S/W Development Completion Report

Project Name	Online Store		
Author	**GRUPPA**	**Approver**	Anna Kyselova
Team	3		

Contents

Revision History

Version	Date	Revised contents	Author	Approver
1	15.12.2025		GRUPPA	Anna K.

Terms and Abbreviation

[term]	*Description*
[abbreviation]	*Description*

References

[1] Author. Title. Volume Number. Edition. Publisher. Company Name. Document Title. Version Number. Date.
[2] <Committee Name>.<Standard Title>.<Version No.>.<Date>

1 Project Result

1.1 Development Period and Responsibilities

<Specify the Development Period and the Responsibilities in Table 1-1. Development Type should be selected out of System Project, S/W Research Project, Outsourcing Project, New Development Project and Up-grade Development Project.>

Table 1-1. Development Term and Responsibilities

Development Period	September – December 2025		
Development Type	New Development Project		
Responsibility	**Team**	**Position**	**Name**
Development PL	3	Author	GRUPPA
S/W Development PL	3	Approver	Anna Kyselova

1.2 Project Achievements

<Specify the basic achievements, acquired technologies, transferable technologies, patent application & future plan, society/journal contribution achievement & plan, product application & utilization of the technology and so on in Table 1-2.>

Table 1-2. Project Achievements

Item	Achievements
Release 1	**SRS creation (Software Requirements Specification)** Domain model established with core entities: Administrator, Customer, and Product. Initial operations for basic login/routing and admin product management (seeding, adding, listing) were implemented.
Release 2	**Adoption of layered architecture: Presentation (UI), Service/Logic, and Domain** UML diagrams for initial technical design. Class design and specifications for core domain entities and services (e.g., LoginService, AdminService). Planned interfaces/abstractions for decoupling, such as IProductRepository and IConsole.
Release 3	**Implementation of validation rules to enforce data integrity.** Planning for exceptional flow and error handling, including preconditions, postconditions, and initial exception policy. Introduction of the Order aggregate and OrderStatus enumeration.
Release 4	**Implementation of file loading (loadAll()) and file saving (saveAll()) for persistent storage.** Management of the memory lifecycle where data is loaded from data.txt into a dynamic array and saved back on

	shutdown. Creation of conversion helpers (parseLine() and toLine()) to serialize and deserialize domain objects (e.g., Transaction objects) to and from plain-text lines.

1.3 Development Effect and Utilization of Results

<Specify the effects of development results and the fields where the development product can be used.>

The final system is a console-based e-commerce prototype built upon a robust, three-tiered layered architecture.

Summary of Final Architecture and Technical Capabilities

- **Architecture:** Follows a layered structure: Presentation → Service/Logic → Domain.
- **Release 4 Extension:** Introduced a **Repository** responsible for managing both in-memory data and persistent data in a text file (`data.txt`) .
- **Key Capabilities:** Supports data **persistence** via `loadAll()` (file → memory) and `saveAll()` (memory → file) operations. It enforces **data validation** across all layers.

Skills Gained

The team gained practical experience in:

- **Memory and Persistence:** Connecting dynamic, in-memory data structures to a persistent file storage .
- **Data Integrity:** Implementing and applying extensive **validation rules** and an **exception handling policy**.
- **Architectural Design:** Maintaining clean separation between the UI, Logic, and Repository layers.

Possibilities for Reuse or Extension

- **Reuse Contexts:** The layered foundation is suitable for any console-based application requiring structured data and simple file persistence, such as **inventory trackers** or **management systems**.
- **Future Extensions:** The architecture is designed to easily replace the lightweight file persistence with a professional database (via planned abstractions like `IProductRepository`). Further improvements include full order management and better authentication.

2 Development History

2.1 Development S/W Structure

<Specify the directory structure of the S/W system and the size of source codes. A diagram may be used to depict the directory structure.>

The system utilizes a simple **layered architecture** with strict one-way dependencies: Presentation → Service/Logic →Domain.

Layer Responsibilities

- **Presentation (UI) Layer:** Handles program entry (`src/app/main.cpp`) and console I/O.
- **Service/Logic Layer:** Implements business workflows and routing (`LoginService`, `AdminService`), operating on in-memory data .
- **Domain Layer:** Defines core entities and data structures (`Product`, `Order`, `Customer`) .
- **Repository Layer:** Manages centralized storage, including both in-memory collections and persistence operations .

Data Flow and Evolution (Release 4)

The data flow is unidirectional: `main` starts the system and routes through Services (e.g., `AdminService`), which interact with domain objects.

The main structural change in **Release 4** was the extension of the **Repository** to handle file persistence . It introduced:

- Functions to manage the **file-to-memory lifecycle** (`loadAll()`, `saveAll()`) .
- Helper functions for data conversion (`parseLine()`, `toLine()`) to serialize and deserialize objects from the text file (`data.txt`) .

This extension allows the Services to continue working solely with in-memory data while the Repository manages the persistent layer beneath them.

2.2 Development Deliverables

<List the deliverables from each phase.>

The development process produced a comprehensive set of documents and materials across the four releases.

Documentation and Diagrams

The project was documented incrementally, reflecting system evolution:

- **Release 1** produced the **SRS (Software Requirements Specification)**.
- Subsequent releases produced multiple versions of the **DLD (Detailed Level Design)**. The final version, **R4 - SW Detailed Level Design**, covers the persistence extension.

- The system structure and operation are described by **UML diagrams**. Specifically, the key operation (Admin Add Product Flow) is documented using both an **Activity Diagram** and a **Sequence Diagram**.

Implementation Files

The final implementation deliverables include:

- The complete **Source code files** written in C++, mapping to the domain and service classes (e.g., `Product.h/.cpp`, `AdminService.h/.cpp`).
- An example file, **data.txt**, which demonstrates the file format used by the Repository for persistent data storage.

2.3 Development Environment

<Specify the Environment such as Hardware, Operating Systems, Compiler and Software in Table 2-1 and Table 2-2.>

Table 2-1. Hardware Environment

Type	Operating System	Major Use	Remarks
Laptop	MacOS	Personal device of one of the authors to work on the project	
Laptop	Windows	-||-	
Laptop	Linux	-||-	

Table 2-2. Software Environment

Type	Model & Spec.	Major Use	Remarks
Programming Language	C++	Core implementation of all layers.	
IDE	CLion	Source code editing, compilation, and debugging.	
Documentation	Google Docs	Creating the SRS and DLD documents.	
Version Control System	Git	Managing source code versions and team collaboration.	
CodeWithMe		Code editing collaboration.	CLion extension