcome`, you'd see your registration page there, instead!

1.4.10 Customize the Templates

Although I personally find our registration and login pages to be incredibly good looking – I realize that you might not share my same design passion!

Flask-Stormpath was built with customizability in mind, and makes it very easy to build your own custom registration and login templates.

Let's start by looking at the built-in templates: https://github.com/stormpath/stormpath-flask/tree/develop/flask_stormpath/templates/flask_stormpath

Here's a quick rundown of what each template is for:

- `base.html` is the base template that the registration and login templates extend. It provides a basic bootstrap based layout, with a couple of blocks for customizing the child templates.
- `facebook_login_form.html` is a simple standalone template that includes a Facebook login button (*for social login, which is covered later on in the guide*).
- `google_login_form.html` is a simple standalone template that includes a Google login button (*for social login, which is covered later on in the guide*).
- `login.html` is the login page. It has some logic to flash error messages to the user if something fails, and also dynamically determines which input boxes to display based on the app's settings.
- `register.html` is the registration page. It has some logic to flash error messages to the user if something fails, and also dynamically determines which input boxes to display based on the app's settings.

If you're comfortable with Jinja2, you can copy these templates to your project directly, and customize them yourself. If you're not already a super Flask guru, continue reading!

The Most Basic Templates

Let's say you want to build your own, fully customized registration and login templates – no problem!

The first thing you need to do is create two templates in the `templates` directory of your project.

First, copy the following code into `templates/register.html`:

```
<form method="post">
  {{ form.hidden_tag() }}

  {# This bit of code displays a list of error messages if anything bad happens. #}
  {% with messages = get_flashed_messages() %}
    {% if messages %}
      <ul>
        {% for message in messages %}
          <li>{{ message }}</li>
        {% endfor %}
      </ul>
    {% endif %}
  {% endwith %}

  {# This block of code renders the desired input boxes for registering users.  #}
  {% if config['STORMPATH_ENABLE_USERNAME'] %}
    {% if config['STORMPATH_REQUIRE_USERNAME'] %}
      {{ form.username(placeholder='Username', required='true') }}
    {% else %}
      {{ form.username(placeholder='Username') }}
    {% endif %}
  {% endif %}
  {% if config['STORMPATH_ENABLE_GIVEN_NAME'] %}
    {% if config['STORMPATH_REQUIRE_GIVEN_NAME'] %}
      {{ form.given_name(placeholder='First Name', required='true') }}
    {% else %}
      {{ form.given_name(placeholder='First Name') }}
    {% endif %}
  {% endif %}
  {% if config['STORMPATH_ENABLE_MIDDLE_NAME'] %}
    {% if config['STORMPATH_REQUIRE_MIDDLE_NAME'] %}
      {{ form.middle_name(placeholder='Middle Name', required='true') }}
    {% else %}
      {{ form.middle_name(placeholder='Middle Name') }}
    {% endif %}
  {% endif %}
  {% if config['STORMPATH_ENABLE_SURNAME'] %}
    {% if config['STORMPATH_REQUIRE_SURNAME'] %}
      {{ form.surname(placeholder='Last Name', required='true') }}
    {% else %}
      {{ form.surname(placeholder='Last Name') }}
    {% endif %}
  {% endif %}
  {{ form.email(placeholder='Email', required='true', type='email') }}
  {{ form.password(placeholder='Password', required='true', type='password') }}

  <button type="submit">Create Account</button>
</form>
```

The simple template you see above is the most basic possible registration page. It's using Flask-WTF to render the form fields, but everything other than that is all standard – nothing special happening.

Next, copy the following code into `templates/login.html`:

```
{# Display errors (if there are any). #}
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul>
      {% for message in messages %}
        <li>{{ message }}</li>
      {% endfor %}
    </ul>
  {% endif %}
{% endwith %}

{# Render the login form. #}
<form method="post">
  {{ form.hidden_tag() }}
  {% if config['STORMPATH_ENABLE_USERNAME'] %}
    {{ form.login(placeholder='Username or Email', required='true') }}
  {% else %}
    {{ form.login(placeholder='Email', required='true') }}
  {% endif %}
  {{ form.password(placeholder='Password', required='true') }}
  <button type="submit">Log In</button>
</form>

{# If social login is enabled, display social login buttons. #}
{% if config['STORMPATH_ENABLE_FACEBOOK'] or config['STORMPATH_ENABLE_GOOGLE'] %}
  <p>Or, log in using a social provider.</p>
  {% if config['STORMPATH_ENABLE_FACEBOOK'] %}
```

```
    {% include "flask_stormpath/facebook_login_form.html" %}
  {% endif %}
  {% if config['STORMPATH_ENABLE_GOOGLE'] %}
    {% include "flask_stormpath/google_login_form.html" %}
  {% endif %}
{% endif %}
```

This is the most basic login template possible (it also includes support for social login, which is covered later in this guide).

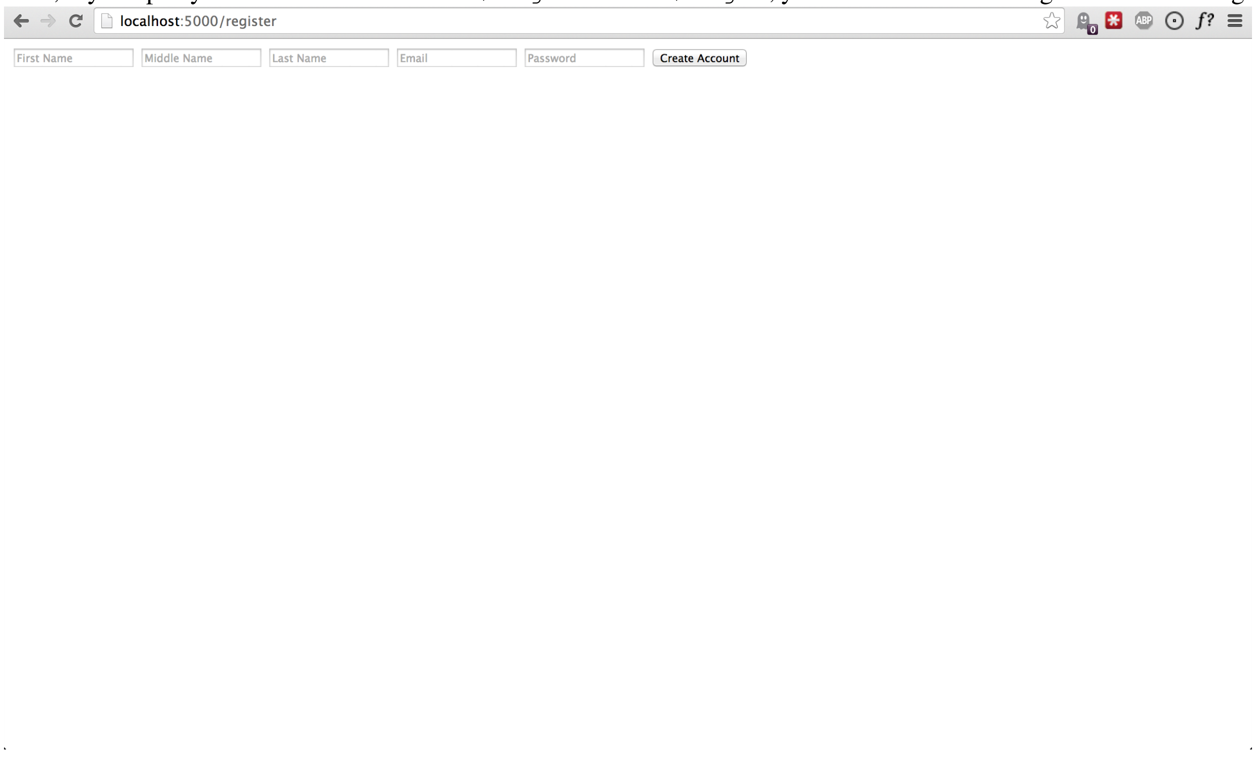
Update Template Paths

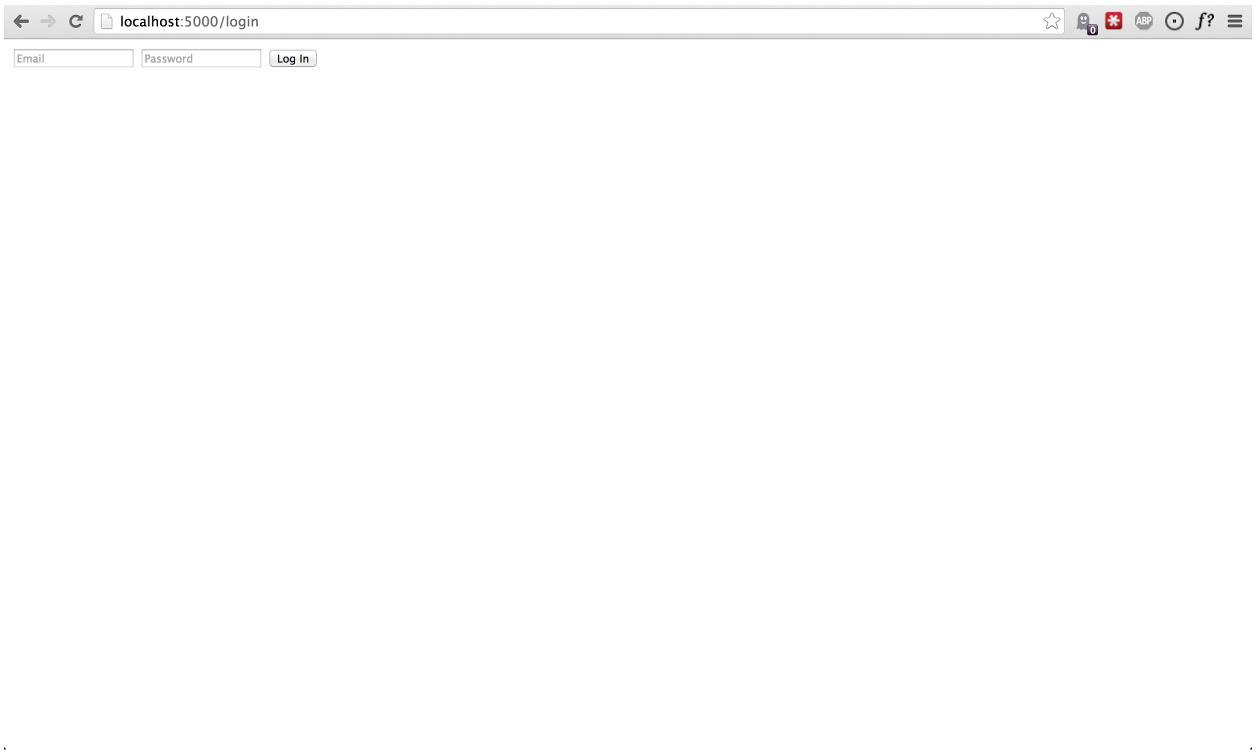
Now that you've got the simplest possible templates ready to go, let's activate them! In your app's config, you'll need to specify the path to your new templates like so:

```
app.config['STORMPATH_REGISTRATION_TEMPLATE'] = 'register.html'
app.config['STORMPATH_LOGIN_TEMPLATE'] = 'login.html'
```

That will tell Flask-Stormpath to render the templates you created above instead of the built-in ones!

Now, if you open your browser and checkout `/register` and `/login`, you should see something like the following:





BAM! That wasn't so bad, was it? You now have your own customized registration and login templates – all you need to do now is design them the way you want!

1.4.11 Disable the Built-in Views

If for some reason you want to write your own registration, login, and logout views (not recommended), you can easily disable all of the automatic functionality described above by modifying your app config and adding the following:

```
app.config['STORMPATH_ENABLE_REGISTRATION'] = False
app.config['STORMPATH_ENABLE_LOGIN'] = False
app.config['STORMPATH_ENABLE_LOGOUT'] = False
```

1.4.12 Use Password Reset

As of Flask-Stormpath **0.2.6**, it is now possible to easily enable a "Password Reset Workflow", which allows your users to reset their passwords automatically.

We highly encourage you to use this feature, as it provides a simple and secure way to allow your users to reset their passwords without hassle.

Configure the Workflow

The first thing you need to do to enable "Password Reset" functionality in your Flask app is visit the Directory Dashboard and select your default user directory.

Next, you should see several options in a tab. You will want to click the "Workflows" button. Once you've landed on this page, you'll then want to click the "show" link to the right of the "Password Reset" header. This section allows you to configure your "Password Reset" settings.

On this page, the only thing you **need** to change is the "Base URL" setting at the top. You need to set this to be: `https://mysite.com/forgot/change`, substituting in your own website address.

For instance, if your site lives at `https://www.mysite.com`, you'll want to set "Base URL" to `https://www.mysite.com/forgot/change`.

This URL determines where a user will be redirected after attempting to reset their password on your website. If you're testing things out locally, you can also set this to a local URL (eg: `http://localhost:5000/forgot/change`).

After setting "Base URL", you can also adjust any of the other settings below – you can customize the email templates that are used to email the user, and a variety of other options.

When you're finished customizing the "Password Reset Workflow", be sure to hit the "Update" button at the bottom of the page.

Enable Password Reset in Your App

Now that you've configured the "Password Reset" settings on Stormpath's side, you need to configure your Flask application to enable password reset.

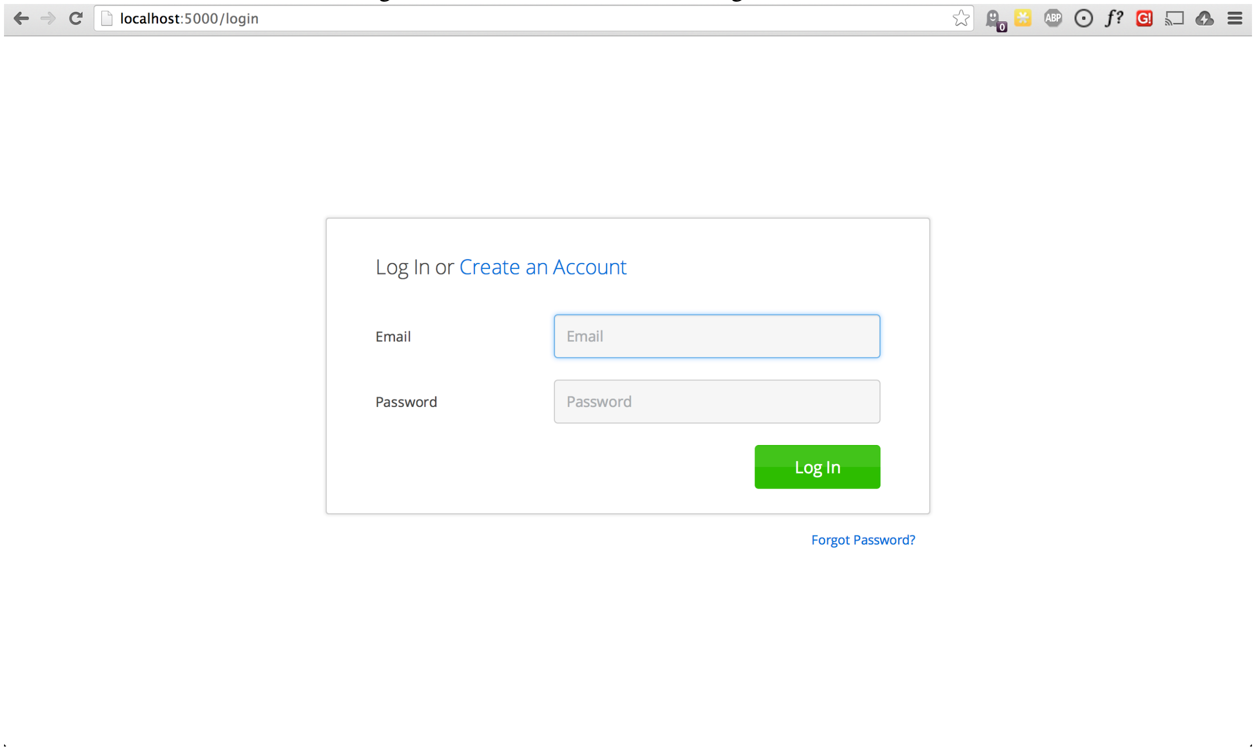
You can do this easily by modifying your application config like so:

```
app.config['STORMPATH_ENABLE_FORGOT_PASSWORD'] = True
```

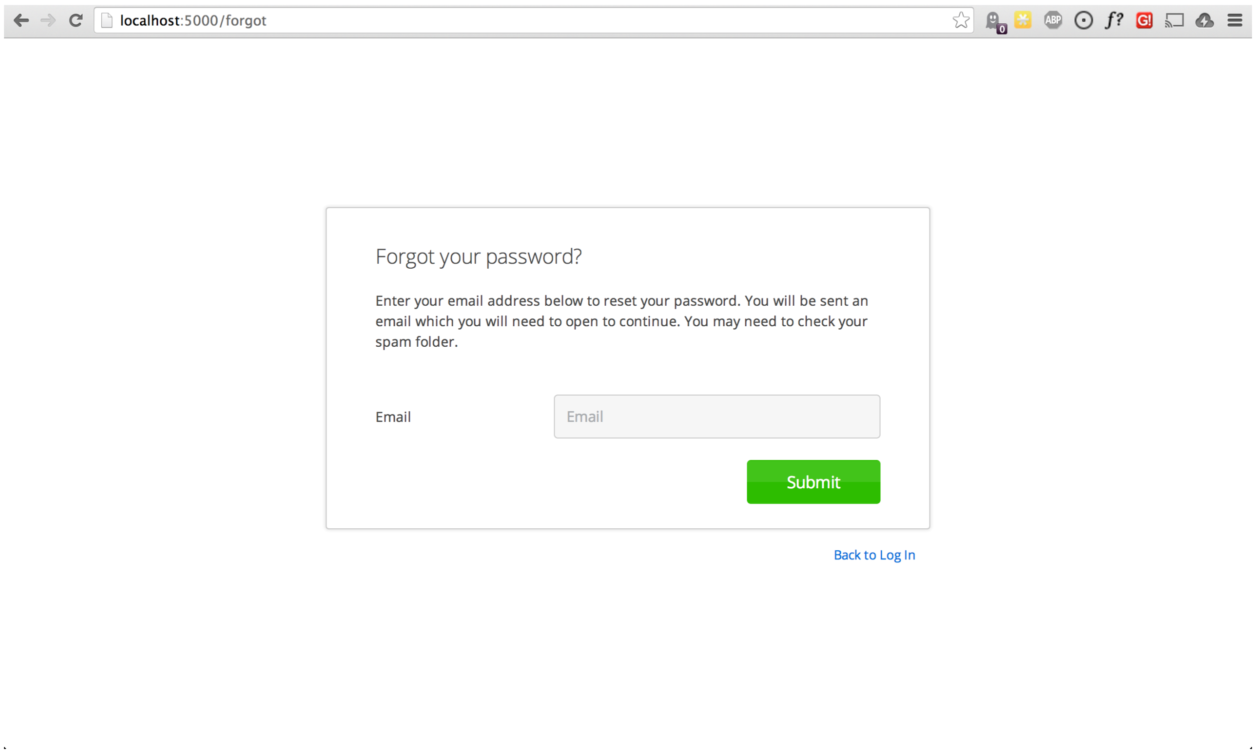
And... That's all you have to do!

Test it Out

Now that you've fully enabled password reset functionality in your app, open up the login page in your Flask app and check it out! You should see a "Forgot Password?" link below the login form which looks like this:



If you click the "Forgot Password?" link, you'll be brought to a password reset page that looks like this:

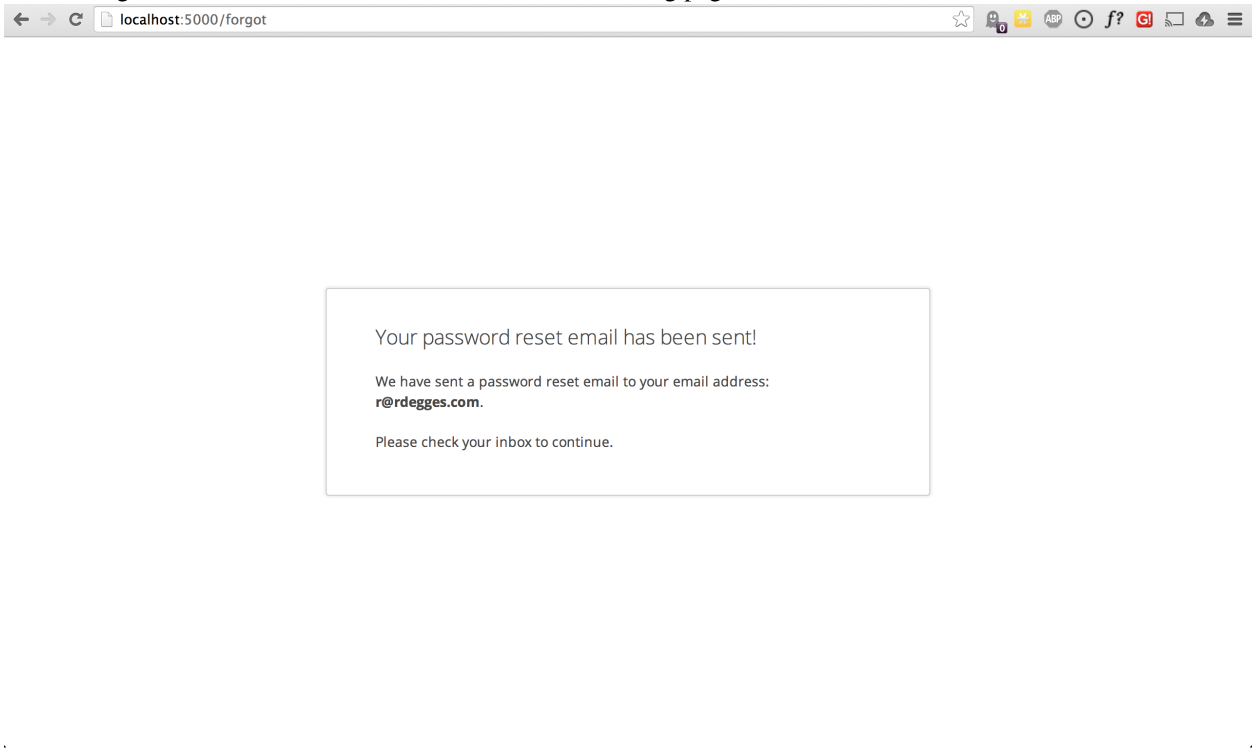

localhost:5000/forgot

Forgot your password?

Enter your email address below to reset your password. You will be sent an email which you will need to open to continue. You may need to check your spam folder.

Email

After filling in their email address, a user will see the following page:

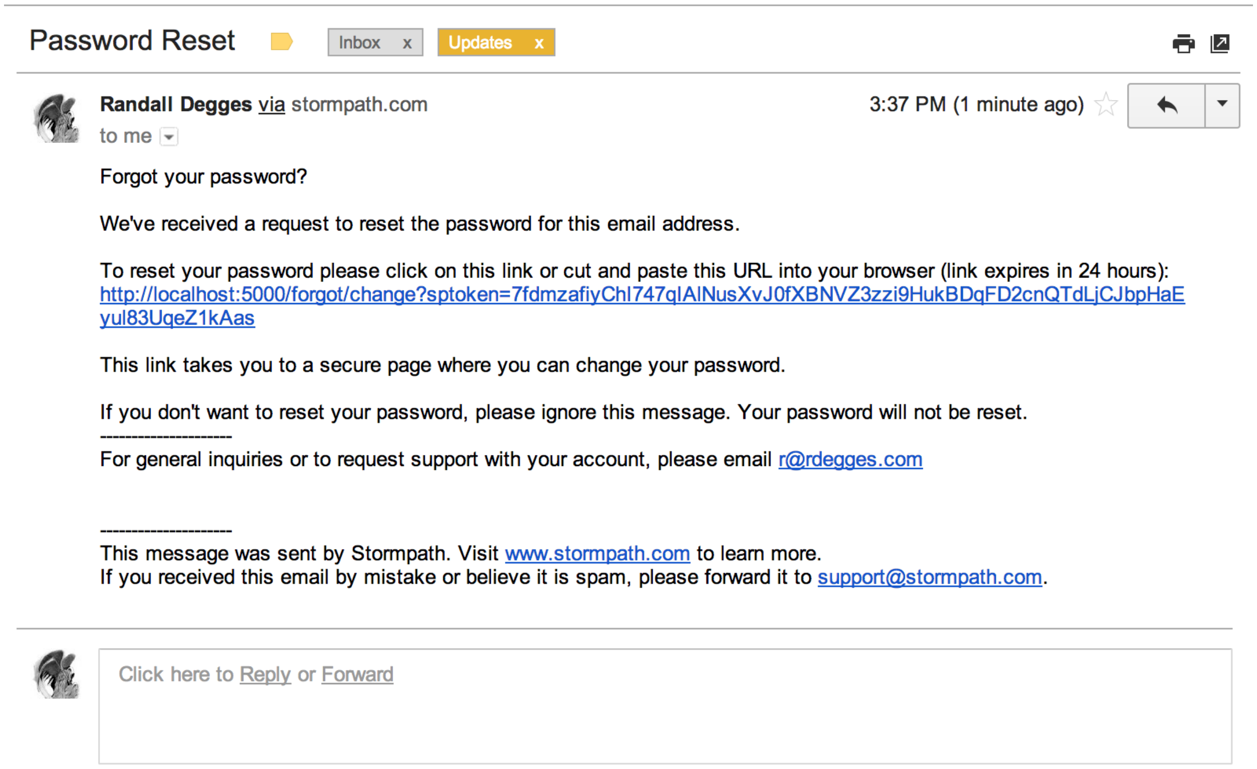

localhost:5000/forgot

Your password reset email has been sent!

We have sent a password reset email to your email address: **r@rdegges.com**.

Please check your inbox to continue.

Then, depending on your "Password Reset Workflow" configuration, the user will see an email that looks like the following:


Password Reset

Randall Degges via stormpath.com

to me

3:37 PM (1 minute ago)

Forgot your password?

We've received a request to reset the password for this email address.

To reset your password please click on this link or cut and paste this URL into your browser (link expires in 24 hours):
http://localhost:5000/forgot/change?sptoken=7fdmzafiyChl747qIAlNusXvJ0fXBNVZ3zzi9HukBDqFD2cnQTdLjCJbpHaEyul83UqeZ1kAas

This link takes you to a secure page where you can change your password.

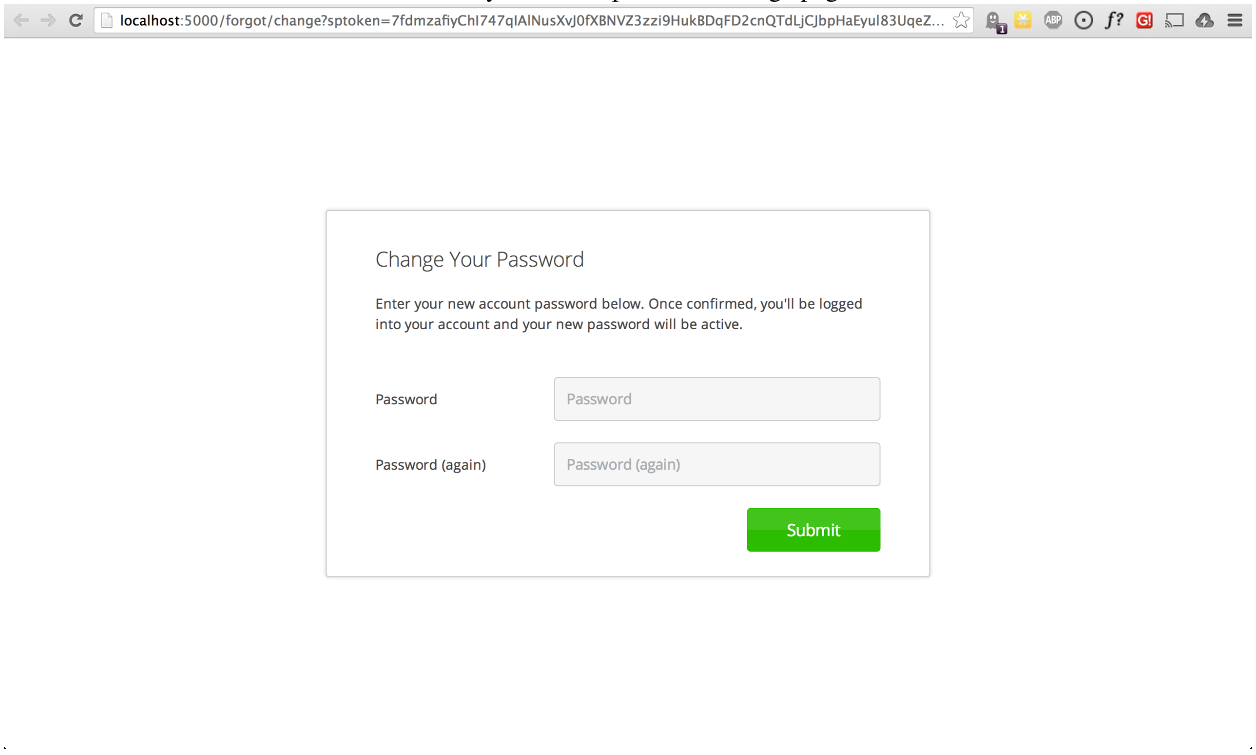
If you don't want to reset your password, please ignore this message. Your password will not be reset.

For general inquiries or to request support with your account, please email r@rdegges.com

----

This message was sent by Stormpath. Visit www.stormpath.com to learn more.
If you received this email by mistake or believe it is spam, please forward it to support@stormpath.com.

When a user clicks the link in their email, they'll reach a password change page that looks like this:


localhost:5000/forgot/change?sptoken=7fdmzafiyChl747qIAlNusXvJ0fXBNVZ3zzi9HukBDqFD2cnQTdLjCJbpHaEyul83UqeZ...

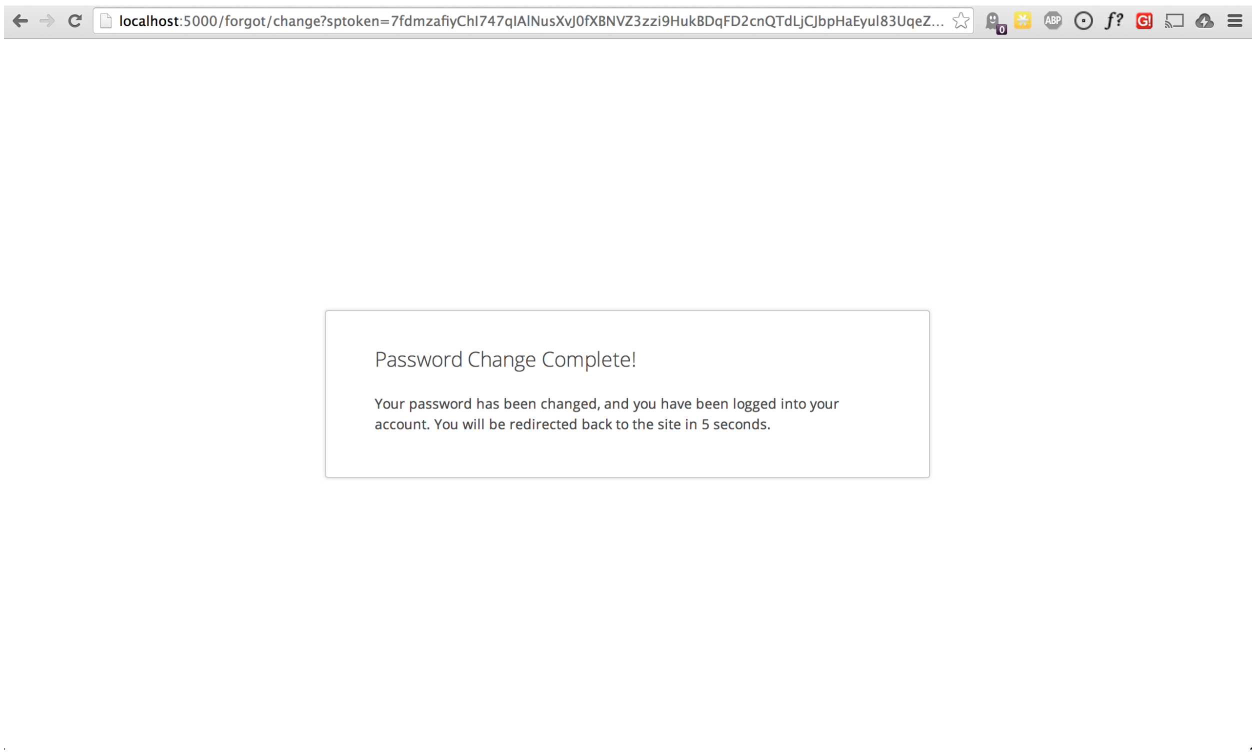
Change Your Password

Enter your new account password below. Once confirmed, you'll be logged into your account and your new password will be active.

Password

Password (again)

And lastly, once a user changes their password successfully, they'll be automatically logged into their account, then redirected to the main page of your site (whatever URL is set as `STORMPATH_REDIRECT_URL` in your configuration). They'll also be shown this page for a few seconds to let them know the change was successful:


localhost:5000/forgot/change?sptoken=7fdmzafiyChl747qlAlNusXvJ0fXBNVZ3zzi9HukBDqFD2cnQTdLjCJbpHaEyul83UqeZ...

Password Change Complete!

Your password has been changed, and you have been logged into your account. You will be redirected back to the site in 5 seconds.

Not bad, right?

Customization

Much like all other Flask-Stormpath features, the password reset feature is completely customizable.

You can easily change the password reset templates by modifying the following configuration variables, respectively:

- `STORMPATH_FORGOT_PASSWORD_TEMPLATE` - The template which is shown when a user clicks the "Forgot Password?" link on the login page.
- `STORMPATH_FORGOT_PASSWORD_EMAIL_SENT_TEMPLATE` - The template which is shown after a user has successfully requested a password reset.
- `STORMPATH_FORGOT_PASSWORD_CHANGE_TEMPLATE` - The template which is shown to a user after they've clicked the link in their email. This template allows the user to change their password.
- `STORMPATH_FORGOT_PASSWORD_COMPLETE_TEMPLATE` - The template which is shown after the user has successfully reset their account password.

If you'd like to override the default templates, you should take a look at the ones included with Flask-Stormpath here: https://github.com/stormpath/stormpath-flask/tree/master/flask_stormpath/templates/flask_stormpath and use these as a base for your own templates.

1.4.13 Use Facebook Login

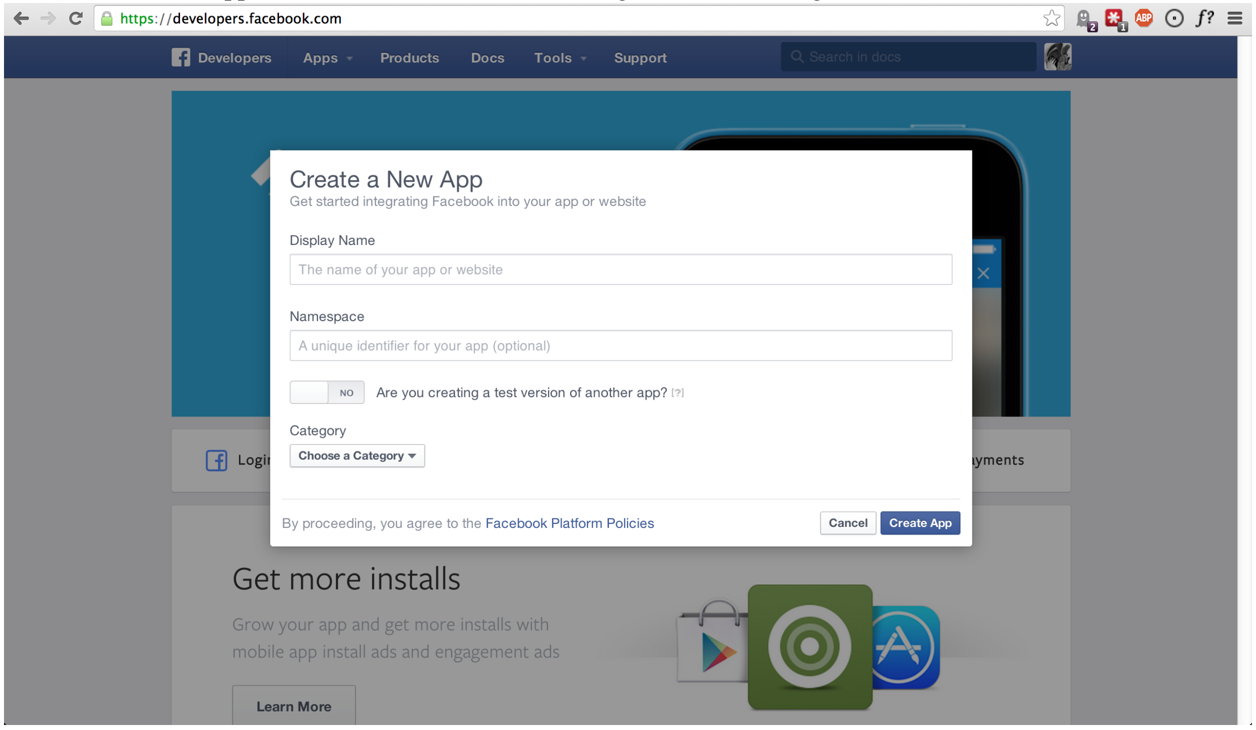
Now that we've covered the basics: let's add Facebook Login support to your app! Stormpath makes it very easy to support social login with Facebook.

In the next few minutes I'll walk you through *everything* you need to know to support Facebook login with your app.

Create a Facebook App

The first thing you need to do is log into the Facebook Developer Site and create a new Facebook App.

You can do this by visiting the Facebook Developer Site and click the “Apps” menu at the top of the screen, then select the “Create a New App” button. You should see something like the following:


https://developers.facebook.com
Developers
Apps
Products
Docs
Tools
Support
Search in docs

Create a New App
Get started integrating Facebook into your app or website

Display Name
The name of your app or website

Namespace
A unique identifier for your app (optional)

NO Are you creating a test version of another app? [?]

Category
Choose a Category

By proceeding, you agree to the Facebook Platform Policies

Cancel Create App

Get more installs
Grow your app and get more installs with mobile app install ads and engagement ads
Learn More

Go ahead and pick a “Display Name” (usually the name of your app), and choose a category for your app. Once you’ve done this, click the “Create App” button.

Specify Allowed URLs

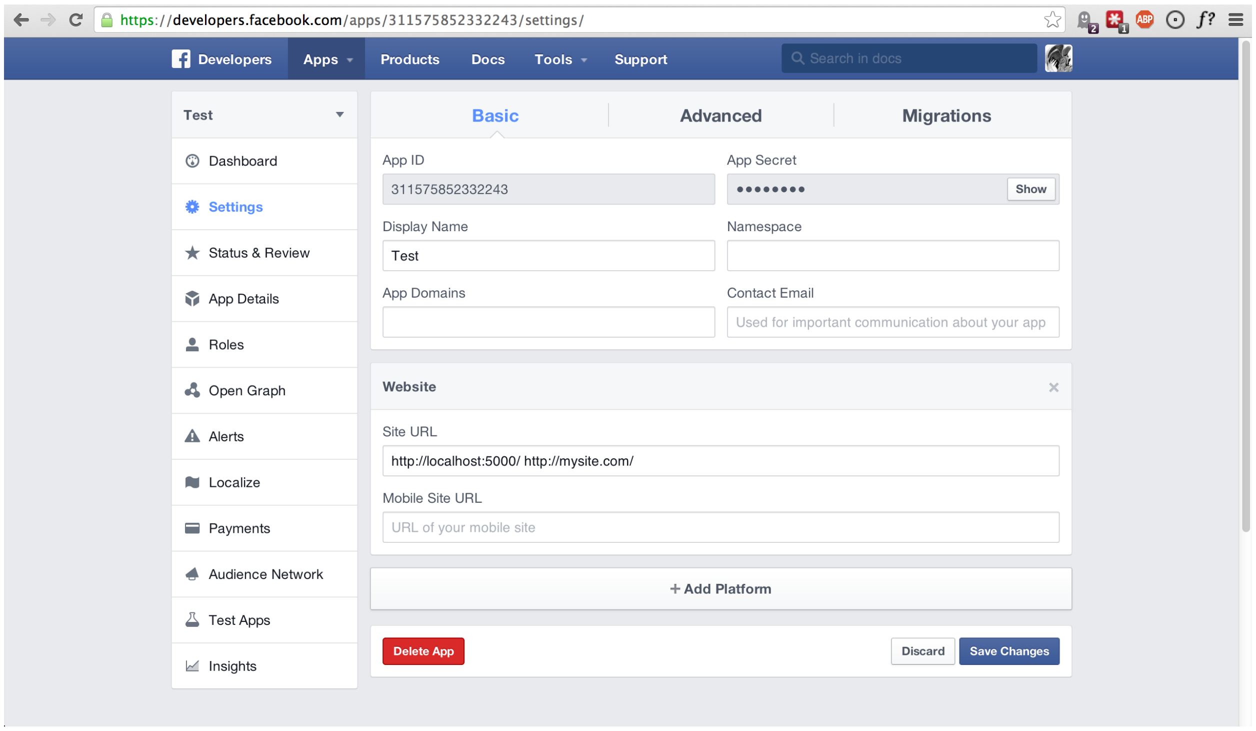
The next thing we need to do is tell Facebook what URLs we’ll be using Facebook Login from.

From the app dashboard page you’re on, click the “Settings” tab in the left menu, then click the “Add Platform” button near the bottom of the page. When prompted, select “Website” as your platform type.

In the “Site URL” box, enter your private and public root URLs. This should be something like `"http://localhost:5000"` or `"http://mysite.com"`. *If you want to allow Facebook Login from multiple URLs (local development, production, etc.) you can just click the “Add Platform” button again and enter another URL.*

Lastly, click the “Save Changes” button to save the changes.

Your settings should now look something like this:


https://developers.facebook.com/apps/311575852332243/settings/

App ID: 311575852332243

App Secret: ••••••••

Display Name: Test

Namespace:

App Domains:

Contact Email: Used for important communication about your app

Website

Site URL: http://localhost:5000/ http://mysite.com/

Mobile Site URL: URL of your mobile site

Configure Your Flask App

Now that we've created a new Facebook App and configured our URLs – we need to enter our Facebook App secrets into our Flask app so that Flask-Stormpath knows about them.

You can find your Facebook App ID and Secret on your App dashboard page, at the top of the screen.

In your app's config, you'll want to add the following settings (*don't forget to substitute in the proper credentials!*):

```
from os import environ

app.config['STORMPATH_ENABLE_FACEBOOK'] = True
app.config['STORMPATH_SOCIAL'] = {
    'FACEBOOK': {
        'app_id': environ.get('FACEBOOK_APP_ID'),
        'app_secret': environ.get('FACEBOOK_APP_SECRET'),
    }
}
```

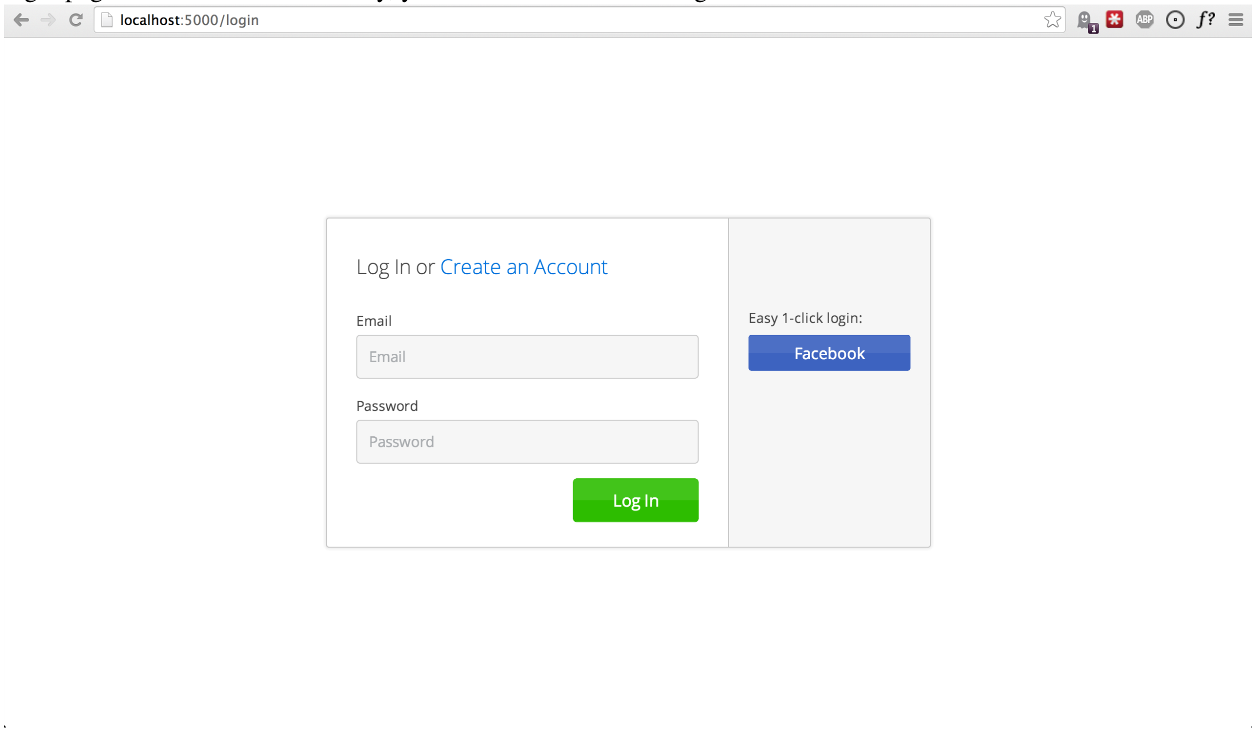
These two settings: `STORMPATH_ENABLE_FACEBOOK` and `STORMPATH_SOCIAL` work together to tell Flask-Stormpath to enable social login support for Facebook, as well as provide the proper credentials so things work as expected.

Note: We recommend storing your credentials in environment variables (as shown in the example above). Please don't hard code secret credentials into your source code!

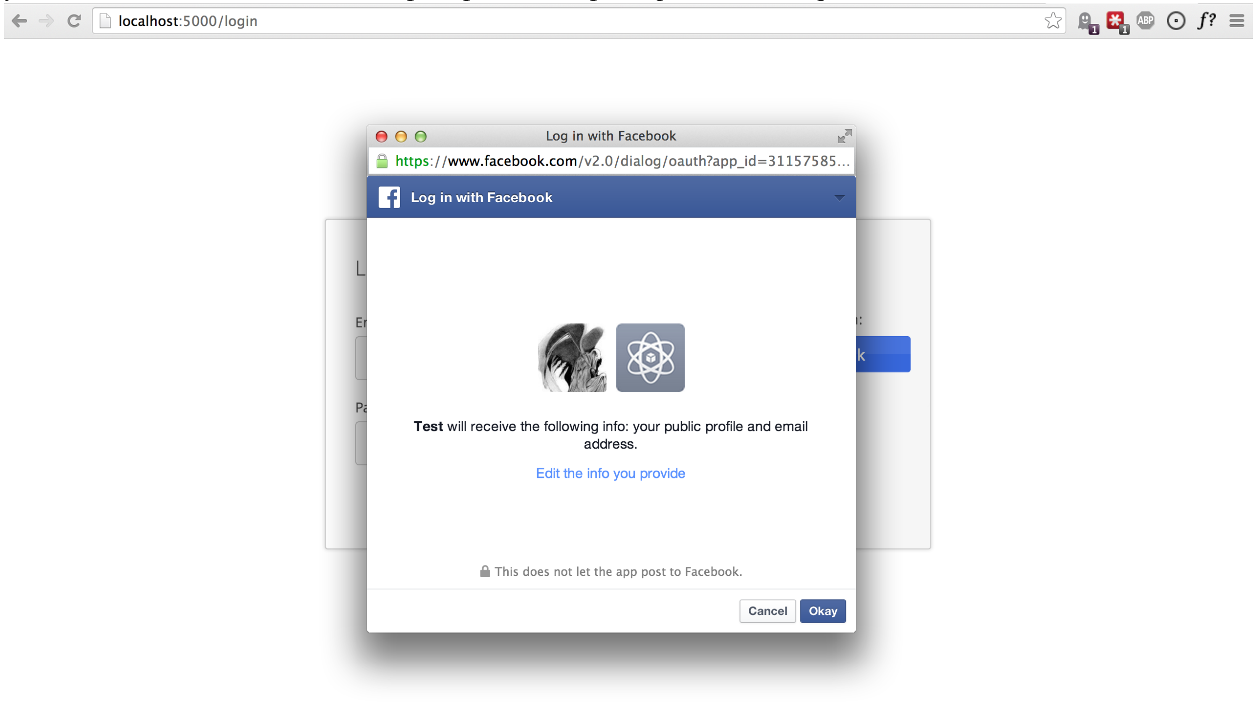
Test it Out

Now that you've plugged your Facebook credentials into Flask-Stormpath, social login should already be working!

Open your Flask app in a browser, and try logging in by visiting the login page (`/login`). If you're using the default login page included with this library, you should see the following:



You now have a fancy new Facebook enabled login button! Try logging in! When you click the new Facebook button you'll be redirected to Facebook, and prompted to accept the permissions requested:



localhost:5000/login
Log in with Facebook
https://www.facebook.com/v2.0/dialog/oauth?app_id=31157585...
Log in with Facebook

Test will receive the following info: your public profile and email address.

Edit the info you provide
This does not let the app post to Facebook.
Cancel
Okay

After accepting permissions, you'll be immediately redirected back to your website at the URL specified by `STORMPATH_REDIRECT_URL` in your app's settings.

Simple, right?!

1.4.14 Use Google Login

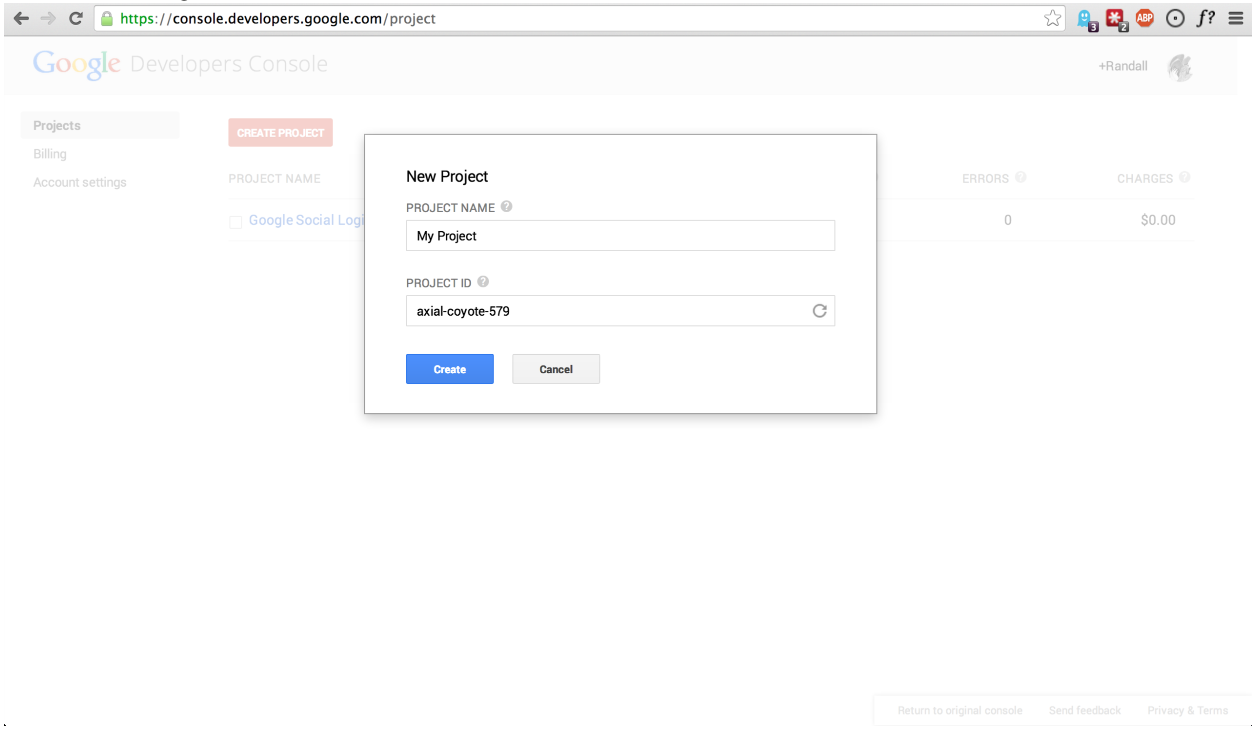
Google Login is incredibly popular – let's enable it!

In the next few minutes I'll walk you through *everything* you need to know to support Google login with your app.

Create a Google Project

The first thing you need to do is log into the Google Developer Console and create a new Google Project.

You can do this by visiting the Developer Console and clicking the "Create Project" button. You should see something like the following:



New Project

PROJECT NAME: My Project

PROJECT ID: axial-coyote-579

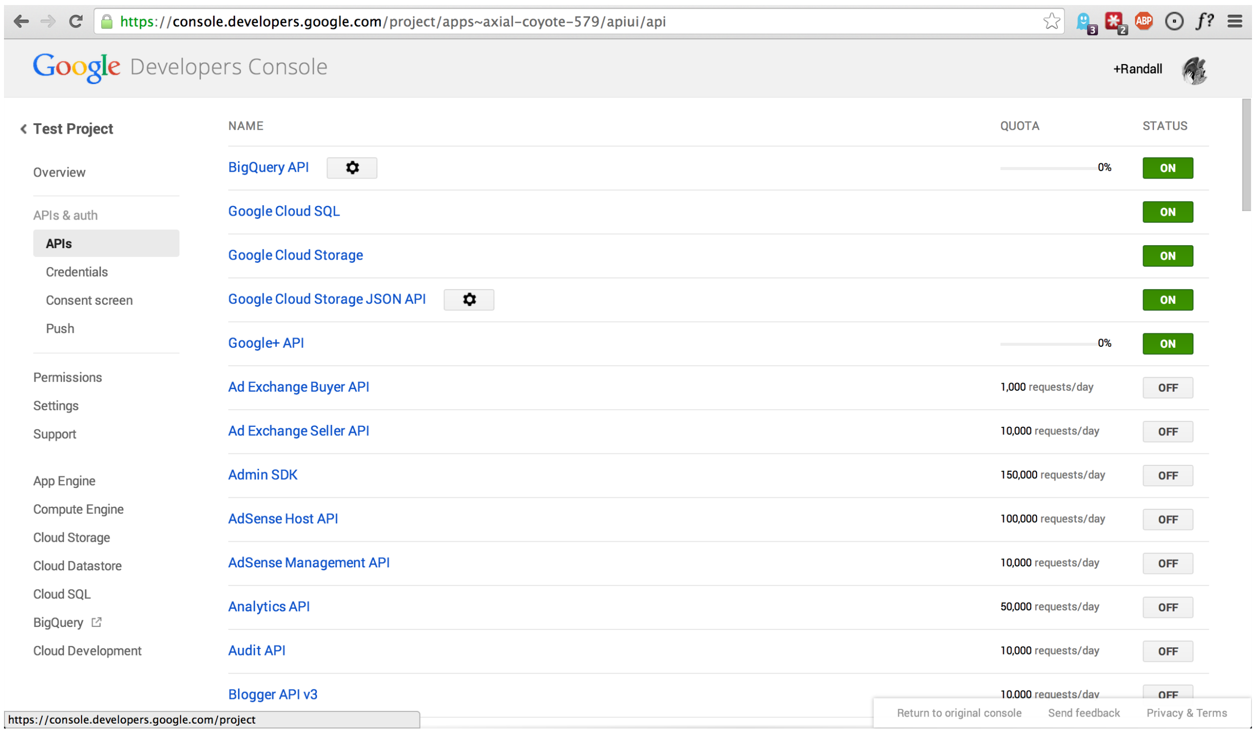
Go ahead and pick a "Project Name" (usually the name of your app), and (*optionally*) a "Project ID".

Enable Google Login

Now that you've got a Google Project – let's enable Google Login. The way Google Projects work is that you have to selectively enable what functionality each Project needs.

From your Console Dashboard click on your new Project, then in the side panel click on the "APIs & auth" menu option.

Now, scroll through the API list until you see "Google+ API", then click the "OFF" button next to it to enable it. You should now see the "Google+ API" as "ON" in your API list:


https://console.developers.google.com/project/apps~axial-coyote-579/apiui/api

Google Developers Console +Randall

‹ Test Project

Overview

APIs & auth
- APIs
- Credentials
- Consent screen
- Push

Permissions
Settings
Support

App Engine
Compute Engine
Cloud Storage
Cloud Datastore
Cloud SQL
BigQuery
Cloud Development

NAME	QUOTA	STATUS
BigQuery API	0%	ON
Google Cloud SQL		ON
Google Cloud Storage		ON
Google Cloud Storage JSON API		ON
Google+ API	0%	ON
Ad Exchange Buyer API	1,000 requests/day	OFF
Ad Exchange Seller API	10,000 requests/day	OFF
Admin SDK	150,000 requests/day	OFF
AdSense Host API	100,000 requests/day	OFF
AdSense Management API	10,000 requests/day	OFF
Analytics API	50,000 requests/day	OFF
Audit API	10,000 requests/day	OFF
Blogger API v3	10,000 requests/day	OFF

https://console.developers.google.com/project

Return to original console Send feedback Privacy & Terms

Create OAuth Credentials

The next thing we need to do is create a new OAuth client ID. This is what we'll use to handle user login with Google.

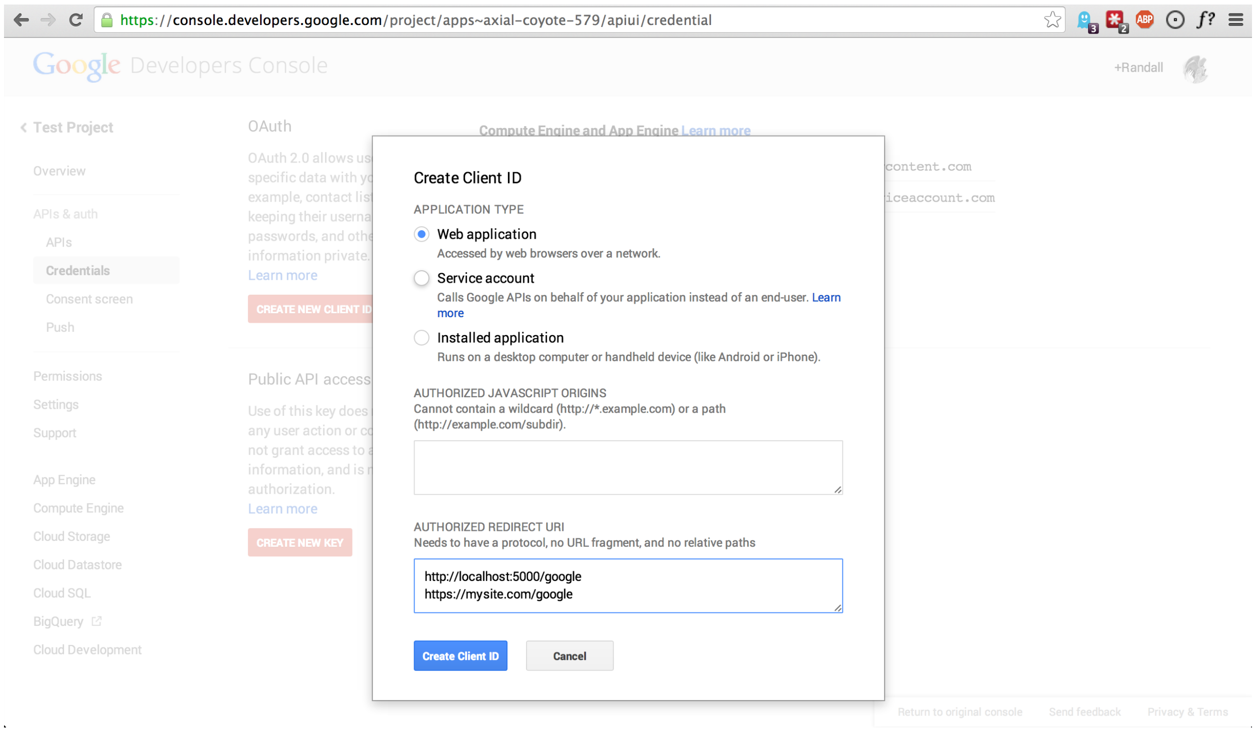
From your Console Dashboard click the "APIs & auth" menu, then click on the "Credentials" sub-menu.

You should see a big red button labeled "Create New Client ID" near the top of the page – click that.

You'll want to do several things here:

1. Select "Web application" for your "Application Type".
2. Remove everything from the "Authorized Javascript Origins" box.
3. Add the URL of your site (both publicly and locally) into the "Authorized Redirect URI" box, with the `/google` suffix. This tells Google where to redirect users after they've logged in with Google.

In the end, your settings should look like this:



Create Client ID

APPLICATION TYPE

[✓ Web application]
Accessed by web browsers over a network.

Service account
Calls Google APIs on behalf of your application instead of an end-user. Learn more

Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

AUTHORIZED JAVASCRIPT ORIGINS
Cannot contain a wildcard (http://*.example.com) or a path (http://example.com/subdir).

AUTHORIZED REDIRECT URI
Needs to have a protocol, no URL fragment, and no relative paths

```
http://localhost:5000/google
https://mysite.com/google
```

Once you've specified your settings, go ahead and click the "Create Client ID" button.

Lastly, you'll want to take note of your "Client ID" and "Client Secret" variables that should now be displayed on-screen. We'll need these in the next step.

Configure Your Flask App

Now that we've created a new Google Project and generated OAuth secrets – we can now enter these secrets into our Flask app so that Flask-Stormpath knows about them.

In your app's config, you'll want to add the following settings (*don't forget to substitute in the proper credentials!*):

```python
from os import environ

app.config['STORMPATH_ENABLE_GOOGLE'] = True
app.config['STORMPATH_SOCIAL'] = {
    'GOOGLE': {
        'client_id': environ.get('GOOGLE_CLIENT_ID'),
        'client_secret': environ.get('GOOGLE_CLIENT_SECRET'),
    }
}
```

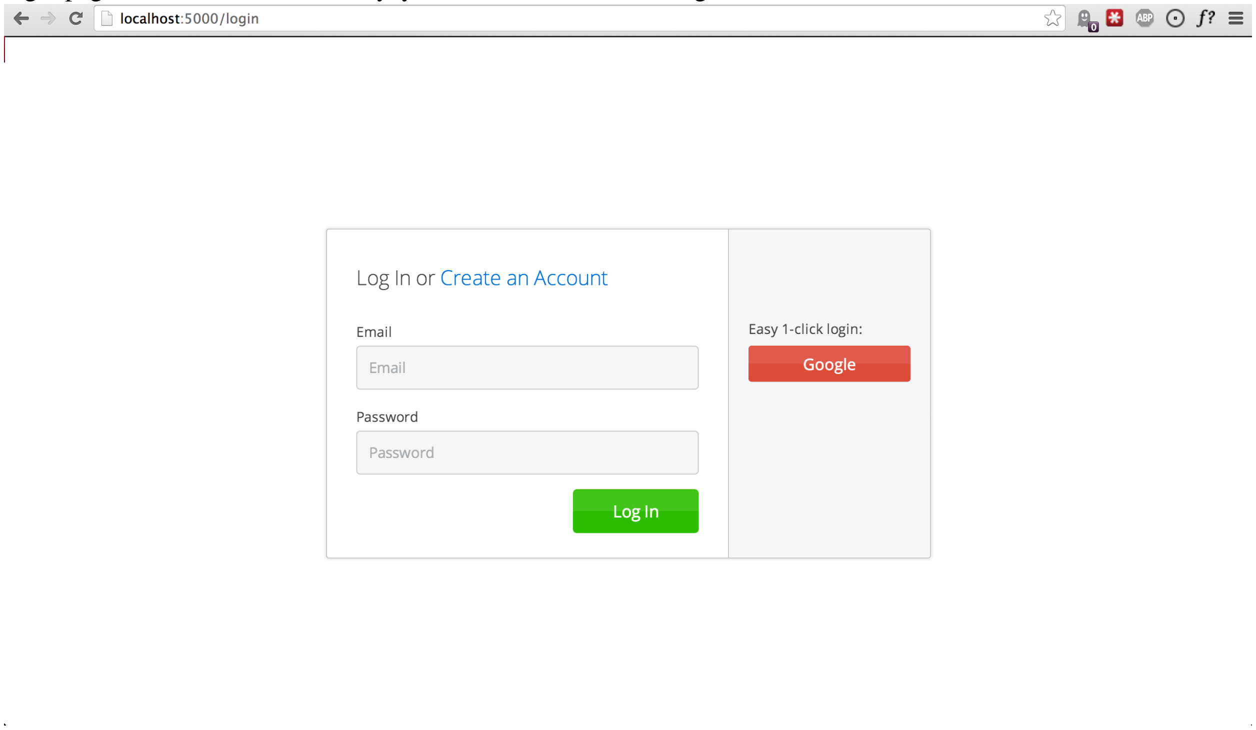
These two settings: `STORMPATH_ENABLE_GOOGLE` and `STORMPATH_SOCIAL` work together to tell Flask-Stormpath to enable social login support for Google, as well as provide the proper credentials so things work as expected.

Note: We recommend storing your credentials in environment variables (as shown in the example above). Please don't hard code secret credentials into your source code!

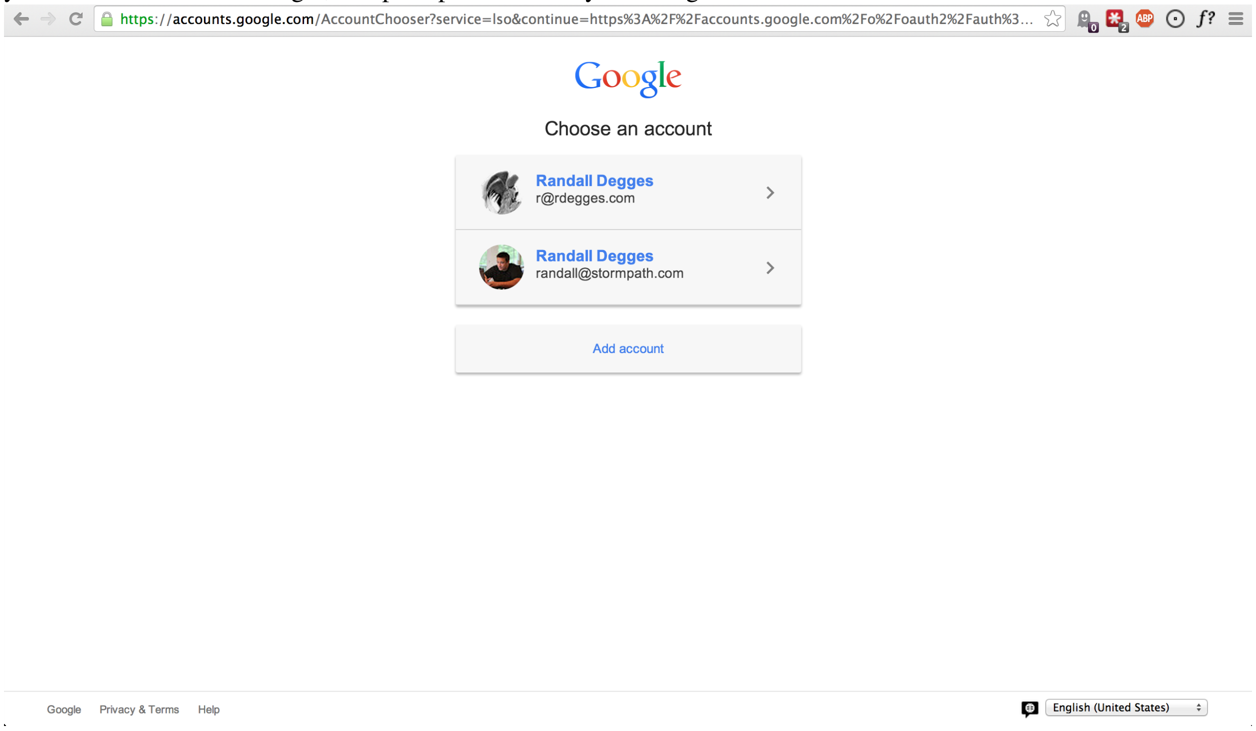
Test it Out

Now that you've plugged your Google credentials into Flask-Stormpath, social login should already be working!

Open your Flask app in a browser, and try logging in by visiting the login page (`/login`). If you're using the default login page included with this library, you should see the following:



You now have a fancy new Google enabled login button! Try logging in! When you click the new Google button you'll be redirected to Google, and prompted to select your Google account:



After selecting your account you'll then be prompted to accept any permissions, then immediately redirected back to your website at the URL specified by `STORMPATH_REDIRECT_URL` in your app's settings.

Simple, right?!

1.4.15 Enable Caching

The best kind of websites are fast websites. Flask-Stormpath includes built-in support for caching. You can currently use either:

- A local memory cache (*default*).
- A memcached cache.
- A redis cache.

All can be easily configured using configuration variables.

There are several configuration settings you can specify to control caching behavior.

Here's an example which shows how to enable caching with redis:

```
from stormpath.cache.redis_store import RedisStore


app = Flask(__name__)
app.config['STORMPATH_CACHE'] = {
    'store': RedisStore,
    'store_opts': {
        'host': 'localhost',
        'port': 6379
    }
}

stormpath_manager = StormpathManager(app)
```

Here's an example which shows how to enable caching with memcached:

```
from stormpath.cache.memcached_store import MemcachedStore


app = Flask(__name__)
app.config['STORMPATH_CACHE'] = {
    'store': MemcachedStore,
    'store_opts': {
        'host': 'localhost',
        'port': 11211,
    }
}

stormpath_manager = StormpathManager(app)
```

If no cache is specified, the default, `MemoryStore`, is used. This will cache all resources in local memory.

For a full list of options available for each cache backend, please see the official Caching Docs in our Python library.

1.5 Getting Help

Have a question you can't find an answer to? Things not working as expected? We can help!

All of the official Stormpath client libraries (including this one!) are officially supported by Stormpath's incredibly amazing-and-hip support team!

If you have a question, or need in-depth technical help, you can drop us an email anytime: support@stormpath.com

If you visit our website (https://stormpath.com/), you can also click the "Chat with us!" button near the bottom right of the page to chat with us live, in real-time!

And lastly, we're always available via Twitter as well! We're @gostormpath on Twitter.

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

This part of the documentation documents all the public classes, functions, and API details in Flask-Stormpath. This documentation is auto generated, and is always a good up-to-date reference.

2.1.1 Configuration

class `flask.ext.stormpath.`**`StormpathManager`**(*app=None*)

This object is used to hold the settings used to communicate with Stormpath. Instances of `StormpathManager` are not bound to specific apps, so you can create one in the main body of your code and then bind it to your app in a factory function.

`client`()

Lazily load the Stormpath Client object we need to access the raw Stormpath SDK.

`application`()

Lazily load the Stormpath Application object we need to handle user authentication, etc.

`login_view`()

Return the user's Flask-Login login view, behind the scenes.

static **`load_user`**(*account_href*)

Given an Account href (a valid Stormpath Account URL), return the associated User account object (or None).

Returns The User object or None.

2.1.2 Models

class `flask.ext.stormpath.`**`User`**(*client*, *href=None*, *properties=None*, *query=None*, *expand=None*)

The base User object.

This can be used as described in the Stormpath Python SDK documentation: https://github.com/stormpath/stormpath-sdk-python

`get_id`()

Return the unique user identifier (in our case, the Stormpath resource href).

is_active()
A user account is active if, and only if, their account status is 'ENABLED'.

is_anonymous()
We don't support anonymous users, so this is always False.

is_authenticated()
All users will always be authenticated, so this will always return True.

classmethod from_login(*login*, *password*)
Create a new User class given a login (*email* or *username*), and password.

If something goes wrong, this will raise an exception – most likely – a *StormpathError* (flask_stormpath.StormpathError).

2.1.3 Decorators

flask.ext.stormpath.**groups_required**(*groups*, *all=True*)
This decorator requires that a user be part of one or more Groups before they are granted access.

Parameters

- **groups** (*list*) – (required) A list of Groups to restrict access to. A group can be:
  - A Group object.
  - A Group name (as a string).
  - A Group href (as a string).
- **all** (*bool*) – (optional) Should we ensure the user is a member of every group listed? Default: True. If this is set to False, we'll let the user into the view if the user is part of at least one of the specified groups.

Usage:

```
@groups_required(['admins', 'developers'])
def private_view():
    '''Only admins and developers will be able to visit this page.'''
    return 'hi!'
```

flask.ext.stormpath.**login_required**(*func*)
If you decorate a view with this, it will ensure that the current user is logged in and authenticated before calling the actual view. (If they are not, it calls the `LoginManager.unauthorized` callback.) For example:

```
@app.route('/post')
@login_required
def post():
    pass
```

If there are only certain times you need to require that your user is logged in, you can do so with:

```
if not current_user.is_authenticated:
    return current_app.login_manager.unauthorized()
```

...which is essentially the code that this function adds to your views.

It can be convenient to globally turn off authentication when unit testing. To enable this, if the application configuration variable *LOGIN_DISABLED* is set to *True*, this decorator will be ignored.

Parameters **func** (*function*) – The view function to decorate.

2.1.4 Request Context

`flask.ext.stormpath.user()`

Acts as a proxy for a werkzeug local. Forwards all operations to a proxied object. The only operations not supported for forwarding are right handed operands and any kind of assignment.

Example usage:

```python
from werkzeug.local import Local
l = Local()

# these are proxies
request = l('request')
user = l('user')


from werkzeug.local import LocalStack
_response_local = LocalStack()

# this is a proxy
response = _response_local()
```

Whenever something is bound to l.user / l.request the proxy objects will forward all operations. If no object is bound a `RuntimeError` will be raised.

To create proxies to `Local` or `LocalStack` objects, call the object as shown above. If you want to have a proxy to an object looked up by a function, you can (as of Werkzeug 0.6.1) pass a function to the `LocalProxy` constructor:

```python
session = LocalProxy(lambda: get_current_request().session)
```

Changed in version 0.6.1: The class can be instanciated with a callable as well now.

CHAPTER 3

Additional Notes

This part of the documentation covers changes between versions and upgrade information, to help you migrate to newer versions of Flask-Stormpath easily.

Flask-Stormpath is made available under the Apache License, Version 2.0. In short, you can do pretty much whatever you want!

3.1 Change Log

All library changes, in descending order.

3.1.1 Version 0.4.6

Released on September 6, 2016.

- Raising a clear error message when Flask-Stormpath fails to find the Stormpath Application. This makes the debugging experience simpler for new developers.
- Replacing deprecated `from flask.ext.xxx` import syntax to work with new Flask release.
- Making error strings UTF-8.
- Handling errors in templates in a better way.
- Adding Python 3 support.
- Upgrading Stormpath dependency to latest release.
- Upgrading oauth2client dependency to latest release.
- Upgrading blinker dependency to latest release.
- Upgrading Flask-Login dependency to latest release.

3.1.2 Version 0.4.5

Released on April 22, 2016.

- Upgrading facebook-sdk dependency.
- Providing Facebook support for hosts which require outbound HTTP proxying.
- Fixing styling issue with CSS forms.

3.1.3 Version 0.4.4

Released on August 31, 2015.

- Upgrading the Stormpath dependency to the latest release.

3.1.4 Version 0.4.3

Released on August 25, 2015.

- Adding in some test fixes.
- Adding in signals for user creation, updating, and deleting.
- Upgrading the Stormpath library to the latest release.

3.1.5 Version 0.4.2

Released on June 12, 2015.

- Adding notes about `TESTING = True` for clarity.
- Fixing error handling error in the 'forgot password' feature. If a user tried to change their password to something that didn't match the password strength rules, they'd get a 500.

3.1.6 Version 0.4.1

Released on May 19, 2015.

- Adding 'profile' scope as a default requested scope for Google login. This allows us to get a user's first and last name in addition to their email address. Thanks to @stauffec for the contribution!

3.1.7 Version 0.4.0

Released on April 15, 2015.

- Adding new setting: `STORMPATH_REGISTRATION_REDIRECT_URL`. This lets users specify where they'd like to redirect a *newly registered user*.

3.1.8 Version 0.3.9

Released on March 27, 2015.

- Removing python 3 compatibility (*due to pip bug with Facebook SDK*). This will be back soon once we find a workaround.

3.1.9 Version 0.3.8

Released on March 26, 2015.

- Making the library 100% python 3 compatible!
- Fixing issue with error messages being flashed incorrectly.

3.1.10 Version 0.3.7

Released on March 2, 2015.

- Fixing exception handling error during password reset when an invalid email address is entered. Thanks @ro-engraft for the report!

3.1.11 Version 0.3.6

Released on February 16, 2015.

- Fixing minor issues in error handling in our registration and login views.
- Adding tests for error handling in our registration and login views.

3.1.12 Version 0.3.5

Released on February 11, 2015.

- Upgrading dependencies.

3.1.13 Version 0.3.4

Released on February 11, 2015.

- Upgrading our Stormpath python dependency. Lots of bugfixes / improvements included.
- Allowing users to customize the base Stormpath template via a new setting: `STORMPATH_BASE_TEMPLATE`.

3.1.14 Version 0.3.3

Released on January 28, 2015.

- Upgrading our Stormpath python dependency. This gets us lots of bugfixes / speed improvements.

3.1.15 Version 0.3.2

Released on January 27, 2015.

- Fixing issue with singletons. We were previously NOT using a client singleton, which means in-memory caching would not work :(

3.1.16 Version 0.3.1

Released on December 23, 2014.

- Fixing critical issue where version info caused startup errors. The resolution is to remove dynamic versioning that depends on `setup.py`.

3.1.17 Version 0.3.0

Released on December 8, 2014.

- Fixing minor issue with user agent.
- Updating stormpath dependency to latest release.
- Adding support for caching (*with local memory, memcached, and redis*).
- Adding caching docs.
- Dynamically handling library versions.

3.1.18 Version 0.2.9

Released on November 7, 2014.

- Adding support for Google login's *hd* attribute.

3.1.19 Version 0.2.8

Released on September 20, 2014.

- Fixing bug in forgot() view – the user object passed to the template wasn't an actual user object.

3.1.20 Version 0.2.7

Released on September 10, 2014.

- Adding the ability to set a user's status when calling `User.create()`.

3.1.21 Version 0.2.6

Released on July 14, 2014.

- Adding in easy 'Password Reset' functionality. If a developer enables this functionality, users can easily reset their passwords securely. This feature is disabled by default.

3.1.22 Version 0.2.5

Released on June 24, 2014.

- Fixing bug in built-in registration view. When new users registered, the first name would be inserted into the last name field.

3.1.23 Version 0.2.4

Released on June 16, 2014.

- Fixing bug which affected the login page when *STORMPATH_ENABLE_REGISTRATION* was disabled.
- Fixing bug which affected the registration page when *STORMPATH_ENABLE_LOGIN* was disabled.

3.1.24 Version 0.2.3

Released on May 22, 2014.

- Adding a proper user agent.

3.1.25 Version 0.2.2

Released on May 20, 2014.

- Adding new setting: `STORMPATH_COOKIE_DOMAIN`. This allows users to specify which domain(s) the session cookie will be good for.
- Adding new setting: `STORMPATH_COOKIE_DURATION`. This allows users to specify how long a session will last (as a `timedelta` object).
- Adding docs on expiring sessions / cookies.

3.1.26 Version 0.2.1

Released on May 16, 2014.

- Fixing bug in package: templates weren't being included.

3.1.27 Version 0.2.0

Released on May 14, 2014.

- Adding customizable user settings.
- Adding support for social login via Gacebook.
- Adding support for social login via Facebook.
- Adding an automatic logout view.
- Adding an automatic login view.
- Adding an automatic registration view.
- Adding built-in routes for logout / login / register.
- Adding customizable registration / login pages.
- Adding built in templates for registration and login (with social included).
- Adding new documentation.

3.1.28 Version 0.1.0

Released on March 26, 2014.

- Adding a simple way to create new user accounts via `User.create()`.
- Adding documentation for new `User.create()` method.
- Adding a groups_required decorator, which makes it easy to assert Group membership in views.
- Adding docs for new groups_required decorator.

- Using the lastest Python SDK as a dependency.

3.1.29 Version 0.0.1

Released on February 19, 2014.

- First release!
- Basic functionality.
- Basic docs.
- Lots to do!

3.2 Upgrade Guide

This page contains specific upgrading instructions to help you migrate between Flask-Stormpath releases.

3.2.1 Version 0.4.4 -> Version 0.4.5

No changes needed!

3.2.2 Version 0.4.3 -> Version 0.4.4

No changes needed!

3.2.3 Version 0.4.2 -> Version 0.4.3

No changes needed!

3.2.4 Version 0.4.1 -> Version 0.4.2

No changes needed!

3.2.5 Version 0.4.0 -> Version 0.4.1

No changes needed!

3.2.6 Version 0.3.9 -> Version 0.4.0

No changes needed!

3.2.7 Version 0.3.8 -> Version 0.3.9

No changes needed!

3.2.8 Version 0.3.7 -> Version 0.3.8

No changes needed!

3.2.9 Version 0.3.6 -> Version 0.3.7

No changes needed!

3.2.10 Version 0.3.5 -> Version 0.3.6

No changes needed!

3.2.11 Version 0.3.4 -> Version 0.3.5

No changes needed!

3.2.12 Version 0.3.3 -> Version 0.3.4

No changes needed!

3.2.13 Version 0.3.2 -> Version 0.3.3

No changes needed!

3.2.14 Version 0.3.1 -> Version 0.3.2

No changes needed!

3.2.15 Version 0.3.0 -> Version 0.3.1

No changes needed!

3.2.16 Version 0.2.9 -> Version 0.3.0

No changes needed!

3.2.17 Version 0.2.8 -> Version 0.2.9

No changes needed!

3.2.18 Version 0.2.7 -> Version 0.2.8

No changes needed!

3.2.19 Version 0.2.6 -> Version 0.2.7

No changes needed!

3.2.20 Version 0.2.5 -> Version 0.2.6

No changes needed!

3.2.21 Version 0.2.4 -> Version 0.2.5

No changes needed!

3.2.22 Version 0.2.3 -> Version 0.2.4

No changes needed!

3.2.23 Version 0.2.2 -> Version 0.2.3

No changes needed!

3.2.24 Version 0.2.1 -> Version 0.2.2

No changes needed!

3.2.25 Version 0.2.0 -> Version 0.2.1

No changes needed!

3.2.26 Version 0.1.0 -> Version 0.2.0

Version 0.2.0 is a feature release which includes a vast amount of library improvements.

In order to make a successful migration, you don't need to make any changes to your existing application.

3.2.27 Version 0.0.1 -> Version 0.1.0

No changes needed!

Python Module Index