Cookiecutter SaaS Documentation

Release 0.1

Jannis Gebauer

Apr 11, 2017

Contents

Contents:

CHAPTER 1

Developing locally

Prereqiuisites

Cookiecutter SaaS uses Docker for development and production, make sure Docker and Docker Compose are installed. If you are planning to push your project to production (you should), you'll also need to install Docker Machine.

On macOS

Follow the Docker for Mac installation instructions. This will install Docker, Docker Compose and Docker Machine on your Mac.

Windows

Follow the Docker for Windows installation instructions. This will install Docker, Docker Compose and Docker Machine on your Windows machine.

Linux

First, you need to install the Docker engine, follow the Docker on Linux installation instructions. Next, you need to install Docker Compose and install Docker Machine.

Make sure everything is installed correctly by running:

```
docker --version
docker-compose --version
docker-machine --version
```

Running the Project

Before running the project for the first time, you need to build it.

Start the build process with:

```
docker-compose -f dev.yml build
```

This tells Docker Compose to:

- pull all required base images from Docker Hub
- install all python dependencies via pip (and npm if you are going to using react)
- copy utility scripts into the image
- set up all required environment variables

This is going to take a while the first time you run it. Subsequent builds will be a lot faster because Docker caches each build step.

While you wait, check out the Third Party Services section. During development, in order to test subscriptions, we just need the Stripe and Octobat keys to be set. Create an account for both services and copy the keys to `config/settings/local.py`.

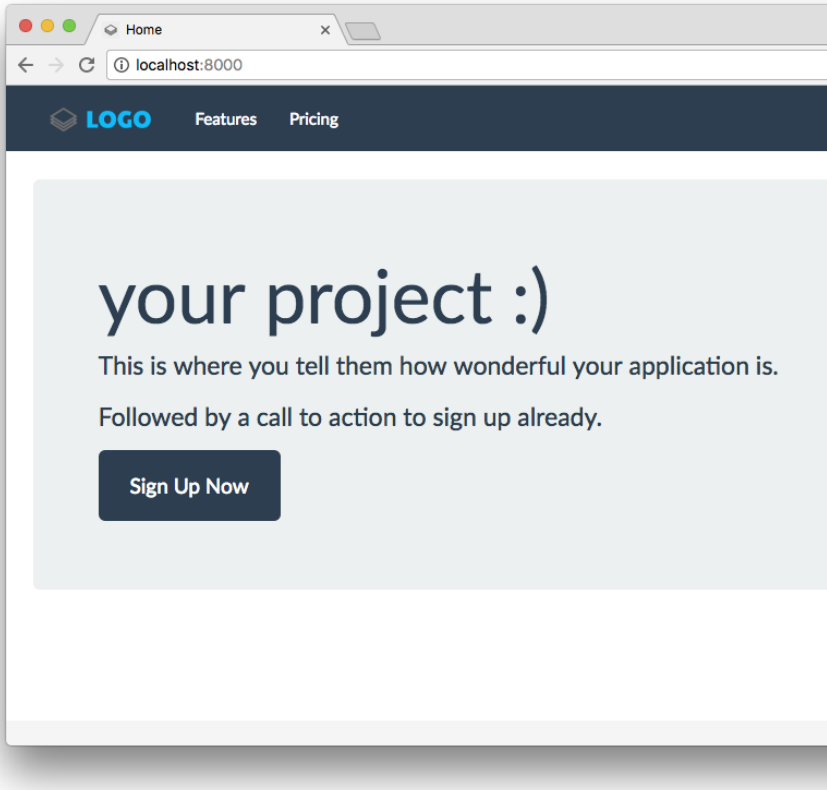
Once the build process is completed, run the project with:

```
docker-compose -f dev.yml up
```

You should see a lot of log messages flying by as each container is initializing itself. Look at the logs.

```
django_1    | Django version 1.10, using settings 'config.settings.local'
django_1    | Development server is running at http://0.0.0.0:8000/
django_1    | Using the Werkzeug debugger (http://werkzeug.pocoo.org/)
django_1    | Quit the server with CONTROL-C.
django_1    |  * Debugger is active!
django_1    |  * Debugger pin code: 241-491-852
```

As soon as you see a log similar to this, your project is ready.



Fire up your browser and go to http://localhost:8000.

If you want, you can now create a super user. Open up a second terminal and run:

```
docker-compose -f dev.yml createsuperuser
```

Now, go ahead and write your app. You might also want to check out the things to change section sooner or later.

CHAPTER 2

Deploying to Production

This is still a work in progress, see #35

In order to deploy Cookiecutter SaaS to production, you need to buy the production addon.

1.) Create a Server with Docker Machine

```
docker-machine create <name> --driver=digitalocean --digitalocean-access-token=<TOKEN>
↪ --digitalocean-region=fra1


docker-machine ip {{ cookiecutter.project_slug }}
```

2.) Set up DNS Records

Before we start to deploy our project to production, we need to set up the DNS records.

3.) Start the Stack

Start it

```
docker-compose -f prod.yml up -d
```

Take a look at the logs

```
docker-compose -f prod.yml logs
```

Once you see a constant stream of messages from `django` and `django-failover` (this is the health check), your app is ready.

Now, create a superuser:

```
docker-compose -f prod.yml run django python manage.py createsuperuser
```

Zero Downtime Deployments

Todo

CHAPTER 3

Scaling

Todo, see #42

CHAPTER 4

Subscriptions

Todo, see #36

CHAPTER 5

React

Todo, see #34

CHAPTER 6

Write Your App

This document is still a work in progress, see #44

Well, this is really up to you :)

Quickstart

Todo

- Once a user logs in, he/she is redirected to `/app`
- This loads the `DashboardView` in `projectname/app/views.py` with the template in `projectname/templates/app/dashboard.html`

CHAPTER 7

Third Party Services

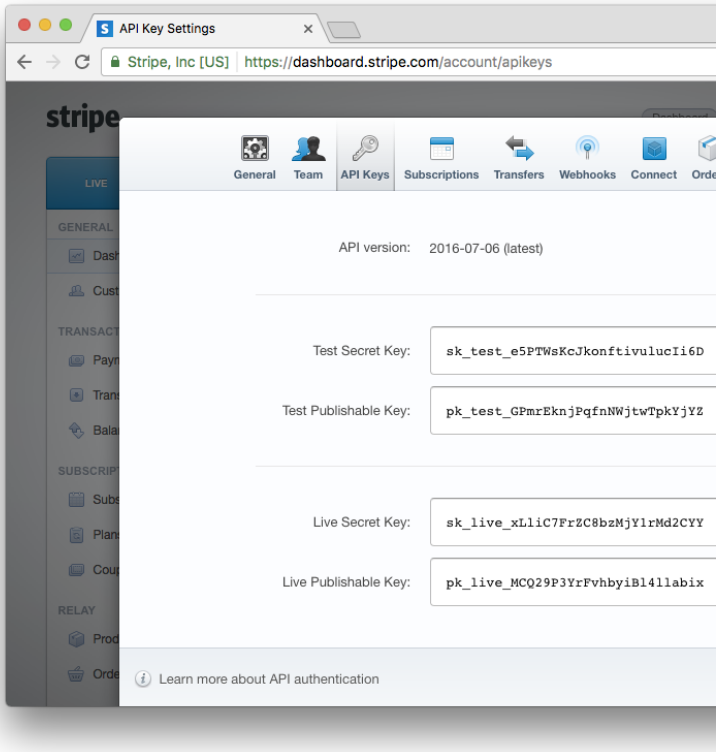
This document is still a work in progress, see #43.

Stripe

Todo

Used in: `development`, `production`

1. Go to stripe.com and create a new account, or log in to an existing one.
2. Click on your profile picture in the upper right corner and then on `account settings`



3. Select the `API Keys` tab and copy the test and live keys.
4. Test keys for `development` are going into `config/settings/local.py`:

- Test Secret Key goes into `PINAX_STRIPE_SECRET_KEY`
- Test Publishable Key goes into `PINAX_STRIPE_PUBLIC_KEY`

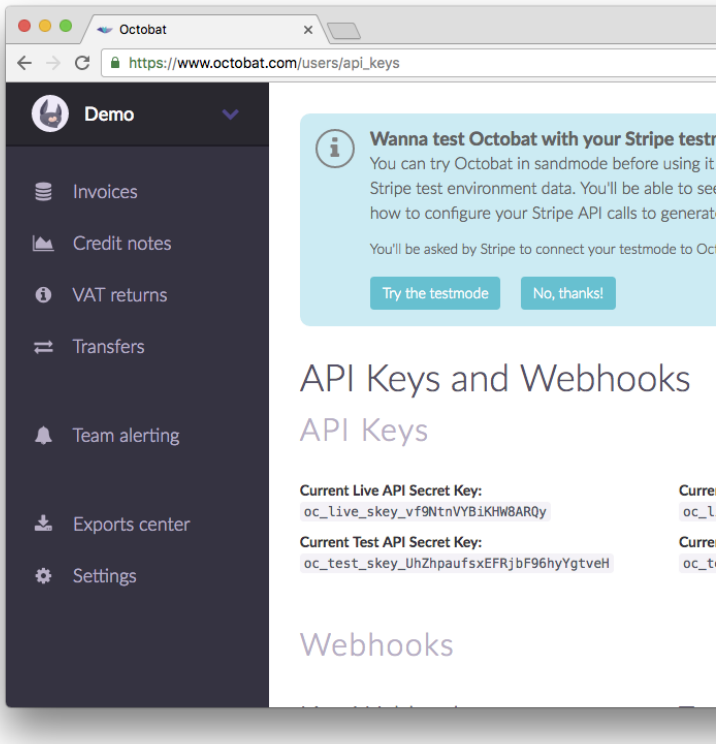
5. Live keys for `production` are going into `.env`. *If you are in beta, you can use your test keys here too.*
   - Live Secret Key goes into `PINAX_STRIPE_SECRET_KEY`
   - Live Publishable Key goes into `PINAX_STRIPE_PUBLIC_KEY`

Octobat

Todo, see #43

Used in: `development`, `production`

1. Go to www.octobat.com and create a new account, or log in to an existing one.
2. Click on the chevron in the upper left corner and then on `API Keys & Webhooks`



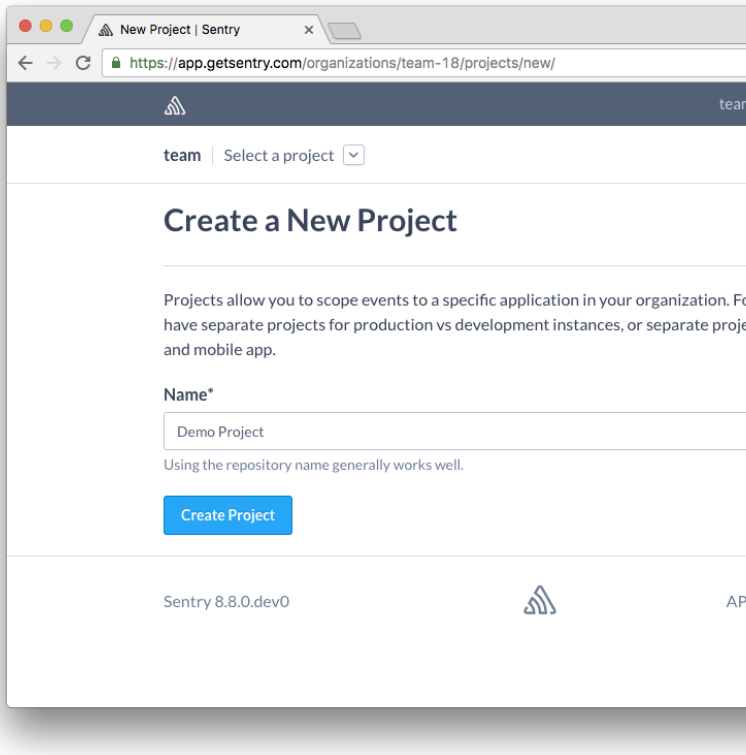
3. Select the `API Keys` tab and copy the test and live keys.
4. Test keys for `development` are going into `config/settings/local.py`:
   - Current Test API Secret Key goes into `OCTOBAT_PRIVATE_KEY`
   - Current Live API Publishable Key goes into `OCTOBAT_PUBLIC_KEY`
5. Live keys for `production` are going into `.env`. *If you are in beta, you can use your test keys here too.*
   - Current Live API Secret Key goes into `OCTOBAT_PUBLIC_KEY`
   - Current Live API Publishable Key goes into `OCTOBAT_PRIVATE_KEY`

Sentry

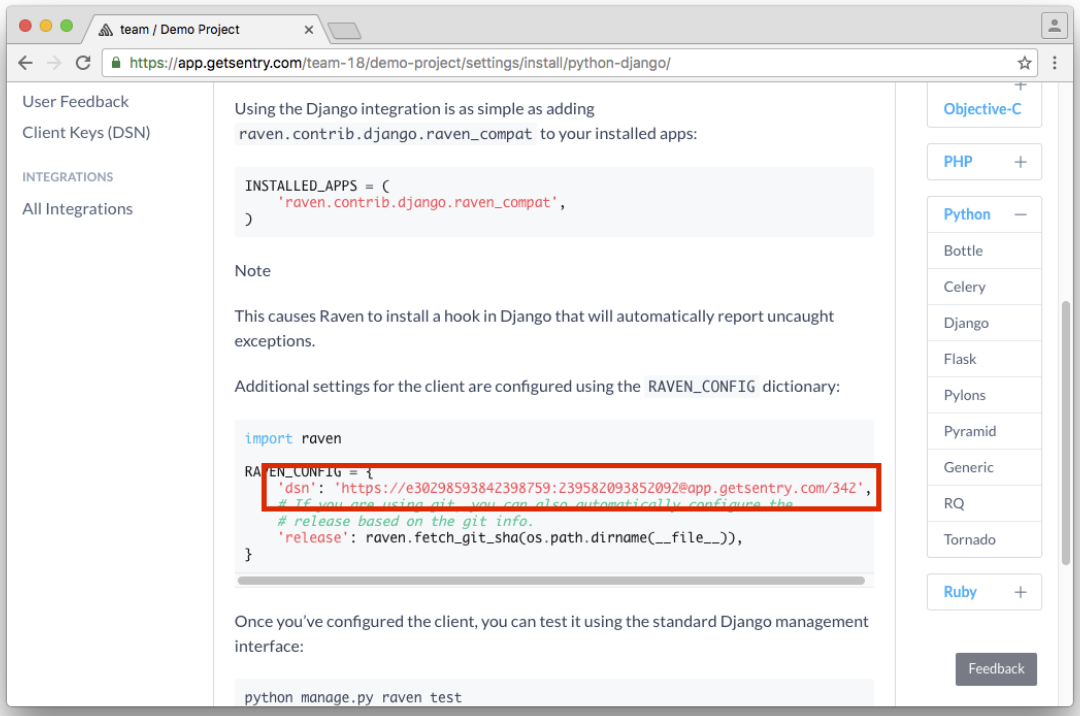
Todo, see #43

Used in: `production`

1. Go to getsentry.com and create a new account, or log in to an existing one.



2. Click on the `New Project` button and fill in a name.
3. On the "Configure your application" screen click on `Django`.
4. Scroll down until you see the declaration of the `RAVEN_CONFIG` dictionary.



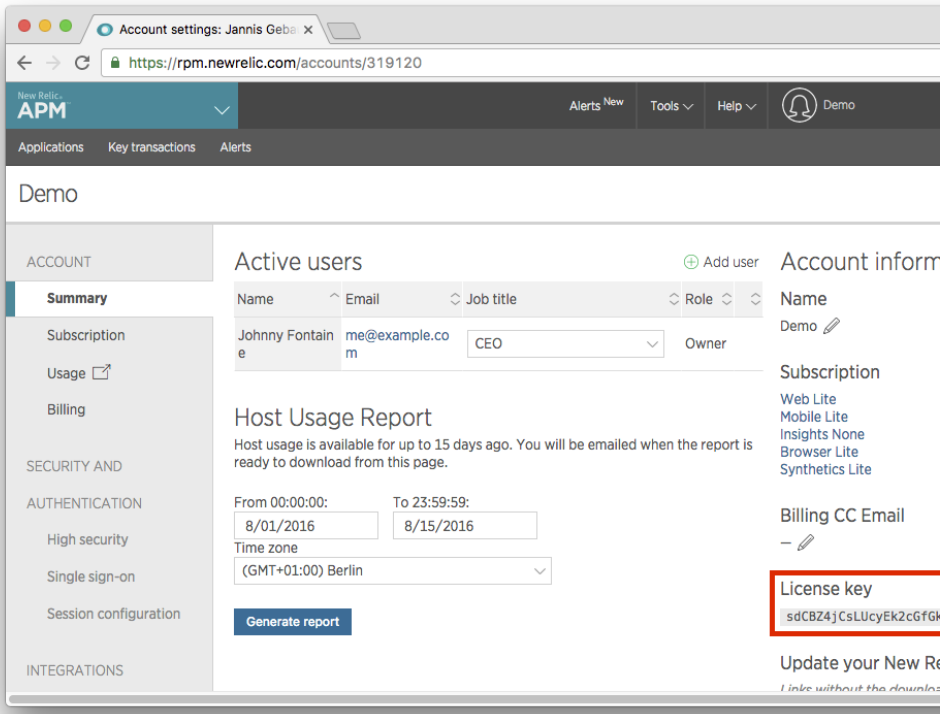
5. Copy the `dsn` into the `.env` file at `DJANGO_SENTRY_DSN`.

New Relic

Todo, see #43

Used in: `production`

1. Go to newrelic.com and create a new account, or log in to an existing one.
2. Click on your profile in the upper right corner and then on `Account Settings`.



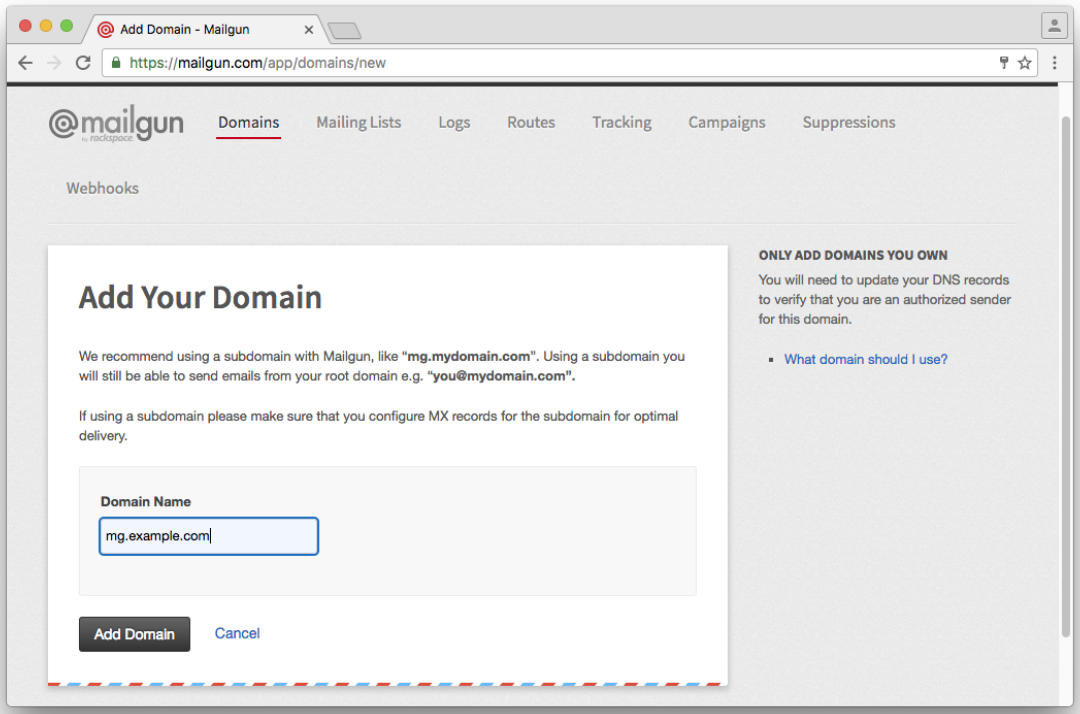
3. Your license key is on the right side.
4. Copy the license key into the `.env` file at `NEW_RELIC_LICENSE_KEY`.

Mailgun

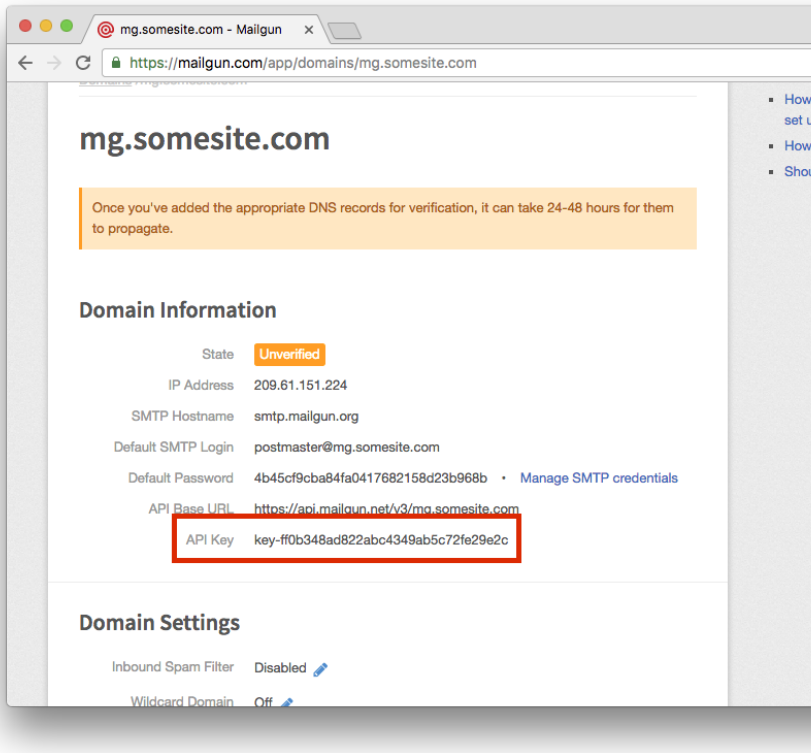
Todo, see #43

Used in: `production`

1. Go to mailgun.com and create a new account, or log in to an existing one.
2. Click on the `Add New Domain` button to create a new domain.



3. Follow the steps on the next page to set up your DNS records.
4. Scroll down and click on the `Continue To Domain Overview` button.



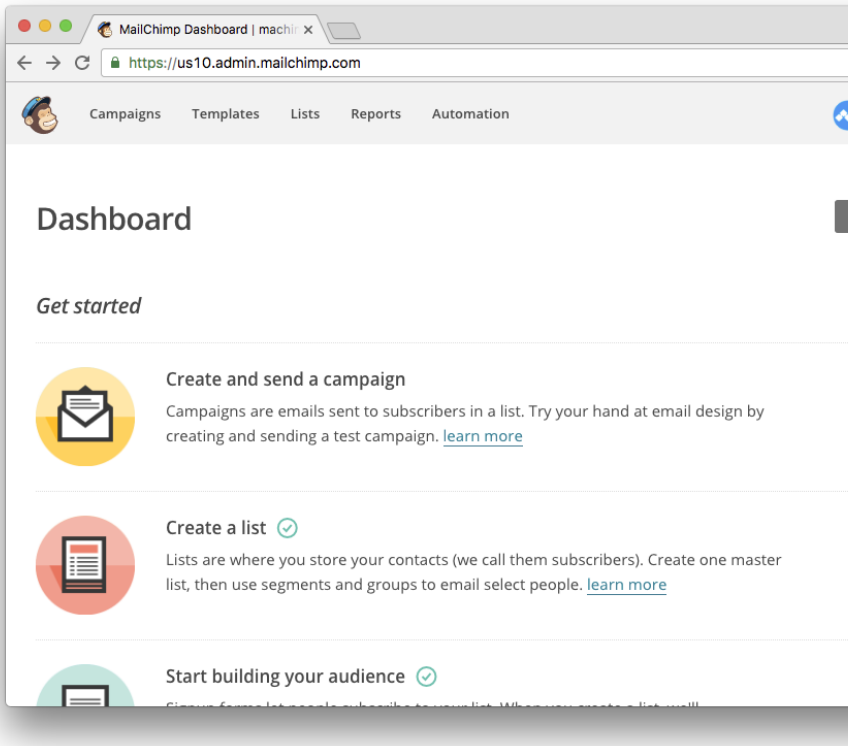
5. Locate your API key under domain information.
6. Copy the API key into the `.env` file at `DJANGO_MAILGUN_API_KEY`.

Mailchimp

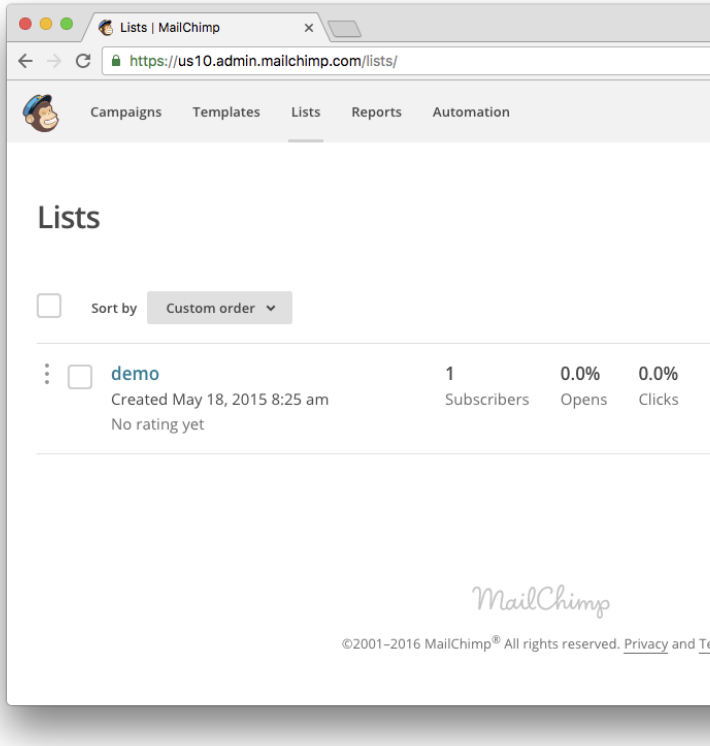
Todo, see #43

Used in: `production`

1. Go to mailchimp.com/ and create a new account, or log in to an existing one.

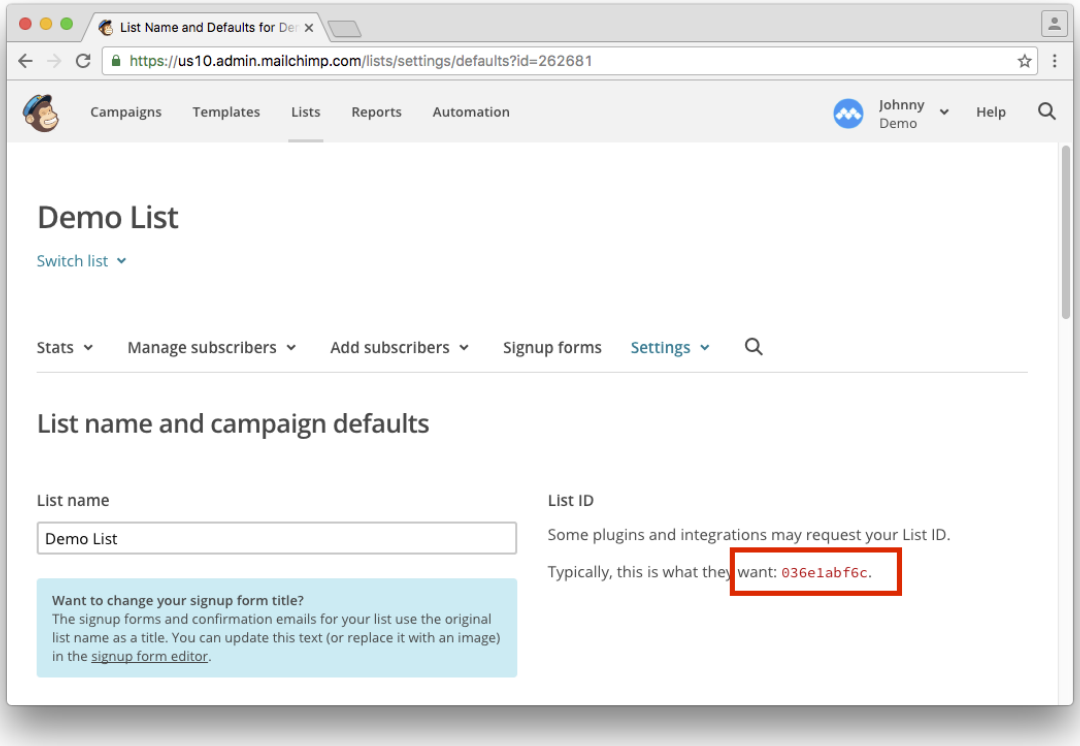


2. On your dashboard, click on `View lists`.

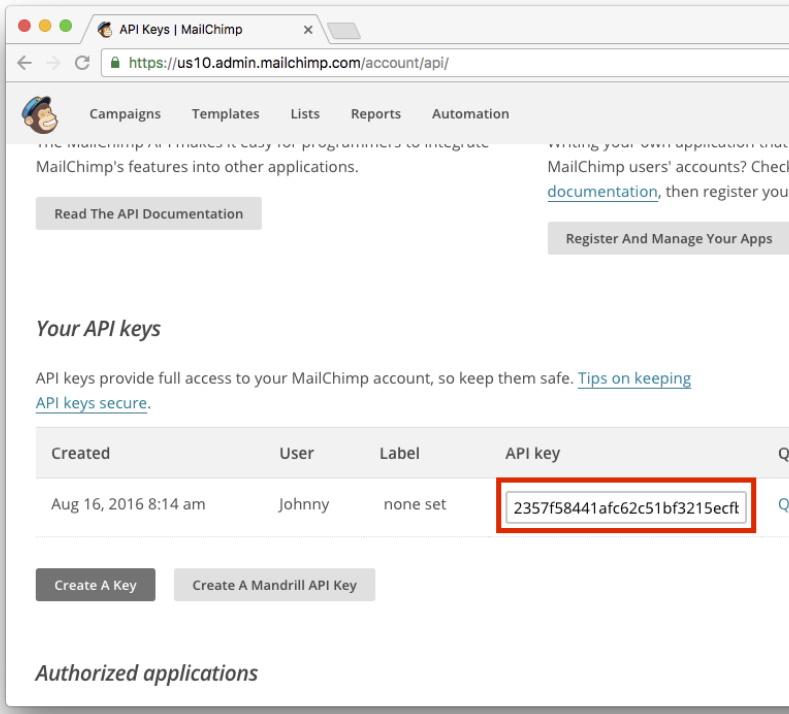


3. Click on `Create List` to create a new list for your site.
4. Fill in the details and click on `Save` at the bottom of the form.

5. Click on `Settings` > `List name and defaults`. The list ID is on the right side.



6. Copy the list ID into the `.env` file at `MAILCHIMP_LIST`.
7. Next, click on your profile menu in the upper right and then on `Profile`.
8. Click on the `Extras` tab and then on `API keys`.



9. If you haven't already, click on `Create A Key`.
10. Copy the API key into the `.env` file at `MAILCHIMP_API_KEY`.

CHAPTER 8

Dependencies

This is a full list of all dependencies used during development, testing and production.

Dependencies

Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Environment: `development`, `production`Link: djangoproject.com

psycopg2

Psycopg is the most popular PostgreSQL adapter for the Python programming language. At its core it fully implements the Python DB API 2.0 specifications. Several extensions allow access to many of the features offered by PostgreSQL.

Environment: `development`, `production`Link: initd.org/psycopg/

django-environ

Django-environ allows you to utilize 12factor inspired environment variables to configure your Django application.

Environment: `development`, `production`Link: github.com/joke2k/django-environ

django-braces

Reusable, generic mixins for Django.

Environment: `development`, `production`Link: github.com/brack3t/django-braces

django-crispy-forms

django-crispy-forms provides you with a |crispy filter and {% crispy %} tag that will let you control the rendering behavior of your Django forms in a very elegant and DRY way. Have full control without writing custom form

templates. All this without breaking the standard way of doing things in Django, so it plays nice with any other form application.

Environment: `development`, `production`Link: github.com/maraujop/django-crispy-forms

django-model-utils

Django model mixins and utilities.

Environment: `development`, `production`Link: github.com/carljm/django-model-utils

Pillow

Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors.

Environment: `development`, `production`Link: pillow.readthedocs.io

django-allauth

Integrated set of Django applications addressing authentication, registration, account management as well as 3rd party (social) account authentication.

Environment: `development`, `production`Link: github.com/pennersr/django-allauth

unicode-slugify

Unicode Slugify is a slugifier that generates unicode slugs. It was originally used in the Firefox Add-ons web site to generate slugs for add-ons and add-on collections. Many of these add-ons and collections had unicode characters and required more than simple transliteration.

Environment: `development`, `production`Link: pypi.python.org/pypi/unicode-slugify

django-autoslug

Django-autoslug is a reusable Django library that provides an improved slug field which can automatically:

- populate itself from another field,
- preserve uniqueness of the value and
- use custom slugify() functions for better i18n.

The field is highly configurable.

Environment: `development`, `production`Link: pypi.python.org/pypi/django-autoslug

pytz

pytz brings the Olson tz database into Python. This library allows accurate and cross platform timezone calculations using Python 2.4 or higher. It also solves the issue of ambiguous times at the end of daylight saving time, which you can read more about in the Python Library Reference (datetime.tzinfo).

Environment: `development`, `production`Link: pythonhosted.org/pytz/

django-redis

Full featured redis cache backend for Django.

Environment: `development`, `production`Link: github.com/niwinz/django-redis

redis

The Python interface to the Redis key-value store.

Environment: `development`, `production`Link: pypi.python.org/pypi/redis

celery

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well. The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, Eventlet, or gevent. Tasks can execute asynchronously (in the background) or synchronously (wait until ready).

Environment: `development`, `production`Link: www.celeryproject.org/

django_compressor

Django Compressor processes, combines and minifies linked and inline Javascript or CSS in a Django template into cacheable static files.

It supports compilers such as coffeescript, LESS and SASS and is extensible by custom processing steps.

Django Compressor is compatible with Django 1.8 and newer.

Environment: `development`, `production`Link: github.com/django-compressor/django-compressor

pinax-stripe

pinax-stripe is a payments Django app for Stripe.

This app allows you to process one off charges as well as signup users for recurring subscriptions managed by Stripe.

Environment: `development`, `production`Link: github.com/pinax/pinax-stripe

PyJWT

JSON Web Token implementation in Python.

Environment: `development`, `production`Link: github.com/jpadilla/pyjwt

django-celery-email

A Django email backend that uses a Celery queue for out-of-band sending of the messages.

Environment: `development`, `production`Link: github.com/pmclanahan/django-celery-email

mailchimp

A Python API client for v2 of the MailChimp API.

Environment: `development`, `production`Link: bitbucket.org/mailchimp/mailchimp-api-python/

django-loginas

"Login as user" for the Django admin.

Environment: `development`, `production`Link: github.com/stochastic-technologies/django-loginas

django-webpack-loader `react == yes`

Django webpack loader consumes the output generated by webpack-bundle-tracker and lets you use the generated bundles in django.

Environment: `development`, `production`Link: github.com/owais/django-webpack-loader

djangorestframework `react==yes`

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Some reasons you might want to use REST framework:

- The Web browsable API is a huge usability win for your developers.
- Authentication policies including packages for OAuth1a and OAuth2.
- Serialization that supports both ORM and non-ORM data sources.
- Customizable all the way down - just use regular function-based views if you don't need the more powerful features.
- Extensive documentation, and great community support.
- Used and trusted by internationally recognised companies including Mozilla, Red Hat, Heroku, and Eventbrite.

Environment: `development`, `production`Link: www.django-rest-framework.org

django-tinymce `blog==yes`

django-tinymce is a Django application that contains a widget to render a form field as a TinyMCE editor.

Environment: `development`, `production`Link: github.com/aljosa/django-tinymce

Development Dependencies

coverage

Coverage.py measures code coverage, typically during test execution. It uses the code analysis tools and tracing hooks provided in the Python standard library to determine which lines are executable, and which have been executed.

Environment: `development`Link: pypi.python.org/pypi/coverage

django-coverage-plugin

A plugin for coverage.py to measure Django template execution

Environment: `development`Link: github.com/nedbat/django_coverage_plugin

django-extensions

Django Extensions is a collection of custom extensions for the Django Framework.

Environment: `development`Link: github.com/django-extensions/django-extensions

Werkzeug

Werkzeug started as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility modules. It includes a powerful debugger, fully featured request and response objects, HTTP utilities to handle entity tags, cache control headers, HTTP dates, cookie handling, file uploads, a powerful URL routing system and a bunch of community contributed addon modules.

Environment: `development`Link: werkzeug.pocoo.org

django-test-plus

Let's face it, writing tests isn't always fun. Part of the reason for that is all of the boilerplate you end up writing. django-test-plus is an attempt to cut down on some of that when writing Django tests.

Environment: `development`Link: github.com/revsys/django-test-plus

factory-boy

factory_boy is a fixtures replacement based on thoughtbot's factory_girl.

As a fixtures replacement tool, it aims to replace static, hard to maintain fixtures with easy-to-use factories for complex object.

Environment: `development`Link: github.com/FactoryBoy/factory_boy

django-debug-toolbar

The Django Debug Toolbar is a configurable set of panels that display various debug information about the current request/response and when clicked, display more details about the panel's content.

Environment: `development`Link: github.com/jazzband/django-debug-toolbar

ipdb

ipdb exports functions to access the IPython debugger, which features tab completion, syntax highlighting, better tracebacks, better introspection with the same interface as the pdb module.

Environment: `development`Link: github.com/gotcha/ipdb

Production Dependencies

uWSGI

The uWSGI project aims at developing a full stack for building hosting services.

Application servers (for various programming languages and protocols), proxies, process managers and monitors are all implemented using a common api and a common configuration style.

Thanks to its pluggable architecture it can be extended to support more platforms and languages.

Currently, you can write plugins in C, C++ and Objective-C.

The "WSGI" part in the name is a tribute to the namesake Python standard, as it has been the first developed plugin for the project.

Versatility, performance, low-resource usage and reliability are the strengths of the project (and the only rules followed).

Environment: `production`Link: github.com/unbit/uwsgi

django-anymail

Django email backends and webhooks for Mailgun, Postmark, SendGrid, SparkPost and more

Environment: `production`Link: github.com/anymail/django-anymail

raven

Raven is a Python client for Sentry. It provides full out-of-the-box support for many of the popular frameworks, including Django, and Flask. Raven also includes drop-in support for any WSGI-compatible web application.

Environment: `production`Link: github.com/getsentry/raven-python

newrelic

New Relic's Python agent monitors your application to help you identify and solve performance issues. You can also extend the agent's performance monitoring to collect and analyze business data to help you improve the customer experience and to make data-driven business decisions. With flexible options for custom instrumentation and agent-specific APIs, New Relic's Python agent offers multiple building blocks to customize the data you need for your app.

Environment: `production`Link: New Relic Docs

CHAPTER 9

Cookiecutter Prompts

Project Name

Prompt: **project_name**

Your project name.

Project Slug

Prompt: **project_slug**

Your project name, importable by python.

Author

Prompt: **author_name**

Name of the primary author, will be added to the admins.

Email

Prompt: **email**

Email of the primary author, will be added to the admins.

Info Email

Prompt: **info_mail**

Email that receives notifications on user sign ups and requests to join the private beta.

Domain Name

Prompt: **domain_name**

Your domain name.

Timezone

Prompt: **timezone**

Sets the timezone, see list of timezones for more.

Django Long Term Support

Prompt: **django_long_term_support**

Installs the LTS release of Django, currently 1.8.

React & Redux Integration

Prompt: **react**

Installs Webpack, React, Redux, Django Rest Framework with session authentication and a react starter template. See react & redux for more.

Basic Blog

Prompt: **blog**

Adds a very simple blog app to the project. See blog for more.

Private Beta Mode

Prompt: **private_beta**

When enabled, user sign up is replaced by an invite system. See private beta mode for more.

Default Subscription Type

Prompt: **free_subscription_type**

Sets the default plan a new user is subscribed to. Select `freemium` to subscribe new users to the **Free** plan, `trial` for the **Trial** plan, and `None` for no subscription at all. See *free subscription type* for more.

CHAPTER 10

Things to Change

This document is still a work in progress, see #45.

Logo & Favicon

The logo is at `static/images/logo.png` and the favicon at `static/images/favicon.png`. Replace them with your own.

Privacy Policy

Add your privacy policy in `templates/pages/privacy_policy.html`.

Terms and Conditions

Add your own terms and conditions in `templates/pages/terms_and_conditions.html`.

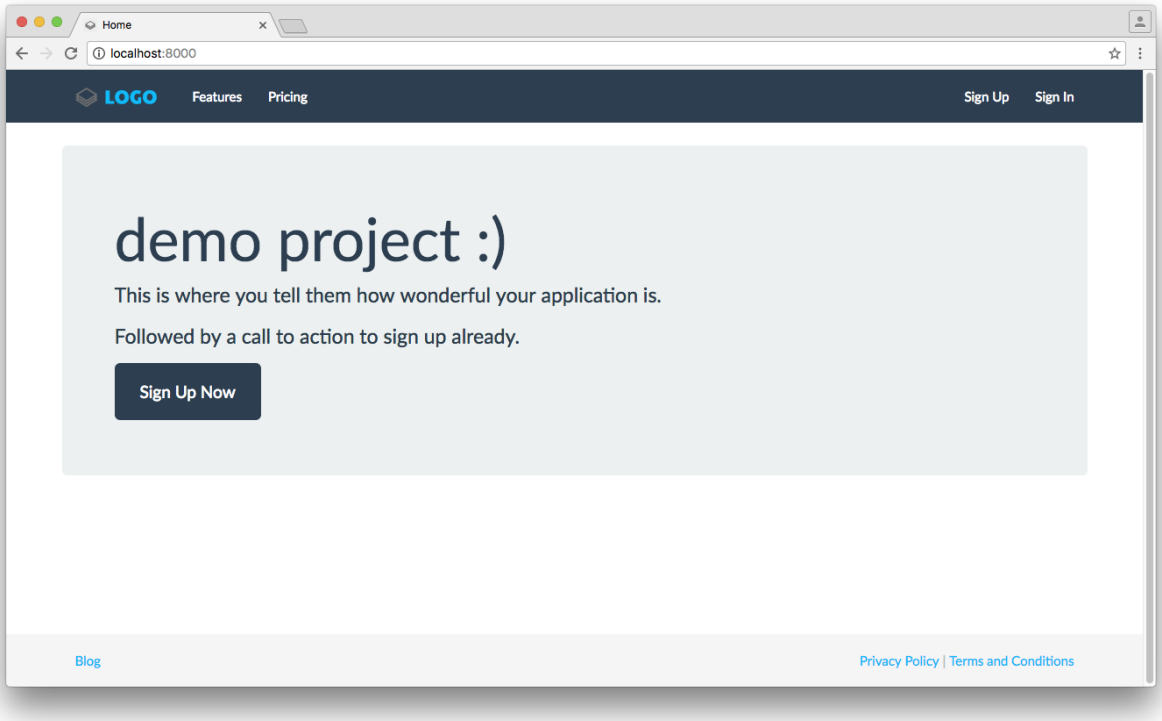
Templates

Cookiecutter SaaS uses a total of three base templates. They are based on bootstrap with basic styles from bootswatch.com's flatly theme. You typically want to replace them all with your own base templates and styling.

Base Template

Located at: `templates/base.html`

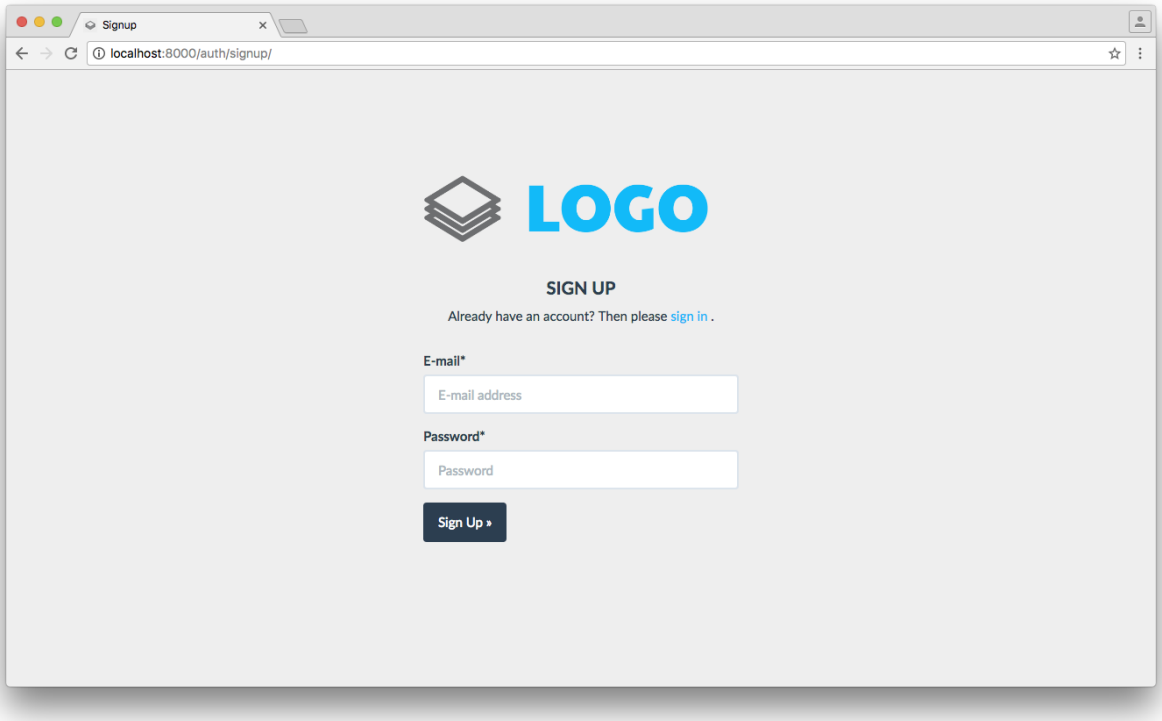
The base template for every page outside of the application. For example the startpage, the features page, the pricing page etc.



Blank Base Template

Located at: `templates/base_blank.html`

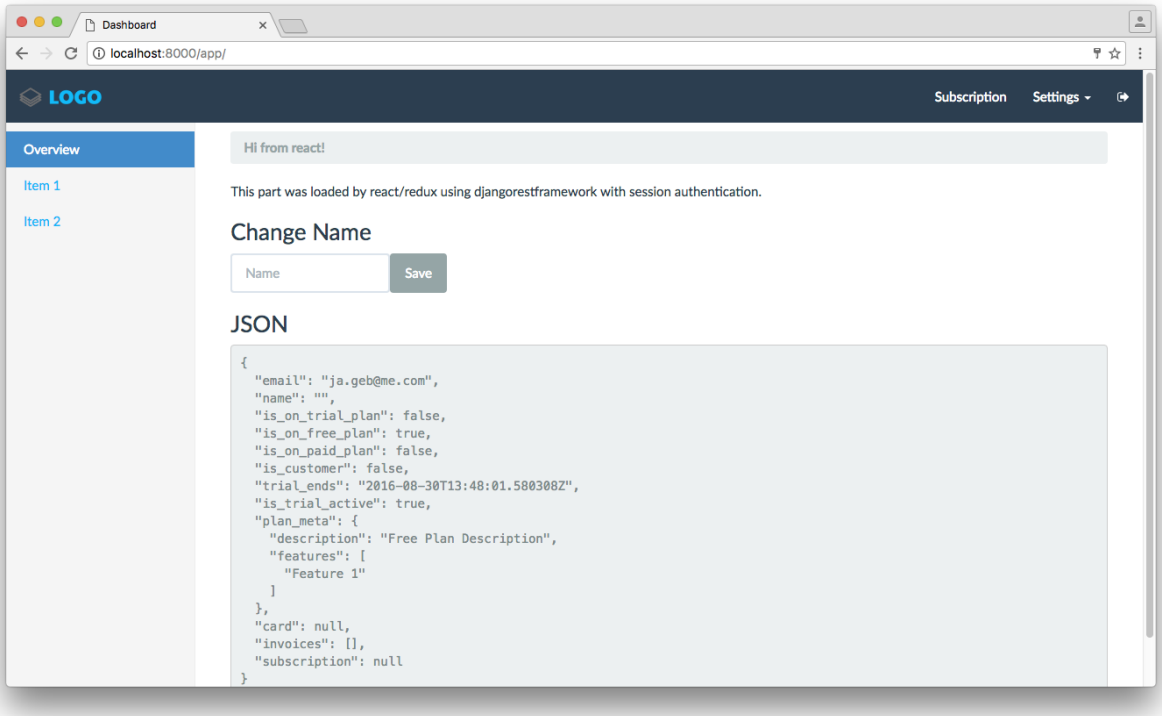
Base template for all pages with very little content and a call to action. For example, user sign ups, email verification, error pages.



App Base Template

Located at: `templates/app/base.html`

This is the base template for your app.


Dashboard
localhost:8000/app/
LOGO
Subscription
Settings
Overview
Item 1
Item 2
Hi from react!
This part was loaded by react/redux using djangorestframework with session authentication.
Change Name
Name
Save
JSON
{
"email": "ja.geb@me.com",
"name": "",
"is_on_trial_plan": false,
"is_on_free_plan": true,
"is_on_paid_plan": false,
"is_customer": false,
"trial_ends": "2016-08-30T13:48:01.580308Z",
"is_trial_active": true,
"plan_meta": {
"description": "Free Plan Description",
"features": [
"Feature 1"
]
},
"card": null,
"invoices": [],
"subscription": null
}

CHAPTER 11

Indices and tables

- genindex
- modindex
- search