TASK-1

Develop a basic e- commerce store with product listings and a shopping cart. Use html/css/javascript for the frontend and a backend framework like django or express.js with a database for product management and order processing.

here's a basic outline of how you can create a simple e-commerce store with product listings and a shopping cart using HTML/CSS/JavaScript for the frontend and Django for the backend with a SQLite database:

1. **Backend Setup (Django)**:
    - Install Django if you haven't already: `pip install django`
    - Create a new Django project: `django-admin startproject ecommerce`
    - Create a new Django app for your store: `python manage.py startapp store`
    - Define your product model in `store/models.py`.
    - Run migrations to create the database schema: `python manage.py makemigrations` and `python manage.py migrate`
    - Create views in `store/views.py` for product listings, product detail, and adding/removing items from the cart.
    - Define URLs in `store/urls.py`.
    - Implement logic for managing the shopping cart, such as adding/removing items and calculating the total price.
    - Create templates in the `store/templates` directory for rendering HTML pages.
2. **Frontend Setup (HTML/CSS/JavaScript)**:
    - Create HTML templates for product listings, product detail page, and the shopping cart.
    - Style your HTML templates using CSS to make them visually appealing.
    - Use JavaScript to handle user interactions such as adding/removing items from the cart, updating the cart total, and handling checkout.
3. **Integrate Frontend with Backend**:
    - Use Django's template engine to render dynamic content in your HTML templates.
    - Fetch product data from the backend and display it on the frontend.
    - Implement AJAX requests to interact with the backend for adding/removing items from the cart without page reloads.
4. **Database Setup**:
    - Django uses SQLite by default, which is suitable for development. For production, you may switch to a more robust database like PostgreSQL or MySQL.
5. **Authentication and Authorization**:
    - Implement user authentication and authorization to allow users to sign up, log in, and manage their carts/orders.
6. **Testing**:
    - Test your application thoroughly to ensure all functionalities work as expected.

    - Test both frontend and backend components, including edge cases and error handling.
7. **Deployment**:
    - Once your application is ready, deploy it to a web server. Popular options include Heroku, AWS, or DigitalOcean.
    - Set up proper security measures, such as HTTPS, to protect user data during transit.
8. **Maintenance and Updates**:
    - Regularly update your application with new features, bug fixes, and security patches.
    - Monitor performance and scalability as your user base grows.
    - Back up your database regularly to prevent data loss.

This is just a basic outline to get you started. Depending on your specific requirements, you may need to add more features and functionalities to your e-commerce store.

Writing the entire program here would be too extensive, but I can provide you with a simplified version of the Django backend and HTML/CSS/JavaScript frontend code to get you started. Let's create a basic e-commerce store with product listings and a shopping cart:

Backend (Django):

1. **models.py** (inside `store` app):

```
python
Copy code
from django.db import models

class Product(models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.title
```

2. **views.py**:

```
python
Copy code
from django.shortcuts import render
from .models import Product

def product_list(request):
    products = Product.objects.all()
    return render(request, 'product_list.html', {'products': products})
```

3. **urls.py** (inside `store` app):

```
python
```

```
Copy code
from django.urls import path
from . import views

urlpatterns = [
    path('', views.product_list, name='product_list'),
]
```

4. **product_list.html** (inside `templates/store` directory):

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product List</title>
</head>
<body>
    <h1>Products</h1>
    <ul>
        {% for product in products %}
        <li>{{ product.title }} - ${{ product.price }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Frontend (HTML/CSS/JavaScript):

1. **index.html**:

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>E-commerce Store</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Welcome to Our Store</h1>
        <div id="product-list"></div>
        <div id="cart"></div>
    </div>
    <script src="scripts.js"></script>
</body>
</html>
```

2. **styles.css**:

```
css
```

```
Copy code
.container {
    width: 80%;
    margin: 0 auto;
    text-align: center;
}

h1 {
    font-size: 2em;
    margin-bottom: 20px;
}

li {
    list-style: none;
}

#cart {
    margin-top: 50px;
}
```

3. **scripts.js**:

```
javascript
Copy code
document.addEventListener('DOMContentLoaded', function() {
    fetch('/api/products')
        .then(response => response.json())
        .then(data => {
            const productList = document.getElementById('product-list');
            data.forEach(product => {
                const item = document.createElement('div');
                item.innerHTML = `<p>${product.title} -
$${product.price}</p>`;
                productList.appendChild(item);
            });
        });
});
```

This is just a starting point. You would need to continue building out your Django backend with views for adding/removing items from the cart, processing orders, implementing user authentication, etc. Similarly, on the frontend side, you would need to handle user interactions like adding items to the cart, updating quantities, and processing the checkout.