

Intégration continue

Florian Crampe, Alexandre Dessolle, Lucas Phelipot,
Junior Turlet, Aurélien Riche

Enseignant référent

Yoann Amsellem

Consigne	4
Sujet	4
IC0	4
IC1	4
IC2	4
Réalisation	6
IC0 : Mise en place de l'équipe et des technologies	6
Répartition des effectifs	6
Liste des US	6
MCD	7
Choix des technologies	7
Maquettes IHM	8
Figure 1 : Page de connexion	8
Figure 2 : Consultation des pièces avec leur coût de production	8
Figure 3 : Consultation des pièces avec leur prix	8
Outil d'intégration continue	8
IC1 : Première batterie de tests	9
Tests	9
Résultats	9
Intégration	10
IC2 : Deuxième batterie de tests	13
Tests individuels	13
Backend	13
Frontend	14
Annexe	15
Utilisation de Jenkins	15
Figure 4 : Dashboard de Jenkins	15

Figure 5 : Configuration d'un build automatique (ici tous les jours à 00h26)
16

Figure 6 : Script d'un pipeline 16

Figure 7 : Pipeline en cours d'exécution 16

Consigne

Sujet

Une société de ventes de pièces détachées automobiles utilise un logiciel pour lister les coûts de production de chacun de ses 1300 articles (3-TIER interne exploité en client lourd, API avec données extractibles depuis le LAN) Elle a besoin d'afficher des prix différenciés pour chacun de ses clients en fonction des contrats cadres. Il y a 25 clients répartis en 5 contrats cadres pouvant évoluer chacun margeant de 5% à 25%, de nouveaux sont potentiellement à venir.

Votre groupe devra développer le logiciel front/back/BDD (très simple et sans design) pour lister les produits et coûts de fabrication. Une API pour authentifier les clients et leur présenter les prix d'achat en fonction de leur contrat cadre. (pour les produits, vous pouvez créer une dizaine d'entrées en dur dans la BDD. Idem pour les 5 contrats cadre) Pour toutes les étapes de l'exercice : Conserver dans un référentiel standard tous les résultats des tests auto avec leur impact si non validés ou validé (docs, exports IHM tests auto et/ou screenshots clairs)

Le langage de développement conseillé est Java.

IC0

Définir un PO et un Scrum Master par groupe (d'autres rôles Agile peuvent être pris par les apprenants si le groupe le souhaite), cadrage organisationnel de l'équipe, définition des User Story+Sprints avec fonctionnalités et affectation du backlog technique aux membres de l'équipe, conception simple du logiciel (modélisation MCD, choix pour la partie Front+API).

Consolider le tout dans un document que vous intégrerez dans un répertoire Doc de votre repository partagé. Il y aura une validation avec Yoann.

IC1

Développer les tests automatisés pour vérifier la cohérence des prix du front par rapport aux valeurs en base de données Creation d'un API simple pour exposer les données en JSON.

IC2

Développer l'API et ses tests automatisés pour valider les données présentées à chaque client + refaire un IC1 pour s'assurer qu'il n'y a pas de regression Creation des 5 règles de marges pour les contrats cadres + affichage prix d'achat par groupe de clients ($\text{prix} = \text{prix de production} + \text{marge} + \text{TVA}$)

Consolider tous les résultats des tests (intermédiaires et finaux) dans un document contenant les screenshots des rapports, à minima pour chaque passage d'un Sprint à un autre. Chaque membre du groupe doit proposer au moins un plan d'action (alimentant le backlog) pour un bug trouvé par les tests (avec résultats des tests KO puis OK après correction).

Réalisation

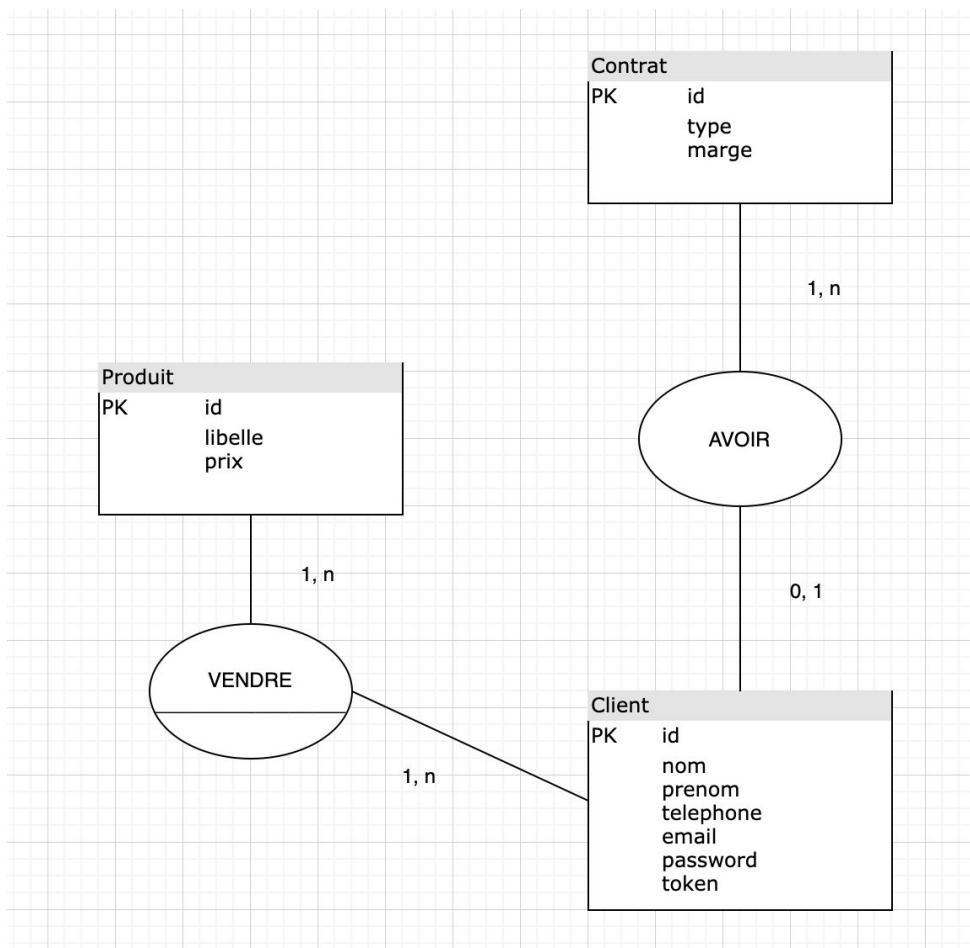
IC0 : Mise en place de l'équipe et des technologies
Répartition des effectifs

Rôle	Effectif
Product Owner (PO)	Florian CRAMPE
Scrum Master	Aurélien RICHE
Équipe opérationnelle	Alexandre DESSOLLE, Junior TURLET et Lucas PHELIPOT

Liste des US



MCD



Choix des technologies

Frontend: JavaScript (React.js)

Backend: Python avec Flask

Base de données: MySQL

Service d'intégration continue (CI): AWS CodeBuild

Maquettes IHM

Se connecter

Adresse mail

Mot de passe

Connexion

Figure 1 : Page de connexion

Société de ventes de pièces détachées automobiles

Liste des pièces avec leur coût de production

Sélectionner un client

Monsieur Aurélien RICHE - Contrat Cadre no. 5

Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€
Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€
Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€	Produit Coût de production: 50€

Figure 2 : Consultation des pièces avec leur coût de production

Société de ventes de pièces détachées automobiles

Liste des pièces avec leur prix (Client no. xxxxxxx)

Déconnexion

Produit Coût du produit: 50€ Acheter	Produit Coût du produit: 50€ Acheter	Produit Coût du produit: 50€ Acheter	Produit Coût du produit: 50€ Acheter
Produit Coût du produit: 50€ Acheter	Produit Coût du produit: 50€ Acheter		

Figure 3 : Consultation des pièces avec leur prix

Outil d'intégration continue

Nous avons décidé d'utiliser AWS CodeBuild car cela nous permet d'avoir le résultat visible depuis Github ce qui est très utile lorsque nous créerons une PR pour pouvoir la *merge* à la branche master lorsque ces tests seront tous passés.

IC1 : Première batterie de tests

Tests

```
18 def getMarge(i):
19     return data["client"][i]["contrat"][0]["marge"]
20
21
22 def getPrice(idProduit, idClient):
23     d = data["produit"][idProduit]["prix"]
24     result = d + (d * 0.2) + (d * (getMarge(idClient) / 100))
25     return result
26
27
28 class UnitTest(unittest.TestCase):
29
30     def test_price(self):
31         self.assertEqual(getPrice(0, 1), 116.87)
32         self.assertEqual(getPrice(1, 1), 30.55)
33         self.assertEqual(getPrice(2, 1), 65)
34
```

La fonction **getMarge()** permet de récupérer le pourcentage de marge du client *i* renvoyé par l'API (json).

La fonction **getPrice()**, quant à elle, permet de définir le prix du produit après ajout de la marge et de la TVA. Tout cela par rapport à un **produit et un client définis**.

Résultats

```
~/Documents/EPPI/cours/integration-continue/societe-pieces-auto/backend(master*) » python3 hello.py
.
-----
Ran 1 test in 0.000s
OK
```

Ici, nous avons utilisé la librairie **Unittest de Python** pour effectuer nos tests. Tous nos tests passent donc la fonction renvoie OK.

Voici le résultat sur notre outil CI: AWS CodeDeploy.

```
23
24 [Container] 2021/01/12 15:11:25 Running command python test.py
25 ..
26 -----
27 Ran 2 tests in 0.000s
28
29 OK
30
```

Intégration

Aurélien Riche s'est occupé de la mise en place de tout le processus d'intégration continue sur AWS en créant un pipeline avec des jobs. AWS ayant simplifié le travail il suffit d'un fichier de configuration:

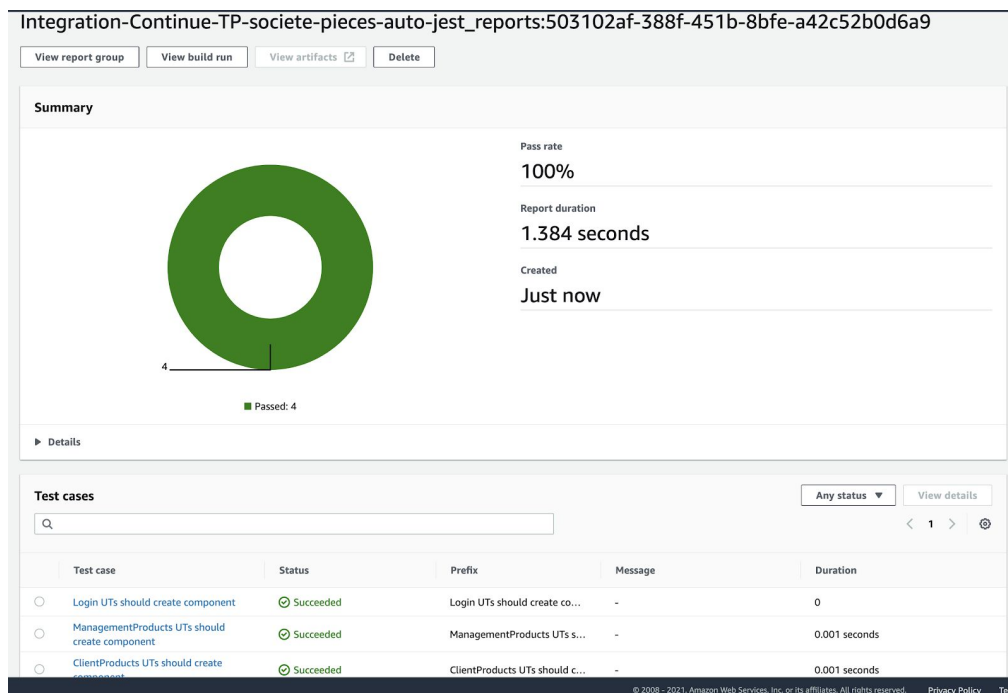
Ce fichier est composé de plusieurs phases: commands et build ainsi que d'un reports.

La phase "commands" permet d'exécuter des commandes à notre convenance, ici cela nous permet de lancer les tests pour le backend pour ensuite lancer les tests pour le frontend. Puis on teste le build de l'application.

22 lines (20 sloc) | 419 Bytes

```
1 version: 0.2
2
3 phases:
4   pre_build:
5     commands:
6       - cd backend
7       - python test.py
8       - cd ../frontend
9       - yarn
10      - npm install --save-dev jest
11      - npm install --save-dev babel-jest regenerator-runtime
12      - yarn test
13   build:
14     commands:
15       - yarn build
16
17   reports:
18     jest_reports:
19       files:
20         - junit.xml
21       file-format: JUNITXML
22       base-directory: frontend/.test
```

La partie "reports" permet de récupérer le rapport généré par Jest sur AWS pour pouvoir avoir accès à cette interface affichant les rapports:



Le pipeline pour gérer le CI (le déploiement continu a été bloqué):

The screenshot shows the AWS CodePipeline console for a pipeline named "Integration-TP-Societe-Pieces-Auto". The pipeline execution ID is "af18cbd9-6dd0-4145-a29a-03d529047e46". The pipeline consists of three stages: "Source", "Build", and "Deploy". The "Source" stage is marked as "Succeeded" and used "GitHub (Version 2)". The "Build" stage is also marked as "Succeeded" and used "AWS CodeBuild". The "Deploy" stage is marked as "Failed". Between the "Source" and "Build" stages, there is a "Disable transition" button. Between the "Build" and "Deploy" stages, there is an "Enable transition" button. The "Deploy" stage is currently blocked, as indicated by the red bar at the bottom of the pipeline view.

Pour le frontend, nous avons ajouté la génération de rapports des tests unitaires pour pouvoir permettre aux équipes de développement d’avoir des statistiques sur le fait que les tests qu’ils ont créés passent ou non. Ceci grâce au paquet “Jest” qui permet de générer des rapports sur AWS.

```
1 import React from 'react';
2 import { screen } from '@testing-library/react';
3
4 describe('Login UTs', () => {
5
6     test('should create component', () => {
7         expect(screen).toBeTruthy();
8     });
9
10 });
```

Voici un exemple de test réalisé côté frontend pour vérifier que le composant s’est bien créé.

Build history							
<div><div>⌂</div><div>Stop build</div><div>View artifacts</div><div>View logs</div><div>Delete builds</div><div>Retry build</div></div>							
<div>< 1 2 3 4 > ⚙</div>							
<input type="checkbox"/>	Build run	Status	Build number	Source version	Submitter	Duration	Completed
<input type="checkbox"/>	Integration-Continue-TP-societe-pieces-auto:5c3e2b6b-5fe2-4731-a7f9-97057411842d	✔ Succeeded	77	arn:aws:s3:::codepipeline-eu-west-3-122107318396/Integration-TP-Socie/SourceArti/YNhfZY9	codepipeline/Integration-TP-Societe-Pieces-Auto	2 minutes 48 seconds	48 minutes ago

Voici le résultat dans la liste AWS :

Developer Tools > CodeBuild > Build projects > Integration-Continue-TP-societe-pieces-auto > Integration-Continue-TP-societe-pieces-auto:ecee3eda-47fe-4f3e-8935-b295c83244dc

Integration-Continue-TP-societe-pieces-auto:ecee3eda-47fe-4f3e-8935-b295c83244dc

Stop buildRetry build

Build status

Status

✔ Succeeded

Initiator

codepipeline/Integration-TP-Societe-Pieces-Auto

Build ARN

arn:aws:codebuild:eu-west-3:883061300465:build/Integration-Continue-TP-societe-pieces-auto:ecee3eda-47fe-4f3e-8935-b295c83244dc

Resolved source version

b456c8e028ed096083ea206270fe976302783658

Start time

Jan 12, 2021 3:52 PM (UTC+1:00)

End time

Jan 12, 2021 3:55 PM (UTC+1:00)

Build number

68

Build logs

Phase details

Reports

Environment variables

Build details

Resource utilization

Report history

↻

View details

View artifacts

🔍

< 1 >

⚙

Report	Type	Status	Line coverage %	Report group	Duration	Created
<div>Integration-Continue-TP-societe-pieces-auto-jest_reports:503102af-388f-451b-8bfe-a42c52b0d6a9</div>	Test	✔ Succeeded	-	Integration-Continue-TP-societe-pieces-auto-jest_reports	1.384 seconds	Just now

Integration-Continue-TP-societe-pieces-auto-jest_reports:503102af-388f-451b-8bfe-a42c52b0d6a9

View report group

View build run

View artifacts

Delete

Summary

4

■ Passed: 4

Pass rate

100%

Report duration

1.384 seconds

Created

Just now

► Details

Test cases

Any status

View details

🔍

< 1 >

⚙

Test case	Status	Prefix	Message	Duration
<div>Login UTs should create component</div>	✔ Succeeded	Login UTs should create co...	-	0
<div>ManagementProducts UTs should create component</div>	✔ Succeeded	ManagementProducts UTs s...	-	0.001 seconds
<div>ClientProducts UTs should create component</div>	✔ Succeeded	ClientProducts UTs should c...	-	0.001 seconds

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Service

On constate en bas de cette page tous les tests unitaires qui sont passés ou non.

12

IC2 : Deuxième batterie de tests

Tests individuels

Backend

Les tests backend ont été réalisés par Alexandre Dessolle et Junior Turllet. Ils concernent le traitement du prix avec la marge TVA.

```
def getMarge(i):  
    return data["Client"][i]["Contrat"][0]["marge"]  
  
def getPrice(idProduit, idClient):  
    d = data["Produit"][idProduit]["prix"]  
    result = d + (d * 0.2) + (d * (getMarge(idClient) / 100))  
    return result  
  
class UnitTest(unittest.TestCase):  
  
    def test_price(self):  
        self.assertEqual(getPrice(0, 1), 116.87)  
  
    def test_price2(self):  
        self.assertEqual(getPrice(2, 1), 65.0)
```

Frontend

Les tests backend ont été réalisés par Florian Crampe et Aurélien Riche. Ils concernent l'application web et l'interaction avec l'API Python.

```
import React from 'react';
import { screen } from '@testing-library/react';

describe('Login UTs', () => {

  test('should create component', () => {
    expect(screen).toBeTruthy();
  });

});
```

```
import React from 'react';
import { screen } from '@testing-library/react';

describe('ManagementProducts UTs', () => {

  test('should create component', () => {
    expect(screen).toBeTruthy();
  });

});
```

```
import React from 'react';
import { screen } from '@testing-library/react';

describe('ClientProducts UTs', () => {

  test('should create component', () => {
    expect(screen).toBeTruthy();
  });

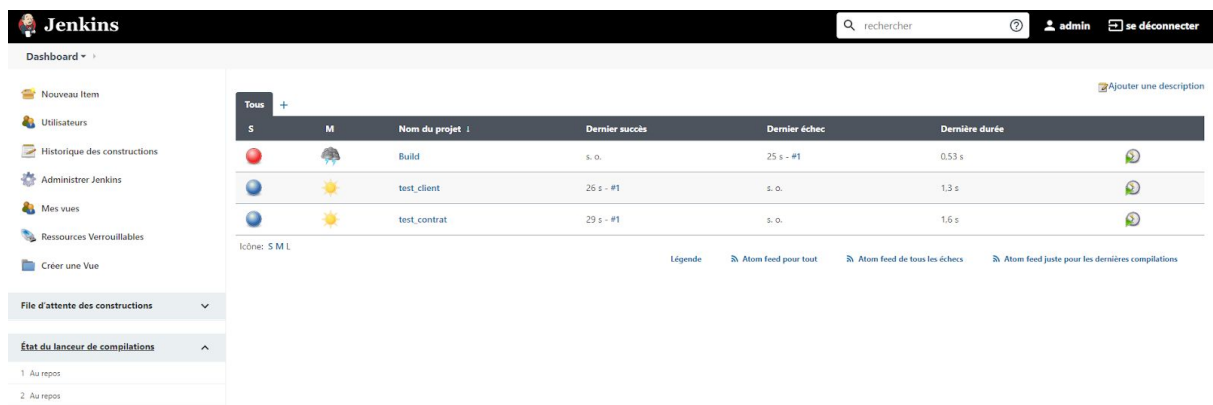
});
```

Annexe

Utilisation de Jenkins

Durant la réalisation de ce projet, nous n'avons pas utilisé Jenkins mais AWS CodeBuild. La seule utilisation de Jenkins de ce module est sa mise en place durant les premiers cours de ce module.

Voici donc deux plusieurs captures d'écran en guise de preuve de l'utilisation de Jenkins au cours du module d'intégration continue.



The screenshot shows the Jenkins Dashboard interface. On the left is a sidebar with navigation links: 'Nouveau Item', 'Utilisateurs', 'Historique des constructions', 'Administrer Jenkins', 'Mes vues', 'Ressources Verrouillables', and 'Créer une Vue'. Below these are sections for 'File d'attente des constructions' and 'État du lanceur de compilations'. The main area displays a table of builds under the 'Tous' filter. The table has columns for status (S, M), project name, last success, last failure, and last duration. Three builds are listed: 'Build' (status S, duration 0.53 s), 'test_client' (status M, duration 1.3 s), and 'test_contrat' (status M, duration 1.6 s). At the bottom, there are links for 'Légende' and 'Atom feed' options.

S	M	Nom du projet	Dernier succès	Dernier échec	Dernière durée
S		Build	s. o.	25 s - #1	0.53 s
	M	test_client	26 s - #1	s. o.	1.3 s
	M	test_contrat	29 s - #1	s. o.	1.6 s

Figure 4 : Dashboard de Jenkins

Build Triggers

☐ Construire après le build sur d'autres projets

☒ Construire périodiquement

Planning

Aurait été lancé à mardi 12 janvier 2021 00 h 26 CET; prochaine exécution à mercredi 13 janvier 2021 00 h 26 CET.

☐ GitHub hook trigger for GITScm polling

☐ Scrutation de l'outil de gestion de version

☐ Désactiver le projet

☐ Période d'attente

☐ Déclencher les builds à distance (Par exemple, à partir de scripts)

Advanced Project Options

Figure 5 : Configuration d'un build automatique (ici tous les jours à 00h26)

Pipeline

Definition

Script


```

1 = node() {
2   stage('Checkout') {
3     checkout(scm)
4     sh 'git clean -xdf'
5   }
6   stage('Build and test') {
7     sh './gradlew build'
8     junit 'build/test-results/test/*.xml'
9   }
10 }
```

☒ Use Groovy Sandbox

Pipeline Syntax

Figure 6 : Script d'un pipeline

Pipeline Build

Build and Run the Programme

Recent Changes

Stage View

Average stage times: 82ms

Stage	Checkout
#3 Jan 12 17:26 No Changes	57ms failed
#2 Jan 12 17:25 No Changes	108ms failed
#1 Jan 12 17:02 No Changes	

Liens permanents

- Dernier build (#3), il y a 3 mn 2 s
- Dernier build en échec (#3), il y a 3 mn 2 s
- Dernier build non réussi (#3), il y a 3 mn 2 s
- Last completed build (#3), il y a 3 mn 2 s

Figure 7 : Pipeline en cours d'exécution

