}

}

Now run the tests again:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feat | ure: | Sh | opp | ing | cart |  |
| In | ord | er | to | buy | produ | cts |
| As | a c | ust | ome | r |  |  |
| Feat | ure: | Sh | opp | ing | cart |  |
| In | ord | er | to | buy | produ | cts |
| As | a c | ust | ome | r |  |  |
| I | need | to | be | able to | | put |

interesting

products into a cart

Scenario: Change product count in the cart Given there is a cart with 5 pieces of 7 When I set 3 pieces for 7 product Then I should have 3 pieces of 7 product Failed asserting that an array has the 7 => 3

)■

Scenario: Remove products from the cart Given there is a cart with 5 pieces of When I add 14 pieces of 7 product And I clear cart Then I should have empty cart

— Failed scenarios:

features/cart.feature:31

6 scenarios (5 passed, 1 failed)

31 steps (30 passed, 1 failed)

0m0■22s (13.85Mb)

In this case, we have seen one failure and a failure report.

# features/cart.feature:31 prod # thereIsAWhichCostsPs()

# iSetPiecesForProduct()

# iShouldHavePiecesOf() subset Array &0 (

# features/cart.feature:36 7 prod # thereIsAWhichCostsPs()

# iAddTheToTheCart()

# iClearCart()

# iShouldHaveEmptyCart()

How it works...

Behat is a BDD testing framework. It facilitates writing preceding human-readable testing scenarios to low-level technical implementation.

When we write scenarios for every feature, we can use a set of operators:

Scenario: Adding products to t Given there is a clean car When I add 3 pieces of 5 p Then I should have 3 piece And I should have 1 produc And the overall cart amoun

he cart t

roduct

s of 5 product t

t should be 3

Behat parses our sentences and finds the associated implementation of the sentence in the context class:

class FeatureContext implements SnippetAcceptingContext {