saleor Documentation

Release 2018.01

Mirumee Software

Feb 22, 2018

Contents

1 Getting Started **3**
- 1.1 Installation for macOS . . . 3
- 1.2 Installation for Windows . . . 5
- 1.3 Installation for Linux . . . 7
- 1.4 Configuration . . . 10
- 1.5 Creating an Administrator Account . . . 11
- 1.6 Example Data . . . 11

2 Customizing Saleor **13**
- 2.1 Using Docker for Development . . . 13
- 2.2 Customizing Templates . . . 14
- 2.3 Customizing Emails . . . 14
- 2.4 Customizing CSS and JavaScript . . . 15
- 2.5 Working with Python Code . . . 15
- 2.6 Internationalization . . . 15
- 2.7 Running Tests . . . 16
- 2.8 Continuous Integration . . . 16
- 2.9 Running with PyPy 3.5 . . . 16

3 Contributing Guides **17**
- 3.1 EditorConfig . . . 17
- 3.2 Coding Style . . . 17
- 3.3 Naming Conventions . . . 18

4 Architecture **21**
- 4.1 Product Structure . . . 21
- 4.2 Stock Management . . . 25
- 4.3 Order Management . . . 26
- 4.4 Internationalization . . . 27
- 4.5 Search . . . 28
- 4.6 Payments . . . 28
- 4.7 Site Settings . . . 29

5 Integrations **31**
- 5.1 Search Engine Optimization (SEO) . . . 31
- 5.2 Social Media Optimization (SMO) . . . 32
- 5.3 Elasticsearch . . . 32

5.4 Google Analytics . . . 33
5.5 Google for Retail . . . 33
5.6 Open Exchange Rates . . . 34

6 Deployment **35**
6.1 Docker . . . 35
6.2 Heroku . . . 35
6.3 Storing Files on Amazon S3 . . . 36

7 Indices and tables **37**

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 8 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.

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

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. Remeber to give your user the `SUPERUSER` privilege so it can create databases and database extensions.

   ..code-block:: console

       $ createuser –superuser –pwprompt saleor

   Enter `saleor` when prompted for password.

6. 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.

7. 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.

8. Prepare front-end assets:

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

9. Compile e-mails:

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

10. 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 3.6 Windows installer from the Python download page and follow the instructions.

Make sure that "**Add Python 3.6 to PATH**" is checked.

Node.js

Version 8 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 Visual C++ build tools.

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. Remeber to give your user the `SUPERUSER` privilege so it can create databases and database extensions.

6. 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.

7. 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.

8. Prepare front-end assets:

```
npm run build-assets
```

9. Compile e-mails:

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

10. 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 3.4 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.

Node.js

Version 8 or later is required. See the installation instructions.

Note: Debian and Ubuntu users who install Node.js using system packages will also need to install the `nodejs-legacy` package.

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` environmental 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 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.

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/assets/`.

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 groups they are assigned to.

1.6 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
```

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.

To build assets you will need node and webpack module bundler.

Note: Our configuration exposes PostgreSQL, Redis and Elasticsearch ports. If you have problems running this docker file because of port conflicts, you can remove 'ports' section from docker-compose.yml

2.1.2 Usage

1. Install JavaScript dependencies

```
$ npm install
```

2. Prepare static assets

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

3. Build emails

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

4. Build the containers using `docker-compose`

```
$ docker-compose build
```

5. Prepare the database

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

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

6. Run the containers

```
$ docker-compose up
```

By default, the application is started in debug mode, will automatically reload code 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` which has two subdirectories. `source` directory contains `*.email` and `*.mjml` files next to each other, grouped by apps' name. Those MJML files are compiled to `*.html` and put into `compiled` directory.

Plain emails in `*.email` include HTML version by referencing 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 pip-tools. Project's direct dependencies are listed in `requirements.in` and running `pip-compile` generates `requirements.txt` that has all versions pinned.

We recommend you use this workflow and keep `requirements.txt` 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:

```
$ pip install transifex-client
```

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
```

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,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.

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

Contributing Guides

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

3.1 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.

3.2 Coding Style

3.2.1 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.

3.3 Naming Conventions

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

3.3.1 Python Modules

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

3.3.2 Django Templates

Use underscore as a word separator.

3.3.3 Static Files

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

CHAPTER 4

Architecture

4.1 Product Structure

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

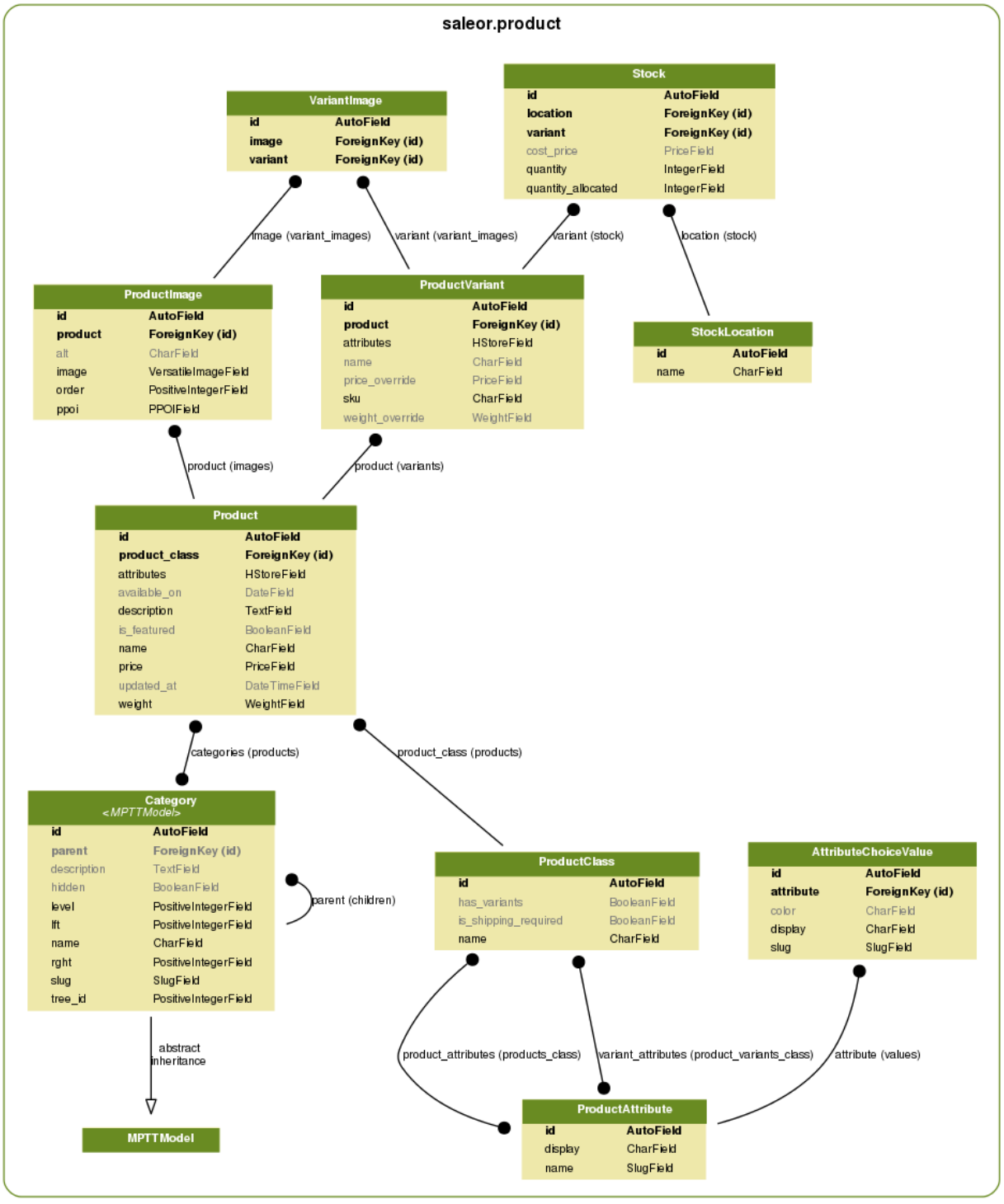
4.1.1 Overview

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*.

4.1.2 Class Diagram



4.1.3 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.

4.1.4 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*.

- **`available_on`** 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.

4.1.5 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.

4.1.6 Attributes

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

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 4.1: Attributes

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

Table 4.2: Product type

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

Table 4.3: Product

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

Table 4.4: 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 4.5: Attributes

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

Table 4.6: Product type

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

Table 4.7: 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.

4.2 Stock Management

Each product variant has a stock keeping unit (SKU) and can have any number of stock records.

A stock record represents that variant's availability in a single location. Multiple stock records are often used to represent different warehouses, different fulfilment partners or separate shipments of the same product that were obtained at different prices.

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

Example: There are five boxes of shoes in warehouse A. 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 stock records also has a *cost price* (the price that your store had to pay to obtain it).

4.2.1 Product Availability

A variant is *in stock* if at least one of its stock records has unallocated quantity.

The highest quantity that can be ordered is the sum of all available quantities in stock records. It allows each ordered product to be fulfilled in multiple order lines with all stock records.

4.2.2 Allocating Stock for New Orders

Once an order is placed, a stock records are selected to fulfil each order line. Default logic will select the stock records with the *lowest cost price* that holds enough stock. Quantity needed to fulfil the order line is immediately marked as *allocated*.

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

4.2.3 Decreasing Stock After Shipment

Once a delivery group is marked as shipped, each stock record used to fulfil its lines 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**.

4.3 Order Management

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

4.3.1 Delivery Group

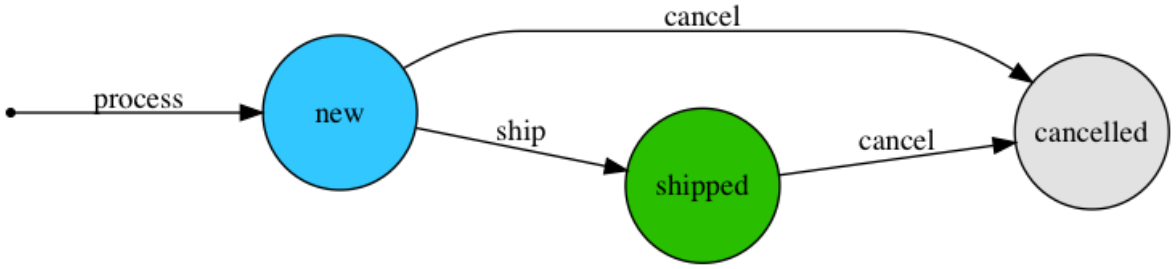
The delivery group represents a group of ordered items. By default, groups are created along with an order by splitting the cart into parts, depending whether a product requires shipping or not.

Most of the order management actions are taken on the delivery groups and include changing their statuses.

There are three possible delivery group statuses:

- **`NEW`** The default status of newly created delivery groups.
- **`SHIPPED`** The delivery group has been marked as shipped by a shop operator. Only `NEW` groups can be marked as shipped.
- **`CANCELLED`** The delivery group has been cancelled by a shop operator. Both `NEW` and `SHIPPED` groups can be marked as cancelled.

4.3.2 Delivery Group statuses flow



Regular flow of statuses is from `NEW` to `SHIPPED`. For managing states Saleor uses django-fsm package. Actions performed within changing status are placed in DeliveryGroup model and are as follows:

- **`process(cart_lines, discounts=None)`** Adds specified cart lines to a group with optionally charged discounts and allocates quantity in corresponding stocks. Performed during creating order in checkout process.
- **`ship()`** Decreases quantity in corresponding stocks. Performed just after shop operator marks group as shipped.
- **`cancel()`** Deallocates or increases quantity in corresponding stocks, depending whether `NEW` or `SHIPPED` group is cancelled respectively. Performed just after shop operator marks group as cancelled.

4.3.3 Order statuses

Order status is deduced based on statuses of its delivery groups. There are two possible statuses:

- **`OPEN`** There is at least one delivery group with the `NEW` status. An action by a shop operator is required to continue order processing.
- **`CLOSED`** There are no delivery groups with the `NEW` status. Order doesn't require further actions by a shop operator.

4.4 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.

4.4.1 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 not currently possible to translate database content (like product descriptions) but it's planned for a future release.

4.4.2 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.

4.5 Search

There are two search mechanisms available in Saleor.

The default is to use PotsgreSQL. 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*.

4.6 Payments

4.6.1 Supported Gateways

Saleor uses django-payments library to process payments.

Default configuration uses the *dummy* backend. It's meant to allow developers to easily simulate different payment results.

Here is a list of supported payment providers:

- Authorize.Net
- Braintree
- Coinbase
- Cybersource
- Dotpay
- Google Wallet
- PayPal
- Sage Pay
- Sofort.com
- Stripe

Please note that this list is only provided here for reference. Please consult django-payments documentation for an up to date list and instructions.

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.

4.6.2 3-D Secure

3-D Secure is a card protection protocol that allows merchants to partially mitigate fraud responsibility. In practice it greatly lowers the probability of a chargeback.

Saleor supports 3-D Secure but whether it's used depends on the payment processor and the card being used.

4.6.3 Fraud Protection

Some of the payment backends provide automatic fraud protection heuristics. If such information is available Saleor will show it in the order management panel.

4.6.4 Authorisation and Capture

Some of the payment backends support pre-authorising payments. Please see django-payments documentation for details.

Authorisation and capture is a two step process.

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

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

This is very useful when an exact price cannot be determined until after the order is prepared. It is also useful if your business prefers to manually screen orders for fraud attempts.

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

4.6.5 Refunds

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

4.7 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.

4.7.1 Context Processor

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

CHAPTER 5

Integrations

5.1 Search Engine Optimization (SEO)

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

5.1.1 Sitemaps

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

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.2 Social Media Optimization (SMO)

5.2.1 Open Graph

For more effective and efficient social media engagement, we've added **'Open Graph Protocol<http://ogp.me/>'_** 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.

5.3 Elasticsearch

5.3.1 Installation

Elasticsearch search backend requires an Elasticsearch server. For development purposes docker-compose will create a Docker container running an Elasticsearch server instance.

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 5.4.3.

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

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

5.3.2 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`

5.3.3 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.

5.3.4 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.

5.3.5 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.

5.4 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."

5.5 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/`

5.6 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.

CHAPTER 6

Deployment

6.1 Docker

You will need to install Docker first.

Before building the image make sure you have all of the front-end assets prepared for production:

```
$ npm run build-assets --production
$ npm run build-emails
$ python manage.py collectstatic
```

Then use Docker to build the image:

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

6.2 Heroku

6.2.1 Configuration

```
$ heroku create --buildpack https://github.com/heroku/heroku-buildpack-nodejs.git
$ heroku buildpacks:add https://github.com/heroku/heroku-buildpack-python.git
$ 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.

6.2.2 Deployment

```
$ git push heroku master
```

6.2.3 Preparing the Database

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

6.2.4 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:

6.2.5 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
```

6.3 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.

Set the following environment variables to use S3 to store and serve media files:

AWS_ACCESS_KEY_ID Your AWS access key.

AWS_SECRET_ACCESS_KEY Your AWS secret access key.

AWS_MEDIA_BUCKET_NAME The S3 bucket name to use for media files.

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.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search