MTAT.03.229 – Enterprise System Integration
Regular Theory Exam (Sample)

Q1. Consider an application to manage room bookings. One of the resources in this application is the booking. The application exposes a container resource /bookings/ that provides access to all the bookings, and one item resource */bookings/{bookingid}* for each booking. A booking consists of a reference to a room, a start date-time and an end date-time. Let us assume that you have to implement an operation to update a booking. Possible updates include changing the room, the start or the end time of the booking, or any combination of these three attributes. What would be the difference between using the **HTTP PUT** method to update a booking versus the **HTTP PATCH** method?

Q2. A simulator of chemical reactions exposes a simple API for submitting simulation jobs. A simulation job is submitted via a **POST** on */api/simulation/*. The request body contains all the input parameters required for the simulation. If the simulation is successful, a *200* code is returned together with the simulation results.

In the backend, the simulation job is handled by a Python script. The script checks the contents of the request, and if it is all correct, it invokes a simulation engine via command-line. When the engine completes, the Python script recovers it and sends it back via the same HTTP connection. Each simulation job takes between 5 and 30 seconds of CPU time. In case of a heavy workload, the backend usually crashes, or it becomes too slow.

How would you re-architect the backend to make the system more scalable? Do your changes have any impact on the REST API? If so, please explain? Note that the manager does not have a budget to buy more hardware. You should come up with a solution that works with the existing hardware.

Q3. An IT company in a faraway town hired a junior developer to work on the implementation of RentIT. The requirements of the systems are the same as those you implemented during the semester. Thus surely you are very experienced on the topic. After two weeks of analysing the project requirements, the project manager asked the junior developer to implement RentIT on a hexagonal architecture. However, the junior developer explained to the manager that it is not possible due to they were using Spring Boot. Accordingly, they have to implement a Model View Controller pattern, and the system should be built on a 3-Tier architecture (data access, application logic and presentation). Would you agree with the junior developer? Explain why you agree or disagree with them, focusing on your RentIT system.

Q4. Do you consider it is beneficial to use HAETOAS on the implementation of an information system? Explain why. If not, what would you propose instead? What is the impact, either positive or negative, of using HAETOAS on your implementation of BuildIT and RentIT?

Q5. Consider the following code snippet concerning the definition of the entity Purchase Order in the RentIT system. What are the limitations of the current representation and strategy for generating the identifiers of the POs? What would be a better choice for representing the identifiers, and why? What is the design pattern implemented by the static method ***PurchaseOrder.of***, and what would the advantage of using it on RentIT?

```
@Entity
@Data
@NoArgsConstructor(force=true,access= AccessLevel.PROTECTED)
public class PurchaseOrder {
    @Id
    @GeneratedValue
    Long id;

 public static PurchaseOrder of(PlantInventoryEntry entry, BusinessPeriod period) {
        PurchaseOrder po = new PurchaseOrder();
        ...
        return po;
    }
 }
```