# WM240: Cyber Context of Software Engineering

Coursework 1

U5565332

February 2025

## Contents

# 1 Introduction

## 1.1 Background and Importance of SecureCart

SecureCart is a **security-focused e-commerce platform** designed to provide a **safe and efficient online shopping experience**. Unlike many platforms that prioritise functionality over security, SecureCart integrates **robust safeguards** to protect customer transactions and administrative operations from **unauthorised access and cyber threats**.

E-commerce platforms are frequent targets for **SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and brute-force attacks**. Without adequate protections, these vulnerabilities can lead to **financial loss, data breaches, and reputational damage**. SecureCart addresses these risks by implementing **secure authentication, encrypted transactions, and best practices in cybersecurity**.

## 1.2 Project Scope and Objectives

SecureCart follows the **V-Model SDLC**, ensuring **security and verification** at each development stage. The key objectives include:

- **Secure Authentication & Access Control** – Django's **authentication framework** enforces **role-based access control (RBAC)**, **password hashing**, and **brute-force attack prevention**.
- **Data & Transaction Security** – SecureCart uses **PostgreSQL** with ORM queries to **prevent SQL injection**, encrypt sensitive data, and enforce **HTTPS** for secure communication.
- **Secure Payment Processing** – The platform currently uses **dummy transactions**, with future plans for a **payment gateway integration**.
- **Mitigating Web Vulnerabilities** – The platform incorporates **CSRF protection**, **input validation** (to prevent XSS attacks), and **secure session cookies** to reduce session hijacking risks.
- **Security-Driven Development** – SecureCart undergoes **automated security testing** (e.g., **OWASP ZAP, SQLMap**) alongside **unit, integration, and penetration testing** to maintain security standards.

## 1.3 Key Functionalities of SecureCart

SecureCart provides **secure and structured functionalities** for both **customers** and **administrators**:

- **Customer Features:** Secure **account creation and login**, product browsing, cart management, and order tracking.
- **Administrator Features:** Role-based access for **product, user, and order management**, stock updates, and **sales reporting**.
- **Security Features:**
  - **CSRF protection** for form submissions.

- **Django's authentication system** for login security.
- **Encrypted session management** to prevent unauthorised access.
- **Secure payment processing** (dummy implementation, with future payment gateway integration).

### 1.4 Security-Driven Development Approach

SecureCart is built on **Django's Model-View-Template (MVT) architecture**, ensuring a **modular and secure structure**:

- **Django's authentication system** manages secure user authentication and session handling.
- **Security middleware** mitigates **common web vulnerabilities** (XSS, Clickjacking, CSRF).
- **PostgreSQL database management** enforces secure queries and **data encryption**.
- **JWT authentication** secures REST API communication.
- **CSRF protection** ensures safe form submissions.
- **Future payment integration** will support **secure transactions** without storing sensitive card details.

By adopting a **security-first approach**, SecureCart prioritises **data protection, secure transactions, and access control**, ensuring compliance with **e-commerce security best practices**.

# 2 Methodology

SecureCart follows the **V-Model Software Development Lifecycle (SDLC)** to integrate security from the earliest stages. This structured approach enables **early identification of vulnerabilities** and ensures **security is embedded throughout development**.

The focus of this project is the **Verification Phase**, which includes:

- **Requirements Gathering** – Defining system functionalities and security needs.
- **System Analysis** – Identifying security threats and designing countermeasures.
- **Software & Module Design** – Structuring the system for **scalability, maintainability, and security**.

Each phase ensures **SecureCart is security-driven before implementation begins**.

## 2.1 Verification Phase: Pre-Implementation Security Planning

### 2.1.1 Requirements Gathering

The first stage defines **functional and security requirements** for customers and administrators.

- **User Roles & Access Control** – **Role-Based Access Control (RBAC)** restricts access:
  - **Customers** browse and purchase products.
  - **Administrators** manage inventory, orders, and user accounts.
- **Security Policies** – Enforces **password complexity** to mitigate brute-force attacks.
- **Data Protection** – Uses **PBKDF2 hashing** for passwords and **TLS encryption** for secure communication.

### 2.1.2 System Analysis

SecureCart evaluates potential vulnerabilities and integrates **preventive security measures**:

- **SQL Injection** – Django's **Object-Relational Mapper (ORM)** ensures parameterised queries, preventing injection attacks.
- **Cross-Site Scripting (XSS)** – **Input sanitisation & Django's template escaping** block malicious scripts.
- **Cross-Site Request Forgery (CSRF)** – **CSRF tokens** protect form submissions.
- **Session Security** – **Auto-expiring login sessions** prevent hijacking attempts.

These controls **fortify SecureCart against common e-commerce vulnerabilities**.

### 2.1.3 Software & Module Design

SecureCart follows **Django's Model-View-Template (MVT) architecture**, ensuring structured **separation of concerns**:

- **Model (Data Layer)** – Stores customer, order, and payment data securely in **PostgreSQL**.
- **View (Business Logic Layer)** – Handles **authentication, authorisation, and processing requests**.
- **Template (Presentation Layer)** – Renders user interfaces **while preventing XSS via Django's built-in security**.

To enhance security, the system is modular, mitigating specific threats:

| Module | Security Measures |
|---|---|
| **User Authentication** | **PBKDF2 password hashing, session-based login, CSRF protection**. |
| **Shopping Cart** | **Session-based cart management to prevent tampering**. |
| **Checkout & Payment** | **Secure transactions via external APIs (dummy payment system for now).** |
| **Admin Dashboard** | **RBAC restricts access to authorised users only**. |
| **API Security** | **Token-based authentication (JWT/session-based), rate-limiting**. |

Each module **adheres to best security practices**, ensuring **data integrity, access control, and user privacy**.

# 3 Requirements Phase

The **Requirements Phase** defines the functional, security, and system expectations for SecureCart, ensuring that the platform meets both **customer** and **administrator needs** while maintaining **industry-standard security practices**. This phase is crucial in aligning **business objectives with technical feasibility**, ensuring that the system is secure, scalable, and user-friendly.

SecureCart's requirements can be categorised into three key areas:

- **Functional Requirements:** Core features that define the system's usability and business logic.
- **Security Requirements:** Safeguards to protect user data and ensure secure transactions.
- **System Requirements:** Underlying architecture and technologies ensuring performance, maintainability, and security.

## 3.1 Functional Requirements

SecureCart provides functionalities tailored for two types of users: **customers** and **administrators**.

### 3.1.1 Customer Fea1tures

Customers interact with SecureCart through a secure, intuitive interface that allows them to browse, purchase, and track their orders.

| ID | Requirement | Description |
|---|---|---|
| F-01 | User Registration | Customers sign up with email verification. |
| F-02 | User Login | Secure login with authentication and session management. |
| F-03 | Browse Products | Users can view product details. |
| F-04 | Shopping Cart | Add, update, and remove items. |
| F-05 | Checkout | Secure checkout with order confirmation. |
| F-06 | Order History | View past orders and track status. |

Table 1: Functional Requirements

### 3.1.2 Administrator Features

Administrators have elevated privileges, allowing them to manage the e-commerce platform efficiently.

| ID | Requirement | Description |
|---|---|---|
| F-07 | Manage Products | Admins can add, update, and remove products from the inventory. |
| F-08 | Manage Orders | Admins can process, approve, and update orders. |
| F-09 | Manage Users | Admins can create, edit, and deactivate customer accounts. |
| F-10 | Sales Analytics | Admins can track order statistics, revenue, and customer trends. |

These functional requirements ensure a **seamless e-commerce experience** while maintaining **role-based access control (RBAC)** to prevent unauthorised administrative actions.

## 3.2 Security Requirements

Security is integral to SecureCart's design, ensuring **data protection, secure transactions, and access control**. The platform incorporates **multiple security layers** to prevent threats such as unauthorised access, data breaches, and injection attacks.

| ID | Security Requirement | Justification |
|---|---|---|
| S-01 | Secure Authentication | Passwords are hashed using **PBKDF2** to prevent credential exposure. |
| S-02 | CSRF Protection | Forms include **CSRF tokens** to prevent unauthorised form submissions. |
| S-03 | HTTPS Enforcement | **TLS encryption** secures all communication between client and server. |
| S-04 | Role-Based Access Control (RBAC) | Distinct privileges for customers and admins prevent unauthorised access. |
| S-05 | Secure Payment Gateway | Payments are handled via **third-party APIs** to avoid storing card details. |
| S-06 | Input Validation | System-wide validation prevents **SQL Injection, XSS, and CSRF** attacks. |
| S-07 | Secure Admin Panel | The admin dashboard is **access-restricted and hidden** from unauthorised users. |

These security controls ensure that SecureCart adheres to **best practices in cybersecurity**, preventing common web vulnerabilities while **maintaining data integrity and confidentiality**.

## 3.3 System Requirements

The system is built to be **scalable, maintainable, and secure**, following a **three-tier architecture** comprising the **frontend, backend, and database layers**.

| Component | Description | Security Measures |
|---|---|---|
| **Frontend (UI)** | Django Templates and Bootstrap for responsive design. | **Prevents XSS** through template escaping and secure rendering. |
| **Backend (API & Business Logic)** | Manages authentication, order processing, and user sessions. | **RBAC, authentication enforcement, and input validation**. |
| **Database (Data Storage)** | PostgreSQL stores users, products, and orders. | Uses **ORM queries to prevent SQL Injection**, with encrypted connections. |

This **layered architecture** ensures that SecureCart is **resilient against cyber threats**, scalable for future expansion, and optimised for efficient e-commerce transactions.

### 3.4 Secure Database Design

The database is designed to **prevent data breaches and unauthorised modifications** by enforcing structured **access control and encryption policies**. The following design considerations ensure security:

- **User Table**: Stores **hashed passwords** and enforces **role-based access control** for customers and administrators.
- **Orders Table**: Ensures that **transaction records are immutable**, preventing unauthorised modifications.
- **Products Table**: Uses **parameterised queries** to prevent SQL injection attacks.
- **Cart Table**: Protects cart data using **session-based security**, ensuring that users cannot modify items outside their session.

SecureCart follows **GDPR compliance principles** for **user data protection** and ensures that all database connections are **encrypted** to prevent interception during data transmission.

# 4 Project Implementation

SecureCart follows **Django's Model-View-Template (MVT) architecture**, ensuring a structured separation between **data management, user interaction, and business logic**. The **backend** handles authentication, order processing, and security enforcement, while the **frontend** provides a seamless and secure shopping experience. PostgreSQL serves as the **database**, ensuring **data integrity and security**.

This section details the **core implementation components** and **security measures** integrated at each level.

## 4.1 System Architecture

SecureCart's architecture consists of **three main layers**:

1. **Presentation Layer (Frontend)** – Django templates & Bootstrap for UI.
2. **Business Logic Layer (Backend)** – Django's models, views, authentication, and APIs.
3. **Data Layer (Database)** – PostgreSQL database with secured queries and **role-based access**.

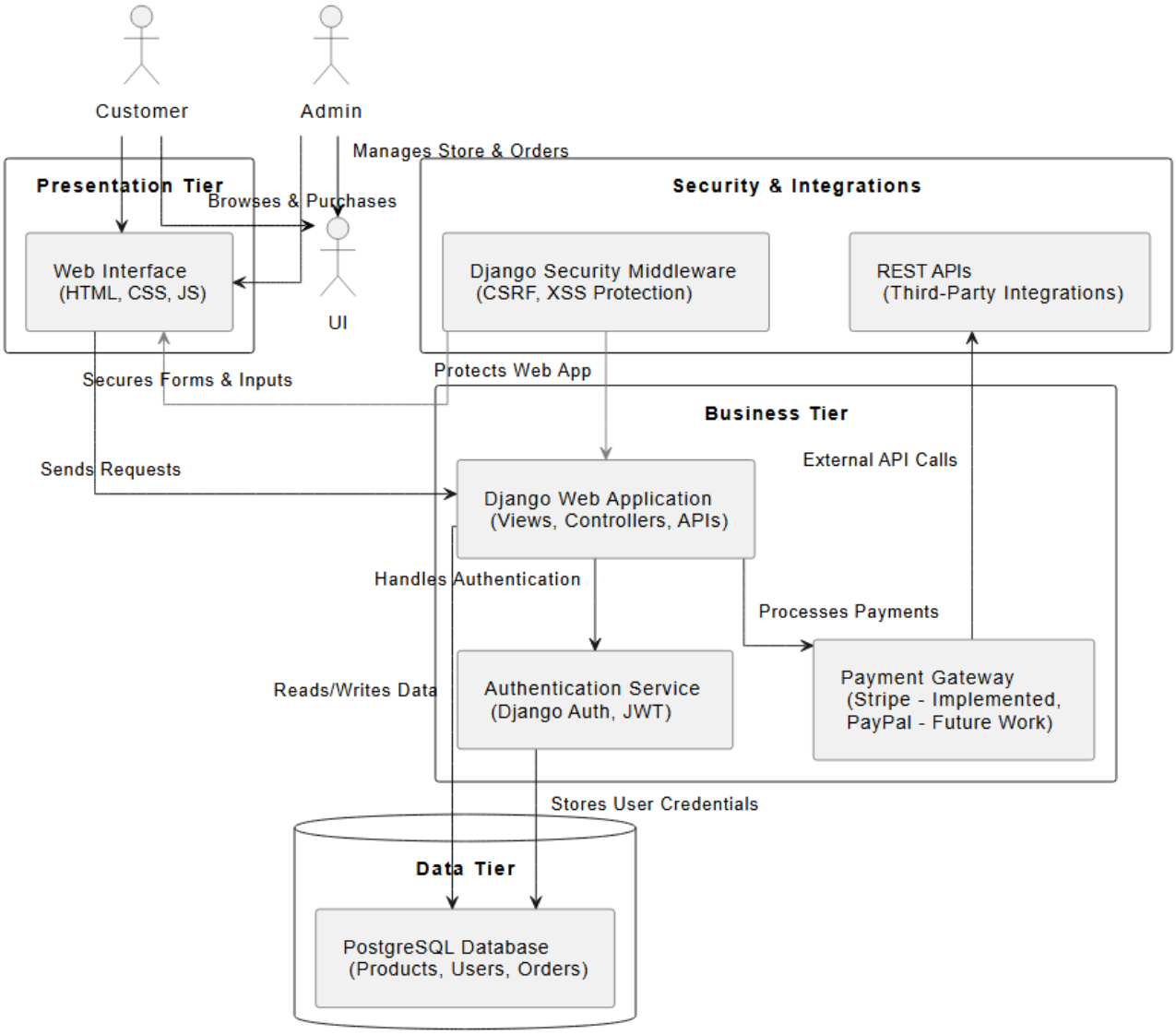


Figure 1: **SecureCart System Architecture**

Each layer enforces **security policies**, ensuring **secure authentication, protected API endpoints, and encrypted data storage**.

## 4.2 Database Configuration

SecureCart uses **PostgreSQL** instead of SQLite, providing **enhanced security, scalability, and performance**. PostgreSQL offers built-in **encryption, role-based access control, and transaction security**.

**Key security enhancements in the database:**

- **User passwords** are stored using Django's authentication system with **PBKDF2 hashing**.
- **Role-based permissions** control which users can modify data.
- **Prepared statements** prevent **SQL injection attacks**.
- **Encrypted database connections** ensure secure data transmission.
- **Transaction consistency** prevents **data loss during checkout**.

```
# Database Configuration
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv("DB_NAME", "securecart"),
        'USER': os.getenv("DB_USER", "postgres"),
        'PASSWORD': os.getenv("DB_PASSWORD", ""),
        'HOST': os.getenv("DB_HOST", "localhost"),
        'PORT': os.getenv("DB_PORT", "5432"),
    }
}
```

Figure 2: PostgreSQL Database Configuration

## 4.3 Authentication System

SecureCart employs **Django's built-in authentication system** for **user login, registration, and role-based access control (RBAC)**.

### 4.3.1 Authentication Features

- **Django authentication** manages login, password hashing, and session handling.
- **Account lockout** limits failed attempts to prevent brute-force attacks.
- **RBAC** restricts access for **customers and administrators**.
- **CSRF tokens** secure authentication requests.
- **Future Enhancement:** Multi-factor authentication (MFA).

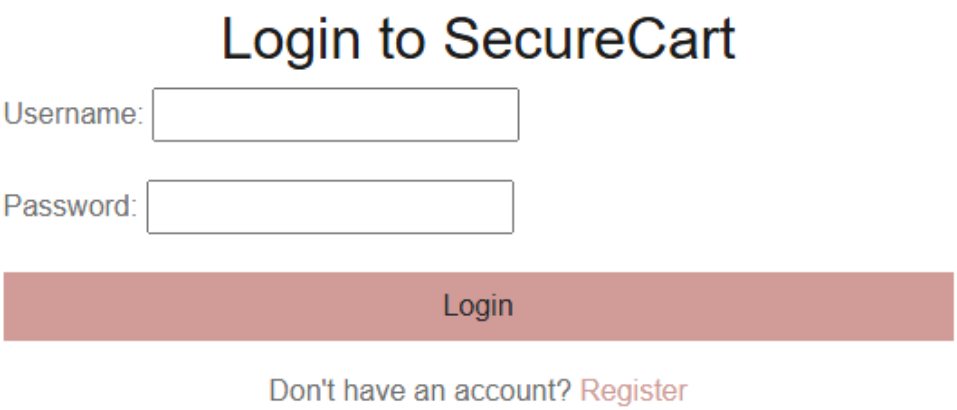


Figure 3: Securecart Login Page

## 4.4 Backend Development with Django

Django provides **a secure, scalable backend** with built-in security mechanisms.

### 4.4.1 Backend Features

- **Django ORM** protects against **SQL injection** via parametrised queries.
- **Middleware security** includes **XSS filtering, CSRF protection, and session security**.
- **Django REST Framework (DRF)** powers **RESTful APIs** for product listings and cart updates.
- **JWT authentication** secures API access.

## 4.5 Role-Based Access Control (RBAC) Implementation

SecureCart enforces **RBAC to restrict access based on user roles**.

| User Type | Permissions |
|---|---|
| **Customer** | Can **browse products, add items to cart, and complete purchases**. |
| **Administrator** | Can **manage users, products, and orders through the admin dashboard**. |

### 4.5.1 RBAC Implementation in Django:

RBAC ensures **users cannot access unauthorised features**, enforcing **least privilege principles**.

## 4.6 Shopping Cart & Checkout Implementation

SecureCart provides a **session-based shopping cart**, enabling users to manage their purchases securely.

### 4.6.1 Shopping Cart Logic

- **Session-based storage** ensures cart integrity.
- **CSRF protection** secures cart modifications.
- **Automatic quantity updates** prevent errors.

### 4.6.2 Checkout Process

- Users confirm orders and complete **dummy payment transactions**.
- The system **verifies stock availability** before processing.
- Orders are **saved in the database** and marked as **"Paid"**.

## 4.7 Dummy Payment System

SecureCart **simulates transactions** without processing real payments.

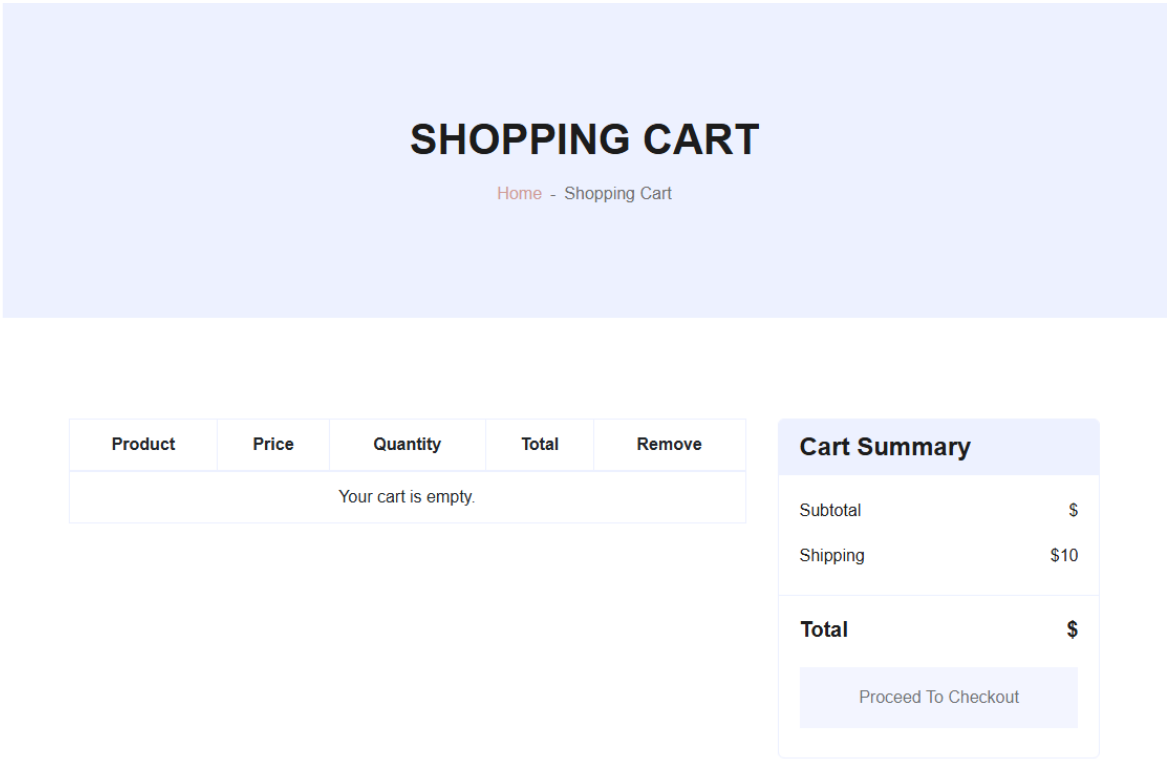


Figure 4: Securecart Cart page

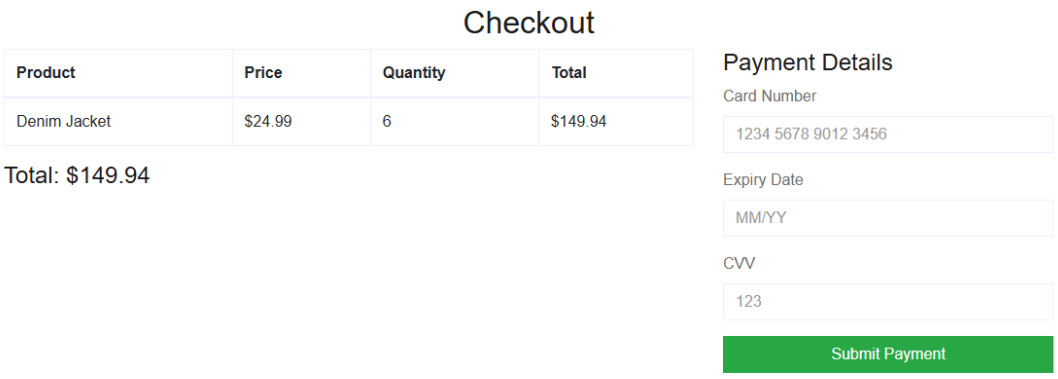


Figure 5: Securecart Checkout page

#### 4.7.1 Payment Workflow

1. User selects **"Dummy Payment"** at checkout.
2. The system **validates payment details** (without actual transactions).
3. The order is marked as **"Paid"**.
4. The user receives a **confirmation message**.

This method **allows testing of payment workflows without external API dependencies**.

### 4.8 Admin Dashboard Implementation

The **admin dashboard** enables administrators to **manage products, users, and orders securely**.

#### 4.8.1 Admin Dashboard Features

- **Manage Products** – Add, update, and delete listings.
- **Manage Orders** – Track order statuses and process refunds.
- **User Management** – Control **account access and permissions**.
-

Django's **admin interface** enforces **RBAC and authentication**, ensuring **secure access control**.

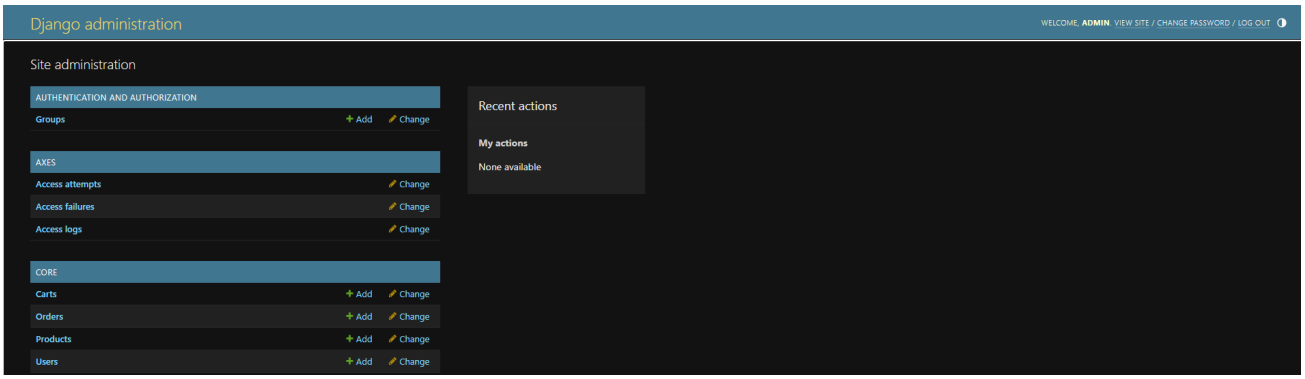


Figure 6: Securecart Admin Dashboard

# 5 Security Implementation

## 5.1 Security Policies

SecureCart enforces **strong password security policies** by using **PBKDF2 hashing with SHA-256**, ensuring that user passwords are **never stored in plaintext**.

### 5.1.1 Security Features:

- **PBKDF2 hashing** makes brute-force attacks significantly harder.
- **Password validators** enforce complexity requirements (minimum length, common password prevention, numeric restrictions).

```
# Password Validation
AUTH_PASSWORD_VALIDATORS = [
    {'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator'},
    {'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator'},
    {'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator'},
    {'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator'},
]
```

Figure 7: Password Security (settings.py)

## 5.2 CSRF Protection

Cross-Site Request Forgery (**CSRF**) attacks occur when a malicious site tricks users into executing **unauthorised actions**. SecureCart **mitigates this risk** by implementing **CSRF protection in all forms**.

### 5.2.1 Implementation:

- **Django automatically integrates CSRF tokens** into templates.
- **Every POST request** must include a **CSRF token** to be processed.

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}  <!-- Uses Django's AuthenticationForm -->
    <button type="submit" class="btn btn-primary w-100">Login</button>
</form>
```

Figure 8: CSRF Protection (login.html)

## 5.3 Session Security

To protect user sessions from **hijacking and unauthorised reuse**, SecureCart enforces multiple session security measures.

### 5.3.1 Security Features:

- **HTTP-only cookies** prevent JavaScript-based XSS attacks.
- **Secure session cookies** ensure encrypted transmission over HTTPS.
- **Auto logout after inactivity** to minimise hijacking risks.
- **Session expiry on browser close** prevents unauthorised reuse.

```
# Security & Session Management
SESSION_COOKIE_SECURE = True  # Ensures session cookies are only sent over HTTPS
CSRF_COOKIE_SECURE = True     # Ensures CSRF cookies are only sent over HTTPS
SESSION_COOKIE_AGE = 1800     # Auto logout after 30 minutes
SESSION_EXPIRE_AT_BROWSER_CLOSE = True  # Logout when browser closes
SESSION_COOKIE_HTTPONLY = True  # Prevents JavaScript from accessing session cookie
```

Figure 9: Session Security (settings.py)

## 5.4 Role-Based Access Control (RBAC)

SecureCart enforces **Role-Based Access Control (RBAC)** to **restrict system access based on user roles**, ensuring **customers and administrators have distinct privileges**.

### 5.4.1 Access Control Rules:

- **Customers** → Can **browse products, add to cart, and complete purchases**.
- **Administrators** → Can **manage users, products, and orders** via the **admin dashboard**.

RBAC ensures **unauthorised users cannot perform restricted actions**, maintaining **data integrity and security**.

```
# Admin Product Management
@login_required
@user_passes_test(lambda u: u.is_staff)
def manage_products(request):
    """Admin-only view for managing products"""
    products = Product.objects.all()
    return render(request, "admin/manage_products.html", {"products": products})
```

Figure 10: RBAC (views.py)

## 5.5 Security Middleware & Protections

Django provides **built-in security middleware** to protect against **common web vulnerabilities**, including **Clickjacking, XSS, and Content Sniffing attacks**.

### 5.5.1 Security Middleware Features:

- **Clickjacking Protection** → Prevents SecureCart from being embedded in an iframe.
- **Cross-Site Scripting (XSS) Protection** → Blocks malicious scripts from executing in the browser.
- **Strict Transport Security (HSTS)** → Enforces **HTTPS connections** for all communications.

```
SECURE_BROWSER_XSS_FILTER = True  # Prevents cross-site scripting (XSS) attacks
# Clickjacking Protection
X_FRAME_OPTIONS = 'DENY'  # Prevents the site from being embedded in an iframe
```

Figure 11: Django Security (settings.py)

# 6 System Design Phase

The **System Design Phase** defines SecureCart's **architecture, components, data flow, and security measures**, ensuring **scalability and maintainability**. SecureCart follows Django's **Model-View-Template (MVT) architecture**, structured within a **three-tier system** to separate the **UI, business logic, and data layers**.

## 6.1 High-Level Architecture

SecureCart follows a **three-tier architecture**:

| **Layer** | Description | Technologies Used |
|---|---|---|
| **Presentation Layer** | User interface for customers and admins. | Django Templates, Bootstrap |
| **Business Logic Layer** | Manages authentication, shopping cart, and order processing. | Django Views, Django REST Framework |
| **Data Layer** | Securely stores user, product, and order data. | PostgreSQL, Django ORM |

Django's **MVT structure** ensures security and modularity:

- **Model (Data Layer):** Manages **secure data storage**.
- **View (Business Logic):** Renders **HTML securely** using Django templates.
- **Controller (Request Handling):** Manages **authentication, requests, and API** interactions.

## 6.2 Key Components & Security Measures

SecureCart is divided into five main components, each incorporating **security best practices** to mitigate vulnerabilities.

| **Component** | **Description** | **Security Measures** |
|---|---|---|
| **Frontend (UI)** | Django Templates and Bootstrap for a responsive design. | **Prevents XSS** through template escaping and secure rendering. |
| **Backend (API & Business Logic)** | Manages authentication, order processing, and user sessions. | **RBAC, authentication enforcement, and input validation**. |
| **Database (Data Storage)** | PostgreSQL stores users, products, and orders. | Uses **ORM queries to prevent SQL Injection**, with encrypted connections. |

## 6.3 Secure Database Design

SecureCart's **Entity-Relationship Diagram (ERD)** defines relationships between **users, products, orders, and cart items**, ensuring a **secure data model**.

### 6.3.1 Database Design Principles

- **Users Table:** Implements **PBKDF2 password hashing** and **RBAC**.
- **Orders Table:** Uses **immutable transaction records** to prevent post-purchase edits.
- **Products Table: Parameterised queries** prevent SQL injection.
- **Cart Table: Session-based storage** prevents tampering.

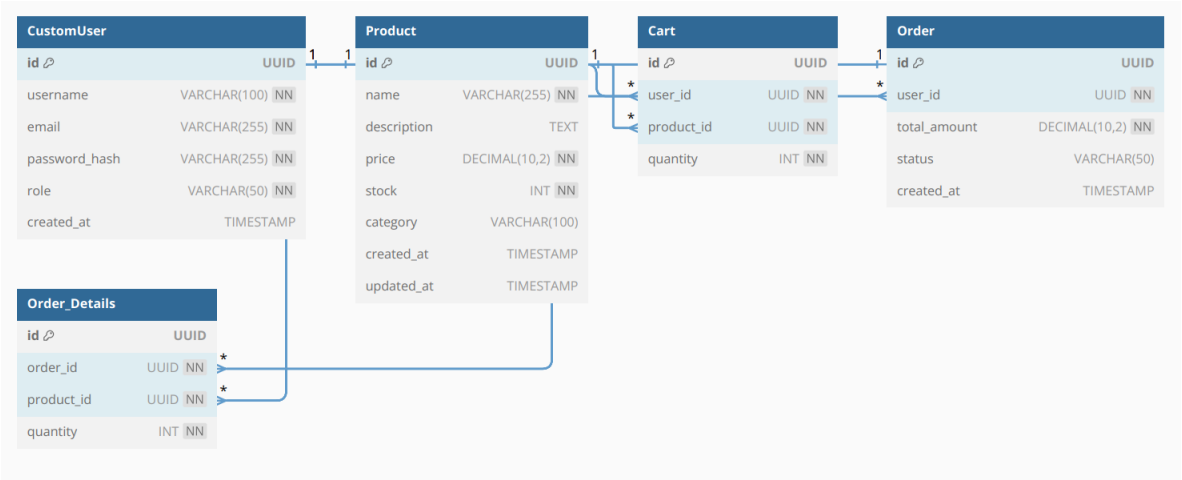


Figure 12: Entity-Relationship Diagram (ERD)

## 6.4 Data Flow & Security

SecureCart ensures **secure authentication, product selection, and transaction processing**.

### 6.4.1 User Login Flow

1. User enters credentials.
2. Django validates password **(hashed using PBKDF2)**.
3. On success, a **secure session token is generated**.
4. User is redirected to the homepage.

### 6.4.2 Checkout Process Flow

1. User adds items to cart, proceeds to checkout.
2. The **CSRF-protected checkout page** loads.
3. User selects **"Dummy Payment"** (no real transaction).
4. Order is saved in the database and marked **"Paid"**.

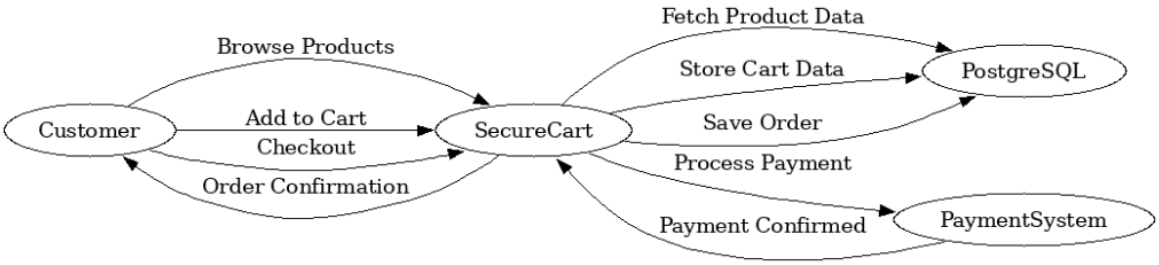


Figure 13: Data Flow Diagram

5. User is redirected to an **order confirmation page**.

This **structured data flow prevents unauthorised access** and ensures **secure communication** between system components.

The sequence diagram below shows the interaction between the **customer, SecureCart system, PostgreSQL database, and dummy payment system** during checkout. It outlines how **user actions trigger backend responses**, ensuring **secure order processing**.

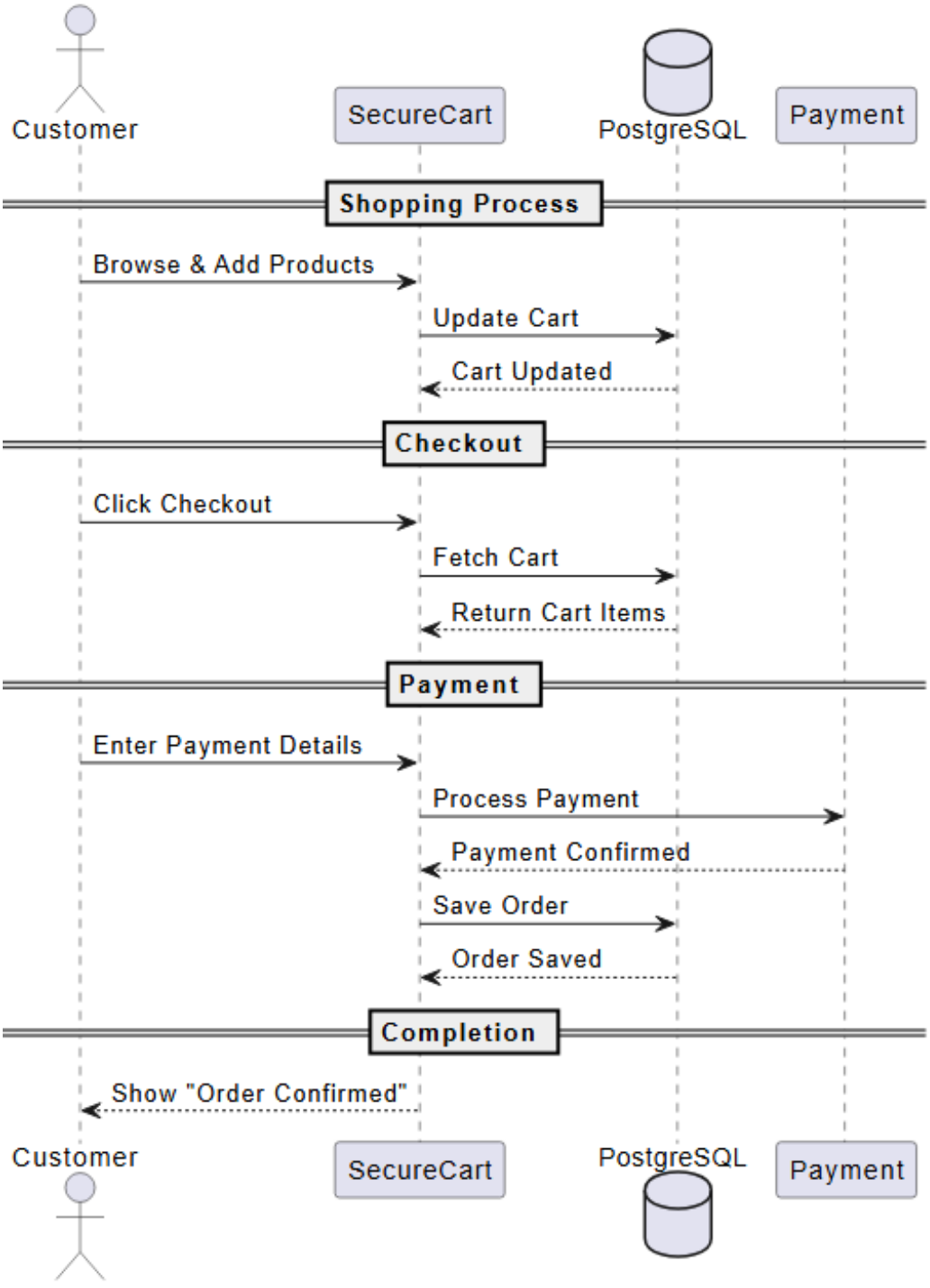


Figure 14: Sequence Diagram

```python
class Product(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField()
    category = models.CharField(max_length=100, default="Uncategorized")
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name  # Show product name in admin panel
```

Figure 16: Product Model

## 6.5 Database Models

SecureCart uses Django's **Object-Relational Mapping (ORM)** to define secure database models.

| Model | Description | Security Measures |
|---|---|---|
| **User** | Stores user data (name, email, password, role). | **Passwords hashed (PBKDF2), role-based access control.** |
| **Product** | Stores product details (name, description, price, stock). | **Prevents SQL injection via Django ORM.** |
| **Cart** | Manages shopping cart items per user. | **Session-based cart, CSRF protection.** |
| **Order** | Stores purchase details and order status. | **Ensures integrity with database constraints.** |

```python
class CustomUser(AbstractUser):
    ROLE_CHOICES = (
        ('admin', 'Admin'),
        ('customer', 'Customer'),
    )
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default='customer')
```

Figure 15: User Model

```python
@csrf_protect
@login_required
@require_POST
def add_to_cart(request, product_id):
    """Handles adding a product to the cart"""
    if request.method == "POST":
        product = get_object_or_404(Product, id=product_id)
        user = request.user

        # Get or create cart item
        cart_item, created = Cart.objects.get_or_create(user=user, product=product)

        if not created:
            cart_item.quantity += 1  # Increase quantity if already in cart
        else:
            cart_item.quantity = 1  # Ensure initial quantity is set

        cart_item.save()

        # Return updated cart info
        updated_cart = Cart.objects.filter(user=user).count()
        return JsonResponse({"success": True, "message": "Product added to cart!", "cart_count": updated_cart})

    return JsonResponse({"success": False, "error": "Invalid request"}, status=400)
```

Figure 17: Add to cart function

## 6.6 Views & Business Logic

SecureCart follows Django's **views-based approach** to process user requests securely.

| Feature | View Type | Security Features |
|---|---|---|
| **User Registration** | Function-Based View | **Password hashing, email verification** |
| **Login & Logout** | Django Auth | **Session-based authentication, login attempt limit** |
| **Shopping Cart** | Function-Based View | **CSRF protection, session-based cart management** |
| **Checkout** | Function-Based View | **Secure API calls for payment processing** |
| **Admin Dashboard** | Django Admin Panel | **RBAC, restricted access** |

## 6.7 Frontend Development (Templates & UI)

SecureCart uses **Django Templates & Bootstrap** for a **responsive and secure** UI.

### 6.7.1 Key UI Features

- **Homepage:** Displays products dynamically.
- **Cart Page:** Allows **item updates and removal**.
- **Checkout Page:** Secures **order processing**.
- **Admin Panel:** Enables **product and sales management**.

```
{% block content %}
<div class="container mt-5">
    <h2 class="text-center">All Products</h2>
    <div class="row">
        {% for product in products %}
        <div class="col-md-4">
            <div class="card">
                <!-- Product Image -->
                <img src="{% static 'img/product-' %}{{ forloop.counter }}.jpg" class="card-img-top" alt="{{ product.name }}">
                <div class="card-body">
                    <!-- Product Details -->
                    <h5 class="card-title">{{ product.name }}</h5>
                    <p class="card-text">${{ product.price }}</p>
                    <a href="#" class="btn btn-primary add-to-cart" data-product-id="{{ product.id }}">
                        Add to Cart
                    </a>
                </div>
            </div>
        </div>
        {% empty %}
        <p class="text-center">No products available.</p>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Figure 18: Product page

Django's **template rendering engine automatically escapes user input**, **preventing XSS attacks**. Additionally, **CSRF tokens protect form submissions**.

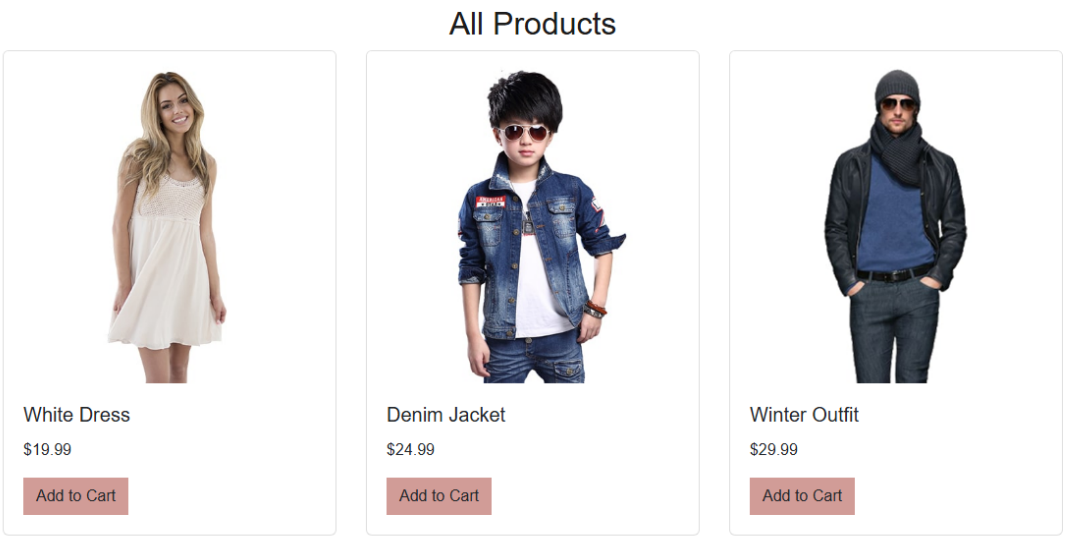


Figure 19: Product page UI

# 7 Conclusion

The **SecureCart platform** successfully implements a **secure and scalable e-commerce system** by adhering to the **V-Model development approach**, ensuring that security measures are integrated at every stage. Key implementations such as **role-based access control (RBAC), secure authentication, CSRF protection, and HTTPS enforcement** provide a **robust foundation** for protecting user data and transactions.

## 7.1 Future Implementation

While the current implementation establishes a **strong security baseline**, future enhancements will focus on **improving authentication, scalability, and overall system performance**. Planned updates include:

- **Two-Factor Authentication (2FA)** to strengthen user account security.
- **Secure Payment Gateway Integration** to support real transactions (replacing the current dummy payment system).
- **Enhanced API Security** through **JWT authentication, rate limiting, and stricter access controls**.
- **Performance Optimisation**, including **database indexing, query optimisation, and caching** to improve speed and efficiency.
- **User Experience Enhancements**, such as **live search, improved navigation, and mobile responsiveness** for a seamless shopping experience.

These improvements will ensure **SecureCart remains scalable, secure, and user-friendly**, adapting to evolving security challenges while maintaining **best practices in data protection and system integrity**.