

# # P10 - CI/CD - BobApp

Contents

Continuous Integration/Continuous Delivery

GitHub Actions

SonarQube

Implémentation des actions

Explications

Back-end

Front-end

Exécution

Rapport

Quality Gates

Recommandation Quality-Gates

Tests

Rapport de couverture

Makefile

Déploiement Docker

Docker Compose

Dockerfile

Nginx

Nginx.conf

Makefile

Docker Image

Commandes

Makefile

Github Actions

Notes et avis des utilisateurs

Points pertinents

Conclusion

Valider les tests

Vérifier la qualité du code


Quality Gates

Builder le projet

Gérer les images Docker

Permettre de lancer les tests

Accéder au rapport de couverture de code des tests

 Repository GitHub : [P10](#) →

## Continuous Integration/Continuous Delivery

### GitHub Actions

Les Github-actions sont des `jobs` qui peuvent être lancés à la détection d'évènements en lien avec le code hébergé.

Un dossier est rajouté à la base du code base.

```
1 .
2 |— .github
3 |   |— workflows
4 |     |— build.yml
5
6 |— back
7 |— front
8 |— .git
9 |— .gitignore
10 |— README.md
```

### SonarQube

Vérification de la qualité du code.

SonarQube vérifie que la qualité du code est maintenue à chaque `push` et `merge` via des *GitHub-Actions*.

### Implémentation des actions

L'implémentation se fait de cette façon pour le projet **BobApp**.

```
1 name: SonarQube
2 on:
3   push:
4     branches:
5       - main
6   pull_request:
7     types: [opened, synchronize, reopened]
8 jobs:
9   build:
10    name: Build and analyze
11    runs-on: ubuntu-latest
12    steps:
13      - uses: actions/checkout@v4
14        with:
15          fetch-depth: 0
16      - name: Set up JDK 11
17        uses: actions/setup-java@v4
18        with:
19          java-version: 11
20          distribution: 'zulu'
21      - name: Cache SonarQube packages
22        uses: actions/cache@v4
23        with:
24          path: ~/.sonar/cache
25          key: ${ runner.os }-sonar
26          restore-keys: ${ runner.os }-sonar
27      - name: Cache Maven packages
28        uses: actions/cache@v4
29        with:
30          path: ~/.m2
31          key: ${ runner.os }-m2-${ hashFiles('**/pom.xml') }
32          restore-keys: ${ runner.os }-m2
33      - name: Build and analyze
34        env:
35          SONAR_TOKEN: ${ secrets.SONAR_TOKEN }
36        run: cd back && mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=maxdlr_p10
37   sonarqube:
38    name: SonarQube
39    runs-on: ubuntu-latest
40    steps:
41      - uses: actions/checkout@v4
42        with:
43          fetch-depth: 0
44      - name: SonarQube Scan
45        uses: SonarSource/sonarqube-scan-action@v4
46        with:
47          projectBaseDir: front/
48          args: >
49            -Dsonar.organization=maxdlr
50            -Dsonar.projectKey=maxdlr_p10
51        env:
52          SONAR_TOKEN: ${ secrets.SONAR_TOKEN }
```

### Explications

#### Back-end

Action	Description	Code snippet
Checkout	L'action <i>checkout</i> le code pour pouvoir l'analyser.	<pre>1 steps: 2   - uses: actions/checkout@v4 3     with: 4       fetch-depth: 0</pre>
Java	SonarQube créer un cache pour stocker des data durant son travail.	<pre>1 - name: Set up JDK 11 2   uses: actions/setup-java@v4 3     with: 4       java-version: 11 5       distribution: 'zulu'</pre>

Installation SonarQube	Le programme d'analyse de SonarQube est téléchargé et conservé dans un cache.	<pre>1 - name: Cache SonarQube packages 2   uses: actions/cache@v4 3   with: 4     path: ~/.sonar/cache 5     key: \${{ runner.os }}-sonar 6     restore-keys: \${{ runner.os }}-sonar</pre>
Installation Maven	L'installation des librairies et plugins est lancé via Maven et conservé dans un cache.	<pre>1 - name: Cache Maven packages 2   uses: actions/cache@v4 3   with: 4     path: ~/.m2 5     key: \${{ runner.os }}-m2-\${{ 6 hashFiles('**/pom.xml') }}   restore-keys: \${{ runner.os }}-m2</pre>
Analyse SonarQube	L'analyse est enfin lancée	<pre>1 - name: Build and analyze 2   env: 3     SONAR_TOKEN: \${{ 4 secrets.SONAR_TOKEN }}   run: cd back &amp;&amp; mvn -B verify   org.sonarsource.scanner.maven:sonar-   maven-plugin:sonar -   Dsonar.projectKey=maxdlr_p10</pre>

Front-end

Action	Description	Code snippet
Checkout	L'action <i>checkout</i> le code pour pouvoir l'analyser.	<pre>1 - uses: actions/checkout@v4 2   with: 3     fetch-depth: 0</pre>
Analyse SonarQube	Pour ce job, SonarQube utilise une action pré-codé <a href="#">SonarSource/sonarqube-scan-action@v4</a>	<pre>1 - name: SonarQube Scan 2   uses: SonarSource/sonarqube-scan- 3 action@v4 4   with: 5     projectBaseDir: front/ 6     args: &gt; 7       -Dsonar.organization=maxdlr 8       -Dsonar.projectKey=maxdlr_p10 9   env: 10     SONAR_TOKEN: \${{ secrets.SONAR_TOKEN 11 }}</pre>

Exécution

Les deux actions sont lancées simultanément dès qu'une action définis dans le fichier est détectée. D'où le début du fichier :

```
1 on:
2   push:
3     branches:
4       - main
5   pull_request:
6     types: [opened, synchronize, reopened]
```

Les actions sont donc lancées au `push` sur la branche main et dès qu'une `pull-request` est ouverte, synchronisée ou réouverte.

Rapport

Summary

Jobs

Build and analyze

SonarQube

Run details

Usage

Workflow file

Triggered via push 39 minutes ago

maxdlr pushed 94b509d main

Status

Success

Total duration

1m 26s

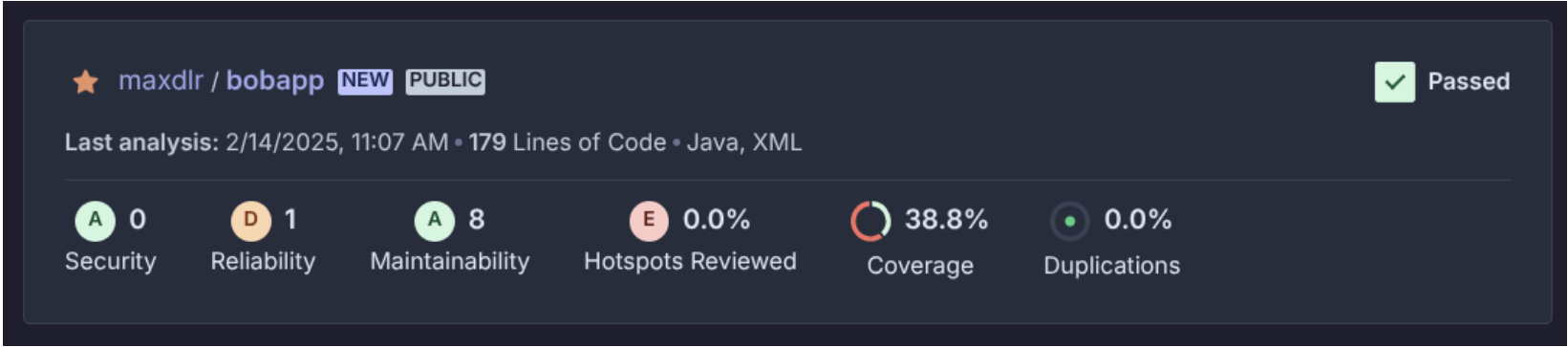
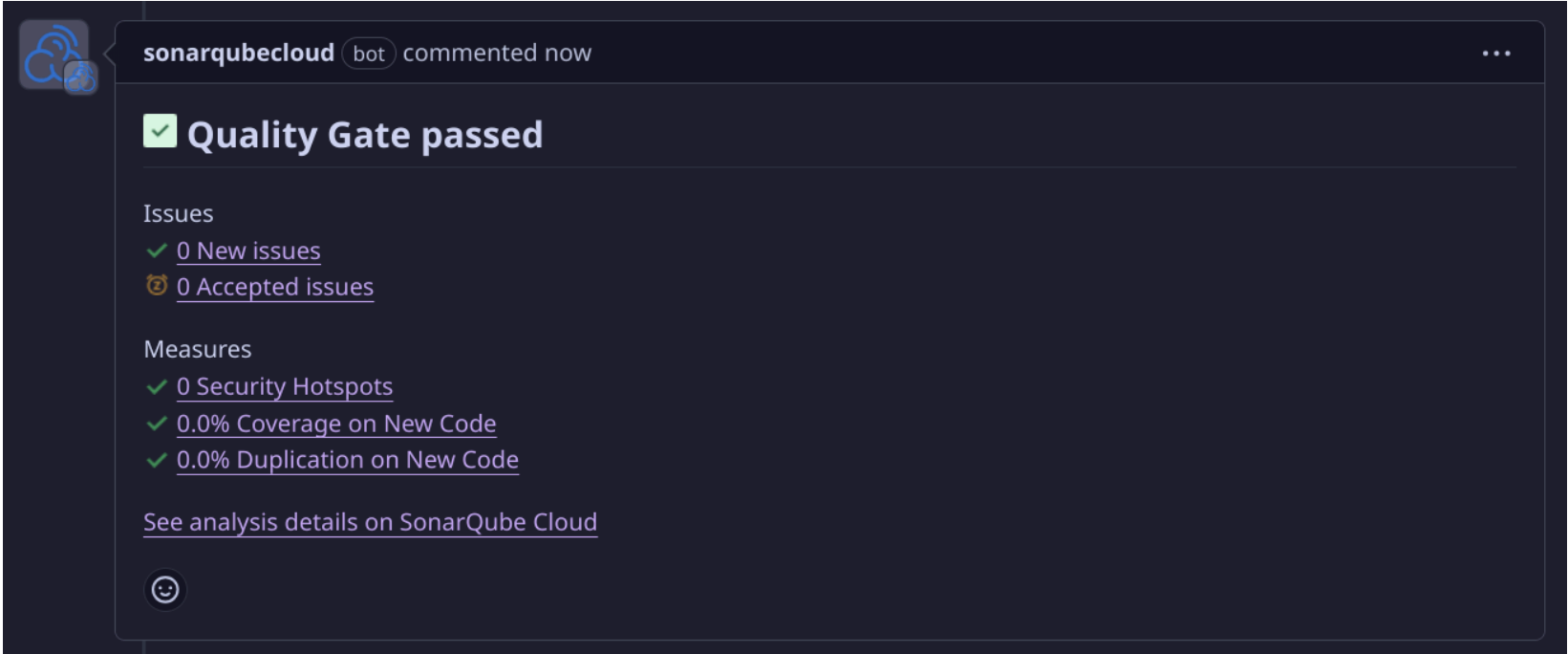
build.yml

on: push

Build and analyze1m 14s

SonarQube51s

Suite à l'implémentation des jobs, j'ai mergé la branche contenant ce code sur `main` et les 2 analyses ont réussis.



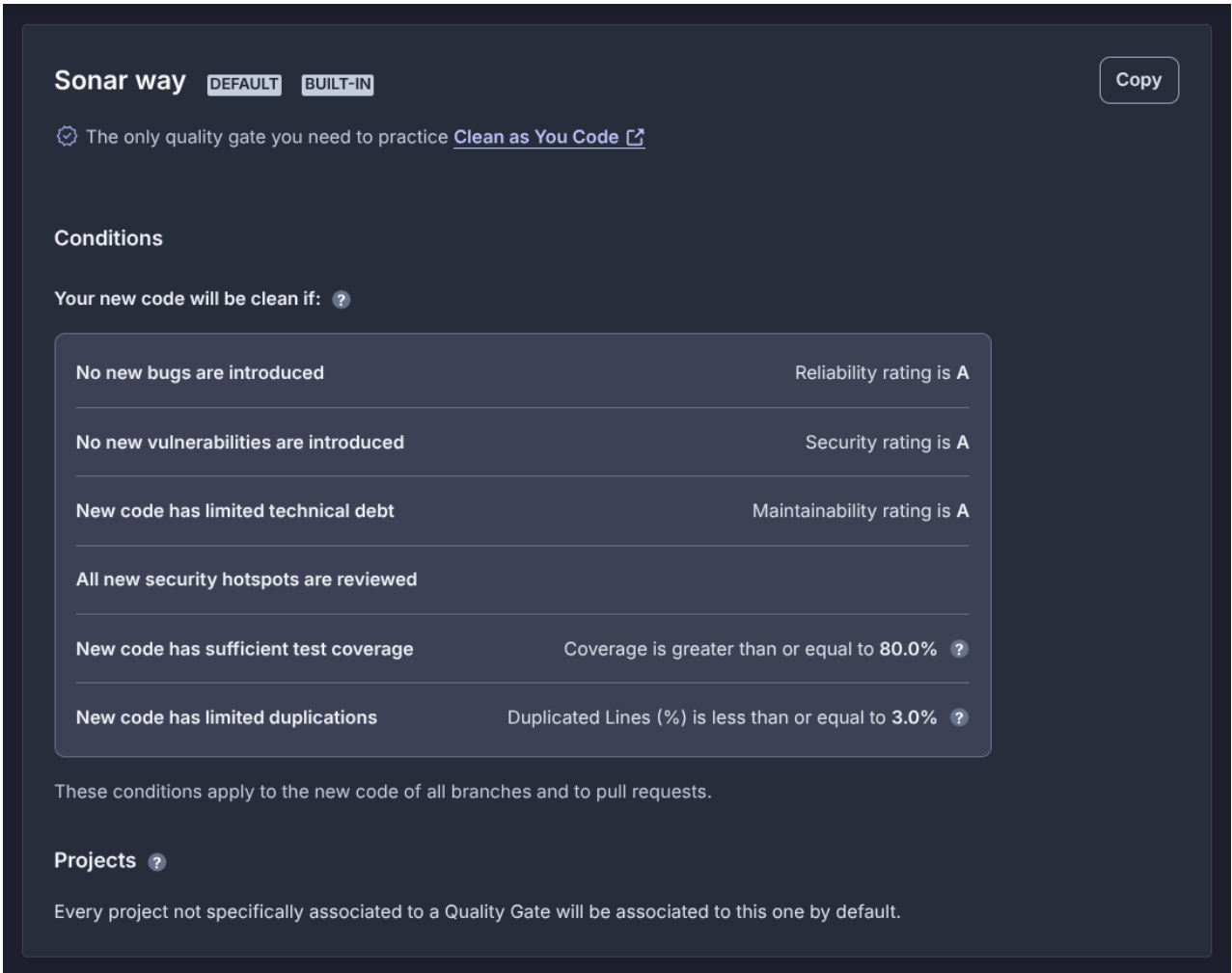
Et les résumés et résultats des analyses sont disponibles sur mon compte **SonarQube**.

### Quality Gates

Les *Quality Gates* sont des règles de qualité de code qui sont vérifiés à chaque fois que SonarQube est sollicité, c'est-à-dire à chaque fois que le job associé sera lancé.

En l'état, il faut un compte payant pour avoir des *Quality Gates* personnalisés.

Pour les comptes gratuits, nous n'avons pas d'autres choix que d'utiliser la liste de règle par défaut de SonarQube : Le "Sonar Way".



### Recommandation Quality-Gates

Honnêtement, le métier de l'application n'a aucun enjeu majeur :

- Le site ne pratique pas de paiement en ligne.
- L'impact social est moindre.
- Un bug mineur ne met pas en jeu le but, en soi, de cette application.

En partant de ce postulat, je trouve que les Quality-Gates proposés, le "Sonay way", plus que suffisant.

Ceci étant dit, dans le cas où nous avons accès à cette configuration, j'aurais au moins :

- Maintenu le taux minimum de couverture de code à 80 %.
- Ouvert la possibilité d'une note minimale du premier point, les nouveaux bugs présentés, à B ou plus.

### Tests

Avant de rajouter les tests aux actions, il serait sans doute intéressant d'empêcher le processus précédent de lancer les tests.

En effet, `mvn -B verify` lance les tests par défaut.

Dans un souci de performance, de maintenabilité et de configuration, nous allons rajouter le flag `-Dmaven.test.skip=true` au processus `Build and analyze`.

```
1 - name: Build and analyze
2   env:
3     SONAR_TOKEN: ${ secrets.SONAR_TOKEN }}
4   run: cd back && mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=maxdlr_p10 -Dmaven.test.skip=true
```

Maintenant, le lancement des tests est rajouté aux GitHub-Actions.


```
1 ...
2 - name: Tests
3   run: cd back && mvn clean test
```

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.141 s - in com.openclassrooms.bobapp.BobappApplicationTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jacoco:0.8.5:report (report) @ bobapp ---
[INFO] Loading execution data file /home/runner/work/p10/p10/back/target/jacoco.exec
[INFO] Analyzed bundle 'bobapp' with 6 classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.322 s
[INFO] Finished at: 2025-02-14T13:55:47Z
[INFO] -----
```

Les tests sont désormais lancés systématiquement.

### Rapport de couverture

Les rapports de couvertures sont générés par Jacoco dans `./back/target/site/jacoco/index.html`.

 bobapp												
bobapp												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">com.openclassrooms.bobapp.model</a>	<div><div></div></div>	0%		n/a	7	7	13	13	7	7	1	1
<a href="#">com.openclassrooms.bobapp.data</a>	<div><div></div></div>	49%	<div><div></div></div>	50%	5	7	8	18	3	5	1	2
<a href="#">com.openclassrooms.bobapp.service</a>	<div><div></div></div>	25%		n/a	1	2	4	7	1	2	0	1
<a href="#">com.openclassrooms.bobapp.controller</a>	<div><div></div></div>	54%		n/a	1	2	1	4	1	2	0	1
<a href="#">com.openclassrooms.bobapp</a>	<div><div></div></div>	37%		n/a	1	2	2	3	1	2	0	1
Total	90 of 134	32%	2 of 4	50%	15	20	28	45	13	18	2	6

### Makefile

J'ai rajouté cette commande dans un fichier `Makefile` pour améliorer l'expérience développeur.

De plus, j'ai aussi rajouté une commande pour rapidement ouvrir le rapport de couverture de code une fois le test terminé.

```
1 MAKEFLAGS += --no-print-directory -s
2
3 default: help
4
5 back-tests: ## Run Back-end tests
6   cd back && mvn clean test
7   make back-tests-report
8
9 back-tests-report: ## Open tests coverage report
10  open back/target/site/jacoco/index.html
11
12 help: ## Show this help menu
13   @awk -F ':|##' '/^[^t].+?:.*?##/ {printf "\033[36m%-30s\033[0m %s\n", $$1, $$NF}' $(MAKEFILE_LIST)
```

## Déploiement Docker

Déploiement continu.

La *dockerisation* du projet se fait en plusieurs étapes. Il faut :

- Créer une composition docker dans un `docker-compose.yml` pour orchestrer la création du/des `container` de l'app.
- Créer un `Dockerfile` unique prenant en charge le front et le back pour la création du `container` et de l'image Docker.
- Modifier la configuration `Nginx` pour remplacer le serveur proxy qui ne fonctionne qu'en développement.
- Et éventuellement, ajouter un `Makefile` pour une meilleure expérience développeur.

### Docker Compose

J'ai ajouté un `docker-compose.yml` à la racine du projet.

Il donne un nom au `container` : "bobapp".

Il expose le port `4200` sur la machine hôte à partir du port `80` qu'expose le `container`.

```
1 services:
2   bobapp:
3     container_name: bobapp
4     build:
5       context: .
6       dockerfile: Dockerfile
7     ports:
8       - "4200:80"
```

### Dockerfile

J'ai remplacé les 2 `Dockerfile` respectivement présents dans `./front/` et `./back/` par un `Dockerfile` unique à la racine du projet.

```
1 FROM node:latest as frontend-build
2 WORKDIR /usr/local/app
3 COPY ./front/ /usr/local/app/
4 RUN yarn
5 RUN npm run build
6
7 FROM maven:3.6.3-jdk-11-slim as backend-build
8 WORKDIR /workspace
9 COPY ./back/pom.xml /workspace
10 COPY ./back/src /workspace/src
11 RUN mvn -B -f pom.xml clean package -DskipTests
12
13 FROM nginx:latest
14 RUN apt update && \
15     apt install -y openjdk-17-jre-headless && \
16     apt clean && \
17     rm -rf /var/lib/apt/lists/*
18
19 COPY --from=frontend-build /usr/local/app/dist/bobapp /usr/share/nginx/html
20 COPY ./front/nginx.conf /etc/nginx/conf.d/default.conf
21
22 COPY --from=backend-build /workspace/target/*.jar /app/app.jar
23
24 RUN echo '#!/bin/bash\n\
25 java -jar /app/app.jar &\n\
26 nginx -g "daemon off;"\n' > /start.sh && \
27 chmod +x /start.sh
28
29 EXPOSE 80 8080
30 CMD ["/start.sh"]
```

Le `Dockerfile` utilise trois images : `node` , `maven` et `nginx` .

`node` nous sert à déployer l'application : fabriquer un `build` .

`maven` nous sert compilé le backend : fabriquer un `package` .

`nginx` nous servira de serveur proxy pour une meilleure gestion des configurations de requêtes avec le fichier `nginx.conf` .

Ensuite, à partir de `nginx` , j'installe `openjdk` , copie les 2 `builds` précédemment construis et la config `nginx.conf` .

Puis, je créé un script `bash` qui exécute le fichier `app.jar` et `nginx` pour lancer le projet.

### Nginx

En développement, `Angular` nous laisse créer un serveur proxy de développement avec un simple fichier de configuration : `proxy.config.json` .

Cela permet d'éviter les erreurs de `CORS` en faisant croire à l'application back-end que l'origine de la requête et celle de la réponse, celle du back-end, est la même.

```
1 {
2   "/api/*": {
3     "target": "http://localhost:8080/",
4     "secure": false,
5     "changeOrigin": true,
6     "logLevel": "debug"
7   }
8 }
```

Avec cette simple configuration, on transforme toutes nos requêtes API, `.../api/*...` en remplaçant l'hôte du back-end avec celle front-end.

En production, nous devons vraiment configurer le serveur pour qu'il reproduise ce comportement.

Ceci dit, il serait sans doute préférable d'implémenter la gestion des origines des requêtes dans le code du back-end.

### Nginx.conf

Voici la nouvelle config dans `.front/nginx.conf`

```
1 server {
2   listen 80;
3   sendfile on;
4   default_type application/octet-stream;
5
6   gzip on;
7   gzip_http_version 1.1;
8   gzip_disable      "MSIE [1-6]\.";
9   gzip_min_length   256;
10  gzip_vary          on;
11  gzip_proxied        expired no-cache no-store private auth;
12  gzip_types          text/plain text/css application/json application/javascript application/x-javascript text/xml application/xml application/xml+rss
13  text/javascript;
14  gzip_comp_level     9;
15
16  root /usr/share/nginx/html;
17
18  location /api {
19    proxy_pass http://localhost:8080;
20    proxy_http_version 1.1;
21    proxy_set_header  Host $host;
22    proxy_set_header  X-Real-IP $remote_addr;
23    proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
24    proxy_set_header  X-Forwarded-Proto $scheme;
25    proxy_connect_timeout 60;
26    proxy_send_timeout 60;
27    proxy_read_timeout 60;
28  }
29
30  location / {
31    try_files $uri $uri/ /index.html =404;
32  }
33 }
```

Ici, je rajoute la configuration (lignes : 17-27) de serveur proxy avec l'hôte `http://back-end:8080` .

### Makefile

C'est optionnel, mais recommandé, on uniformise les commandes à lancer pour certaines actions liées au projet.

Cela optimise aussi l'expérience développeur.

```
1 MAKEFLAGS += --no-print-directory -s
2
3 default: help
4
5 run: ## Start project
6     docker compose up --force-recreate --build --remove-orphans
7
8 back-tests: ## Run Back-end tests
9     cd back && mvn clean test
10    make back-tests-report
```

```
11
12 back-tests-report: ## Open tests coverage report
13     open back/target/site/jacoco/index.html
14
15 help: ## Show this help menu
16     @awk -F ' :|##' '/^[^t].+?:.*?##/ {printf "\033[36m%-30s\033[0m %s\n", $$1, $$NF}' $(MAKEFILE_LIST)
```

Docker Image

J'ai créé un repository sur DockerHub → pour pouvoir mettre à jour l'image du projet.

Commandes

Cela se fera avec 2 commandes :

- 1. D'abord, on build l'image.

```
1 docker build -t maxdlr/bobapp:latest .
```

- 2. Puis on l'envoie sur le repository distant.

```
1 docker push maxdlr/bobapp:latest
```

Makefile

Je me suis permis de rajouter ces commandes dans une commande Makefile .

```
1 MAKEFLAGS += --no-print-directory -s
2
3 default: help
4
5 run: ## Start project
6     docker compose up --force-recreate --build --remove-orphans
7
8 back-tests: ## Run Back-end tests
9     cd back && mvn clean test
10    make back-tests-report
11
12 back-tests-report: ## Open tests coverage report
13     open back/target/site/jacoco/index.html
14
15 update-docker-image: ## Build and Push new Docker image
16     docker build -t maxdlr/bobapp:latest .
17     docker push maxdlr/bobapp:latest
18
19 help: ## Show this help menu
20     @awk -F ' :|##' '/^[^t].+?:.*?##/ {printf "\033[36m%-30s\033[0m %s\n", $$1, $$NF}' $(MAKEFILE_LIST)
```

Github Actions

J'ai rajouté une action GitHub à exécuter sur tous les push vers la branche main . Le fichier est dans le même dossier que le précédent : deploy.yml .

```
1 .
2 |— .github
3 |   |— workflows
4 |     |— build.yml
5 |     |— deploy.yml
6
7 |— back
8 |— front
9 |— .git
10 |— .gitignore
11 |— README.md
```

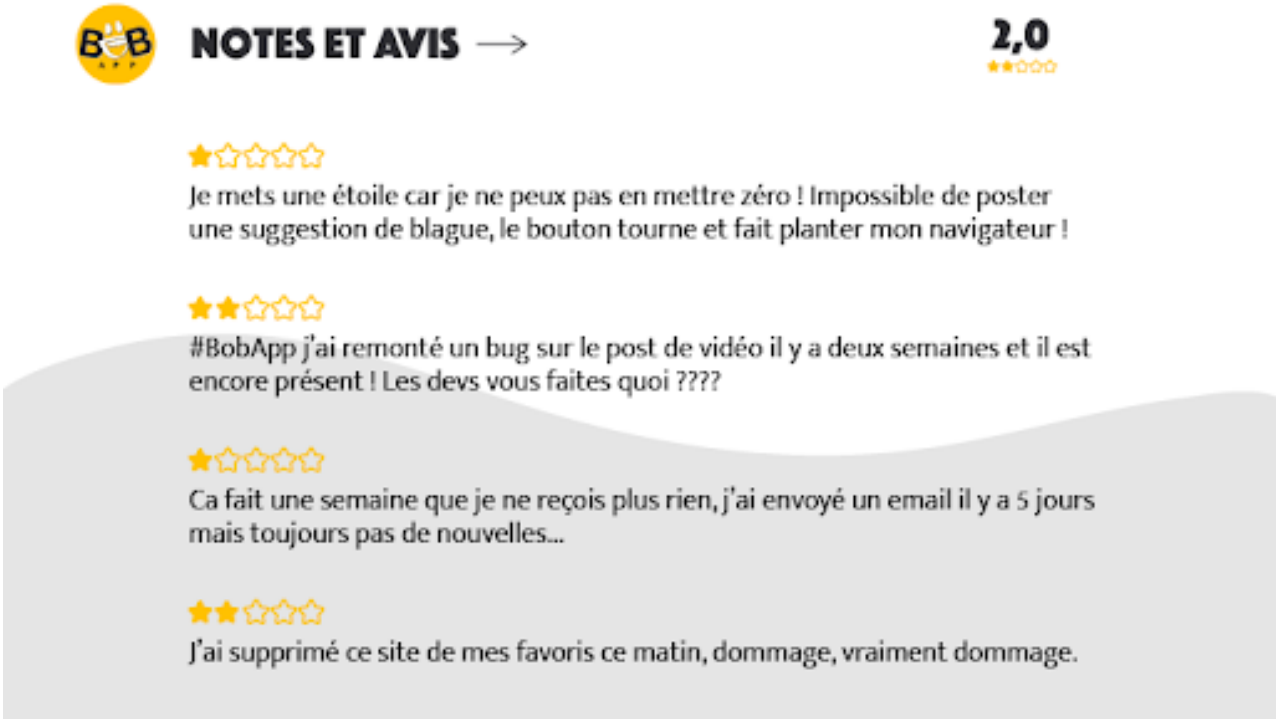
L'action reproduit la commande du Makefile.

```
1 on:
2   push:
3     branches:
4       - main
5 jobs:
6   docker:
7     runs-on: ubuntu-latest
8     steps:
9
10      - name: Login to Docker Hub
11        uses: docker/login-action@v3
12        with:
13          username: ${ secrets.DOCKERHUB_USERNAME }
14          password: ${ secrets.DOCKERHUB_TOKEN }
15
16      - name: Set up QEMU
17        uses: docker/setup-qemu-action@v3
18
19      - name: Set up Docker Buildx
20        uses: docker/setup-buildx-action@v3
21
22      - name: Build and push
23        uses: docker/build-push-action@v6
24        with:
25          push: true
26          tags: ${ secrets.DOCKERHUB_USERNAME }/bobapp:latest
```

Notes et avis des utilisateurs

Pour rappel :





Points pertinents

Voici mes observations :

- Les fonctionnalités sont buguées
- Les utilisateurs se plaigne que rien ne soit fais ou communiqués.

Globalement, ils ont la sensation que la maintenance du site est abandonnée.

Donc, dans la reprise du développement, il sera important d’aborder toutes les problématiques identifiées dans ce document dans l’ordre suivant :

1. En commençant le développement, communiquez à travers le site est actuellement en développement actif pour conclure le silence radio dont les utilisateurs se plaignent.
2. Dans l’urgence, réparer, si possible, les bugs précis mentionnés : “poster une suggestion de blague”, “poster une vidéo”, “je ne reçois plus rien”.
3. Se familiariser avec le nouvel environnement de développement
4. Retravailler la manière de coder, re-factoriser si possible des parties du code pour s’adapter aux nouveaux critères de qualité de code du repository et mettre en production avec le nouveau processus.

Conclusion

Voici un récapitulatif des demandes du client et des implémentations correspondantes qui ont été mises en place sur la CI / CD.

Valider les tests

Dans la [GitHub-Action](#) décrite dans `build.yml` :

1. Les tests sont lancés aux `push` sur `main`
2. Les tests sont lancés sur chaque nouvelle `pull-request` soumise.

Vérifier la qualité du code

1. La qualité du code est vérifiée par une association [SonarQube](#) directement dans les GitHub-Actions.
2. Des rapports de cette analyse sont disponibles directement dans SonarQube, mais un rapport de cette analyse est aussi posté dans la conversation de la `pull-request` .

Quality Gates

1. Les [Quality Gates](#) personnalisés sont payants, donc pour l’instant, nous ne fonctionnerons qu’avec un service gratuit.
2. Le Quality Gates “Sonar way” utilise par défaut pour les comptes gratuits vérifie :
  - a. S’il y a des nouveaux bugs ajoutés
  - b. Si de nouvelles vulnérabilités sont ajoutées
  - c. Si le nouveau code souffre de dettes techniques
  - d. Si le code a une couverture de code suffisante : 80 %
  - e. S’il y a peu de duplication de code : < 3 %

Builder le projet

Dans la [GitHub-Action](#) :

- Le projet est *buildé* automatiquement à chaque `pull-request` et au push vers `main` comme décrits dans `build.yml` .

Gérer les images Docker

1. Un repository est créé sur [DockerHub](#).
2. Les secrets `DOCKERHUB_USERNAME` ET `DOCKERHUB_TOKEN` sont rajoutés à la configuration du repository GitHub avec mon *username* DockerHub et un token créé juste pour ce projet.
3. La création et le `push` de l’image Docker est faisable directement en développement avec la commande : `make update-docker-image` .
4. La GitHub-Action `deploy.yml` automatise ce processus uniquement sur les `push` vers la branche `main` .

Permettre de lancer les tests

1. Les tests sont lançables via la commande `make back-tests` , en développement.
2. Les tests sont lancés dans la CI / CD, comme décrits dans `build.yml` .

Accéder au rapport de couverture de code des tests

1. Les rapports de couverture sont lançables via la commande `make back-tests-report` .

