saleor Documentation

Release 2018.01

Mirumee Software

Feb 21, 2019

Contents

An open source storefront written in Python.

Contents:

CHAPTER 1

Getting Started

1.1 Installation for macOS

1.1.1 Prerequisites

Before you are ready to run Saleor you will need additional software installed on your computer.

Node.js

Version 10 or later is required. Download the macOS installer from the Node.js downloads page.

PostgreSQL

Saleor needs PostgreSQL version 9.4 or above to work. Get the macOS installer from the PostgreSQL download page.

Command Line Tools for Xcode

Download and install the latest version of "Command Line Tools (macOS 10.x) for Xcode 9.x" from the Downloads for Apple Developers page.

Then run:

```
$ xcode-select --install
```

Homebrew

Run the following command:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Python 3

Use Homebrew to install the latest version of Python 3:

```
$ brew install python3
```

Git

Use Homebrew to install Git:

```
$ brew install git
```

Gtk+

Use Homebrew to install the graphical libraries necessary for PDF creation:

```
$ brew install cairo pango gdk-pixbuf libffi
```

1.1.2 Installation

1. Clone the repository (or use your own fork):

   ```
   $ git clone https://github.com/mirumee/saleor.git
   ```

2. Enter the directory:

   ```
   $ cd saleor/
   ```

3. Install all dependencies:

   We strongly recommend creating a virtual environment before installing any Python packages.

   ```
   $ pip install -r requirements.txt
   ```

4. Set `SECRET_KEY` environment variable.

   We try to provide usable default values for all of the settings. We've decided not to provide a default for `SECRET_KEY` as we fear someone would inevitably ship a project with the default value left in code.

   ```
   $ export SECRET_KEY='<mysecretkey>'
   ```

   Warning: Secret key should be a unique string only your team knows. Running code with a known `SECRET_KEY` defeats many of Django's security protections, and can lead to privilege escalation and remote code execution vulnerabilities. Consult Django's documentation for details.

5. Create a PostgreSQL user:

   Unless configured otherwise the store will use `saleor` as both username and password. Remember to give your user the `SUPERUSER` privilege so it can create databases and database extensions.

   ```
   $ createuser --superuser --pwprompt saleor
   ```

Enter `saleor` when prompted for password.

6. Create a PostgreSQL database:

   Unless configured otherwise the store will use `saleor` as the database name.

```
$ createdb saleor
```

7. Prepare the database:

```
$ python manage.py migrate
```

> **Warning:** This command will need to be able to create database extensions. If you get an error related to the `CREATE EXTENSION` command please review the notes from the user creation step.

8. Install front-end dependencies:

```
$ npm install
```

> **Note:** If this step fails go back and make sure you're using new enough version of Node.js.

9. Prepare front-end assets:

```
$ npm run build-assets
```

10. Compile e-mails:

```
$ npm run build-emails
```

11. Start the development server:

```
$ python manage.py runserver
```

1.2 Installation for Windows

This guide assumes a 64-bit installation of Windows.

1.2.1 Prerequisites

Before you are ready to run Saleor you will need additional software installed on your computer.

Python

Download the latest **3.7** Windows installer from the Python download page and follow the instructions.

Make sure that "**Add Python 3.7 to PATH**" is checked.

Node.js

Version 10 or later is required. Download the Windows installer from the Node.js downloads page.

Make sure that "**Add to PATH**" is selected.

PostgreSQL

Saleor needs PostgreSQL version 9.4 or above to work. Get the Windows installer from the project's download page.

Make sure you keep track of the password you set for the administration account during installation.

GTK+

Download the 64-bit Windows installer.

Make sure that "**Set up PATH environment variable to include GTK+**" is selected.

Compilers

Please download and install the latest version of the Build Tools for Visual Studio.

1.2.2 Installation

All commands need to be performed in either PowerShell or a Command Shell.

1. Clone the repository (replace the URL with your own fork where needed):

```
git clone https://github.com/mirumee/saleor.git
```

2. Enter the directory:

```
cd saleor/
```

3. Install all dependencies:

   We strongly recommend creating a virtual environment before installing any Python packages.

```
python -m pip install -r requirements.txt
```

4. Set `SECRET_KEY` environment variable.

   We try to provide usable default values for all of the settings. We've decided not to provide a default for `SECRET_KEY` as we fear someone would inevitably ship a project with the default value left in code.

```
$env:SECRET_KEY = "<mysecretkey>"
```

> **Warning:** Secret key should be a unique string only your team knows. Running code with a known `SECRET_KEY` defeats many of Django's security protections, and can lead to privilege escalation and remote code execution vulnerabilities. Consult Django's documentation for details.

5. Create a PostgreSQL user:

   Use the **pgAdmin** tool that came with your PostgreSQL installation to create a database user for your store.

   Unless configured otherwise the store will use `saleor` as both username and password. Remember to give your user the `SUPERUSER` privilege so it can create databases and database extensions.

6. Create a PostgreSQL database

   See PostgreSQL's createdb command for details.

   Note: Database name is extracted from the `DATABASE_URL` environment variable. If absent it defaults to `saleor`.

7. Prepare the database:

```
python manage.py migrate
```

   Warning: This command will need to be able to create a database and some database extensions. If you get an error related to these make sure you've properly assigned your user `SUPERUSER` privileges.

8. Install front-end dependencies:

```
npm install
```

   Note: If this step fails go back and make sure you're using new enough version of Node.js.

9. Prepare front-end assets:

```
npm run build-assets
```

10. Compile e-mails:

```
npm run build-emails
```

11. Start the development server:

```
python manage.py runserver
```

1.3 Installation for Linux

Note: If you prefer using containers or have problems with configuring PostgreSQL, Redis and Elasticsearch, try *Using Docker for Development* instructions.

1.3.1 Prerequisites

Before you are ready to run Saleor you will need additional software installed on your computer.

Python 3

Saleor requires Python 3.5 or later. A compatible version comes preinstalled with most current Linux systems. If that is not the case consult your distribution for instructions on how to install Python 3.6 or 3.7.

Node.js

Version 10 or later is required. See the installation instructions.

PostgreSQL

Saleor needs PostgreSQL version 9.4 or above to work. Use the PostgreSQL download page to get instructions for your distribution.

Gtk+

Some features like PDF creation require that additional system libraries are present.

Debian / Ubuntu

Debian 9.0 Stretch or newer, Ubuntu 16.04 Xenial or newer:

```
$ sudo apt-get install build-essential python3-dev python3-pip python3-cffi libcairo2␣
↪libpango-1.0-0 libpangocairo-1.0-0 libgdk-pixbuf2.0-0 libffi-dev shared-mime-info
```

Fedora

```
$ sudo yum install redhat-rpm-config python-devel python-pip python-cffi libffi-devel␣
↪cairo pango gdk-pixbuf2
```

Archlinux

```
$ sudo pacman -S python-pip cairo pango gdk-pixbuf2 libffi pkg-config
```

Gentoo

```
$ emerge pip cairo pango gdk-pixbuf cffi
```

1.3.2 Installation

1. Clone the repository (or use your own fork):

   ```
   $ git clone https://github.com/mirumee/saleor.git
   ```

2. Enter the directory:

```
$ cd saleor/
```

3. Install all dependencies:

   We strongly recommend creating a virtual environment before installing any Python packages.

```
$ pip install -r requirements.txt
```

4. Set `SECRET_KEY` environment variable.

   We try to provide usable default values for all of the settings. We've decided not to provide a default for `SECRET_KEY` as we fear someone would inevitably ship a project with the default value left in code.

```
$ export SECRET_KEY='<mysecretkey>'
```

> **Warning:** Secret key should be a unique string only your team knows. Running code with a known `SECRET_KEY` defeats many of Django's security protections, and can lead to privilege escalation and remote code execution vulnerabilities. Consult Django's documentation for details.

5. Create a PostgreSQL user:

   See PostgreSQL's createuser command for details.

Note: You need to create the user to use within your project. Username and password are extracted from the `DATABASE_URL` environmental variable. If absent they both default to `saleor`.

> **Warning:** While creating the database Django will need to create some PostgreSQL extensions if not already present in the database. This requires a superuser privilege.
>
> For local development you can grant your database user the `SUPERUSER` privilege. For publicly available systems we recommend using a separate privileged user to perform database migrations.

6. Create a PostgreSQL database

   See PostgreSQL's createdb command for details.

Note: Database name is extracted from the `DATABASE_URL` environment variable. If absent it defaults to `saleor`.

---

7. Prepare the database:

```
$ python manage.py migrate
```

> **Warning:** This command will need to be able to create database extensions. If you get an error related to the `CREATE EXTENSION` command please review the notes from the user creation step.

8. Install front-end dependencies:

```
$ npm install
```

Note: If this step fails go back and make sure you're using new enough version of Node.js.

9. Prepare front-end assets:

```
$ npm run build-assets
```

10. Compile e-mails:

```
$ npm run build-emails
```

11. Start the development server:

```
$ python manage.py runserver
```

1.4 Configuration

We are fans of the 12factor approach and portable code so you can configure most of Saleor using just environment variables.

1.4.1 Payments Gateways

CHECKOUT_PAYMENT_GATEWAYS This contains the list of enabled payment gateways, with the payment friendly name to show to the user on the payment selection form.

For example, to add braintree to the enabled gateways, you can do the following:

```
CHECKOUT_PAYMENT_GATEWAYS = {
    DUMMY: pgettext_lazy('Payment method name', 'Dummy gateway'),
    BRAINTREE: pgettext_lazy('Payment method name', 'Brain tree')
}
```

The supported payment providers are:

- `DUMMY` (for tests purposes only!);
- `BRAINTREE`;
- `RAZORPAY`;
- `STRIPE`.

PAYMENT_GATEWAYS For information on how to configure payment gateways (API keys, miscellaneous information, ...), see *the list of supported payment gateway and their associated environment variables*.

1.4.2 Environment variables

ALLOWED_HOSTS Controls Django's allowed hosts setting. Defaults to `localhost`.

Separate multiple values with comma.

CACHE_URL or REDIS_URL The URL of a cache database. Defaults to local process memory.

Redis is recommended. Heroku's Redis will export this setting automatically.

Example: `redis://redis.example.com:6379/0`

> **Warning:** If you plan to use more than one WSGI process (or run more than one server/container) you need to use a shared cache server. Otherwise each process will have its own version of each user's session which will result in people being logged out and losing their shopping carts.

DATABASE_URL Defaults to a local PostgreSQL instance. See *Using Docker for Development* for how to get a local database running inside a Docker container.

Most Heroku databases will export this setting automatically.

Example: `postgres://user:password@psql.example.com/database`

DEBUG Controls Django's debug mode. Defaults to `True`.

DEFAULT_FROM_EMAIL Default email address to use for outgoing mail.

EMAIL_URL The URL of the email gateway. Defaults to printing everything to the console.

Example: `smtp://user:password@smtp.example.com:465/?ssl=True`

INTERNAL_IPS Controls Django's internal IPs setting. Defaults to `127.0.0.1`.

Separate multiple values with comma.

SECRET_KEY Controls Django's secret key setting.

SENTRY_DSN Sentry's Data Source Name. Disabled by default, allows to enable integration with Sentry (see *Error tracking with Sentry* for details).

MAX_CART_LINE_QUANTITY Controls maximum number of items in one cart line. Defaults to `50`.

STATIC_URL Controls production assets' mount path. Defaults to `/static/`.

VATLAYER_ACCESS_KEY Access key to vatlayer API. Enables VAT support within European Union.

To update the tax rates run the following command at least once per day:

```
$ python manage.py get_vat_rates
```

DEFAULT_CURRENCY Controls all prices entered and stored in the store as this single default currency (for more information, see *Handling Money Amounts*).

DEFAULT_COUNTRY Sets the default country for the store. It controls the default VAT to be shown if required, the default shipping country, etc.

CREATE_IMAGES_ON_DEMAND Whether or not to create new images on-the-fly (`True` by default). Set this to `False` for speedy performance, which is recommended for production. Every image should come with a pre-warm to ensure they're created and available at the appropriate URL.

1.5 Creating an Administrator Account

Saleor is a Django application so you can create your master account using the following command:

```
$ python manage.py createsuperuser
```

Follow prompts to provide your email address and password.

You can then start your local server and visit `http://localhost:8000/dashboard/` to log into the management interface.

Please note that creating users in this way gives them "superuser" status which means they have all privileges no matter which permissions they have granted.

1.6 Debug tools

We have built in support for some of the debug tools.

1.6.1 Django debug toolbar

Django Debug Toolbar is turned on if Django is running in debug mode.

1.6.2 Silk

Silk's presence can be controled via environmental variable

`ENABLE_SILK` Controls django-silk. Defaults to `False`

1. Set environment variable.

```
$ export ENABLE_SILK='True'
```

2. Restart server

1.7 Example Data

If you'd like some data to test your new storefront you can populate the database with example products and orders:

```
$ python manage.py populatedb --createsuperuser
```

The `--createsuperuser` argument creates an admin account for `admin@example.com` with the password set to `admin`.

CHAPTER 2

Customizing Saleor

2.1 Using Docker for Development

Using Docker to build software allows you to run and test code without having to worry about external dependencies such as cache servers and databases.

> **Warning:** The following setup is only meant for local development. See *Docker* for production use of Docker.

2.1.1 Local Prerequisites

You will need to install Docker and docker-compose before performing the following steps.

> **Note:** Our configuration uses docker-compose.override.yml that exposes Saleor, PostgreSQL and Redis ports and runs Saleor via `python manage.py runserver` for local development. If you do not wish to use any overrides then you can tell compose to only use *docker-compose.yml* configuration using *-f*, like so *docker-compose -f docker-compose.yml up*.

2.1.2 Using local assets

By default we do not mount assets for development in the Docker, reason being is those are built in the Docker at build-time and aren't present in the cloned repository, so what was built on the Docker would be overshadowed by empty directories from the host.

However, we do know that there might be a case that you wish to mount them and see your changes reflected in the container, thus before proceeding you need to modify *docker-compose.override.yml*.

In order for Docker to use your assets from the host, you need to remove `/app/saleor/static/assets` volume and add `./webpack-bundle.json:/app/webpack-bundle.json` volume.

Additionally if you wish to have the compiled templated emails mounted then you need to also remove `/app/templates/templated_email/compiled` volume from web and celery services.

2.1.3 Usage

1. Build the containers using `docker-compose`

```
$ docker-compose build
```

2. Prepare the database

```
$ docker-compose run --rm web python3 manage.py migrate
$ docker-compose run --rm web python3 manage.py collectstatic
$ docker-compose run --rm web python3 manage.py populatedb --createsuperuser
```

   The `--createsuperuser` argument creates an admin account for `admin@example.com` with the password set to `admin`.

3. Run the containers

```
$ docker-compose up
```

By default, the application is started in debug mode and is configured to listen on port `8000`.

2.2 Customizing Templates

Default storefront templates are based on Bootstrap 4.

You can find the files under `/templates/`.

2.3 Customizing Emails

2.3.1 Sending Emails

Emails are sent with Django-Templated-Email.

2.3.2 Customizing Email Templates

Templates for emails live in `templates/templated_email`. App-specific directories contain `*.email` files that define each specific message type.

The `source` directory contains `*.mjml` files. Those MJML files are compiled to `*.html` and put into `compiled` directory.

Emails defined in `*.email` files include their HTML versions by referencing the compiled `*.html` files.

2.3.3 Compiling MJML

Source emails use MJML and must be compiled to HTML before use.

To compile emails run:

```
$ npm run build-emails
```

2.4 Customizing CSS and JavaScript

All static assets live in subdirectories of `/saleor/static/`.

Stylesheets are written in Sass and rely on postcss and autoprefixer for cross-browser compatibility.

JavaScript code is written according to the ES2015 standard and relies on Babel for transpilation and browser-specific polyfills.

Everything is compiled together using webpack module bundler.

The resulting files are written to `/saleor/static/assets/` and should not be edited manually.

During development it's very convenient to have webpack automatically track changes made locally. This will also enable *source maps* that are extremely helpful when debugging.

To run webpack in *watch* mode run:

```
$ npm start
```

Warning: Files produced this way are not ready for production use. To prepare static assets for deployment run:

```
$ npm run build-assets --production
```

2.5 Working with Python Code

2.5.1 Managing Dependencies

To guarantee repeatable installations, all project dependencies are managed using Pipenv. Project's direct dependencies are listed in `Pipfile` and running `pipenv lock` would generate `Pipfile.lock` that has all versions pinned.

For users who are not using Pipenv, `requirements.txt` and `requirements_dev.txt` are also provided. They should be updated by `pipenv lock -r > requirements.txt` and `pipenv lock -r --dev > requirements_dev.txt`, if any dependencies are changed in `Pipfile`.

We recommend you use this workflow and keep `Pipfile` as well as `Pipfile.lock` under version control to make sure all computers and environments run exactly the same code.

2.6 Internationalization

2.6.1 Pulling Translations From Transifex

First make sure you have the Transifex command-line client installed by `python -m pip install transifex-client` or `python -m pip install -r requirements_dev.txt`. For Pipenv users, it can also be installed by `pipenv install --dev`.

Then use the `pull` command to pull translations:

```
$ tx pull
```

Note: To create locale directories for newly created translations you will need to call `tx pull` with the `--all` flag.

2.6.2 Compiling Message Catalogs

This is required for Django to see the translations.

```
$ python manage.py compilemessages
```

Note: On Windows, you will need to install GNU's gettext command. To do that you need to install gettext-iconv. During the installation, make sure to check "Add to PATH".

Don't forget to restart your terminal or software after the installation to take the changes into effect.

2.6.3 Extracting Messages to Translate

This will update the English language files with messages that appear in your code.

For the backend code and templates:

```
$ python manage.py makemessages -l en --extension=email,html,mjml,py,txt --ignore=
↪"templates/templated_email/compiled/*"
```

For JavaScript code:

```
$ python manage.py makemessages -l en -d djangojs --ignore="_build/*" --ignore="node_
↪modules/*" --ignore="saleor/static/assets/*"
```

2.7 Running Tests

Before you make any permanent changes in the code you should make sure they do not break existing functionality.

The project currently contains very little front-end code so the test suite only covers backend code.

Before running tests, development packages should be installed by `python -m pip install -r requirements_dev.txt`. For Pipenv users, `pipenv install --dev` should do the same.

To run backend tests use pytest:

```
$ py.test
```

You can also test against all supported versions of Django and Python. This is usually only required if you want to contribute your changes back to Saleor. To do so you can use Tox:

```
$ tox
```

2.8 Continuous Integration

The storefront ships with a working CircleCI configuration file. To use it log into your CircleCI account and enable your repository.

2.9 Running with PyPy 3.5

Saleor works well with PyPy 3.5 and using it is an option when additional performance is required.

The default PostgreSQL driver is not compatible with PyPy so you will need to replace it with a `cffi`-based one.

Please consult the installation instructions provided by psycopg2cffi.

CHAPTER 3

Supported Payment Gateways

You will find below the list of payment gateways supported by Saleor and their configuration guide.

3.1 Braintree (supports PayPal and Credit Cards)

This gateway implements payments using Braintree.

Environment variable	Description
`BRAINTREE_SANDBOX_MODE`	Whether to use a sandbox environment for testing, `True` (default) or `False`
`BRAINTREE_MERCHANT_ID`	Merchant ID assigned by Braintree
`BRAINTREE_PUBLIC_KEY`	Public key assigned by Braintree
`BRAINTREE_PRIVATE_KEY`	Private key assigned by Braintree

Note: This backend does not support fraud detection.

Warning: Make sure that Braintree's currency is the same as your shop's, otherwise, customers will be charged the wrong amount.

3.2 Razorpay (supports only the paisa currency)

This gateway implements payments using Razorpay.

First of all, to create your API credentials, you need to go in your Razorpay account settings, then in the API Keys section.

Environment variable	Description
RAZORPAY_PUBLIC_KEY	Your Razorpay **key id**
RAZORPAY_SECRET_KEY	Your Razorpay **secret key id**
RAZORPAY_PREFILL	Pre-fill the email and customer's full name if set to `True` (default)
RAZORPAY_STORE_NAME	Your store name
RAZORPAY_STORE_IMAGE	An absolute or relative link to your store logo

Warning: Only the paisa (INR) currency is supported by Razorpay as of now.

3.3 Stripe (supports Credit Cards)

This gateway implements payments using Stripe.

Environment variable	Description
STRIPE_PUBLIC_KEY	Your Stripe public key (test or live)
STRIPE_SECRET_KEY	Your Stripe secret key (test or live)
STRIPE_STORE_NAME	Your store name to show in the checkout form
STRIPE_STORE_IMAGE	An absolute or relative link of your store logo to show in the checkout form
STRIPE_PREFILL	Prefill the email adddress in the checkout form if set to `True` (default)
STRIPE_REMEMBER_ME	Add "Remember Me" for future purchases in the checkout form if set to `True` (default)
STRIPE_LOCALE	Specify `auto` to display checkout form in the user's preferred language (default)
STRIPE_ENABLE_BILLING_ADDRESS	Collect the user's billing address in the checkout form if set to `True`. The default is `False`
STRIPE_ENABLE_SHIPPING_ADDRESS	Collect the user's shipping address in the checkout form if set to `True`. The default is `False`

The default configuration only uses the *dummy* backend (see *how to enable/disable payment gateways*). It's meant to allow developers to easily simulate different payment results.

For an how-to guide on adding new payments into your Saleor project please check *Payments*.

Note: All payment backends default to using sandbox mode. This is very useful for development but make sure you use production mode when deploying to a production server.

CHAPTER 4

Contributing Guides

Please use GitHub issues and pull requests to report problems and discuss new features.

4.1 EditorConfig

EditorConfig is a standard configuration file that aims to ensure consistent style across multiple programming environments.

Saleor's repository contains an .editorconfig file that describes our formatting requirements.

Most editors and IDEs support this file either directly or via plugins. Consult the list of supported editors and IDEs for instructions.

Please make sure that your programming environment respects the contents of this file and you will automatically get correct indentation, encoding, and line endings.

4.2 Coding Style

4.2.1 Python

Always follow PEP 8 but keep in mind that consistency is important.

String Literals

Prefer single quotes to double quotes unless the string itself contains single quotes that would need to be needlessly escaped.

Wrapping Code

When wrapping code follow the "hanging grid" format:

```
some_dict = {
    'one': 1,
    'two': 2,
    'three': 3}
```

```
some_list = [
    'foo', 'bar', 'baz']
```

Python is an indent-based language and we believe that beautiful, readable code is more important than saving a single line of `git diff`. Please avoid dangling parentheses, brackets, square brackets or hanging commas even if the Django project seems to encourage this programming style:

```
this_is_wrong = {
   'one': 1,
   'two': 2,
   'three': 3,
}
```

Please break multi-line code immediately after the parenthesis and avoid relying on a precise number of spaces for alignment:

```
also_wrong('this is hard',
           'to maintain',
           'as it often needs to be realigned')
```

Linters

Use isort to maintain consistent imports.

Use pylint with the `pylint-django` plugin to catch errors in your code.

Use pycodestyle to make sure your code adheres to PEP 8.

Use pydocstyle to check that your docstrings are properly formatted.

`python -m pip install -r requirements_dev.txt` can install these tools. For Pipenv users, they can also be installed by `pipenv install --dev`.

4.3 Naming Conventions

To keep a consistent code structure we follow some rules when naming files.

4.3.1 Python Modules

Try to have the name reflect the function of the file. If possible avoid generic file names such as `utils.py`.

4.3.2 Django Templates

Use underscore as a word separator.

4.3.3 Static Files

Use dashes to separate words as they end up as part of the URL.

CHAPTER 5

Architecture

5.1 Handling Money Amounts

Saleor uses the Prices and django-prices libraries to store, calculate and display amounts of money, prices and ranges of those and django-prices-vatlayer to handle VAT tax rates in European Union (optionally).

5.1.1 Default currency

All prices are entered and stored in a single default currency controlled by the *DEFAULT_CURRENCY* settings key. Saleor can display prices in a user's local currency (see *Open Exchange Rates*) but all purchases are charged in the default currency.

> **Warning:** The currency is not stored in the database. Changing the default currency in a production environment will not recalculate any existing orders. All numbers will remain the same and will be incorrectly displayed as the new currency.

5.1.2 Money and TaxedMoney

In Saleor's codebase, money amounts exist either as *Money* or *TaxedMoney* instances.

Money is a type representing amount of money in specific currency: 100 USD is represented by *Money(100, 'USD')*. This type doesn't hold any additional information useful for commerce but, unlike *Decimal*, it implements safeguards and checks for calculations and comparisons of monetary values.

Money amounts are stored on model using *MoneyField* that provides its own safechecks on currency and precision of stored amount.

If you ever need to get to the *Decimal* of your *Money* object, you'll find it on the *amount* property.

Products and shipping methods prices are stored using *MoneyField*. All prices displayed in dashboard, excluding orders, are as they have been entered in the forms. You can decide if those prices are treated as gross or net in dashboard `Taxes` tab.

Prices displayed in orders are gross or net depending on setting how prices are displayed for customers, both in storefront and dashboard. This way staff users will always see the same state of an order as the customer.

5.1.3 TaxedMoneyRange

Sometimes a product may be available under more than single price due to its variants defining custom prices different from the base price.

For such situations *Product* defines additional *get_price_range* method that return *TaxedMoneyRange* object defining minimum and maximum prices on its *start* and *stop* attributes. This object is then used by the UI to differentiate between displaying price as "10 USD" or "from 10 USD" in case of products where prices differ between variants.

5.2 Product Structure

Before filling your shop with products we need to introduce 3 product concepts - *product types*, *products*, *product variants*.

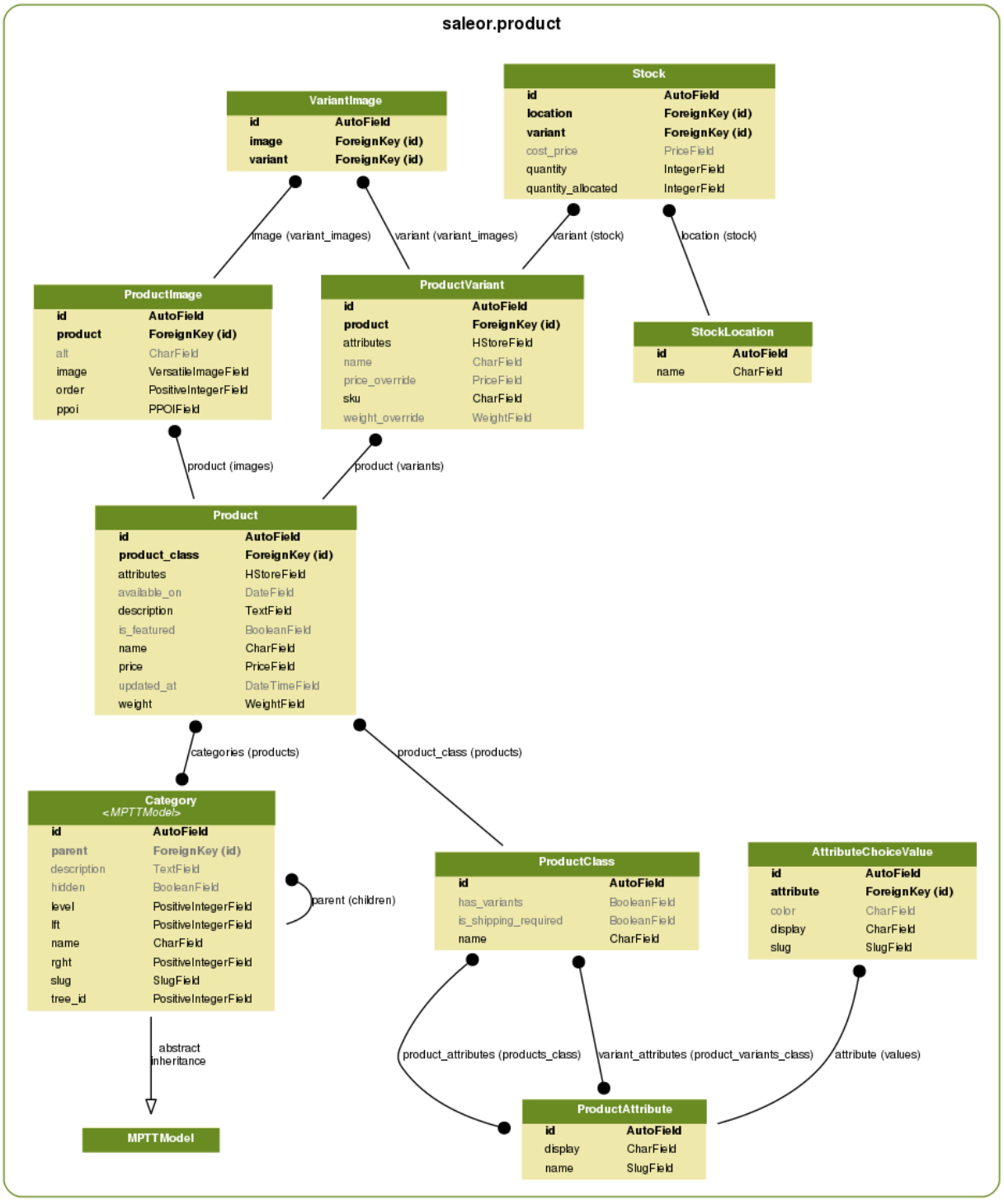
5.2.1 Overview

Consider a book store. One of your *products* is a book titled "Introduction to Saleor".

The book is available in hard and soft cover, so there would be 2 *product variants*.

Type of cover is the only attribute which creates separate variants in our store, so we use *product type* named "Book" with variants enabled and a "Cover type" *variant attribute*.

5.2.2 Class Diagram



5.2.3 Product Variants

Variants are the most important objects in your shop. All cart and stock operations use variants. Even if a product doesn't have multiple variants, the store will create one under the hood.

5.2.4 Products

Describes common details of a few *product variants*. When the shop displays the category view, items on the list are distinct *products*. If the variant has no overridden property (example: price), the default value is taken from the *product*.

- **`publication_date`** Until this date the product is not listed in storefront and is unavailable for users.
- **`is_featured`** Featured products are displayed on the front page.

5.2.5 Product Types

Think about types as templates for your products. Multiple *products* can use the same product type.

- **`product_attributes`** Attributes shared among all *product variants*. Example: publisher; all book variants are published by same company.
- **`variant_attributes`** It's what distinguishes different *variants*. Example: cover type; your book can be in hard or soft cover.
- **`is_shipping_required`** Indicates whether purchases need to be delivered. Example: digital products; you won't use DHL to ship an MP3 file.
- **`has_variants`** Turn this off if your *product* does not have multiple variants or if you want to create separate *products* for every one of them.

  This option mainly simplifies product management in the dashboard. There is always at least one *variant* created under the hood.

Warning: Changing a product type affects all products of this type.

Warning: You can't remove a product type if there are products of that type.

5.2.6 Attributes

Attributes can help you better describe your products. Also, the can be used to filter items in category views.

The attribute values display in the storefront in the order that they are listed in the list in attribute details view. You can reorder them by handling an icon on the left to the values and dragging them to another position.

There are 2 types of *attributes* - choice type and text type. If you don't provide choice values, then attribute is text type.

Examples

- *Choice type*: Colors of a t-shirt (for example 'Red', 'Green', 'Blue')
- *Text type*: Number of pages in a book

Example: Coffee

Your shop sells Coffee from around the world. Customer can order 1kg, 500g and 250g packages. Orders are shipped by couriers.

Table 1: Attributes

Attribute	Values
Country of origin	• Brazil • Vietnam • Colombia • Indonesia
Package size	• 1kg • 500g • 250g

Table 2: Product type

Name	Product attributes	Variants?	Variant attributes	Shipping?
Coffee	• Country of origin	Yes	• Package size	Yes

Table 3: Product

Product type	Name	Country of origin	Description
Coffee	Best Java Coffee	Indonesia	Best coffee found on Java island!

Table 4: Variants

SKU	Package size	Price override
J001	1kg	$20
J002	500g	$12
J003	250g	$7

Example: Online game items

You have great selection of online games items. Each item is unique, important details are included in description. Bought items are shipped directly to buyer account.

Table 5: Attributes

Attribute	Values
Game	• Kings Online • War MMO • Target Shooter
Max attack	—

Table 6: Product type

Name	Product attributes	Variants?	Variant attributes	Shipping?
Game item	• Game • Max attack	No	—	No

Table 7: Products

Product type	Name	Price	Game	Max attack	Description
Game item	Magic Fire Sword	$199	Kings Online	8000	Unique sword for any fighter. Set your enemies on fire!
Game item	Rapid Pistol	$2500	Target Shooter	250	Fastest pistol in the whole game.

5.3 Thumbnails

Saleor uses VersatileImageField replacement for Django's ImageField. For performance reasons, in non-debug mode thumbnails are pregenerated by the worker's task, fired after saving the instance. Accessing missing image will result in 404 error.

In debug mode thumbnails are generated on demand.

5.3.1 Generating Products Thumbnails Manually

Create missing thumbnails for all ProductImage instances.

```
$ python manage.py create_thumbnails
```

5.3.2 Deleting Images

Image renditions are not deleted automatically with the Image instance, so is the main image. More on deleting images can be found in VersatileImageField documentation

5.4 Stock Management

Each product variant has a stock keeping unit (SKU).

Each variant holds information about *quantity* at hand, quantity *allocated* for already placed orders and quantity *available*.

Example: There are five boxes of shoes. Three of them have already been sold to customers but were not yet dispatched for shipment. The stock records **quantity** is **5**, **quantity allocated** is **3** and **quantity available** is **2**.

Each variant also has a *cost price* (the price that your store had to pay to obtain it).

5.4.1 Product Availability

A variant is *in stock* if it has unallocated quantity.

The highest quantity that can be ordered is the available quantity in product variant.

5.4.2 Allocating Stock for New Orders

Once an order is placed, quantity needed to fulfil each order line is immediately marked as *allocated*.

Example: A customer places an order for another box of shoes. The stock records **quantity** is **5**, **quantity allocated** is now **4** and **quantity available** becomes **1**.

5.4.3 Decreasing Stock After Shipment

Once order lines are marked as shipped, each corresponding stock record will have both its quantity at hand and quantity allocated decreased by the number of items shipped.

Example: Two boxes of shoes from warehouse A are shipped to a customer. The stock records **quantity** is now **3**, **quantity allocated** becomes **2** and **quantity available** stays at **1**.

5.5 Order Management

Orders are created after customers complete the checkout process. The *Order* object itself contains only general information about the customer's order.

5.5.1 Fulfillment

The fulfillment represents a group of shipped items with corresponding tracking number. Fulfillments are created by a shop operator to mark selected products in an order as fulfilled.

There are two possible fulfillment statuses:

- **`NEW`** The default status of newly created fulfillments.
- **`CANCELED`** The fulfillment canceled by a shop operator. This action is irreversible.

5.5.2 Order statuses

There are four possible order statuses, based on statuses of its fulfillments:

- **`UNFULFILLED`** There are no fulfillments related to an order or each one is canceled. An action by a shop operator is required to continue order processing.

- **PARTIALLY FULFILLED** There are some fulfillments with `FULFILLED` status related to an order. An action by a shop operator is required to continue order processing.
- **FULFILLED** Each order line is fulfilled in existing fulfillments. Order doesn't require further actions by a shop operator.
- **CANCELED** Order has been canceled. Every fulfillment (if there is any) has `CANCELED` status. Order doesn't require further actions by a shop operator.

There is also `DRAFT` status, used for orders newly created from dashboard and not yet published.

5.6 Events

Note: Events are autogenerated and will be triggered when certain actions are completed, such us creating the order, cancelling fulfillment or completing a payment.

5.6.1 Order events

Code	GraphQL API value	Description
`placed`	`PLACED`	An order was placed by the customer.
`draft_placed`	`PLACED_FROM_DRAFT`	An order was created from draft by the staff user.
`oversold_items`	`OVERSOLD_ITEMS`	An order was created from draft, but some line items were oversold.
`canceled`	`CANCELED`	The order was cancelled.
`order_paid`	`ORDER_PAID`	The order was fully paid by the customer.
`marked_as_paid`	`MARKED_AS_PAID`	The order was manually marked as fully paid by the staff user.
`updated`	`UPDATED`	The order was updated.
`email_sent`	`EMAIL_SENT`	An email was sent to the customer.
`captured`	`CAPTURED`	The payment was captured.
`refunded`	`REFUNDED`	The payment was refunded.
`voided`	`VOIDED`	The payment was voided.
`fulfillment_canceled`	`FULFILLMENT_CANCELED`	Fulfillment for one or more of the items was canceled.
`restocked_items`	`RESTOCKED_ITEMS`	One or more of the order's items have been resocked
`fulfilled_items`	`FULFILLED_ITEMS`	One or more of the order's items have been fulfilled.
`note_added`	`NOTE_ADDED`	A note was added to the order by the staff.
`other`	`OTHER`	Status used during reimporting of the legacy events.

5.7 Internationalization

By default language and locale are determined based on the list of preferences supplied by a web browser. GeoIP is used to determine the visitor's country and their local currency.

Note: Saleor uses Transifex to coordinate translations. If you wish to help please head to the translation dashboard.

All translations are handled by the community. All translation teams are open and everyone is welcome to request a new language.

5.7.1 Translation

Saleor uses `gettext` for translation. This is an industry standard for translating software and is the most common way to translate Django applications.

Saleor's storefront and dashboard are both prepared for translation. They use separate translation domains and can be translated separately. All translations provide accurate context descriptions to make translation an easier task.

It is possible to translate database content (like product descriptions) with Saleor, more on it can be found in the *Model Translations* section.

5.7.2 Localization

Data formats

Saleor uses Babel as the interface to Unicode's CLDR library to provide accurate number and date formatting as well as proper currency designation.

Address forms

Google's address format database is used to provide locale-specific address formats and forms. It also takes care of address validation so you don't have to know how to address a package to China or whether United Arab Emirates use postal codes (they don't).

Currency conversion

Saleor can use currency exchange rate data to show price estimations in the visitor's local currency. Please consult *Open Exchange Rates* for how to set this up for Open Exchange Rates.

Phone numbers format

Saleor uses Google's libphonenumber library to ensure provided numbers are correct. You need to choose prefix and type the number separately. No matter what country has been chosen, you may enter phone number belonging to any other country format.

5.8 Model Translations

Note: At this stage, model translations are only accessible from the Python code. The backend and the storefront are prepared to handle the translated properties, but GraphQL API and UI views will be added in the future releases.

5.8.1 Overview

Model translations are available via `TranslationProxy` defined on the to-be-translated `Model`.

`TranslationProxy` gets user's language, and checks if there's a `ModelTranslation` created for that language.

If there's no relevant `ModelTranslation` available, it will return the original (therefore not translated) property. Otherwise, it will return the translated property.

5.8.2 Adding a ModelTranslation

Consider a product.

```
from django.db import models

from saleor.core.utils.translations import TranslationProxy


class Product(models.Model):
    name = models.CharField(max_length=128)
    description = models.CharField(max_length=256)
    ...

    translated = TranslationProxy()
```

The product has several properties, but we want to translate just its `name` and `description`.

We've also set a `translated` property to an instance of `TranslationProxy`.

We will use `ProductTranslation` to store our translated properties, it requires two base fields:

- **`language_code`** A language code that this translation correlates to.
- **`product`** `ForeignKey` relation to the translated object (in this case we named it *product*)

... and any other field you'd like to translate, in our example, we will use `name` and `description`.

Warning: `TranslationProxy` expects that the `related_name`, on the `ForeignKey` relation is set to `translations`

```
from django.db import models


class ProductTranslation(models.Model):
    language_code = models.CharField(max_length=10)
    product = models.ForeignKey(
        Product, related_name='translations', on_delete=models.CASCADE)
    name = models.CharField(max_length=128)
    description = models.CharField(max_length=256)

    class Meta:
        unique_together = ('product', 'language_code')
```

Note: Don't forget to set `unique_together` on the `product` and `language_code`, there should be only one translation per product per language.

Warning: `ModelTranslation` fields must always take the same arguments as the existing translatable model, eg. inconsistency in `max_length` attribute could lead to UI bugs with translation turned on.

5.8.3 Using a ModelTranslation

Given the example above, we can access translated properties via the `TranslationProxy`.

```
translated_name = product.translated.name
```

Note: Translated property will be returned if there is a `ModelTranslation` with the same `language_code` as a user's currently active language. Otherwise, the original property will be returned.

5.9 Search

There are two search mechanisms available in Saleor.

The default is to use PostgreSQL. This is a fairly versatile solution that does not require any additional resources.

A more sophisticated search backend can be enabled if an Elasticsearch server is available. Elasticsearch offers a lot of advanced features, such as boosting to tune the relevance of a query or "more like this" queries. See the official Elasticsearch website to read more about its features. Please note that enabling the Elasticsearch backend does not currently enable any additional features in Saleor.

For installation and configuration instructions see *Elasticsearch*.

5.10 Payments Architecture

5.10.1 Authorization and Capture

Some of the payment backends support pre-authorizing payments.

Authorization and capture is a two-step process.

Firstly the funds are locked on the payer's account but are not transferred to your bank.

Then depending on the gateway and the card type, you have between a few days and a month to charge the card for an amount not exceeding the authorized amount.

This is very useful when an exact price cannot be determined until after the order is prepared, or we want to capture the money as soon as we ship the order. It is also useful if your business prefers to manually screen orders for fraud attempts.

When viewing orders with pre-authorized payments Saleor will offer options to either capture or void the funds.

5.10.2 Refunds

You can issue partial or full refunds for all captured payments. When editing an order and removing items, Saleor will also offer to automatically issue a partial refund.

Saleor uses the concept of Payments and Transactions to fulfill the payment process.

5.10.3 Payment Methods

Represents transactable payment information such as credit card details, gift card information or a customer's authorization to charge their PayPal account.

All payment process related pieces of information are stored at the gateway level, we are operating on the reusable token which is a unique identifier of the customer for given gateway.

Several payment methods can be used within a single order.

Payment has 3 possible charge statuses:

Code	GraphQL API value	Description
charged	CHARGED	Funds were taken off the customer founding source, partly or completely covering the payment amount.
not-charged	NOT_CHARGED	No funds were take off the customer founding source yet.
fully-refunded	FULLY_REFUNDED	All charged funds were returned to the customer.

5.10.4 Transactions

Transaction represent attempts to transfer money between your store and your customers, within a chosen payment method.

There are 5 possible transaction kinds:

Code	GraphQL API value	Description
auth	AUTH	An amount reserved against the customer's funding source. Money does not change hands until the authorization is captured.
capture	CAPTURE	A transfer of the money that was reserved during the authorization stage.
charge	CHARGE	Authorization and capture in a single step.
void	VOID	A cancellation of a pending authorization or capture.
refund	REFUND	Full or partial return of captured funds to the customer.

5.10.5 Transaction errors

Saleor unifies error codes across all gateways.

Code	Graphql API value	Description
incorrect_number	INCORRECT_NUMBER	Incorrect card number
invalid_number	INVALID_NUMBER	Invalid card number
incorrect_cvv	INCORRECT_CVV	Incorrect CVV (or CVC)
invalid_cvv	INVALID_CVV	Invalid CVV (or CVC)
incorrect_zip	INCORRECT_ZIP	Incorrect postal code
incorrect_address	INCORRECT_ADDRESS	Incorrect address (excluding postal code)
invalid_expiry_date	INVALID_EXPIRY_DATE	Incorrect card's expiration date
expired	EXPIRED	Expired payment's method token
declined	DECLINED	Transaction was declined by the gateway
processing_error	PROCESSING_ERROR	Default error used for all cases not covered above

5.11 Shippings

Saleor uses the concept of Shipping Zones and Shipping Methods to fulfill the shipping process.

5.11.1 Shipping Zones

The countries that you ship to are known as the shipping zones. Each `ShippingZone` includes `ShippingMethods` that apply to customers whose shipping address is within the shipping zone.

Each `ShippingZone` can contain several countries inside, but the country might belong to a maximum of one `ShippingZone`.

Some examples of the `ShippingZones` could be *European Union*, *North America*, *Germany* etc.

There's also a possibility to create a default Shipping Zone which will be used for countries not covered by other zones.

5.11.2 Shipping Methods

`ShippingMethods` are the methods you'll use to get customers' orders to them. You can offer several ones within one `ShippingZone` to ensure the varieties of delivery speed and costs at the checkout.

Each `ShippmentMethod` could be one of the two types:

- **`PRICE_BASED`** Those methods can be used only when the order price is within the certain range, eg. from 0 to 50$, 50$ and up etc.
- **`WEGHT_BASED`** Same as the `PRICE_BASED`, but with the total order's weight in mind.

These methods allow you to cover most of the basic use cases, eg.

- Listing several methods with different prices and shipping time for different countries.
- Offering a free (or discounted) shipping on orders above certain price threshold.
- Increasing the shipping price for heavy orders.

5.11.3 Weight

Weight is used to calculate the `WEIGHT_BASED` shipping price.

Weight is defined on the `ProductType` level and can be overridden for each `Product` and each `ProductVariant` within a `Product`.

5.12 Site Settings

Site settings module allows your users to change common shop settings from dashboard like its name or domain. Settings object is chosen by pk from `SITE_SETTINGS_ID` variable.

5.12.1 Context Processor

Thanks to `saleor.site.context_processors.settings` you can access *Site settings* in template with `settings` variable.

5.13 Pages

5.13.1 Setting up custom pages

You can set up pages such as "About us" or "Important Announcement!" in the Pages menu in dashboard. Note that if you are not an admin, you need to be in group with proper permissions.

5.13.2 Referencing pages in storefront

If you want to add a link to recently created page in storefront, all you need to do is to put the following code:

```
<a href="{% url "page:details" slug="terms-of-service" %}">Terms of Service</a>
```

in the proper template.

5.14 GDPR Compliance

Saleor handles few aspects of the GDPR regulation by default.

5.14.1 Deleting users

A user account can be deleted from the dashboard, by a staff user. This action takes place immediately.

From the storefront, a user can request his account deletion from within his profile settings, in such case, a confirmation email will be sent to the email address associated with the account.

Deleting a user will delete his account instance, but will leave all the data used for the checkout process untouched, for the financial record. This behavior is in accordance with the GDPR regulations.

5.14.2 Cookies

All cookies used by Saleor are strictly necessary to move around the website and use its features, therefore there's no need to notify the users about them.

5.15 Manual actions required

5.15.1 Privacy Policy and Terms of Service

Make sure your Terms of Service or Privacy Policy properly communicate to your users who you are and how you are using their data. We recommend you ensure your policies are up to date and clear to your readers.

5.16 GraphQL API (Beta)

Note: The GraphQL API is in the early version. It is not yet fully optimized against database queries and some mutations or queries may be missing.

Saleor provides a GraphQL API which allows to query and modify the shop's data in an efficient and flexible manner.

Learn more about GraphQL language and its concepts on the official website.

5.16.1 Endpoint

API is available under `/graphql` endpoint. Requests must be sent using HTTP `POST` method and `application/json` content type.

With the `DEBUG=True` setting enabled, Saleor exposes an interactive GraphQL editor under `/graphql`, that allows accessing the API from the browser.

5.16.2 Example Query

Querying for data in GraphQL can be very easy with tool GraphiQL, which can be used from a web browser.

Here is an example query that fetches three products:

```
query {
  products(first: 3){
    edges {
      node {
        name
        price {
          amount
        }
      }
    }
  }
}
```

results in the following result:

```
{
  "data": {
    "products": {
      "edges": [
        {
          "node": {
            "name": "Ford Inc",
            "price": {
              "amount": 64.98
            }
          }
        },
        {
          "node": {
            "name": "Rodriguez Ltd",
            "price": {
              "amount": 18.4
            }
          }
        },
        {
          "node": {
            "name": "Smith Inc",
```

(continues on next page)

(continued from previous page)

```
            "price": {
              "amount": 48.66
            }
          }
        }
      ]
    }
  }
}
```

5.16.3 Authorization

By default, you can query for public data such as published products or pages. To fetch protected data like orders or users, you need to authorize your access. Saleor API uses a JWT token authentication mechanism. Once you create a token, you have to include it as a header with each GraphQL request.

The authorization header has the following format:

```
Authorization: JWT token
```

Create a new JWT token with the `tokenCreate` mutation:

```
mutation {
  tokenCreate(email: "admin@example.com", password: "admin") {
    token
  }
}
```

Verification and refreshing the token is straightforward:

```
mutation tokenVerify($token: String!) {
  verifyToken(token: $token) {
    payload
  }
}
```

```
mutation tokenRefresh($token: String!) {
  tokenRefresh(token: $token) {
    token
    payload
  }
}
```

CHAPTER 6

Integrations

6.1 Search Engine Optimization (SEO)

Out of the box Saleor will automatically handle certain aspects of how search engines see and index your products.

6.1.1 Sitemaps

A special resource reachable under the `/sitemap.xml` URL serves an up to date list of products, categories and collections from your site in an easy to parse Sitemaps XML format understood by all major search engines.

6.1.2 Meta Tags

Meta keywords are not used as they are ignored by all major search engines because of the abuse this feature saw in the years since it was introduced.

Meta description will be set to the product's description field. This does not affect the search engine ranking but it affects the snippet of text shown along with the search result.

6.1.3 Robots Meta Tag

The robots meta tag utilize a page-specific approach to controlling how an individual page should be indexed and served to users in search results.

We've restricted Dashboard Admin Panel from crawling and indexation, content-less pages (eg. cart, sign up, login) are not crawled.

6.1.4 Structured Data

Homepage and product pages contain semantic descriptions in JSON-LD Structured Data format.

It does not directly affect the search engine ranking but it allows search engines to better understand the data ("this is a product, it's available, it costs $10").

It allows search engines like Google to show product photos, prices, availability, ratings etc. along with their search results.

6.1.5 Nofollow links

Search engine crawlers can't sign in or register as a member on your site, no reason to invite them to follow "register here" or "sign in" links, as there will be little to none valuable content.

This will optimize time spent by the crawler on the website, giving it time it to index more content-related pages.

6.2 Social Media Optimization (SMO)

6.2.1 Open Graph

For more effective and efficient social media engagement, we've added Open Graph Protocol to the Homepage and all products/categories.

Open Graph meta tags allows to control what content shows up (description, title, url, photo, etc.) when page is shared on social media, turning your web page into a rich object in a social graph.

6.3 Email Markup

Saleor is using schema.org markup to highlight the most important information within an email and allow users to easily interact with it. Order confirmation email will be displayed as an interactive summary card.

Email Markup is enabled by default, however your customers won't see it until you Register with Google

6.4 Elasticsearch

6.4.1 Installation

Elasticsearch search backend requires an Elasticsearch server. For the installation guide, please refer to *the official documentation <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>*.

Integration can be configured with set of environment variables. When you're deploying on Heroku - you can use add-on that provides Elasticsearch as a service. By default Saleor uses Elasticsearch 6.3.

If you're deploying somewhere else, you can use one of following services:

- http://www.searchly.com/
- https://www.elastic.co/cloud

6.4.2 Environment variables

ELASTICSEARCH_URL or BONSAI_URL or SEARCHBOX_URL URL to elasticsearch engine. If it's empty - search will be not available.

Example: `https://user:password@my-3rdparty-es.com:9200`

6.4.3 Data indexing

Saleor uses Django Elasticsearch DSL as a wrapper for Elasticsearch DSL to enable automatic indexing and sync. Indexes are defined in documents file. Please refer to documentation of above projects for further help.

Initial search index can be created with following command:

```
$ python manage.py search_index --rebuild
```

By default all indexed objects (products, users, orders) are reindexed every time they are changed.

6.4.4 Search integration architecture

Search backends use Elasticsearch DSL for query definition in saleor/search/backends.

There are two backends defined for elasticsearch integration, storefront and dashboard. Storefront search uses only storefront index for product only search, dashboard backend does additional searches in users and orders indexes as well.

6.4.5 Testing

There are two levels of testing for search functionality. Syntax of Elasticsearch queries is ensured by unit tests for backend, integration testing is done using VCR.py to mock external communication. If search logic is modified, make sure to record new communication and align test fixtures accordingly! Pytest will run VCR in never-recording mode on CI to make sure no attempts of communication are made, so make sure most recent cassettes are always included in your repository.

6.5 Google Analytics

Because of EU law regulations, Saleor will not use any tracking cookies by default.

We do however support server-side Google Analytics out of the box using Google Analytics Measurement Protocol.

This is implemented using google-measurement-protocol and does not use cookies at the cost of not reporting things impossible to track server-side like geolocation and screen resolution.

To get it working you need to export the following environment variable:

GOOGLE_ANALYTICS_TRACKING_ID Your page's Google "Tracking ID."

6.6 Google for Retail

Saleor has tools for generating product feed which can be used with Google Merchant Center. Final file is compressed CSV and saved in location specified by `saleor.data_feeds.google_merchant.FILE_PATH` variable.

To generate feed use command:

```
$ python manage.py update_feeds
```

It's recommended to run this command periodically.

Merchant Center has few country dependent settings, so please validate your feed at Google dashboard. You can also specify there your shipping cost, which is required feed parameter in many countries. More info be found at Google Support pages.

One of required by Google fields is *brand* attribute. Feed generator checks for it in variant attribute named *brand* or *publisher* (if not, checks in product).

Feed can be downloaded from url: `http://<yourserver>/feeds/google/`

6.7 Open Exchange Rates

This integration will allow your customers to see product prices in their local currencies. Local prices are only provided as an estimate, customers are still charged in your store's default currency.

Before you begin you will need an Open Exchange Rates account. Unless you need to update the exchange rates multiple times a day the free Subscription Plan should be enough but do consider paying for the wonderful service that Open Exchange Rates provides. Start by signing up and creating an "App ID".

Export the following environment variable:

`OPENEXCHANGERATES_API_KEY` Your store's Open Exchange Rates "App ID."

To update the exchange rates run the following command at least once per day:

```
$ python manage.py update_exchange_rates --all
```

Note: Heroku users can use the Scheduler add-on to automatically call the command daily at a predefined time.

6.8 Error tracking with Sentry

Saleor provides integration with Sentry - a comprehensive error tracking and reporting tool.

To enable basic error reporting you have to export an environment variable:

`SENTRY_DSN` Sentry Data Source Name

If you need to customize the service, please go to the official Sentry's documentation for Django for more details.

CHAPTER 7

Deployment

7.1 Docker

You will need to install Docker first.

Then use Docker to build the image:

```
$ docker build -t mystorefront .
```

Then you can run Saleor container with the following settings:

```
$ docker run -e SECRET_KEY=<SECRET_KEY> -e DATABASE_URL=<DATABASE_URL> -p 8000:8000␣
↪saleor
```

Please refer to *Configuration* for more environment variable settings.

7.2 Heroku

7.2.1 Configuration

Within the repo, git should already be initialized. All you have to do now is add the *heroku* remote with your *app-name*

```
$ heroku git:remote -a 'app-name'
$ heroku buildpacks:set heroku/nodejs
$ heroku buildpacks:add heroku/python
$ heroku addons:create heroku-postgresql:hobby-dev
$ heroku addons:create heroku-redis:hobby-dev
$ heroku addons:create sendgrid:starter
$ heroku config:set ALLOWED_HOSTS='<your hosts here>'
$ heroku config:set NODE_MODULES_CACHE=false
$ heroku config:set NPM_CONFIG_PRODUCTION=false
$ heroku config:set SECRET_KEY='<your secret key here>'
```

Note: Heroku's storage is volatile. This means that all instances of your application will have separate disks and will lose all changes made to the local disk each time the application is restarted. The best approach is to use cloud storage such as Amazon S3. See *Storing Files on Amazon S3* for configuration details.

7.2.2 Deployment

```
$ git push heroku master
```

7.2.3 Preparing the Database

```
$ heroku run python manage.py migrate
```

7.2.4 Updating Currency Exchange Rates

This needs to be run periodically. The best way to achieve this is using Heroku's Scheduler service. Let's add it to our application:

```
$ heroku addons:create scheduler
```

Then log into your Heroku account, find the Heroku Scheduler addon in the active addon list, and have it run the following command on a daily basis:

```
$ python manage.py update_exchange_rates --all
```

7.2.5 Enabling Elasticsearch

By default, Saleor uses Postgres as a search backend, but if you want to switch to Elasticsearch, it can be easily achieved using the Bonsai plugin. In order to do that, run the following commands:

```
$ heroku addons:create bonsai:sandbox-6 --version=5.4
$ heroku run python manage.py search_index --create
```

7.3 Storing Files on Amazon S3

If you're using containers for deployment (including Docker and Heroku) you'll want to avoid storing files in the container's volatile filesystem. This integration allows you to delegate storing such files to Amazon's S3 service.

7.3.1 Base configuration

AWS_ACCESS_KEY_ID Your AWS access key.

AWS_SECRET_ACCESS_KEY Your AWS secret access key.

7.3.2 Serving media files with a S3 bucket

If you want to store and serve media files, set the following environment variable to use S3 as media bucket:

AWS_MEDIA_BUCKET_NAME The S3 bucket name to use for media files.

If you are intending into using a custom domain for your media S3 bucket, you can set this environment variable to your custom S3 media domain:

AWS_MEDIA_CUSTOM_DOMAIN The S3 custom domain to use for the media bucket.

Note: The media files are every data uploaded through the dashboard (product images, category images, etc.)

7.3.3 Serving static files with a S3 bucket

By default static files (such as CSS and JS files required to display your pages) will be served by the application server.

If you intend to use S3 for your static files as well, set an additional environment variable:

AWS_STORAGE_BUCKET_NAME The S3 bucket name to use for static files.

If you are intending into using a custom domain for your static S3 bucket, you can set this environment variable to your custom S3 domain:

AWS_STATIC_CUSTOM_DOMAIN The S3 custom domain to use for the static bucket.

Note: You will need to configure your S3 bucket to allow cross origin requests for some files to be properly served (SVG files, Javascript files, etc.). For that, you have to the below instructions in your S3 Bucket's permissions tab under the CORS section.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>HEAD</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>*</AllowedHeader>
</CORSRule>
</CORSConfiguration>
```

CHAPTER 8

Guides

8.1 Orders

Saleor gives a possibility to manage orders from dashboard. It can be done in dashboard `Orders` tab.

8.1.1 Draft orders

To create draft order, first you must go to dashboard `Orders` tab and choose circular **+** button visible above the list of existing orders.

Those orders can be fully edited until confirmed by clicking *Create order*. You can modify ordered items, customer (also just set an email), billing and shipping address, shipping method and discount. Any voucher you apply will cause automatic order recalculation to fit actual state of an order every time it changes.

Confirming an order by clicking *Create order* will change status to unfulfilled and disable most of the edit actions. You can optionally notify customer - if attached any - about that order by sending email.

8.1.2 Marking orders as paid

You can manually mark orders as paid if needed in order details page. This option is visible only for unpaid orders as an action in *Payments* card.

> **Warning:** You won't be able to refund a payment handled manually. This is due to lack of enough data required to handle transaction.

8.2 Payments

8.2.1 Integrating a new Payment Gateway into Saleor

We are using a universal flow, that each gateway should fulfill, there are several methods that should be implemented.

Your changes should live under the `saleor.payment.gateways.<gateway name>` module.

Note: After completing those steps you will also need to integrate your payment gateway into your SPA Storefront's workflow.

---

get_client_token(connection_params)

A client token is a signed data blob that includes configuration and authorization information required by the payment gateway.

These should not be reused; a new client token should be generated for each payment request.

Example

```
def get_client_token(connection_params: Dict) -> str:
    gateway = get_payment_gateway(**connection_params)
    client_token = gateway.client_token.generate()
    return client_token
```

authorize(payment_information, connection_params)

A process of reserving the amount of money against the customer's funding source. Money does not change hands until the authorization is captured.

Example

```
def authorize(
        payment_information: Dict,
        connection_params: Dict) -> Dict:

    # Handle connecting to the gateway and sending the auth request here
    response = gateway.authorize(token=payment_information['token'])

    # Return a correct response format so Saleor can process it,
    # the response must be json serializable
    return {
        'is_success': response.is_success,
        'transaction_id': response.transaction.id,
        'kind': 'auth',
        'amount': response.amount,
        'currency': response.currency,
        'error': get_error(response),
        'raw_response': get_payment_gateway_response(response),
    }
```

refund(payment_information, connection_params)

Full or partial return of captured funds to the customer.

Example

```
def refund(
        payment_information: Dict,
        **connection_params: Dict) -> Dict:

    # Handle connecting to the gateway and sending the refund request here
    response = gateway.refund(token=payment_information['token'])

    # Return a correct response format so Saleor can process it,
    # the response must be json serializable
    return {
        'is_success': response.is_success,
        'transaction_id': response.transaction.id,
        'kind': 'refund',
        'amount': response.amount,
        'currency': response.currency,
        'error': get_error(response),
        'raw_response': get_payment_gateway_response(response),
    }
```

capture(payment_information, connection_params)

A transfer of the money that was reserved during the authorization stage.

Example

```
def capture(
        payment_information: Dict,
        connection_params: Dict) -> Dict:

    # Handle connecting to the gateway and sending the capture request here
    response = gateway.capture(token=payment_information['token'])

    # Return a correct response format so Saleor can process it,
    # the response must be json serializable
    return {
        'is_success': response.is_success,
        'transaction_id': response.transaction.id,
        'kind': 'refund',
        'amount': response.amount,
        'currency': response.currency,
        'error': get_error(response),
        'raw_response': get_payment_gateway_response(response),
    }
```

void(payment_information, connection_params)

A cancellation of a pending authorization or capture.

Example

```
def void(
        payment_information: Dict,
        connection_params: Dict) -> Dict:

    # Handle connecting to the gateway and sending the void request here
    response = gateway.void(token=payment_information['token'])

    # Return a correct response format so Saleor can process it,
    # the response must be json serializable
    return {
        'is_success': response.is_success,
        'transaction_id': response.transaction.id,
        'kind': 'refund',
        'amount': response.amount,
        'currency': response.currency,
        'error': get_error(response),
        'raw_response': get_payment_gateway_response(response),
    }
```

charge(payment_information, connection_params)

Authorization and capture in a single step.

Example

```
def charge(
        payment_information: Dict,
        connection_params: Dict) -> Dict:

    # Handle connecting to the gateway and sending the charge request here
    response = gateway.charge(
        token=payment_information['token'],
        amount=payment_information['amount'])

    # Return a correct response format so Saleor can process it,
    # the response must be json serializable
    return {
        'is_success': response.is_success,
        'transaction_id': response.transaction.id,
        'kind': 'refund',
        'amount': response.amount,
        'currency': response.currency,
        'error': get_error(response),
        'raw_response': get_payment_gateway_response(response),
    }
```

process_payment(payment_information, connection_params)

Used for the checkout process, it should perform all the necessary steps to process a payment. It should use already defined functions, like authorize and capture.

Example

```
def process_payment(
        payment_information: Dict,
        connection_params: Dict) -> Dict:

    # Authorize, update the token, then capture
    authorize_response = authorize(
        payment_information, connection_params)
    payment_information['token'] = authorize_response['transaction_id']

    capture_response = capture(
        payment_information, connection_params)

    # Return a list of responses, each response must be json serializable
    return [authorize_response, capture_response]
```

Parameters

name	type	description
payment_information	dict	Payment information, containing the token, amount, currency and billing.
connection_params	dict	List of parameters used for connecting to the payment's gateway.

Example

```
payment_information = {
    'token': 'token-used-for-transaction',  # provided by gateway
    'amount': Decimal('174.32'),  # amount to be authorized/captured/charged/refunded
    'currency': 'USD',  # ISO 4217 currency code
    'billing': {  # billing information
        'first_name': 'Joe',
        'last_name': 'Doe',
        'company_name': 'JoeDoe Inc.',
        'street_address_1': '3417 Bridge Street',
        'street_address_2': '',
        'city': 'Pryor',
        'city_area': '',
        'postal_code': '74361',
        'country': 'US',
        'country_area': 'OK',
        'phone': '+19188249023'},
    'shipping': {  # shipping information
        'first_name': 'Dollie',
        'last_name': 'Sullivan',
        'company_name': '',
        'street_address_1': '2003 Progress Way',
```

(continues on next page)

(continued from previous page)

```
        'street_address_2': '',
        'city': 'Waterloo',
        'city_area': '',
        'postal_code': '50797',
        'country': 'US',
        'country_area': 'IA',
        'phone': '+19188249023'},
    'order': 117,  # order id
    'customer_ip_address': '10.0.0.1',  # ip address of the customer
    'customer_email': 'joedoe@example.com',  # email of the customer
}
```

Returns

name	type	description
gateway_response	dict or list[dict]	Dictionary or list of dictionaries containing details about every transaction, with is_success set to True if no error occurred.
client_token	str	Unique client's token that will be used as his indentifier in the payment process.

Gateway response fields

name	type	description
transaction_id	str	Transaction ID as returned by the gateway.
kind	str	Transaction kind, one of: auth, capture, charge, refund, void.
is_success	bool	Status whether the transaction was successful or not.
amount	Decimal	Amount that the gateway actually charged or authorized.
currency	str	Currency in which the gateway charged, needs to be an ISO 4217 code.
error	str	An error message if one occured. Should be None if no error occured.

Additional fields can be sent for logging/debug purposes. The only requirement is that they're serializable by DjangoJSONEncoder. They will be saved in gateway_response field on Transaction model.

8.3 Example

8.3.1 Handling errors

Gateway-specific errors should be parsed to Saleor's universal format. More on this can be found in *Payments Architecture*.

8.3.2 Adding payment method to the old checkout (optional)

If you are not using SPA Storefront, there are some additional steps you need to perform in order to enable the payment method in your checkout flow.

Add a Form

Payment on the storefront will be handled via payment form, it should implement all the steps necessary for the payment to succeed. The form must implement *get_payment_token* that returns a token required to process payments. All payment forms should inherit from `django.forms.Form`.

Your changes should live under `saleor.payment.gateways.<gateway name>.forms.py`

Example

```
class BraintreePaymentForm(forms.Form):
    amount = forms.DecimalField()
    payment_method_nonce = forms.CharField()

    def get_payment_token(self):
        return self.cleaned_data['payment_method_nonce']
```

Implement create_form(data, payment_information, connection_params)

Should return the form that will be used for the checkout process.

Note: Should be added as a part of the provider's methods.

Example

```
def create_form(data, payment_information, connection_params):
    return BraintreePaymentForm(
        data, payment_information, connection_params)
```

Implement TEMPLATE_PATH

Should specify a path to a template that will be rendered for the checkout.

Example

```
TEMPLATE_PATH = 'order/payment/braintree.html'
```

Add template

Add a new template to handle the payment process with your payment form. Your changes should live under `saleor.templates.order.payment.<gateway name>.html`

8.3.3 Adding new payment gateway to the settings

```
PAYMENT_GATEWAYS = {
    'braintree': {
        'module': 'saleor.payment.gateways.braintree',
        'connection_params': {
            'sandbox_mode': get_bool_from_env('BRAINTREE_SANDBOX_MODE', True),
            'merchant_id': os.environ.get('BRAINTREE_MERCHANT_ID'),
            'public_key': os.environ.get('BRAINTREE_PUBLIC_KEY'),
            'private_key': os.environ.get('BRAINTREE_PRIVATE_KEY')
        }
    }
}
```

Please take a moment to consider the example settings above.

- **`braintree`** Gateway's name, which will be used to identify the gateway during the payment process. It's stored in the `Payment` model under the `gateway` value.
- **`module`** The path to the integration module (assuming that your changes live within the `saleor.payment.gateways.braintree.__init__.py` file)
- **`connection_params`** List of parameters used for connecting to the payment's gateway.

Note: All payment backends default to using sandbox mode. This is very useful for development but make sure you use production mode when deploying to a production server.

8.3.4 Enabling new payment gateway

Last but not least, if you want to enable your payment gateway in the checkout process, add it's name to the `CHECKOUT_PAYMENT_GATEWAYS` setting.

8.3.5 Tips

- Whenever possible, use `currency` and `amount` as **returned** by the payment gateway, not the one that was sent to it. It might happen, that gateway (eg. Braintree) is set to different currency than your shop is. In such case, you might want to charge the customer 70 dollars, but due to gateway misconfiguration, he will be charged 70 euros. Such a situation should be handled, and adequate error should be thrown.

8.4 Navigation

Saleor gives a possibility to configure storefront navigation. It can be done in dashboard `Navigation` tab.

You can add up to 3 levels of menu items inside every menu you create. Each menu item can point to an internal page with Category, Collection, Page or an external website by passing an extra URL.

8.4.1 Managing menu items

To manage menu items, first you must go to dashboard `Navigation` tab and choose menu to edit. If you want to manage nested menu items, you can navigate through listed menu items up and down.

To add new menu item, choose `Add` button visible above the list of menu items. Then fill up the form and click `Create`.

To edit menu item, choose `Edit` button visible next to a menu item on the list or `Edit menu item` button below menu item details, if you're inside menu item details view. Make any changes and click `Update`.

To remove a menu item, choose `Remove` button visible next to a menu item on the list or `Remove menu item` button below menu item details, if you're inside menu item details view. This action will remove all descendant items and can't be undone.

The menu items display on the storefront in the order that they are listed in menu items list. You can reorder them by handling an icon on the left to the menu items and dragging them to another position.

8.4.2 Managing menus

Dashboard gives you a possibility to add new menus.

There can be two active menus at once (one for the navbar, the other one for the footer, they can be the same).

Currently assigned menus can be changed via dashboard's `Navigation` panel.

Menu is rendered as a vertical list by default. You can change it by passing an extra `horizontal=True` argument. Horizontal menus with nested items appear as a dropdown menu on desktops.

8.5 Taxes

Saleor gives a possibility to configure taxes. It can be done in dashboard `Taxes` tab.

Taxes are charged according to the rates applicable in the country to which the order is delivered. If tax rate set for the product is not available, standard tax rate is used by default.

For now, only taxes in European Union are handled.

8.5.1 Configuring taxes

There are three ways in which you can configure taxes:

1. All products prices are entered with tax included

   If selected, all prices entered and displayed in dashboard will be treated as gross prices. For example: product with entered price 4.00 € and 19% VAT will have net price calculated to 3.36 € (rounded).

2. Show gross prices to customers in the storefront

   If selected, prices displayed for customers in storefront will be gross. Taxes will be properly calculated at checkout. Changing this setting has no effect on displaying orders placed in the past.

3. Charge taxes on shipping rates

   If selected, standard tax rate will be charged on shipping price.

8.5.2 Tax rates preview

You can preview tax rates in dashboard `Taxes` tab. It lists all countries taxes are handled for. You can see all available tax rates for each country in its details view.

8.5.3 Fetching taxes

Assuming you have provided a valid `VATLAYER_ACCESS_KEY`, taxes can be fetched via following command:

```
$ python manage.py get_vat_rates
```

If you do not have a VatLayer API key, you can get one by subscribing for free here.

> **Warning:** By default, Saleor is making requests to the VatLayer API through HTTP (insecure), if you are using a paid VatLayer subscription, you may want to set the settings `VATLAYER_USE_HTTPS` to `True`.

8.6 ReCaptcha

8.6.1 Pre-requirements

You can get your API key set from Google ReCaptcha.

8.6.2 Enable and Set-up

To enable ReCaptcha, you need to set those keys in your environment:

1. `RECAPTCHA_PUBLIC_KEY` to your public/ site API key;
2. `RECAPTCHA_PRIVATE_KEY` to your secret/ private API key.

Note: You are not required to set your public and private keys for development use. You can use the following development keys:

Key	Value
RECAPTCHA_PUBLIC_KEY	6LeIxAcTAAAAAJcZVRqyHh71UMIEGNQ_MXjiZKhI
RECAPTCHA_PRIVATE_KEY	6LeIxAcTAAAAAGG-vFI1TnRWxMZNFuojJ4WifJWe

You only have to set them up if you are using Saleor for production (Google will remind you if you do not).

8.7 Email Configuration and Integration

Saleor offers a few ways to set-up your email settings over SMTP servers and relays through the below environment variables.

8.7.1 `EMAIL_URL`

You can set the environment variable `EMAIL_URL` to the SMTP URL, which will contain a straightforward value as shown in below examples.

Description	URL
GMail with **SSL on**.	`smtp://my.gmail.username@gmail.com:my-password@smtp.gmail.com:465/?ssl=True`
OVH with **STARTTLS on**.	`smtp://username@example.com:my-password@pro1.mail.ovh.net:587/?tls=True`
A SMTP server unencrypted.	`smtp://username@example.com:my-password@smtp.example.com:25/`

Note: If you want to use your personal GMail account to send mails, you need to enable access to unknown applications in your Google Account.

Warning: Always make sure you set-up correctly at least your SPF records, and while on it, your DKIM records as well. **Otherwise your production mails will be denied by most mail servers or intercepted by spam filters.**

8.7.2 DEFAULT_FROM_EMAIL

You can customize the sender email address by setting the environment variable `DEFAULT_FROM_EMAIL` to your desired email address. You also can customize the sender name by doing as follow `Example Is Me <your.name@example.com>`.

Sendgrid Integration

After you created your sendgrid application, you need to set the environment variable `EMAIL_URL` as below, but by replacing `YOUR_API_KEY_HERE` with your API key.

```
smtp://apikey:YOUR_API_KEY_HERE@smtp.sendgrid.com:465/?ssl=True
```

Then, set the environment variable `DEFAULT_FROM_EMAIL` *as mentioned before*.

Note: As it is not in the setup process of sendgrid, if your 'from email' address is your domain, you need to make sure you at least correctly set your SPF DNS record and, optionally, set your DKIM DNS record as well.

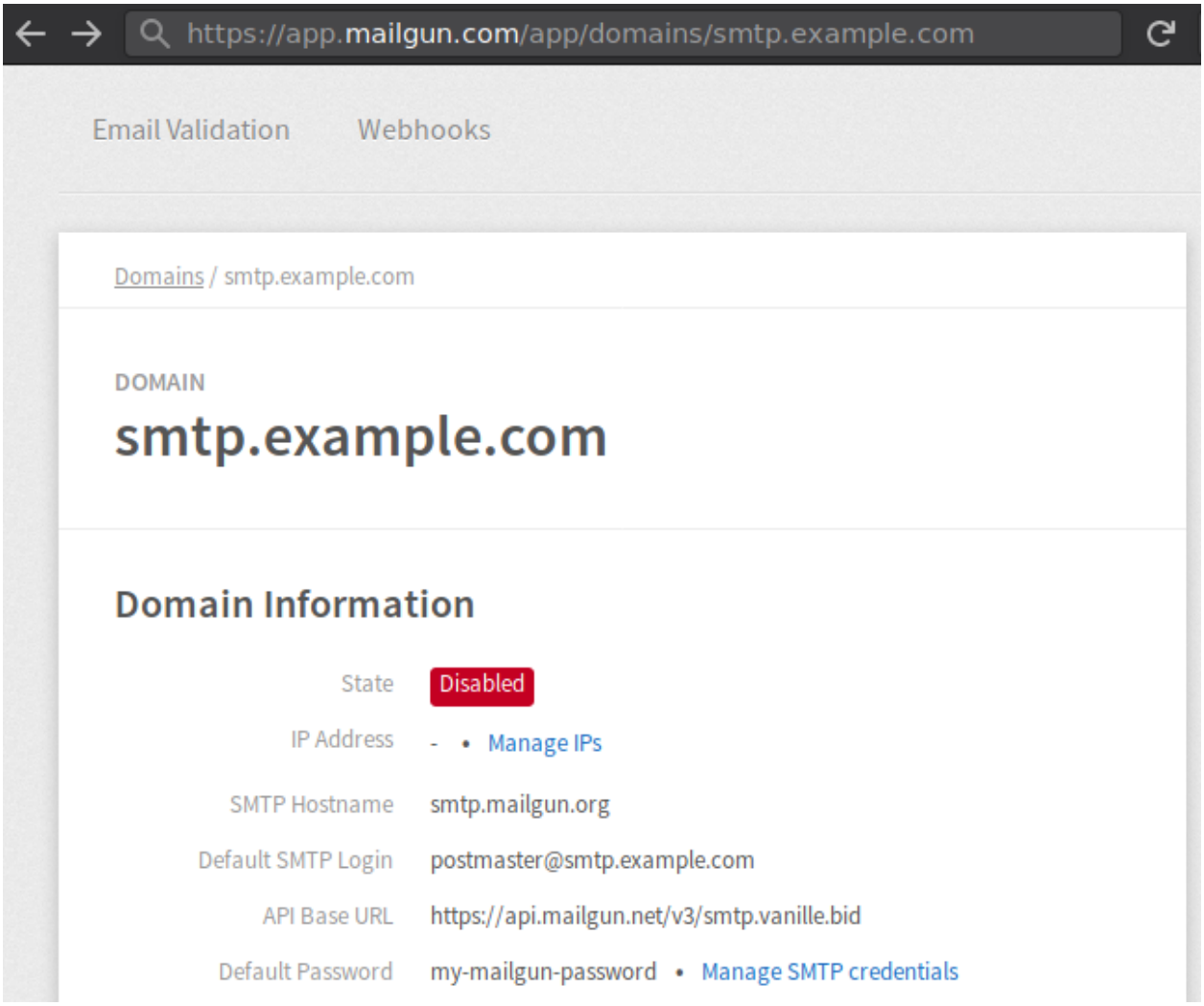
Mailgun Integration

After you added your domain in Mailgun and correctly set-up your domain DNS records, you can set the environment variable `EMAIL_URL` as below, but by replacing everything capitalized, with your data.

```
smtp://YOUR_LOGIN_NAME@YOUR_DOMAIN_NAME:YOUR_DEFAULT_MAILGUN_PASSWORD@smtp.mailgun.org:465/?ssl=True
```

8.7.3 Example

Let's say my domain name is `smtp.example.com` and I want to send emails as `john.doe@smtp.example.com` and my password is `my-mailgun-password`.



I have to set `EMAIL_URL` to:

```
smtp://john.doe@smtp.example.com:my-mailgun-password@smtp.mailgun.org:465/?
ssl=True
```

Mailjet Integration

After adding your domain in Mailjet, you have to set the environment variable `EMAIL_URL` as below, but by replacing everything capitalized, with your data, available at this URL.

```
smtp://YOUR_MAILJET_USERNAME:YOUR_MAILJET_PASSWORD@in-v3.mailjet.com:587/?
tls=True
```

Then, set the environment variable `DEFAULT_FROM_EMAIL` *as mentioned before*.

Amazon SES Integration

After having verified your domain(s) in AWS SES, and set-up DKIM and SPF records, you need to create your SMTP credentials.

Then, you can use this data to set-up the environment variable `EMAIL_URL` as below, by replacing everything capitalized, with your data.

```
smtp://YOUR_SMTP_USERNAME:YOUR_SMTP_PASSWORD@email-smtp.YOUR_AWS_SES_REGION.
amazonaws.com:587/?tls=True
```

Then, set the environment variable `DEFAULT_FROM_EMAIL` *as mentioned before*.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search