Low Level Design (LLD) – Behave (Behavior Driven Development) Lab

1. Introduction – This Low-Level Design (LLD) provides information detailing the specific tests that will be used within the Behave Lab.
2. Scenarios Used:
   1. Scenario 1 – Adding an object to the cart:
      1. Given the fruit stand application is available.
      2. When the user adds an object to the cart
      3. Then the object should be added to the cart.
   2. Scenario 2 – Removing an object from the store cart:
      1. Given the user has items in the cart.
      2. When the user removes an object from the cart
      3. Then the object will be removed from the cart
   3. Scenario 3 – Applying a discount code to cart total:
      1. Given the fruit stand application is open.
      2. And the user has items in the cart.
      3. When the user applies a discount code ‘DISCOUNT20’
      4. Then the cart total should be updated with a 20% discount
   4. Scenario 4 – Removing an item not in cart:
      1. Given the fruit stand application is open.
      2. And the user has items in the cart.
      3. When the user attempts to remove an incorrect item from the cart
      4. Then the cart contents should remain unchanged.
   5. Scenario 5 – Cleared cart after purchase:
      1. Given the fruit stand application is open.
      2. And the user has items in the cart.
      3. When the user clicks the ‘Purchase’ button
      4. Then the cart should be empty
3. Scenario Steps:
   1. Scenario 1 and 2
      1. @given(‘the fruit stand application is open’)  
         def step\_open\_fruit\_store(context):  
          context.store = FruitStand()  
          context.store.open()  
           
         @when(‘the user adds a fruit to their cart’)  
         def step\_add\_fruit\_to\_cart(context):  
          context.store.add\_to\_cart(context.fruit)  
           
         @then(‘The fruit should be added to the cart’)  
         def step\_check\_fruit\_added\_to\_cart(context):  
          assert context.fruit in context.store.cart  
           
         @given(‘the user has fruits in the cart’)  
         def step\_user\_has\_fruit\_in\_cart(context):  
          context.fruit = “fruit\_object”  
          context.store.add\_to\_cart(context.fruit)  
           
         @when(‘the user removes a fruit from the cart’)  
         def step\_remove\_fruit\_from\_cart(context):  
          context.store.remove\_from\_cart(context. fruit)  
           
           
         @then(‘the fruit object should be removed from the cart’)  
         def step\_check\_fruit\_removed\_from\_cart(context):  
          assert context.fruit not in context.store.cart

* 1. Scenario 3
     1. @given(‘the user has fruit items in the cart’)  
        def step\_user\_has\_items\_in\_cart(context):  
         context.fruit = “fruit\_object”  
         context.store = FruitStand()  
         context.store.open()  
         context.store.add\_to\_cart(context.fruit)  
          
        @when(‘the user attempts to apply a discount code {DISCOUNT20}’)  
        def step\_apply\_discount\_code(context, discount\_code):  
         context.discount\_code = discount\_code  
         context.store.apply\_discount\_code(context.discount\_code)  
          
        @then(‘the cart total should be updated with a 20% discount’)  
        def step\_check\_discount\_applies(context):  
         original\_total = context.store.calculate\_cart\_total(context.fruit)  
         discounted\_total = context.store.calculate\_discounted\_total(original\_total, 0.2)  
         assert context.store.cart\_total == discounted\_total
  2. Scenario 4
     1. @when(‘the user attempts to remove fruit from the cart’)  
        def step\_attempt\_remove\_incorrect\_item(context, fruit):  
         context.item\_to\_remove = fruit  
         context.initial\_cart = context.store.cart.copy()  
          
        @then(‘the cart contents should remain unchanged’)  
        def step\_check\_cart\_unchanged(context):  
         assert context.item\_to\_remove not in context.store.cart  
         assert context.store.cart == context.initial\_cart
  3. Scenario 5
     1. @when(‘the user clicks the “Purchase” button’)  
        def step\_click\_purchase\_button(context):  
         context.store.process\_purchase()  
          
        @then(‘the cart should be emptied’)  
        def step\_check\_cart\_empty(context):  
         assert not context.store.cart