Operimus*

A novel, modular and event-driven enterprise operations management

J. R. Almeida, L. Bastião, P. Almeida
v18.09.2025

** Name generated with ChatGPT to add realism to the project.*

Index

1 Project objectives 2

1.1 Project assignment objectives 2

1.2 Team and roles 2

2 Project specification 4

2.1 Product technical details 4

2.1.1 Project requirements and technical specifications 4

2.1.2 API documentation 4

2.2 Required practices 4

2.2.1 Agile *backlog* management (plan & track) 4

2.2.2 Feature-branching workflow 5

2.2.3 Containers-based deployment 5

2.2.4 Versioning and release management 5

2.2.5 Project repository (Git) 5

2.3 Project schedule: iteration plan 6

2.3.1 Sprints planning 6

2.3.2 Deliverables 6

2.3.3 Expected results 7

2.3.4 Sprints guidelines 7

3 Project topics/components 8

1 Project objectives

1.1 Project assignment objectives

This project involves building a modular enterprise solution, where each group is assigned to develop a distinct component. Each component must be designed as a standalone application capable of operating independently. However, because the overall system is integrated, some components will depend on information from others to function correctly.

To facilitate communication and data sharing between these components, they will use a message queue system with a predefined message format and protocols. This means components will publish events and subscribe to messages relevant to their function, enabling real-time, asynchronous interaction across the system.

Since groups may not have their dependent components ready at the start, each team is required to create mock messages that simulate real data from other components (only the required messages). This allows them to develop and test their features without waiting for the entire system to be completed. These mock messages should be clearly identified as such to avoid confusion. Further technical details, including message formats and communication protocols, are described in Section Product technical details.

During the project, some requirements will change, be added, or deprecated. Therefore, the project environment is highly volatile. It is essential to adopt a project management methodology that can effectively accommodate and respond to constant changes in the final solution.

1.2 Team and roles

Each team/group should assign the roles described in Table 1. Note that the team should perform the activities collaboratively. The idea of *roles* is to have people in the team to lead a specific part of the work and act as a spokesperson for that specific area.

Table 1: Selected roles for the software development team.

Role	**Key responsibilities**
Scrum master (coordinator)	Actively promote the best collaboration in the team and take the initiative to address problems that may arise. Ensure the necessary discussion so there is a fair distribution of tasks and that members work according to the plan. Ensure that the requested project outcomes are delivered in time.

Role	Key responsibilities
Product owner	Represents the interests of the stakeholders. Has a deep understand of the product and the application domain; the team will turn into the Product Owner to clarify questions about product features/requirements. Responsible for accepting the solution increments.
QA engineer	Collaborates with the entire team to build quality into the product from the start. Works with the Product Owner to define acceptance criteria for new features. Responsible for representing the Q/A decisions on the project.
Service Analyst	Partners with the team to ensure the operational health and reliability of the services. Knows which monitors key performance indicators (KPIs) and Service Level Objectives (SLOs) were stablished for the project. Coordinates incident response and performs root cause analysis to prevent future issues.
DevOps engineer	Supports the team in delivering and operating software more efficiently, including the CI/CD pipeline. Takes responsibility for decisions on Infrastructure as Code (IaC) to manage and provision environments. Ensures the infrastructure and applications are portable, scalable, and reliable across different environments.
Developer	**ALL members contribute to the development tasks.**

2 Project specification

2.1 Product technical details

Each group is expected to propose, conceptualize, and implement a multi-layer, enterprise-class solution observing the guidelines in the following sections.

2.1.1 Project requirements and technical specifications

The project documentation should be kept in the main branch of the repository ([main]/reports) and updated accordingly. The requirements should be created only on the board as GitHub issues.

The "Project Specification Report" [see sample template on the course page] should cover:

A. Product concept
   A.1. Vision statement
   A.2. Personas
   A.3. Supported scenarios (user stories)
B. Architecture notebook
   B.1. Key quality requirements
   B.2. Architectural view
   B.3. Module interactions [dynamic view]
C. Information model

2.1.2 API documentation

Deliver (and maintain) an autonomous, self-contained report (or an interactive web page) that fully describes the service's API. The documentation should present a clear overview of the API's purpose and structure, detail the available endpoints/methods (including request parameters, response formats, and possible error codes), and outline the expected usage patterns. Where applicable, include examples of typical requests and responses to illustrate correct implementation. Consider using the OpenAPI/Swagger framework to create the API documentation.

2.2 Required practices

2.2.1 Agile *backlog* management (plan & track)

The project will use a shared backlog to prioritize, assign and track the development work. Each team should collaborate for the same board, using the defined tags to simplify the coordination.

This backlog follows the principles of “agile methods” and use the concept of “user story” as the unit for planning. Stories are briefly documented declaring the benefit that a given *persona* wants to get from the system.

Stories have points, which “quantifies” the shared expectation about the effort the team plans for the story, and prioritized, at least, for the current iteration. Developers start work on the stories on the top of the current iteration queue, adopting an agreed workflow.

The backlog management should be done using GitHub Projects + Scrum boards. The backlog should be consistent with the development activities of the team; both the project management environment and Git repository activity log should provide faithful evidence of the teamwork. It will be provided by the teaching staff.

2.2.2 Feature-branching workflow

There are several strategies to manage the shared code repository, and you are required to adopt one in your team. Consider using the “GitHub Flow”. The **feature-branches should be traceable to user-stories** in the backlog. Complement this practice by issuing a “pull request” (a.k.a. merge request) strategy to review, discuss and optionally integrate increments in the *main*.

2.2.3 Containers-based deployment

The first version of the MVP (Minimum Viable Product) must follow the principle of **separation of responsibilities**. This means that each service should be deployed in its own specialized container (e.g., Docker containers), ensuring clear boundaries between functional components. In most cases, these containers will correspond to the logical layers of your system architecture (e.g., presentation layer, business logic layer, data layer). The solution must run across **multiple containers** rather than a single monolithic instance.

2.2.4 Versioning and release management

Version tags should be applied to stable releases, following a **Semantic Versioning** scheme (`MAJOR.MINOR.PATCH`), allowing clear identification of production-ready versions and simplifying rollback procedures if necessary. The **MAJOR** version number is determined by the sprint index plus one (see Table 2). For example, the release after sprint 3 will have the version **2.x.x** of the software. The **MINOR** number is incremented for new features delivered within the same major cycle, and the **PATCH** number is incremented for corrections or small improvements.

2.2.5 Project repository (Git)

All project deliverables must be stored and maintained in a cloud-based Git repository, serving as the central source of truth for the team’s work. In addition to the source code, the repository should contain all other required project outcomes, such as technical documentation, diagrams, test results, and any other materials

explicitly requested during the course. It should be created using the provided GitHub Classroom link: https://classroom.github.com/a/VvF-EGdy.

The repository name must follow the standardized naming convention: *ES2526_<TP><Group ID>,* where <TP> corresponds to the practical class identifier, and <Group ID> is the assigned group number. For example, group 1 from TP1 must use the repository name: *ES2526_101*.

Teams should ensure the repository is consistently updated, well-organized (using a clear folder structure), and always reflects the current state of the project, as this will be used as primary evidence of team progress and collaboration.

2.3 Project schedule: iteration plan

The project will be developed in 2-week iterations. Active management of the product backlog will function as the main source of progress tracking and work assignment.

2.3.1 Sprints planning

The tasks for each sprint are planned and defined during the class sessions, forming the official sprint backlog. Any features or work developed outside of this agreed sprint plan will not be counted towards the sprint's deliverables, regardless of their quality or usefulness. This ensures that the team follows the project methodology strictly and maintains focus on the agreed priorities.

Throughout the sprint, the team is encouraged to identify and propose new user stories or tasks, but these should be added only to the overall product backlog for future consideration. Adding new work directly into the ongoing sprint is not allowed, as it can disrupt the sprint focus and goals. Always remember that your group is just one piece of a much larger machine.

2.3.2 Deliverables

Each iteration should be associated to a git tag. For instance, iteration I1 should belong to tag 1.x.x. The release should have the changelog with the contributors of each feature/issues/pull request associated. Example:

```
### Added

- **New user interface**: Description (#PR, contributed by First_Last_Name).

### Changed

- **Enhancement of user interface**: Description (#PR, contributed by First_Last_Name).
```

Multiple releases can occur between sprints, but the sprint number should always be reflected in the leftmost part of the version tag. For example, during sprint 5, all release tags should begin with 5.

2.3.3 Expected results

Expected results from project iterations are described on Table 2.

*Table 2: Expected results for each sprint **(at project initiation).***

Sprint ID	Submission deadline	Objectives
1	01/oct	Establish the technical backlog, implement the continuous integration process, define core user stories, and design the component architecture.
2	15/oct	Deliver the first MVP, incorporating initial user stories
3	29/oct	Release software version 2.x.x, introduce the first iteration of testing strategies, include new requirements and integrate the solution into the ecosystem.
4	12/nov	Release software version 3.x.x, including tests for new features, the introduction of Infrastructure as Code (IaC), and the first version of Service Level Objectives (SLOs), while addressing new client requirements.
5	26/nov	Release software version 4.x.x, adding system observability, enhancing Service Level Objectives, and integrating new client requirements.
6	10/dec	Release software version 5.x.x, create the environment for post-mortems, and apply final refinements to the system.
7	16/dec*	Final presentations and deliver the technical report.

* *The last sprint has a smaller cycle.*

2.3.4 Sprints guidelines

Sprint review meetings will take place during class, on the day following the defined sprint deadline. In each meeting, students are expected to present, the **iteration backlog** of the project and the **sprint objectives** that have been implemented, so they can be validated by the client (*i.e.,* the teaching staff).

3 Project topics/components

The main goal of this project is to build a modular, event-driven enterprise operations management system. The project is divided into several distinct topics, and each group is required to select and focus exclusively on one specific topic. As development progresses, interdependencies between the various topics will naturally arise, requiring coordination and communication among the groups. Since the number of topics exceeds the number of groups, it is possible that some topics may not be assigned. The possible topics are the following:

1. Inventory & warehouse management
2. Vehicle & parking management
3. Risk & threat modelling platform
4. People management system (Staff and visitors)
5. eCommerce platform
6. Marketing automation platform
7. Payment & billing system
8. Shipping & logistics management

Further details for each topic are provided in the appendix.