

[Online Shop with Distributed Services] – Project Description

[Dawid Faruga(354886), Haoran Li(355420)]

[Jan Munch Pedersen , Troels Mortensen

]

[2608]

[Software Technology and Engineering]

[Semester ,3 ,2025]

[02,12,2025]

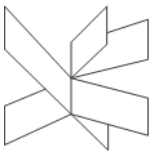


Table of content

1. Problem Domain 1

2. Problem statement..... 4

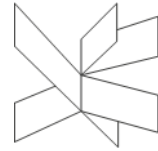
3. Delimitation 6

4. Choice of methods 8

5. Time schedule 12

6. Risk assessment..... 15

7. References 19



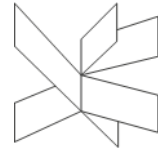
1. Problem Domain

The rapid growth of e-commerce over the past decade has transformed how consumers purchase goods and how businesses operate in the digital marketplace. Online shopping systems now serve millions of users daily and must support complex interactions involving product browsing, order placement, payment processing, inventory management, and customer support. This evolution places increasing demands on the underlying technological infrastructure, requiring systems that are scalable, reliable, secure, and capable of supporting distributed and heterogeneous services. As the complexity of digital commerce continues to grow, understanding how such systems function has become both a technical and societal priority.

E-commerce platforms depend not only on user-friendly interfaces but also on robust backend architectures that can integrate multiple services built with different technologies and communication protocols. With modern systems often adopting microservice-inspired approaches, distributed architectures have become increasingly relevant (Newman, 2015). These architectures rely on independent components communicating through APIs, message queues, or other network mechanisms, enabling flexibility but also introducing challenges in terms of interoperability, fault tolerance, and consistent data handling.

The importance of this topic is further highlighted by the increasing expectations for system responsiveness and availability. Even minor service interruptions can affect user satisfaction and business credibility (Kurnia et al., 2020). Moreover, the technological diversity in existing e-commerce infrastructures—combining web frontends, multi-language backends, and various database systems—creates a need to understand how heterogeneous components can function together as a unified system. This complexity forms a crucial part of the problem domain, particularly in educational contexts where students must learn how to design, integrate, and reflect on distributed systems using practical and relevant examples.

Within this broader domain, a common challenge arises when multiple services built with different programming languages need to communicate and behave consistently. Existing commercial e-commerce platforms such as Shopify, WooCommerce, and Magento provide integrated solutions, but they do not transparently expose or explain the underlying distributed system interactions, nor are they suitable for instructional purposes requiring hands-on implementation. For educational environments, there



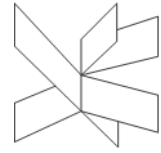
remains a need for simplified yet realistic system environments that demonstrate how distributed components can coordinate tasks such as product handling, order processing, and logging.

The specific context of this project focuses on a simplified online shop scenario in which a web frontend interacts with backend services written in different languages, such as Java and C#. These services may be responsible for tasks including processing user requests, managing data, and communicating with a database. This narrower domain highlights the engineering challenge of integrating heterogeneous components into a coherent system that reflects real-world architectural concerns. Understanding this domain requires examining how users interact with the system, how data flows through multiple services, and how distributed interactions affect system reliability and maintainability.

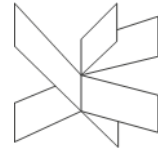
At the most specific level, the project examines the challenges associated with coordinating multiple backend services, each running independently but forming part of the same online shopping environment. The problem domain includes considerations related to communication methods, data consistency, network reliability, and the user's expectation of seamless interaction. By exploring these contextual elements, the project establishes a foundation for defining a problem statement without yet proposing a solution, in accordance with the requirements of systematic engineering analysis.

References:

1. Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.
2. Kurnia, S., Choudrie, J., Mahbub, M., & Alzougool, B. (2020). E-commerce technology adoption: A systematic review. *Journal of Business Research*, 122, 571–588.
<https://doi.org/10.1016/j.jbusres.2020.01.015>
3. Laudon, K. C., & Traver, C. G. (2021). *E-commerce 2021: Business, technology, society* (16th ed.). Pearson.
4. Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.
5. Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley.



[Title of the Project] – Project Description



2. Problem statement

Based on the previously presented problem domain, this project focuses on understanding how an online shopping system can operate effectively when built on a distributed architecture consisting of heterogeneous technologies. The system includes a web frontend and multiple backend services implemented in Java and C#, each responsible for different aspects of the system's functionality. These services must communicate reliably, manage data consistently, and support a seamless user experience despite operating across different platforms and technologies.

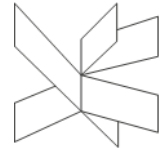
The project therefore seeks to investigate the challenges, dependencies, and interactions that arise when such a distributed system is constructed and used in an online shopping context.

Main Problem

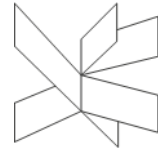
How can the interaction and coordination between a web frontend and multiple heterogeneous backend services be understood and designed so that a distributed online shopping system operates reliably and provides a consistent user experience?

Sub-questions

- What requirements govern the flow of data and interactions between the web frontend and the multiple backend services in a distributed e-commerce system?
- What technical factors must be considered when enabling communication between backend services implemented in different programming languages, such as Java and C#?



- How should the system address issues of data consistency across services, and how do these requirements influence the architectural structure of the system?
- What role does the database play in supporting functionality and data integrity within a distributed online shopping system?
- Which architectural factors influence the maintainability, scalability, and reliability of a multi-component distributed system?
- How do common risks in distributed systems—such as communication failures or service unavailability—affect an online shopping scenario, and what engineering challenges do these risks reveal?



3. Delimitation

The problem domain of e-commerce systems and distributed architectures is broad, involving numerous technical, organizational, and business-related aspects. To ensure that the project remains feasible within the given timeframe and aligns with the educational purpose of the course, specific delimitations are introduced. These delimitations clarify which parts of the overall problem space will be included in the project and which related topics fall outside its scope. By narrowing the focus, the project can concentrate on understanding and analyzing the interaction between heterogeneous distributed services within a simplified online shopping context.

Included Focus Areas

This project centers on:

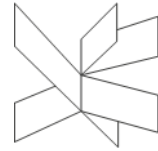
- The interaction between a web frontend and multiple backend services.
- Communication between heterogeneous backend technologies (Java and C#).
- Data handling and consistency in a distributed architecture.
- The role of a database in supporting basic e-commerce functionality.
- Architectural considerations that influence reliability, maintainability, and scalability.

These elements are directly rooted in the problem domain and form the core of the system investigated in this project.

Excluded Areas

To maintain a manageable scope, several aspects from the broader e-commerce domain will **not** be addressed in this project:

- **Full-scale business logic**, such as payment processing, authentication security layers, or complex inventory management.

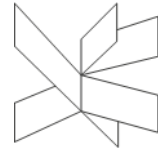


- **Advanced scalability mechanisms**, such as load balancing, container orchestration, or auto-scaling in cloud environments.
- **High-level security architectures**, including encryption standards, token-based authentication frameworks, or enterprise-level access control.
- **Front-end user experience research**, such as usability studies, accessibility standards, or interaction design methodologies.
- **Comprehensive microservice orchestration**, event-driven messaging platforms, or message-broker-based architectures.

These exclusions are made to ensure the project remains focused on the interaction of distributed, heterogeneous technical components rather than on the full operational complexity of commercial e-commerce platforms.

Justification for Delimitations

The delimitations reflect a conscious decision to focus on the foundational engineering challenges introduced in the problem domain—specifically, communication and coordination between distributed components. Exploring every area within the e-commerce landscape would exceed the practical scope of the project. Therefore, narrowing the focus enables a deeper examination of the architectural and communication-related issues that arise when integrating heterogeneous backend services with a shared frontend and database.



4. Choice of methods

This project applies a combination of theoretical, analytical, and practical engineering methods to address the problem statement and explore how a distributed online shopping system can be understood, designed, and evaluated. The methods were chosen to ensure coverage of all aspects of the problem, including knowledge acquisition, system analysis, architectural design, implementation, testing, and project management. Each method described below is directly relevant to the questions presented in the Problem Statement.

4.1 Knowledge and Data Collection Methods

Literature Review

A structured review of academic and technical literature will be conducted to gather knowledge about:

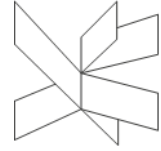
- distributed systems and heterogeneous service communication,
- architectural styles such as layered and microservice-based architectures (Newman, 2015),
- database consistency and data flow in multi-service environments,
- challenges in e-commerce system design.

The literature review supports understanding of the broader context and provides theoretical grounding for the system analysis and design decisions.

Analysis of Existing Systems

Existing e-commerce platforms and distributed backend architectures will be examined to identify:

- common patterns in system decomposition,



- communication mechanisms between services,
- typical data-handling workflows.

This method ensures that the project is informed by current industry practices while remaining within an academic scope.

4.2 Analysis Methods

System and Architectural Analysis

System analysis will be used to examine:

- the interaction between frontend and backend services,
- data flow across heterogeneous components,
- communication protocols and integration challenges.

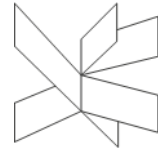
This analysis directly addresses the sub-questions concerning data consistency, system reliability, and architectural considerations.

Use-Case and Requirement Analysis

Use cases and requirements will be developed to clarify:

- the system's functional boundaries,
- actors and their interactions,
- essential processes within the online shopping scenario.

This provides a structured basis for evaluating which components are necessary for the project.



4.3 Design, Construction, and Testing Methods

Architectural Modelling

UML and system diagramming techniques will be used to model:

- component interactions,
- service responsibilities,
- communication pathways.

These models provide a visual basis for understanding coordination between heterogeneous backend services.

Prototyping

Prototypes will be built to test:

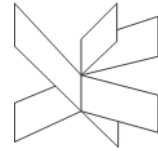
- Java–C# communication mechanisms,
- database integration,
- frontend–backend interaction flows.

Prototyping reduces architectural uncertainty early in the process, supporting the risk-driven principles of the Unified Process.

Implementation and Experimentation

The system will be implemented using:

- Java (backend service)
- C# (backend service)



- Web technologies for the frontend
- PostgreSQL database
- REST and Socket communication

The implementation itself functions as an experiment to explore how heterogeneous services behave in practical distributed environments.

Testing

Testing will include:

- integration testing between services,
- communication reliability testing,
- functional testing of data workflows,
- basic verification of user interactions.

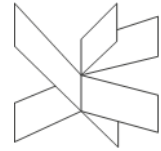
Testing ensures insight into system reliability, consistency, and architectural constraints.

4.4 Project Planning and Management Methods

Unified Process (UP)

The UP framework will guide the project through its four phases:

- Inception: understanding the problem domain
- Elaboration: reducing risk through early validation
- Construction: iterative implementation
- Transition: integration and documentation



UP supports the structure needed to gradually refine system understanding and ensure alignment with the problem domain.

Scrum

Scrum principles will be applied to manage iterative development, including:

- sprint planning,
- short alignment meetings,
- incremental reviews,
- retrospectives.

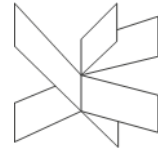
Scrum allows the team to adapt to new insights gained during the project and organize individual learning needs within a PBL context.

5. Time schedule

The time schedule presents the overall project period, major milestones, expected workload, and deadlines. It functions as a planning tool that supports both the Unified Process phases and the iterative rhythm of Scrum. Although the schedule is included in the Project Description, it is considered dynamic and will be adjusted as the project progresses.

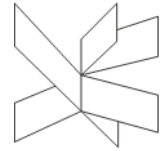
5.1 Milestones and Timeframe Overview

The table below outlines the planned timeframe, milestones, and expected deadlines.



Phase / Sprint	Duration	Key Activities	Milestones
Inception	Week 1	Understanding problem domain, initial research, defining project scope, drafting Problem Statement & Delimitation	M1: Project scope defined
Elaboration	Week 2–3	Architecture exploration, technology validation, prototyping Java–C# communication, database schema planning, requirement refinement	M2: Architecture validated & requirements approved
Sprint 1 (Construction)	Week 4–6	Backend service implementation (Java/C#), basic frontend integration, internal API definition	M3: First working system prototype
Sprint 2 (Construction)	Week 6–9	Full system integration, distributed communication, data flow testing, refinement of UI	M4: Fully integrated system
Transition Phase	Week 10–12	System testing, documentation writing, final adjustments, preparation for submission	M5: Final report and system completed
	Week 13		
Final Delivery	Around 2025 12.12	Submission of final report and project deliverables	M6: Project delivered

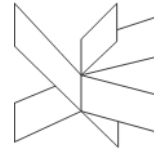
5.2 Additional Scrum-Based Time Structuring



Although the UP phases guide the overall structure, Scrum supports the weekly planning:

- **Weekly sprint planning:** define priorities for the week
- **Daily-style check-ins:** short updates on progress
- **Weekly review:** validate completed features (frontend–backend interactions, service communication, database operations)
- **Retrospectives:** reflect on workflow and adjust planning

These recurring activities ensure progress remains aligned with milestones.



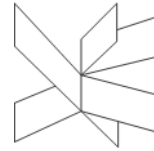
6. Risk assessment

A number of internal and external factors may disrupt the progress of the project or impede the team's ability to complete the planned activities within the given timeframe. Because this project involves a distributed architecture, heterogeneous technologies, and coordination between two team members, a structured risk assessment is necessary.

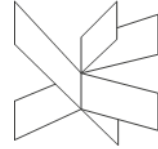
The following matrix identifies the most relevant and realistic risks for this specific project, evaluates their likelihood and severity, and outlines mitigation strategies and indicators to support early detection.

6.1 Risk Assessment Matrix

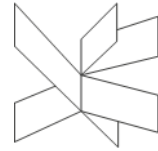
Risk	Likelihood (1–5)	Severity (1–5)	Risk Value	Risk Mitigation (Preventive & Responsive Actions)	Identifiers	Responsible
R1: Communication failure between Java and C# services	3	5	15	Preventive: Build early prototypes to validate REST/Socket communication; test integration continuously. Responsive: Switch to simplified communication or fallback	Services fail to send/receive requests; timeout errors	D



				methods if needed.		
R2: Lack of access to required test data for database development	4	4	16	Responsive: Use structured dummy data if real sample data cannot be obtained in time.	Missing tables/records ; incomplete DB queries	L
R3: Incorrect estimation of workload due to complexity of distributed architecture	3	4	12	Preventive: Re-scope tasks during sprint planning; reduce non-essential features. Responsive: Redistribute tasks based on capacity.	Falling behind sprint plan; incomplete modules	L
R4: Delays caused by unfamiliar technologies (C#, REST APIs, Web communication)	2	5	10	Preventive: Allocate learning time early in the schedule. Responsive: Simplify implementation if delays	Slow progress on integration tasks; repeated errors	L



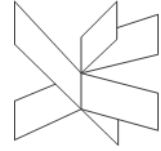
				become critical.		
R5: Team member unavailability due to illness or personal issues	2	4	8	Preventive: Document code and decisions thoroughly; keep tasks modular. Responsive: Reassign responsibilities and adjust scope.	Missed meetings; unusual downtime	D
R6: Version control or merge conflicts affecting project progress	3	4	12	Preventive: Establish clear Git workflow. Responsive: Roll back to stable commit; perform pair review before merging.	Conflicting files; broken builds	L
R7: Difficulty integrating frontend with distributed backend services	3	3	9	Preventive: Test API endpoints early with mock tools (e.g., Postman). Responsive: Temporarily	Frontend cannot fetch data; repeated HTTP errors	D



				mock missing services.		
R8: Falling behind schedule due to underestimated documentation workload	2	4	8	Preventive: Start documentation in parallel with development; maintain working drafts. Responsive: Reduce detail in non-essential sections.	Documentation backlog; missed internal deadlines	D

Explanation

- **Likelihood** represents how probable it is that the risk will occur.
- **Severity** indicates how strongly the risk impacts the project's ability to be completed.
- **Product** (Likelihood × Severity) helps identify which risks require the most attention.
- **Mitigation** includes both *preventive* and *responsive* actions.
- **Identifiers** help the team detect early warning signs so risks do not escalate.
- **Responsible** assigns one group member to monitor each risk and take action when needed.



7. References

1. Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.
2. Kurnia, S., Choudrie, J., Mahbub, M., & Alzougool, B. (2020). E-commerce technology adoption: A systematic review. *Journal of Business Research*, 122, 571–588. <https://doi.org/10.1016/j.jbusres.2020.01.015>
3. Laudon, K. C., & Traver, C. G. (2021). *E-commerce 2021: Business, technology, society* (16th ed.). Pearson.
4. Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley.
5. Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley.