TQS: Project Assignment Guidelines

I. Oliveira, v2023-04-21

1 Project objectives and iterations

1.1 Project assignment objectives

Students should work in teams to specify and implement a medium-sized project, comprising:

- Development of a viable MVP (functional specification, system architecture and implementation), applying software enterprise architecture patterns.
- Specification and enforcement of a *Software Quality Assurance* (SQA) strategy, applied throughout the software engineering process.

1.2 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 in time.
Product owner	Represents the interests of the stakeholders. Has a deep understand 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.

1.3 Iterations plan

The project will be developed in 2-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. # (start)	**Main focus/activities for the iteration**	**Results to be presented in class (*Práticas*)**
I1, prep 20/04	• Define the product concept. • Outline the architecture. • Team resources setup: code repository, shared documents, team development workflow,... • Launch backlog.	n/a
I2 04/05	• Define system architecture. • Define the SQE tools and practices. • Product specification report (draft version; product concept & requirements section) • Backlog management system setup. • UI prototyping (partial)	• Product concept. • Software/system architecture proposal.
I3 11/05	• A few core stories detailed and implemented. • QA Manual • CI Pipeline.	• Present your SQE strategy. • UI prototypes[1] for the core user stories. • Plan for next iteration: user stories to implement.

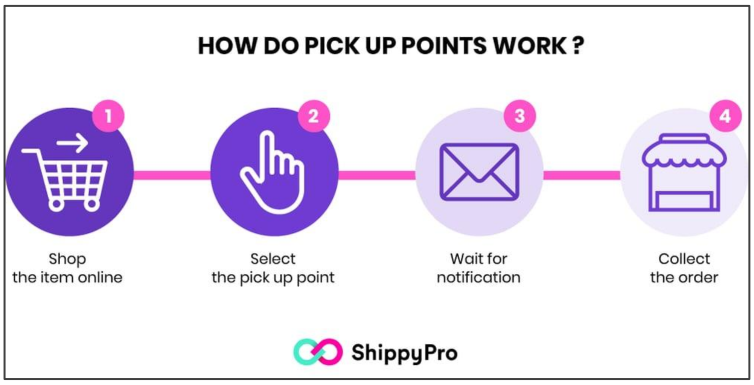
[1] These "prototypes" would be early versions of the web pages, already implemented with the target technologies (and not mockups) but more or less "static" (not yet integrated with the backend/business logic).

Iter. # (start)	**Main focus/activities for the iteration**	**Results to be presented in class (*Práticas*)**
I4 18/05	• Develop a few core user stories involving data access & persistence. • Comprehensive API. • Set up the CD pipeline.	• Product increment with (at least) a couple of core user stories. • CI Pipeline, QG and merge-requests policy. • Plan for next iteration: user stories to address and implement.
I5 25/5	• Stabilize the Minimal Viable Product (MVP). • All deployments available in the server. • Relevant/representative data included in the repositories (not a "clean state"). • Product specification report (final version.) • Non-functional tests and systems observability.	• Comprehensive REST API. • CD Pipeline: services deployed to containers (or cloud). • Quality dashboard.
01/06	• Minimal Viable Product (MVP) backend deployed in the server (or cloud).	Oral presentation/defense. P1 → 1/06 P2, P3 → 2/6.

2 Product definition

2.1 Business scenario

To make the projects comparable across team (scope and difficulty) there are general guidelines for the project should do. Overall theme: **a network of pickup points as a service**.



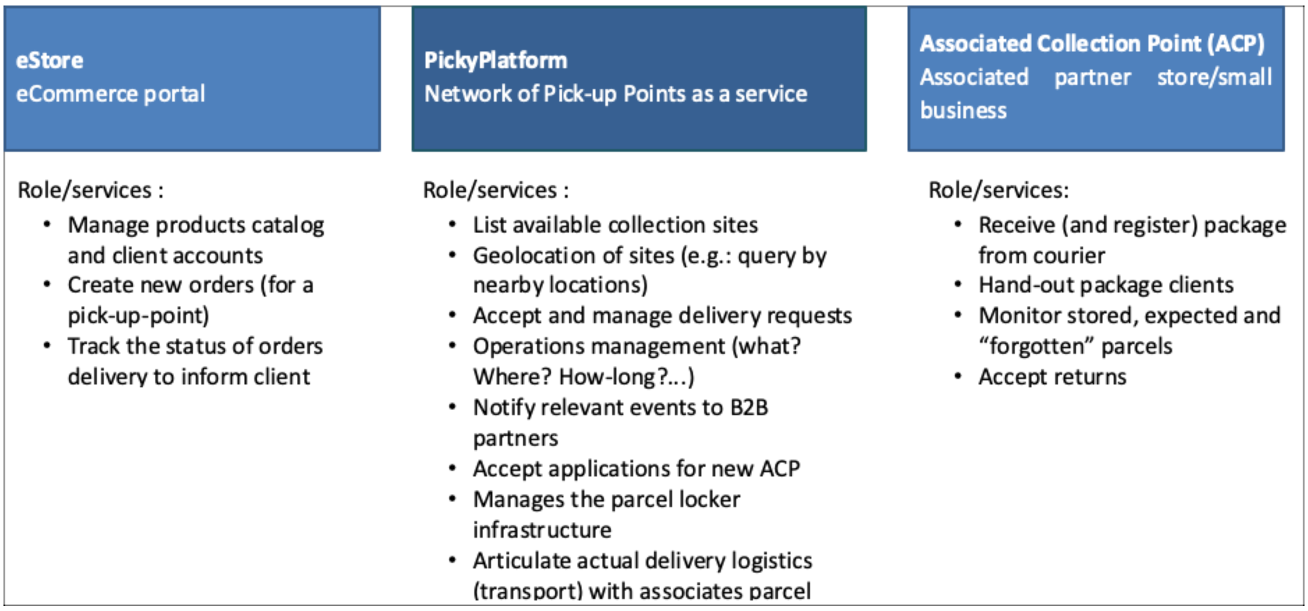
[2]

The Picky[3] platform works with two withdrawal modes to better fit the customers' expectations:

A) Associated collection points (ACP): a network of partners like the neighborhood grocery store or the laundry near home.
B) Parcel lockers: they are often located in public places (shopping centers, railway stations) and even private (campus of a big company), in general available 24/7. The buyer gets a notification with the delivery information (e.g.: location, unlock code) and can go to collect his order.

[2] Picture from https://www.blog.shippypro.com/en/advantages-drop-off-points-for-ecommerce

[3] Fictious name for the pickup points network. Don't name your own as Picky 😉



eStore
eCommerce portal

Role/services :
- Manage products catalog and client accounts
- Create new orders (for a pick-up-point)
- Track the status of orders delivery to inform client

PickyPlatform
Network of Pick-up Points as a service

Role/services :
- List available collection sites
- Geolocation of sites (e.g.: query by nearby locations)
- Accept and manage delivery requests
- Operations management (what? Where? How-long?...)
- Notify relevant events to B2B partners
- Accept applications for new ACP
- Manages the parcel locker infrastructure
- Articulate actual delivery logistics (transport) with associates parcel

Associated Collection Point (ACP)
Associated partner store/small business

Role/services:
- Receive (and register) package from courier
- Hand-out package clients
- Monitor stored, expected and "forgotten" parcels
- Accept returns

Figure 1: Overview business participants and their roles.

The overall business scenario is illustrated in the following hypothetic example (Figure 1):
- Fairy eStore sells premium fantasy costumes in the web. To simplify the deliveries and increase convenience for its customers, Fairy decided to work with a network of pickup points. For that, Fairy subscribed to Picky Platform services, which offers competitive prices and extended cities coverage.
- At checkout, Fairy's clients may choose a pickup point (operated by Picky) to collect their orders.
- When the courier delivers the package to the pickup point, a confirmation is recorded in the system and the Fairy's client gets a notification informing to collect the order.
- The customers go to the pickup point and collect their packages, providing some "token" for security and/or ID for security.

Note that this is an overview of the business context; **your group can adapt** and introduce different flavors/innovations, especially if you are adding value to the basic scenario.

2.2 Scope and technologies constraints for the system

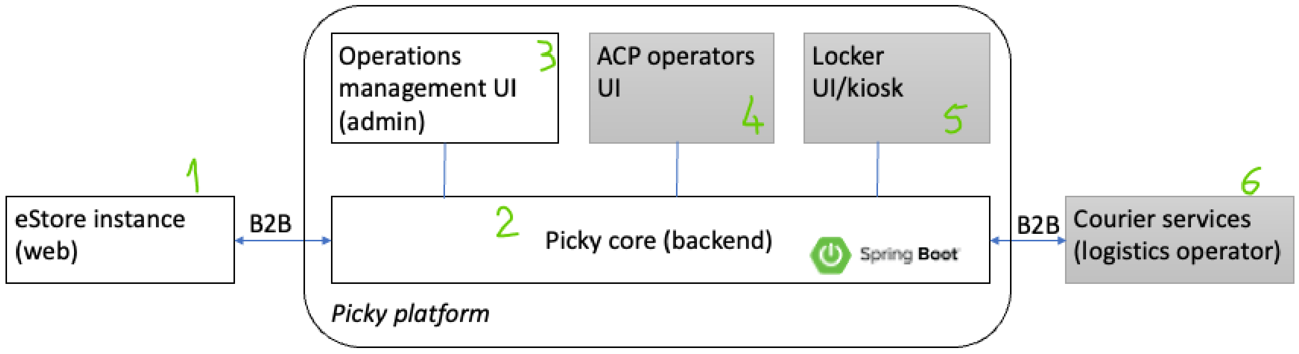


Figure 2: High-level view of the intended system.

What are the technology constraints?
- eStore (#1): a very simple web application that allows users to place orders and track order status progress. You may choose the implementation strategy and supporting framework/languages.
- Platform backend (#2): the backend is to be implemented in **Spring Boot**. A REST API is required for B2B. Services should be fully tested.

- Web UI for the platform management staff (#3): frontend to support the Picky operations management, such as browsing parcels waiting delivery, check operational statistics, accept/add ACP operators,.. You may choose framework/languages. UAT expected.
- Web UI for the associated partner's staff (#4): the interface used by the operators at associated sites to check-in/checkout parcels. UAT expected.
- Locker kiosk (#5): the required interactions to handle placing parcels in the locker (by couriers) and collecting them (by customers). This module should also manage the state of the locker drawers (not only frontend). The group can choose technologies (but must also implement tests).
- Courier services (#6) as microservice to abstract the basic operations of the courier. No technology constraints.

Which are the mandatory components (Figure 2) for the assignment?
- "white" boxes are mandatory.
- one of the two withdraw approaches is required, i.e., you may choose to use the ACP scenario (box #4) or the Locker scenario (box #5). Obviously, you still can do both...
- the implementation of (simplified) B2B integration with Courier services is not mandatory; "box" 6 is totally optional.
- all implemented components should have a QA/testing. For frontends, UAT is expected; for the core backend, a multi-layer and comprehensive testing strategy should be implemented.

3 QA and development practices

3.1 Practices to implement

3.1.1 Agile (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 other tests).
The backlog should be consistent with the development activities of the team; both the backlog 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 are expected to complement this practice by issuing a "pull request" (a.k.a. merge request) 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). Static code analysis should be performed before accepting the pull-request.

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 use **TDD practices** (which are 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 and reporting. The main submission channel 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 staff.
Expected structure:
README.md
docs/
projX/
projY/
...with the following content:

- docs/ → 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 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!
    - Project Backlog (e.g.: JIRA, GitLab agile planning)
    - Related repositories (there may be other related code repositories...)
    - Related workspaces in a cloud drives, if applicable (e.g.: shared Google Drive)
    - API documentation (link the landing page for your API documentation)
    - Static analysis (quick access to the quality dashboard)
    - CI/CD environment (quick access to the results dashboard)
    - ...

For most CI/CD pipelines, it would be **better to use more than one repo**, i.e., specific git repositories for different projects/modules. In that case, consider:

- using a "main repository" and be sure to link related repositories in the README.md, or

- create a “group” of repositories (if your cloud infrastructure has that concept) and share the group.

3.2.2 Project reporting and technical specifications

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

1. Product Specification report [→ template available in the project resources]
2. QA Manual [→ template available in the project resources]

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, Postman documentation, or similar, to create the API documentation.

3.2.4 System observability

Provide dashboards and alarms on the operation conditions of infrastructure and deployed services. Instrumentation to monitor the production environment (e.g.: Nagios alarms, *ELK stack* for application logs analysis,...)

3.3 Extra credit

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

- Physical locker simulator.