TQS: Development environment and labs logistics

v2025-09-25

0.1 Submission of labs

Each student is expected to keep an “individual portfolio” with the work proposed in the labs. This is a personal git repository into which lab work should be committed. Remember to:

- Keep a **clean organization** that maps the Lab exercises’ structure, e.g.: 🗁lab1/; 🗁lab1/lab1_1; 🗁lab1/lab1_2; 🗁lab2/lab2_1; …
- Keep your repo (portfolio) up to date and complete. Commit the work of a week before the next one.
- In addition to the code projects that are usually assigned for lab activities, you are required to maintain a "notebook" for each lab. This notebook should contain study notes, summaries of key concepts, basic cheat sheets, useful links, and **evidence of lab completion**. These notes should enable you to study independently later.

  These **personal lab notes** go into a “**readme.md**” file in each lab root entry.

Git repository

How to access the repo for the Individual Portfolio (at **GitHub**):

a) You need a GitHub account.
b) Make sure you are signed to the “detiuaveiro” organization. You may need to authorize GitHub to SSO with “DETI - UA”.
   Note: if you are using SSH Keys to sign-in to GitHub, you also need to extend the SSH authorization to the “detiuaveiro” organization.
c) Accept the assignment using this invite link → https://classroom.github.com/a/GmjMJxTL
d) After accepting the assignment, be sure to link your personal GitHub account to the pre-existing TQS’ repositories entries (link your user to the student number in the TQS roster).

0.2 Development environment

Coding assignments will be based in Java and “Java for enterprise apps”, in our case, the Spring Boot framework.

System configuration requirements and setup:

— **JDK** (Java development environment): long-term support JDK recommended. OpenJDK v21 LTS will be used as reference. Note that you should install it into a path without spaces or special characters (e.g.: avoid \Users\José Conceição\Java).

— Java projects **build tool**: Maven will be used as reference; you may use Gradle too. Be sure Maven v3.9+ is accessible from the command line:

```
$ mvn --version
```

— **IDE**: the recommended Java/Spring Boot capable IDEs are IntelliJ IDEA (version “Ultimate”) and VS Code (with extensions); pick one!

— **OS**: no operating system recommendation; many find Linux offers more previsible experience. You will need to run Docker (but not for the initial couple of labs)

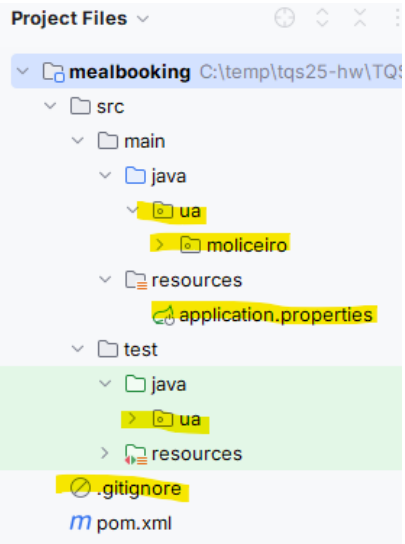
0.3 Build tool (Maven)

Professional development relies on team-friendly practices, including the **portability of projects** (can be opened anywhere). Build-environment (build tools) can help by describing the project elements and dependencies in a platform neutral way; then, at build time, the project description files is evaluated to find:

- Which parts of the project are new and need recompiling;
- Which declared dependencies require to fetch components/libraries from the internet
- Are there tests to run (and fail the build if they don't pass)?

Maven is a (popular and mature) build tool for Java projects. Another alternative for the Java ecosystem is Gradle, a more modern framework (also leveraging on Maven repositories).

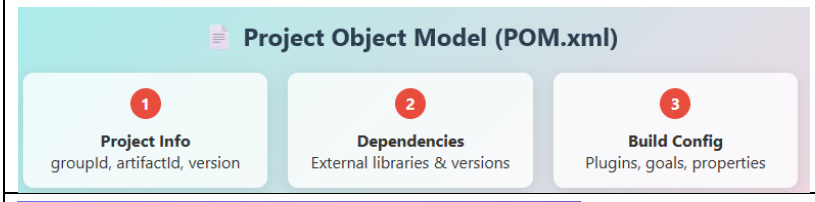
Anatomy of a Maven-based project:



There's a template for Maven projects file's structure, as illustrated.

New (production) code goes into:
<project root> → src → main → java → <your packages>
Tests go into:
<project root> → src → test → java → <your packages
Usually, you don't need to care about the project structure, as any Maven capable IDE takes care with keeping the project neat and tidy.

Note: highlighted entries are project specific, not mandatory parts of the Maven template.



Project metadata, dependencies, and required build extensions are defined in the **pom.xml** file, in the root of the project.

A build goes through a sequence of phases (*aka* Maven goals), with predefined semantics. Issuing:

```
$ mvn package
```

means that the sequence of phases *validate* → *compile* → *test* → *package* will be activated.
The build is interrupted and fails if (and as soon as) a phase produces errors.

A maven build is a transformation process that:

- Uses the configuration in the POM.xml to learn about the build requirements;
- Gets required dependencies, if needed, from local cache or remote repositories;
- Compile the source code (incrementally; parts that were not touched don't need recompiling)
- Package the output into an executable, usually a Java archive (JAR)
- Some builds can define extra stages, supported by additional "plug-ins", such as tests reports, site generation or deploying to Docker images.

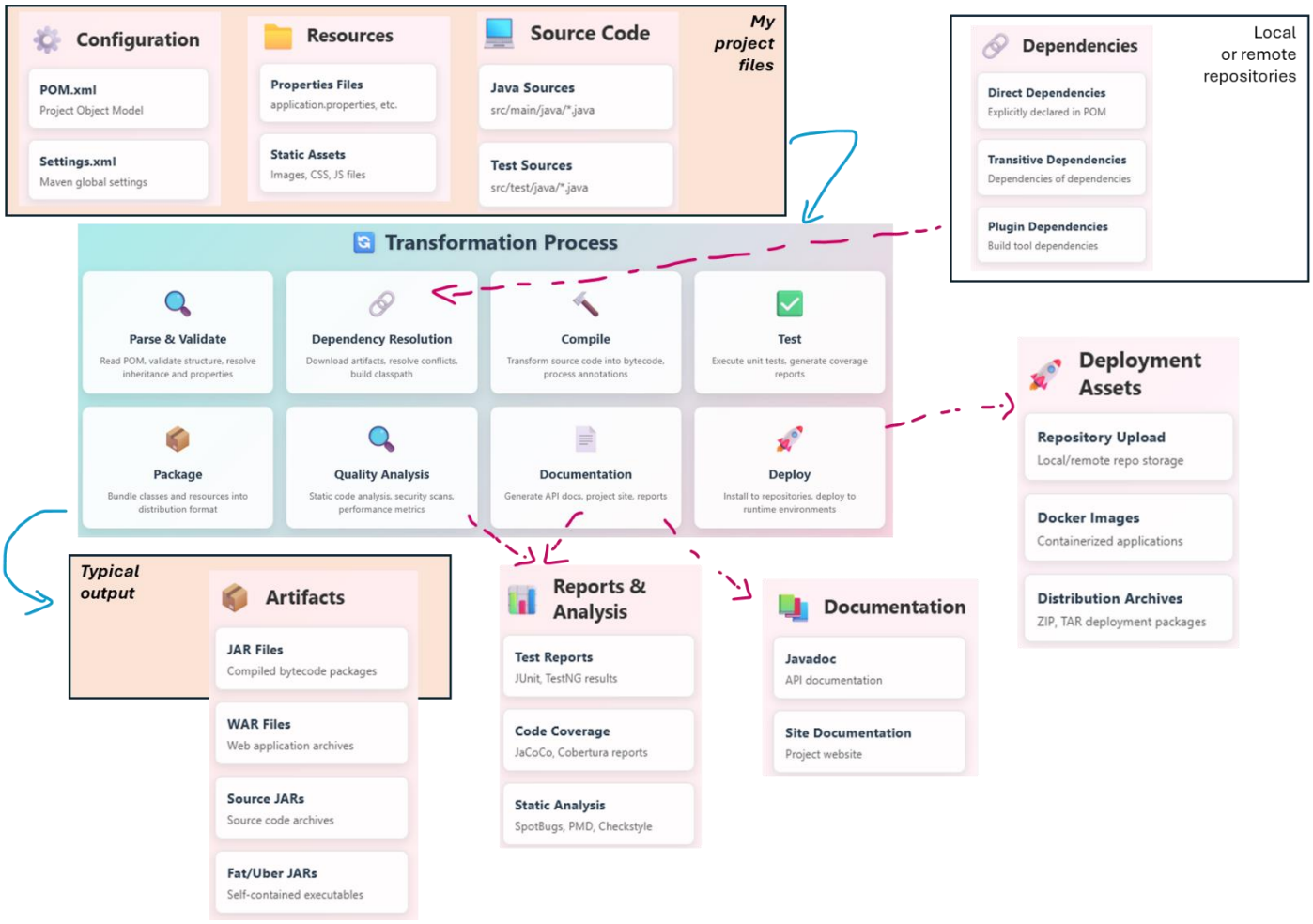


Figure 1: inputs and outcomes involved in the "Maven build" process.

0.4 Selected best practices

Do:

- Do use .gitignore. You can add it at root level, propagating the rules to the subfolders.
- Always lock file encoding to UTF-8 (your code will likely travel multiple platforms…).
- Keep the POM.xml as clean and slim as possible. Avoid the use of old Maven archetypes to generate a project.
- Use Generative AI as a productivity tool and a brainstorming assistant.

0.5 Policy for the use of AI assistants

In TQS, the use of AI is welcome and expected. However, keep in mind that you must be in control and understand every element of the software. It is ok to leave the “typist” role to AI, but not the design/architecture.

Keep in mind that there are risks with generated code, especially:

- lack of context and business Logic: AI models tend to lack the comprehensive, nuanced understanding of a specific project's architecture, business requirements, and long-term goals.
- Suboptimal security and quality design: if not properly reviewed, it can contain security vulnerabilities, inefficient logic, outdated dependencies, etc, or failing to anticipate future needs.