Software Engineering

Part 4: Test and continuous integration

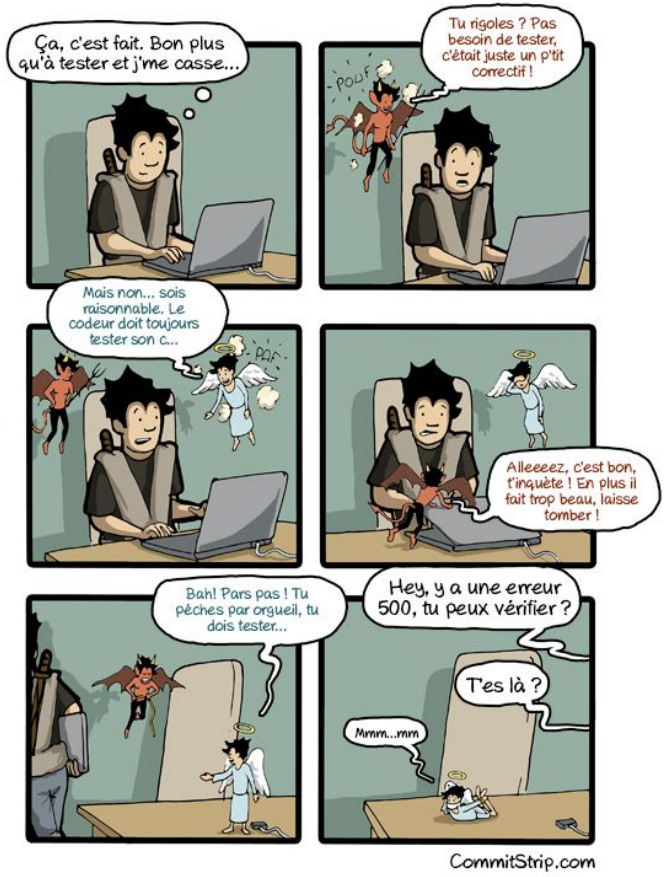
Guillaume Swaenepoel & Thomas Aubin

Tests

Testing ensures consistency and reliability of the software

Tests

A very important phase in the life of a developer is testing, which is both kind of boring and difficult to do properly. Testing is a task that has to be done repeatedly, because very often a change (new feature, bug fix) breaks something that used to work. For big projects, you have "test suites" that just run the software through a lot of controls and checks that everyone of them is passed

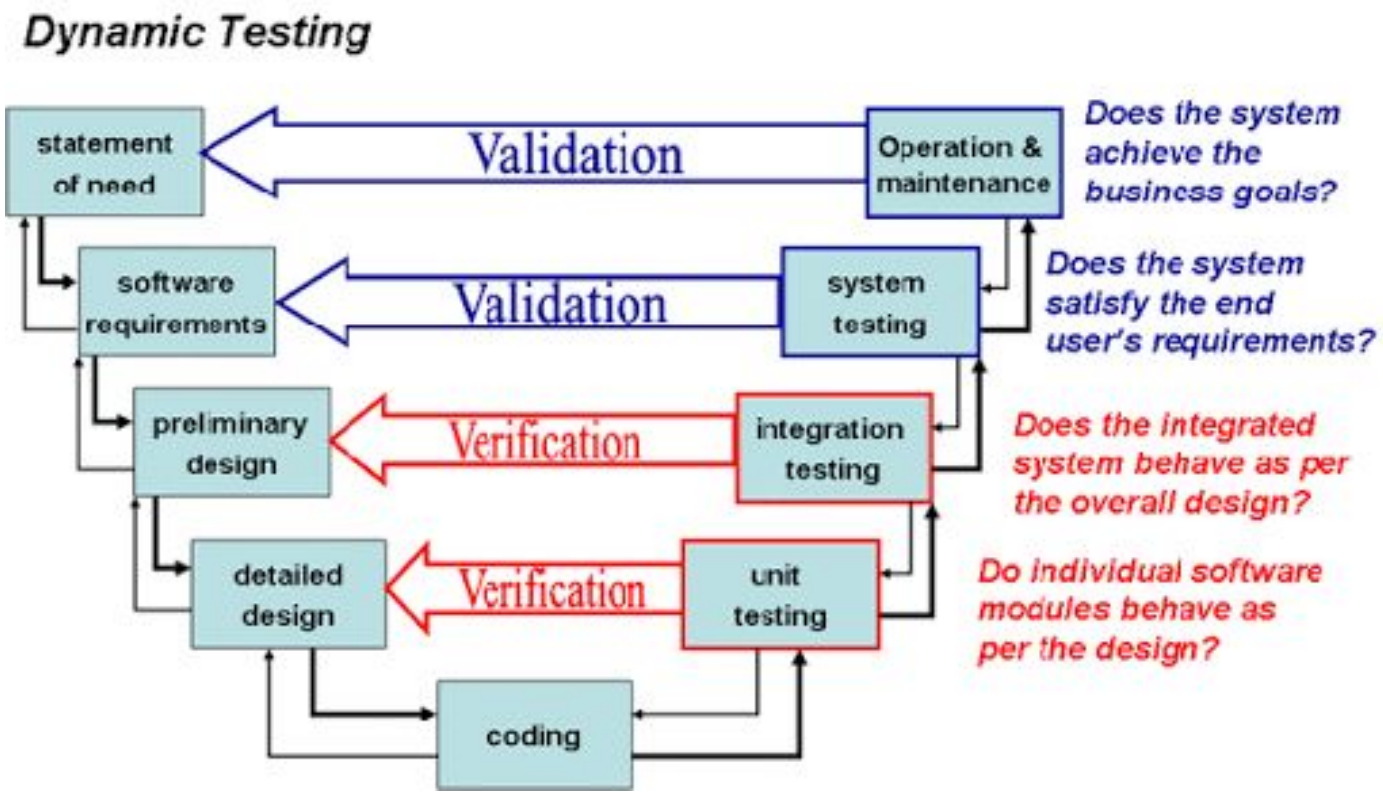


Why testing ?

- Check expected result
- Check new change doesn't break something. (Non regression test)
- Check what the user wanted? (User Acceptance Test)
- Check performance

“Bad programmers have all the answers. Good testers have all the questions.” (Gil Zilberfeld)

Come back with life cycles ...



Unit tests

The functions of a program take part in a large set, and when an error occurs, it is not always easy to locate the source of the error. This often happens when you change the code of a function used by many others. The purpose of unit testing is to ensure the quality of the code. They make it possible to verify that a modification does not have unexpected repercussions. The rule is that we add a unit test for each function added to the code and each bug discovered. In practice, if you don't do it right away, you rarely do it. It is unthinkable to sell or write an open source project without these tests. Most languages allow you to write this quickly.

Unit tests are written by the developer !

Testing libraries

- Python: unittest, nose2, pytest
- Java: JUnit
- C++: Cutter, Google C++ Testing Framework, Boost Test Library
- PHP: PHPUnit, SimpleTest
- Ruby: Test::Unit
- Javascript: qunit (jQuery), Jarvis, jfUnit, google-js-test

Example Python

The heart of each test is a call to *assertEqual()* to verify an expected result; *assertTrue()* or *assertFalse()* to check a condition; or *assertRaises()* to verify that a particular exception is thrown. These methods are used in place of the assert keyword so that the test runner can retrieve the results of all tests and produce a report.

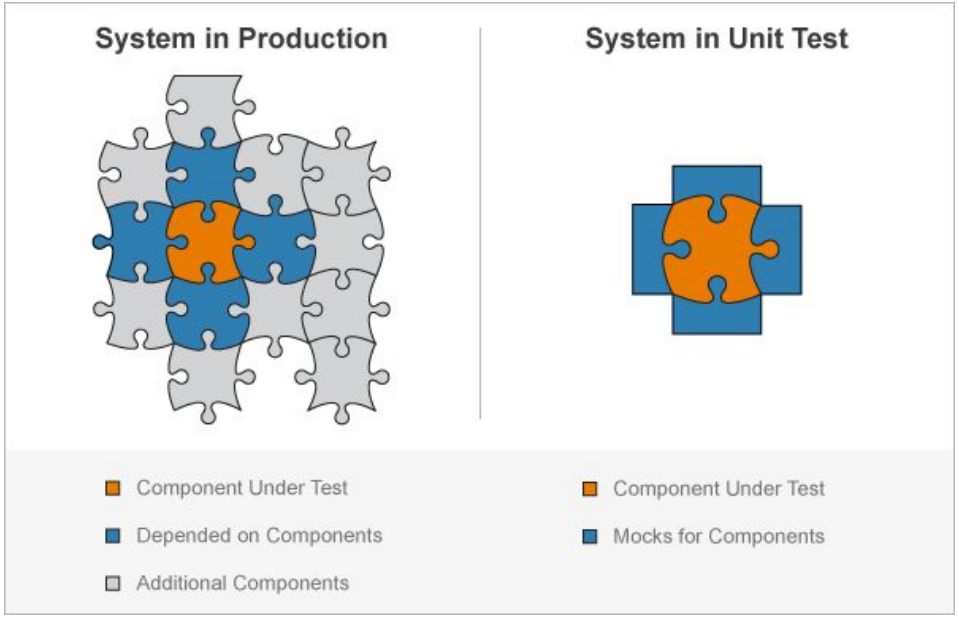
https://docs.python.org/fr/3.8/library/unittest.html

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

# bash: python -m unittest test.py
```

Unit tests: Mock

The tactic is usually to create very simple objects and methods that simply simulate the real thing. Instead of getting data from a database, you'll always return the same data from a small collection. Instead of getting a message from a remote server, you'll get it from a hard-coded method. The word "mock" (meaning "imitation" or

"fake") is often used and can be found in the names of many products helping to generate (with reflection...) code for testing.



Example Mock

```python
import unittest
from requests.exceptions import Timeout
from unittest.mock import Mock

# Mock requests to control its behavior
requests = Mock()

# function imported from your code
def create_user(user, password):

    r = requests.post('http://localhost/api/users',
                      data={"user": user, "password": password})
    if r.status_code == 200:
        return r.json()
    return None

class TesCreateUser(unittest.TestCase):

    def test_create_user_timeout(self):
        # Test a connection timeout
        requests.post.side_effect = Timeout
        with self.assertRaises(Timeout):
            create_user("jacky", "chan")
```

https://docs.python.org/fr/3.8/library/unittest.mock.html

Integration tests

The purpose of integration tests is to test the proper functioning of several components at the same time. The goal is to verify that the two components are compatible. the same libraries as unit tests can be used.

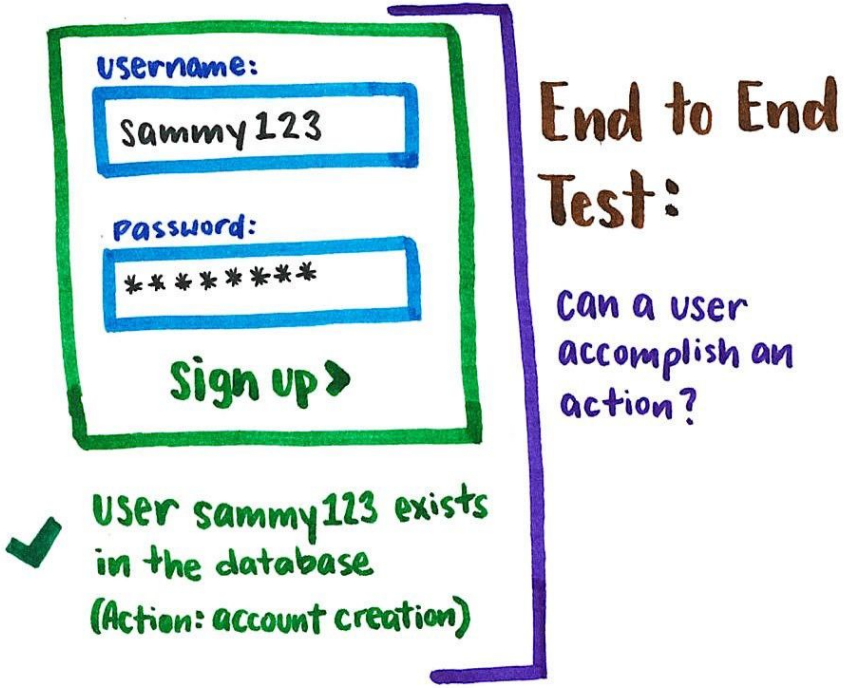
Unit tests and intergration tests approaches should be used together, instead of doing just one approach over the other. When a system is comprehensively unit tested, it makes integration testing far easier because many of the bugs in the individual components will have already been found and fixed.

Unit Test vs Integration Test

End to end (E2E) tests

“E2E tests allow us to cover sections of the application that unit tests and integration tests don’t cover. This is because unit tests and integration tests only take a small piece of the application and assess that piece in isolation.”—@xphong

End to end tests are slower that unit test and integration test and cannot be always automated.
These tests are sometimes written by the client himself.

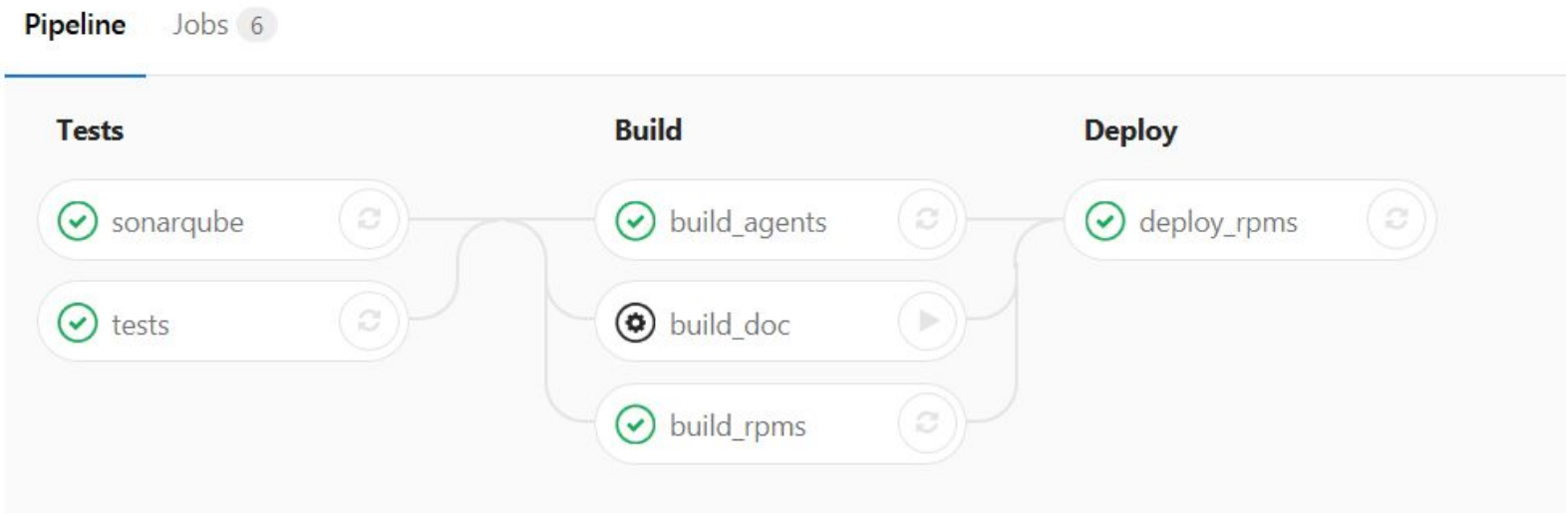


Continuous integration

The practice of automating the integration of code changes from multiple contributors into a single software project.

Continuous integration

Continuous integration is a set of practices consisting in verifying each time a source code is modified that the result of the modifications does not produce a regression in the application developed.
Today CI automated other task as source build, packaging and/or deployment.



Softwares

Jenkins

Gitlab Runner

Github packages

Travis CI

And some others Circle CI, TeamCity, Bamboo, ...

Gitlab example : GIT

esiea Menu

Search GitLab

TD_2021

- Project information
- Repository
  - Files
  - Commits
  - Branches
  - Tags
  - Contributors
  - Graph
  - Compare
- Issues 0
- Merge requests 4
- CI/CD
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Settings

GLPOO-Laval › TD_2021 › Repository

TD3+integration | TD_2021 / | History | Find file | Web IDE | Clone

Update .gitlab-ci.yml file
SWAENEPOEL Guillaume authored 10 hours ago
4e896548

Name	Last commit	Last update
controller	Ajout de la correction d'intégration contin...	11 hours ago
model	Ajout de la correction d'intégration contin...	11 hours ago
tests	Ajout de la correction d'intégration contin...	11 hours ago
view	Ajout de la correction d'intégration contin...	11 hours ago
.gitignore	Ajout de la correction d'intégration contin...	11 hours ago
.gitlab-ci.yml	Update .gitlab-ci.yml file	10 hours ago
README.md	Ajout de la correction d'intégration contin...	11 hours ago
exceptions.py	Ajout de la correction d'intégration contin...	11 hours ago
main.py	Ajout de la correction d'intégration contin...	11 hours ago
requirements.txt	Ajout de la correction d'intégration contin...	11 hours ago
tests.sh	Ajout de la correction d'intégration contin...	11 hours ago

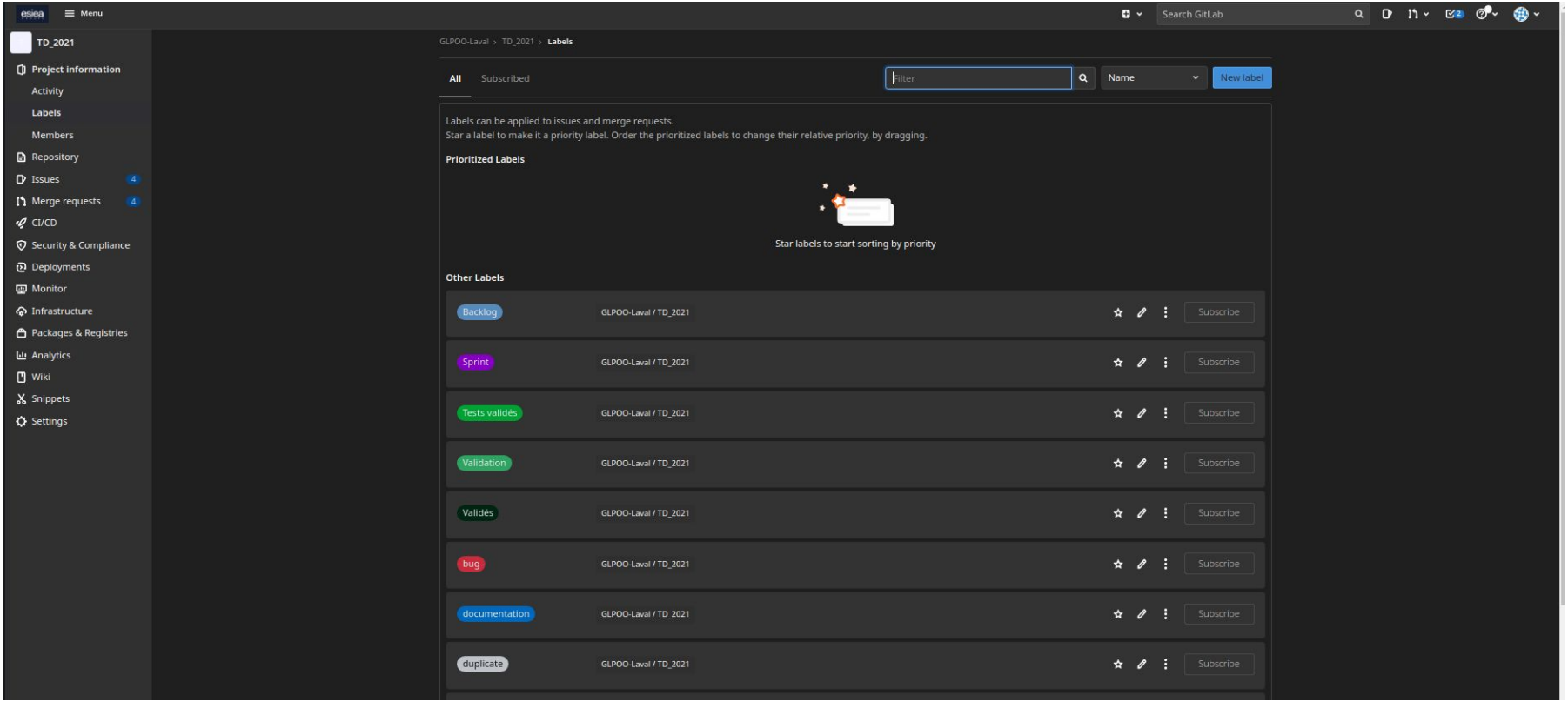
README.md

TD cours de genie logiciel 2021

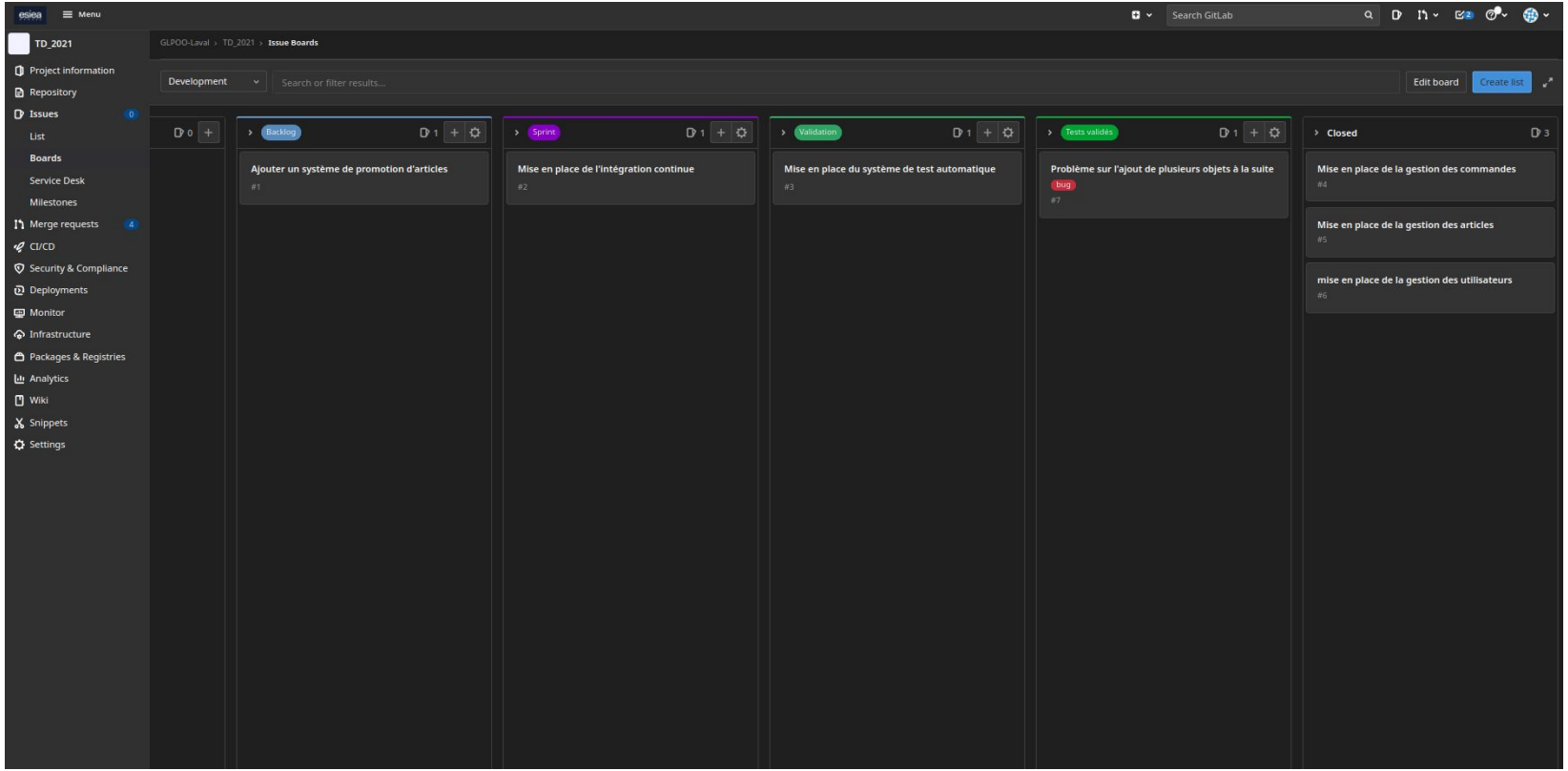
Vous trouverez le code de chacun des TD dans les différentes branches de ce repo

TD1

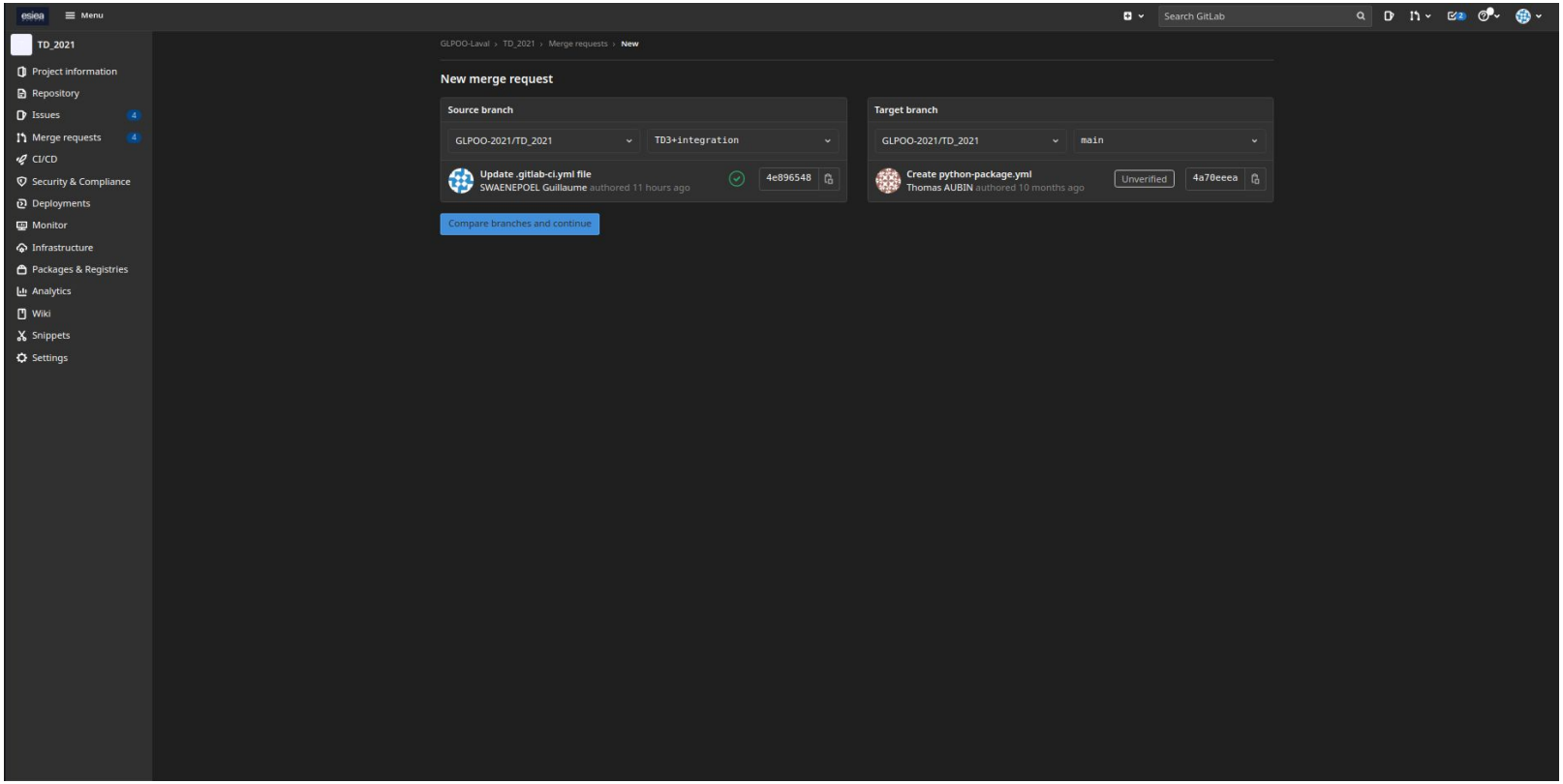
Labels


esiea
Menu
Search GitLab
TD_2021
Project information
Activity
Labels
Members
Repository
Issues 4
Merge requests 4
CI/CD
Security & Compliance
Deployments
Monitor
Infrastructure
Packages & Registries
Analytics
Wiki
Snippets
Settings
GLPOO-Laval › TD_2021 › Labels
All
Subscribed
Filter
Name
New label
Labels can be applied to issues and merge requests.
Star a label to make it a priority label. Order the prioritized labels to change their relative priority, by dragging.
Prioritized Labels
Star labels to start sorting by priority
Other Labels
Backlog
GLPOO-Laval / TD_2021
Subscribe
Sprint
GLPOO-Laval / TD_2021
Subscribe
Tests validés
GLPOO-Laval / TD_2021
Subscribe
Validation
GLPOO-Laval / TD_2021
Subscribe
Validés
GLPOO-Laval / TD_2021
Subscribe
bug
GLPOO-Laval / TD_2021
Subscribe
documentation
GLPOO-Laval / TD_2021
Subscribe
duplicate
GLPOO-Laval / TD_2021
Subscribe

Gitlab example : Tasks



Gitlab example : merge request



Gitlab example : Pipeline List

esiea Menu

Search GitLab

TD_2021

- Project information
- Repository
- Issues 4
- Merge requests 4
- CI/CD
  - Pipelines
  - Editor
  - Jobs
  - Schedules
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Settings

GLPOO-Laval › TD_2021 › Pipelines

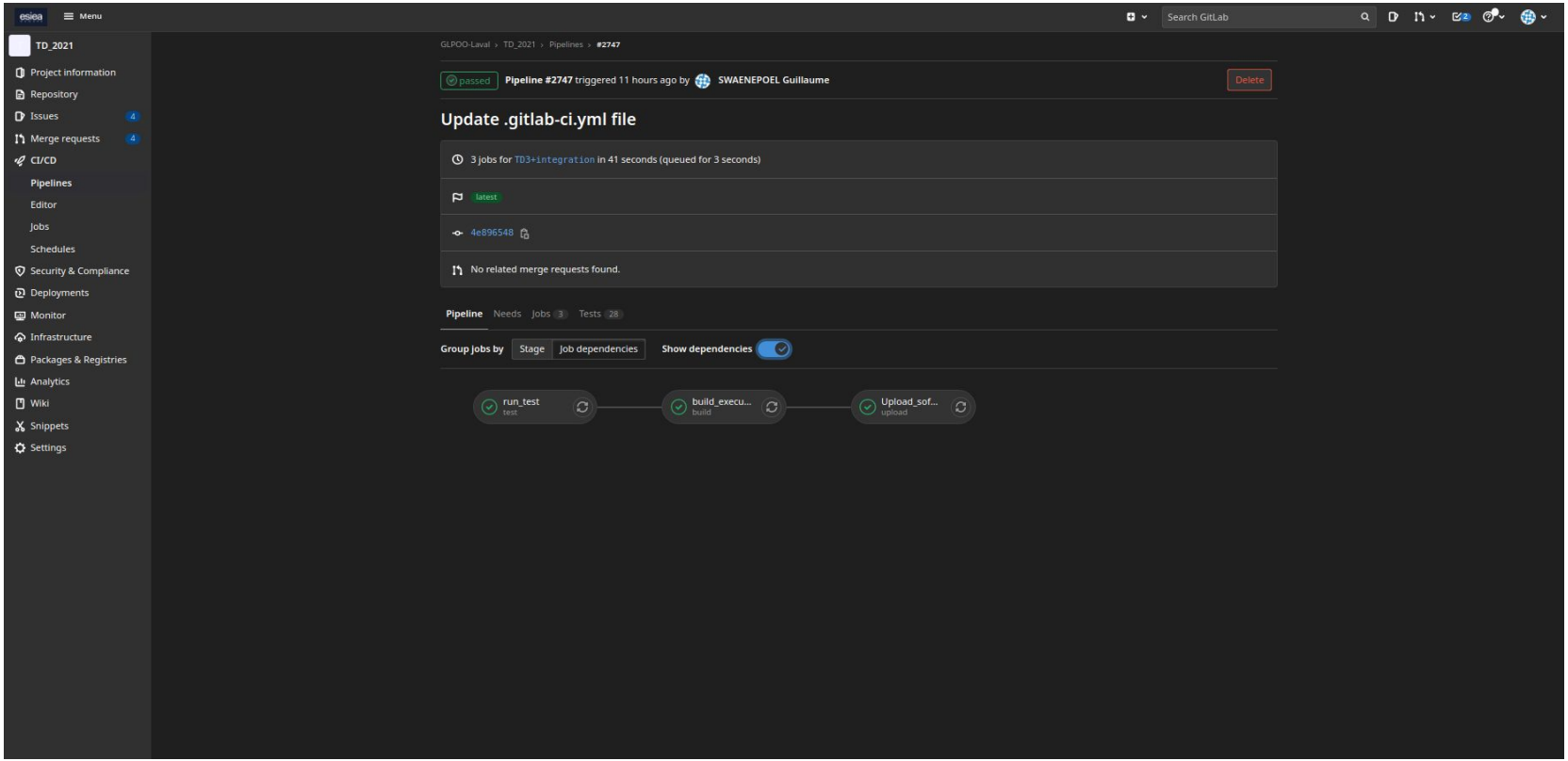
All 16 | Finished | Branches | Tags

Clear runner caches | CI lint | Run pipeline

Filter pipelines | Show Pipeline ID

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
passed	#2747 latest		TD3+integra… 4e896548 Update .gitlab-ci.yml file		00:00:41 11 hours ago
passed	#2746		TD3+integra… 14c2489e Update .gitlab-ci.yml file		00:00:45 11 hours ago
passed	#2745		TD3+integra… 6806d8a1 Update .gitlab-ci.yml file		00:00:37 11 hours ago
passed	#2744		TD3+integra… 6f19508a Update .gitlab-ci.yml file		00:01:05 11 hours ago
passed	#2743		TD3+integra… 29c4204c Update .gitlab-ci.yml file		00:01:10 11 hours ago
passed	#2742		TD3+integra… 8a65159b Update .gitlab-ci.yml file		00:00:30 11 hours ago
failed	#2741 yaml invalid error		TD3+integra… b9f05c09 Update .gitlab-ci.yml file		11 hours ago
failed	#2740		TD3+integra… 91f0207f Update .gitlab-ci.yml file		00:00:16 11 hours ago
failed	#2739		TD3+integra… f68d93e3 Update .gitlab-ci.yml file		00:00:19 11 hours ago
failed	#2738		TD3+integra… 3a85c4a5 Update .gitlab-ci.yml file		00:00:20 11 hours ago
failed	#2737		TD3+integra… b7e7ecf9 Update .gitlab-ci.yml file		00:00:35 11 hours ago

Gitlab example : Pipeline details



Gitlab example : Pipeline Tests details

esiea Menu

Search GitLab

TD_2021

- Project information
- Repository
- Issues 4
- Merge requests 4
- CI/CD
  - Pipelines
  - Editor
  - Jobs
  - Schedules
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Settings

GLPOO-Laval › TD_2021 › Pipelines › #2747

passed **Pipeline #2747** triggered 11 hours ago by SWAENEPOEL Guillaume

Delete

Update .gitlab-ci.yml file

3 jobs for TD3+integration in 41 seconds (queued for 3 seconds)

latest

4e896548

No related merge requests found.

Pipeline Needs Jobs 3 Tests 28

run_test

28 tests | 0 failures | 0 errors | 100% success rate | 239.00ms

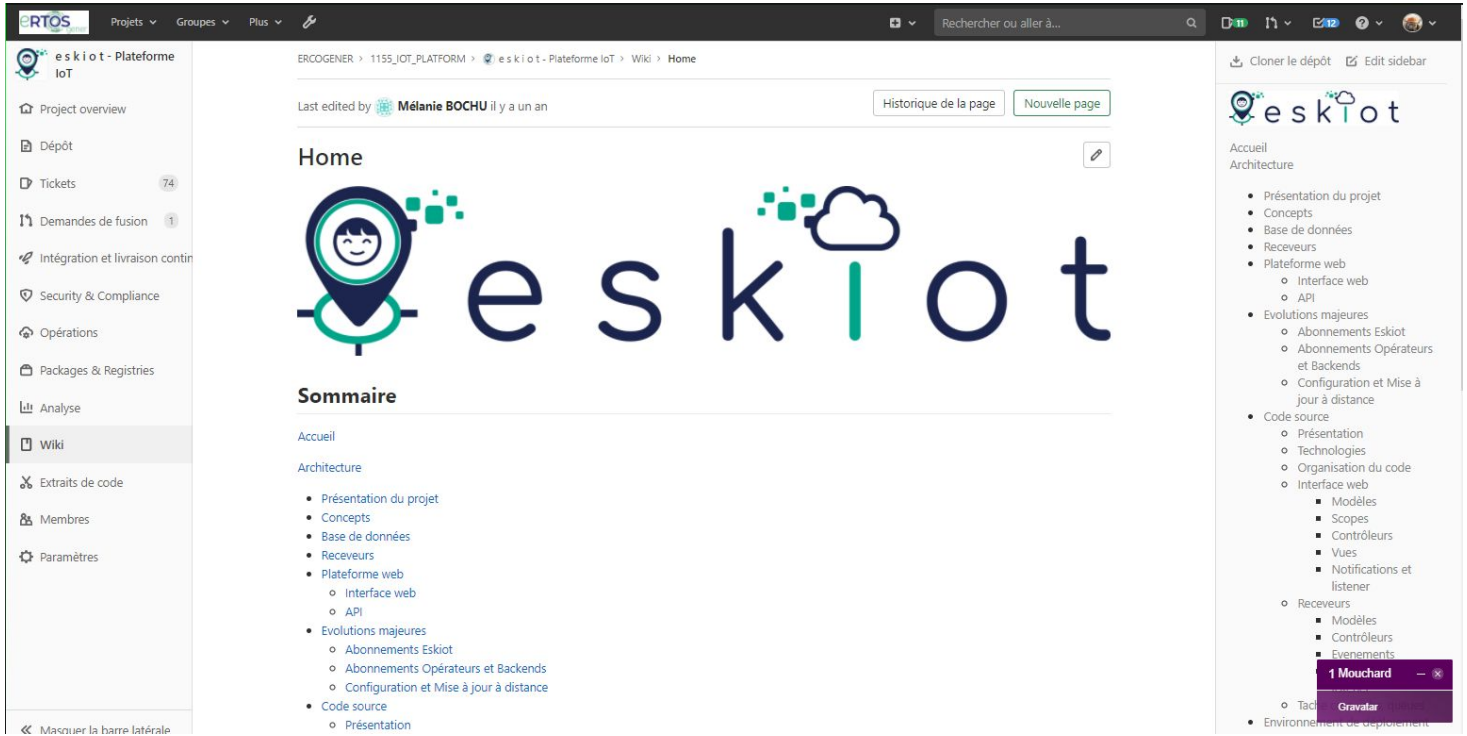
Tests

Suite	Name	Filename	Status	Duration	Details
tests.integration.test_command_builder.TestCommandBuilder	test_add_article		✓	64.00ms	View details
tests.integration.test_user_dao.TestUserDAO	test_get_user_by_username_not_found		✓	23.00ms	View details
tests.integration.test_user_dao.TestUserDAO	test_get_user_by_name_not_found		✓	21.00ms	View details
tests.integration.test_user_dao.TestUserDAO	test_get_user		✓	19.00ms	View details
tests.integration.test_user_dao.TestUserDAO	test_get_user_by_name		✓	18.00ms	View details
tests.integration.test_user_dao.TestUserDAO	test_get_user_by_username		✓	17.00ms	View details
tests.integration.test_user_dao.	test_get_user_not_found		✓	17.00ms	View details

Gitlab yml example

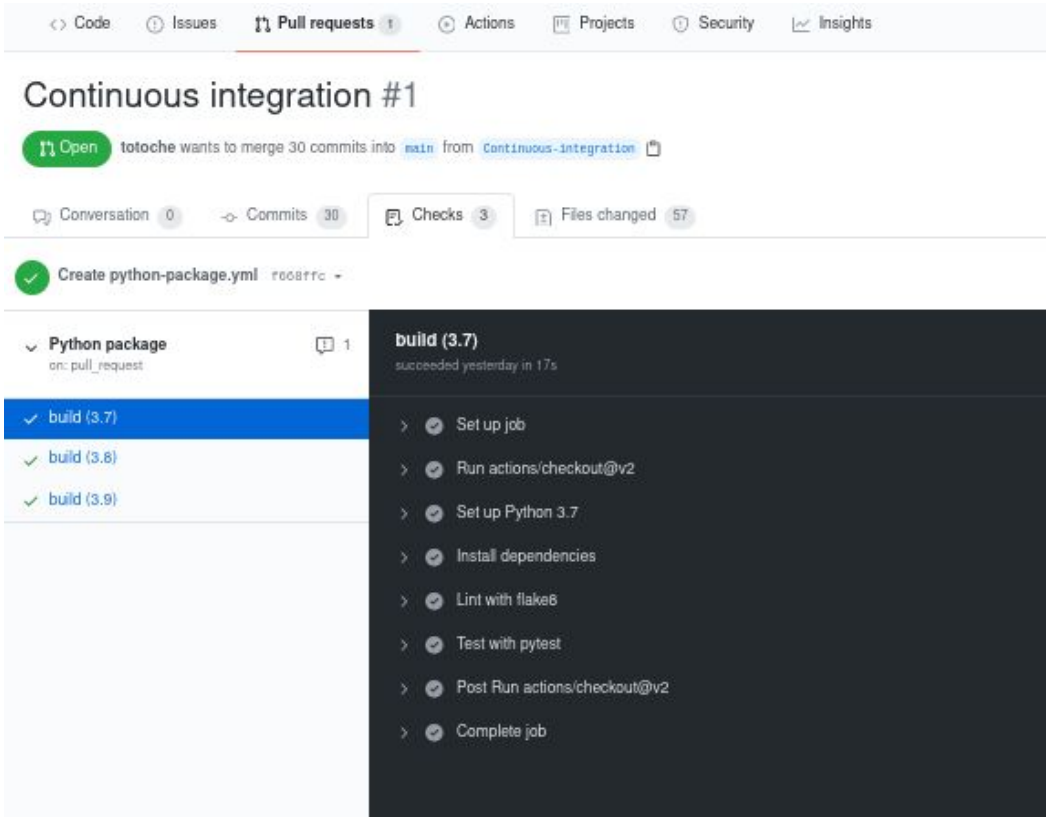
```yaml
image: python:latest

stages:
  - test
  - build
  - upload

variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

cache:
  paths:
    - .cache/pip
    - venv/

run_test:
  stage: test
  only:
    - TD3+integration

  before_script:
    - python --version  # For debugging
    - pip install virtualenv
    - virtualenv venv
    - source venv/bin/activate
    - pip install -r requirements.txt
    - pip install flake8 pytest
  script:
    - pytest --junitxml=report.xml
  artifacts:
    when: always
    reports:
      junit: report.xml
….
…
build_executable:
  stage: build
  only:
    - TD3+integration
  script:
    # Use a script to create a package here
    - echo "not implemented"
  needs: [run_test]

Upload_software:
  stage: upload
  only:
    - TD3+integration
  script:
    # Use a script to push your software online
    - echo "not implemented"
  needs: [run_test, build_executable]
```

Gitlab example : Wiki



Github packages example

https://docs.github.com/en/packages/quickstart



Questions ?

Projet

Réalisez votre Logiciel Orienté Objet ! (3-4 personnes)

- Dossier du projet (word/pdf):
  - Liste des étudiants du projet.
  - Nom & lien du projet Gitlab.
  - Explication du logiciel.
  - Explication de la répartition des tâches/organisation du travail (explications des branches git, utilisations des tâches gitlab, utilisations des merge request, etc.)
  - Schéma des cas d'utilisation du logiciel
  - Schéma séquentiel du logiciel
  - Diagramme de classe du logiciel

- Code logiciel
  - Entièrement sur Gitlab
  - MVC
  - Python3
  - Base de données comme vous le souhaitez (SQLAlchemy ou autre...)
  - Interface graphique Qt/Tkinter.
  - 2-3 fonctionnalitées (Comme TD : Gestion utilisateur, gestion articles, gestion commandes)
  - Tests (Bonus)
  - Intégration continue (Bonus)