Milestone 2 (MS02)

Milestone 2 focuses on incorporating domain validations and property-based testing. Building upon the foundational MVP developed in Milestone 1, the second milestone addresses deeper domain logic and consistency properties that govern valid schedules, resource constraints, and task sequencing in a production environment. Milestone 2 introduces logic to ensure that the generated schedules adhere to real-world constraints and domain invariants, such as exclusive resource usage, valid time computation, and proper task completion. This milestone also leverages **ScalaCheck** for randomized and property-based testing to stress test the domain logic and catch edge cases.

Main Decisions

Property-Based Testing Framework

ScalaCheck was used as the primary tool for property-based testing, as mandatory by the curricular unit teacher.

Test Properties Defined

A set of critical properties were defined to validate the correctness and feasibility of the schedules. Key properties include:

1. The same resource cannot be used at the same time by two tasks;
2. The complete schedule must schedule all the tasks of all the products needed;
3. Human resources must have the skill required by the physical resource used;
4. Tasks should only use resources explicitly required in their product/task definition;
5. Every task must have both physical and human resources allocated;
6. A product from a task is produced according to the defined quantity;
7. The sum of all scheduled task durations must be equal or less than max time;
8. Each task must have at least one human resource assigned;
9. Each task must have at least one physical resource assigned;
10. Each task must have a duration greater than 0 and less than or equal to 240 minutes.

Each property was validated with randomized test cases to identify minimal failing cases.

Test Case Generators

We created specific generators for different complexity levels:

- **Simple generators** for small domain classes(e.g. duration, quantity).
- **Complex generators** for bigger classes involving orders, products, and resources.
- **Dependent generators** to ensure logical consistency across nested domain elements.
- **Test case generators** for deliberately invalid scenarios (e.g., tasks with zero duration) to validate error detection.

Alternatives

Flat generators (e.g., generating complete TaskSchedule instances in one step) could be an alternative to data generation, but less flexible when testing individual components.

Future Work

The next steps for possible improvements on the project, which will be tackled in Milestone 3 are:

- Support for Parallel and Non-Sequential Scheduling
    - Currently, tasks are scheduled sequentially across orders and product units. Future work could introduce parallel scheduling of independent tasks, concurrent order processing based on resource availability or even a smart time-slot allocation algorithm, such as earliest-fit or resource-aware heuristics.
- Dynamic Resource Management
    - Extend the project to include reusable resources (Resources become available again after task completion) or time windows or working hours constraints per resource.
- Advanced Scheduling Algorithms
    - Incorporate more sophisticated approaches to optimize performance and throughput:
        - Greedy heuristics, priority queues, or critical path methods.
        - Constraint-based solvers to maximize utilization or minimize total production time.