Django-danceschool Documentation

Release 0.1.0

Lee Tucker

Apr 23, 2019

Contents:

CHAPTER 1

Production Deployment

If you are looking to immediately using the Django Dance School project as quickly as possible, then you've come to the right place! Using the Django Dance School project's production template repository, you can have a complete installation of the project and all its dependencies running within minutes. The production template also automatically incorporates industry-standard open source tools, so you can be confident that your site's deployment is stable and secure, and you can get back to running your school.

Note: If you are setting up a Docker or Heroku project for the first time, it is recommended that you find someone with a background in the use of these systems to ensure that your deployment goes as smoothly as possible. If you encounter errors or missing functionality in setting up this template, please submit an issue to the issue tracker, or email django.danceschool@gmail.com.

1.1 Docker

Docker is a popular open-source platform for deployment that is built on the concept of containerization. Each piece of your stack (of which the project's Django server is only one) is deployed inside as separate container, and they interact with one another over an overlay network that is isolated from regular network traffic. This makes Docker stacks easy to configure and reproduce.

To use Docker, you will need access to a server that has both Docker and Docker Compose installed. Luckily, it is easy to deploy such a preconfigured server using popular cloud services such as DigitalOcean droplets or Amazon EC2. The project will run successfully (and is suitable for most individual schools) on servers with as little as 1 GB of RAM available, which corresponds to DigitalOcean's smallest $5/month droplet size. And, the template can automatically use LetsEncrypt, so that your site traffic is always automatically SSL encrypted for free.

Docker can also be deployed on your local machine, allowing you to develop with a full server stack that is identical to the one you use in production, even on Windows. However, for most users, because of the additional layer of abstraction associated with containerization, use of the development template (see **:ref:'installation_development'_**) is still recommended. See **:ref:'docker_development'_** for more details below.

1.1.1 Docker installation

You Will Need: - Docker >= 17.12.0 - Docker Compose >= 1.14.0 - Environment that can run Bash scripts (Linux, MacOS, or

Windows 10 WSL)

Note These steps assume that you are using the included LetsEncrypt capabilities for SSL. If you are planning to provide your own SSL certificate, or you need to use OpenSSL because you are testing on a server that is not associated with any domain name, you will be prompted for that when you run the Bash script.

1. On your production server, clone the production template repository:

```
git clone https://github.com/django-danceschool/production-template.git
```

2. Edit the file `env.web` to insert value for the following environment variables: - `ALLOWED_HOST`: Your site's domain name (e.g. mydanceschool.com) - `VIRTUAL_HOST`: Your site's domain name - `LETSENCRYPT_HOST`: Your site's domain name - `LETSENCRYPT_EMAIL`: Your email address at which you want to receive error

   notices related to LetsEncrypt.

3. Run the included Docker `setup-stack.sh` Bash script. You will be prompted to provide a range of pieces of information that are needed for setup. And, as part of the script, you will also be prompted to take the usual steps needed for a Django deployment of the project (running initial migrations, collecting static files, creating a superuser, and running the `setupschool` command to initialize the database).

```
cd production-template
docker/setup_stack.sh
```

4. Use the `docker` command to deploy your stack!

```
docker stack deploy -c docker-compose.yml school
```

1.1.2 Additional Steps/Commands

- **To see that your stack is running, run `docker ps` and see that there should** be 6 different running containers (Gunicorn, Huey, Redis, PostgreSQL, Nginx proxy, and LetsEncrypt).
- To remove your stack (stop the server), run `docker stack rm school`.
- If you modify the `docker-compose.yml` file, you will need to re-deploy your stack.
- If you modify `settings.py` or other Python files (other than custom apps), you will need to rebuild the Django container. The easiest way to do this is to run the `setup_stack.sh` script again, choosing options to avoid overwriting pre-existing credentials, and skipping the setupschool script.

For more information about Docker and how to interact with your Docker stack, see the Docker documentation.

1.1.3 Developing with Docker

Docker is cross-platform and available for desktops, which means that you can easily get the production template running on your own machine. This can be a great way to test out custom server configurations and/or custom apps. For Linux and MacOS, just install Docker and follow the steps listed above from the command line.

If you are on Windows, then in order to run the included Bash script, you will need to install Windows Subsystem for Linux (WSL).

- Follow the instructions from Microsoft: WSL Installation Guide.
- Once you are on a WSL command line, use the following guide to allow the Linux Docker client to connect to your Window's machine's Docker server: Setting Up Docker for Windows and WSL To Work Flawlessly.

1.2 Heroku

Heroku is a popular platform-as-a-service (PaaS) in which each process in your school's project runs in a separate container known as a dyno. Dynos can be created or destroyed at whim, allowing you to easily scale your project up or down as needed. However, for most standard dance schools, you will need only one dyno for each of the following processes:

- Web: The Django instance that serves all web content to users. By default, this dyno serves Django using the popular Gunicorn.
- Worker: The Dance School project uses the Huey system to handle asynchronous tasks such as sending emails, creating automatically-generated expenses, and closing classes for registration depending on elapsed time.
- Redis: Huey's tasks are queued in the Redis data store, which is automatically configured by this template.
- Database: This template is set up for Heroku to store all of your data in a standard PostgreSQL database which is automatically configured by this template.

Heroku's pricing is based on hours of use for these dynos, as well as their size. Although Heroku does provide a free tier, with hours of usage limitations, this tier presents issues for the project because there are numerous tasks that are required to run at regular intervals (such as creating expense items and closing classes for registration). Therefore, we suggest that you use "hobby" dynos. As of October 2017, hobby web and worker dynos cost $7/month each, a "Hobby Basic" database dyno costs $9/month, and a hobby Redis dyno is free, which means that a standard setup that is suitable for most schools will cost $23/month to host on Heroku.

1.2.1 Initial Heroku Setup

Button-Based Deployment

1. Click the button above and follow the instructions to deploy your app. This will take several minutes.
2. When the initial deployment has finished, click on "Manage App" to open your app in the Heroku dashboard. From there, click the button at the top right labeled "More" and select "Run console." In the field that pops up, type in "bash" to access a command line console for your app.
3. At the command-line console, run the following, and follow the prompts at the command line to create a superuser and perform your school's initial setup: - Create a superuser: `python3 manage.py createsuperuser` - Setup the school with initial pages and sensible defaults: `python3 manage.py setupschool`
4. Type `exit` to close the command line process, close out of the console, navigate to https://<your-app>.herokuapp.com/ and enjoy!

Manual Method (Recommended for Customization)

You will need: - A command line Git client installed (you can install one from here). - The Heroku command line client (get it here). - A Heroku account - An Amazon AWS account (Heroku dynos cannot store user uploaded files such as instructor photos locally) - An email address or other method for your site to send emails - An account with one or more payment processors that you wish to use (Paypal, Square, and Stripe are all supported by default)

1. Open a command line in the location where you would like the local copy of your installation to live. Clone this repository to your local folder:

   ```
   git clone https://github.com/django-danceschool/production-template.git
   ```

2. Login to Heroku:

   ```
   heroku login
   ```

3. Create a new Heroku app:

   ```
   heroku create <your-app-name>
   ```

4. Push your project to Heroku, where it will now be deployed (this will take a few minutes the first time that you do it):

   ```
   git push heroku master
   ```

5. Use one-off dynos to run the initial database migrations that your project needs and to create a superuser (you will be prompted for a username and password):

   ```
   heroku run python3 manage.py migrate
   heroku run python3 manage.py createsuperuser
   ```

6. **Optional, but strongly recommended:** Run the easy-installer setup script, and follow all prompts. This script will guide you through the process of setting initial values for many things, creating a few initial pages that many school use, and setting up user groups and permissions that will make it easier for you to get started running your dance school right away.

   ```
   heroku run python3 manage.py setupschool
   ```

7. Go to your site and log in!

1.3 Additional Setup Needed

1.3.1 Amazon S3 Setup (Heroku)

Heroku's dynos are not set up to store your user uploaded files permamently. Therefore, you must set up a third-party storage solution or else your user uploaded files (instructor photos, receipt attachments for expenses, etc.) will be regularly deleted.

To enable file upload to Amazon S3, you will first need to create an S3 bucket, with access permissions set so that uploaded files can be publicly read. Then, in order for Heroku to access S3, you must set all of the following environment variables: - `AWS_ACCESS_KEY_ID` - `AWS_SECRET_ACCESS_KEY` - `AWS_STORAGE_BUCKET_NAME`

Once these settings have been set, Amazon S3 upload of your files should be automatically enabled!

1.3.2 Payment Processor Setup

This installation is configured to read your payment processor details from environment variables. If you have added the appropriate payment processor details needed for the three standard payment processors, then the appropriate payment processor app will automatically be added to `INSTALLED_APPS`, so that you do not need to edit the settings file at all in order to begin accepting payments.

For details on how to get the credentials that you will need for each payment processor, see: **:ref:'setup_payments'_**.

1.3.3 Email Setup

Your project needs a way to send emails, so that new registrants will be notified when they register, so that you can email your students, so that private event reminder emails can be sent, etc.

By default, this installation uses the `dj-email-url` app for simplified email configuration. You can specify a simple email URL that will permit you to use standard services such as Gmail. This installation template also has built-in functionality for the popular Sendgrid email system. For most small dance schools, the Sendgrid free tier is adequate to send all school-related emails, but Sendgrid allows larger volume emailing as well.

Examples

- **Sendgrid:** set `$SENDGRID_API_KEY` to your SendGrid API key, set `$SENDGRID_USERNAME` to your SendGrid username and set `$SENDGRID_PASSWORD` to your SendGrid password. SendGrid will then be enabled as your email service automatically.
- **Gmail:** set `$EMAIL_URL` to 'smtps://user@domain.com:pass@smtp.gmail.com:587'. Note that Gmail allows only approximately 100-150 emails per day to be sent from a remote email client such as your project installation.

CHAPTER 2

Local Installation for Development

This page provides instuctions for a local development installation, using the provided development template so that you can quickly begin to explore the project's features locally, and so that you can develop custom apps and functionality quickly in a local environment.

If you are seeking to deploy a production installation quickly, this is probably not the right guide for you. It is strongly recommended that you check out the production installation guide, which provides installation instructions for rapid production deployment using standard tools such as Docker and Heroku.

2.1 What you need:

- Python 3.4, 3.5, or 3.6
- The ability to create a virtual environment (on Linux, install the `python-virtualenv` package)
- pip3 - the Python package manager
- A suitable database. For development and testing, SQLite is used by default, so you do not need to do anything to get started. For production use, it is **strongly Recommended** to use PostgreSQL server 9.4+
- External library dependencies for Pillow, used for basic image processing (see the Pillow Documentation).
- **Recommended for production use:** *Redis server <https://redis.io/>* for asynchronous handling of emails and other tasks
- **For Paypal integration only:** SSL and FFI libraries needed to use the Paypal REST SDK (see *the Github repo <https://github.com/paypal/PayPal-Python-SDK>* for details)

Linux

- If you are using a package manager (such as apt), you can usually directly install the needed dependencies for Pillow. For example, on Ubuntu:

```
sudo apt-get install libjpeg zlib
sudo apt-get install redis-server
sudo apt-get install libssl-dev libffi-dev
```

Mac

- You'll have to use homebrew to `brew install` dependencies above. Beware you may run into the zlib issue which can be answered here.

2.2 Basic Installation Process

1. Create a subfolder for your new Django project, and enter it:

```
mkdir django
cd django
```

2. Create a new virtual environment and enter it:

```
python3 -m virtualenv .
source bin/activate
```

   - *Note:* Depending on your system, you may need to follow slightly modified instructions in order to create a virtual environment. No matter which method you use, be sure that your environment is set to use Python 3 by default.

3. Install the django-danceschool from PyPi. This will also install all of the necessary dependencies (which may take awhile)

   ```
   pip3 install django-danceschool
   ```

   Note: Additionally, depending on your operating system, you may need to install certain program dependencies in order to install the Pillow package and the psycopg2 package (as listed in requirements.txt). If you run into issues at this step of the installation, look for these issues first.

4. Start your Django project, using the `django-admin` command. To avoid having to set a large number of settings manually, we strongly recommend that you use the preexisting installation template as follows. Make sure that you are in the folder where you would like your project to be located when you do this.

```
django-admin startproject --template https://github.com/django-danceschool/
↪development-template/archive/master.zip <your_project_name>
```

   Mac/OS X Users: If you having SSL issues trying to run this command on the Mac check out this Stack Overflow response or download the file manually, move it to the local install folder, and run this modified version of the `django-admin` command.

```
django-admin startproject --template development-template-master.zip <your_
↪project_name>
```

5. Perform initial database migrations

```
cd <your_project_name>
python3 manage.py migrate
```

6. Create a superuser so that you can log into the admin interface (you will be prompted for username and password)

```
python3 manage.py createsuperuser
```

7. **Optional, but strongly recommended:** Run the easy-installer setup script, and follow all prompts. This script will guide you through the process of setting initial values for many things, creating a few initial pages that many

school use, and setting up user groups and permissions that will make it easier for you to get started running your dance school right away.

```
python3 manage.py setupschool
```

8. Run the server and try to log in!

```
python3 manage.py runserver
```

Following steps 1-8 above will give you a working installation for testing purposes. However, additional steps are needed to setup emails, payment processor integration, and other automated processes. These are described below.

2.2.1 Settings Customization and Production Deployment

After performing steps 1-8 above, you should have a working instance of the danceschool project. However, in order to make the site usable for your purposes, you will, at a minimum, need to do some basic setting of settings and preferences

There are two types of settings in this project:

1. Hard-coded settings needed to run the project at all (located in settings.py)
2. Runtime settings that customize the site's functionality (stored in the database using the django-dynamic-preferences app, and then cached)

If you have used the default setup provided in step 4 above, then you have already been provided with appropriate default settings for a development instance of the project. For example, Django's debug mode is on, and the project uses a SQLite backend to store data instead of a production database such as PostgreSQL. Before thoroughly testing out the project, the only settings that you are required so set in `settings.py` are for features such as email and the Paypal/Stripe integration, because these features cannot be enabled by default until you have entered credentials for those services. However, before you deploy this project for production purposes, you will need, *at a minimum*, to customize settings for a payment processor (e.g. Paypal/Stripe/Square), email, and a production-appropriate database. Also, often time, if your workflow involves both a development installation and a production installation, there will be different settings required for each installation.

For more details on the types of settings that you may need to modify, see manual_settings_list. Be sure to also check out the guide to setting up email, setup_email, the guide to setting up payment processors, setup_payments, and the guide to the Huey asynchronous task queue, *Huey (and Redis) setup for production*.

Customizing runtime settings is even easier. Simply log in as the superuser account that you previously created, and go to http://yoursite/settings/global/. There, you will see organized pages in which you can change runtime settings associated with various functions of the site. If you have run the `setupschool` command as instructed in step 7 above, you will find that sensible defaults for all of the most important runtime settings have already been put into place for you.

CHAPTER 3

Manual Project Setup Guide

3.1 Introduction

In setting up your project, it is strongly recommended that you deploy your new project by running the following:

```
django-admin startproject --template https://github.com/django-danceschool/
↪development-template/archive/master.zip <your_project_name>
```

However, it is also possible to deploy a new project by manually editing `settings.py` to enter the needed values.

This section describes how to do such an installation. Keep in mind that manual installation is only recommended for advanced users who wish to integrate the danceschool app into a pre-existing Django installation. If you are just looking to customize the project, then you likely want to follow the Development Installation guide.

3.2 Importing Third-Party Settings

Setting up the Django-danceschool project requires setting a large number of configuration options for third-party apps. However, these options can be imported automatically so that you do not need to enter them yourself. Near the top of the `settings.py` file, add the following:

```
from danceschool.default_settings import *
```

Note also that any of the options specified in `danceschool.default_settings` can readily be overridden in `settings.py`. Just be sure to set your chosen setting values *below* the import command above.

3.3 Installed Apps

In addition to the various apps that are components of the danceschool project, there are several other apps that need to be added to your project's `INSTALLED_APPS`. It is important to note that the order in which apps are added often

matters. In particular, because Django's template loading and URL pattern matching functions use the first matching template/pattern, some apps need to be loaded before others in order for them to function correctly.

First, list the Django CMS app in `INSTALLED_APPS`, followed by the Django dynamic preferences app. These apps go first so that they can find and register CMS plugins and dynamic preferences from other apps:

:: 'cms', 'dynamic_preferences',

Next, list the core danceschool app, preceded by the themes app. The core app provides all of the necessary functionality of the project, and is required. The themes app is optional, but it is highly recommended because it provides the functionality necessary to enabled the project's built-in themes.

If you have setup any custom app which overrides the templates used by the danceschool project, then this should also be listed here:

:: # '<your_custom_app>', 'danceschool.themes', 'danceschool.core',

The `danceschool.core` app contains all of the necessary basic functionality of the project. However, depending on your needs, you may want to install some of all of the following apps by adding them to `INSTALLED_APPS`:

```
'danceschool.financial',        # Financial reporting and expense/revenue␣
↪tracking
'danceschool.private_events',   # Non-public events and calendar with␣
↪reminders and feeds
'danceschool.discounts',        # Configurable registration discounts
'danceschool.vouchers',         # Vouchers, gift certificates, and the␣
↪referral program
'danceschool.prerequisites',    # Configurable prerequisites for specific␣
↪classes
'danceschool.stats',            # School performance statistics
'danceschool.news',             # A simple news feed
'danceschool.faq',              # A simple FAQ system
'danceschool.payments.paypal',  # Paypal Express Checkout payment processor
'danceschool.payments.stripe',  # Stripe Checkout payment processor
```

Then, before including the Django contrib apps, add the following apps (the order of these does not matter, but some apps *must* be listed before `django.contrib.admin`):

```
'adminsortable2',
'allauth',
'allauth.account',
'allauth.socialaccount',
'ckeditor_filebrowser_filer',
'crispy_forms',
'dal',
'dal_select2',
'daterange_filter',
'djangocms_admin_style',
'djangocms_forms',
'djangocms_text_ckeditor',
'easy_pdf',
'easy_thumbnails',
'filer',
'huey.contrib.djhuey',
'menus',
'polymorphic',
'sekizai',
'treebeard',
```

If you have enabled the `danceschool.themes` app, then you will also need:

:: 'djangocms_icon', 'djangocms_link', 'djangocms_picture', 'djangocms_bootstrap4', 'djangocms_bootstrap4.contrib.bootstrap4_alerts', 'djangocms_bootstrap4.contrib.bootstrap4_badge', 'djangocms_bootstrap4.contrib.bootstrap4_card', 'djangocms_bootstrap4.contrib.bootstrap4_carousel', 'djangocms_bootstrap4.contrib.bootstrap4_collapse', 'djangocms_bootstrap4.contrib.bootstrap4_content', 'djangocms_bootstrap4.contrib.bootstrap4_grid', 'djangocms_bootstrap4.contrib.bootstrap4_jumbotron', 'djangocms_bootstrap4.contrib.bootstrap4_link', 'djangocms_bootstrap4.contrib.bootstrap4_listgroup', 'djangocms_bootstrap4.contrib.bootstrap4_media', 'djangocms_bootstrap4.contrib.bootstrap4_picture', 'djangocms_bootstrap4.contrib.bootstrap4_tabs', 'djangocms_bootstrap4.contrib.bootstrap4_utilities',

Finally, be sure that the following django contrib apps are all listed in `INSTALLED_APPS` at the bottom:

```
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'django.contrib.sites',
'django.contrib.sitemaps',
'django.contrib.admin',
```

3.4 Template settings

Django CMS requires some specialized context processors to be enabled. So, add the following to `TEMPLATES['OPTIONS']['context_processors']`:

```
'cms.context_processors.cms_settings',
'sekizai.context_processors.sekizai',
'danceschool.core.context_processors.site',
```

3.5 Middleware

Django CMS requires the following to be added to `MIDDLEWARE_CLASSES`:

At the top:

```
'cms.middleware.utils.ApphookReloadMiddleware',
```

Anywhere in MIDDLEWARE_CLASSES:

```
'django.middleware.locale.LocaleMiddleware',
'cms.middleware.user.CurrentUserMiddleware',
'cms.middleware.page.CurrentPageMiddleware',
'cms.middleware.toolbar.ToolbarMiddleware',
'cms.middleware.language.LanguageCookieMiddleware',
```

3.6 Site ID and Language Code

Because Django CMS makes use of `django.contrib.sites`, in order for a default URL to be available for pages, the CMS needs to know the database identifier ofyour default site. For most installations, this means adding:

```
SITE_ID = 1
```

Django CMS also uses slightly different language designations than Django as a whole. By default, Django's `settings.py` ships with `LANGUAGE_CODE = 'en-us'`. Assuming that your site will be running in English, you should change this to `LANGUAGE_CODE = 'en'`.

3.7 URL Handling

The Danceschool project has a single `urls.py` file which handles all of the URLs for the project and its core dependencies. Similarly, Django CMS requires a catch-all URL pattern that tries to match any unmatched URLs to CMS pages. So, be sure to add the following code to the bottom of your `urls.py`.

```
from django.conf.urls import include, url

...


# Add this at the bottom of urls.py
urlpatterns += [
    # Include your own app's URLs first to override default app URLs
    # url(r'^', include('<yourapp>.urls')),
    # Now, include default app URLs and CMS URLs
    url(r'^', include('danceschool.urls')),
    url(r'^', include('cms.urls')),
]
```

Note: If for any reason you wish to modify any of the default URL paths provided by the project, you can do so by adding your own URLs prior to the inclusion of `danceschool.urls`.

3.8 Other Settings You May Wish to Modify

As with all Django projects, you are generally free to modify other settings as you see fit. However, there are certain other settings that are commonly modified for each installation, and that you will likely wish to modify.

For more information on these settings, see the Django documentation.

Static file storage/upload settings:

- `STATIC_URL` (set to "/static/" by default)
- `STATIC_ROOT`
- `MEDIA_ROOT`
- `MEDIA_URL`
- `CKEDITOR_UPLOAD_PATH`

Django email settings (needed for confirmation emails, etc.)

For more details on email setup, see the dedicated email setup documentation: email_setup.

- host: `EMAIL_HOST`
- port: `EMAIL_PORT`
- username: `EMAIL_HOST_USER`

- password: `EMAIL_HOST_PASSWORD`
- use_tls: `EMAIL_USE_TLS`
- use_ssl: `EMAIL_USE_SSL`

Django database settings (recommended to change from default SQLite for production applications):

- `DATABASES['default']['ENGINE']`
- `DATABASES['default']['NAME']`
- `DATABASES['default']['USER']`
- `DATABASES['default']['PASSWORD']`
- `DATABASES['default']['HOST']`
- `DATABASES['default']['PORT']`

Django-filer settings

See the Django-filer documentation for more details:

- `FILER_STORAGES`
- `DEFAULT_FILER_SERVERS`

** Payment processors **

These are just the settings listed above in *Paypal*, *Stripe*, and *Square*.

For Paypal:

- `PAYPAL_MODE` (either "sandbox" or "live")
- `PAYPAL_CLIENT_ID`
- `PAYPAL_CLIENT_SECRET`

For Stripe: - `STRIPE_PUBLIC_KEY` - `STRIPE_PRIVATE_KEY`

For Square: - `SQUARE_APPLICATION_ID` - `SQUARE_ACCESS_TOKEN` - `SQUARE_LOCATION_ID`

3.9 Huey (and Redis) setup for production

Certain website tasks are best run asynchronously. For example, when a student successfully registers for a class, the website does not need to wait for the confirmation email to be sent in order for the process to proceed. Similarly, other tasks such as closing of class registration are run at regular intervals and do not depend on user interaction. For these reasons, this project uses the Huey task queue. Huey is run as a separate process from your webserver, and when tasks are submitted to Huey via functions in each app's `tasks.py`, they are handled by this separate process.

If you followed the quick start instructions, then Huey is already installed and a default setup is enabled that will enable you to get going quickly. On a separate command line from your test server, simply type in `python3 manage.py run_huey` to run a Huey instance that will handle sending emails, etc., automatically. Your site will continue to these features as well as recurring tasks for as long as this process continues to run.

The default settings for Huey involve storing the task queue data in SQLite-based file storage. Upon running Huey, you will see a newly created SQLite file in the same directory as your project's manage.py file, which stores the task queue data. Although this approach allows for convenient setup for testing purposes using the project's default settings, it is not recommended to use Huey's SQLite storage backend for production purposes. Instead, it is strongly recommended that you set up the popular Redis cache server, and modify your `settings.py` file to use Huey's Redis integration.

Note that These instructions are designed for Linux, and they assume that you will be running Redis locally using default settings. Getting Redis running on Windows may require a slightly different process, and configuring Huey to use a remote Redis installation will also involve modifying site settings.

1. Install the Redis client for Python: `pip3 install redis`
2. Start the Redis server: `sudo service redis-server start`
3. Add the following to `settings.py` (this basic setup can be customized, see the Huey documentation).

```
from huey import RedisHuey
from redis import ConnectionPool
pool = ConnectionPool(host='localhost', port=6379, max_connections=20)
HUEY = RedisHuey('danceschool',connection_pool=pool)
```

4. As before, run Huey in its own command shell: *python3 manage.py run_huey*

With these two steps, your installation should now be able to send emails programmatically, and your site should also run recurring tasks as long as both Redis and Huey continue to run.

CHAPTER 4

Setup: Payment Processors

Before you can begin to accept payment for online registrations, you must set up one or more payment processors. Luckily, the project already provides integration with three of the most common payment processors: Paypal, Stripe, and Square. This page explains how to enable these processors for your site's use.

4.1 Which Processor Should I Use?

Choosing a payment processor is a matter of personal preference, of course. Your choice may depend on such factors as which processor you are already using, whose user interface you like most,

- All three established processors have similar fee structures for transactions within the United States. If you are outside the U.S., check the processor's website for information about availability and applicable fees.
- As of 2018, only Square provides a solution for point-of-sale integration that works smoothly with mobile web applications such as this one. So, if you wish to take electronic payments for full registrations at the door, Square may be the best solution for you.
- There is nothing in this project that prevents you from using multiple payment processors simultaneously (though of course seeing multiple forms might confuse your customers!)

4.2 Paypal

In order to accept and process Paypal payments, you will need to set up the credentials for your Paypal account. The Django Dance School project uses the Paypal REST SDK. Older versions of this repository used the Paypal IPN system, but this software is no longer maintained, and it is highly recommended that you upgrade to using the REST API.

4.2.1 REST API Setup

1. Go to the Paypal developer website and log in using the Paypal account at which you wish to accept payments.

2. On the dashboard, under "My Apps & Credentials", find the heading for "REST API apps" and click "Create App." Follow the instructions to create an app with a set of API credentials
3. Once you have created an app, you will see credentials listed. At the top of the page, you will see a toggle between "Sandbox" and "Live." If you are setting up this installation for testing only, then choose "sandbox" credentials so that you can test transactions without using actual money. For your public installation, use "live" credentials.
4. Next steps depend on whether you have performed a production installation or a development/manual installation of the project:

- **Development/manual installation: Edit `settings.py` to add:**
  - `INSTALLED_APPS`: Uncomment/enter `danceschool.payments.paypal`
  - `PAYPAL_MODE`: Either "sandbox" or "live"
  - `PAYPAL_CLIENT_ID`: The value of "Client ID"
  - `PAYPAL_CLIENT_SECRET`: The value of "Secret". **Do not share this value with anyone, or store it anywhere that could be publicly accessed**
- **Production template installation**: Instead of modifying `settings.py`, you can add the values of `PAYPAL_MODE`, `PAYPAL_CLIENT_ID`, and `PAYPAL_CLIENT_SECRET` as environment variables in the environment where Django operates:
  - For Docker, uncomment and enter the values of `PAYPAL_MODE`, `PAYPAL_CLIENT_ID`, and `PAYPAL_CLIENT_SECRET` into `env.default`, using the guidelines above. These values will be loaded into the environment of your Docker container.
  - **For Heroku, use the web interface to add the environment variables** `PAYPAL_MODE`, `PAYPAL_CLIENT_ID`, and `PAYPAL_CLIENT_SECRET` using the guidelines above.
  - Once these environment variables are specified, the `danceschool.payments.paypal` app will then be automatically enabled.

4.2.2 Adding a Paypal "Pay Now" button to the registration page

Because this project is designed to be configurable and to accept different payment providers, the "Pay Now" button is not included by default on the registration summary page (the last step of the registration process). If you have setup your installation by running the "setupschool" script, and if the `danceschool.payments.paypal` app was listed in `INSTALLED_APPS` at the time you did so, then a "Pay Now" button will already be in place.

However, if you have not used the setupschool script, or if you wish to enable another payment processor after initial setup, then adding a "Pay Now" button is very straightforward. Follow the steps for one of these two methods:

Method 1: The Command Line Method (Recommended)

1. Go to the command line in your project's environment or container and type in `python3 manage.py setup_paypal`. The setup script will check that your configuration variables allow you to connect to Paypal, and you will be prompted with the option to add the button plugin to the registration summary page. If the button is already present, then it will not add a duplicate.

Method 2: The CMS method

1. Log in as a user with appropriate permissions to edit pages and other CMS content (the superuser is fine)

2. Proceed through the first two pages of the registration process. Entering fake information is fine, as you will not be completing this registration.
3. When you get to the registration summary page, click the button in the toolbar labeled "Edit Page," then choose "Structure" mode to edit the layout of the page.
4. You will see a placeholder for the payment button, called "Registration_Payment_Placeholder". Click the plus sign (+) next to this placeholder to add a plugin, and from the "Paypal" section of plugins choose "Paypal Pay Now Form"
5. Configure the plugin (choose which pages to send customers to when they have completed/cancelled payment), and you're all set!

To add a gift certificate form to allow customers to purchase gift certficates, follow a similar procedure, adding the "Paypal Gift Certificate Form" plugin to any page of your choosing.

4.3 Stripe

As with Paypal, Stripe integration makes use of a modern API that does not require you to store any sensitive financial information on your own server, and it requires only that you enable the app and place your API keys in your `settings.py` file.

4.3.1 Stripe API Setup

1. Go to Stripe.com and log into your account, or sign up for a new account (**Note:** Before running transactions in live mode, you will need to activate your account, which may involve providing a Tax ID, etc.)
2. In the dashboard on the left hand side, select "API" to get access to your API keys. You will see test credentials, and if your account has been activated, you will also see live credentials. Choose the ones that you need.
3. Next steps depend on whether you have performed a production installation or a development/manual installation of the project:
   - **Development/manual installation**: Edit `settings.py` to add: - `INSTALLED_APPS`: Uncomment/enter `danceschool.payments.stripe` - `STRIPE_PUBLIC_KEY`: Your publishable key. - `STRIPE_PRIVATE_KEY`: Your secret key. **Do not share this value with anyone, or store it anywhere that could be publicly accessed**
   - **Production template installation**: Instead of modifying `settings.py`, you can add the values of `STRIPE_PUBLIC_KEY` and `STRIPE_PRIVATE_KEY` as environment variables in the environment where Django operates: - For Docker, uncomment and enter the values of `STRIPE_PUBLIC_KEY` and `STRIPE_PRIVATE_KEY` into `env.default`, using the guidelines above. These values will be loaded into the environment of your Docker container.
     - For Heroku, use the web interface to add the environment variables `STRIPE_PUBLIC_KEY` and `STRIPE_PRIVATE_KEY` using the guidelnes above.
     - Once these environment variables are specified, the `danceschool.payments.stripe` app will then be automatically enabled.

4.3.2 Adding a Stripe "Checkout Now" button to the registration page

Because this project is designed to be configurable and to accept different payment providers, the "Checkout Now" button is not included by default on the registration summary page (the last step of the registration process). If you

have setup your installation by running the "setupschool" script, and if `danceschool.payments.stripe` was listed in `INSTALLED_APPS` at the time you did so, then a "Checkout Now" button may already be in place.

However, if you have not used the setupschool script, or if you wish to enable another payment processor after initial setup, then adding a "Checkout Now" button is very straightforward. Follow the steps for one of these two methods:

Method 1: The Command Line Method (recommended)

1. Go to the command line in your project's environment or container and type in `python3 manage.py setup_stripe`. The setup script will check that your configuration variables allow you to connect to Stripe, and you will be prompted with the option to add the button placeholder on the registration summary page. If the button is already present, then it will not add a duplicate.

Method 2: The CMS method

1. Log in as a user with appropriate permissions to edit pages and other CMS content (the superuser is fine)
2. Proceed through the first two pages of the registration process. Entering fake information is fine, as you will not be completing this registration.
3. When you get to the registration summary page, click the button in the toolbar labeled "Edit Page," then choose "Structure" mode to edit the layout of the page.
4. You will see a placeholder for the payment button, called "Registration_Payment_Placeholder". Click the plus sign (+) next to this placeholder to add a plugin, and from the "Stripe" section of plugins choose "Stripe Checkout Form"
5. Configure the plugin (choose which pages to send customers to when they have completed/cancelled payment), and you're all set!

To add a gift certificate form to allow customers to purchase gift certficates, follow a similar procedure, adding the "Stripe Gift Certificate Form" plugin to any page of your choosing.

4.4 Square

You are now able to use the popular Square payment processor in place of either Paypal or Stripe. In addition to a standard online checkout option that is similar to Paypal or Stripe, Square *also* allows for easy setup of point-of-sale payments that can be seamlessly integrated with the Django Dance School system, by allowing your registration person to click a button that sends them into the Android or iOS point of sale app, with all details loaded, and by then reporting the results of your transaction back to your website at a special "callback" URL. As with the other payment processors, Square's modern API means that you are not responsible for the storage of any sensitive financial information. For these reasons, Square is a particularly attractive payment option for schools who need to take payments at the door.

Please note that this project uses version 2 of the Square Connect APIs. As of September 2017, this API is only available in certain countries. Please see the Square documentation for more details.

Additionally, please note that *both* the Square point-of-sale integration and the Square checkout form require that you have HTTPS enabled on your site. For the checkout form, any page on which the checkout form shows up must be accessed by HTTPS, or the checkout form will not display. The checkout form *will* work on a local test server without HTTPS for testing purposes only. If you are using the production template on Docker or Heroku, then HTTPS should be enabled by default. Setting up HTTPS in other environments is beyond the scope of this documentation.

4.4.1 Square API Setup

1. Go to Squarup.com and log into your account, or sign up for a new account. Go to the "Dashboard".
2. In the dashboard on the left hand side, select "Apps," then select the tab for "My Apps", and click to define a new set of app credentials that will be used for your website.
3. From the "My Apps" page, click on "Manage App", and you will see the credentials that you need. If you are only seeking to test online payments, then you may opt to use the Sandbox credentials (however, be advised that Sandbox credentials cannot be used to test point-of-sale payments at this time).
4. Next steps depend on whether you have performed a production installation or a development/manual installation of the project:
   - **Development/manual installation**: Edit `settings.py` to add: - `INSTALLED_APPS`: Uncomment/enter `danceschool.payments.square` - `SQUARE_APPLICATION_ID`: Your application identifier - `SQUARE_ACCESS_TOKEN`: Your personal access token. **Do not share this value with anyone, or store it anywhere that could be publicly accessed**
     - `SQUARE_LOCATION_ID`: The first listed value of Location ID listed under "Locations." Please note that the Danceschool project currently does not permit distinguishing among multiple locations in the Square payment system.
   - **Production template installation**: Instead of modifying `settings.py`, you can add the values of `SQUARE_APPLICATION_ID`, `SQUARE_ACCESS_TOKEN`. and `SQUARE_LOCATION_ID` as environment variables in the environment where Django operates: - For Docker, uncomment and enter the values of `SQUARE_APPLICATION_ID`, `SQUARE_ACCESS_TOKEN`, and `SQUARE_LOCATION_ID` into `env.default`, using the guidelines above. These values will be loaded into the environment of your Docker container.
     - For Heroku, use the web interface to add the environment variables `SQUARE_APPLICATION_ID`, `SQUARE_ACCESS_TOKEN`, and `SQUARE_LOCATION_ID` using the guidelnes above.
     - Once these environment variables are specified, the `danceschool.payments.square` app will then be automatically enabled.
5. **If you intend to use point of sale integration, you will also need** to specify a "callback URL," which is the URL that Square's point of sale app sends the details of your transaction to after you successfully complete it using their app. To set this URL, from the "Manage App" page on which you accessed your API credentials, click on the "Point of Sale API" tab at the top of the page. Then, under "Web," look for an input labeled "Web Callback URLs." In this box, enter your callback URL. If you are using the default URL configuration, this URL will be `https://yourdomain.com/square/process_pointofsale/`. However, you can also check to get the exact URL by running `python3 manage.py setup_square` from the command line of your project's environment.

4.4.2 Adding a Square Checkout form and/or point of sale button to the registration page

Because this project is designed to be configurable and to accept different payment providers, Square's checkout form and its point-of-sale button are not included by default on the registration summary page (the last step of the registration process). If you have setup your installation by running the "setupschool" script, and if `danceschool.payments.square` was listed in `INSTALLED_APPS` at the time you did so, then a checkout form and/or point of sale button may already be in place.

However, if you have not used the setupschool script, or if you wish to enable another payment processor after initial setup, then adding the form and button are very straightforward. Follow the steps for one of these two methods:

Method 1: The Command Line Method (recommended)

1. Go to the command line in your project's environment and type in `python3 manage.py setup_square`. The setup script will check that your configuration variables allow you to connect to Square, and you will be prompted with the option to add the checkout form plugin and the point of sale button plugin on the registration summary page. If these plugins are already present, then it will not add duplicates.

Method 2: The CMS method

1. Log in as a user with appropriate permissions to edit pages and other CMS content (the superuser is fine)
2. Proceed through the first two pages of the registration process. Entering fake information is fine, as you will not be completing this registration.
3. When you get to the registration summary page, click the button in the toolbar labeled "Edit Page," then choose "Structure" mode to edit the layout of the page.
4. You will see a placeholder for the payment button, called "Registration_Payment_Placeholder". Click the plus sign (+) next to this placeholder to add a plugin, and from the "Square" section of plugins, choose the plugin that you desire.
5. Configure the plugin (choose which pages to send customers to when they have completed/cancelled payment), and you're all set!

User Permissions for Square Point of Sale Integration

Unlike with online payment solutions, with point of sale payment, you do not want most users to see the point of sale button, and you likely only want it to show up in circumstances where you will be accepting this type of payment (i.e. at the door). So, the following restrictions are in place:

- Only users with the `square.handle_pos_payments` permission set can see the point of sale button. Since superusers have all permissions by default, you will see the button if you are logged into a superuser account. No other users see the button by default, so it is strongly recommended that give this permission to the specific Users who run your registration by going to *Apps* > *Users* on the CMS toolbar.
- Only at-the-door registrations (marked as such during step 1 of the registration process) see the button, regardless of who the user is that is performing the registration.
- For transactions that take place on a platform other than Android or iOS, the point of sale button will display, but it will be disabled and greyed out, to reflect the fact that Square point of sale integration only works on Android or iOS platforms.

CHAPTER 5

Setup: Email

This page is intended as a quick guide to setting up email functionality quickly. For a more thorough discussion of the way that Django handles email providers, see the Django documentation.

The Django Dance School project is designed for use with a transactional email service. Many emails are automated (e.g. registration confirmation emails, password reset emails, and event reminders), while manual ones can be sent to groups of customers or staff using in-built forms.

Until email settings are configured, users will experience errors related to emails as the application will be unable to connect to an email service.

Additionally, because most emails in this project are sent asynchronously, you will need to run the Huey task queue manager. If Huey is not running, then these tasks will be silently queued until it is started. For production installations, this is handled automatically. For development installations, you will need to type `python3 manage.py run_huey` to start Huey.

5.1 Development installations

For development installations in which you need to test email functionality, you will likely need to modify `settings.py` to enter the following settings:

- backend: `EMAIL_BACKEND` (for SMTP, `django.core.mail.backends.smtp.EmailBackend`)
- host: `EMAIL_HOST`
- port: `EMAIL_PORT`
- username: `EMAIL_HOST_USER`
- password: `EMAIL_HOST_PASSWORD`
- use_tls: `EMAIL_USE_TLS`
- use_ssl: `EMAIL_USE_SSL`

5.2 Production installations

The production template for Django-Danceschool has been designed for simple email configuration through the use of SendGrid or Gmail, the latter of which allows you to send up to 100 emails a day with its free plan, and recommended for its simplicity.

To configure either, edit the file `env.default` and scroll down to the `EMAIL_URL` parameter. Two examples are provided to configure emails using Gmail or SendGrid: note that Gmail requires an email address, while SendGrid instead requests the username associated with an account.

If you decide to use Gmail, you may need to enable the 'Less secure apps' access for your account so that your SMTP requests are not rejected, which is explained on this page of Google's support website.

CHAPTER 6

Basic Usage

This page provides an overview of some of the key back-end functionality of the project, so that you can get started using the project right away. Since this project is built on the Django CMS framework, many basic usage tasks are much more thoroughly documented in the Django CMS documentation.

It may also be helpful to have an understanding of the Django admin site, which is documented here.

6.1 Adding/Editing Pages

The Django-danceschool project is built using Django CMS, a flexible and highly customizable content management system framework that is built around a system of pages, placeholders, and plugins. Pages are the basic building block of a website, and each page contains one or more placeholders, which typically correspond to the sections of a page. Within each placeholder is contained one or more plugin instances, which may be used to provide everything from simple text and images to sophisticated functionality. Pages exist in a hierarchy, with some pages existing as children of other pages. By default, Django CMS also adds each page to an automatically-generated menu, and using the included templates, this menu is displayed at the top of the page.

If you are logged in, you will also see another set of menus called the toolbar, with the Django CMS logo on the left and a series of menus and buttons. The toolbar provides links to most of the day-to-day tasks that a staff user will encounter. Users only see toolbar links for the tasks for which they have permissions, so if you are not logged in as the superuser, some toolbar items may be unavailable to you.

If you have run the `setupschool` script, then several key pages that are commonly used on most dances school websites have already been created for you. These pages show up in the menu at the top of the page, and can be edited by following the instructions detailed below.

6.1.1 Adding a New Page

The easiest way to add a new page (once you are logged in), is to click the "Create" button on the right-hand side of the CMS toolbar. You will be given the choice to create either a page at this level of the hierarchy, or a page that is below the current page in the hierarchy. You will be able to input the page's title, an optional slug (for the URL of the page), and some basic text content. For simple, static pages with descriptive text, this will often be sufficient.

6.1.2 Editing an Existing Page

The simplest way to navigate to the page that you wish to edit and click the button to the right of the page that says "Edit page." Once you have done this, you will be in one of Django CMS's primary edit modes: "Structure" mode or "Content" mode. You will also see in the toolbar the buttons to toggle between these two modes.

Why two edit modes? One of the handiest features of Django CMS is "front-end editing."" That is, you can easily make changes to a page and immediately see what they will look like by making edits while looking directly at the page. So, for basic text edits, choose "Content" mode. Then, double-click on the text that you wish to change and proceed to make the edits that you wish.

For more sophisticated changes, such as adding new plugins to the page, you will need to select "Structure" mode. From here, you can add, delete, and rearrange the plugins within the page to enabled new functionality.

Once you have completed your changes, be sure to click "Publish page changes" in the toolbar to make your changes visible to the public.

6.1.3 Changing the Page's template

Often, you want different pages of your site to have different layouts to suit their content or functionality. By changing a page's template, you can quickly modify the layout of a page. This project ships with basic templates to support single-column and two-column layouts, as well as a plain one-column template without the standard menu that may be used for administrative functions. You can also create custom templates to enable any layout that you wish.

To change a page's template, enter Edit page mode for the desired page. Then, select the Page menu within the CMS toolbar, and select a template from the Templates submenu. Note that depending on the template that you have chosen, your page may now have additional placeholders (places within the page to put plugins). These will show up in Structure mode.

Note that as with content changes, you will need to publish the page with your template change so that it can be made public.

6.1.4 Unpublishing/Deleting a page

Enter edit mode for the page in question, and in the Page menu of the toolbar select either "Unpublish page" or "Delete page."

6.1.5 Altering the Page Hierarchy

To alter the page hierarchy (and other mass functions), select the "Apps" menu within the CMS toolbar, and then select "Pages." You will see a collapsible page hierarchy displayed, and this will allow you to make changes such as altering the page hierarchy (by dragging and dropping), publishing/unpublishing pages simultaneously, and other changes.

6.2 Adding your first Event/Series

Under the "Events" menu, choose "Add a Class Series" or "Add a Public Event" to add an event. You will be presented with a form to add the series or event and begin taking registrations. Enter the required information, click "Save," and you're ready to go! For fields that reference another model within the database (such as Location), you can also use the included "Add" and "Edit" buttons to add/edit existing choices.

6.3 FAQs on Adding a Class

6.3.1 What's a Class Description/Where Do I Enter The Class Description?

It is typical for a class series to be offered multiple times. To reduce the amount of duplicated effort in setting up each class series, class descriptions are set up to be reusable. If you have not yet created the class description that you would like, click the "Add" button (plus sign) next to the class description field and create one.

6.3.2 What's a pricing tier/How do I set prices?

Most classes use a common pricing structure. Additionally, if you are using the discounts app, it's often desirable to create discounts that apply to only certain types of classes or events. Therefore, instead of custom per-series pricing, each class or event that has registration requires a Pricing Tier, which defines the base prices for both online registration and at-the-door registration. There is no limit on the number of tiers, so if you need custom pricing for a specific event for some reason, just create a new pricing tier.

If you haven't yet created any pricing tiers, click the "Add" button (plus sign) next to the pricing tier field and create one.

6.3.3 How do I close a series for registration?

By default, Series close for registration a number of days after the first occurrence. This can be set per-event, or it can be set in the site's global settings. To close a series manually, click "Override Display/Registration/Capacity," and then select a value for "Status" such as "Registration disabled" that closes registration.

6.3.4 How do I hide a series from the registration page?

Click "Override Display/Registration/Capacity," and under "Status," choose "Event hidden and registration closed," or one of the other choices that hides the event from the registration page.

6.4 Running Registration (and checking in customers)

It is typical at the outset of a dance class series to "check-in" customers, to ensure that they have registered and paid. This project provides a method for check-in that is straightforward and also mobile-friendly. To access it, click on the "View Registrations" button in the CMS toolbar. You will be presented with a list of recent and upcoming class series. Choose one, and you will see a table with a list of customers who have registered for the class.

The table lists the following:

- The customer's name
- The amount paid for this class. If discounts or vouchers have been applied, then this will list both a gross (undiscounted) price as well as the net (discounted) price.
- The amount paid this customer's registration. A customer who registered for multiple classes simultaneously will have the total amount that they owed for the registration listed here.
- Whether the individual indicated that they are a HS/college/university student (in which case you may wish to check their student ID)
- The customer's email address
- The number of classes for which the customer has ever registered

If the user performing checkins has the appropriate permissions, then on the righthand side will also have a series of buttons the link to the items related to the registration, which may be useful for diagnosing non-payment or other technical issues.

If everything in the database is as expected, and if the customer has paid their entire balance for the registration, then the table will appear as normal. However, in the event of an issues, the price-related cells of the table may be color coded as follows:

- Yellow: Something is wrong or unexpected about this registration. For example, if the customer has not yet paid, or if a cash payment has not yet been marked as received, then this will show up here. Or, if the various tables in the database do not match up in some way (e.g. the total reported in the financial app does not match the total on the registration's invoice), then this will show up here. Use the links at the righthand side of the table to identify the source of the issue.
- Blue: The customer has been refunded a portion of the registration price for either this event, or for another event for which they registered simultaneously. Typically, this will require no further action.

To mark a customer as checked in, simply check the checkbox on the lefthand side. When you have finished checking in customers, be sure to click the "Submit Checkins" button at the bottom of the page so that your changes are recorded.

6.5 Refunding students

To refund students in part or in full for a registration, go to "View Registrations" for the event for which the student registered, and click on the "Refund" button on the righthand side of the table. Enter the amount that you wish to refund, click submit, and click through the confirmation page to process the refund.

If you refund a customer in full for a registration, then that registration will be automatically marked as cancelled, and it will no longer show up in the "View Registrations" page.

Note that automatic refunds depend on the payment processor being used to handle the refund. The Paypal and Stripe apps that are included with this project do handle automated refunds.

6.6 Creating Invoices for Registrations

Whenever a customer goes through the registration process, a unique Invoice is automatically generated for that registration. If the customer pays immediately, then they will typically never see this invoice. However, it is also possible to submit a registration for a customer such that they will receive a link to the Invoice which they can pay at a later date.

To register a customer and send them an invoice, log in as a user with appropriate permissions to register users at the door, and go to your site's Registration page. Before selecting the events for which to register them, select the checkbox at the top of the form labeled "This is a door/invoice registration." Then proceed as normal, entering the customer's information in Step 2 of the registration process.

When you reach the registration summary page in Step 3, you will then see an option at the bottom of the page to send an Invoice to the customer. Enter their email address, click submit, and they will receive an email inviting them to view and/or pay the invoice. Once the invoice has been paid, their registration will be automatically processed.

6.7 Emailing Students

There are many reasons to send mass emails to current or recent students, such as cancellations, class notes, or other announcements. This project provides a simple way to email recent students.

Under the "Staff" menu on the CMS toolbar, select "Email Students." On the form, you will be able to select the series and/or recent months for which you would like to email students. You can enter a custom from address and name, as well as subject and message. Furthermore, if you have set up custom Email Templates, you can also make use of these to simplify the process of sending common emails. Note that at this time, the email students form only accepts plain-text emails (i.e. it does not handle HTML email).

Once you are finished, click "Submit," and you will be prompted for a confirmation before your emails are sent.

6.8 Substitute Teachers

When teachers are unable to teach a scheduled class, it is typical to find another staff member to serve as a substitute. To simplify the process of keeping track of such substitutes, this project includes a substitute teacher reporting form.

To use the form, choose "Report Substitute Teaching" within the "Staff" menu of the CMS toolbar. The form will allow you to indicate the specific classes for which one instructor was a substitute for another instructor. The form does not allow duplicate reports for the same staff member in the same occurrence, so any staff member may fill out the form. Additionally, the form automatically prevents submissions for events in which the instructors have already been paid, to help prevent overpayment issues.

6.9 Class requirements/prerequisites

If you have installed the `danceschool.prerequisites` app, then you have the ability to restrict registration based on whether or not a customer has met certain specified prerequisites. For example, a customer may be required to have taken a certain number of classes at a lower level before registering for a higher level class. These requirements can also be enabled in the global settings to be either "hard" (preventing registration entirely) or "soft" (warning the customer before they are permitted to register). Go to "Registration Requirements/Prerequisites" to manage these prerequisites.

Additionally, some classes, such as audition-only classes, may require explicit permission on a per-customer basis before they can register. If you have created such a requirement, then customers will be unable to register unless you mark them as able to register. To this, go to "Customers" under "Related Items" in the "Events" menu, search for the customer using their name or email address, and add a "Customer Requirement" to indicate whether or not they meet a specific requirement.

6.10 Measuring School Performance (Stats)

If you have installed the `danceschool.stats` app, then you have access to a range of graphs and information to keep track of your school's performance. Some of the things that can be automatically tracked include:

- **Monthly performance:** How many registrations does the school get each month, and how many students are in the average class?
- **Cohort retention:** How many students continue to take classes, vs. how many take only a single class?
- **Performance by class type**
- **Performance by location**
- **Door registrations and student discounts**
- **Marketing referrals:** By linking to special URLs, one can keep track of how many students are registering after clicking on Facebook ads, Google ads, etc.
- **Most active students and teachers**

If you have used the `setupschool` script to prepare your project, then all of these graphs should be automatically shown. Just go to the "Events" menu within the CMS toolbar and select "View School Performance Stats."

Each graph is implemented as a CMS plugin template, so if you have not used the `setupschool` script to automatically add all of the graphs to the Stats page, then you will need to add them manually by going to the Stats page and editing the page in Structure mode, then adding plugins.

CHAPTER 7

Discounts, Vouchers, Gift Certificates

7.1 Overview of Pricing

Most classes use a common pricing structure. Therefore, instead of custom per-series pricing, each class or event that has registration requires a Pricing Tier, which defines the base prices for both online registration and at-the-door registration. There is no limit on the number of tiers, so if you need custom pricing for a specific event for some reason, just create a new pricing tier.

When creating a new class/series, you will be able to select from existing pricing tiers, or create a new one by clicking the "Add" button (plug sign) next to the pricing tier field. Additionally, if you ran `manage.py setupschool` to set up your initial installation, then it is likely that you have already created an initial pricing tier.

7.2 Discounts

If you have installed the `danceschool.discounts` app, then you are able to offer sophisticated discounts based on the content of a customer's registration. Here are some examples of things that the discount app can handle:

- x% discount for all first-time customers
- $y discount for registering for two or more regular classes at once
- An 'all-in pass' price where the user can register for as many classes as they would like for a flat fee
- An 'early bird discount' discount that expires several days before a class series begins
- A student discount that applies only for HS/college/university students.
- A temporary "sale price" discount that automatically expires at the end of a period of time.

Discounts are applied automatically, and the lowest-price available discount in each discount category is automatically applied. As of version 0.3.0, combining multiple discounts on the same registration by putting them in different discount categories is feasible as well. If you're trying to combine discounts, see *Notes on Combining Multiple Discounts* below.

7.2.1 Creating A Simple Discount

All discounts are determined using a straightforward points-based system. Each pricing tier can assigned both a "point group" (a type of points, defined by you) and a number of points. A customer's registration is determined to be eligible for a specific discount if the contents meet or exceed the required number of points needed for that discount to apply.

For example, suppose that in your school, you would like to offer a simple discount of $10 off to students who register for two or more of your regular weekly class series at the same time. To implement this discount, just follow these steps:

1. In the CMS toolbar, go to **Events > Related Items > Pricing Tier** and select the pricing tier that applies to your regular weekly class series to edit it.
2. Under "Pricing tier discount groups," select "Add another pricing tier discount group" to define the point group and the number of points that each regular weekly class receives. For a simple discount such as this, we can simply add a new point group, such as "Individual weekly class series points," and make each item under the regular pricing tier worth one point.
3. Return to the CMS toolbar and go to **Finances > Related Items > Discounts** to access the list of existing discounts, and on the listing page, select the "Add Discount" button to create a new discount.
4. Fill out the information for the new discount. Give the discount a name and category, ensure that the "Active" box is checked, and under "Discount Type," select "Dollar Discount from Regular Price." Then, under "Amount of Dollar Discount," enter 10 for a $10 flat discount. Under "Required components of discount," select the point group that you created (i.e. "Individual weekly class series points") and enter 2 for quantity.
5. Click save and you're done! Now, all students who sign up for two or more regular weekly class series at the same time will automatically receive $10 off the regular price.

7.2.2 More Advanced Discounts (Examples)

Discounts Based on Hours of Class

Because of the point-based nature of the discount system, it is easy to create more sophisticated discounts. For example, suppose that you want to provide a sliding discount based on the number of hours of class that a student signs up for. If you have different pricing tiers corresponding to classes of different lengths (e.g. two weeks of class, four weeks of class, etc.), just give each pricing tier a number of points corresponding to the number of hours (e.g. 2 and 4 points, respectively). Then, define discounts that automatically apply based on the number of those points.

All-in Passes

"All-in passes" are also easy to create using the discount system. When entering the required components for an "Exact Specified Price" discount, simply check the "Applies to all within Point Group" box to ensure that the flat price is applied to the given quantity *or more* or points within the specified point group.

Early Registration Discounts

Creating any type of early registration discount is straightforward. When editing a discount, you will see a field entitled "Must register _ days in advance." Enter the number of days in advance of the beginning of class that you would like students to register in order for the discount to apply. Then, simply enter the components required for the discount to apply (a simple early registration discount that applies to all of your Series classes, for example, may only require 1 point in your default point group).

Notice also that by default, early registration discounts always close at midnight (in your server's local time) at the end of the day in which the discount no longer applies. So, for example, if you are holding a class on Friday at 8pm,

and you specify that students must register 3 days in advance for the early registration discount, then they will be able to receive the discount until 11:59pm on Tuesday evening.

Notice also that in order to receive an early registration discount, it is not necessary that *all* elements of a registration be the specified number of days in advance. It is only necessary that the components needed for the combination to apply are satisfied by elements of a registration that are at least that many days in advance. So, for example, a student who signs up for your Friday evening class on Tuesday afternoon, but who also signs up at the same time for a Wednesday evening class, may be considered eligible for the early registration discount, as long as the points associated with the Friday afternoon class are enough to satisfy the components of the discount.

Student Discounts

As of version 0.3.0, the core app no longer provides separate configurable pricing for high school/college/university students. To set up a student discount, you must now use the discounts app. Fortunately, creating a student discount in the discounts app is simple. In the Discount admin interface, just be sure to select the checkbox labeled "Discount for HS/college/university students only", and also to select the minimum requirements for the discount to apply. Finally, if you wish this student discount to be applied simultaneously with other discounts, be sure that it is defined within its own discount category.

7.2.3 Notes on Combining Multiple Discounts

The discounts app is based on both *points* (the requirements needed for a discount to apply) and on *categories* (different types of discount). By default, the app finds the best discount (providing the lowest price) in each category, and it applies that discount. The order in which discounts are applied is determined by the value of "order" for each discount category, with categories having lower values of "order" applied first.

The conditions needed for a discount to apply are always "x or more" points conditions. This is important, because the conditions required for discounts to apply are often progressive in nature. For example, if you have a discount that applies to students who register for two or more classes, as well as a discount that applies to students who register for three or more classes, then of course, technically, a student who registers for three classes will be eligible for both of these discounts. Similarly, a student who registers for four classes will also be eligible for both of these discounts. However, in both cases, it is unlikely to be desirable to give that student *both* the two-class discount and the three-class discount at the same time. Placing these multi-class discounts in the same category ensures that a customer will not simultaneously receive two or more of the multi-class discounts–only the best discount will be applied. On the other hand, placing these multi-class discounts in different categories ensures by default that a customer *will* be able to receive two or more of these discounts simultaneously. Therefore, it is important to specify and order discount categories carefully.

There is one exception to the above logic. If you have discounts that you do *not* wish to be applied along with other discounts, then you may select the option "Cannot be combined" in the Discount Category admin. Discounts in categories with this box checked cannot be combined with *any* other discount. They will only be applied if they provide a lower price than any other single discount or combination of discounts.

7.3 Vouchers

If you have installed the `danceschool.vouchers` app, then you are able to offer voucher codes, gift certificates, and referral/referree discounts.

Vouchers can be public (e.g. publicly advertised discount codes), or they can be forcibly private (only usable by a specific customer). They can be restricted to apply only to first-time students, or to apply only to prior students. It is also possible to add credits to previously generated voucher codes, which can be useful if you are, for example, providing vouchers to students who volunteer for the school.

7.3.1 Creating Vouchers

To create a new Voucher, just use the CMS toolbar to go to ** Finances > Related Items > Vouchers ** and click the "Add Voucher" button on the listing page. There, you will need to add the following pieces of information

- The voucher code (must be unique)
- A name (to be displayed when the customer enters the voucher code)
- A category (Some basic categories are defined, or you can also create new categories for different types of vouchers you may want to offer).
- An "original amount" for the voucher.

You can also optionally add the following pieces of information/restrictions:

- A description (for internal use only)
- A maximum amount per use. For public vouchers that are meant as discount codes, be sure to enter this field, and to enter an "Original Amount" that is large enough to apply numerous times. E.g. for a $10 discount for the first 100 customers, enter $1,000 as the original amount, and a $10 max amount per use.
- An expiration date. If no expiration is specified, then the voucher never expires.
- Restrictions that limit a voucher to a single use, and restrictions that limit a voucher to use by first-time customers (customers not in the database) or existing customers (in the database) only.
- Restrictions that allow a voucher to be used only for specific dance types/levels, for specific class series (specified by the Class Description), or for specific customers.
- Additional voucher credits, for "topping up" a voucher.

Note on Voucher Restrictions

If specified, voucher restrictions (e.g. based on dance level) require that *all* items within a user's registration meet *one of* the restrictions specified. So, for example, if I want a customer to be able to use a voucher for either our "Lindy 1" or "Lindy 2" classes, I would specify *both* Lindy 1 and Lindy 2 as the dance level voucher restriction. Then, if that customer registers for *either* Lindy 1 or Lindy 2, and they enter the voucher code, they will receive the discount. However, if their registration also includes items that are not Lindy 1 or Lindy 2, then they will be considered to be ineligible for the voucher code.

7.4 Gift Certificates

A gift certificate is simply a voucher code. However, if it is enabled by your payment processor (as it is for the built-in Paypal and Stripe integrations), then it is possible to accept online payments for gift certificates. In this case, a voucher code is generated automatically for the amount paid, and the submitting user is sent an email with a PDF attachment that they can choose to print and give as a gift. The system does all of the work for you, so you don't need to do anything but add the option

To add gift certificate functionality, go to the page where you want to allow users to purchase gift certificates, Edit the page in structure mode, and in the block of the page where you want the purchase button to be located (e.g. "Content"), add either a "Payapl Gift Certificate Form" plugin or a "Stripe Gift Certificate Form" plugin. You will be asked to enter both the default amount of the gift certificate (which can be changed by the customer), and the page to which the customer will be redirected after they have purchased their gift certificate. Save the page, and you're all set!

Both the default text of the gift certificate email as well as the default text of the PDF attachment are loaded as email templates when the school is set up. To modify their contents, just use the CMS toolbar to navigate to ** Content > Manage Email Templates ** and select either the "Gift Certificate Purchase Confirmation Email" or the

"Gift Certificate Purchase PDF Text" templates to edit them. You may also wish to override the template `vouchers/pdf/giftcertificate_template.html` in your custom app to generate a PDF gift certificate with a different layout, or to add your own logo, etc.

7.5 Referral/Referree Program

Some schools like to offer referral discounts to encourage their students to advertise on their behalf. This project provides a simple way of running a referral/referee discount system. If enabled, then each of your customers will automatically be given the ability to refer other customers using a special voucher code. Customers who use this voucher code will receive a "referree" discount when they sign up for their first class, while the customer whose voucher code is used will receive a "referrer" discount.

To enable to the referral program and set the amounts for referree and referrer discounts, use the CMS toolbar to navigate to **Apps > Global Settings**. From there, select the "Referrals" preference page, where you will find a checkbox to enable/disable the referral program, and the ability to modify the discount amounts.

7.5.1 Accessing a Customer's Referral Code

In order to access the referral system, your customers must create a user account. Once they have done so, they can login to access the "My Account" page (It is the page that is automatically shown when a user logs in). On this page, they should see a code under "Customer Referral ID." This is the voucher code that they need to use in order to refer customers. When another new customer enters this voucher code, that (new) customer will automatically receive the referree discount, and the customer whose code is used will receive a referral discount that will be automatically applied against their next registration.

7.5.2 Examples of Usage

- Send an email to customers including their referral code and encouraging them to sign up their friends using it
- Give customers flyers with their referral code written on them and encourage them to post flyers to get discounts

CHAPTER 8

Financial Functions

The `danceschool.financial` app, if installed, provides the ability to do detailed financial accounting of a school's revenues and expenses. Because this system hooks into the core app, it can automatically keep track of revenues received from students registering for classes. Additionally, it can be set to automatically generate expense items for instructors, substitute teachers, venue rental, and other expenses that are either paid by the hour or on a daily/weekly/monthly basis.

This page describes how to submit individual expense and revenue items, how to set up autogeneration of expense and revenue items, and how to generate reports of financial activity.

8.1 Reporting Individual Expense and Revenue Items

For miscellaneous expenses and other overhead, the app includes an expense reporting form that permits easy entry of expenses, including optional file attachment for receipts. To use this form, go to the CMS toolbar and choose *Finances > Submit Expenses.* This form will allow users to submit both flat-price expenses as well as hours of work (for things such as administrative labor). The total expense for hours submitted is calculated using the default rate for the Expense category chosen.

When submitting expenses for reimbursement as opposed to compensation, be sure to check the "Reimbursement" box. This ensures that individuals' taxable compensation is recorded accurately.

Users with sufficient permissions may also have the option to mark expenses as approved and paid at the time of submission. If you record payments in this way, it is strongly recommended that you enter the payment date in which the expense was actually paid. This ensures that your periodic accounting statements remain accurate.

Miscellaneous revenues are less common, but may be used for things like practice sessions where students pay in cash and the standard registration system is not needed. To use this form, choose *Finances > Submit Revenues* from the CMS toolbar. Revenues can be associated with a specific class series or event, and receipts can be attached. If you allow instructors or other staff members to collect cash payments temporarily, then you can use the "Cash currenly in possession of" field to indicate the individual that collected the revenue, so that you can be sure that the money is eventually given to the person responsible for its deposit.

8.2 Automatic Generation of Expense and Revenue Items

8.2.1 Enabling Generation

If you ran the `setupschool` script when installing your project, then you have already selected whether or not to automatically enable automatic generation of expense and revenue items for the above items. However, if you did not take this step, then you can enable automatic generation by going to the site preferences from the CMS toolbar at Apps > Global Settings. From there, select the Financial section, and you will see checkboxes for enabling/disabling automatic generation of expenses. You will also see options to set the categories into which these automatically generated expenses are filed (usually the defaults will be fine), and an option to restrict how far back in time the auto-generation process for Registration proceeds (which you are unlikely to need to change).

There are no further steps to setting up automatic generation of Revenue Items for registrations. However, for venue rental expenses, staff expenses, or other repeated expenses, you will need to set up the rules with which expenses are generated (see below).

Note: By default, automatic generation of expense and revenue items happens once per hour. Through the Repeated Expense Rule admin interface, it is also possible to generate expenses immediately by selecting the rules you wish to apply and choosing "Generate expenses for selected repeated expense rules" from the dropdown. If you are not seeing automatic generation of expenses occurring hourly, be sure that Huey is running. See *Huey (and Redis) setup for production* for more details.

8.2.2 Setting Repeated Expense Rules

There are two ways to set up repeated expense generation rules for Locations and Staff Members:

1. Through the administrative change view for the specific Location or Staff Member whose rules you wish to set/change. Expand the collapsible section entitled "Locations' rental information" or "Staff members' wage/salary information" to edit details from here.
2. Through the Repeated Expense Rule admin, accessible from the CMS toolbar at *Finances* > *Related Items* > *Repeated Expense Rules.*

Rules can be set to generate expenses hourly, daily, weekly, or monthly. The "Day starts at," "week starts at," and "month starts at," parameters define the time point at which expense windows begin and end ("week starts at" and "month starts at" are ignored for expenses that are not weekly or monthly, respectively). And, if you use the admin interface option #2, you also have the ability to specify explicit start/end dates beyond which expenses will not be generated.

For "generic" repeated expenses that are not associated with a Location or a Staff Member, you will need to use the admin interface as specifed in option 2. Here, you will be presented with the option to give this rule a name for record-keeping (e.g. "web hosting fees"), along with options to specify the category of the expense and the recipient of the expense payment. Finally, you have the option to automatically designate these expenses as approved and/or paid.

For expense items associated with Locations or Staff Members, the script generates expenses only in response to the existence of events. So, for example, an hourly expense for an Instructor's instructional time is generated only for the hours in which the individual is scheduled to teach. Daily, weekly, and monthly expenses are generated only if the location or staff member was scheduled in that period. In contrast, "generic" repeated expenses are generated regardless of whether any events were scheduled in the period in question. So, be sure to apply "hourly" and "daily" generic expense rules with caution.

If you have automatically generated expenses erroneously, you may simply go to the Expense Item admin interface from the CMS toolbar at *Finances* > *Related Items* > *Expense Items,* filter by the rule that was used to generate the expenses, and delete the offending expense items. It is also acceptable to modify or delete expense items that are exceptions to the regular rule from this point. However, be advised that if you delete or change the time window of

an expense item and do not update the generation rule that was used to create it, then you may end up with repeated expense items at the next occasion when that rule is applied.

8.2.3 Categories for Staff Members

If you are editing the wage/salary information for an instructor or other staff member, you will see a "Category" field, populated with the different categories of event staffing that you have defined on your site (Class Instruction and Substitute Teaching are automatically specified as categories). This field is optional, but recommended. When the script runs that generates expense items, it looks first for a rule to apply that is specific to the category of staffing that is scheduled. If no such rule is found, it then looks for a "catch all" rule in which no category of staffing is specified. If no such rule is found, then the script does not automatically generate expenses for that staffer.

Note that there can be only one rule per instructor and category, so that there is always a unique rule that is applied for any given staffing of an event.

8.3 Financial Summaries

Because the financial app allows for comprehensive expense and revenue tracking, it is possible to produce simple financial summaries that allow you to get a snapshot of your school's financial performance.

- **Monthly summary:** Go to "View Monthly Financial Summary" in the "Finances" menu of the CMS toolbar.
- **Event summary:** Go to "View Financial Summary By Event." Note that these event-specific entries do not include any revenues or expenses that are not associated with a specific event, such as administrative expenses
- **Detailed categorical breakdowns:** Go to "Detailed Breakdown" and select the period desired, or select a specific month from the monthly summary page. From the detailed categorical breakdown, you can can also quickly find links to specific expense and revenue items in order to make corrections.

Note also that for accounting purposes, monthly and detailed summaries can be constructed on several different bases, which can be selected within each page:

- **Payment basis:** Summaries are constructed based on when money is received or spent. This is what is typically used for "cash accounting" for tax purposes.
- **Accrual basis:** Summaries are constructed based on a notion of when the money is "owed" (e.g. at the end of a class series for instructors). This can help to get a more accurate picture of your financial performance Please be advised that the accrual basis constructed in this project almost certainly does not correspond to generally accepted accounting practices, and therefore it is not recommended to use these statements as a basis for so-called "accrual accounting" for tax purposes.
- **Submission basis:** Summaries are constructed based on when revenue and expense items are submitted to the database.

8.4 Exporting Financial Data

If you need to export financial data for analysis using another method (e.g. Excel), you can do so from the Monthly Financial Summary view by clicking the buttons under the "Export Financial Data (CSV)" heading.

CHAPTER 9

Advanced Usage

9.1 Private Events

If you have enabled the `danceschool.private_events` app (enabled in the default setup), then you have the capability to keep track of both publicly scheduled events and class series as well as private events and reminders in a handy calendar view. This feature is flexibly designed to allow you to keep track of the use of private spaces, schedule of staff meetings, set up to-do reminders, etc.

To access the calendar view, simply go to *Events* > *School Calendar* on the CMS toolbar. You will see a calendar with several display options:

- Display all public and private events (default)
- Display only public events
- Display only your personal calendar, including only events for which you are an instructor or other type of staff member.
- Display a calendar of public and private events restricted by location.

Clicking on any event on the calendar will provide you with an option to edit details for that event if you have the permissions to do so, and it will provide a link to the URL for the event if there is one.

The easiest add a new private event is simply to click on the day on which you wish to schedule the event. You will see options to create repeated events, options to restrict the visibility of the event to only a subset of staff members on your site, and options to send email reminders to yourself or to others about the event.

If you have enabled the private lessons app (described below), then you will also see scheduled private lessons on the private calendar app. Depending on your user's permissions, you may see only the lessons for which you are scheduled to be an instructor, or you may see all scheduled private lesson events.

9.2 Private Lesson Scheduling

If you have enabled the `danceschool.private_lessons` app, then you have access to a full private lesson scheduling system. The private lesson system leverages the same pricing tier structure that is used to price registration

for class series and public events. This means that it's easy to set up sophisticated discounts and use vouchers, just like one does for public lesson scheduling.

The `danceschool.private_lesson` app is not installed by default, but installation is easy. Just take the following two steps:

1. In your project's `settings.py` file, uncomment the line that lists `danceschool.private_lessons` under `INSTALLED_APPS`. If you do not see this line, then you may add the line yourself after the other danceschool apps that you have enabled are listed.
2. At the command line for your project's environment, run `python3 manage.py migrate` to set up the database for private lesson scheduling.

If you enable private lesson scheduling, it is also strongly recommended that you enable the `danceschool.private_events` app, so that instructors have calendar access to see the lessons that are scheduled for them.

9.2.1 Configuring Private Lesson Scheduling

There are several different ways in which studios typically handle private lesson scheduling:

- Full online registration, including online pricing and payment
- Online registration and scheduling, without the option for online payment (for example, if students are asked to pay instructors directly).
- No public registration, only private scheduling (for example, for over-the-phone scheduling only)

The private lesson scheduling system is designed to handle each of these cases seamlessly, although by default it is set up for full online registration. There are also a number of other useful configuration options. To set these, go to *Apps > Global Settings* in the CMS toolbar and select the "Private Lessons" section to begin configuration.

Unless you have set up the private lessons app prior to running the `setupschool` script, you will also need to add a link to your site's menu to allow customers to access the private lesson scheduling system, and you may also wish to set up permissions for different Users or Groups to access the system appropriately.

9.2.2 Setting up Instructor Availability

Instructors are not set up to permit private lesson booking for them by default. Before customers can sign up for private lessons with a particular instructor, three things need to happen:

1. The "Available for private lessons" flag needs to be set on the Instructor's record. Go to the "Manage Instructors" admin view from the Staff menu of the CMS toolbar, select an instructor, and scroll down to set this flag.
2. Additional "private lesson details" information needs to be specified for the instructor. These include a default pricing tier for the instructor's lesson slots, the roles for which that instructor is willing to teach lessons, and flags that indicate whether they are available for lessons with couples or small groups. This information can be set from the same view specified in Step 1 above.
3. Instructor availability slots must be created so that there are specified time slots in which customers can sign up for lessons.

Instructor availability for private lessons is based on slots, which are set by going to *Staff > Private Lesson Availability* in the CMS toolbar. Depending on your user's permissions, you may have the ability to set availability for each of the school's instructors, or only for yourself.

To add new slots, simply click and drag over the time period in which you would like to create availability slots. In the pop-up modal, set the initial status for these slots (usually "Available"), and optionally set the pricing and location in which any lesson in that slot will occur. If you need to set larger blocks of time at once (for example, several hours

of availability each day), you may do so from the "month" view of the calendar. Also, note that if no pricing tier is selected, then customers will not be able to pay for their lesson online; their lesson will simply be scheduled.

All private lesson pricing is per slot. If you wish to provide pricing for private lessons that is "non-linear" (i.e. with lower per-minute pricing for longer lessons), then the best way to accomplish that is using the discount system. Generally, the best approach is to create a separate "point group" for private lessons, so that the discounts applied to private lessons may be fully separate from the discounts applied to class series and public events.

9.2.3 Booking Lessons

For staff users, the booking view can be accessed directly by going to *Staff* > *Schedule a private lesson* in the CMS toolbar.

Once a private lesson is booked, by default a confirmation email will be sent to both the student and the instructor of the lesson. If you do not desire the instructor confirmation email to be sent, you may do so from the Private Lessons site preferences.

Once scheduled, private lessons also show up on the instructor's private calendar.

9.3 Groups and Permissions

9.3.1 Overview

For a more general overview of Django's permissions system, see The Django Documentation.

When you installated this project, you created a superuser. In general, this user automatically has permissions to do anything and everything on the site. However, for larger schools, there are typically different types of

All permissions in Django may be granted on either a per-user or a per-group basis. So, for example, if I have two instructors named Alice and Bob, and I want to give them permission to email students, I can either individually give them permissions to email students, or I can create a Group (say, `Instructors`), make them each members of the group, and then simply give the permission to email students to anyone in that group.

By default, the `setupchool` script creates three primary groups, which correspond to a typical stratification of user roles and permissions:

- **Board:** This group is designed for the individuals who manage the school. By default, the Board group has permissions to do all day-to-day tasks. For security reasons, a handul of database operations still require the superuser by default, including modifying Groups and permissions.
- **Instructors:** This group is designed for instructors, who do not need permission to edit most content on the site, but who may run the registration process, submit expenses and revenues, email students, report substitute teaching, etc.
- **Registration Desk:** This group is designed for users who run the registration process (including at-the-door registrations). By default, these users can do all registration-related tasks except for processing refunds. In addition, they cannot email students, submit expenses or revenue outside of the normal registration system, or otherwise edit the site's content.

Remember, you can always edit the permissions given to each group (as well as create/delete groups) by logging in as the superuser, going to "Administration" in the apps menu, and then choosing "Groups" under "Authentication and Authorization."

9.3.2 Updating User Permissions on Upgrade

If you have recently upgraded your version of the project to one that has new features, then your users will not automatically be given new permissions to manage those new object. Fortunately, if you have used the default groups created by the setupschool script, it is easy to keep those permissions up to date. From a command line in your project's environment, just type `python3 manage.py setup_permissions`, and all of the default permissions, including any permissions associated with new features, can be granted to the "Board", "Instructors," and "Registration Desk" groups created by the script. No permissions are removed by this procedure, so any custom permissions that you have set at the User or Group level will not be impacted by doing this.

9.3.3 Detailed List of Permissions

In addition to the permissions automatically generated by Django (add/edit/delete permissions for each Model), this project defines the following permissions which are used to enable/disable various functionality on a per-user basis.

Core app

Name	Description
view_staff_directory	Can access the staff directory view
view_school_stats	Can view statistics about the school's performance.
update_instructor_bio	Can update instructors' bio information
view_own_instructor_stats	Can view one's own statistics (if an instructor)
view_other_instructor_stats	Can view other instructors' statistics
view_own_instructo_rfinances	Can view one's own financial/payment data (if an instructor)
view_other_instructor_finances	Can view other instructors' financial/payment data
report_substitute_teaching	Can access the substitute teaching reporting form
can_autocomplete_users	Able to use customer and User autocomplete features (in various admin forms)
view_other_user_profiles	Able to view other Customer and User profile pages
view_registration_summary	Can access the series-level registration summary view
checkin_customers	Can check-in customers using the summary view
accept_door_payments	Can process door payments in the registration system
register_dropins	Can register students for drop-ins.
override_register_closed	Can register students for series/events that are closed for registration by the public
override_register_soldout	Can register students for series/events that are officially sold out
override_register_dropins	Can register students for drop-ins even if the series does not allow drop-in registration.
send_email	Can send emails using the SendEmailView
view_all_invoices	Can view invoices without passing the validation string.
send_invoices	Can send invoices to students requesting payment
process_refunds	Can refund customers for registrations and other invoice payments.
choose_custom_plugin_template	Can enter a custom plugin template for plugins with selectable template.

Financial app

Name	Description
mark_expenses_paid	Mark expenses as paid at the time of submission
export_financial_data	Export detailed financial transaction information to CSV
view_finances_bymonth	View school finances month-by-month
view_finances_byevent	View school finances by Event
view_finances_detail	View school finances as detailed statement

Prerequisites app

Name	Description
ignore_requirements	Can register users for series regardless of any prerequisites or requirements

Banlist app

Name	Description
view_banlist	Can view the list of banned individuals.
ignore_ban	Can register users despite banned credentials

Private lessons app

Name	Description
edit_own_availability	Can edit one's own private lesson availability.
edit_others_availability	Can edit other instructors' private lesson availability.
view_others_lessons	Can view scheduled private lessons for all instructors

CHAPTER 10

Basic Customization

10.1 Customizing Email Templates

By default, the site sends out a confirmation email whenever a customer successfully completes their registration and submits payment. It also sends out a confirmation email when a customer purchases a gift certificate. The templates for these emails are completely configurable, and they are stored in the database, so you can customize them without requiring access to the underlying code. The first time that you run the server, the templates are populated with default content using

To edit these email templates (and to create other custom email templates for your own purposes), simply log in as the superuser (or another user with appropriate permissions) and go to http://yoursite/admin/core/emailtemplate/. You will see the templates listed there, simply click on them and edit as needed.

Note also that these custom email templates are processed much like standard Django templates, with the exception that some functionality is disabled for security purposes.

TODO: Explain further.

10.2 Customizing Page Templates

You will almost certainly want to customize your site's layout and look somewhat, that means that you will need to add one or more custom templates to your project. To understand how to adapt custom templates for your site, you should first understand that Django uses something called *template inheritance*. That is, if you want to define a specific template for a specific page, it is generally not necessary to recreate all of the logic and code to describe the way that the page is laid out. Rather, you can create a custom template that inherits from another, more general template, changing only the pieces of the page that differ from the parent template.

Many templates are also designed not for laying out an entire page, but for laying out only one section of a page. For example, the navigation section of a page is often the same across all public-facing pages, but it may be more convenient to keep the navigation layout in a separate file and simply use an `{% include %}` tag to include it in other templates as needed. Similarly, CMS plugins that are used to display pieces of information like lists of upcoming classes or lists of instructors use templates to describe how that information should be laid out.

With that in mind, most projects will need to override only a couple of key templates in order to accomplish the vast majority of customization desired (all of these templates are located in `danceschool/core/templates/`):

- `cms/home.html`: The base template for all public-facing pages. By default, this shows all information in a single column, and all of the other templates that are included for public-facing pages (`twocolumns_rightsidebar.html` and `twocolumns_leftsidebar.html`, as well as various other templates) inherit from this template.
- `cms/navbar.html`: The template that is used to show the navigation at the top of the page. By default, this template produces a dropdown menu that goes across the top of the page, with two levels of pages displayed.
- `cms/admin_home.html`: The base template for all private and administrative within-site functions, such as the various reporting forms and financial summaries. The defaults for this template are very plain but also very usable, so you may find that you do not need to override this template at all.

All templates can be overridden, but here are a few other templates that you may wish to consider overriding:

- `core/event_registration.html`: The template used for the first step of the registration process.
- `core/individual_class.html` and `core/individual_event.html`: The templates used on the automatically-generated pages for each class and/or event.
- `core/account_profile.html`: The template used for the "customer profile" page that is displayed when a customer logs in. If you are not allowing customers to sign up or log into the site, then you will likely not need to change this template.

10.2.1 Where should I put my custom templates?

When looking for a requested template, Django uses the first template with the appropriate file name that it encounters. So, when providing custom templates, there are two places to put them:

1. In a `templates` folder within the root folder of your project
2. In the `templates` folder of a custom app that is listed in INSTALLED_APPS *before* the original template's app.

Notice also that templates in this project are *namespaced*, meaning that they are contained within a subfolder with the name of the app for which they are designed. So, if I have created a new `cms/home.html` template, which defines the basic layout for public-facing pages, I can either save it as `<BASE_DIR>/templates/cms/home.html`. or I can save it as `<BASE_DIR>/my_custom_app/templates/cms/home.html`, where `my_custom_app` is the name of an app that has been added to INSTALLED_APPS before danceschool.core.

10.2.2 Custom Django CMS Templates

Django CMS (the content management system that is used to manage most public-facing pages) allows you to select the appropriate template for each page. However, not all templates are designed for laying out CMS pages. By default, the project provides a few CMS-appropriate templates:

- `cms/home.html`: For public-facing one-column layouts
- `cms/twocolumn_rightsidebar.html`: A two-column layout with a main "content" region on the left-hand side and a sidebar on the right.
- `cms/twocolumn_leftsidebar.html`: A two-column layout with a main "content" region on the right-hand side and a sidebar on the left.
- `cms/admin_home.html`: A one-column plain layout for administrative functions.

If these templates are insufficient for your needs, you may wish to add entirely new templates, not just to override preexisting templates. For example, perhaps you want the front page of your site to be a splash page, which looks different from the more content-focused pages of your site. In that case, you will need to do the following:

1. Add your custom template to either the `templates` folder of your project's root directory, or to the templates folder within a custom app.
2. Add the template's filename and a brief description to the setting `CMS_TEMPLATES` within your project's settings.py
3. Restart the server for your Django project so that the settings are reloaded.

Once you have done these steps, you should see your custom template available as an option for any new or existing pages that you create.

10.2.3 Sources of Templates to Customize

Although you have complete control over the layout of your site using custom templates, it is often handy to work from a pre-existing template. To assist in this process, this project is built using the popular Bootstrap CSS and Javascript framework. There are many existing free and paid themes available that are built on the Bootstrap framework. Here are a couple of sources for these types of templates:

- Start Bootstrap
- BootstrapMade
- Bootswatch

For more details on how to customize templates for use with Django CMS, see the Django CMS Documentation.

For more general information on Django templates, how they work, and how to customize them, see the Django Documentation.

CHAPTER 11

Advanced Customization

11.1 Customizing the Registration Form

Since all danceschools operate somewhat differently, it is common for schools to wish to collect custom information during the registration process. By default, this project's registration process proceeds in three steps:

1. Choose the classes/events that you wish to register for
2. Enter your contact information, any voucher codes that you wish to use, etc.
3. Finalize your payment (using Paypal's pay now functions, or by submitting information in a door registration)

Most of the time, when a studio wants to customize the information that they collect, they wish to do so in step 2. So, this project has been designed to make it relatively easy to do this, using the power of Python's class inheritance.

Before proceeding, if you are unfamiliar with Django (or with object-oriented programming), you will need to understand the meaning of a couple of terms:

- A *class* is a generic type of object, which you can often think of as representing a type of real world object. Classes can contain *properties* (e.g. if we had a Dancer class, it could have a property `defaultRoles` that provides a list of roles that the dancer dances, such as "Lead" and "Follow") as well as *methods,* which are, in essence, functions within the class that define ways of interacting with the class (e.g. our Dancer class could have a method `askToDance()` that responds with either "Yes" or "No" depending on whatever logic we want to implement).
- An *instance* of a class represents one object within the class. So, each dancer in a ballroom might be associated with one instance of a Dancer class. Properties are stored for each instance. So, for example, one Dancer instance might have only "Follow" in `defaultRoles`, while another might have both "Lead" and "Follow."
- A *Form* refers to the class which defines which fields are displayed, how they are displayed, and how they should be validated.
- A *View* refers to the class or function which decides what is displayed when a request is made, including (for example), the displaying of form. In the case of a page displaying a form, it also determines what should be done when a form is valid.

- A *Model* refers to the class which is used to define a specific piece of data (like a row in a table representing a Registration, for example).

One last very important thing: classes can inherit from other classes. So, for example, if I wanted to create a DanceCompetitor class, with properties and methods that are specific to competitors, I wouldn't need to redefine all of the properties and methods associated with a DanceCompetitor. I could, instead, have the DanceCompetitor class inherit those things from the Dancer class. In that case, all DanceCompetitor instances would also be Dancer instances, while not all Dancer instances would necessarily be DanceCompetitor instances.

Now that we have that out of the way, here are the steps to customizing your registration form. These should all be added to your custom application, and that application must be listed *before* the `danceschool.core` app under `INSTALLED_APPS`.

1. Subclass the RegistrationContactForm (located in `danceschool.core.forms`) to create your own custom form in its place.

   The RegistrationContactForm class, like several other forms in this project, uses the app django-crispy-forms to make it easier to customize functionality and display. So that you do not need to re-specify all of the fields in the form, the RegistrationContactForm conveniently provided three methods, `get_top_layout()`, `get_mid_layout()`, and `get_bottom_layout()`, each of which provides a django-crispy-forms Layout object that includes the fields in that portion of the form. So, for example, if I want to add a new field called "favoriteDancer" to the bottom portion of the form, I can simply override the method `get_bottom_layout()` as follows:

```
from django import forms
from danceschool.core.forms import RegistrationContactForm

class MyCustomForm(RegistrationContactForm):
    favoriteDancer = forms.CharField(label='Name Your Favorite Dancer',␣
↪required=False)

    def get_bottom_layout():
        layout = super(MyCustomForm,self).get_bottom_layout()
        layout.append('favoriteDancer')
        return layout
```

   Additional details on working with Django-crispy-forms for form customization can be found in its documentation on Layouts.

2. In your app's `urls.py`, override the default URL for the view `getStudentView` to use the newly-created form. For example, if the registration contact form is normally found at the url /register/getinfo/, then you can add the following to your app's `urls.py`:

```
from django.conf.urls import url
from danceschool.core.classreg import StudentInfoView
from .forms import MyCustomForm

urlpatterns = [
    # This should override the existing student info view to use our custom form.
    url(r'^register/getinfo/$', StudentInfoView.as_view(form_class=MyCustomForm),␣
↪name='getStudentInfo'),
]
```

3. That's it!

11.1.1 But what happens to the data from my custom form field?

In anticipation of the fact that many dance schools need to ask custom questions at registration time, the TemporaryRegistration and Registration models have a field called data which can hold arbitrary form data from the registration process. The contents of the data field are serialized into a JSON object, so the data are stored as a set of key-value pairs. By default, any additional data that you collect during the registration process will be saved to the data field of the associated TemporaryRegistration. When that customer has completed their payment, then the data are transferred to the Registration object as well.

11.2 Processing custom fields in the registration form using built-in signals

When a TemporaryRegistration is created (right before the user is given options for payment), and when a Registration is finalized after payment has been processed, the registration system sends a *Signal*, which can be handled by your own custom signal handlers to do further processing based on the data.

For example, suppose that you have some mailing list functionality in a separate app, and when a registration is complete, you want to see whether they checked the box requesting to be added to the mailing list, so that you can add them to the mailing list. In your custom app, define a signal handler that listens for and receives signals from the `post_registration` signal. That signal will automatically pass the finalized registration information to your handler function, and from there, you can proceed to sign the user up for the mailing list if they requested it.

For more details on Django signals and signal handlers, see the Django documentation.

11.3 Adding a Custom Payment Processor

The danceschool project supports two of the most popular online payment processors (Paypal and Stripe). It also contains basic functionality for keeping track of cash payments. However, depending on your location, you may want or need to accept online payments from other payment processors.

Since payment processors vary in the way that they handle transactions from websites, this documentation cannot provide comprehensive instructions. However, this document provides a starting point for understanding how to implement a custom payment processor. If you are attempting to do this, it is highly recommended to look at the code for the existing payment processor apps, `danceschool.payments.paypal` and `danceschool.payments.stripe`, to see how they work.

11.3.1 The PaymentRecord Model

The `danceschool.core` app provides a PaymentRecord model that is designed to be a polymorphic model using the *Django-polymorphic <http://django-polymorphic.readthedocs.io/en/stable/>* app. It provides several key fields that are common to all payment processors:

- A foreign key relationship to the Invoice model (since payments are associated with invoices).
- Creation date and modification date fields
- Several methods and descriptive properties that may need to be overridden on a per-payment processor basis, such as the `refundable` property to indicate whether a payment is refundable, and the `refund()` method to actually process a refund.

To create a new payment processor, first create a new model in your app's `models.py` that simply subclasses the `danceschool.core.models.PaymentRecord` model. Because of the way Django-polymorphic works, your payments will now be recognized just as payments from other payment processors.

Add any fields that your particular payment processor may need (for example some kind of transaction identifier field). Then, be sure to override the following from the parent model:

- The `refundable` property (decorated method): defaults to False. This can usually just return True if your payment method is refundable.
- The `recordId` property (decorated method). This will usually just return the identifier used by the payment processor, but since different payment processor apps must store this information differently, `recordId` ensures that the information is always available to the parent app.
- The `methodName` property (decorated method). This just returns a readble name for the type of payment processor used, such as "Paypal Express Checkout" or "Stripe Checkout."
- The `netAmountPaid` property (decorated method). This should return the amount that was paid *net of any refunds*.
- The `refund()` method. If your API allows refunds of transactions, this should be handled here. An `amount` parameter should be accepted to permit partial refunds.
- The `getPayerEmail()` method. This method should return the email address of the person who paid, in case they need to be contacted. Many payment processors store this information automatically, but if yours does not, then you can potentially create a model field to store it.

11.3.2 Payment Processor Views

Your payment processor will need to define a view that receives data from the processor's website. The view also needs to do the following:

1. Determine whether a payment is being made on a Temporary Registration or an existing Invoice.
2. If a payment is being made on a Temporary Registration, then it needs to create a new Invoice using the `get_or_create_from_registration()` class method of the Invoice class.
3. If the payment is successful, then your view should call the `processPayment()` method of the associated invoice to record that the payment has been made. The `processPayment()` method will handle finalizing the registration if applicable, sending the appropriate email notifications, etc.
4. Your view will either need to return an `HttpResponseRedirect()` to an appropriate success URL, or your plugin template will need to use Javascript to redirect the user after a successul payment is made (or notify the user if a payment is unsuccessful).
5. Additional steps may be necessary if you intend to use your payment processor to allow customers to purchase gift certificates (as the Paypal and Stripe apps allow).

It is highly recommended that you follow along with the `views.py` of the existing payment processors to be sure that you follow the appropriate steps.

11.3.3 Creating a CMS Plugin

To add a checkout button for your payment processor, you will need to create a CMS plugin that can be added to the page Placeholder where checkout happens. To do this, create a file in your app called `cms_plugins.py`, and define a new plugin here as a class that inherits from `cms.plugin_base.CMSPluginBase`. Within the plugin class, specify template the contains whatever is needed for your payment processor, and use a custom `render()` method to add any additional context data that may be needed for your page.

The existing payment processor apps are a good resource for understanding how to implement one of these plugins. It is also a good idea to read the *Django CMS documentation <http://docs.django-cms.org/en/release-3.4.x/how_to/custom_plugins.html>* to learn more.

Once you have created your CMS plugin, you will need to manually add it to the "Registration Summary" page. To do so, follow these steps:

1. Log in as a user with appropriate permissions to edit pages and other CMS content (the superuser is fine)
2. Proceed through the first two pages of the registration process. Entering fake information is fine, as you will not be completing this registration.
3. When you get to the registration summary page, click the button in the toolbar labeled "Edit Page," then choose "Structure" mode to edit the layout of the page.
4. You will see a placeholder for the payment button, called "Registration_Payment_Placeholder". Click the plus sign (+) next to this placeholder to add a plugin, and from the available plugins choose your payment processor's plugin.

CHAPTER 12

Version History

12.1 0.8.6 - April 22, 2019

- **NEW:** Financial performance by date (uses same view as financial performance by month)
- Added ability to submit class registrations via Ajax and receive a JSON response
- Fixed RevenueItem paymentMethod not editable in admin
- Documentation improvements re: emails
- Miscellaneous bug fixes

12.2 0.8.5 - April 3, 2019

- Added view for manual generation of repeated financial items, customizable by rule.
- Fixed event not editable on RevenueItem admin.
- Improved reference links in admin between revenue/expense items, events, and event financial detail views.
- Fixed #139, instance_of reference in EventAutocompleteForm creating issues with initial migration because of content_type reference.
- Re: #140, pinned Huey to version <2.0 to avoid compatibility issues.

12.3 0.8.4 - March 22, 2019

- Added EventAutocompleteForm for easier selecting of events in Expense/Revenue reporting forms (#134)
- Improved Revenue Reporting form by adding adjustments/fees and adding ability to mark revenue as received (#135)
- Fixed PublicEvent model showing UTC instead of local time (#136)

- Added direct registration link to PublicEventAdmin (#125)
- Fixed bug that excluded prior-year expenses from financial detail view for events (#137)
- Misc. bug fixes

12.4 0.8.3 - March 19, 2019

- Added event-specific financial detail view
- Fixed issue with financial detail view with explicit start/end dates
- Fixed issue with reverse() call when prior site history is missing (e.g. viewing an invoice directly from an email link)
- Fixed extra column with total registrations in finances by event view

12.5 0.8.2 - February 24, 2019

- Fixed issue with ExpenseItem changelist form treating payTo as a required field
- Fixed issue with display of most popular vouchers in school stats (#120)
- Fixed access to QuerySet object in core/handlers.py (#121)
- Fixed configuration issue with Redis dependency (#122)
- Fixed issue with reversed occurrence dates in individual series view
- Fixed incorrect page template for staff list in setupschool script (#115)

12.6 0.8.1 - January 9, 2019

NOTE: To avoid an issue in migrations, it is recommended to upgrade directly to this version or higher and skip version 0.8.0.

- Fixed issue with financial app migration arising from lack of User and StaffMember methods available in migration.

12.7 0.8.0 - January 8, 2019

NOTE: The upgrade to version 0.8.0 makes database migrations in the way that financial records are kept that are not designed to be reversed. It is *strongly* recommended that you backup your site's database immediately before upgrading.

- **NEW**: Pay at the door payment processor app that allows customers to commit to pay at the door, and individuals running registration at the door to rapidly process at-the-door cash payments.
- Substantial under-the-hood improvements to the way in which financial records keep track of transaction parties.
- Month and weekday names now sort logically rather than alphabetically in EventListPlugin as well as registration pages.
- New site-history helper function that improves UX in the admin by redirecting users back to the appropriate previous pages.

- Fixed issues with Square point-of-sale and refund processing callbacks arising from an API change.
- Fixed version incompatibility with Django-easy-pdf (for gift certificates)

12.8 0.7.2 - November 20, 2018

- Added default compensation by staff category, with an updated action for resetting/deleting staff member custom compensation.
- Added notes field to manually-added guest list entries.
- fixed EventOccurrence string format issue.
- improvements to EventStaffMemberInline.

12.9 0.7.1 - November 13, 2018

- Vouchers can now be restricted to specific series/event categories or sessions (#98)
- Payment processor scripts updated to reflect changed CMS logic (#97)
- Fixed timezone issues with "Duplicate events" view

12.10 0.7.0 - November 13, 2018

NOTE: After upgrading to 0.7.0, it is recommended to run `python manage.py setup_permissions` to ensure that staff have appropriate permissions for the new guestlist app.

- **New:** Customizable guest lists by individual event, category, or session, with rules for adding staff members.
- Improved management of staff members and instructors in the admin (Instructor is now non-polymorphic).
- Misc. bug fixes and improvements.

12.11 0.6.3 - September 21, 2018

- Fixed bug that led EventListPlugin instances to differ between draft and publication.
- Fixed event registration card spacing on mobile.
- Fixed margins on instructor images in Instructor list template.

12.12 0.6.2 - September 19, 2018

- **New:** Added short description to Event and submodels.
- Improvements to Event List plugin for greater configurability and filtering.
- Added DJs as a default event staff member category.
- Fixed issues with category-specific templates

12.13 0.6.1 - September 18, 2018

- Updated use of Square API to reflect new method of loading access token.

12.14 0.6.0 - September 9, 2018

- **New:** Themes app for easier customization of your initial site templates. The project now uses the djangocms-bootstrap4 app by default as well, for much easier development of sophisticated layouts.
- **New:** Event "sessions" that can be used to group events for registration. The registration page is also much more easily reorganized without creating custom templates, by choosing the default organization method in registration site settings.
- **New:** Customer groups, to which customers can be easily assigned. Both discounts and vouchers may be group-specific as well as customer- specific. And, through an admin action, it is easy to email a group of customers all at once.
- **New:** Discounts that apply based on the number of existing registrants, including temporary registrants that are still in the registration process. This makes it easy to employ popular "First X to register" special pricing.
- Fixed issue with admin template that led many sideframe pages to have scrolling disabled (#89)
- Draft FAQs are now properly treated as drafts, and can be published and unpublished by admin action (#96 and #95)
- Added dates to refund view for easier processing (#87)
- Fixed discounts not showing up when viewing registrations (#88)
- Removed vestiges of prior Python 2 support

12.15 0.5.6 - August 8, 2018

- Fixed xhtml2pdf version incompatibility issues.

12.16 0.5.5 - April 26, 2018

- Fixed banlist module reference issue.

12.17 0.5.4 - April 26, 2018

- Fixed html5lib version inconsistency issue.
- Fixed missing URLs for djangocmsforms app.
- Simplified README.
- Misc. cleanup

12.18 0.5.3 - April 14, 2018

- Improved admin listing of expense items.
- Fixed duplicate listing of venue expense items in FinancialDetailView.

12.19 0.5.2 - February 21, 2018

- Discounts now show up on the registration cart page, not just the final page (#79)
- Fixed issue with change in name of CKEditor theme (#83)
- Fixed Django 1.11 migration issues with registration template
- Permitted configurable rules for determination of event months

12.20 0.5.1 - February 7, 2018

- Updated to use Django 1.11 and Django CMS 3.5
- Overhaul and simplification of event templates
- Fixed dependency version issues with Django-dynamic-preferences and django-polymorphic apps
- Misc. bugfixes and linting improvements

12.21 0.5.0 - October 3, 2017

- **New:** All templates have been overhauled to use the latest Bootstrap 4 beta. This will ensure long-term compatibility of your website design.
- **New:** Discounts can now be customer specific, so that they will only be available to certain customers.
- **New:** Additional stats charts are now available, including information on the usage of discounts and vouchers, details regarding time of advance registration, and details on multi-class registrations.
- Improved the templates for stats charts for more consistent formatting using Bootstrap 4 cards.
- Fixed issue with refunding sales tax for complete refunds.
- Fixed issue with footer templates repeating on certain pages. Footer is now a static placeholder by default for easy editing.
- Registration page templates now use Bootstrap 4 cards for easier themeing and configuration
- Added management tasks for all cron jobs for easier Heroku integration
- Numerous small bug fixes and template improvements.

12.22 0.4.1 - Septmeber 19, 2017

- Fixed bug with iCal calendar feed slicing in the core app
- Fixed bug with discount categories that have no applicable discount ordered before discount categories with applicable discount codes

- Fixed template inheritance issue on registration offline template.

12.23 0.4.0 - September 14, 2017

- **New:** Square payment processor integration, with the option for both online payments and point-of-sale transactions with a Square card reader.
- **New:** A full private lesson scheduling system, with the ability to either use the default registration and pricing tier system, or the ability to do scheduling only. Includes notifications for instructors and students, and scheduled lessons automatically show up on the instructor's private internal calendar.
- **New:** More flexible internal calendaring options, including the option to view internal calendars by location and by room
- **New:** The ability to create generic invoices for non-registration items, specify specific invoice recipients, and easily email notification updates to invoice recipients.
- Private events can now specify rooms as well as locations, and will show up on the location/room calendars
- All built-in payment processors now handle sales taxes appropriately (#59)
- On refunds, changes to fees are now allocated across invoiceitems, ensuring that the associated revenue items remain correct (#57)
- Fixed CSRF verification error with Ajax sign-in on the student info page (#58)
- Invoice emails now contain appropriate page protocol in invoice URLs so that they will show up in notification emails as clickable links (#56)
- numerous small bug fixes and improvements

12.23.1 Upgrade notes:

Version 0.4.0 is a fully backwards compatible release. However, a number of small template changes and improvements have been incorporated on admin and registration templates, so if you are overriding registration templates, you may wish to check that the defaults have not changed.

12.24 0.3.0 - September 1, 2017

- **New:** Added discount categories, with the lowest-priced discount *per category* automatically applied as a method of permitting multiple simultaneous discounts. Categories are orderable so that discounts are always applied in the same order.
- Moved discounted student pricing from the core app to the discounts app. Core app PricingTiers now contain only onlinePrice, doorPrice, and dropInPrice values.
- Temporary Registration objects now expire and are deleted (along with expired session data) by a Huey cron task (if enabled). By default, Temporary Registrations expire 15 minutes after the registration process begins, with time extended as they proceed through the process.
- When beginning the registration process, the system looks at both completed registrations and in-process registrations (unexpired TemporaryRegistration instances) to determine if registration is allowed. This prevents accidental overregistration.
- Fixed issue with the `settings.py provided in the default_setup.zip file that prevented adding or modidying CMS plugin instances.

- Added separate `setup_paypal`, `setup_stripe`, and `setup_permissions` commands that can be used separately to handle setup of Paypal, Stripe, and group permissions without running the entire `setupschool` management command script.

12.24.1 Upgrade notes:

Because student pricing in the core app has been deleted, individuals upgrading to version 0.3.0 who wish to maintain separate pricing for students will need to create discounts in the discounts app to do so. All student pricing information will be deleted when the upgrade takes place. No existing registrations will be affected by this change.

Upon upgrade, all existing TemporaryRegistration objects will be marked as expired. If any customers are in the process of registering at the time of upgrade, they will be asked to begin the registration process again.

12.25 0.2.4 - August 25, 2017

- **New:** Added a "ban list" app that allows schools to enter a list of names and emails that are not permitted to register, with the option to add photographs and notes.

12.26 0.2.3 - August 23, 2017

- **New:** Added the ability to automatically generate "generic" expense items daily/weekly/monthly using the same rule-based logic as automatic generation of expenses for locations and staff members.
- Minor admin cleanup in the Financial app.

12.27 0.2.2 - August 21, 2017

- Removed hard-coded references to "Lead" and "Follow" roles in certain stats graphs so that they show stats based on all configurable roles.
- Added default ordering to EventOccurrence and other fields to avoid unexpected ordering issues.
- Added the ability to add Events to the registration using a "pre_temporary_registration" signal handler based on information collected by the student information form.

12.28 0.2.1 - August 16, 2017

- Fixed bug in which adding voucher/discount restrictions caused the changelist admin to fail.

12.29 0.2.0 - August 15, 2017

- **New:** Improved automatic generation of expenses for venues and event staff, including flexible options for expenses to be generated per day, per week, or per month for simplified accounting.
- **New:** Locations can now have multiple Rooms, with specified capacities for each.
- **New:** Time-based (early bird) discounts for registration based on the number of days prior to class beginning.

- Series and Event categories can now be flagged for easier separate display on the main Registration page, with easier override of display format for specific categories.
- Through the Customer admin, it is now possible to email specific customers using the standard email form.
- In the prerequisites app, it is now possible to lookup specific customers to determine whether they meet class requirements.
- New options for customer prerequisite items, such as allowing partial simultaneous overlap
- Numerous admin action improvements for easier bulk operations.
- Default installation now uses Huey's SQLite integration for easier setup of development instances
- Improvements to "Add Series" view, now using moment.js and datepair.js
- CMS toolbar menu ordering and display bug fixes
- Numerous admin UI improvements
- Many small bug fixes

12.30 0.1.2

- Fixed bug where default navigation menu would not expand on mobile browsers
- Added automatic creation of a Logout link to the default navigation using the setupschool script.

12.31 0.1.1

- Fixed bug where email context was not being rendered for HTML emails
- Fixed bug where i18n template tag was not loaded for successful form submission template.

12.32 0.1.0

- Initial public release
- Added Stripe Checkout integration
- Updated and simplified payment processor integration
- Added initial tests of basic functionality

CHAPTER 13

Indices and tables

- genindex
- modindex
- search