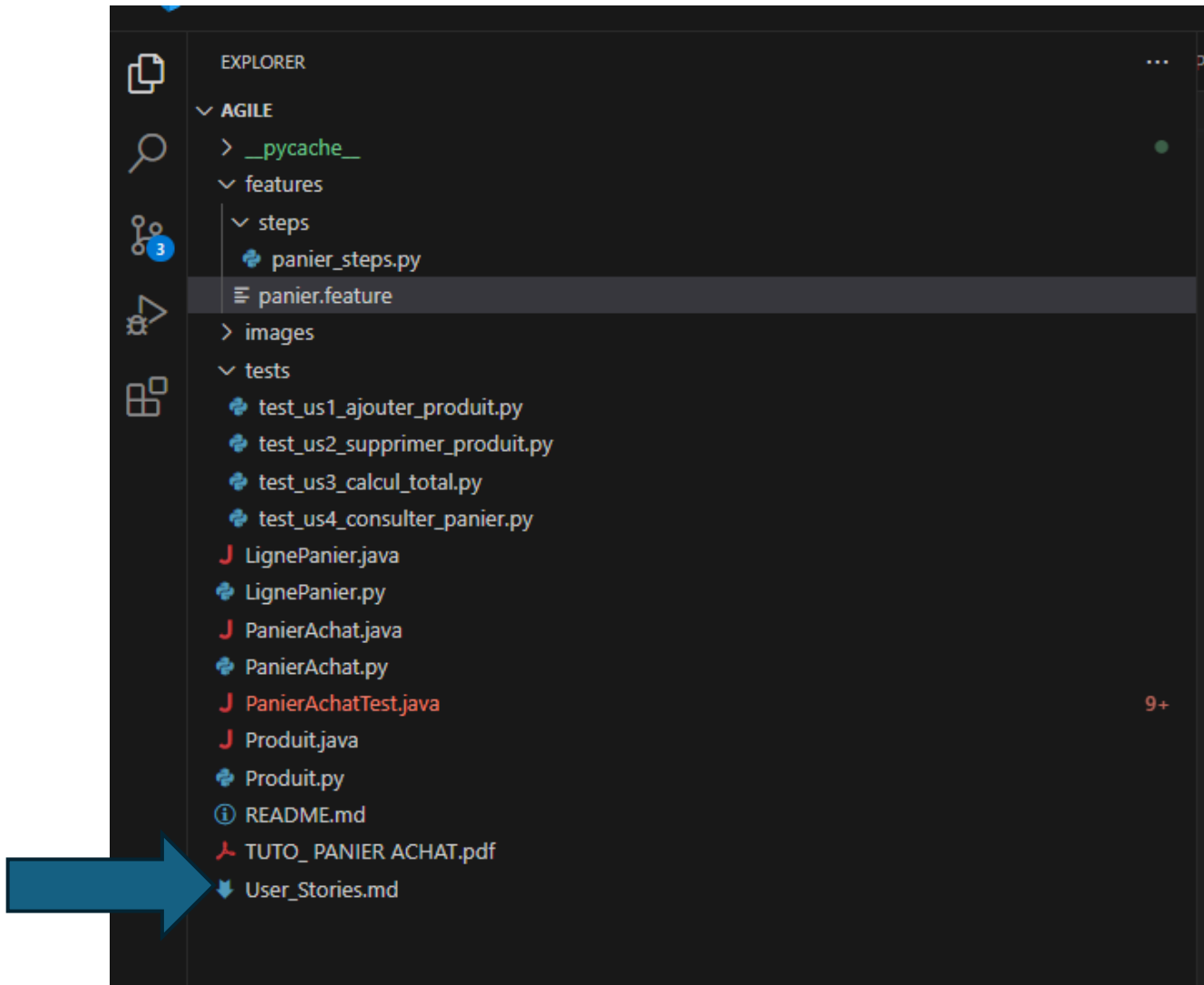


1-User Stories

Les User Stories ont été formalisées dans un fichier Markdown (User_Stories.md).
Ce fichier est disponible dans le dépôt Git du projet.



2-ECRITURES DES CODES + TEST UNITAIRES PLUS FONCTIONNELS

2.1. Traduction du code en Python

Le code initialement développé en Java a été traduit en Python

PanierAchat:

```
PanierAchat.py X
PanierAchat.py
1 from LignePanier import LignePanier
2
3 class PanierAchat:
4
5     # Constructeur
6     def __init__(self):
7         # attributs : état du panier
8         self.lignes = {}
9
10    # Ajout d'un produit
11    def ajouter_produit(self, produit, quantite):
12        if not produit.aktif() or quantite < 1 or quantite > produit.get_stock():
13            print("Impossible d'ajouter le produit :", produit.get_nom())
14            return
15
16        nom_produit = produit.get_nom()
17
18        if nom_produit in self.lignes:
19            ligne = self.lignes[nom_produit]
20            nouvelle_qte = ligne.get_quantite() + quantite
21            if nouvelle_qte > produit.get_stock():
22                print("Stock insuffisant pour", nom_produit)
23                return
24            ligne.set_quantite(nouvelle_qte)
25        else:
26            self.lignes[nom_produit] = LignePanier(produit, quantite)
27
28        print(quantite, nom_produit, "ajoutés au panier.")
29
30    # Suppression d'un produit
31    def supprimer_produit(self, nom_produit):
32        if nom_produit in self.lignes:
33            del self.lignes[nom_produit]
34            print(nom_produit, "supprimé du panier.")
35        else:
36            print("Produit absent du panier.")
37
38    # Calcul du total
39    def calculer_total(self):
40        total = 0.0
41        for ligne in self.lignes.values():
42            total += ligne.get_sous_total()
43        return total
44
45    # Affichage du panier
46    def afficher_panier(self):
47        if not self.lignes:
48            print("Panier vide.")
49            return
50
51        for ligne in self.lignes.values():
52            print(ligne)
53
54        print("Total général :", self.calculer_total())
55
56    # Getter pour tests unitaires
57    def get_lignes(self):
58        return self.lignes
59
60
```

LignePanier:

```
LignePanier.py X
LignePanier.py
1 from Produit import Produit
2
3 class LignePanier:
4
5     # Constructeur
6     def __init__(self, produit, quantite):
7         self.produit = produit
8         self.quantite = quantite
9
10    def get_quantite(self):
11        return self.quantite
12
13    def set_quantite(self, quantite):
14        self.quantite = quantite
15
16    def get_sous_total(self):
17        return self.produit.get_prix() * self.quantite
18
19    def __str__(self):
20        return (
21            self.produit.get_nom()
22            + " | Prix: "
23            + str(self.produit.get_prix())
24            + " | Qté: "
25            + str(self.quantite)
26            + " | Sous-total: "
27            + str(self.get_sous_total())
28        )
29
```

Produit:

```
Produit.py
class Produit:
    # Constructeur
    def __init__(self, nom, prix, stock, actif):
        self.nom = nom
        self.prix = prix
        self.stock = stock
        self.actif = actif

    def get_nom(self):
        return self.nom

    def get_prix(self):
        return self.prix

    def get_stock(self):
        return self.stock

    def est_actif(self):
        return self.actif

    def retirer_stock(self, quantite):
        if quantite <= self.stock:
            self.stock -= quantite
```

2-2-Features(.feature)

Les fonctionnalités du panier ont été décrites sous forme de scénarios BDD dans un fichier .feature. Ces scénarios traduisent les User Stories en langage naturel structuré (Given / When / Then) et servent de référence fonctionnelle.

```
panier.feature
features > panier.feature
1 Feature: Gestion du panier d'achat
2 Le client peut gérer un panier d'achat en respectant les règles métier
3 (stock, produit actif, calcul du total).
4
5 Scenario: US1 - Ajouter un produit au panier
6 Given un panier vide
7 And un produit "Livre" actif avec un prix de 15 et un stock de 5
8 When j'ajoute 2 exemplaires du produit "Livre" au panier
9 Then le panier contient 1 ligne
10 And la quantité du produit "Livre" est 2
11 And le total du panier est 30
12
13 Scenario: US2 - Supprimer un produit du panier
14 Given un panier contenant le produit "Livre" avec une quantité de 2
15 When je supprime le produit "Livre" du panier
16 Then le panier est vide
17
18 Scenario: US3 - Calculer le total du panier
19 Given un panier contenant le produit "Livre" avec une quantité de 2
20 And un produit "Stylo" actif avec un prix de 5 et un stock de 10
21 When j'ajoute 1 exemplaire du produit "Stylo" au panier
22 Then le total du panier est 35
23
24 Scenario: US4 - Consulter un panier vide
25 Given un panier vide
26 When je consulte le panier
27 Then le panier est vide
28
```

2.3- Steps Behave

Les étapes définies dans les fichiers .feature ont été reliées au code Python via des **steps Behave**.

Chaque étape appelle directement les méthodes du code métier afin de valider le comportement du système.

```

panier_steps.py X
features > steps > panier_steps.py
7 @given("un panier vide")
8 def step_panier_vide(context):
9     context.panier = PanierAchat()
10
11
12 @given("un produit '{nom}' actif avec un prix de {prix:d} et un stock de {stock:d}")
13 def step creer_produit(context, nom, prix, stock):
14     produit = Produit(nom, prix, stock, True)
15     context.produits = getattr(context, "produits", [])
16     context.produits[nom] = produit
17
18
19 @given("un panier contenant le produit '{nom}' avec une quantité de {quantite:d}")
20 def step_panier_avec_produit(context, nom, quantite):
21     context.panier = PanierAchat()
22     produit = Produit(nom, 15, 10, True)
23     context.produits = {nom: produit}
24     context.panier.ajouter_produit(produit, quantite)
25
26
27 @when("j'ajoute {quantite:d} exemplaire du produit '{nom}' au panier")
28 @when("j'ajoute {quantite:d} exemplaires du produit '{nom}' au panier")
29 def step_ajouter_produit(context, quantite, nom):
30     produit = context.produits[nom]
31     context.panier.ajouter_produit(produit, quantite)
32
33
34
35 @when("je supprime le produit '{nom}' du panier")
36 def step_supprimer_produit(context, nom):
37     context.panier.supprimer_produit(nom)
38
39
40 @when("je consulte le panier")
41 def step_consulter_panier(context):
42     context.panier.afficher_panier()
43
44
45 @then("le panier contient {nb:d} ligne")
46 def step_verifier_nombre_lignes(context, nb):
47     assert len(context.panier.get_lignes()) == nb
48
49
50 @then("la quantité du produit '{nom}' est {quantite:d}")
51 def step_verifier_quantite(context, nom, quantite):
52     assert context.panier.get_lignes()[nom].get_quantite() == quantite
53
54
55 @then("le panier est vide")
56 def step_panier_est_vide(context):
57     assert len(context.panier.get_lignes()) == 0
58
59
60 @then("le total du panier est {total:d}")
61 def step_verifier_total(context, total):
62     assert context.panier.calculer_total() == total

```

2.4- Test

Ces tests permettent de garantir la cohérence fonctionnelle et la fiabilité du panier d'achat.

```

test_us1_ajouter_produit.py X
tests > test_us1_ajouter_produit.py
1 import unittest
2
3 from PanierAchat import PanierAchat
4 from Produit import Produit
5
6
7 class TestUS1AjouterProduit(unittest.TestCase):
8
9     def test_ajout_produit_valide(self):
10         panier = PanierAchat()
11         produit = Produit("Livre", 15, 5, True)
12
13         panier.ajouter_produit(produit, 2)
14
15         self.assertEqual(1, len(panier.get_lignes()))
16         self.assertEqual(2, panier.get_lignes()["Livre"].get_quantite())
17         self.assertEqual(30, panier.calculer_total())
18
19
20 if __name__ == "__main__":
21     unittest.main()
22

```

```

test_us2_supprimer_produit.py X
tests > test_us2_supprimer_produit.py
1 import unittest
2
3 from PanierAchat import PanierAchat
4 from Produit import Produit
5
6
7 class TestUS2SupprimerProduit(unittest.TestCase):
8
9     def test_suppression_produit(self):
10         panier = PanierAchat()
11         produit = Produit("Livre", 15, 5, True)
12
13         panier.ajouter_produit(produit, 2)
14         panier.supprimer_produit("Livre")
15
16         self.assertEqual(0, len(panier.get_lignes()))
17
18
19 if __name__ == "__main__":
20     unittest.main()
21

```

```
test_us3_calcul_total.py X
tests > test_us3_calcul_total.py
1 import unittest
2
3 from PanierAchat import PanierAchat
4 from Produit import Produit
5
6
7 class TestUS3CalculTotal(unittest.TestCase):
8
9     def test_calcul_total_panier(self):
10         panier = PanierAchat()
11
12         livre = Produit("Livre", 15, 5, True)
13         stylo = Produit("Stylo", 5, 10, True)
14
15         panier.ajouter_produit(livre, 2)
16         panier.ajouter_produit(stylo, 1)
17
18         self.assertEqual(35, panier.calculer_total())
19
20
21 if __name__ == "__main__":
22     unittest.main()
23
test_us4_consulter_panier.py X
tests > test_us4_consulter_panier.py
1 import unittest
2
3 from PanierAchat import PanierAchat
4
5
6 class TestUS4ConsulterPanier(unittest.TestCase):
7
8     def test_panier_vide(self):
9         panier = PanierAchat()
10
11         self.assertEqual(0, len(panier.get_lignes()))
12         self.assertEqual(0, panier.calculer_total())
13
14
15 if __name__ == "__main__":
16     unittest.main()
17
```

3. RUN

Exécution des tests fonctionnels avec Behave.

Les scénarios US1, US2 et US4 sont validés avec succès.

Le scénario US3 échoue initialement en raison d'une étape non définie dans les steps Behave (1 exemplaire au singulier).

Le problème a été résolu en généralisant la définition de l'étape afin de gérer à la fois le singulier et le pluriel.

```
PS C:\Users\Adrien\Documents\GitHub\AGILE> behave
USING RUNNER: behave.runner.Runner
Feature: Gestion du panier d'achat # features/panier.feature:1
  (Le client peut gérer un panier d'achat en respectant les règles métier
  (stock, produit actif, calcul du total))
Scenario: US1 - Ajouter un produit au panier # features/panier.feature:5
  Given un panier vide # features/steps/panier_steps.py:7 0.000s
  And un produit "Livre" actif avec un prix de 15 et un stock de 5 # features/steps/panier_steps.py:12 0.001s
  When j'ajoute 2 exemplaires du produit "Livre" au panier # features/steps/panier_steps.py:27 0.000s
  Then le panier contient 1 ligne # features/steps/panier_steps.py:43 0.000s
  And la quantité du produit "Livre" est 2 # features/steps/panier_steps.py:48 0.000s
  And le total du panier est 30 # features/steps/panier_steps.py:58 0.000s
Scenario: US2 - Supprimer un produit du panier # features/panier.feature:13
  Given un panier contenant le produit "Livre" avec une quantité de 2 # features/steps/panier_steps.py:19 0.001s
  When je supprime le produit "Livre" du panier # features/steps/panier_steps.py:33 0.000s
  Then le panier est vide # features/steps/panier_steps.py:53 0.000s
Scenario: US3 - Calculer le total du panier # features/panier.feature:18
  Given un panier contenant le produit "Livre" avec une quantité de 2 # features/steps/panier_steps.py:19 0.001s
  And un produit "Stylo" actif avec un prix de 5 et un stock de 10 # features/steps/panier_steps.py:12 0.001s
  When j'ajoute 1 exemplaire du produit "Stylo" au panier # features/steps/panier_steps.py:27 0.000s
  Then le total du panier est 35 # features/steps/panier_steps.py:58 0.000s
  When j'ajoute 1 exemplaire du produit "Stylo" au panier # None
  Then le total du panier est 35 # None
CAPTURED STDOUT: scenario
2 Livre ajoutés au panier
----- CAPTURED_SCENARIO_OUTPUT_END -----
Scenario: US4 - Consulter un panier vide # features/panier.feature:24
  Given un panier vide # features/s # Given un panier vide # features/steps/panier_steps.py:7 0.000s
  When je consulte le panier # features/s # When je consulte le panier # features/steps/panier_steps.py:38 0.000s
  Then le panier est vide # features/s # Then le panier est vide # features/steps/panier_steps.py:53 0.001s
Error: scenarios:
  features/panier.feature:18 US3 - Calculer le total du panier
0 features passed, 0 failed, 1 error, 0 skipped
3 scenarios passed, 0 failed, 1 error, 0 skipped
10 steps passed, 0 failed, 1 skipped, 1 undefined
Took 0m10.000s
You can implement step definitions for undefined steps with these snippets:
from behave import pending, step
import StepNotImplementedError
@when(u'j'ajoute 1 exemplaire du produit "Stylo" au panier')
def step_impl(context):
    raise StepNotImplementedError(u'When j'ajoute 1 exemplaire du produit "Stylo" au panier')
```

```

PS C:\Users\adrien\Documents\GitHub\AGILE> behave
USING RUNNER: behave.runner:Runner
Feature: Gestion du panier d'achat # features/panier.feature:1
  Le client peut gérer un panier d'achat en respectant les règles métier
  (stock, produit actif, calcul du total).
    And le total du panier est 30 # features/steps/panier_steps.py:60 0.000s
    And le total du panier est 30 # features/steps/panier_steps.py:60
    Then le panier est vide # features/steps/panier_steps.py:55 0.000s
    Then le panier est vide # features/steps/panier_steps.py:55
    Then le total du panier est 35 # features/steps/panier_steps.py:60 0.000s
    Then le total du panier est 35 # features/steps/panier_steps.py:60
Scenario: US4 - Consulter un panier vide # features/panier.feature:24
  Given un panier vide # features/steps/panier_steps.py:7 0.001s
  When je consulte le panier # features/steps/panier_steps.py:40 0.000s
  Then le panier est vide # features/steps/panier_steps.py:55 0.000s

1 feature passed, 0 failed, 0 skipped
4 scenarios passed, 0 failed, 0 skipped
16 steps passed, 0 failed, 0 skipped
Took 0min 0.006s

```

Les quatre tests unitaires correspondant aux User Stories US1 à US4 ont été exécutés avec succès.

Tous les tests sont validés, confirmant le bon fonctionnement des méthodes et des règles métier du panier d'achat.

```

Took 0min 0.006s
PS C:\Users\adrien\Documents\GitHub\AGILE> python -m unittest discover -s tests -p "test_*.py"
2 Livre ajoutés au panier.
.2 Livre ajoutés au panier.
Livre supprimé du panier.
.2 Livre ajoutés au panier.
1 Stylo ajoutés au panier.
..
-----
Ran 4 tests in 0.001s
OK

```