E-commerce WebDevelopment

- **Kyle Schwartz**
  Team Leader,
  Front End Developer
- **Sai Ram Ankata**
  Backend Developer
- **Vira Vasylevska**
  Cloud Application Developer
- **Brenden Dane**
  Database Developer
- **Teja Reddy Palle**
  QA Specialist
- **Yijun Liu**
  Cloud Security Specialist

SOUTHEAST MISSOURI STATE UNIVERSITY · 1873®

CS 506 Distributed Cloud Computing

Project Analysis

Motivation

The rapid growth of e-commerce has created a high demand for efficient and user-friendly platforms, yet many small businesses face challenges in establishing an online presence due to limited technical expertise. Recognizing this gap, we aim to develop a scalable e-commerce solution that enhances the shopping experience for users while supporting small businesses.

Problems We Aim to Solve:

Accessibility	User Experience	Scalability

What We Hope to Achieve:

- A fully functional, visually appealing e-commerce website.
- Hands-on experience with Django and cloud-based database integration.
- An impactful solution that bridges the gap between small businesses and online commerce.

Project Objectives

- Create a User-Friendly Platform
- Build a Functional Shopping System
- Enhance Data Security
- Ensure Basic Scalability
- Provide Responsive Design
- Facilitate Simple Maintenance
- Promote Accessibility

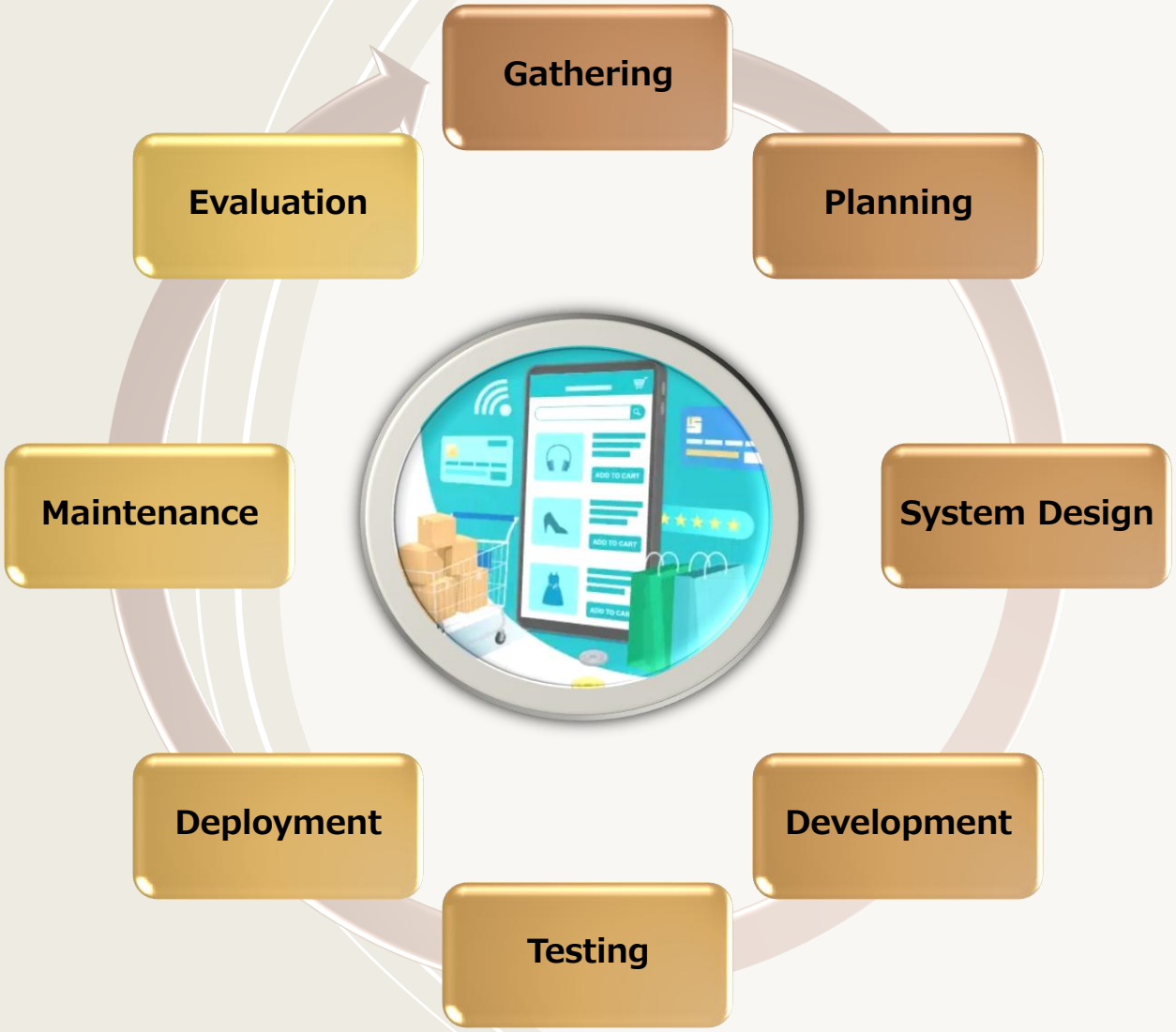
SDLC Model

I. Gathering

- Understand the project objectives and gather user requirements.
- Identify the key features: user authentication (login/signup), product listing, product details, cart, checkout, and order confirmation.
- Document user needs, such as a user-friendly interface and secure payment handling for future enhancements.

II. Planning

- Define the project scope, timeline, and deliverables.
- Assign roles within the team (e.g., front-end, back-end, database integration).
- Select tools and technologies: HTML, CSS, JavaScript for the front end, Python Django for the back end.



SDLC Model

III. System Design:

- Create wireframes for the user interface (UI) design of the website.
- Define the layout for pages like login, sign-up, product details, cart, and checkout.
- Design the database schema for storing user information, product details, and orders.

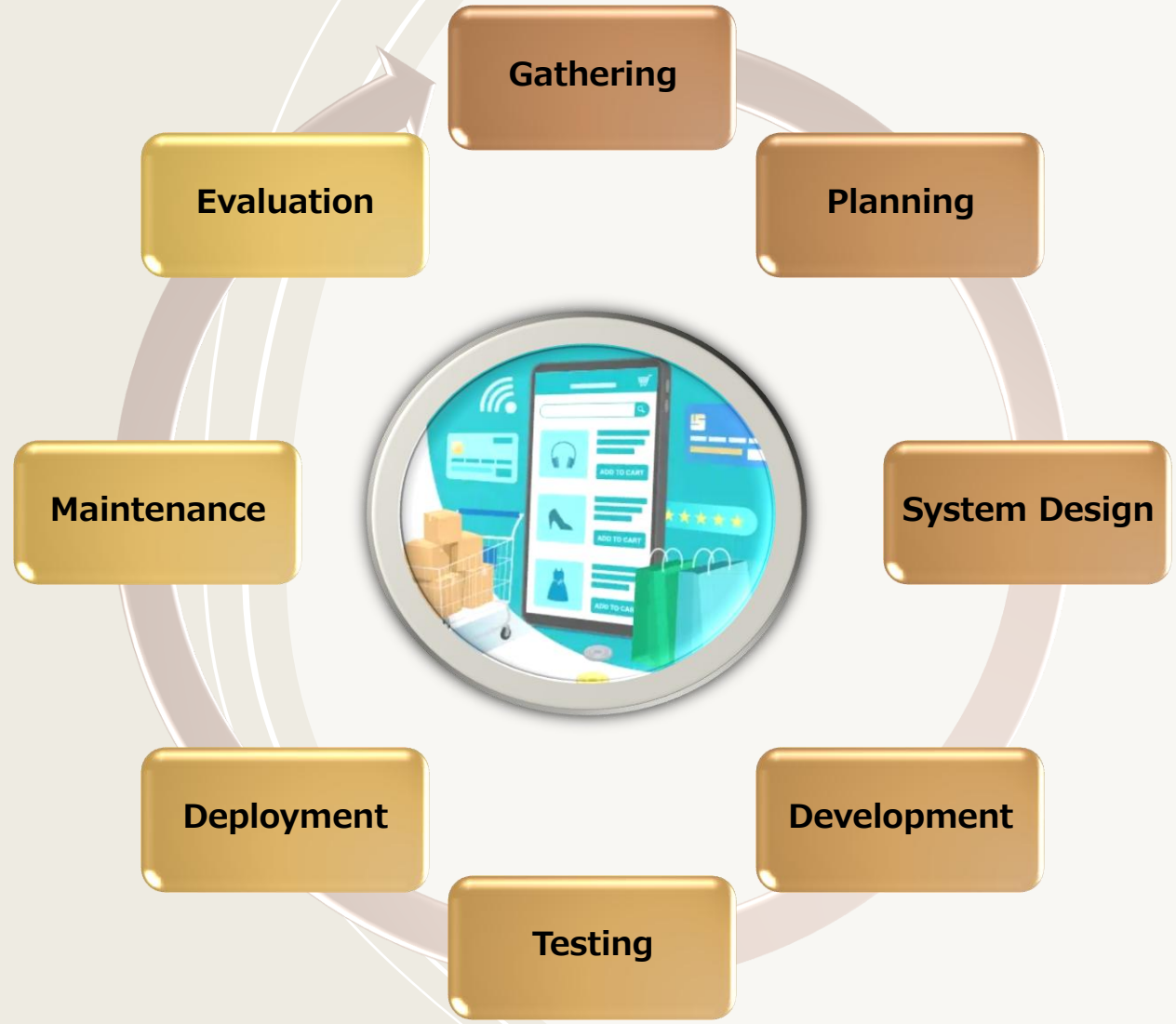
IV. Development

Front-End Development:

- Implement responsive UI using HTML, CSS, and JavaScript.
- Develop pages such as Login, Sign-Up, Product Listing, Product Details, Cart, and Checkout.

Back-End Development:

- Use Python Django to build the server-side logic and APIs.
- Integrate database models for users, products, and transactions.



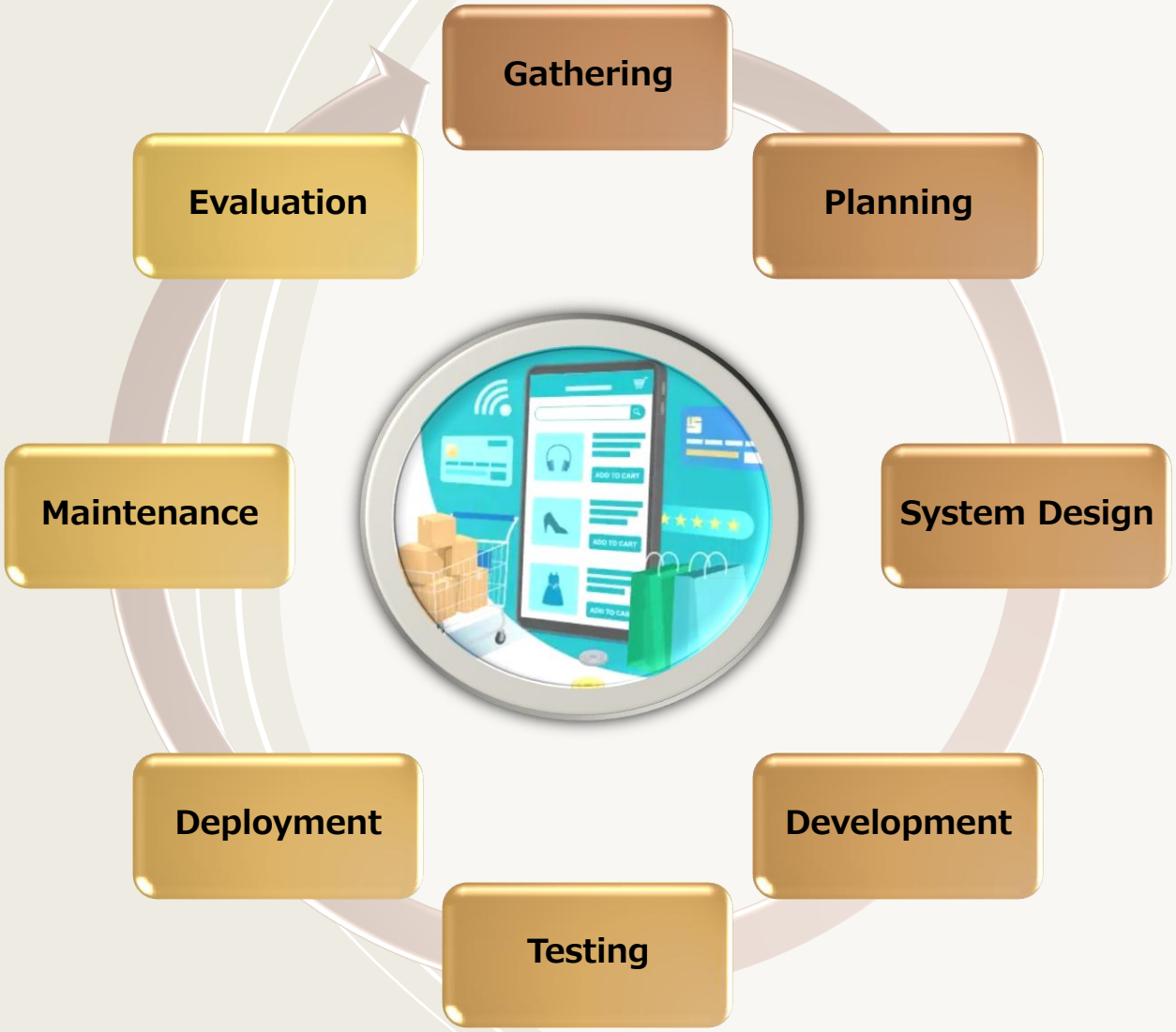
SDLC Model

V. Testing

- Perform unit testing for individual modules (e.g., cart functionality, user authentication).
- Conduct integration testing to ensure all components work together.
- Test for usability, responsiveness, and bug fixes.

VI. Deployment:

- Deploy the e-commerce website to a cloud platform or a local server.
- Ensure the website is accessible to end-users and performs well under varying traffic conditions.



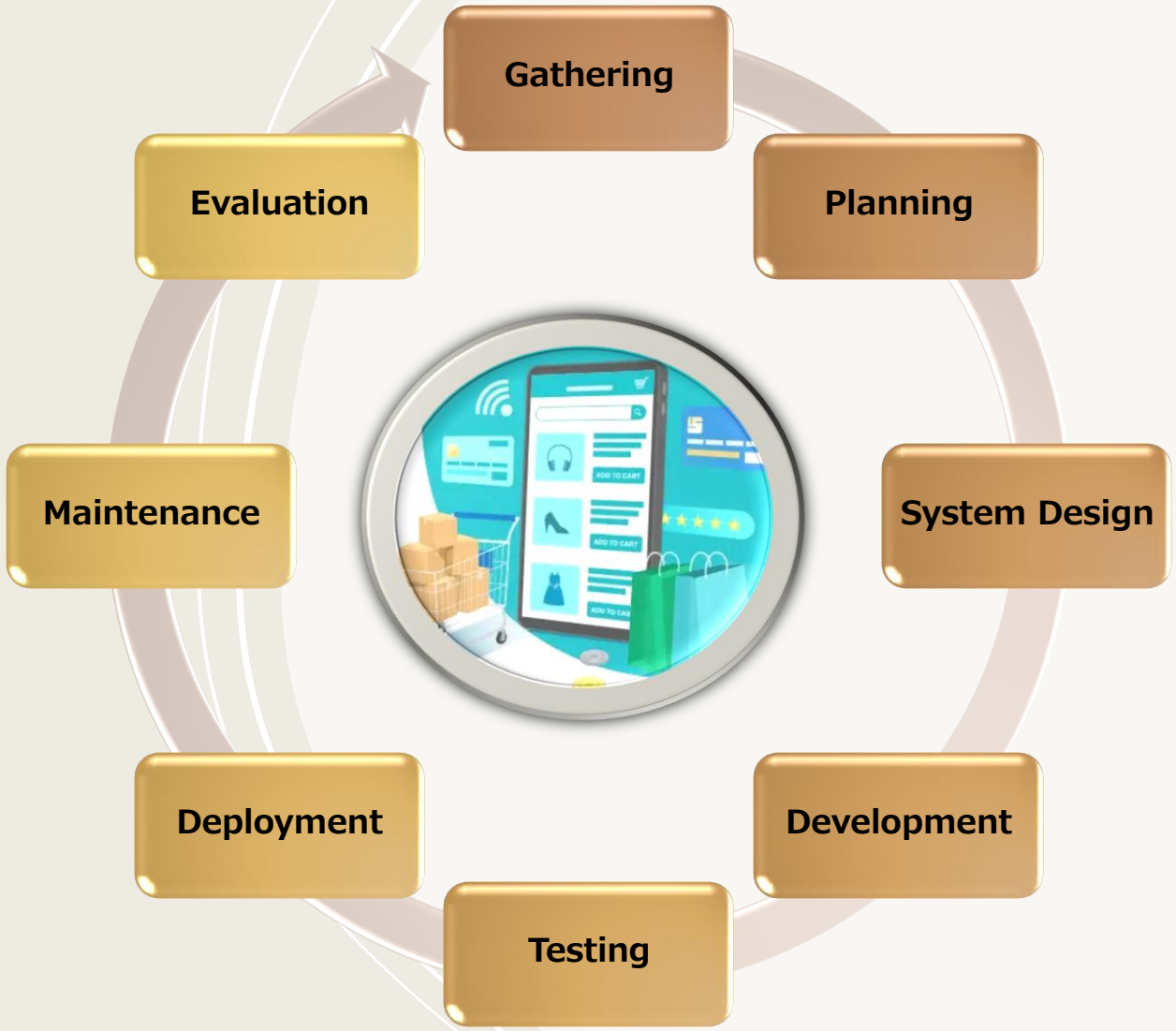
SDLC Model

VII. Maintenance

- Monitor website performance and fix any bugs reported by users.
- Update features based on user feedback, such as adding more payment methods or enhancing the UI.
- Implement regular security updates to protect user data.

VIII. Evaluation

- Gather user feedback post-deployment to evaluate the success of the project.
- Analyze the performance metrics and identify areas for improvement.
- Prepare for scaling the system to handle additional features or higher traffic.



Functional Requirements

- User Management
- Product Management
- Cart Management
- Order Management
- Checkout Process
- Session Management

The e-commerce website provides secure **User Management**, allowing users to register, log in, reset passwords, and for admins to manage products.

Product Management lets admins define product details like name, price, and stock, while users can browse and view product information.

In **Cart Management**, users can add, adjust, or remove items, with the cart automatically calculating the total price.

Order Management enables users to place orders, track details like prices and dates, and review their order history.

The **Checkout Process** is simple, with future plans for payment integration.

Finally, **Session Management** ensures secure, uninterrupted access until users log out or their session expires.

Non-Functional Requirements

The app need to have **fast performance**, with pages loading in 2-3 seconds and features like the cart updating in real time. It also needs to be **scalable**, so it can handle at least 1,000 users at the same time and support growth as the platform expands.

Security is critical, requiring user data, like passwords, to be encrypted, and the site must use HTTPS for secure communication. Role-based permissions will be used to control admin access. The system must also be reliable, with 99.9% uptime and quick recovery in case of issues.

We want the website to be easy to use, with a simple, mobile-friendly design that works on different devices and browsers (like Chrome, Safari, and Edge). **Accessibility** is also important, so we'll follow standards like WCAG, offering features like high-contrast mode and text resizing.

Finally, we'll make sure the code is **well-organized and documented** for future updates and maintenance.

Alternative Solution Analysis

Backend (Django Framework)

The backend is built using Django and Django REST Framework (DRF) to manage authentication, product management, user accounts, cart operations, order processing, and payment handling.

Frontend (HTML, CSS, JavaScript)

Provides an interactive user interface for customers to browse products, manage carts, and place orders.

The chosen stack balances:

- ✓ Scalability
- ✓ Security
- ✓ Ease of use

Alternative Solution Analysis
Backend Alternatives

	Pros:	Cons:
Flask (Python)	• Lightweight and minimalistic. • Ideal for smaller projects with fewer dependencies. • Easier to set up for custom APIs.	• Requires more manual setup for features like authentication. • Less out-of-the-box support for admin interfaces compared to Django.
Express.js (Node.js)	• Fast and efficient for real-time applications. • Large ecosystem of plugins and tools. • High performance with non-blocking I/O.	• Relies on third-party packages for features like authentication. • Requires more effort for building admin panels.

Alternative Solution Analysis
Frontend Alternatives

	Pros:	Cons:
React.js	• Component-based architecture allows reusable code. • Great for building dynamic and interactive UIs. • Strong community support.	• Steeper learning curve for beginners. • Requires state management libraries (e.g., Redux) for larger applications.
Vue.js	• Simpler syntax compared to React. • Combines the best features of Angular and React. • Easy to integrate into existing projects.	• Smaller community and fewer plugins than React. • May not be ideal for highly complex applications.

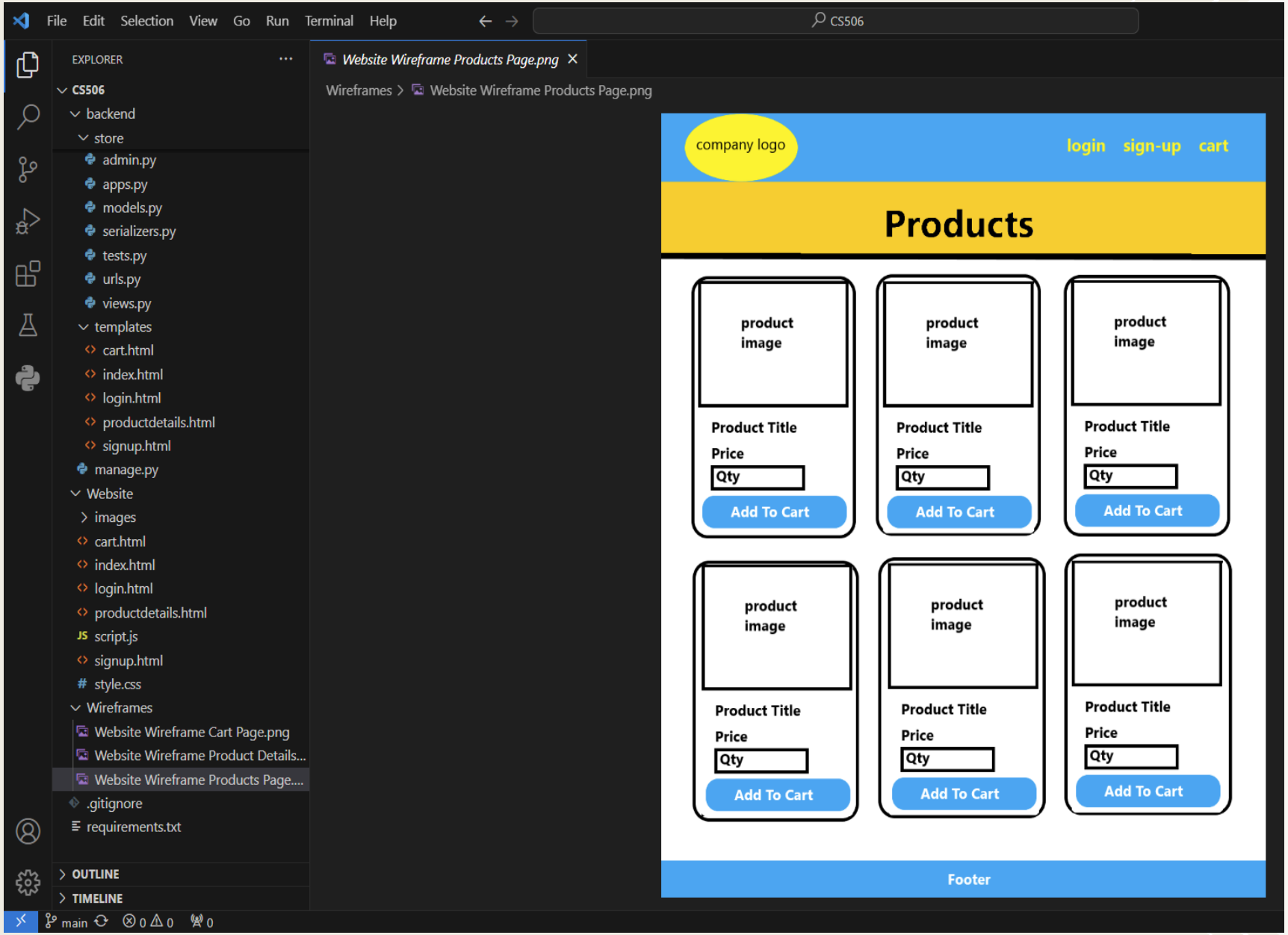
Alternative Solution Analysis
Database Alternatives

	Pros:	Cons:
PostgreSQL	• Advanced features like JSON support and full-text search. • Better handling of complex queries.	• Slightly more resource-intensive than MySQL.
MongoDB	• NoSQL database with high flexibility for unstructured data. • Scalability for large datasets.	• Not suitable for complex relational data. • Requires additional effort to enforce data consistency.

Alternative Solution Analysis
Authentication Alternatives

	Pros:	Cons:
OAuth2	• Standardized protocol with widespread adoption. • Easy integration with third-party login providers like Google or Facebook.	• Slightly more complex setup compared to JWT.
Session-Based Authentication	• Simple to implement with Django's default setup. • No token management required.	• Less suitable for API-based architectures. • Requires server-side storage for sessions.

Project Design



Example: Fig.2. Website Wireframe Products Page

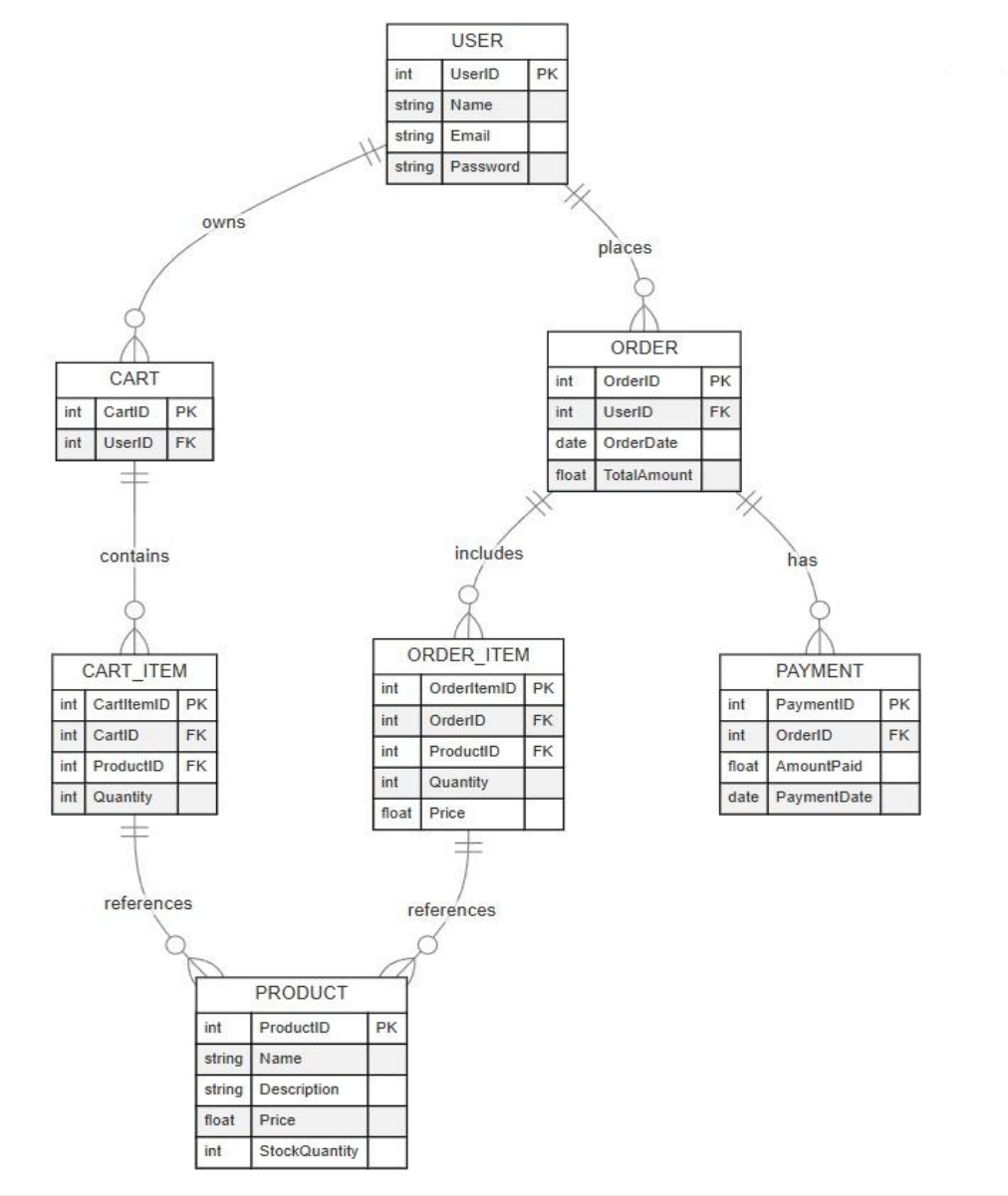
During the Project Design stage, we:

- **Develop Wireframes for Pages:**

Design the layout for the homepage, product details page, cart, and checkout page.

- **Draft a Sequence Diagram:**

Visualize the process for placing an order.



Example Fig1. Initial ER diagram

- **Created Use Case Scenarios**:

Example: "A customer browses products and adds items to their cart."
Example: "An admin updates stock for an existing product."

- **Designed an ER Diagram**:

Included entities like User, Product, Order, Cart, and the relationships between them.
Built a Use Case Diagram:

Showed the interactions between actors (Users, Admins) and the system.

- **Built a Use Case Diagram:**

Showed the interactions between actors (Users, Admins) and the system

Application Key Features:

- **Models**: CustomUser, Product, Cart, CartItem, Order, OrderItem, Category, Payment.
- **Views**: Provide RESTful APIs for registration, login, product listing/details, cart management, and order processing.
- **Serializers**: Used for data validation and serialization/deserialization.
- **JWT Authentication**: Secures the APIs with JSON Web Tokens (JWT).
- **Admin Interface**: Allows admin to manage all models in a user-friendly interface.

Development

Backend Development

The backend of the website is comprised of numerous serializers, views, and models. We do not have time to cover this full implementation as it was comprised of well over 400 lines of code.

Backend Development

This is an example of a serializer that allows complex data to be transitioned to native Python data types that are easily rendered on a page.

```python
class CartItemSerializer(serializers.ModelSerializer):
    product_name = serializers.CharField(source='product.title')
    product_price = serializers.DecimalField(source='product.price', max_digits=10, decimal_places=2)
    total_price = serializers.SerializerMethodField()
    image = serializers.ImageField(source='product.image', read_only=True)

    class Meta:
        model = CartItem
        fields = ['id', 'product', 'product_name', 'quantity', 'product_price', 'product_id', 'total_price', 'image']

    def get_total_price(self, obj):
        return obj.product.price * obj.quantity
```

```python
class CartAPIView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        # Get or create the cart for the logged-in user
        cart, created = Cart.objects.get_or_create(user=request.user)
        serializer = CartSerializer(cart)
        return Response(serializer.data)

    def post(self, request):
        # Add a product to the cart
        product_id = request.data.get('product_id')
        quantity = request.data.get('quantity', 1)

        try:
            product = Product.objects.get(id=product_id)
        except Product.DoesNotExist:
            return Response({"error": "Product not found"}, status=status.HTTP_404

        # Get or create the cart for the logged-in user
        cart, created = Cart.objects.get_or_create(user=request.user)

        # Check if the product already exists in the cart
        cart_item, created = CartItem.objects.get_or_create(cart=cart, product=product)

        # if not created:
        # If the product is already in the cart, update the quantity
        cart_item.quantity = quantity
        cart_item.save()

        return Response(CartSerializer(cart).data)
```

Backend Development

This is an example of a view to obtain data and post data to the database. 7 custom views were developed to manage the data transactions.

```python
class CustomUser(AbstractUser):
    phone_number = models.CharField(max_length=15, blank=True, null=True)
    address = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.username


class Product(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField()  # Number of items available
    image = models.ImageField(upload_to='products/')  # Product image
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title


class Cart(models.Model):
    user = models.OneToOneField('CustomUser', on_delete=models.CASCADE, related_name='cart')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Cart for {self.user.username}"


class CartItem(models.Model):
    cart = models.ForeignKey(Cart, related_name='cart_items', on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)

    def __str__(self):
        return f"{self.quantity} of {self.product.title}"
```

Backend Development

This is an example of a model that defines the database itself. When the program runs this database is created in MySQL automatically. This is how we generate the database in an AWS EC2 similar to a local machine.

Frontend Development

HTML pages are served using the Python Django render framework allowing us to dynamically render pages from the backend. The entire website is based on 4 HTML templates.

```
def signup(request):
  template = 'signup.html'
  return render(request, template)
```

```
# Define a view function for the login page
def login_page(request):
    return render(request, 'login.html')

# Define a view function for the login page
def product_details(request):
    return render(request, 'productdetails.html')
```

```
def view_cart(request):
    return render(request, 'cart.html')

def index(request):
  template = 'index.html'
  return render(request, template)
```

Frontend Development

Data is Posted and Pulled from the database using JavaScript to call the APIs. We use this setup whenever we need to get or send data to and from the database.

For GET requests, once the data is obtained from the database, we utilize the DOM to dynamically generate HTML content containing the data.

For POST request, we send data as JSON objects based on HTML data attributes and based on the acual values contained in form fields.

Frontend Development

GET and POST requests examples

```
// Fetch products from the backend and display them
fetch('/store/api/products/', {
    method: 'GET',
      headers: {
          'Content-Type': 'application/json',
          'Authorization': 'Bearer ' + localStorage.getItem('token'),
      },
      })
  .then(response => response.json())
  .then(products => {
    const productGrid = document.querySelector('.product-grid');
    cartCount=0
    productGrid.innerHTML = ''; // Clear any existing content

    products.forEach(product => {
      const productDiv = document.createElement('div');
      productDiv.classList.add('product');
      cartCount+=product.quantity;

      productDiv.innerHTML = `
        <div class="img-container" onclick="location.href='productdetails.html?id=${product.id}';">
          <img src="${product.image}" alt="${product.productName}">
        </div>
        <h3>${product.title}</h3>
        <p>Price: $${product.price}</p>
        <p>Quantity in stock: <span id="stock-${product.id}">${product.stock}</span></p>
        <div class="quantity-actions">
          <button class="decrement">-</button>
          <input class="quantity" type="number" value="1" min="1" id="quantity-${product.id}">
          <button class="increment">+</button>
        </div>
        <button class="add-to-cart" data-id="${product.id}">Add To Cart</button>
      `;

      productGrid.appendChild(productDiv);
    });
```

```
// Manage the signup form information
const signUpForm = document.querySelector('.sign-up');

// If a signup form class is found add an event listener to the form
if (signUpForm) {
  document.querySelector('form').addEventListener('submit', (event) => {
    event.preventDefault(); // Prevent default form submission

    const formData = new FormData(event.target);
    const data = Object.fromEntries(formData.entries());

    fetch('/store/api/register/', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    })
    .then(response => response.json())
    .then(response => {
      if (response.success) {
        // Failed registration
        const modal = document.getElementById('modal');
        const modalMessage = document.getElementById('modal-message');
        modalMessage.textContent = 'Registration failed. Please check the form and try again.';
        modal.style.display = 'block';
      } else {
        // Successful registration
        const modal = document.getElementById('modal');
        const modalMessage = document.getElementById('modal-message');
        modalMessage.textContent = 'Registration successful! You can now log in.';
        modal.style.display = 'block';
      }
    })
```

Frontend Development

For security purposes, most APIs require an authorization token to access them. For demonstration purposes, this token is stored in local storage when the user logs in and times out after a given number of hours.

Frontend Development: User Authentication

View to obtain key pair:

```python
class MyTokenObtainPairView(TokenObtainPairView):
    def post(self, request, *args, **kwargs):
        username = request.data.get('username')
        password = request.data.get('password')


        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return super().post(request, *args, **kwargs)
        else:
            return Response({'detail': 'Invalid credentials'}, status=status.HTTP_401_UNAUTHORIZED)
```

JavaScript to store token:

```javascript
fetch('/store/login/', {
method: 'POST',
headers: {
    'Content-Type': 'application/json'
},
body: JSON.stringify(data)
})
.then(response => response.json())
.then(response => {
localStorage.setItem('token', response.access);
localStorage.setItem('username', document.getElementById("username-field").value);
window.location.href = "/store/index.html";
```

A call to an API using the token:

```javascript
fetch('/store/api/cart/', {
  method: 'GET',
  headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + localStorage.getItem('token'),
  },
  })
```

Frontend Development

Whenever we need to reference product ids such as when we need to add them to the cart or when we want to display a product details page, we reference the product id as either an HTML data attribute connected to a button, or we pass the product id to the URL that is connected to an API to display product details.

Product IDs to reference specific products in the HTML code:

```
.then(products => {
  const productGrid = document.querySelector('.product-grid');
  cartCount=0
  productGrid.innerHTML = ''; // Clear any existing content

  products.forEach(product => {
    const productDiv = document.createElement('div');
    productDiv.classList.add('product');
    cartCount+=product.quantity;

    productDiv.innerHTML = `
      <div class="img-container" onclick="location.href='productdetails.html?id=${product.id}';">
        <img src="${product.image}" alt="${product.productName}">
      </div>
      <h3>${product.title}</h3>
      <p>Price: $${product.price}</p>
      <p>Quantity in stock: <span id="stock-${product.id}">${product.stock}</span></p>
      <div class="quantity-actions">
        <button class="decrement">-</button>
        <input class="quantity" type="number" value="1" min="1" id="quantity-${product.id}">
        <button class="increment">+</button>
      </div>
      <button class="add-to-cart" data-id="${product.id}">Add To Cart</button>
    `;
```

Frontend Development

Continued on next slide

The URL to navigate to dynamically generated product pages based on product id.

```
path('api/product/<int:id>/', ProductDetailAPIView.as_view(), name='product_detail'),
```

```
function initializeAddToCartActions() {
  const addToCartButtons = document.querySelectorAll('.add-to-cart');

  addToCartButtons.forEach(button => {
    button.addEventListener('click', () => {
      const productId = button.getAttribute('data-id');
      const quantityInput = parseInt(button.parentElement.querySelector('.quantity').value);
      console.log(quantityInput);
      //const newQuantity = parseInt(quantityInput.value);

      /*
      if (isNaN(newQuantity) || newQuantity < 1) {
        console.error('Invalid quantity');
        return;
      }
        */

      const data = {
        product_id: productId,
        quantity: quantityInput
      };
      // Send the updated quantity to the server
      fetch(`/store/api/cart/`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': 'Bearer ' + localStorage.getItem('token'),
        },
        body: JSON.stringify(data)
      })
      .then(response => response.json())
      .then(updatedProduct => {
        console.log('Product updated:', updatedProduct);
        //const stockDisplay = document.getElementById(`stock-${productId}`);
        //stockDisplay.textContent = updatedProduct.quantity; // Update the displayed stock quantity
        //location.reload();
      })
      .catch(error => {
        console.error('Error updating product:', error);
      });
    });
  });
```

Frontend Development

The function to reference the data attributes stored in the HTML button elements so the correct item is always added to the cart.

Development Questions?

Due to the amount of code and the complexity of the project, I could not possibly explain all of the details in a few PowerPoint slides. This was only a brief overview of the main backbone functions of the website.

If you have any questions about the details of the development or the design of this project, please reach out for a more detailed explanation.

Integration and testing

Testing Process in E-Commerce Project

Objective: Ensure the application meets functional and non-functional requirements.

Testing Methods Used:

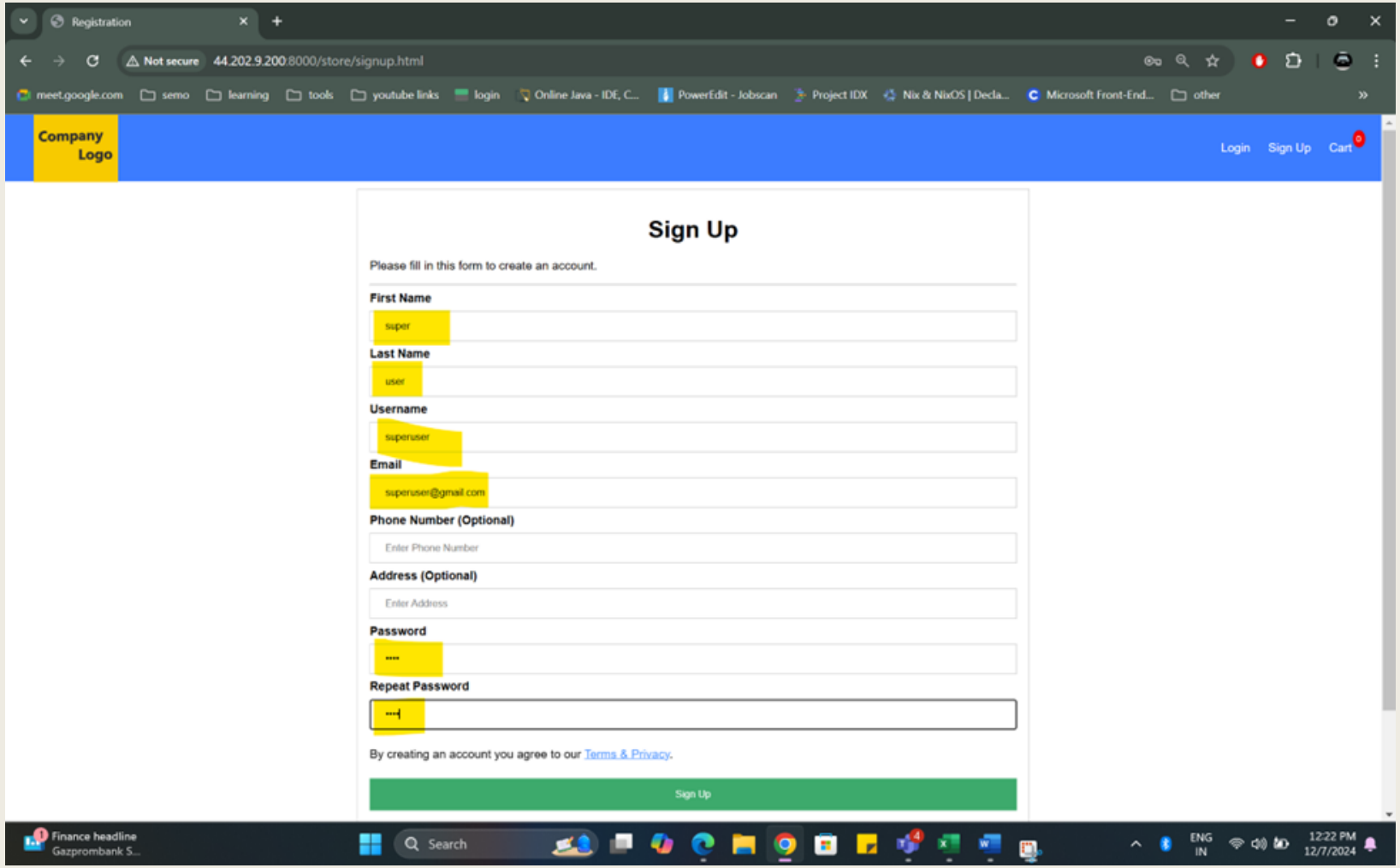
Unit Testing	Tested individual components like product addition, cart updates, and user login.
Integration Testing	Validated API responses with frontend functionality to ensure seamless communication between the frontend, backend, and database
System Testing	Simulated complete user workflows, such as browsing products, adding to cart, and checking out.
Performance Testing	Evaluated system response times under different loads to identify bottlenecks.
Tools Used	Browser Developer Tools, Postman, and Selenium.

Example of Test Case Report

Title: Test Case: Add Product to Cart

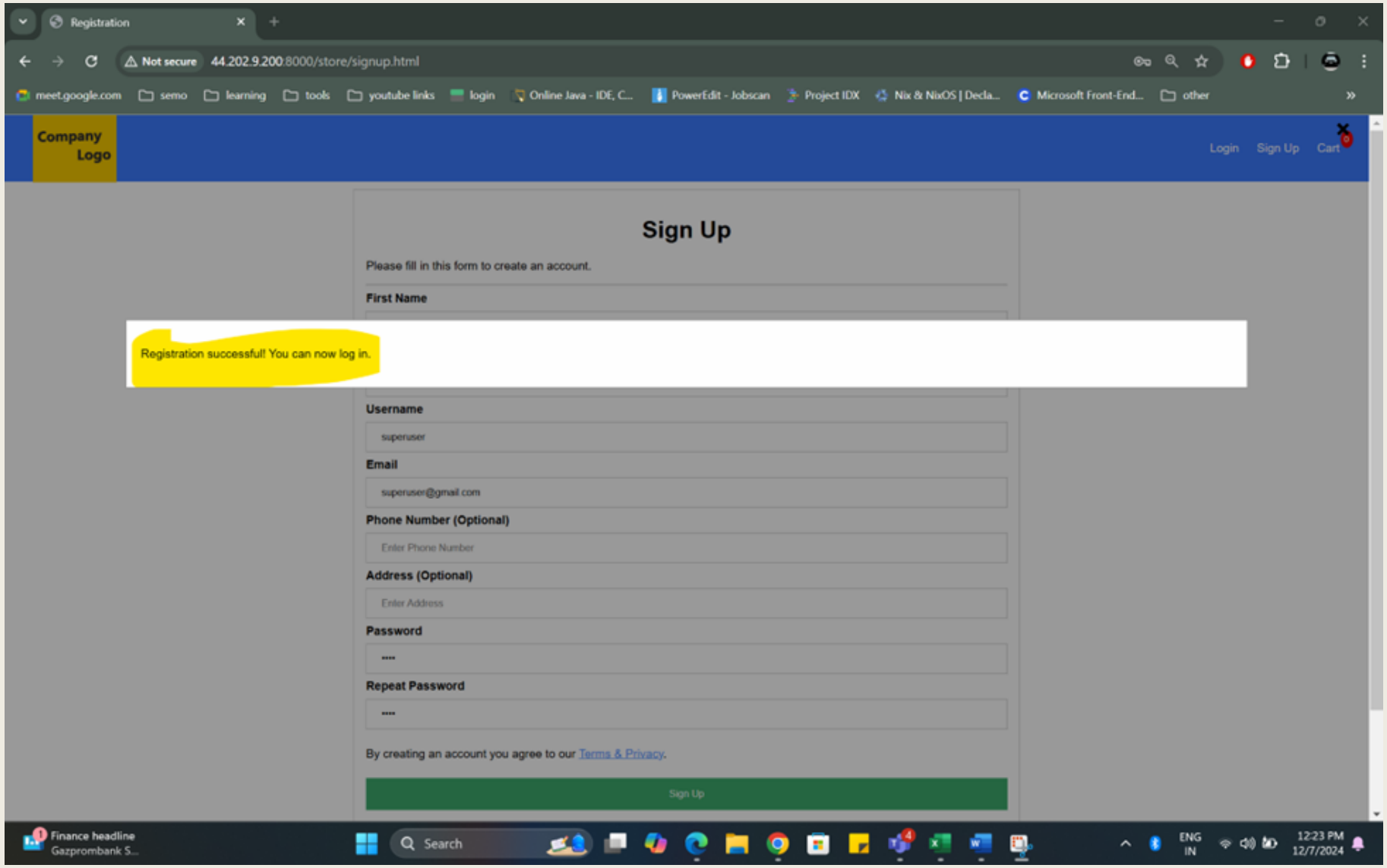
Test Case ID	TC001 - Add Product to Cart
Precondition	User must be logged in and browsing the product catalog
Description	Verify that a user can add a product to their cart successfully
Test Steps	1. Navigate to the product catalog. 2. Select a product and click "Add to Cart." 3. View the cart.
Expected Output	Product is displayed in the cart with the correct quantity and total price updated.
Actual Output	Product added successfully; total price updated.
Status	Pass

Screenshots:

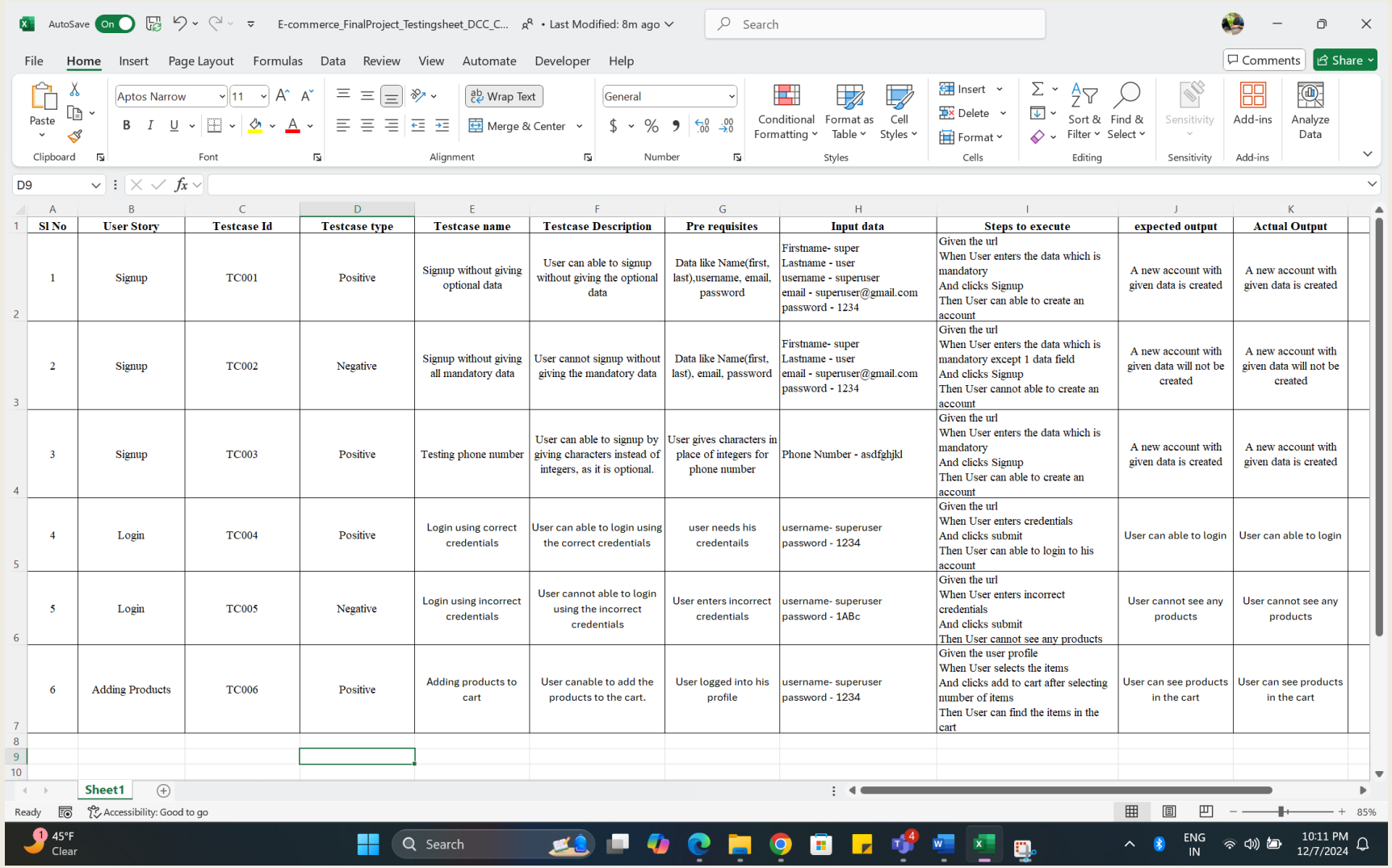


Here we have given all the mandatory details and trying to sign up. We need to sign up (create account)

As per our expected result, we can create an account. So, the testcase got passed.



Screenshot of scenarios and testcases



Sl No	User Story	Testcase Id	Testcase type	Testcase name	Testcase Description	Pre requisites	Input data	Steps to execute	expected output	Actual Output
1	Signup	TC001	Positive	Signup without giving optional data	User can able to signup without giving the optional data	Data like Name(first, last),username, email, password	Firstname- super Lastname - user username - superuser email - superuser@gmail.com password - 1234	Given the url When User enters the data which is mandatory And clicks Signup Then User can able to create an account	A new account with given data is created	A new account with given data is created
2	Signup	TC002	Negative	Signup without giving all mandatory data	User cannot signup without giving the mandatory data	Data like Name(first, last), email, password	Firstname- super Lastname - user email - superuser@gmail.com password - 1234	Given the url When User enters the data which is mandatory except 1 data field And clicks Signup Then User cannot able to create an account	A new account with given data will not be created	A new account with given data will not be created
3	Signup	TC003	Positive	Testing phone number	User can able to signup by giving characters instead of integers, as it is optional.	User gives characters in place of integers for phone number	Phone Number - asdfghjkl	Given the url When User enters the data which is mandatory And clicks Signup Then User can able to create an account	A new account with given data is created	A new account with given data is created
4	Login	TC004	Positive	Login using correct credentials	User can able to login using the correct credentials	user needs his credentails	username- superuser password - 1234	Given the url When User enters credentials And clicks submit Then User can able to login to his account	User can able to login	User can able to login
5	Login	TC005	Negative	Login using incorrect credentials	User cannot able to login using the incorrect credentials	User enters incorrect credentials	username- superuser password - 1ABc	Given the url When User enters incorrect credentials And clicks submit Then User cannot see any products	User cannot see any products	User cannot see any products
6	Adding Products	TC006	Positive	Adding products to cart	User canable to add the products to the cart.	User logged into his profile	username- superuser password - 1234	Given the user profile When User selects the items And clicks add to cart after selecting number of items Then User can find the items in the cart	User can see products in the cart	User can see products in the cart

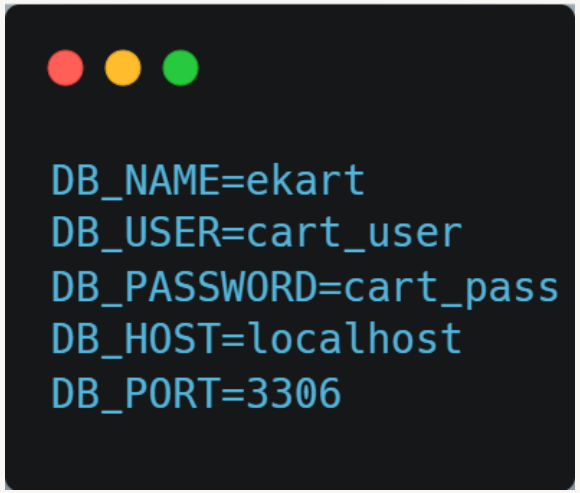
Implementation

Setting Up the EC2 Instance

- Deployed an EC2 instance running Ubuntu 20.04.
- Configured security group rules - Opened ports: 22 (SSH), 80 (HTTP), and 443 (HTTPS).
- Connected to the instance via SSH to begin setup.
- Cloned the project repository to the EC2 instance

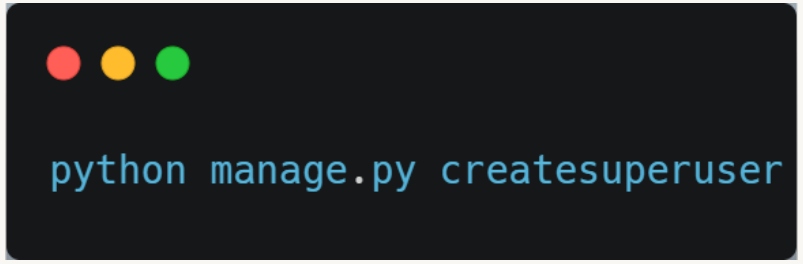
Configuring the Django Environment

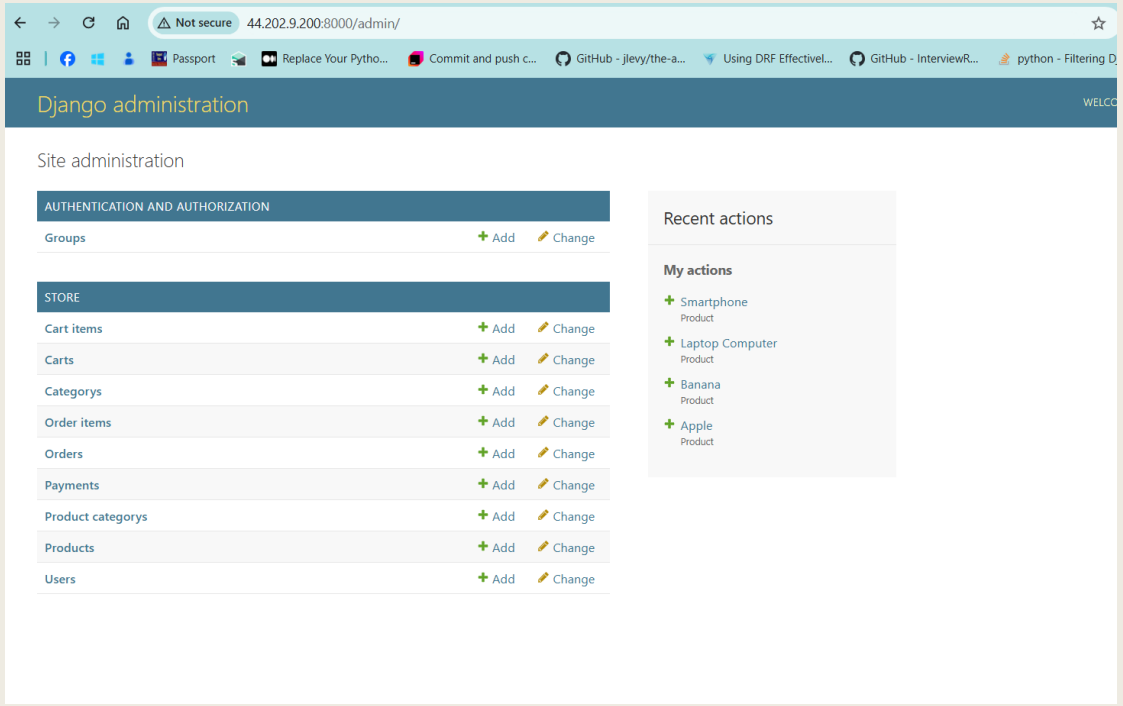
- Created an `.env` file for environment-specific settings:
- Installed project dependencies using `pip`
- Migrated the database schema

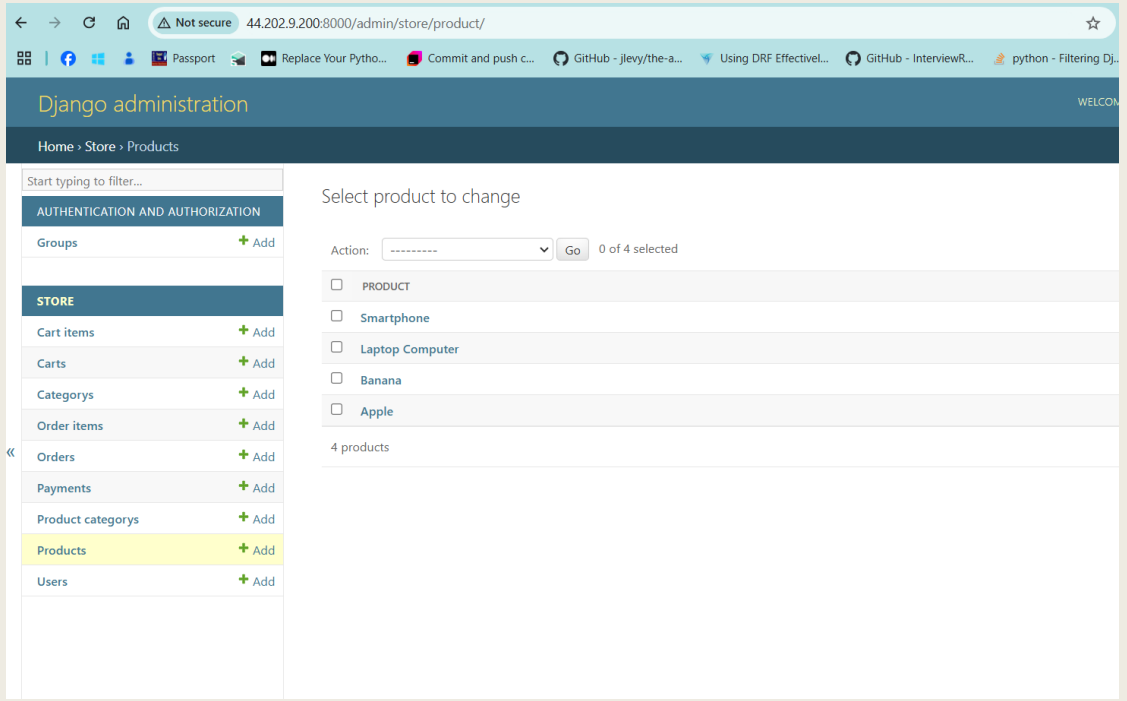


Creating a Superuser and Populating the Database

- Created a superuser for admin access
- Logged into the Django admin panel at `http://44.202.200:8000/admin` to populate the database




Not secure
44.202.9.200:8000/admin/
Passport
Replace Your Pytho...
Commit and push c...
GitHub - jlevy/the-a...
Using DRF Effectivel...
GitHub - InterviewR...
python - Filtering D
Django administration
WELCO
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
STORE
Cart items + Add Change
Carts + Add Change
Categorys + Add Change
Order items + Add Change
Orders + Add Change
Payments + Add Change
Product categorys + Add Change
Products + Add Change
Users + Add Change
Recent actions
My actions
+ Smartphone
Product
+ Laptop Computer
Product
+ Banana
Product
+ Apple
Product


Not secure
44.202.9.200:8000/admin/store/product/
Passport
Replace Your Pytho...
Commit and push c...
GitHub - jlevy/the-a...
Using DRF Effectivel...
GitHub - InterviewR...
python - Filtering Dj...
Django administration
WELCOM
Home › Store › Products
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
STORE
Cart items + Add
Carts + Add
Categorys + Add
Order items + Add
Orders + Add
Payments + Add
Product categorys + Add
Products + Add
Users + Add
Select product to change
Action: --------- Go 0 of 4 selected
PRODUCT
Smartphone
Laptop Computer
Banana
Apple
4 products

Configuring the Database

- Installed and secured MySQL
- Configured the MySQL database
- Verified the database connection using the `.env` file and updated Django settings

```
sudo apt update
sudo apt install mysql-server
sudo mysql_secure_installation
```

```
CREATE DATABASE ekart;
CREATE USER 'cart_user'@'localhost' IDENTIFIED BY 'cart_pass';
GRANT ALL PRIVILEGES ON ekart.* TO 'cart_user'@'localhost';
FLUSH PRIVILEGES;
```

```
[Unit]
Description=gunicorn daemon for Django project
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/project/CS506/backend
ExecStart=/home/ubuntu/project/CS506/venv/bin/gunicorn --workers 3 --bind
unix:/home/ubuntu/project/CS506/backend/gunicorn.sock ekart.wsgi:application

[Install]
WantedBy=multi-user.target
```

Setting Up Gunicorn

- Gunicorn systemd service file
- Started Gunicorn service

```
sudo systemctl start gunicorn
sudo systemctl enable gunicorn
```

Setting Up NGINX

- NGINX configuration file
- Restarted NGINX

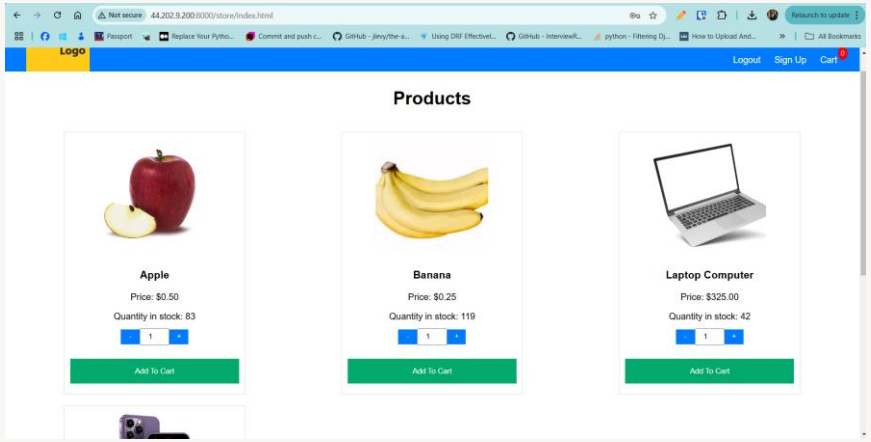
```
server {
    listen 80;
    server_name 44.202.9.200;

    # Serve static files
    location /static/ {
        alias /home/ubuntu/project/CS506/backend/staticfiles/;
    }

    # Serve media files
    location /media/ {
        alias /home/ubuntu/project/CS506/backend/media/;
    }

    # Proxy requests to Gunicorn
    location / {
        proxy_pass http://unix:/home/ubuntu/project/CS506/backend/gunicorn.sock;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    error_log /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
}
```

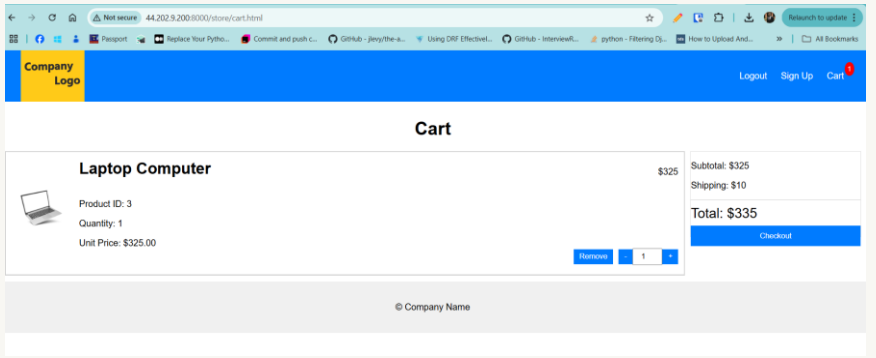
```
sudo systemctl restart nginx
```

Testing and Finalization

Verified the deployment:

- Accessed the application via http://44.202.9.200.
- Tested core features: user authentication, product browsing, cart management, and order placement.


Not secure
44.202.9.200:8000/store/index.html
Logo
Logout
Sign Up
Cart
0
Products
Apple
Price: $0.50
Quantity in stock: 83
1
Add To Cart
Banana
Price: $0.25
Quantity in stock: 119
1
Add To Cart
Laptop Computer
Price: $325.00
Quantity in stock: 42
1
Add To Cart


Not secure
44.202.9.200:8000/store/cart.html
Company Logo
Logout
Sign Up
Cart
1
Cart
Laptop Computer
$325
Product ID: 3
Quantity: 1
Unit Price: $325.00
Remove
1
Subtotal: $325
Shipping: $10
Total: $335
Checkout
© Company Name

Code and Webpage Screenshots

This section includes some random screenshots of the code and it shows some of the various webpages of the site. We wanted to include this section to display some of the tools, code, and pages that helped to make and make up this site.

If you would like to visit our site it is available at this URL, http://44.202.9.200:8000/store/index.html

Dashboard

canvas.semo.edu

File Edit Selection View Go Run Terminal Help

CS506

EXPLORER

CS506

- .venv
- backend
  - ekart
  - media
  - staticfiles
    - images
    - cart.js M
    - login.js
    - script_index.js M
    - script_productdetails.js
    - script.js M
    - style.css
  - store
  - templates
  - manage.py
- Website
- Wireframes
- .env
- .gitignore
- requirements.txt

OUTLINE

TIMELINE

script.js M

backend > staticfiles > script.js > addEventListener('submit') callback

```
135
136  // Update the cart quantity span element in the nav section
137  function displayCartQuantity() {
138    fetch('/store/api/cart/', {
139      method: 'GET',
140      headers: {
141          'Content-Type': 'application/json',
142          'Authorization': 'Bearer ' + localStorage.getItem('token'),
143      },
144      })
145      .then(response => response.json())
146      .then(data => {
147        let totalItems = 0;
148        data.cart_items.forEach(item => {
149          totalItems += item.quantity;
150        });
151        document.getElementById('cart-count').textContent = totalItems;
152      })
153      .catch(error => {
154        console.error('Error fetching cart data:', error);
155      });
156    }
157    displayCartQuantity();
158
159
160
161
162  // Manage the signup form information
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

PS C:\Users\kyboo\Desktop\Kyle College\Fall 2024\CS506\Group Project\CS506>

main* 0 0 0 Ln 167, Col 73 Spaces: 2 UTF-8 CRLF JavaScript Go Live

53°F Light rain

Search

7:40 PM 12/8/2024

Dashboard

canvas.semo.edu

File Edit Selection View Go Run Terminal Help

CS506

EXPLORER

CS506

- .venv
- backend
  - ekart
  - media
  - staticfiles
  - store
    - __pycache__
    - migrations
    - __init__.py
    - admin.py
    - apps.py
    - models.py
    - serializers.py M
    - tests.py
    - urls.py
    - views.py
  - templates
  - manage.py
- Website
- Wireframes
- .env
- .gitignore
- requirements.txt

OUTLINE

TIMELINE

views.py

backend > store > views.py > view_cart

```python

class PlaceOrderAPIView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request):
        # Get the user's cart
        cart, created = Cart.objects.get_or_create(user=request.user)

        if not cart.cart_items.exists():
            return Response({"error": "Cart is empty"}, status=status.HTTP_400_BAD_REQUEST)

        # Create the order
        order = Order.objects.create(user=request.user, total_amount=0)

        # Add cart items to the order
        for cart_item in cart.cart_items.all():
            order_item = OrderItem.objects.create(
                order=order,
                product=cart_item.product,
                quantity=cart_item.quantity,
                price=cart_item.product.price
            )
            remaining_stock = cart_item.product.stock - cart_item.quantity
            final_stock = remaining_stock if remaining_stock > 0 else 0
            cart_item.product.stock = final_stock
            cart_item.product.save()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

PS C:\Users\kyboo\Desktop\Kyle College\Fall 2024\CS506\Group Project\CS506>

main* 0 0 0 Ln 190, Col 8 Spaces: 4 UTF-8 CRLF Python 3.12.5 ('.venv': venv) Go Live

53°F Light rain

Search

7:35 PM 12/8/2024

Dashboard

canvas.semo.edu

File Edit Selection View Go Run Terminal Help

CS506

EXPLORER

CS506

- .venv
- backend
  - ekart
  - media
  - staticfiles
  - store
  - templates
    - cart.html
    - index.html
    - login.html
    - productdetails.html
    - signup.html
  - manage.py
  - Website
  - Wireframes
- .env
- .gitignore
- requirements.txt

OUTLINE

TIMELINE

index.html

backend > templates > index.html > html > body > script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Products</title>
    {% load static %}
    <link type="text/css" rel="stylesheet" href="{% static 'style.css' %}">
</head>
<body>
    <header>
        <div class="logo">
            <a href="index.html"><img src="{% static 'images/company-logo.png' %}" alt="Company Logo"></a>
        </div>
        <nav>
            <!--{% if user.is_authenticated %}-->
                <!--<p>Welcome, {{ user.username }}!</p>-->
            <!--{% endif %}-->
            <p id="user-greeting"></p>
            <a href="login.html" id="nav-login-btn">Login</a>
            <a href="signup.html">Sign Up</a>
            <a href="cart.html">Cart<span id="cart-count">0</span></a>
        </nav>
    </header>

    <main>
        <h1>Products</h1>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

PS C:\Users\kyboo\Desktop\Kyle College\Fall 2024\CS506\Group Project\CS506>

main* 0 0 0 Ln 54, Col 41 Spaces: 4 UTF-8 CRLF HTML Go Live

Account Dashboard Courses Groups Calendar Inbox History SupportNET Studio

53°F Light rain

Search

7:37 PM 12/8/2024

Dashboard

canvas.semo.edu

File Edit Selection View Go Run Terminal Help CS506

EXPLORER

CS506

- .venv
- backend
  - ekart
  - media
  - staticfiles
    - images
    - cart.js M
    - login.js
    - script_index.js M
    - script_productdetails.js
    - script.js M
    - style.css
  - store
  - templates
  - manage.py
- Website
- Wireframes
- .env
- .gitignore
- requirements.txt

OUTLINE

TIMELINE

style.css

backend > staticfiles > # style.css > .price

```
164
165  .cart-items {
166      display: flex;
167      flex-direction: column;
168      gap: 0px;
169      width: 80%;
170      border: 2px solid lightgray;
171      margin-right: 10px;
172  }
173
174  .cart-item {
175      display: flex;
176      justify-content: space-between;
177      align-items: center;
178      border-bottom: 1px solid #ddd;
179      padding: 10px;
180      position: relative;
181      padding-bottom: 30px;
182  }
183
184  .product-info {
185      display: flex;
186      align-items: center;
187  }
188
189  .product-info img {
190      width: 100px;
191      margin-right: 20px;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

PS C:\Users\kyboo\Desktop\Kyle College\Fall 2024\CS506\Group Project\CS506>

main* 0 0 0 Ln 219, Col 3 Spaces: 4 UTF-8 CRLF CSS Go Live

53°F Light rain

Search

7:38 PM 12/8/2024

Dashboard

canvas.semo.edu

Products

Not secure | 44.202.9.200:8000/store/index.html

Company Logo

Welcome, KySchwartz!

Logout Sign Up Cart 0

Products

Apple

Price: $0.50

Quantity in stock: 83

- 1 +

Add To Cart

Banana

Price: $0.25

Quantity in stock: 119

- 1 +

Add To Cart

Laptop Computer

Price: $325.00

Quantity in stock: 42

- 1 +

Add To Cart

Account

Dashboard

Courses

Groups

Calendar

Inbox 15

History

SupportNET

Studio

53°F Light rain

Search

7:41 PM 12/8/2024

Dashboard

canvas.semo.edu

Products

Not secure | 44.202.9.200:8000/store/index.html

Company Logo

Welcome, KySchwartz!

Logout Sign Up Cart 0

Products

Apple

Price: $0.50

Quantity in stock: 83

- 1 +

Add To Cart

Welcome Elements Console Sources Network Performance Memory Application

```
<!DOCTYPE html>
<html lang="en"> scroll
 <head> ... </head>
 <body>
  <header> ... </header> flex
  <main>
   <h1>Products</h1>
   <section class="product-grid"> ... </section> grid == $0
  </main>
  <footer> ... </footer>
  <script src="/static/script.js"></script>
  <script src="/static/script_index.js"></script>
 </body>
</html>
```

html body main section.product-grid

Console Issues

Styles Computed Layout Event Listeners

Filter :hov .cls

```
element.style {
}
@media (max-width: 768px) {                    style.css:154
  .product-grid {
    grid-template-columns: 1fr;
  }
}
.product-grid {                                style.css:56
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
  padding: 20px;
  justify-items: center;
}
section {                                      user agent stylesheet
  display: block;
  unicode-bidi: isolate;
}
Inherited from body
body {                                         style.css:1
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
}
```

margin – border – padding 20

Account Dashboard Courses Groups Calendar Inbox 15 History SupportNET Studio

53°F Light rain

Search

7:42 PM 12/8/2024

Dashboard

canvas.semo.edu

Cart

Not secure | 44.202.9.200:8000/store/cart.html

Company Logo

Logout Sign Up Cart 7

Cart

Apple

$1

Product ID: 1

Quantity: 2

Unit Price: $0.50

Remove - 2 +

Banana

$0.75

Product ID: 2

Quantity: 3

Unit Price: $0.25

Remove - 3 +

Laptop Computer

$325

Product ID: 3

Quantity: 1

Unit Price: $325.00

Remove - 1 +

Subtotal: $511.75

Shipping: $10

Total: $521.75

Checkout

Account
Dashboard
Courses
Groups
Calendar
Inbox 15
History
SupportNET
Studio

53°F Light rain

Search

7:43 PM 12/8/2024

Dashboard

canvas.semo.edu

Cart

Not secure | 44.202.9.200:8000/store/productdetails.html?id=1

Company Logo

Logout Sign Up Cart 7

Apple

Product ID: 1

A crisp, juicy, and sweet treat, packed with vitamins and antioxidants. Perfect for a healthy snack or a delicious addition to your favorite recipes.

Price: $0.50

Quantity in stock: 83

- 1 +

Add To Cart

© Company Name

Account
Dashboard
Courses
Groups
Calendar
Inbox 15
History
SupportNET
Studio

53°F Light rain

Search

7:44 PM 12/8/2024

Dashboard

canvas.semo.edu

Select product to change | Djan

Not secure 44.202.9.200:8000/admin/store/product/

Account

Dashboard

Courses

Groups

Calendar

Inbox 15

History

SupportNET

Studio

Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Store › Products

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups + Add

STORE

- Cart items + Add
- Carts + Add
- Categorys + Add
- Order items + Add
- Orders + Add
- Payments + Add
- Product categorys + Add
- Products + Add
- Users + Add

Select product to change

ADD PRODUCT +

Action: --------- Go 0 of 4 selected

PRODUCT
Smartphone
Laptop Computer
Banana
Apple

4 products

53°F Light rain

Search

7:45 PM 12/8/2024

Dashboard

canvas.semo.edu

Add product | Django site admi

Not secure 44.202.9.200:8000/admin/store/product/add/

Account

Dashboard

Courses

Groups

Calendar

Inbox 15

History

SupportNET

Studio

Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Store › Products › Add product

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups + Add

STORE

- Cart items + Add
- Carts + Add
- Categorys + Add
- Order items + Add
- Orders + Add
- Payments + Add
- Product categorys + Add
- Products + Add
- Users + Add

Add product

Title:

Description:

Price:

Stock:

Image: Choose File No file chosen

SAVE | Save and add another | Save and continue editing

53°F Light rain

Search

7:45 PM 12/8/2024

Evaluation and Future Work

The final product of our project, the E-Commerce Application eKart, is a robust and scalable platform designed to simplify online shopping for users and support small businesses. The platform incorporates key features that provide a seamless shopping experience while ensuring user data security

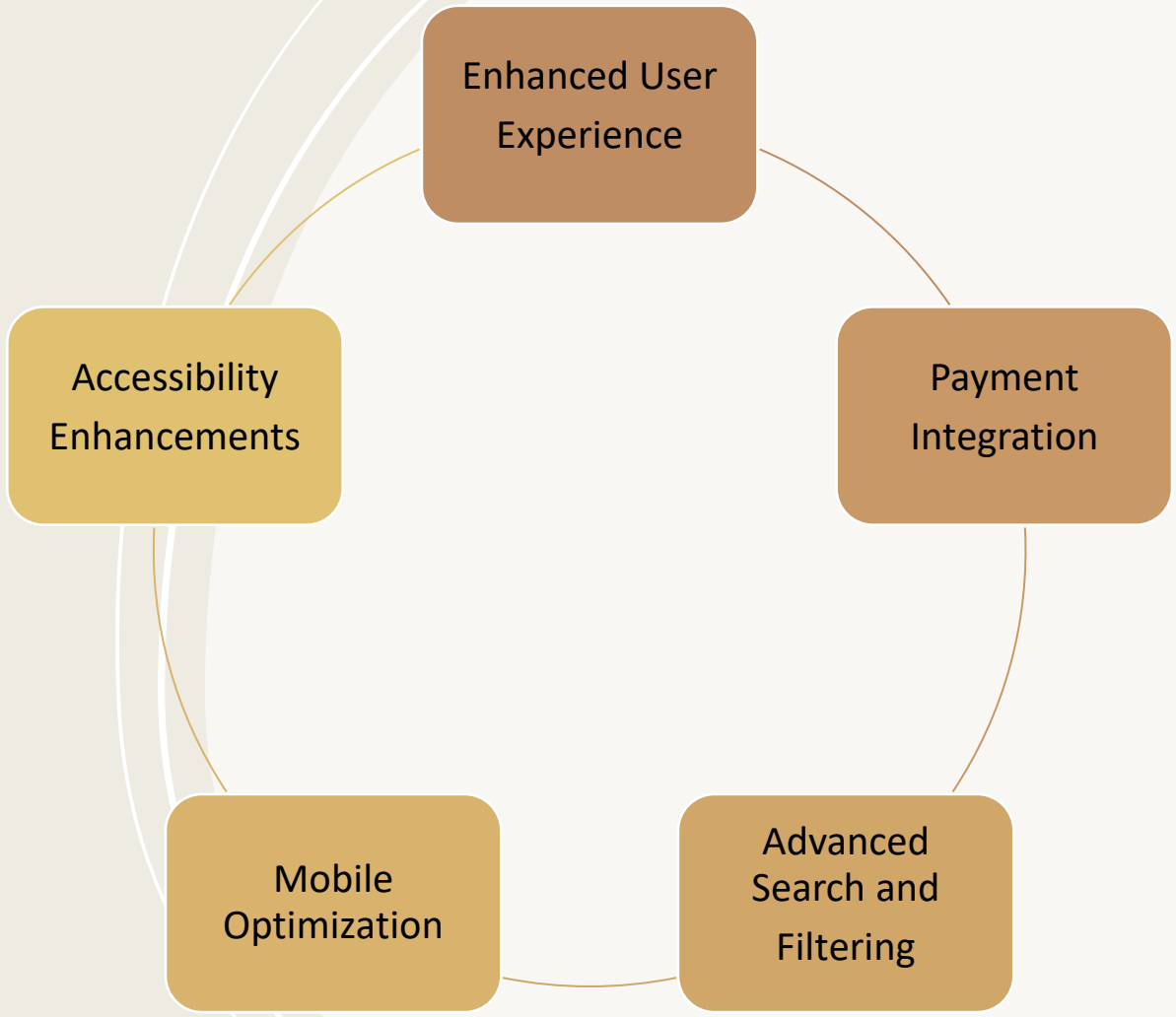
Key Functionalities Developed:

- Users can register, log in, and securely manage their accounts.
- Admins can add, update, and delete products, including managing stock and pricing.
- A fully functional cart allows users to add, update, and remove items, with total prices calculated dynamically.
- Users can place and track orders.
- The platform supports responsive web design, ensuring compatibility across devices.
- The backend is secured with JWT-based authentication, providing role-based access control for admin features.
- A user-friendly interface ensures ease of navigation and accessibility for users of varying technical expertise.

The e-commerce platform successfully meets the foundational requirements for a secure and scalable online store.

It provides essential features like user management, product browsing, cart operations, and order placement, all integrated into a cohesive and responsive system.

While the project has achieved its core objectives, certain areas require further attention



Future Improvements

Looking forward, several enhancements can extend the platform’s capabilities:

Improving Scalability	• Implement load balancing and caching mechanisms to handle higher user traffic. • Explore cloud hosting solutions for greater flexibility and reliability.
Adding Features	• Include wish lists, product reviews, and user notifications. • Develop analytics dashboards for admins to track sales, inventory, and user engagement.
Enhanced Security	Introduce two-factor authentication for users and advanced encryption methods for sensitive data.
Automated Workflows	Automate inventory updates and email notifications for order status.
Integration with Third-Party Tools	• Connect with social media platforms for user authentication (e.g., OAuth2). • Incorporate shipping and delivery tracking APIs for a complete order fulfillment cycle.

Thank You!