

# Grocery Store App Documentation

GitHub repository link: [https://github.com/AAST-MVPs/GroceryShop/tree/build\\_deploy/](https://github.com/AAST-MVPs/GroceryShop/tree/build_deploy/)

Host link: <https://grocery-app-i7qb.onrender.com/dashboard/index/>

Username: [ziad@example.com](mailto:ziad@example.com)

Password: ziad1234

**Note:** The service goes down due to inactivity. Therefore, the first request will take a few minutes, then it will be normal.

## **Group Members:**

Ahmad Mongy Saad - 120200033

Ahmed Abdelkader Ahmed - 120200028

Mahmoud Akrm Mohamed - 120200045

Mohamed Ayman Mohamed - 120200081

Peter Fayez Shafiq - 120200073

Ziad Hesham Al-Safy - 120200078

## **To:**

Dr. Mohamed Abbasy

Eng. Ahmed Hesham

## Contents

1. Introduction: .....	3
2. Project Structure: .....	3
Main Apps: .....	3
Account: .....	3
Checkout: .....	3
Dashboard: .....	3
Product: .....	4
Order: .....	5
3. Database: .....	5
Database Schemas: .....	6
ERD diagrams: .....	8
4. Features: .....	10
5. Web scrapping & Database Population: .....	10
Scrapped Data: .....	10
Dependencies: .....	11
Configuration: .....	11
Safety Measures: .....	11
Data Population: .....	11
6. Deployment: .....	11
Local: .....	11
Prerequisites: .....	11
Installation: .....	12
Configuration: .....	12
On Cloud (Render.com): .....	13
1. Create a PostgreSQL Database on Render: .....	13
2. Building and Starting the Application: .....	13
3. Deployment: .....	13
7. Division of work: .....	14
Main app Order & main app Checkout: .....	14
Main app Product: .....	14
Models of Product app (Category, Brand): .....	14
Deployment & Database: .....	14
Web Scrapping & Database population: .....	14
Frontend & Templates & Integration from backend to frontend: .....	14

# 1. Introduction:

Welcome to the documentation for the grocery store project. This application is a web-based grocery shopping Application developed using Django (Python), and PostgreSQL.

## 2. Project Structure:

The project follows the general Django structure (MVT). In MVT, M stands for Model, which is used to create actual database tables and their fields. V stands for views, which are python functions that accept web requests and deliver web responses. T stands for Templates, which contain the static content of a Django project like Html, CSS, and JavaScript, along with the image used in the project.

- *static/*: Static files (CSS, JS, images)
- *templates/*: HTML templates
- *groceryshop/*: Main website. Contains all web apps (all-auth, checkout, dashboard, etc.).
- *settings.py*: Django settings file.

### Main Apps:

**Account:** contains the main functions to deal with the user, authentication, and passwords' security. Its main functionalities are inherited from the “all-auth” library.

**Checkout:** contains the main functions to deal with checkouts, order creation, and user addresses.

#### Models:

**CheckoutLine:** The model for viewing the price of each product in the user's order separately. This model is used to simplify the larger Checkout model. It contains a foreign key for the product in this specific line and another key for the larger checkout model.

**Checkout:** the larger model for dealing with creating the user order and saving it in the database. It contains one or more *CheckoutLines*, each with a different product. Each checkout is related to one user and one shipping address.

#### Views - URLs:

1. Index checkout view: *host/checkout*
2. Address creation and edit: *host/checkout/create-edit/address*
3. CheckoutLine deletion (Item deletion): *host/checkout/delete/<pk>*
4. Order Creation: *host/checkout/create/order*

### Dashboard:

#### Views-URLs:

- 1- Dashboard Index: *host/dashboard/index*

## 2- Product:

### Admin / Staff:

- Create-Update-Delete Product: *host/dashboard/product/{create, update, delete}*
- Create-Delete Product Image: *host/dashboard/product/{create, delete}/image/<int:pk>*

### User:

- List Product: *host/dashboard/product/list*
- Product Image List: *host/dashboard/pk/list/images*

## Product:

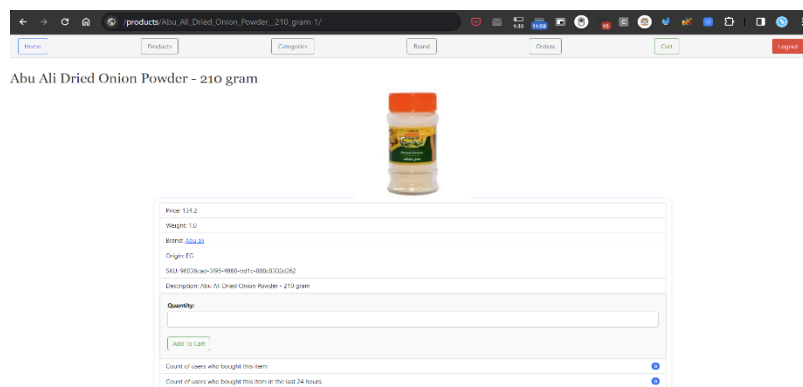
### Models:

**Product:** Represents individual products in the grocery store. It stores essential information such as the product name, description, price, weight, SKU (Stock Keeping Unit), stock quantity, expiration date, and references to the associated brand and category. This model is central to managing and displaying detailed information about each product in the system.

**Product image:** is responsible for storing information about the brands of products available in your grocery store. Brands often play a significant role in customer preferences, and associating products with specific brands allows for better organization and presentation of products. This model contains a name field to identify the brand.

**Product category:** categorizes products based on their type or classification. It helps organize the products into logical groups, making it easier for users to navigate and find what they're looking for. This model contains a name field to represent the category name.

### Views:



\_\_\_\_\_/PRODUCTS/ABU\_ALI\_DRIED\_ONION\_POWDER\_\_210\_GRAM-1/

- 1- *products/<str:slug>*: Shows product details, other users purchase history, and add to cart option.

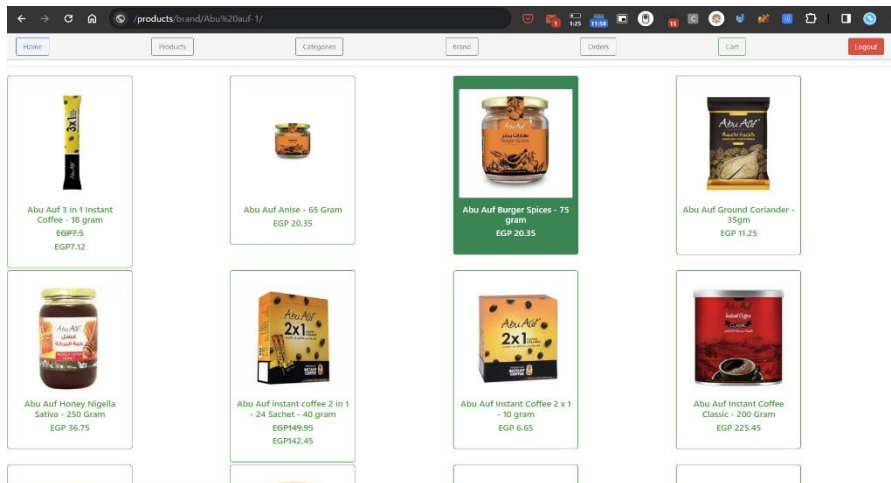
Ex. /PRODUCTS/ABU\_ALI\_DRIED\_ONION\_POWDER\_\_210\_GRAM-1/

- 2- *products/category/<str:slug>*: Shows Category details and Description.

Ex: /PRODUCTS/CATEGORY/GROCERY-1/

3- *product/brand/<str:slug>*: Shows the products available under this brand.

Ex: */PRODUCTS/BRAND/ABU%20AUF-1/*



*/PRODUCTS/BRAND/ABU%20AUF-1/*

## Order:

### Models:

**Order:** represents a customer's order in the app. It includes information such as the order ID, creation timestamp, status (e.g., pending, shipped), customer email, checkout token, optional note, total order amount, total discount applied, and references to the associated shipping information and user.

**Orderline:** represents individual lines within an order, each corresponding to a specific product. It includes an order line ID, a reference to the order to which it belongs, the product being ordered, the quantity of that product, and the price of the product at the time of the order. This model facilitates tracking the specific products and quantities within each order.

### Views:












- 1- *order/<int:pk>*: Shows the details of current order including order id, time of creation, status of the order, customer email, checkout token, optional note, total order amount, total discount applied.

## 3. Database:







In a web application for a grocery store, various database models can be employed to organize and manage data efficiently. The choice of a specific database model depends on the application's requirements, scalability needs, and the nature of the data. Here in this application, a Relational Database Model (RDBMS) is used since it is structured, well-suited for transactional data and supports complex queries.

# Database Schemas:





In the following we illustrate the Properties of the tables:

Column Name	#	Data type	Identity	Collation	Not Null	Default
 id	1	int8	By Default		[v]	
 name	2	varchar(255)		<a href="#">default</a>	[v]	
 slug	3	varchar(255)		<a href="#">default</a>	[v]	
 description	4	text		<a href="#">default</a>	[v]	
 price	5	float8			[v]	
 weight	6	float8			[v]	
 sku	7	uuid			[v]	
 stock	8	int4			[v]	
 exp_date	9	date			[v]	
 brand_id	10	int8			[ ]	
 category_id	11	int8			[ ]	


Product Table Schema

Column Name	#	Data type	Identity	Collation	Not Null	Default
 created	1	timestamptz			[v]	
 email	2	varchar(254)		<a href="#">default</a>	[v]	
 token	3	uuid			[v]	
 note	4	text		<a href="#">default</a>	[v]	
 shipping_id	5	int8			[ ]	
 user_id	6	int4			[ ]	

Checkout Schema

Column Name	#	Data type	Identity	Collation	Not Null	Default
 id	1	int8	By Default		[v]	
 quantity	2	int4			[v]	
 checkout_id	3	uuid			[v]	
 product_id	4	int8			[v]	

CheckoutLine Schema

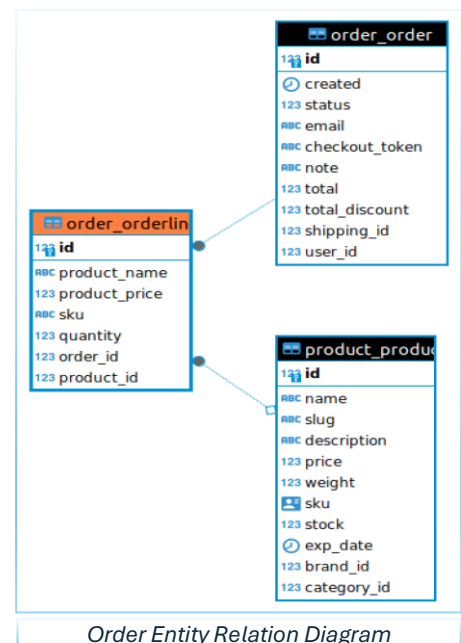
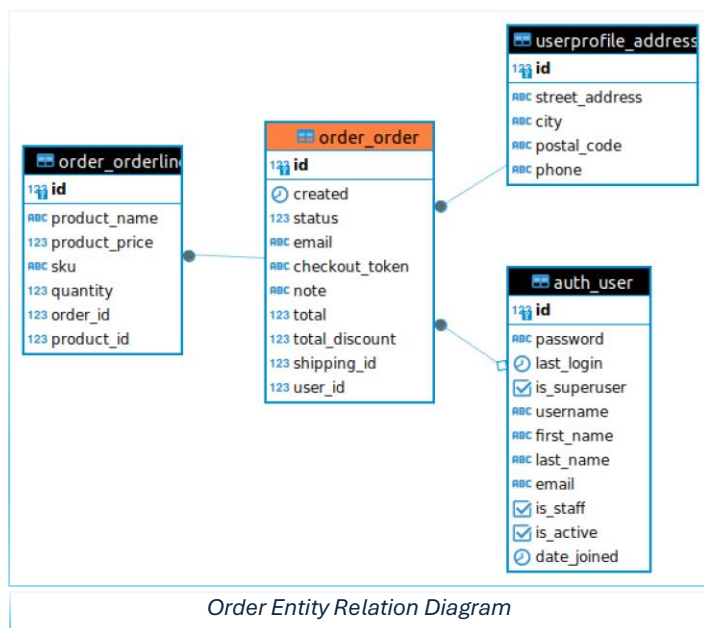
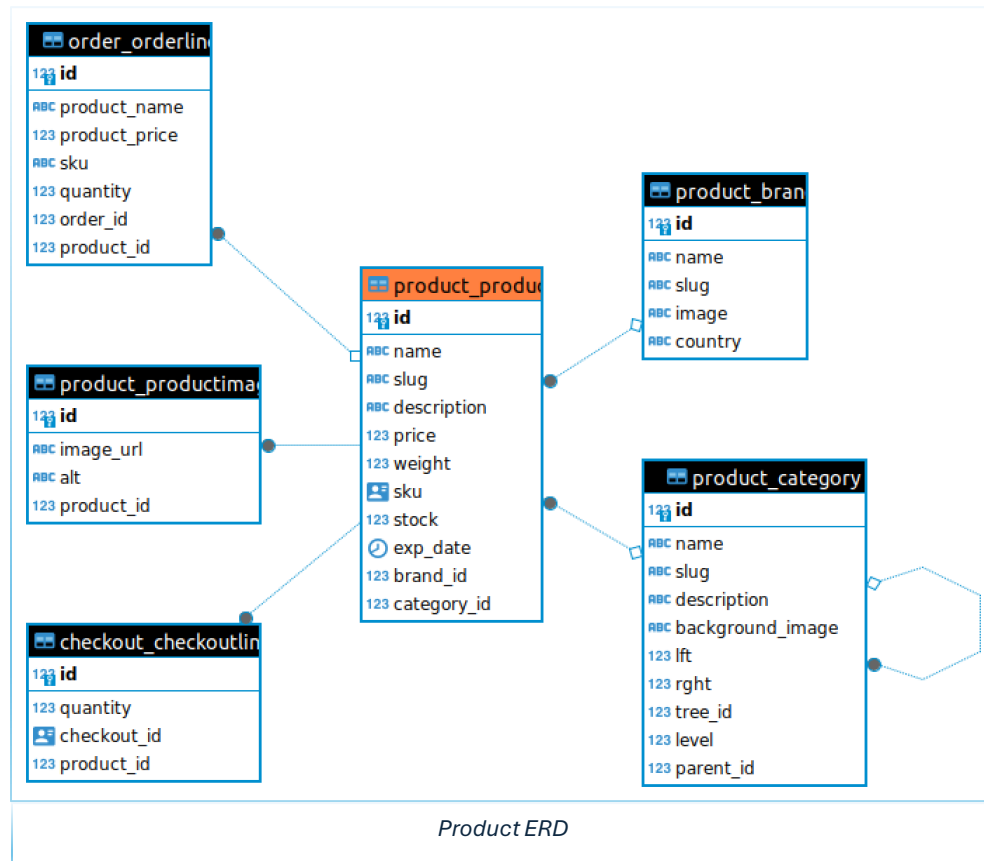
Column Name	#	Data type	Identity	Collation	Not Null	Default
id	1	int8	By Default		[v]	
 created	2	timestamptz			[v]	
status	3	int4			[v]	
email	4	varchar(254)		<a href="#">default</a>	[v]	
checkout_tok	5	varchar(36)		<a href="#">default</a>	[v]	
note	6	text		<a href="#">default</a>	[v]	
total	7	float8			[v]	
total_discoun	8	float8			[v]	
shipping_id	9	int8			[v]	
user_id	10	int4			[ ]	
Order Schema						

Column Name	#	Data type	Identity	Collation	Not Null	Default
id	1	int8	By Default		[v]	
product_name	2	varchar(255)		default	[v]	
product_price	3	float8			[v]	
sku	4	varchar(50)		default	[v]	
quantity	5	int4			[v]	
order_id	6	int8			[v]	
product_id	7	int8			[ ]	

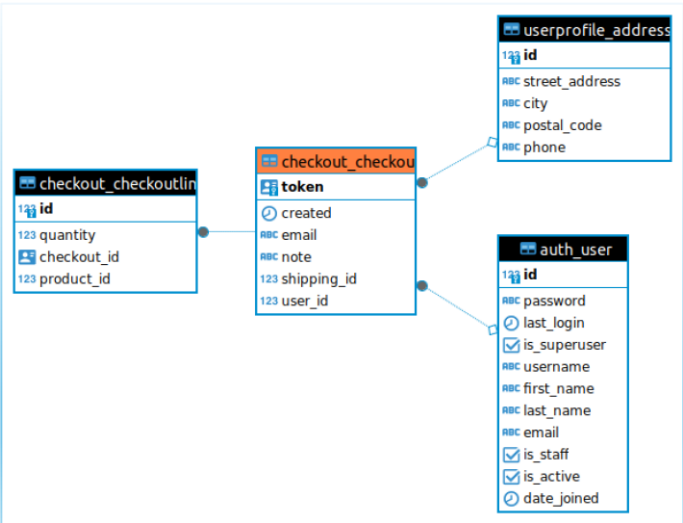
OrderLine Schema

## ERD diagrams:

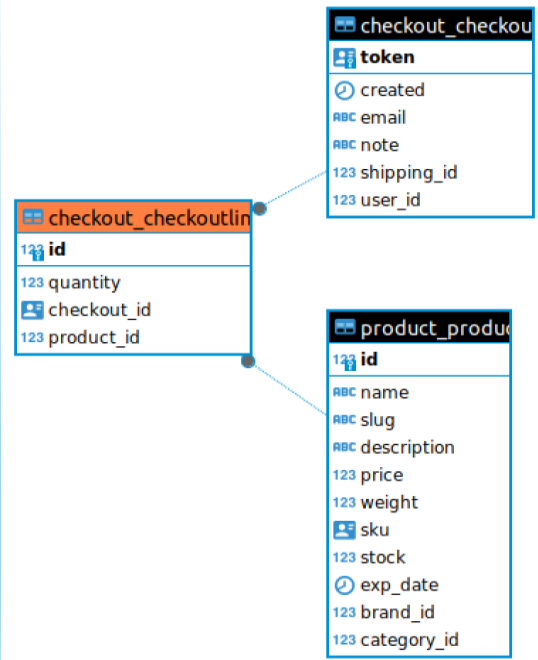
They illustrate the tables with its relations and different attributes used in the project database:







Checkout Order Entity Relation Diagram



CheckoutLine Order Entity Relation Diagram

## 4. Features:

**USER AUTHENTICATION FEATURES:** Registration and Email Confirmation, Login / Logout, Password Reset

**BROWSING PRODUCTS:** Users can view a list of available products and their details. User Can view Products under Specific category or brand.

**FILTER & SEARCH (BONUS):** The user can filter the list of groceries based on the price range, the brand name, and the brand nationality. Also, users can filter the products views for bestsellers.

**NEAR-EXPIRY DISCOUNTS (BONUS):** The website automatically generates discounts on the products approaching their expiry date.

**HIDE EXPIRED PRODUCTS (BONUS):** The website automatically hides expired products from user views.

**ADDING TO CART:** Logged-in users and guest users can add products to their shopping cart.

**CHECKOUT PROCESS (BONUS):** Users can proceed to checkout, providing delivery details and payment details (payment simulation).

**ORDER HISTORY (BONUS):** Registered Users can view their order history.

## 5. Web scrapping & Database Population:

The web scraper is a crucial component of our data acquisition process for the grocery store app. It is designed to extract product information from Amazon Best Sellers in the Grocery category.

The web scraper utilizes Python, Requests, BeautifulSoup, and Pandas to extract data from Amazon's Best Sellers in the Grocery category. It retrieves information such as product description, brand, rating, price, and image URL, providing a comprehensive dataset for our grocery store app.

The web scraper has another version to scrape data from Carrefour website which uses the same dependencies plus Json, but the implementation is different to work with dynamic behavior of carrefour website.

### Scrapped Data:

- Product Name
- Product Description
- Product Category
- Product Price
- Product Image URL

## Dependencies:

```
$ pip install requests JSON beautifulsoup4 pandas
```

## Configuration:

Adjust the 'no\_pages' variable to specify the number of pages to scrape from Amazon's Best Sellers. Additionally, ensure that the "User-Agent" header in the 'headers' variable complies with Amazon's policies.

## Safety Measures:

- Avoid aggressive scraping that may lead to IP bans or legal consequences.
- Ensure compliance with Amazon's and Carrefour's policies and terms of service.
- Add header to authenticate the scraper as human.

## Data Population:

We used the [django-extensions](#) library in order to make a script that populates the database schema with the scraped data. We made the script "data.py" which inserts in the database each instance of the scraped data in the following order to considerate the model dependencies:

- 1- Category
- 2- Brand
- 3- Product
- 4- Product Image

We used the command "runscript" to run this script in the django shell when uploading to render's environment. We ensure the data validity and coherency using DBeaver software to visualize the database and its tables in detail.

## 6. Deployment:

### Local:

### Prerequisites:

Before you begin, ensure you have the following prerequisites installed:

- Python 3.7
- PostgreSQL
- Configure and Create User and an Empty Database

## Installation:

Follow these steps to set up the project locally:

Create a virtual environment in a new folder, source activate the virtual environment and then follow these steps :

```
$ git clone -b localhost --single-branch https://github.com/AAST-MVPs/GroceryShop.git
$ pip install -r requirements.txt
```

## Configuration:

Adjust the configuration settings in settings.py:

- Database configuration

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'db_name',
        'USER': 'db_user',
        'PASSWORD': 'db_user_password',
        'HOST': 'localhost',
        'PORT': 'portnumber',
    }
}
```

- Secret key

This value [the SECRET\_KEY setting] is the key to securing signed data – it is vital you keep this secure, or attackers could use it to generate their own signed values.

```
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py runserver
```

You will then see something similar to this:

```
Django version 5.0, using settings 'groceryshop.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Use that URL to run the server.

## On Cloud (Render.com):

### 1. Create a PostgreSQL Database on Render:

- Set up a PostgreSQL database on Render for your application.

### 2. Building and Starting the Application:

- Setup your Environment Variables:

```
ALLOWED_HOSTS = "*" localhost"
DATABASE = <database_url>
DEBUG = False
SECRET_KEY = <generate_secret_key>
```

- Run the following commands:

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

**NOTE:** For the initial deployment, uncomment line 9 in build.sh to create a superuser for the Django admin page and populate the database with data from products.csv.

- For more details view scripts/data.py

### 3. Deployment:

- Finally, for deploying, run this command **./build.sh**
- Once the initial setup is completed, you can deploy your application on Render.com.

#### USAGE:

Upon successful deployment, visit the application URL to access the grocery shop demo. Use the features to browse products, make purchases, and experience the functionality of the web application.