Software Requirement Specification (SRS)

Project: Grocery Store Manager (GSM)

Version: 1.0

Author: Sofie Thorlund & Viktor Mekis Bach

Date: 29.11.2025

1. Introduction

1.1 Purpose

The purpose of this document is to specify the requirements for the Grocery Store Manager (GSM) web application. This SRS describes the system's functionality, user interactions, constraints and architecture. It will be used as the basis for development, testing and future maintenance.

1.2 Scope

Grocery Store Manager (GSM) is a web-based management tool designed for small grocery store owners. The application enables store owners to:

- Manage their list of products (name, quantity, units, price per unit, description).
- View and track customer orders.
- Add new products and modify or remove existing products.
- Receive updates on new incoming orders.

The system consists of:

- A frontend user interface.
- A backend REST API.
- A MySQL database.
- An external public API integration (like weather information displayed in the UI).

This SRS covers functional and non-functional requirements for the system.

1.3 Definitions

- CSM – Grocery Store Manager
- Owner – The grocery store owner using the application
- Customer – Any user placing an order through the store's order functionality
- Product – An item available in the grocery store
- Order – A set of products submitted by a customer

1.4 System Overview

The system is a web application accessed through the browser. The owner uses the toll to manage store inventory and view orders. Customers can submit orders via the "New Order" interface.

2. System Description

2.1 User Roles

Role	Description
Store Owner	Can manage products and view/manage orders.
Customer	Can create and submit new product orders.

2.2 Major features

- Product management (CRUD operations).
- Order management (view incoming orders).
- External API integration.
- Two-screen UI: "Orders" and "Products".

3. Functional Requirements

3.1 Order Management Screen

Description:

This is the home screen for the groceru stor owner.

FR-1: View Orders

The system should display a list of all orders in a table/grid.

FR-2: Display Order Details

Each order should show:

- Date
- Customer name
- Order content (summary or quantity/variety)

FR-3: Highlight New Orders

Newly added orders should appear at the top of the list.

FR-4: Create New Order

The customer should be able to create a new order by clicking the "New Order" button.

3.2 Product Management Screen

Description:

Accessible by the store owner through the "Manage Products" view.

FR-5: View Products

The system should display a table of products containing:

- Product name
- Unit available
- Price per unit

FR-6: Add Product

The owner should be able to click "New Product" and enter:

- Product name
- Description
- Quantity/units
- Price

The product should be added to the database.

FR-7: Modify Product

The owner should be able to update product details.

FR-8: Delete Product

The owner should be able to remove products.

3.3 External API Integration

FR-9: Weather Widget

The system should integrate with a public API (like OpenWeather API) to display:

- Temperature
- Conditions
- Location

Display on the owner dashboard.

4. Non-Functional Requirements

4.1 Usability

- Clean and intuitive UI styled with Bootstrap.
- Buttons and tables must be easy to navigate for non-technical users.

4.2 Performance

- Pages should load within 2 seconds under normal conditions.
- The system must support at least 50 concurrent users during performance testing.

4.3 Reliability

- The system must handle invalid user input gracefully.
- API errors must be logged and shown with user-friendly massages.

4.4 Security

- Only store owners can access the product/order management pages.
- Field validation must be performed on frontend and backend.

5. System Architecture

5.1 Overview

The application follows a standard three-tier architecture:

Frontend

- HTML, CSS, JavaScript
- Bootstrap for layout and styling
- Users Fetch API / AJAX to communicate with backend

Backend

- Python Flask server
- REST API (JSON-based)
- Routes for orders, products and external API calls

Database

- MySQL relational database
- Tables:
    - products
    - Orders
    - Order_items
    - Users (if Authentication added)

External APIs

- Public weather API or similar

6. Testing Requirements

6.1 Static Testing

- Code linters for Python and JavaScript
- Static analysis tool (SonarQube)

6.2 Unit and Integration Testing

- Python unittest / pytest
- Parameterized test cases
- Tests must reflect black-box test cases previously designed
- Backend integration tests for endpoints

6.3 API Testing

- Postman
- Positive and negative cases
- Test for:
    - HTTP status codes
    - JSON responses
    - Error handling
    - Response time

6.4 End-to-End UI Testing

- Selenium WebDriver (Python or JS bindings)
- Tests run through the browser, simulating real user actions

6.5 Performance and Stress Testing

- Apache JMeter
- Load, spike and stress tests
- Reports included

6.6 Usability Testing (Design Only)

- List of performance metrics
- Task-based scenarios
- Success criteria defined

7. Constraints

- Must be delivered before 14 December 2025, 23:59.
- Must include a frontend, backend, database and external API.
- Must be delivered as a ZIP file no external links.

8. Future Enhancements (Not required for hand in, but consider though)

Possible improvements:

- User authentication
- Product categories
- Order history analytics
- Mobile application support

First draft:

The web application named Grocery Store Manager (GSM), will functions as an online web management tool for small grocery stores. I should help them gain and keep an overview of the products in store and product details, such as product description, product quantity. The grocery store should also be able to see any orders made on the web app.

The web app will have two screens, one for maintaining orders. If the customer want to make an order, whey click the button which says "New Order", in the grid there will be a list of orders with date, verarity and name. The grocery store owner should see this screen as their home screen. Thereby they can see any new orders immediately in the top of the grid.

The second screen will be the manage products screen. There the owner would be able to see all products in store. In the table showing the products, one products name, units and price per unit will be listed. If a new product needs to be added to the list, the owner can simply click on the button saying "New Product", fill out the necessary details and add it to the table of products.

The tech architecture of this project is:

- Frontend: The UI will be build in HTML, CSS, JS and Bootstrap.
- Backend: The backend will be build in Python, Flask server.
- Database: will be build in MySQL

Throughout the process of building the app linters and unittests will be widely used so secure the best possible coverage and integrity. The endpoints and external API will be tested through integration tests to secure connections and security, performed with postman. The frontend will be tested with end-to-end tests, performed with Selenium webdriver. Stress performance tests will be run on the app with Apache Jmeter. Furthermore will usability tests be designed, not conducted.