D03 – Persistence models
Evaluation procedure

- The lecturers will check the following general sufficient conditions for failure, if applicable:
    - An interaction with your system results in an HTTP error.
    - An interaction with your system results in a panic.
    - Submitting a form with wrong data is not detected.
    - There are missing CRUD use cases.
    - An actor can list, edit, or delete data that belongs to another actor.
- Regarding management, the lecturers will check that:
    - You've followed the delivery instructions that are provided in document "On your deliverables".
    - Your project was properly managed in ProjETSII. They'll pay special attention to checking that your tasks make sense, that they were not created in a batch before the deadline, and that you've allocated time to study the lessons, to work on the problems, the deliverable, to discuss it with your partner, and so on.
- Regarding your Eclipse/Maven projects, the lecturers will check that:
    - You've instantiated and customised the project template according to the guidelines that we provided to you. They'll pay special attention to checking that the databases and the URLs are in accordance with the name of the project.
- Regarding the models, the lecturers will check that:
    - You use your customer's vocabulary, which uses English terms.
    - The conceptual model represents the requirements faithfully.
    - The conceptual model doesn't have any void attributes.
    - The conceptual model's not been scaffolded.
    - The UML domain model doesn't have any artefacts that aren't supported by Java.
    - The UML domain model's not been scaffolded.
    - The Java domain model is clean and efficient.
    - The Java domain model includes every annotation that is required, including @NotNull and @Valid.
    - You use wrapper and primitive types correctly in your Java domain model.
- Regarding your persistence model, the lecturers will check that:
    - It actually represents your UML domain model faithfully.
    - Your "PopulateDatabase.xml" file specifies enough objects of each type and it accounts for enough variability, e.g., if entity "A" can be related to several entities of type "B", then they'll check that your "PopulateDatabase.xml" specifies an "A" entity that is related to as many "B" as possible.
    - Utility "PopulateDatabase.java" can be executed from within Eclipse and persists the entities in your "PopulateDatabase.xml" file correctly.
    - Utility "QueryDatabase.java" can execute your JPQL queries and that they return the expected results, which you must properly document.