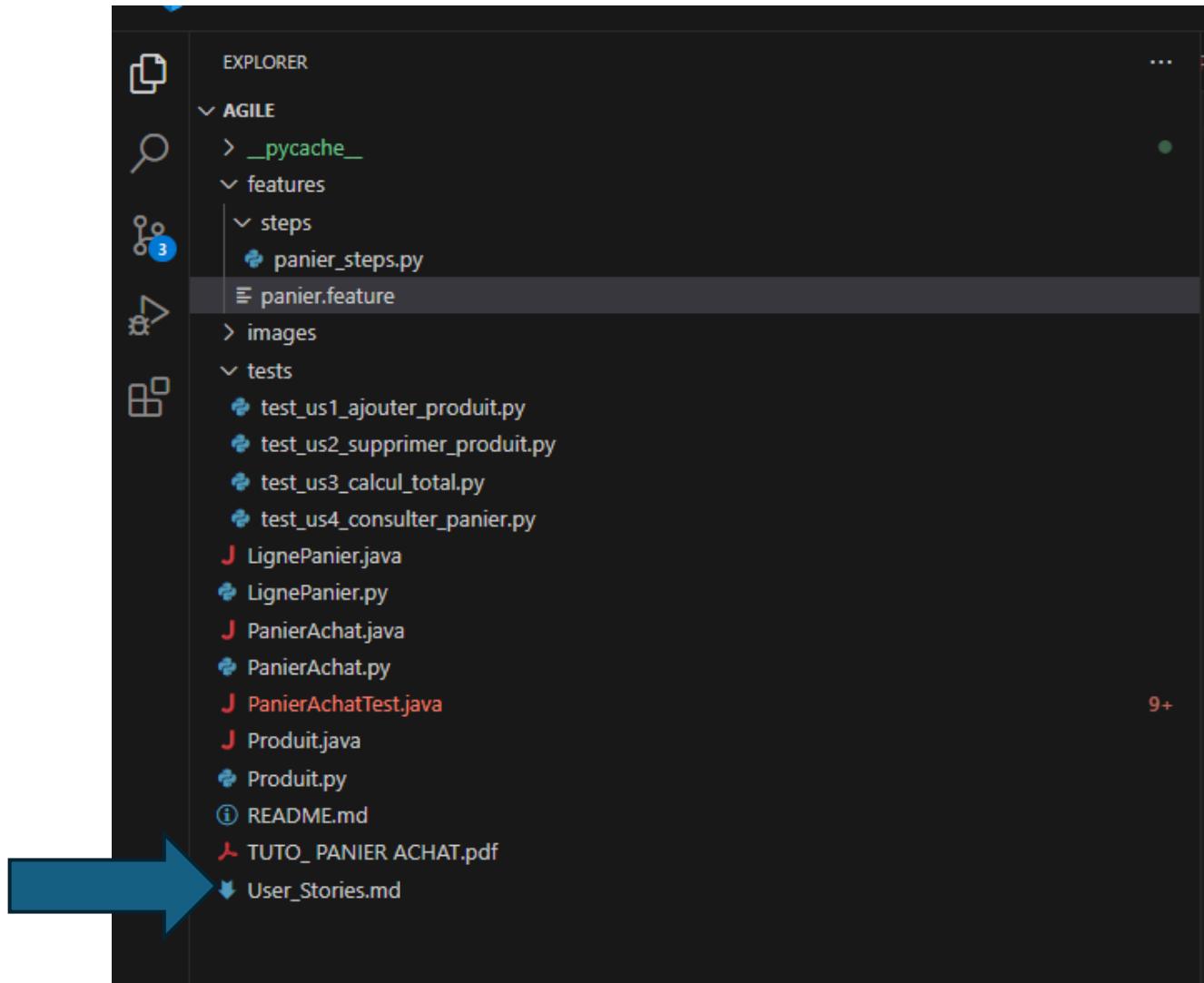


1-User Stories

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

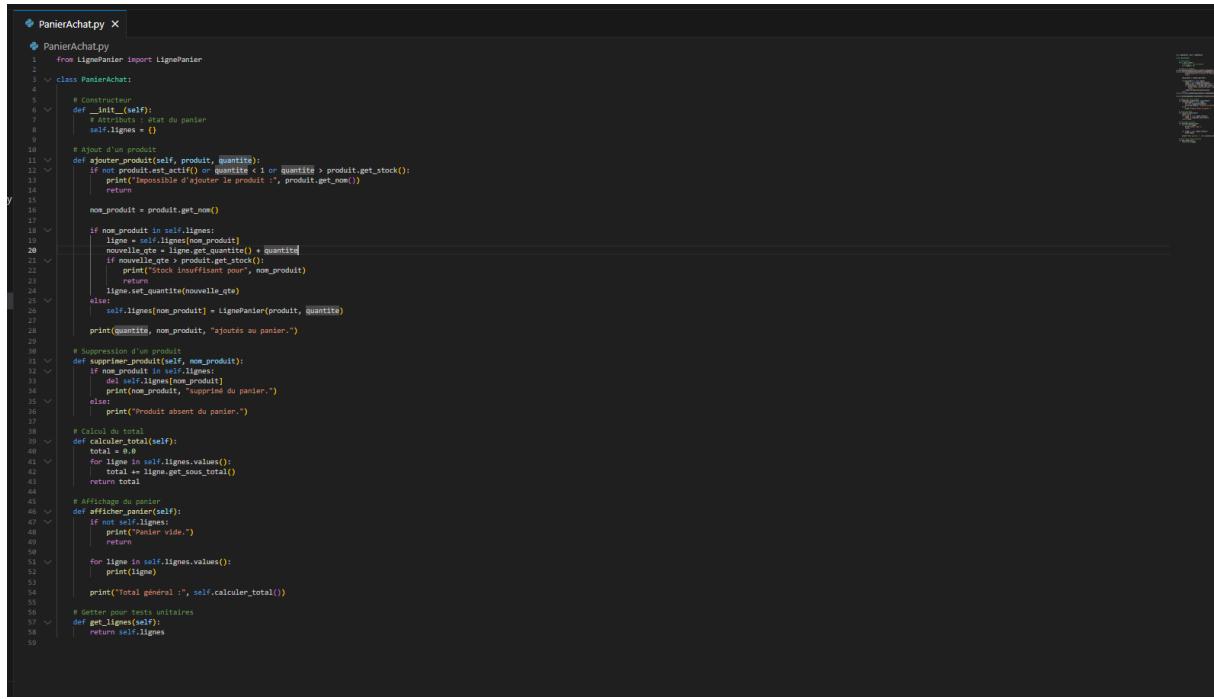


2-ÉCRITURES DES CODES + TEST UNITAIRES PLUS FONCTIONNELS

2.1. Traduction du code en Python

Le code initialement développé en Java a été traduit en Python

PanierAchat:



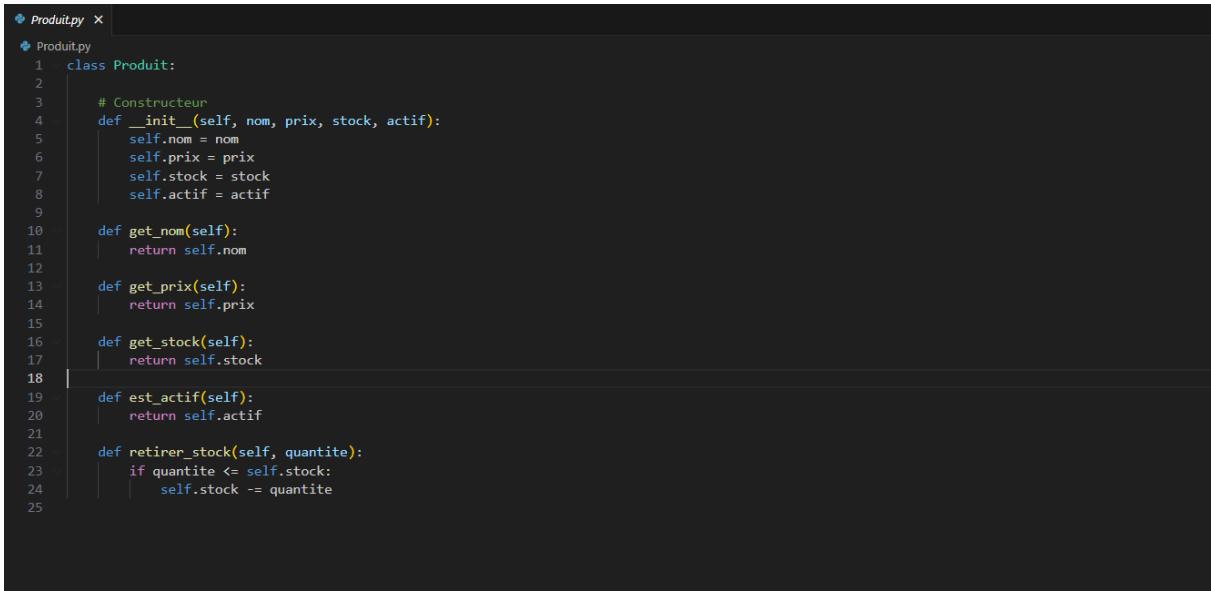
```
❶ PanierAchat.py X
❷ 
❸ #!/usr/bin/python
❹ # coding: utf-8
❺
❻ from LignePanier import LignePanier
❼
❼ class PanierAchat:
❼
❼     # Constructeur
❼     def __init__(self):
❼         # Attributs : état du panier
❼         self.lignes = []
❼
❼     # Ajout d'un produit
❼     def ajouter_produit(self, produit, quantite):
❼         if not produit.est_actif() or quantite < 1 or quantite > produit.get_stock():
❼             print("Impossible d'ajouter le produit", produit.get_nom())
❼             return
❼
❼         nom_produit = produit.get_nom()
❼
❼         if nom_produit in self.lignes:
❼             ligne = self.lignes[nom_produit]
❼             nouvelle_qte = ligne.get_quantite() + quantite
❼             if nouvelle_qte > produit.get_stock():
❼                 print("Stock insuffisant pour", nom_produit)
❼                 return
❼             ligne.set_quantite(nouvelle_qte)
❼         else:
❼             self.lignes[nom_produit] = LignePanier(produit, quantite)
❼
❼         print(quantite, nom_produit, "ajoutés au panier.")
❼
❼     # Suppression d'un produit
❼     def supprimer_produit(self, nom_produit):
❼         if nom_produit in self.lignes:
❼             del self.lignes[nom_produit]
❼             print(nom_produit, "supprimé du panier.")
❼         else:
❼             print("Produit absent du panier.")
❼
❼     # Calcul du total
❼     def calculer_total(self):
❼         total = 0
❼         for ligne in self.lignes.values():
❼             total += ligne.get_sous_total()
❼
❼         return total
❼
❼     # Affichage du panier
❼     def afficher_panier(self):
❼         if not self.est_vide():
❼             print("Panier vide")
❼
❼         for ligne in self.lignes.values():
❼             print(ligne)
❼
❼         print("Total général :", self.calculer_total())
❼
❼     # Getter pour tests unitaires
❼     def get_lignes(self):
❼         return self.lignes
```

LignePanier:



```
❶ LignePanier.py X
❷ 
❸ #!/usr/bin/python
❹ # coding: utf-8
❺
❻ from Produit import Produit
❼
❼ class LignePanier:
❼
❼     # Constructeur
❼     def __init__(self, produit, quantite):
❼         self.produit = produit
❼         self.quantite = quantite
❼
❼     def get_quantite(self):
❼         return self.quantite
❼
❼     def set_quantite(self, quantite):
❼         self.quantite = quantite
❼
❼     def get_sous_total(self):
❼         return self.produit.get_prix() * self.quantite
❼
❼     def __str__(self):
❼         return (
❼             self.produit.get_nom()
❼             + " | Prix: "
❼             + str(self.produit.get_prix())
❼             + " | Qté: "
❼             + str(self.quantite)
❼             + " | Sous-total: "
❼             + str(self.get_sous_total())
❼         )
```

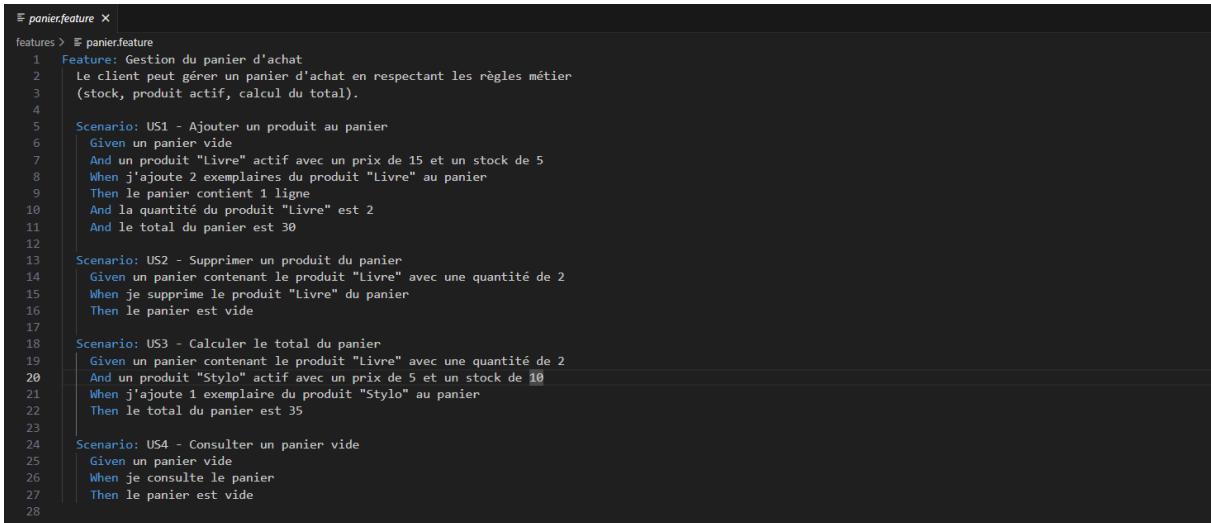
Produit:



```
Produit.py
1  class Produit:
2
3      # Constructeur
4      def __init__(self, nom, prix, stock, actif):
5          self.nom = nom
6          self.prix = prix
7          self.stock = stock
8          self.actif = actif
9
10     def get_nom(self):
11         return self.nom
12
13     def get_prix(self):
14         return self.prix
15
16     def get_stock(self):
17         return self.stock
18
19     def est_actif(self):
20         return self.actif
21
22     def retirer_stock(self, quantite):
23         if quantite <= self.stock:
24             self.stock -= quantite
25
```

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
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.

```

features > steps > panier_steps.py
1  # Given('un panier vide'):
2  def step_panier_vide(context):
3      context.panier = PanierAchat()
4
5  # Given('un produit "{nom}" actif avec un prix de {prix:d} et un stock de {stock:d}')
6  def step_creer_produit(context, nom, prix, stock):
7      produit = Produit(nom, prix, stock, True)
8      context.produits[nom] = produit
9      context.produits[nom].stock = stock
10
11  # Given('un panier contenant le produit "{nom}" avec une quantité de {quantite:d}')
12  def step_panier_avec_produit(context, nom, quantite):
13      context.panier = PanierAchat()
14      produit = Produit(nom, 15, 5, True)
15      context.produits[nom] = produit
16      context.panier.ajouter_produit(produit, quantite)
17
18  # When('J\'ajoute {quantites:d} exemplaire du produit "{nom}" au panier')
19  # When('J\'ajoute {quantite:d} exemplaires du produit "{nom}" au panier')
20  def step_ajouter_produit(context, quantite, nom):
21      produit = context.produits[nom]
22      context.panier.ajouter_produit(produit, quantite)
23
24  # Then('je consulte le panier')
25  def step_consulter_panier(context):
26      context.panier.afficher_panier()
27
28  # Then('le panier contient {nb:d} ligne')
29  def step_verifier_nombre_lignes(context, nb):
30      assert len(context.panier.get_lignes()) == nb
31
32  # Then('la quantité du produit "{nom}" est {quantite:d}')
33  def step_verifier_quantite(context, nom, quantite):
34      assert context.panier.get_lignes()[nom].get_quantite() == quantite
35
36  # Then('le panier est vide')
37  def step_panier_est_vide(context):
38      assert len(context.panier.get_lignes()) == 0
39
40  # Then('le total du panier est {total:d}')
41  def step_verifier_total(context, total):
42      assert context.panier.calculer_total() == total

```

2.4- Test

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

```

tests > test_us1_ajouter_produit.py X
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

```

```

tests > test_us2_supprimer_produit.py X
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

```

```

tests > ⚡ test_us3.calcul_total.py X
tests > ⚡ test_us4.consulter_panier.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

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 ENVIRONEMENT: C:\Users\Adrien\Documents\GitHub\AGILE\env\Scripts\activate
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 et calcul du total
  Scenario: US1 - Ajouter un produit au panier
    Given un panier vide                                # features/panier.feature:5
    And un produit "Livre" avec un prix de 15 et un stock de 5          # features/steps/panier_steps.py:7 0.000s
    When j'ajoute 2 exemplaires du produit "Livre" au panier          # features/steps/panier_steps.py:19 0.001s
    Then le panier contient 1 ligne                           # features/steps/panier_steps.py:47 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:50 0.000s
  Scenario: US2 - Supprimer un produit du panier
    Given un panier contenant le produit "Livre" avec une quantité de 2          # features/panier.feature:13
    When je supprime le produit "Livre" du panier          # features/steps/panier_steps.py:33 0.000s
    Then panier est vide                                     # features/steps/panier_steps.py:53 0.000s
  Scenario: US3 - Calculer le total du panier
    Given un panier contenant le produit "Livre" avec une quantité de 2          # features/panier.feature:18
    And un produit "Stylo" activé avec un prix de 5 et un stock de 10           # features/steps/panier_steps.py:19 0.001s
    When j'ajoute 1 exemplaire du produit "Stylo" au panier          # features/steps/panier_steps.py:12 0.001s
    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
    Given un panier vide                                # features/panier.feature:20
    When je consulte le panier                         # features/s/ Given un panier vide
    Then le panier est vide                            # features/s/ When je consulte le panier
                                                    # features/s/ Then le panier est vide
  ----- ERRORS -----
  Error scenarios:
  features/panier.feature:18 US3 - Calculer le total du panier
  0 features passed, 0 failed, 1 error, 0 skipped
  1 feature passed, 0 failed, 1 error, 0 skipped
  14 steps passed, 0 failed, 1 skipped, 1 undefined
  Took 0min 0.006s
  You can implement step definitions for undefined steps with these snippets:
  from behave.api.PendingStep import StepNotImplementedError
  @when(u'j\'ajoute 1 exemplaire du produit "Stylo" au panier')
  def step_implementation(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
        And le total du panier est 30
        Then le panier est vide
        Then le panier est vide
        Then le total du panier est 35
        Then le total du panier est 35
      Scenario: US4 - Consulter un panier vide # features/panier.feature:24
        Given un panier vide
        When je consulte le panier
        Then le panier est vide
    # features/steps/panier_steps.py:60 0.000s
    # features/steps/panier_steps.py:60
    # features/steps/panier_steps.py:55 0.000s
    # features/steps/panier_steps.py:55
    # features/steps/panier_steps.py:60 0.000s
    # features/steps/panier_steps.py:60
    # features/steps/panier_steps.py:55 0.000s
    # features/steps/panier_steps.py:55
    # features/steps/panier_steps.py:60
  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.

```
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
```