**Info 409**

**Online Shop Application –**

**Django Web site**

**59113 – Lynn Al Karkaf**

**59185 – Noura Achkar**

Summary

[1 Introduction 3](#_Toc163338543)

[2 Class Diagram 4](#_Toc163338544)

[3 Website Description 4](#_Toc163338545)

[3.1 Models 4](#_Toc163338546)

[3.2 Views 5](#_Toc163338547)

[3.3 Forms 6](#_Toc163338548)

[4 Technical Issues 7](#_Toc163338549)

[4.1 The Issue 7](#_Toc163338550)

[4.2 Solution 7](#_Toc163338551)

# Introduction

The Django website is built upon an existing Online Electronics Shop application created using DW and UML.

Here's a succinct overview of the database schema:

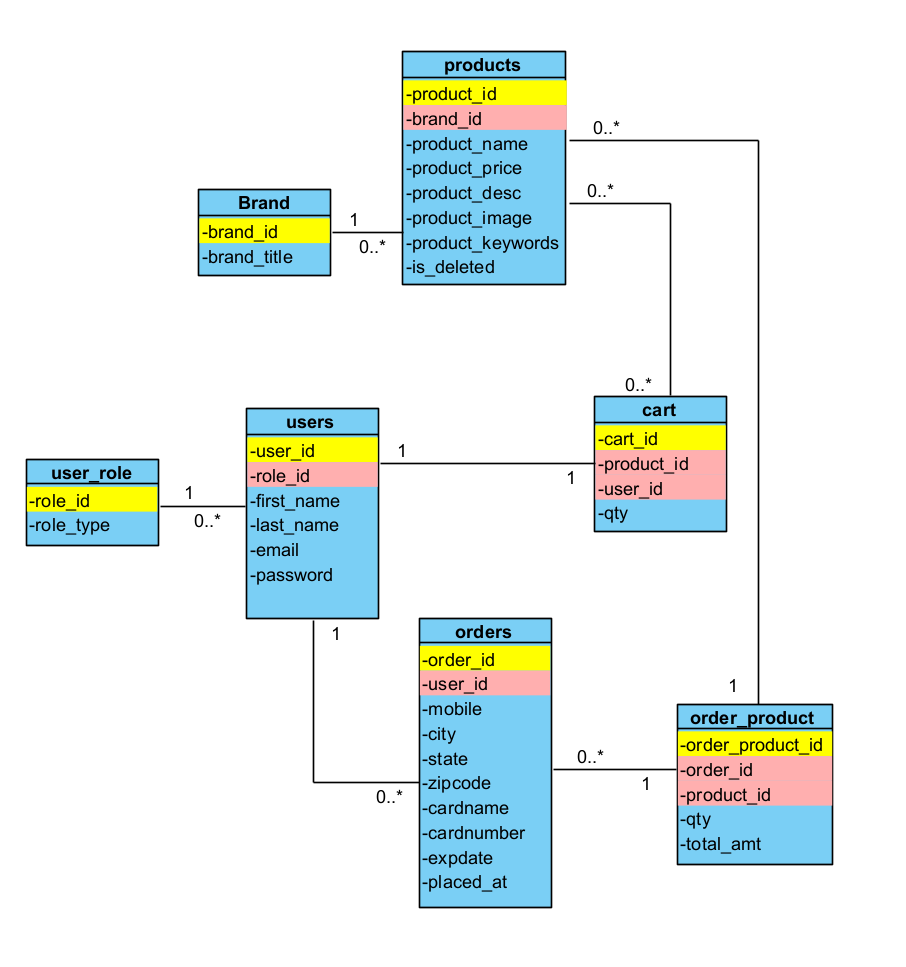
* **Products**: Representing the items available for sale, each associated with details such as name, description, price, and brand.
* **Brands**: Containing information about the brands of electronics offered in the shop.
* **Users**: Including profiles with roles, and contact information, enabling interactions such as purchasing, selling, and cart management.
* **Cart**: Holding temporary information about products selected by users for purchase before they proceed to checkout.
* **Orders**: Recording finalized transactions, containing details like order ID, date, and customer information.
* **User\_Role**: Defining roles for users, possibly distinguishing between Seller and Buyer.
* **Order\_Product**: Facilitating the relationship between orders and products, allowing for efficient tracking of purchased items within each order.

For the project division, our team strategically split the workload between Lynn and Noura, ensuring efficient progress and specialization in key areas.

* + Noura spearheaded the **authentication** aspect, focusing on user-centric functionalities such as login, registration, and password management. Additionally, she led the development of the **seller** functionalities, empowering users to list products for sale, manage their profiles, and browse products seamlessly. Noura was responsible for handling the intricacies of the *brands*, *user*, and *products tables*, ensuring their smooth integration and functionality within the system.
  + Lynn, on the other hand, tackled the **buyer**-centric features, concentrating on functionalities like product browsing, cart management, and checkout processes. She meticulously worked on the *cart*, *products* and *orders* tables, ensuring smooth transactions and a seamless user experience during the purchasing journey.

As a result, we gained valuable experience in Django development, further enhancing our capabilities in web application development.

# Class Diagram



# Website Description

In our Django project, we've structured it into three distinct apps, each catering to specific functionalities.

1. Authentication: This app serves as the gateway for users to interact securely with the platform. Here, users can seamlessly log in, create new accounts through registration, and ensure account security by changing passwords when needed.
2. OnlineApp: Focusing on empowering sellers, this app allows them to efficiently manage their products. Within this app, sellers can add new products to their inventory, remove existing ones, modify product details, and search for specific products. Additionally, sellers have the capability to update their profiles to reflect accurate information.
3. Buyer: Dedicated to streamlining the purchasing process for customers, this app enables buyers to browse through available products, add desired items to their carts for later purchase, and proceed through the checkout process smoothly to finalize their transactions. This app ensures a seamless and enjoyable shopping experience for our customers.

## Models

* The **Authentication** app doesn't incorporate any custom models as we're leveraging Django's pre-existing User model.
* The **OnlineApp** app models:

In this model, we've employed foreign keys to establish connections between various tables, including many-to-many relationships, such as the link between the products and brands tables.

* The **buyer** app models:  
  In these models, we utilize foreign keys to establish relationships between different tables, facilitating seamless data management. This includes managing one-to-one, one-to-many, and many-to-many relationships, ensuring efficient organization and retrieval of information across the database.

## Views

* + **authentication views.py**: This Django views manage crucial aspects of user authentication. They handle user login (loginUser), logout (logoutUser), registration (signup), and password change (changePassword). These functions ensure secure access, group assignment, and password updates, collectively forming a comprehensive system for user authentication within the application.
  + **onlineApp views.py**: These Django views collectively manage essential functionalities within the OnlineApp, including adding, editing, and deleting products, as well as editing user profiles and searching for products based on specific criteria.  
    The *add\_product* function allows sellers to add new products to their inventory. It verifies seller status, validates form input, assigns the seller to the product, and saves the product to the database.

The *my\_products* function displays products owned by the logged-in seller. It ensures only sellers can access this page and retrieves the seller's products from the database.

The *edit\_product* function verifies seller status, retrieves the product to edit, validates form input, and updates the product in the database.

The *delete\_product* function allows sellers to delete their products. It verifies seller status, retrieves the product to delete, and removes it from the database.

The *edit\_profile* function handles user profile editing. It updates the user's profile information based on form input.

Lastly, the *search\_product* function facilitates product searching by brand and custom text. It retrieves brands from the database, filters products based on search criteria, and displays search results.

* + **Buyer views.py:**  
    The *home* function renders the home page (home1.html).  
    The *products* function fetches products from the database, filters them by brand if specified, calculates cart-related metrics like quantity and total amount, and renders the products page (Buyer/products.html).  
    The *search* function handles product search queries, filtering products based on search keywords and brand, and renders the search results page (Buyer/search.html).  
    The *productdetails* function retrieves product details from the database, including descriptions, and renders the product details page (Buyer/productdetail.html).  
    The *deletefromcart* function manages removing products from the cart, updating quantities, and redirects back to the previous page with a success message.  
    The *cart* function retrieves cart items for the current user, calculates total quantity and amount, and renders the cart page (Buyer/cart.html).  
    The *clearcart* function clears all items from the cart and redirects to the cart page.  
    The *checkout* function processes checkout requests, validates form input, creates order records, and redirects to the home page with a success message upon completion.  
    The *updateProfile* function handles profile updates, retrieves user information, and renders the profile update page (Buyer/updateprofile.html).

## Forms

Within our project, we've embraced a blend of Django's built-in forms and bespoke forms tailored to our distinct needs. Alongside utilizing Django's pre-built forms, we've fine-tuned some to better suit our project's intricacies. This hybrid strategy capitalizes on Django's robust form capabilities while seamlessly accommodating our project's unique design and validation necessities.

1. **RegistrationForm**: Utilized in the signup view, this form streamlines the user registration process by capturing essential user details and facilitating group selection for improved access control.
2. **login\_form**: Integrated into the loginUser view, this form facilitates secure user login by capturing user credentials and verifying them against stored data, ensuring authenticated access to the platform.
3. **changePasswordForm**: Employed in the changePassword view, this form enables users to modify their passwords securely, ensuring proper validation and data integrity during the password change process.
4. **SearchForm**: Utilized within the search\_product view, this form empowers users to search for products based on brand and custom text, enhancing the product discovery experience within the application.
5. **AddProductForm**: Integrated into the add\_product view, this form simplifies the process of adding new products to the platform, capturing essential product details and ensuring seamless data submission.
6. **EditProfileForm**: Utilized in the edit\_profile view, this form facilitates the updating of user profiles by capturing user details and enabling seamless profile modification, enhancing user control and customization within the application.
7. **CheckoutForm**: Utilized in the checkout view, this form facilitates the checkout process by capturing user details such as first name, last name, mobile number, city, state, and email address, ensuring smooth order processing and delivery.
8. **UpdateProfileForm**: Employed in the updateProfile view, this form allows users to update their profile information, including first name, last name, email, mobile number, state, and city, enhancing user customization and personalization options within the application.

# Technical Issues

## The Issue

During the migration process in our Django project, several technical issues arose that required attention and resolution. One common issue was related to database schema changes conflicting with existing data or constraints, leading to migration failures. This occurred when adding new fields to existing models or modifying existing fields' attributes.

## Solution

We carefully reviewed the changes made to models and their corresponding migrations to ensure they aligned with our database schema and data integrity requirements. We double-checked the migration files generated by Django to verify that they accurately reflected the intended schema changes.