Project assignment guidelines

J. L. Oliveira, I. C. Oliveira, L. Bastião | v2022-10-29

1 Project assignment overview 1
1.1 Assignment objectives 1
1.2 Team and roles 1
2 Solution concept 2
2.1 Generic functional requirements 2
2.2 Technical architecture 3
3 Project implementation 4
3.1 Project iterations plan 4
3.2 Required practices 5
3.2.1 Agile *backlog* management (plan & track) 5
3.2.2 Feature-branching workflow 5
3.2.3 Containers-based deployment 6
3.3 Project outcomes/artifacts 6
3.3.1 Project repository (Git) 6
3.3.2 Project requirements and technical specifications 6
3.3.3 API documentation 7
3.4 Extra credits (advanced topics) 7

1 Project assignment overview

1.1 Assignment objectives

The project assignment will require students to apply best practices for software engineering.

As a team, students will:

- Develop the concept for a software solution that handles distributed data collection and its analysis in a web environment.
- 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 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.

2 Solution concept

Each group is expected to propose, conceptualize, and implement a multi-layer, enterprise-class application. The project theme and scope should adhere to the guidelines in the following sections.

2.1 Generic functional requirements

The proposed solution should include:

- distributed (remote) **generation of data streams**, either from real data sources (e.g.: collecting data from devices) or by virtual software "agents" (e.g.: simulating the behavior of sensor)
- **data publishing** using a lightweight, message-oriented protocol (one-way: remote origin → backend).
- long term **storage** of data using a persistent database engine.
- **central processing**, ensuring the detection of relevant events (e.g.: alarms) and data aggregation (e.g.: weekly evolution)
- **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, the users will be able to track the current updates (access the upstream data) and query/filter stored data (historical views). In addition, you may provide a **mobile application** as an alternative presentation frontend.

The teams should find a suitable application area and validate the scope with the teachers. Application domains such as precision agriculture, health/fitness diaries, smart spaces/homes, city infrastructures management, food-deliveries performance tracking systems, etc., offer good examples.

Here is a worked example (Table 1) of the intended scope, considering an integrated platform to monitor and control remote wind turbines, in wind farms, at multiple sites.

Table 1: Project example: wind farms management system.

Layer/component	Sample implementation
Data acquisition layer	wind turbines telemetry (rpm, energy generation output,...) is being continuous collected; the environment parameters (wind speed, wind orientation,…) are also tracked.
Data publishing	at each wind farm, there is site gateway, acting as a local area aggregator and sends the telemetry data to the cloud, using a bandwidth-savvy protocol.

Layer/component	Sample implementation
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 an hourly basis (data aggregation) to build historic trends
Integration API	the computational platform exposes endpoints that allows other authorized applications to programmatically list the park information system (e.g.: farm location, details of each tower,…) and telemetry readings, both current and for past intervals of interest.
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,...); adjust operational parameters, actuating in the wind farm (e.g.: stop a turbine)

2.2 Technical architecture

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

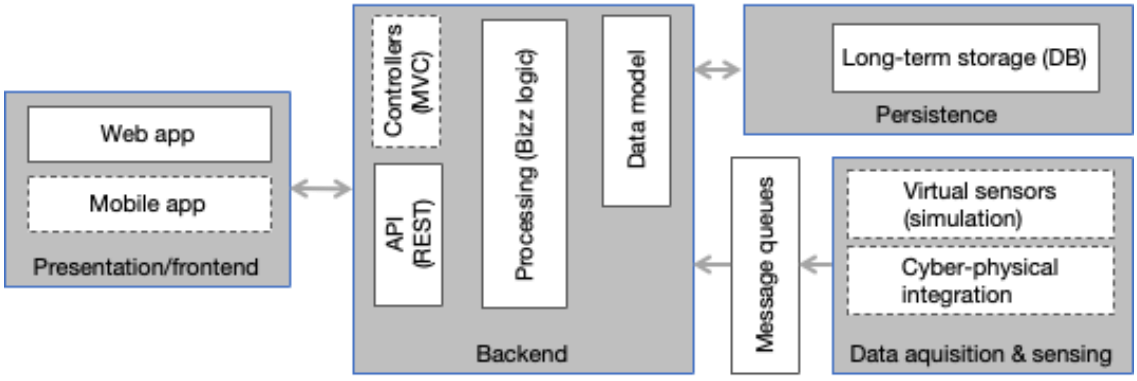

Figure 1: Reference architecture (should be adapted to each project; dashed components may be relevant or not, depending on the approach).

Notes on the expected components and implementation 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 access the controllers in the backend directly. If you choose a JavaScript-based framework, for example, the presentation layer would interact with the API.
b) **Mobile app** (optional): a simple Android (or Flutter) application that interacts with the backend using the REST API. Push notifications would be an additional challenge.
c) **Data acquisition & sensing:** the system should ingest data streams, typically generated at cyber-physical systems (e.g.: environmental sensors, cameras, location tracking, devices telemetry, body sensors, etc.) or by virtual software "sensors". The use of hardware devices is optional, and you may use "digital twins" instead (virtual representation of a device, simulated in software); or event "agents" that generate data streams that are not related to hardware (e.g.: outcome lab tests for COVID-19). Data streams should provide **realistic data** for the specific domain.
d) **Message queues**. Data ingestion (into the backend) should use lightweight, message-oriented middleware, with asynchronous messages.
e) **API (boundary)**. 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 be divided into several sub-modules, 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 model**. 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 Spring Data.
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.

3 Project implementation

3.1 Project iterations plan

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

During the project, Practical classes will mostly work in this way:

- Iteration planning in the first class/week of the iteration.
- Iteration review in the second class/week of the iteration.
- As usual, classes are open to students' questions.

Table 2: main activities (focus) and results (outcomes) for each project iteration (dates refer to the starting day of the week)

Iter. #	Iteration focus	Required outcomes
I1 31/10 7/11	Project inception (define the concept, define core user stories). Project initiation (select tools, agree on work practices, setup developer environment). Define the product architecture.	• Backlog management system setup. Core stories defined and prioritized. • Team repository in place. • Draft Project Specification (report); must include the Architecture Notebook part. • Prototypes[1] for the core user stories.
I2 14/11 21/11	Data generation/"sensing" with relevant events. Develop a few core user stories involving data access. Demonstrate the architecture end-to-end (full-stack proof-of-concept: from data generation to user-interface).	• Basic data pipeline in-place: data streams generation, transmission, and storage. • Product increment covering (at least) a user story related to data access (e.g.: browse the 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 (not mockups) but more or less "static" (not yet integrated with the business logic).

Iter. #	Iteration focus	Required outcomes
I3 28/11 05/12	Develop a few user stories requiring data processing. Stabilize the API and system integrations.	• Additional user stories required for a functional MVP deployed, specially covering data aggregation/ visualization of historic data. • Required user story: alarms/events detection on data streams and feedback to the UI. • REST API deployed in the server. • Implement integrations with external services (if applicable, e.g.: consuming public web services) • Integrate the cypher-physical layer (if applicable)
I4 & Mile-stone 1 12/12 (19/12)	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). • Project presentation/defense.

3.2 Required practices

3.2.1 Agile *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 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. For more information on user stories, check this FAQ on story-oriented development (note: you don't need to implement the Acceptance Criteria parts).

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:

- Atlassian Jira (with Scrum boards) ← suggested.
- GitLab Project management (with boards).
- GitHub Projects + boards (e.g.: ZenHub).

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.

3.2.2 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 feature-driven workflow and define a policy for the integration of increments, such as trunk-based, GitHub Flow (suggested) or the stricter "Gitflow workflow".

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 *master*. All major Git cloud-platforms support the pull-request workflow (e.g.: GitHub, GitLab, Bitbucket). See the overall practice in

3.2.3 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). Your containers will likely 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[2].

3.3 Project outcomes/artifacts

3.3.1 Project repository (Git)

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*/

Part	**Content:**
readme.md	Context and project bookmarks placeholder. 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/	Project specifications, as PDF files.
🗁 presentations/	Materials used in project-related presentations.
🗁 *projX*/…	The source code for subproject "projX". The amount and names of subprojects depend on each problem.

Note: in some cases, it may be helpful to use additional repositories (instead of sub-folders). Check with your teacher, if needed.

3.3.2 Project requirements and technical specifications

The project documentation should be kept updated in the master branch of the repository ([master]/reports). The "**Product Specification Report**" [see sample template] should cover:

A. Product concept
   A.1. Vision statement
   A.2. Personas & motivations
   A.3. Main scenarios
B. Architecture notebook
   B.1. Key quality requirements
   B.2. Architectural view
   B.3. Module interactions [dynamic view]

[2] Although it is likely that your use Docker containers, other options may be considered, depending on the hosting infrastructure.

C. Information view

3.3.3 API documentation

Provide an autonomous resource describing the services of your own API. This should explain the overall organization, available methods and the expected usage. See related example.

Consider using tools such as OpenAPI/Swagger framework or Postman documentation tool to create the API documentation.

3.4 Extra credits (advanced topics)

The following challenges are not mandatory, but may give you extra credits in the project, depending on the extent and quality of the implementation:

- 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** (e.g.: integrate a framework for streams processing at the backend)
- System observability by implementing instrumentation to monitor the production environment (e.g.: *ELK stack* for application logs analysis)