**Enterprise System Integration UT 2021 -  
Project**

# 

[General Notes](#_nd6z2eggdy4i)

[Tasks](#_g805c6leybee)

[Backend service](#_q0brzrjj3cyh)

[Testing](#_e2f72se9hopw)

[Documentation](#_embc9v7397pz)

[Deployment](#_43nw2k6p3ubh)

[CI/CD](#_r1kk78oiu8b9)

[Project management](#_ce0t5fmhumql)

[Presentation](#_g01a4uvbrbpe)

[Grading](#_ewxweb58zgot)

# 

# **General Notes**

In this project you will build a service to support the operations of BuildIT. *Buildit* is a construction company specialized in public works (roads, bridges, pipelines, tunnels, railroads, etc.). Before starting, familiarize yourself with the complete scenario: <https://docs.google.com/document/d/1N2HLMjWlzvbtketMguOMBa0bGfSUyM33Cwn-oeSw2CM/edit>

# Tasks

## Backend service

Create a backend service in Golang that supports the following functionality. You are free to make technical choices in terms of database and exposed service interface. However, you should follow the Golang hexagonal architecture guidelines:

* Packaging
* Struct-oriented
* Dependency injection
* Interfaces

CC1. The system should allow site engineers to create a plant hire request.

CC2. The system should allow site engineers to modify a plant hire request prior to its approval by the works engineer.

CC3. The system should allow site engineers to cancel a plant hire request. Cancellations are allowed until the day before the plant is due to be delivered. If a cancellation is requested after the PO has been sent, a request for cancellation should be sent to the supplier.

CC4. The system should allow site engineers to check the status of a plant hire request.

CC5. The system should allow works engineers to approve, reject or modify a plant hire request.

CC6. The system should produce a PO for every approved plant hire request and forward it to the corresponding supplier. The supplier may respond that the plant being requested is no longer available (which means the PO is rejected), or it may respond with a confirmation of the PO.

CC7. The system should allow Buildit employees to view all submitted POs.

CC8. The system should allow site engineers to request an extension in order to keep a plant longer than its initial period of engagement. When an extension is requested, the system should produce a modified PO and forward it to the supplier. The supplier may accept/reject the modified PO.

CC9. The system should allow a supplier (RentIT) to submit invoices for a given plant hire (via Kafka or RabbitMQ depending on how the invoicing is implemented in RentIT).

CC10. When an invoice is received, the system must check that the PO number in the invoice corresponds to an existing and unpaid Purchase Order. If the PO does not exist, an error is logged to stdout.

CC11. The system must allow site engineers to approve an invoice and to retrieve the PO associated with an invoice.

CC12. The system must submit a remittance advice to the supplier after the invoice is approved (normally the remittance advice should only be sent after the payment has been triggered, but in this project we do not deal with sending payment orders to the bank).

CC13. Come up with a (crazy)unique feature for BuildIT.

## Testing

* All functionality should be externally tested.
* A test stage is defined in the CI/CD environment, where tests are being executed automatically.
* Define (thoroughly, with examples and clear expectations of input/output) acceptance tests for each case when BuildIT interacts with RentIT. For this part, just documentation is required.

## Documentation

* Produce human-readable documentation that can be shared with project managers and clients.
* Technical developer-focused documentation regarding architecture, technology choices, instructions how to run the project, tests etc.
* Acceptance tests (see Testing).
* Add the documentation either in the /wiki folder or Wiki section in your Github repo.

## Deployment

* A deployment stage is defined in the CI/CD environment.
* The deployment stage will deploy the service to the Hetzner cloud (or your other cloud of choice).

## CI/CD

Use Github Actions: <https://github.com/features/actions>

Should have the following stages:

* Lint (runs a linter check, golanci-lint recommended <https://github.com/golangci/golangci-lint>)
* Build (checks against compile errors)
* Test (executes all tests, could be splitted into test-unit and test-external stages)
* Deploy (deploys to a cloud of choice)

## Project management

Use Trello (<https://trello.com/>), Asana (<https://asana.com/>) or any other similar platforms to manage the project internally within the team. Basic column setup:

* Backlog
* Current sprint
* Blocked
* Done

You are expected to perform at-least one internal team meeting per week to catch up with teammates, identify blockers, unblock teammates, re-shuffle priorities, check what tasks have been done etc. Please, document each meeting including:

* Every person's answer to the following questions (Standup style <https://www.atlassian.com/agile/scrum/standups>):
  + What did I work on in the past day(s)?
  + What am I working on?
  + What issues are blocking me?
* Include screenshots of the project management board (Trello, Asana etc.) at the beginning of the meeting and at the end of the meeting (before & after), showing the result of the sprint review (<https://www.atlassian.com/agile/scrum/sprint-reviews>) i.e. moved cards.
* Add the documentation either in the /wiki folder or Wiki section in your Github repo.

## Presentation

Date/Time: before lecture May 27 (playable link needs to be available)

Time limit: 8 minutes

* Overview of the architecture, development decisions and choices.
* Short demo showing how one rental request flows all the way from its creation to the corresponding invoice being paid. During the demo, you should show at least one usage scenario where the site engineer requests a deadline extension for keeping the plant longer.
* Showcase the unique feature.
* Freestyle: issues encountered, suggestions, comments on CI/CD, technical aspects, project management, testing, deployment, documentation etc.

# Grading

Attention: We are not 100% sure about the grading scheme and are considering giving a bit less points for coding and more for the presentation - if you have an opinion about that, please contact any of your instructors and share so we can consider that.

* Code and running tests 50% (we ask for 100% coverage of all requirements with the tests and will only grade if tests run)
* Documentation 20% (including a good pitch story and the relations to implementation)
* Task/team log 10% (who has done what for how long and when/hour sheet)
* Presentation 20% (verify sub-points above)