**Enterprise System Integration UT 2021 -  
Homework 2**

# **General Notes**

In these homework series you will build step-by-step a service called RentIT. RentIT is a plant hire supplier. Before starting, familiarize yourself with the complete scenario: <https://docs.google.com/document/d/1N2HLMjWlzvbtketMguOMBa0bGfSUyM33Cwn-oeSw2CM/edit>

The requirements that need to be addressed in this homework are:

PS1. The system should allow a customer to list the available plants

PS2. The system should allow a customer to check the price for a given plant (given the plant identifier and a time period (start date, end date))

PS3. The system should allow a customer to check the availability of a given plant during a given time period

To provide more context, RentIT has two database admins. One of them is a fan of PostgreSQL, the other is a MongoDB diehard, therefore half of the plant types are stored in PostgreSQL and MongoDB for the other half respectively. The system should support both, as it will be faster than to convince each DB admin to migrate to the opposing technology (welcome to the enterprise world :)).

Additionally, the RentIT project manager has read an article on Redis and wants you to implement caching as well for the service, so that plant requests are cached and served faster from memory.

RentIT has several clients which whom they had agreed on the service interface:

* BuildIT requires HTTP
* ReBuildIT requires WebSocket
* DestroyIT are feeling a little hipster, they require gRPC or graphQL

**Tasks**

1. Design and build the application core fulfilling:
   * Functional/Features
     + PS1
     + PS2
     + PS3
   * Technical
     + The service should support the following databases:
       - PostgreSQL
       - MongoDB
         * Use your imagination to define types of plant, each database stores half of plant types.
       - Redis
     + The service should support the interfaces:
       - HTTP
       - WebSocket
       - gRPC or graphQL

Make sure to use the hexagonal architecture guidelines:

* + Packaging
  + Object-oriented
  + Dependency injection
  + Interfaces

1. Testing
   * One unit test on the service layer
   * External tests for each functionality that run with docker-compose
2. Documentation, imagine it will be shared with external clients
   * BuildIT - HTTP interface documentation
   * ReBuildIT - Websocket interface documentation
   * DestroyIT - gRPC or graphQL interface documentation
   * Markdown documentation should be enough
3. Deploy the service on the cloud
   * The RentIT service should start with docker-compose
   * Docker images could be built locally on the machine, no container-repository required

* Perform self-audit of each of the four tasks (see guidelines in HW1)