## Students

* Javadagha Salmanov
* Adil Valizada
* Gulandam Saidbayli
* Cafar Abbasov
* Nuran Amirkhanov
* Chingiz Gurbanov
* Eljan Rustamov

# Setup Documentation

This documentation outlines the setup process for the Django project configured in [settings.py](2%7D%5D%22shop-site-lesson-project\\shop\\settings.py%22). Follow these steps to configure your environment correctly.

## Prerequisites

* Python 3.8 or higher
* Django 4.1.3
* PostgreSQL
* An SMTP server for email sending capabilities

## Environment Variables

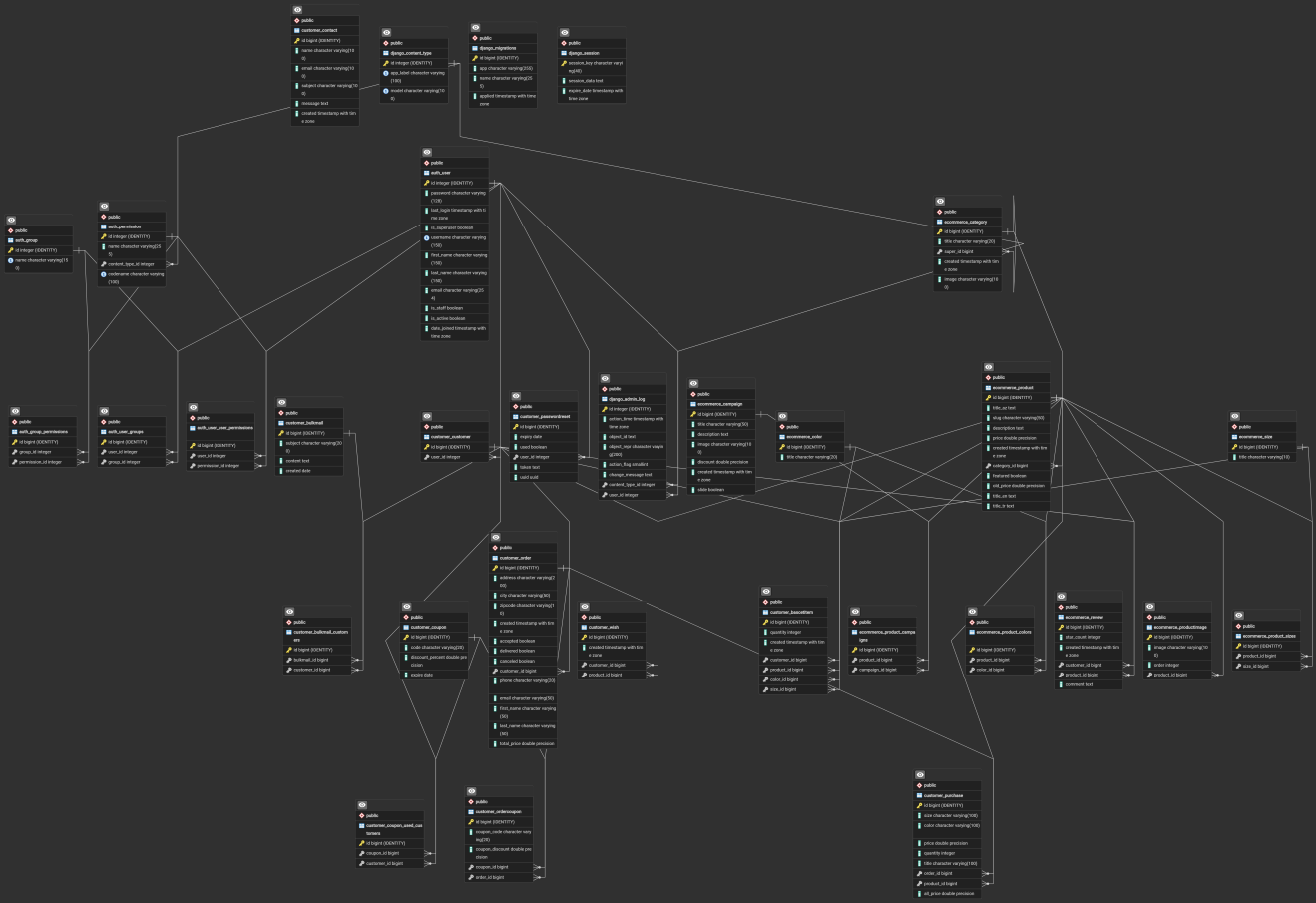
The project relies on environment variables for configuration. Set the following variables in your environment:

* [SECRET\_KEY](shop/settings.py): A secret key for Django. Generate one using Django’s get\_random\_secret\_key() function.
* [DEBUG](shop/settings.py): Set to True for development, False for production.
* [ALLOWED\_HOSTS](shop/settings.py): A comma-separated list of hosts/domains the app can serve. Example: localhost,127.0.0.1
* Database Configuration:
  + DB\_NAME: The name of your PostgreSQL database.
  + DB\_USER: The username for your database.
  + DB\_PASSWORD: The password for your database user.
  + DB\_HOST: The host of your database, typically localhost for a local setup.
  + DB\_PORT: The port your database listens on, usually 5432 for PostgreSQL.
* Email Configuration:
  + [EMAIL\_HOST](shop/settings.py): The host of your SMTP server.
  + [EMAIL\_PORT](shop/settings.py): The port your SMTP server uses.
  + [EMAIL\_HOST\_USER](shop/settings.py): The username for your SMTP server.
  + [EMAIL\_HOST\_PASSWORD](shop/settings.py): The password for your SMTP server.
  + EMAIL\_USE: Set to SSL or TLS depending on your SMTP server’s configuration.
* Security Settings:
  + [SECURE\_HSTS\_SECONDS](shop/settings.py): The number of seconds HTTP Strict Transport Security will be enforced.
  + [SECURE\_SSL\_REDIRECT](shop/settings.py): Set to True to redirect all non-HTTPS requests to HTTPS.
  + [SECURE\_HSTS\_INCLUDE\_SUBDOMAINS](shop/settings.py): Set to True to include subdomains in the HSTS policy.
  + [SESSION\_COOKIE\_SECURE](shop/settings.py): Set to True to make the session cookie secure (only sent over HTTPS).
  + [CSRF\_COOKIE\_SECURE](shop/settings.py): Set to True to make the CSRF cookie secure.
  + [SECURE\_HSTS\_PRELOAD](shop/settings.py): Set to True to allow preloading of the site’s HSTS policy.

## Database Setup

Ensure PostgreSQL is installed and running. Create a database that matches the DB\_NAME environment variable you’ve set.

## Database



## Running the Project

1. Install dependencies:

pip install -r requirements.txt

1. Apply migrations to create the database schema:

python manage.py migrate

1. Run the development server:

python manage.py runserver

## Docker Configuration

This project is configured to run with Docker, simplifying the setup and deployment processes. Here’s how to get started with Docker:

### Prerequisites

* Docker
* Docker Compose

### Running the Project with Docker

1. **Build the Docker Images**

* Navigate to the project root directory and run the following command to build the Docker images specified in the docker-compose.yml file:
* docker-compose build

1. **Start the Containers**

* After building the images, start the containers with: bash docker-compose up This command starts all the services defined in docker-compose.yml. The backend service will be accessible at http://localhost:8000, and the database service uses a volume shop\_db\_data to persist data.

1. **Environment Variables**

* The Docker configuration uses an .env file to set environment variables for the containers. Ensure you have an .env file at the root of your project with the necessary variables. Refer to .env.example for an example configuration.
* Database Initialization
* With the containers running, you may need to apply database migrations. You can do this by executing the following command:

## Static and Media Files

* Static files are served from the staticfiles/ directory and configured under STATIC\_URL and STATIC\_ROOT.
* Media files are uploaded to and served from the media/ directory, configured under MEDIA\_URL and MEDIA\_ROOT.

## Internationalization

The project is configured for multiple languages with default language set to English. Translations are located in the locale directory.

## Security

Review the security settings section and ensure appropriate values are set before deploying to production.

## Email Configuration

Configure the SMTP settings according to your email service provider to enable email sending capabilities.

## Caching

The project uses file-based caching configured under the CACHES setting. Ensure the cache directory is writable by the web server.

## Unit Testing

This project uses Django’s built-in TestCase class for unit testing. Tests are organized by application within the project structure. Here’s how to run and write tests for the project:

### Running Tests

To run all tests in the project, execute the following command from the root directory:

python manage.py test

To run tests for a specific application, specify the application name:

python manage.py test customer  
python manage.py test ecommerce

### Writing Tests

Tests are defined in the tests.py file within each application directory. Each test case class should inherit from django.test.TestCase.

#### Example Test Case

Below is an example of a test case for the Product model in the ecommerce application:

from django.test import TestCase  
from .models import Product  
  
class ProductTest(TestCase):  
 def setUp(self):  
 Product.objects.create(name="Test Product", price=10.00)  
  
 def test\_product\_creation(self):  
 product = Product.objects.get(name="Test Product")  
 self.assertEqual(product.price, 10.00)

This test case creates a [Product](ecommerce/models.py) instance in the [setUp](ecommerce/tests.py) method and then tests that the product was created with the correct price.

### Best Practices

* Group related tests into the same [TestCase](virtualenvs/ecommerce-shop-site-lesson-project-7M1L-YYh/Lib/site-packages/django/test/testcases.py) class.
* Use the [setUp](ecommerce/tests.py) method to create common test data for each test method.
* Use descriptive names for your test methods to clearly indicate what they test.

For more information on testing in Django, refer to the [official Django testing documentation](https://docs.djangoproject.com/en/stable/topics/testing/).

## Admin Panel

The project’s Django admin panel is configured through customizations in the [admin.py](%22customer\\admin.py%22) files located in the [customer](mmerce\\lesson\\customer%22) and [ecommerce](mmerce\\lesson\\ecommerce%22) applications. Below is a summary of these configurations:

### Customer App Admin Configuration ([customer/admin.py](%22customer\\admin.py%22))

* **Models Registered Directly:**
  + [Customer](%22customer/models.py%22)
  + [Wish](%22customer/models.py%22)
  + [BascetItem](%22customer/models.py%22)
  + [Contact](%22customer/models.py%22)
  + [Purchase](%22customer/models.py%22)
  + [Coupon](%22customer/models.py%22)
  + [OrderCoupon](%22customer/models.py%22)
  + [PasswordReset](%22customer/models.py%22)
  + [BulkMail](%22customer/models.py%22)
* **Custom Admin Classes:**
  + [OrderAdmin](%22customer/admin.py%22): Configures the admin interface for the [Order](%22customer/models.py%22) model. It uses two inline classes, [OrderCouponInline](%22customer/admin.py%22) and [PurchaseInline](%22customer/admin.py%22), to display related [OrderCoupon](%22customer/models.py%22) and [Purchase](%22customer/models.py%22) instances within the same page as the [Order](%22customer/models.py%22) instance.
    - [OrderCouponInline](D): Displays [OrderCoupon](D) instances in a tabular inline format. It is set to not allow adding extra instances directly from the [Order](D) admin page ([extra = 0](D)).
    - [PurchaseInline](D): Displays [Purchase](D) instances in a stacked inline format, also without allowing extra instances to be added directly ([extra = 0](D)).

### Ecommerce App Admin Configuration ([ecommerce/admin.py](%22ecommerce\\admin.py%22))

* **Models Registered Directly:**
  + [Size](%22ecommerce/models.py%22)
  + [Color](%22ecommerce/models.py%22)
  + [Category](%22ecommerce/models.py%22)
  + [Campaign](%22ecommerce/models.py%22)
  + [ProductImage](%22ecommerce/models.py%22)
* **Custom Admin Classes:**
  + [ProductAdmin](%22ecommerce/admin.py%22): Customizes the admin interface for the [Product](%22ecommerce/models.py%22) model. It specifies [readonly\_fields](%22ecommerce/admin.py%22) to include id, slug, and created. It also includes two inline classes, [ProductImageInline](%22ecommerce/admin.py%22) and [ReviewInline](%22ecommerce/admin.py%22), to display related [ProductImage](%22ecommerce/models.py%22) and [Review](%22ecommerce/models.py%22) instances within the same page as the [Product](%22ecommerce/models.py%22) instance.
    - [ProductImageInline](%22ecommerce/admin.py%22): Configures a tabular inline view for [ProductImage](%22ecommerce/models.py%22) instances. It includes image, image\_tag, and order fields, with image\_tag being read-only. It allows adding one extra [ProductImage](%22ecommerce/models.py%22) instance directly from the [Product](%22ecommerce/models.py%22) admin page ([extra = 1](%22customer/admin.py%22)).
    - [ReviewInline](%22ecommerce/admin.py%22): The configuration for this inline class is not provided in the excerpt, but it is implied to be similar in purpose to [ProductImageInline](%22ecommerce/admin.py%22), allowing [Review](%22ecommerce/models.py%22) instances to be managed directly from the [Product](%22ecommerce/models.py%22) admin page.

These configurations enhance the Django admin interface by allowing detailed management of related models directly from the admin pages of their parent models, improving the efficiency of administrative tasks.