TQS: Project assignment guidelines

v2025-05-13

1 Introduction

1.1 Project assignment objectives

Students should work in teams to implement a medium-sized software project, applying Software Quality Assurance (SQA) principles and practices. Groups are expected to:

- Specify and deliver a Minimal Viable Product (MVP) applying software enterprise architecture patterns (specifically, Jakara based frameworks, such as Spring Boot).
- Design and apply a SQA of a strategy, applied throughout the software engineering process.
- Apply the engineering practices of DevOps, including Continuous Testing (CT), Continuous Integration (CI) and Continuous Delivery (CD).

The project should use the technologies used for labs, namely the Spring Boot framework for backend implementation.

1.2 Assessment criteria

The project results will be assessed using the following criteria:

Topic	Description	Assessment Items
Goals	How well does the team meet the intended learning objectives and project requirements?	Integration of SQA and DevOps practices (CI/CD/CT) Relevant user stories defined and delivered (MVP) Agile methodology applied.
Complexity	How well were the technical challenges addressed? (breadth and sophistication of implementation and tools integration)	Robust integration of SQA tools and solutions Scope and ambition of the MVP Appropriate use of architecture/design patterns Integration of multiple systems or services (product) Handling of non-trivial edge cases or requirements

Effort	Participation, engagement and teamwork across the development lifecycle	Evidence of regular collaboration and task distribution Contribution to iterative development and testing cycles

2 Product concept

2.1 Business scenario

Nikola EV Services (NikEV)[1] is a forward-thinking technology company focused on optimizing the electric vehicle (EV) ecosystem through its cutting-edge platform for seamless charging interoperability. The company addresses one of the key pain points in the EV adoption, fragmented charging services, by offering a unified digital solution that integrates real-time searching, booking, and payment functionalities across diverse charging networks.

Key actors include:

- EV Driver: searches for chargers, books slots, charges the vehicle, makes payments, plan trips,...
- Station Operator: manages charging station availability and maintenance, monitor daily operations, configure off-peak charging slots and discounts,...
- Third-party Services: Geolocation/maps, payment gateways, charging station APIs.

Possible Epics:

- Station Discovery: locate and filter for chargers.
- Slot Booking & Scheduling: book time slots, prevent double bookings/overbooking.
- Charging: unlock chargers; consumption accounting.
- Payment Integration: Pay-per-use or subscription models.
- User Profiles & Charging History: View past usage and stats.

2.2 Features scope and MVP

Building on the business scenario proposed, you are expected to identify, specify and prioritize the platform requirements.

Minimal product scope

You can take alternative options, providing you meet the requested MVP scope:

- Driver services: search for charging stages, book a slot, unlock and use. Dashboard to monitor the personal electric charging consumption.
- Backoffice services: register charging stations, maintenance (change availability), monitor consumptions.
- Map view for aggregate stats.
- Payment system integration: simulated or using sandbox services.

Extended features

You can enrich your platform with additional services/features to add value to consumers and promoters. Here are some suggestions (in no order), but you are welcome to bring your own:

- Create a testbed for other projects, simulating realistic services to be expected from the EV station API (i.e., provide “virtual stations” that other projects could use for an integration testbed).
- Rich visualizations. Integrate dashboard-style visualizations; allow users to quickly grasp habits, consumptions, etc.

[1] This is a fictional company. In your project, **use other** scenario/company.

- Research actual EV stations API/integration protocols (e.g.: OCPI, OCPP). Use these requirements to design your “connectors”.
- Monitor/forecast the occupation level of stations so you can plan ahead. This can be based on recent data for the station or typical usage for the period.
- Monitoring the ecological CO2 footprint. Allow drivers to visualize CO2 savings (compared to petrol-based engines) for different time periods.

3 DevOps and SQA practices checklist

3.1 Agile Project Management

Agile project management uses iterative development (key value is identified as epics, the project is divided into sprints that deliver a focused increment), tracks work as user stories (small but recognizable product features) and adapts/prioritizes according to business value.
Stories have points, which reveal the shared expectation about the effort the team plans for the story, and are prioritized, at least, for the current iteration. Developers adopt an agreed workflow.
Stories should also be used as units for feature-branching and acceptance.

Practices (PR) checklist:

PR1. Use an **agile project management environment** (Scrum-like): project backlog, sprints backlog and (current) iteration boards (e.g., Jira)
PR2. **Adopt user stories** as the granularity for work items. Define story points and acceptance criteria for each story.
PR3. **Integrate project management and git repository** (exchange meaningful events seamlessly, e.g. feature branching).
PR4. **Track progress** with project reporting tools, e.g., burndown charts, velocity,…
PR5. Include **requirements coverage** **assessment** by integrating a test management system in the planning environment (e.g.: Jira + Xray).

3.2 QA: Testing

Developers are expected to deliver both production and testing code. The “definition of done”, establishing the required conditions to accept an increment from a contributor should define what kind of tests are required. Not all kinds of tests apply all the time; you should develop a robust, yet practical, testing strategy.

Practices checklist:

PR6. Include acceptance criteria in user stories and link to BDD automation when possible, using structure scenarios specification (Gerkin).
PR7. Apply test-driven development (TDD) where feasible (develop tests before final code).
PR8. Implement tests for business use cases and business logic. While business relevant use cases are likely to be end-2-end tests, business logic will likely originate unit tests.
PR9. Run integration tests for inter-service communication. Segregate integration tests and end-2-end tests (not to be executed in all builds).
PR10. Define Service Level Objectives and include non-functional testing, especially performance/load testing for critical features (e.g., K6).
PR11. If you have a scenario with multiple input variables and their respective values, and you foresee some risk related with these variables and their interaction, consider testing combinations of these variables, using pairwise-testing (e.g. you can use a tool such as PICT)

3.3 QA: Code quality

PR12. Use static code analysis tools (e.g., SonarQube). Prepare a code analysis dashboard for continuous quality assessment (e.g.: SonarQube dashboard). Include code coverage analysis (e.g., JaCoCo)
PR13. Include assessments of vulnerabilities in all builds (e.g.: force in quality gates)
PR14. Conduct peer code reviews using pull requests.
PR15. Provide effective guidelines for contributors (code style). This is not to be an extensive document, but rather a selection of key shared practices (point to other complementary references).

3.4 DepOps practices

PR16. Use a feature-branching Git workflow that should be coherent with user stories (e.g.: Jira + Git integration; Git events update Jira status).
PR17. Enable branch protection rules and merge only via pull requests.
PR18. Set up a CI pipeline. Automate builds and run tests on every commit. Enforce blocking Quality Gateways as part of the CI process.
PR19. Configure automatic feedback for building failures (e.g.: integrate IDE and CI engine). Configure build status badges.
PR20. Use infrastructure-as-code (e.g., Docker) to describe the production environment.
PR21. Set up automatic deployment to different environments (staging, production).

3.5 Additional practices

PR22. Use an observability framework to monitor services continuity. Provide dashboards and alarms on the operation conditions of infrastructure and deployed services.

4 Implementation schedule

4.1 Team and roles

All students need to contribute as *developers* to the solution. Each team/group should assign additional roles:

Role	Responsibilities
Team Coordinator (a.k.a. Team Leader)	Ensure that there is a fair distribution of tasks and that members work according to the plan. Actively promote the best collaboration in the team and take the initiative to address problems that may arise. Ensure that the requested project outcomes are delivered on time.
Product owner	Represents the interests of the stakeholders. Has a deep understanding of the product and the application domain; the team will turn to the Product Owner to clarify the questions about expected product features. Should be involved in accepting the solution increments.
QA Engineer	Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure que quality of the deployment. Monitors that team follows agreed QA practices.
DevOps master	Responsible for the (development and production) infrastructure and required configurations. Ensures that the development framework works properly. Leads the preparing the deployment machine(s)/containers, git repository, cloud infrastructure, databases operations, etc.
Developer	ALL members contribute to the development tasks which can be tracked by monitoring the pull requests/commits in the team repository.

4.2 Reference iterations plan

The project will be developed in *1-week iterations*. Active management of the product backlog will be the main source of progress tracking and work assignment. Each "Prática" class will be used to monitor the progress of each iteration; column on the right lists the minimal outcomes that you should prepare to show in class.

Expected project iterations:

Iter. # (start)[2]	**Show** (results to be presented in Práticas)	**Start** (main focus/activities for the iteration)
I0 < 8/5	n/a	• Define the product concept. Work on Personas, main scenarios, and expected Epics. • Team resources setup: code repository, collaborative documents space, ... • Start backlog (JIRA)
I1 8/5	Outcomes from I0: • Product concept (overview of epics and stories)	• Define system architecture. • Define the SQE tools and practices (initial version). • CI Pipeline (initial version). • Product specification report (draft version) • Backlog management system setup.
I2 15/5	Outcomes from I1: • Software/system architecture proposal. • Main elements of the SQA strategy (planned and/or configured)	• A couple of core user stories detailed and implemented. • Initial API and repository. • CI pipeline (full featured) • QA Manual (report) • Test management environment & automation.
I3 22/5	• Product increment with (at least) a couple of core user stories. • CI Pipeline, QG enforcement and merge-request policy. • Requirements *x* tests traceability.	• Set up the CD pipeline. • Services API (complete). • User stories involving data access & persistence (for customers and staff).
I4 29/05	• Comprehensive REST API. • CD Pipeline: services deployed to containers (or cloud). • Quality dashboards (XRay, Sonar)	• Stabilize the Minimal Viable Product (MVP). • All deployments are available on the server. • Relevant/representative data included in the repositories (not a "clean state"). • Non-functional tests and systems observability.
I5 05/6	• Minimal Viable Product (MVP), deployed in the server (or cloud). • Oral presentation/defense.	n/a

4.3 Project deliverables

Git repository

A cloud-based Git repository for the project code and reporting. The main *submission method will be the git repository*.

Besides the code itself, teams are expected to include other project outcomes, such as requested documentation. The project must be shared with the faculty. Expected structure for main repo:

[2] The dates are the start date for the iteration, for P1 and P2. For P3 and P4, with classes on Friday, add one day.

README.md
docs/
projX/
projY/

...with the following content:

- docs/ → reports should be included in this folder, as PDF files, especially the QA Manual and the Project Specification report. Presentation support should also be copied here.
- projX/ → the source code for subproject "projX", etc.
- **README.md** → be sure to include an informative README with the sections:
  a) Project abstract: title and concise description of the project.
  b) Project team: students' identification (and the assigned roles, if applicable).
  c) Project bookmarks: link **all relevant resources** here and, at least, the links to Project Backlog, Related repositories (if applicable), API documentation, static analysis dashboard.

For certain CI/CD pipelines, it could be better to use more than one repo, i.e., specific git repositories for different projects/modules[3]. However, this can increase the complexity of project management integration. For most projects, modules (sub-projects) that are managed and deployed "together" can be kept in a single repo.

Report: technical specifications
A brief report with requirements analysis and proposed architecture.
The report can be incrementally developed and should be available in the /docs folder of the project repository.
[Template available]

Report: QA Manual
Report on the SQA strategies, practices and tools defined for the project. It should also include evidence of the working solutions put in place.
The report is expected to be developed incrementally developed; the updated version should be available from /docs folder. [Template available]

[3] In that case, consider using an organization and a "main repository" and be sure to link related repositories in the README.md.