TQS: Project Assignment Guidelines

I. Oliveira, J. S. Pinto v2020-05-04 [work in progress...]

1 Project objectives and iterations

1.1 Project assignment objectives

Students should work in teams to specify and implement a medium-sized project, comprising a multi-layered, enterprise web application and a *Software Quality Assurance* (SQA) strategy, applied throughout the software engineering process.
The results should include:

- Axis I: software product (functional specification, system architecture and implementation).
- Axis II: *Software Quality Environment* (SQE).

1.2 Team and roles

While the team may assign specific roles to each member, all students are required to contribute as *developers* to the solution.
Each team/group should assign the following roles:

Role	Responsibilities
Team manager	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 in due time.
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 the questions about expected product features. Should be involved in accepting the solution increments.
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.

1.3 Iterations plan

The project will be developed in 1-week iterations. Active management of the product backlog will act as 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 results from project iterations:

Iter. #	Iteration focus/activities	Outcomes to be presented in class (Práticas)
I1	• Project initiation (define the concept, setup the tools, validate the approach). • Define the product architecture. • Define the SQE tools and practices. • Product specification report (draft version; product concept)	• Present the product concept. • Software/system architecture proposal.
I2	• Backlog management system setup. A few core stories defined and prioritized. • Team repository in place. • QA Manual (draft version) • Product specification report (architeture)	• UI prototypes[1] for the core user stories. • Present your SQE strategy. • Next iteration planning (user stories to address)
I3	• Develop a few core user stories involving data access. • Set up CI pipeline. • QA Manual (final)	• Increment covering (at least) a couple of core user stories: search + buy/order. • CI Pipeline. • Next iteration planning (user stories to address)
I4	• Set up CD pipeline. • Product specification report (API).	• Comprehensive REST API. • CD Pipeline: services deployed to containers (or cloud). • Product owner reviews priorities.

[1] These "prototypes" would be early versions of the web pages, already implemented with the target technologies (and not mockups) but more or less "static" (not yet integrated with the bizz logic).

Iter. #	Iteration focus/activities	Outcomes to be presented in class (Práticas)
I5	Stabilize the Minimal Viable Product (MVP). • Stabilize the presentation layer (for end-users) • Stabilize the production environment. • Update documentation (specifications and product documentation).	• MVP backend deployed in the server (or cloud); relevant/representative data included in the repositories (not a "clean state"). • Oral presentation/defense.

2 Product definition

2.1 General requirements of the *online marketplace*

An *online marketplace* system is an e-commerce solution in which associated partners (vendors) publish their services/products and an operator processes the transactions (the "platform" promoter). The promoter provides the technological platform for aggregation/intermediation and charges a commission on the transactions carried out. End-users (clients) use the platform to search and compare products/services in an integrated way.

Each project should implement **a specialized internet marketplace** with a specific value proposition for its Promoter. [need related examples?]
Teams should develop requirements/use cases for their case study and propose a solution considering the architectural constraints defined in section 2.2.

2.2 Architectural constraints

The following components should appear in the overall architecture of the solution (other modules can be divided or added to the solution):

- **Backend** implemented with Spring Boot/Java EE technology. The *backend* includes the business logic, persistence (database) and at least one REST API (can provide additional services, e.g. *streaming)*
- **Web app** that corresponds to the main interface of the marketplace, online. This application is intended for end users (clients) in general.
- **External application** that consumes the services API. It can be one or both of the following situations:
  - (a) the Mobile application intended for the final consumer (a simplified version)
  - (b) an External Client that would represent a system existing in the Partner to implement a *Business-to-Business* scenario. For example, in a platform similar to booking.com, the sample "External Client" would represent an application that is already used by hotel owners in their day-to-day life and should be integrated with the new platform. This module should be greatly simplified and function essentially as a service API client that demonstrates a system integration scenario.

3 QA and development practices

3.1 Practices to implement

3.1.1 User-stories *backlog* management

The team will use a backlog to prioritize, assign and track the development work. This backlog follows the principles of "agile methods" and should use the concept of "user story" as the unit for planning.

Stories have points, which reveal 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.
Stories have acceptance criteria. This will provide the context/examples to write user-facing tests (or inspire othrs).
The backlog should be consistent with the development activities of the team; both the backlog tools and Git repository activity log should provide a faithful evidence of the teamwork.
For more information on user stories, check this FAQ on story-oriented development (note: Acceptance Criteria **is required**).

3.1.2 Feature-branching workflow with code reviews

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"/feature-driven workflow or the "Gitflow workflow".
The feature-branch should match the user-story in the backlog.
You may complement this practice by issuing a "pull request" (a.k.a. merge request) strategy to review, discuss and optionally integrate increments in the *master*. All major Git cloud-platforms support the pull-request abstraction (e.g.: GitHub, GitLab, Bitbucket).

3.1.3 Software tests

Software developers are expected to deliver both production code 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.
There should be traceability between the user story acceptance criteria and the tests included in an increment.
Projects should provide evidence if they user TDD practices (recommended).

3.1.4 CI/CD pipeline

The team should define, setup and a adopt a CI/CD pipeline. The project should offer evidence that the contributions by each member go through a CI pipeline, explicit including tests and quality gates assessment (static code analysis). The team should also offer CD, either on virtual machines or in cloud infrastructures.

3.1.5 Containers-based deployment

Your logical architecture should apply the principle of responsibility segregation to identify layers. Accordingly, the deployment of services should separate the services into specialized containers (e.g.: Docker containers).
Your solution needs to be deployed to a server environment (e.g.: Cloud infrastructures) using more than one "slice".

3.2 Project outcomes/artifacts

3.2.1 Project repository

A cloud-based **Git repository** for the project code. Besides the code itself, teams are expected to include other project outcomes, such as requested documentation. The project must be shared with the faculty staff.
Expected structure:

```
README.md
reporting/
projX/
projY/
```

...with the following content:

- reporting/ → reports should be included in this folder, as PDF files, specially the QA Manual and the Project Specification report.

- projX/ → the source code for subproject “projX”, etc.
- **README.md** → be sure to include the sections:
  a) Project abstract: title and brief description of the project features.
  b) Project team: students’ identification and the assigned roles.
  c) Project bookmarks: link **all relevant resources** here
     - Project Backlog (e.g.: Pivotal Tracker)
     - Related code repositories (there may be other repositories...)
     - API documentation
     - Static analysis (results dashboard)
     - CI/CD environment (results dashboard)
     - ...

3.2.2 Project reporting and technical specifications

The project documentation should be kept in the master branch of the repository and updated accordingly.
There are two main reports to be (incrementally) prepared:

1. Product Specification report [→ template available]
2. QA Manual [→ template available]

3.2.3 API documentation

Provide an autonomous report (or a web page) describing the services API. This should explain the overall organization, available methods and the expected usage. See related example.
Consider using the OpenAPI/Swagger framework to create the API documentation.

3.3 Extra credit

The following challenges are not mandatory, but will give you extra credits in the project:

- A cyber-physical component (e.g.: sensors or actuators connected to a RPi board) to show the current state of the build/pipeline.
- Instrumentation to monitor the production environment (e.g.: Nagios alarms, *ELK stack* for application logs analysis,...)