Grocery Store Web Application - Technical Documentation

1. Project Overview

Purpose & Core Functionality

The Grocery Store Web Application is a comprehensive e-commerce platform designed to facilitate online grocery shopping. It provides a seamless interface for customers to browse, purchase groceries, and for sellers to manage their products and orders.

Target Audience & Use Cases

- **End Users (Customers)**
  - Browse and search for grocery products
  - Manage shopping cart and wishlists
  - Place and track orders
  - Rate and review products
  - Manage delivery addresses
- **Sellers**
  - List and manage products
  - Process orders
  - Track inventory
  - Manage product categories
  - View sales analytics
- **Administrators**
  - Manage user accounts
  - Oversee platform operations
  - Monitor transactions
  - Maintain product categories

Key Features

- User Authentication and Authorization
- Product Catalog with Categories and Subcategories
- Shopping Cart Management
- Order Processing and Tracking
- Seller Dashboard
- Product Rating and Reviews
- Wishlist Management
- Address Management
- Real-time Stock Updates
- Search Functionality
- Responsive Design
- Secure Payment Processing

2. Technology Stack

Backend

- **Language**: Python 3.x
- **Framework**: Flask
- **Key Libraries**:
  - Flask-SQLAlchemy (Database ORM)
  - Flask-Login (User Authentication)
  - Werkzeug (Security and Utilities)
  - SQLAlchemy (Database Operations)

Frontend

- **Core Technologies**:
  - HTML5
  - CSS3
  - JavaScript
- **Frameworks/Libraries**:
  - TailwindCSS (Styling)
  - Alpine.js (Interactive UI)
  - Jinja2 (Templating)

Database

- **System**: SQLite
- **ORM**: SQLAlchemy
- **Schema Design**:
  - Normalized database structure
  - Efficient relationships between entities
  - Proper indexing for performance
  - Cascade delete handling

DevOps

- **Version Control**: Git
- **Development Environment**: VS Code
- **Static Files**: Served through Flask
- **Image Storage**: Local file system with secure naming

3. Project Folder Structure

“‘ project-root/ app.py # Main application file requirements.txt # Python dependencies README.md # Project readme static/ # Static assets css/ # Stylesheets js/ # JavaScript files uploads/ # User uploads products/ # Product images templates/ # HTML templates auth/ # Authentication templates pages/ # Main page templates partials/ # Reusable components instance/ # Instance-specific files node_modules/ #

Node.js packages package.json # Node.js dependencies package-lock.json # Node.js lock file tailwind.config.js # TailwindCSS configuration “‘

4. Python Code Deep Dive

Core Components

Authentication System “‘python @app.route(‘/login’, methods=[‘GET’, ‘POST’]) def login(): # Handle user login # Validate credentials # Create user session

@app.route(‘/register’, methods=[‘GET’, ‘POST’]) def register(): # Handle user registration # Validate input # Create new user account “‘

Database Models “‘python class User(UserMixin, db.Model): id = Column(Integer, primary_key=True) name = Column(String(100), nullable=False) email = Column(String(120), unique=True, nullable=False) # … other fields

class Product(db.Model): id = Column(Integer, primary_key=True) name = Column(String(200), nullable=False) price = Column(Float, nullable=False) # … other fields “‘

Business Logic Flow

1. **User Registration**:
   - Validate input data
   - Hash password
   - Create user record
   - Initialize cart
2. **Product Management**:
   - Upload and validate images
   - Update inventory
   - Manage categories
3. **Order Processing**:
   - Validate stock availability
   - Calculate total
   - Process payment
   - Update inventory
   - Send confirmation

5. Database Schema

Entity-Relationship Diagram

“‘mermaid erDiagram USERS ||–o{ ORDERS : places USERS ||–o{ ADDRESSES : has USERS ||–o{ CART : owns USERS ||–o{ WISHLIST_ITEMS : saves USERS ||–o{ RATINGS : gives PRODUCTS ||–o{ CART_ITEMS :

contains PRODUCTS ||–o{ ORDER_ITEMS : includes PRODUCTS ||–o{ RATINGS : receives PRODUCTS ||–o{ WISHLIST_ITEMS : saved_in PRODUCTS }|–|| SUBCATEGORIES : belongs_to SUBCATEGORIES }|–|| CATEGORIES : belongs_to ORDERS ||–|{ ORDER_ITEMS : contains CART ||–|{ CART_ITEMS : contains ORDERS ||–|| ADDRESSES : ships_to “‘

Key Tables

Users

- Primary user information
- Authentication details
- Role management (customer/seller)

Products

- Product details
- Inventory management
- Category relationships
- Image storage

Orders

- Order tracking
- Payment information
- Shipping details
- Order items

6. Frontend Architecture

Template Structure

- Base template with common elements
- Page-specific templates
- Reusable components
- Responsive design elements

Key Components

“‘html <!DOCTYPE html>

{% include ‘partials/header.html’ %} {% block content %}{% endblock %} {% include ‘partials/footer.html’ %}

“‘

JavaScript Functionality

- Cart management

- Form validation
- Dynamic content loading
- Interactive UI elements

7. API Endpoints

Endpoint	Method	Description	Authentication
/api/products	GET	List all products	No
/api/cart	GET	Get cart contents	Yes
/api/cart/add	POST	Add to cart	Yes
/api/orders	POST	Place order	Yes
/api/subcategories/:id	GET	Get subcategories	No

8. Deployment & Configuration

Environment Setup

1. Python virtual environment
2. Install dependencies
3. Configure environment variables
4. Initialize database

Database Setup

“‘python def init_db(): with app.app_context(): db.create_all() # Create initial categories # Add admin user “‘

Security Measures

- Password hashing
- CSRF protection
- Secure file uploads
- Input validation
- Session management

9. Testing & Validation

Form Validation

- Client-side validation
- Server-side validation
- Custom validators

Security Testing

- Input sanitization

- Authentication checks
- Authorization validation

10. Future Scope

Planned Features

- Payment gateway integration
- Real-time order tracking
- Mobile application
- Advanced analytics
- Inventory forecasting

Performance Optimization

- Database query optimization
- Caching implementation
- Image optimization
- Load balancing

11. Conclusion

The Grocery Store Web Application provides a robust platform for online grocery shopping with comprehensive features for both customers and sellers. The modular architecture ensures scalability and maintainability, while the security measures protect user data and transactions.

12. Appendices

Glossary

- **ORM**: Object-Relational Mapping
- **CSRF**: Cross-Site Request Forgery
- **JWT**: JSON Web Token
- **API**: Application Programming Interface

References

- Flask Documentation
- SQLAlchemy Documentation
- TailwindCSS Documentation
- Python Best Practices Guide

License