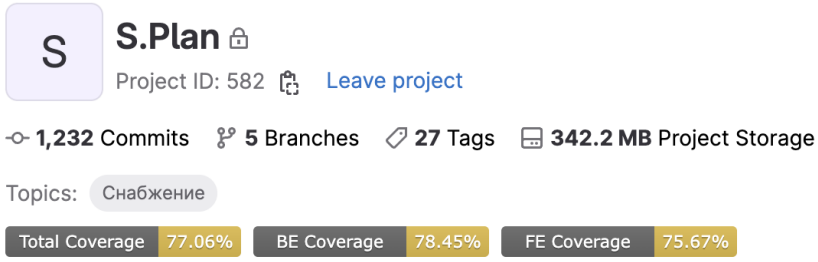
Поэтапное повышение Coverage

Цель

Вывод процента покрытия кода внутри репозитория:

- Total Coverage
- Backend (BE Coverage)
- Frontend (FE Coverage)



Условия

- Гайд написан для стандартного лэйаута проекта ГК Самолет (для кастомных лэйаутов путь будет другой)
- Гайд подразумевает, что на проекте есть DevOps / Release Engineer
- `.pre-commit-config.yaml` служит источником правды для DevOps / Release Engineer при переносе команд в пайплайн

Подготовительные шаги по BE

Чтобы интеграция в пайплайн CI / CD прошла успешно, нужно сделать определенные подготовительные шаги.

Шаги ниже производятся ролями TechLead или Senior Backend.

1. Создаем ветку `pre-commit`
2. Добавляем инструмент .pre-commit в проект - `poetry add pre-commit --group dev`
3. Добавляем black - `poetry add black --group dev`
4. Добавляем конфиг black в `pyproject.toml` - пример конфига по ссылке
5. Пропускаем код через black - `black app`
6. Добавляем isort - `poetry add isort --group dev`
7. Добавляем конфиг isort в `pyproject.toml` - пример конфига по ссылке
8. Пропускаем код через isort - `isort app`
9. Добавляем pytest - `poetry add pytest --group dev`
10. Добавляем pytest-django - `poetry add pytest-django --group dev`
11. Добавляем конфиг pytest-django в `pyproject.toml` - пример конфига по ссылке
12. Добавляем coverage - poetry add coverage --group dev
13. Добавляем конфиг coverage в `pyproject.toml` - пример конфига по ссылке
14. Добавляем .pre-commit хуки в `.pre-commit-config.yaml` - примеры конфига по ссылке
15. Мержим `pre-commit` в `main`
16. Каждый разработчик на проекте выполняет команды на своей локальной машине:
    a. `git pull main`
    b. `poetry install`
    c. `pre-commit clean`
    d. `pre-commit install`
17. Опционально, можно добавить интеграцию `black` в редактор кода - подробности по ссылке
18. Опционально, можно добавить интеграцию `isort` в редактор кода - подробности по ссылке

Интеграция в пайплайн CI / CD

Чтобы вывести покрытие на странице репозитория, нужно сделать следующие шаги.

Шаги ниже производятся ролями DevOps / Release Engineer.

1. Переходим в репозиторий с джобами
2. Переходим в папку кластера или создаем ее, если нет
3. Создаем новую папку проекта, называем `project` (например, `splan`)

4. Копируем содержимое папки splan в качестве примера, при необходимости изменяем команды запуска тестов под свой проект. Для того чтобы `coverage` корректно читался, тесты должны выводить его в консоль, для этого `pytest` нужно запускать с параметром `--cov-report term-missing`, а `jest` с `--coverage --coverageReporters=cobertura --coverageReporters="text-summary"`.
5. Копируем содержимое файла, создаем файл `project-python-ci.yaml` и меняем пути
6. Добавляем переменную `DOCKER_AUTH_CONFIG` в основной файл воркфлоу, созданный на предыдущем шаге - пример тут
7. Приводим команды из образца в соответствие со своими командами запуска в `.pre-commit-config.yaml`, но в режиме проверки
8. Сконфигурированные юнит-тесты (`pytest` для бэка и `jest` для фронта) начинают запускаться в пайплайне при мердж реквестах и запуске с ветки `main`.
9. Контролируем, что артефакты корректно загружаются в GitLab.
10. Если `coverage` успешно читается, то его можно увидеть в правой панели при открытии соответствующей джобы.

Стадии в пайплайне

Пайплайн состоит из 3 stages:

1. **build**. На стадии build происходит сборка докер образов, эта стадия проходит одинаково при любом запуске.
2. **test**. На стадии test запускаются форматеры, линтеры и юнит-тесты.
3. **deploy**. На стадии deploy происходит деплой образа.

Стадии **build** и **deploy** не требуют изменений.

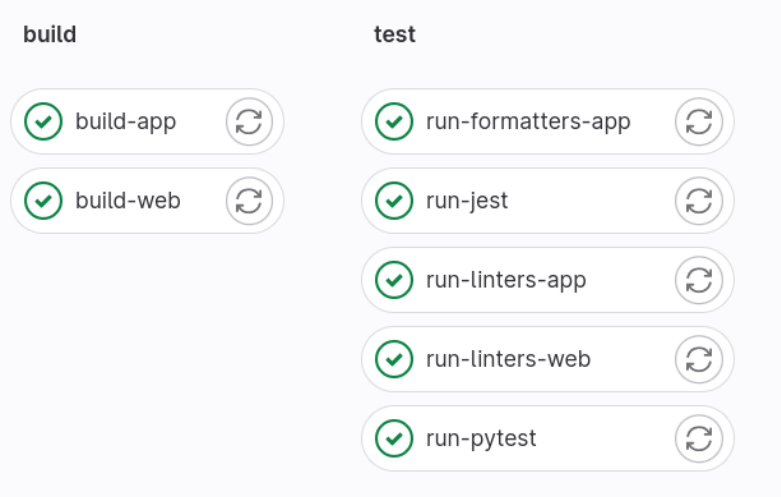
Запуск джоб на стадии tests

- Форматеры и линтеры запускаются при создании MP
- Юнит-тесты запускаются при создании MP
- Юнит-тесты запускаются при коммите в `main`.

Джобы которые запускаются при создании MR:

- **run-formatters-app** (запускает `autoflake`, `isort` и `black`)
- **run-jest** (запуск юнит-тестов фронта)
- **run-pytest** (запуск юнит-тестов бэка)
- **run-linters-app** (запускает `pylint`, `flake8`, `mypy` и `bandit`)
- **run-linters-web**

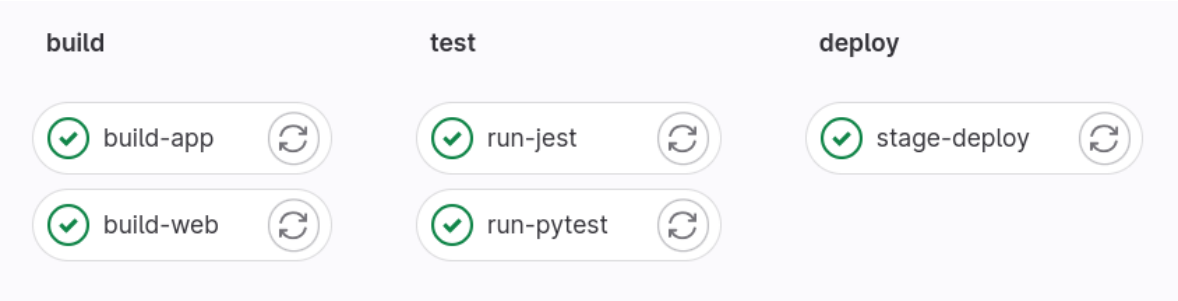
Пример запуска пайплайна при создании или изменении MP:



Джобы которые запускаются при коммите в main:

- **run-jest** (запуск юнит-тестов фронта)
- **run-pytest** (запуск юнит-тестов бэка)

Пример запуска пайплайна с ветки `main` после принятия MP:



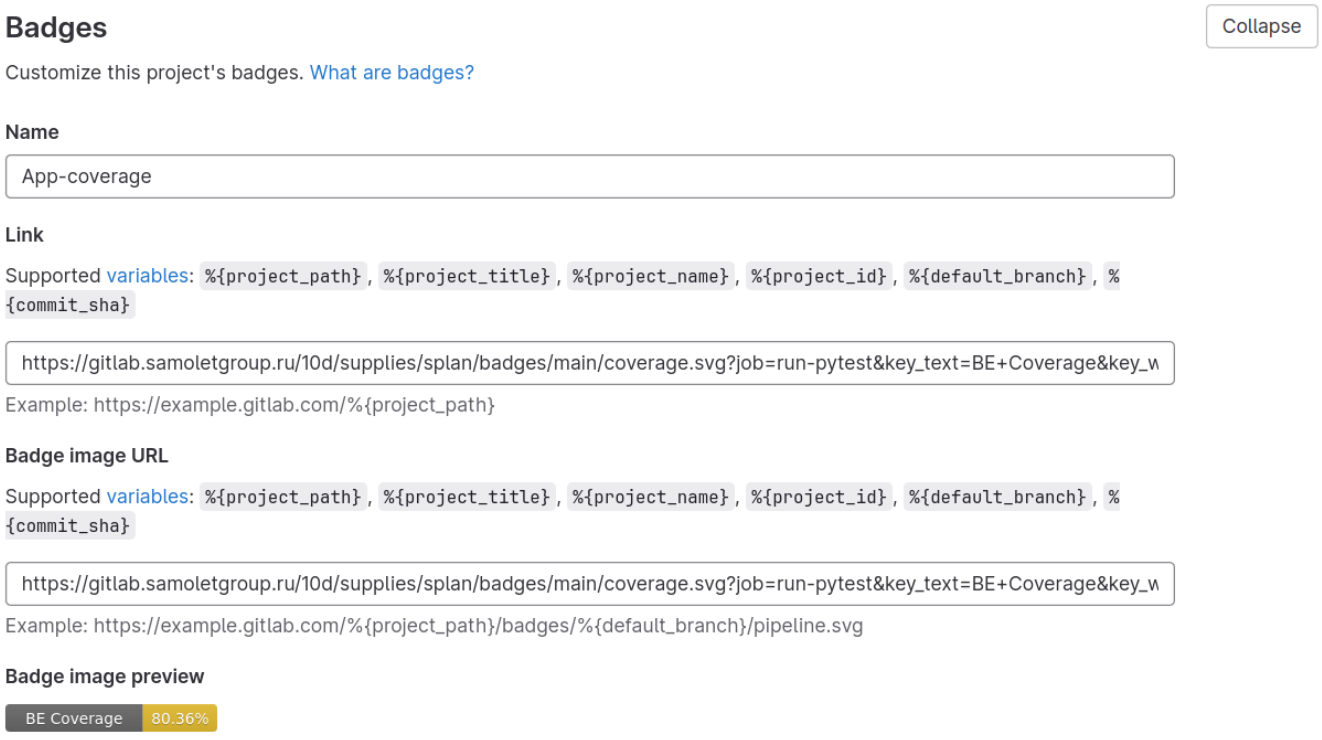
Добавление бэйджей Coverage

Для добавления бейджей заходим в settings, general, badge. Добавляем 3 бейджа. Поле name можно заполнить произвольно, оно не выводится. В оба поля для ссылок вставляем ссылки приведенные ниже, подставляя вместо 10d/supplies/splan путь до своего проекта. Лучше соблюдать очередность добавления, т. к. отображаться бейджи будут в этом порядке.

https://gitlab.samoletgroup.ru/10d/supplies/splan/badges/main/coverage.svg?key_text=Total+Coverage&key_width=100

https://gitlab.samoletgroup.ru/10d/supplies/splan/badges/main/coverage.svg?job=run-pytest&key_text=BE+Coverage&key_width=100

https://gitlab.samoletgroup.ru/10d/supplies/splan/badges/main/coverage.svg?job=run-jest&key_text=FE+Coverage&key_width=100



Примечания

1. При любом изменении `.pre-commit-config.yaml` нужно уведомлять релиз инженера для того чтобы поддерживать пайплайн в актуальном состоянии.
2. Дополнительный запуск тестов при коммите в `main` нужен для отоброжения coverage в бейджах, т.к. при создании бейджа нужно указать ветку, указать MR нельзя.