Project assignment guidelines

J. L. Oliveira, I. C. Oliveira, v2020-11-15

1 Project activities and methods

1.1 Project assignment objectives

The project assignment will require that students apply selected software engineering practices, including:

- Develop a product specification, from the usage scenarios to the technical design.
- Propose, justify and implement a software architecture, based on enterprise *frameworks*.
- Apply collaborative work practices, both in code development and agile project management.

1.2 Team and roles

Each team/group should assign the following roles:

Role	Key responsibilities
Team manager (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 due 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.
Architect	Deep understanding of the proposed architecture and supporting technologies. The team will turn into the Architect to explain the expected behavior of each software component and interactions between modules.
DevOps master	Responsible for the infrastructure and its configuration; ensures that the development framework works properly. In-charge of 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 2-week iterations. Active management of the product backlog will act as the main source of progress tracking and work assignment.

Expected results from project iterations:

Iter. #	Focus	Required outcomes
I1 16/11 23/11	Project initiation (define the concept, setup the tools, prioritize user stories). Define the product architecture.	• Backlog management system setup. Core stories defines and prioritized. • Team repository in place. • Draft Project Specification (report); must include the Architecture Notebook part. • Prototypes[1] for the core user stories.
I2 30/11 07/11	Develop a few core user stories involving data access. Demonstrate the architecture from end-to-end.	• Basic data pipeline in-place: data streams generation, transmission, and storage. • Product increment covering (at least) a user story related to data access (browse current values from streams, in the web presentation layer.) • Increment deployed (in containers) at the server environment.

[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. #	Focus	Required outcomes
I3 14/12 21/12	Develop a few user stories requiring data processing. Stabilize the services API.	• New user stories required for a functional MVP deployed, specially covering data aggregation/ visualization. • Required user story: alarms/events detection on data streams and feedback to the presentation. • Draft REST API deployed in the server. • Implement integrations with external services (if applicable; e.g.: consuming public web services) • Integrate the cypher-physical layer.
I4 & Mile-stone 1 4/1 11/1	Stabilize the Minimal Viable Product (MVP). Present the first release of the MVP.	• Stabilize the presentation layer (for end-users) • Stabilize the REST API. • Stabilize the production environment. • MVP backend deployed in the server (or cloud); relevant/representative data included in the repositories (not a "clean state"). • Update documentation (project specifications and software documentation). • Oral presentation/defense.

1.4 Required practices

1.4.1 Active *backlog* management (plan & track)

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 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.

There are sever options for the backlog management:

- GitLab Project management (with boards).
- GitHub Project boards.
- Atlassian Jira (with Scrum/kanban boards)
- PivotalTracker (public projects can use all features).

The backlog should be consistent with the development activities of the team; both the project management environment 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: you don't need to implement the Acceptance Criteria parts).

1.4.2 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"/feature-driven workflow or the "Gitflow workflow".

The **feature-branch should match the user-story** 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 *master*. All major Git cloud-platforms support the pull-request workflow (e.g.: GitHub, GitLab, Bitbucket).

1.4.3 Containers-based deployment

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

1.5 Project outcomes/artifacts

1.5.1 Project repository

The project outcomes should be organized in a cloud-based 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 repository structure:

readme.md
reports/
presentations/
projX/
projY/

...with the following content:

- readme.md → be sure to include the sections:
  Project abstract: title and brief description of the project features.
  Project team: students' identification and the assigned roles.
  Project bookmarks: links to quickly access all project resources, such as project management boards, editable versions of the reports in the cloud, entry point for your API documentation,....
- reports/ → you are expect to prepare and submit specifications along the project. These reports should be included in this folder, as PDF files.
- Presentations/ → materials used in project-related presentations
- projX/ → the source code for subproject "projX". You may have different modules.

1.5.2 Project requirements and technical specifications

The project documentation should be kept in the master branch of the repository ([master]/reports) and updated accordingly.

The "Project Specification Report" [see sample template] should cover:

A. Product concept
   A.1. Vision statement
   A.2. Personas
   A.3. Key scenarios
B. Architecture notebook
   B.1. Key quality requirements
   B.2. Architectural view
   B.3. Modules interactions
C. Information model

1.5.3 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.

2 Product description

2.1 Generic requirements

The project will implement an enterprise-class solution, including:

- distributed (remote) **generation of data streams** (e.g.: collecting data from smart devices)
- **data publishing** using a lightweight, message-oriented protocol (one-way: remote → central backend).
- long term **storage** of data using a persistent database engine;
- **central processing**, ensuring the detection of relevant events and data aggregation.
- **service API** (REST) providing a comprehensive set of endpoints to access data and manage the system.
- a **web portal** to implement the core user stories. Using the web portal, users will be able to track the current updates (upstream data) and query/filter stored data. In addition, you may provide a **mobile** application as an alternative presentation.

The teams should find a suitable application area and validate the scope with the teachers. Applications like precision agriculture, health/fitness diaries, smart homes, city infrastructures management, food-deliveries performance tracking systems, etc., provide good examples. Here is an example of the intended scope:

Project theme: wind farms management system

Goal: an integrated platform to monitor and control remote wind turbines, in wind farms at multiple sites.

Sensing layer: wind turbines telemetry (rpm, generation output,...) is being continuous collected; the environment parameters (wind speed,...) are also tracked.

Data publishing: at each wind farm, there is site gateway, that act as a local area aggregator and sends the telemetry data to the cloud, using a bandwidth-savvy protocol.

Processing & bizz logic: hazards conditions are detected when the telemetry reveals that operating thresholds are compromised. In this case, the alarms are also forwarded to the mobile devices of the people in charge. Peak conditions are detected in a daily basis (data aggregation) to build a historic trend.

Integration API: the exposed endpoints allow to programmatically list the park information system (e.g.: characteristics of each tower) and telemetry readings, both for current and past intervals.

Web portal: the web application allows basic tracking of operational conditions (telemetry dashboard of the turbines); browse descriptive information on each tower/device (model, location,...); it is also possible to adjust operational parameters, actuating in the wind farm (e.g.: stop a turbine).

2.2 Architectural requirements

The solution should adopt, in general, a multilayer architecture (Figure 1).

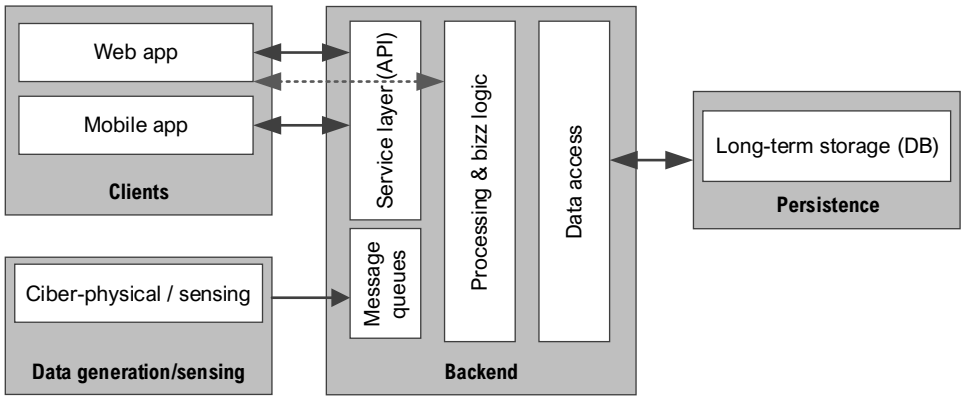

Figure 1: Reference architecture (generic; should be adapted).

Notes on the expected components and technologies:

a) **Web app**: you may choose the web development framework and integrate with the backend using the services API. If you choose to use a Java-related technology (e.g.: Thymeleaf and Spring MVC), the web layer may use directly the components in the backend (bypassing the API). If you choose a JavaScript-based framework, for example, the presentation layer would interact with your API.
b) **Mobile app** (optional): a simple Android application (or Flutter) that interacts with the backend using REST. Push notifications would be an additional challenge.
c) **Cyber-physical/sensing:** the system should ingest data streams, typically being generated at cyber-physical systems (e.g.: environmental sensors, cameras, location tracking, devices telemetry, body sensors, etc). The use of real devices (hardware) is optional and you may use "digital twins" instead (virtual devices, simulated in software).
d) **Message queues**. Data ingestion (from streams into the backend) should use message-oriented middleware, with asynchronous messages.
e) **Service layer (API)**. Endpoints to allow other systems and different presentation layers to connect to the system services. The API should follow the RESTful style, with JSON payloads.
f) **Processing & bizz logic**. This module will likely divide into several sub-components, depending on the specific problem domain. You should have some processing module to analyze the incoming streams (likely to integrate with the message broker).
g) **Data access**. Data from streams, and the information system to support other parts of the system (e.g.: users, profiles, preferences), is to be persisted using an object-database mapping framework, like Java Persistence API.
h) **Long-term storage**. The system will save the date into a convenient database technology. While the Relational databases are obvious options (e.g.: Postgresql, MySql), you may use other solutions that better fit the project requirements (e.g.: time-based queries will benefit from a timeseries repository).

Additional notes:

i) You need to implement at least one presentation platform: web app or mobile app.

2.3 Extra credits (advanced topics)

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

- Use of relevant cyber physical components (e.g.: sensors or actuators connected to a RPi board) to deliver value to the end-users.
- Robust data analytics (integrate a framework for streams processing at the backend)
- Instrumentation to monitor the production environment (e.g.: Nagios alarms, *ELK stack* for application logs analysis,...)