

## Component Implementation & Unit Testing: Component Cart Management

### Members

Kittiwat	Chatkositpat	55070501060
Kanyawee	Pornsawangdee	56070501070
Chanakarn	Laoveerawat	56070501078
Wiroon	Hirunyawiroon	56070501088

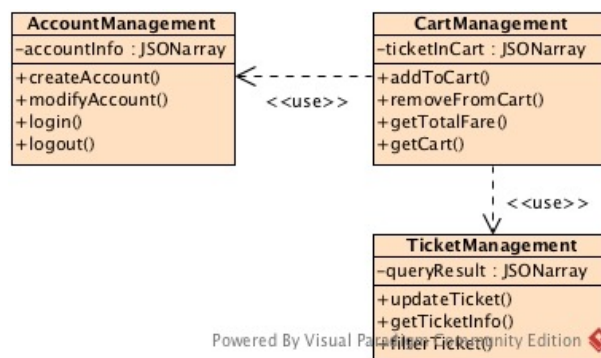
## 1. Design Specification and Functional Specification

### 1.1 Processing Narrative

Component Cart Management ทำหน้าที่ให้บริการลูกค้าในขั้นตอนการเลือกและจองตั๋ว กระบวนการข้างต้นรวมถึงการจัดการตะกร้าสินค้าของลูกค้า การคำนวณราคา และการดำเนินการจองตั๋ว และบันทึกลงในฐานข้อมูล

### 1.2 Processing Detail

#### 1.2.1 Class Hierarchy



รูปที่ 1.2.1 Class Hierarchy ของ Component Cart Management

#### 1.2.2 Restriction / Limitation

- Component Cart Management จะถูกเรียกใช้งานทันทีที่มีการเข้าสู่ระบบของลูกค้า
- ผู้ใช้งานต้องทำการเข้าสู่ระบบผ่าน Component Account Management ในฐานะลูกค้าก่อนจึงจะสามารถใช้งาน component นี้ได้

#### 1.2.3 Performance Issue

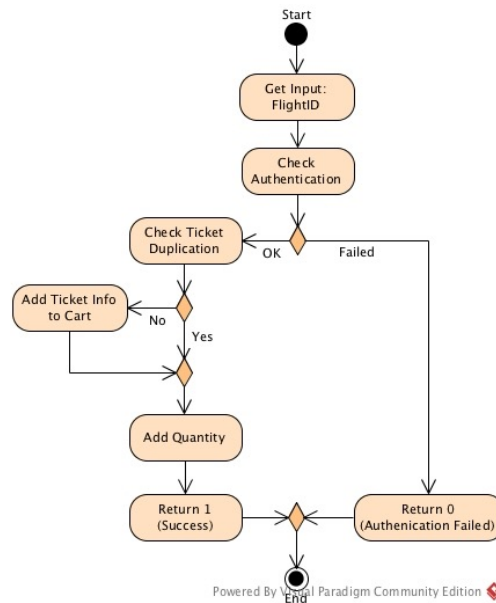
- การตอบสนองของ operation ใน component นี้บนหน้าเว็บควรทำได้ในเวลา 3 วินาที

#### 1.2.4 Design Constraints

- Component ต้องมีการเก็บค่าสินค้าในตะกร้าไว้ตลอดการใช้งานในทุกหน้าเว็บ ข้อมูลการเลือกซื้อสินค้าจึงจำเป็นต้องมีการเก็บไว้ชั่วคราวในตัวแปร session

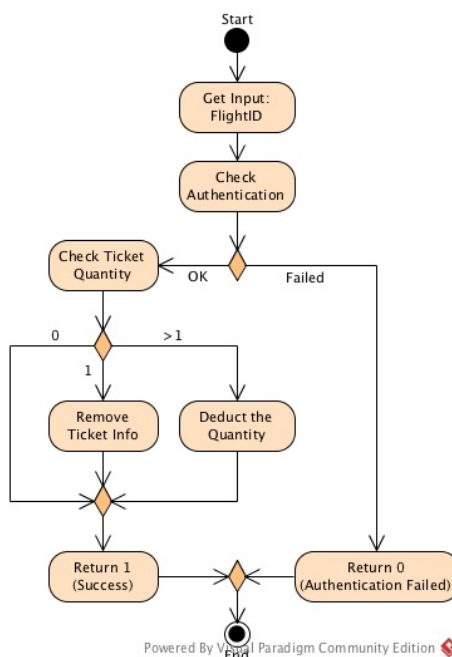
### 1.2.5 Operations

(a) addToCart() เป็นฟังก์ชันสำหรับการเพิ่มตั๋วลงในตะกร้าสินค้า ฟังก์ชันจะทำการรับข้อมูลตัวจากการกดเลือกเพิ่มลงในสินค้าลงตะกร้าสินค้าของลูกค้า แล้วเพิ่มข้อมูลดังกล่าวลงในตัวแปร session ฟังก์ชันนี้คืนค่า 1 เพื่อแสดงว่าฟังก์ชันสามารถทำงานสำเร็จ และคืนค่า 0 เพื่อแสดงว่าทำงานไม่สำเร็จ



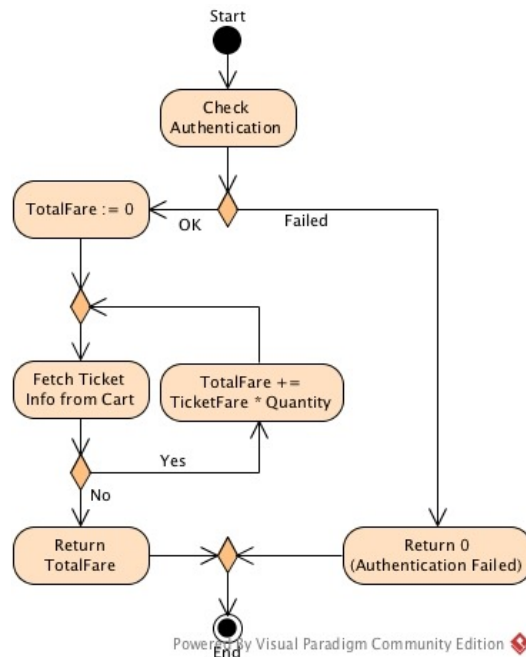
รูปที่ 1.2.5(a) Algorithmic Model ของ operation addToCart()

(b) removeFromCart() เป็นฟังก์ชันสำหรับการนำตั๋วออกจากตะกร้าสินค้า ฟังก์ชันจะทำการรับข้อมูลตัวจากการกดเลือกนำออกจากตะกร้าสินค้าของลูกค้า หรือเมื่อการจ่ายเงินสมบูรณ์ แล้วลบข้อมูลดังกล่าวออกจากตัวแปร session ฟังก์ชันนี้คืนค่า 1 เพื่อแสดงว่าฟังก์ชันสามารถทำงานสำเร็จ และคืนค่า 0 เพื่อแสดงว่าทำงานไม่สำเร็จ



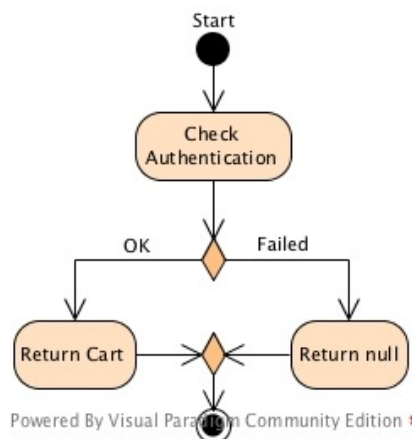
รูปที่ 1.2.5(b) Algorithmic Model ของ operation removeFromCart()

(c) getTotalFare() เป็นฟังก์ชันสำหรับการคำนวณราคารวมของตั๋วในตะกร้าของลูกค้า ฟังก์ชันจะถูกใช้งานอัตโนมัติเมื่อมีการเปลี่ยนแปลงของรายการสินค้าในตะกร้า ฟังก์ชันไม่มีการรับค่าใดๆ แต่จะมีการเรียกใช้ค่าจากตัวแปร session เพื่อใช้ในการคำนวณหาผลรวมและเก็บผลลัพธ์ ฟังก์ชันจะคืนค่าผลลัพธ์เป็นราคารวมของตั๋วในตะกร้า และคืนค่า 0 เพื่อแสดงว่าทำงานไม่สำเร็จ



รูปที่ 1.2.5(c) Algorithmic Model ของ operation getTotalFare()

(d) getCart() เป็นฟังก์ชันในการเข้าถึงตัวแปร session เพื่อเรียกดูสินค้าในตะกร้า ฟังก์ชันจะถูกเรียกใช้เมื่อต้องมีการใช้ข้อมูลเกี่ยวกับสินค้าในตะกร้า ฟังก์ชันจะไม่มีการรับค่าใดๆ และคืนค่าข้อมูลสินค้าในตะกร้าจากตัวแปร session และคืนค่า null เพื่อแสดงว่าทำงานไม่สำเร็จ



รูปที่ 1.2.5(d) Algorithmic Model ของ operation getCart()

## 2. Source Code with Comment

### Cart.php (Component)

```
<?php
require_once("Account.php");
require_once("Ticket.php");

//Component Implementation
class Cart {
    public static function addtocart($flightid) {
        //check authentication, if no authen, exit
        if(Account::authenticate() != 1) return 0;
        //start session if not yet
        if(session_status() == PHP_SESSION_NONE) session_start();
        //create cart if not has
        if(!isset($_SESSION['cart'])) $_SESSION['cart'] = array();
        $ticket = Ticket::getTicketInfo($flightid); //get full ticket info
        //search for duplicate ticket, if found add its qty
        foreach($_SESSION['cart'] as &$i)
            if($i['item'] == $ticket) {
                $i['qty'] += 1;
                return 1;
            }
        //if no duplication, add anew
        $_SESSION['cart'][] = array('item'=>$ticket, 'qty'=>1);
        return 1;
    }
    public static function removefromcart($flightid) {
        //check authentication, if no authen, exit
        if(Account::authenticate() != 1) return 0;
        //start session if not yet
        if(session_status() == PHP_SESSION_NONE) session_start();
        $newcart = array(); //new blank cart for remove process
        if(sizeof($_SESSION['cart']) != 0) {
            //search for ticket to remove
            foreach($_SESSION['cart'] as &$i) {
                //if found decrease its qty
                if($i['item']['FlightID'] == $flightid) $i['qty'] -= 1;
                //every item with more than 0, move to new cart
                if($i['qty'] > 0) $newcart[] = $i;
            }
            $_SESSION['cart'] = $newcart; //new cart replace the old one
        }
        return 1;
    }
    public static function gettotalfare() {
        //check authentication, if no authen, exit
        if(Account::authenticate() != 1) return 0;
        //start session if not yet
        if(session_status() == PHP_SESSION_NONE) session_start();
        $totalfare = 0.00; //initialize total fare
        //calculate total fare from cart
        foreach($_SESSION['cart'] as $i) $totalfare += ($i['qty'] *
        $i['item']['Fare']);
        return $totalfare; //return total fare
    }
    ...
}
```

## Cart.php (Component; Continued)

```
...
public static function getcart() {
    //check authentication, if no authen, exit
    if(Account::authenticate() == 1) {
        //start session if not yet
        if(session_status() == PHP_SESSION_NONE) session_start();
        //create cart if not has
        if(!isset($_SESSION['cart'])) $_SESSION['cart'] = array();
        return $_SESSION['cart']; //return cart
    } else return null;
}
}
?>
```

## CartTester.php (Unit Test)

```
<?php
class CartTester extends PHPUnit_Framework_TestCase {
    public function setUp() {
        @session_start(); //prevent session_start() error
        parent::setUp();
    }
    public function testGetCartWithoutLogin() {
        //assert that use getcart() w/o login will return null
        $this->assertEquals(null, Cart::getcart());
    }
    public function testGetCartBlank() {
        Account::login("user2", "user2"); //login
        //assert that use getcart() w/ login will return blank array
        $this->assertEquals(array(), Cart::getcart());
        Account::logout(); //logout
    }
    public function testAddWithoutLogin() {
        //assert that using addtocart() w/o will return 0
        $this->assertEquals(0, Cart::addtocart(1));
    }
    public function testAddOneTicket() {
        Account::login("user2", "user2");
        $this->assertEquals(1, Cart::addtocart(1));
        $expected = array(array('item'=>Ticket::getTicketInfo(1),
'qty'=>1));
        $result = Cart::getcart();
        //assert that using addtocart() 1 time, have 1 ticket w/ 1 qty
        $this->assertEquals($expected, $result);
        Account::logout();
    }
    public function testAddDuplicatedTicket() {
        Account::login("user2", "user2");
        $this->assertEquals(1, Cart::addtocart(1));
        $this->assertEquals(1, Cart::addtocart(1));
        $expected = array(array('item'=>Ticket::getTicketInfo(1),
'qty'=>2));
        $result = Cart::getcart();
        //assert that using addtocart() 2 time, have 1 ticket w/ 2 qty
        $this->assertEquals($expected, $result);
        Account::logout();
    }
    ...
}
```

## CartTester.php (Unit Test; Continued)

```
...
public function testAddDifferentTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::addtocart(2));
    $expected = array(array('item'=>Ticket::getTicketInfo(1),
'qty'=>1), array('item'=>Ticket::getTicketInfo(2), 'qty'=>1));
    $result = Cart::getcart();
    //assert that using addtocart() 1 ticket, have 1 ticket w/ 1 qty
    $this->assertEquals($expected, $result);
    Account::logout();
}

public function testRemoveWithoutLogin() {
    //assert that using removefromcart() w/o login return 0
    $this->assertEquals(0, Cart::removefromcart(1));
}

public function testRemoveOneTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::removefromcart(1));
    $expected = array(array('item'=>Ticket::getTicketInfo(1),
'qty'=>1));
    $result = Cart::getcart();
    //assert that using removefromcart() will deduct qty
    $this->assertEquals($expected, $result);
    Account::logout();
}

public function testRemoveAllTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::removefromcart(1));
    $expected = array();
    $result = Cart::getcart();
    //assert that using removefromcart() can remove item when qty = 0
    $this->assertEquals($expected, $result);
    Account::logout();
}

public function testRemoveTicketNotInCart() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::removefromcart(1));
    $expected = array();
    $result = Cart::getcart();
    //assert that using removefromcart() emptycart result in emptycart
    $this->assertEquals($expected, $result);
    Account::logout();
}

public function testTotalFareWithoutLogin() {
    //assert that using gettotalfare() w/o login return 0
    $this->assertEquals(0, Cart::gettotalfare());
}
...
```

## CartTester.php (Unit Test; Continued)

```
...
public function testTotalFareOneTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $ticket1 = Ticket::getTicketInfo(1);
    $expected = $ticket1['Fare'] * 1;
    $result = Cart::gettotalfare();
    //assert that using gettotalfare() can work on one item
    $this->assertEquals($expected, $result);
    Account::logout();
}
public function testTotalFareDuplicatedTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::addtocart(1));
    $ticket1 = Ticket::getTicketInfo(1);
    $expected = $ticket1['Fare'] * 2;
    $result = Cart::gettotalfare();
    //assert that using gettotalfare() can work on duplicated item
    $this->assertEquals($expected, $result);
    Account::logout();
}
public function testTotalFareDiferrentTicket() {
    Account::login("user2", "user2");
    $this->assertEquals(1, Cart::addtocart(1));
    $this->assertEquals(1, Cart::addtocart(2));
    $ticket1 = Ticket::getTicketInfo(1);
    $ticket2 = Ticket::getTicketInfo(2);
    $expected = $ticket1['Fare'] * 1 + $ticket2['Fare'] * 1;
    $result = Cart::gettotalfare();
    //assert that using gettotalfare() can work on different item
    $this->assertEquals($expected, $result);
    Account::logout();
}
public function testTotalFareNoTicket() {
    Account::login("user2", "user2");
    $expected = 0;
    $result = Cart::gettotalfare();
    //assert that using gettotalfare() can work on empty cart
    $this->assertEquals($expected, $result);
    Account::logout();
}
}
?>
```

### 3. Test Cases and Results

ฟังก์ชัน getcart()

No.	Description	Test Data	Pre-Condition	Expected Result	Actual Result	Status
1	ใช้ getCart() เพื่อดูตะกร้าเมื่อไม่ได้เข้าสู่ระบบ	-	-	ไม่สามารถเรียกดูตะกร้าได้ (คืนค่า null)	ไม่สามารถเรียกดูตะกร้าได้ (คืนค่า null)	ผ่าน
2	ใช้ getCart() เพื่อดูตะกร้าเมื่อเข้าสู่ระบบแล้วและไม่มีสินค้าในตะกร้า	-	login ด้วย Component Account	สามารถเรียกดูตะกร้าสินค้าได้ (คืนค่า array เปล่า)	สามารถเรียกดูตะกร้าสินค้าได้ (คืนค่า array เปล่า)	ผ่าน

ฟังก์ชัน addtocart()

No.	Description	Test Data	Pre-Condition	Expected Result	Actual Result	Status
1	ใช้ addtocart() เพื่อใส่ตัวในตะกร้าเมื่อไม่ได้เข้าสู่ระบบ	flightID = 1 (เพิ่ม 1 ครั้ง)	-	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ผ่าน
2	ใช้ addtocart() เพื่อใส่ตะกร้าครั้งเดียวเมื่อเข้าสู่ระบบแล้ว	flightID = 1 (เพิ่ม 1 ครั้ง)	login ด้วย Component Account	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = 1	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = 1	ผ่าน
3	ใช้ addtocart() เพื่อใส่ตะกร้าหลายครั้งด้วย flightID เดียวกันเมื่อเข้าสู่ระบบแล้ว	flightID = 1 (เพิ่ม 2 ครั้ง)	login ด้วย Component Account	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = จำนวนครั้งที่เรียกใช้	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = จำนวนครั้งที่เรียกใช้	ผ่าน
4	ใช้ addtocart() เพื่อใส่ตะกร้าหลายครั้งด้วย flightID ต่างกันเมื่อเข้าสู่ระบบแล้ว	flightID = 1, flightID = 2 (เพิ่ม 1 ครั้ง/ 1 flightID)	login ด้วย Component Account	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = 1 สำหรับแต่ละ flightID	สามารถเพิ่มตัวที่มี flightID นั้นลงในตะกร้าและมี qty = 1 สำหรับแต่ละ flightID	ผ่าน



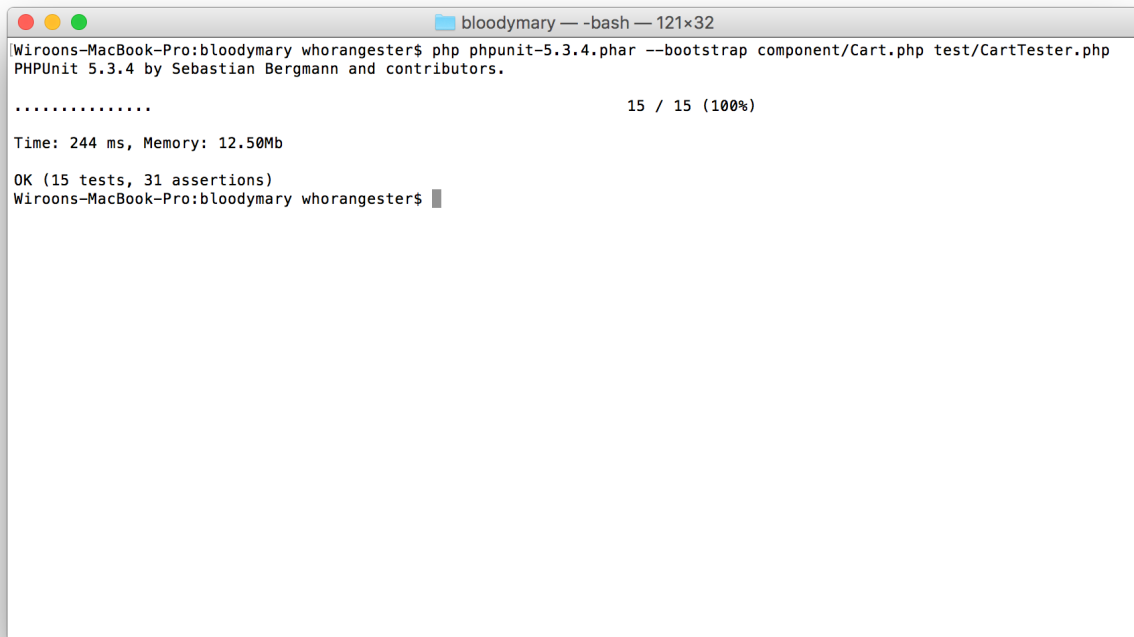
ฟังก์ชัน removefromcart()

No.	Description	Test Data	Pre-Condition	Expected Result	Actual Result	Status
1	ใช้ removefromcart() เพื่อนำตัวในตะกร้าออกเมื่อไม่ได้เข้าสู่ระบบ	flightID = 1 (ลบ 1 ครั้ง)	-	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ผ่าน
2	ใช้ removefromcart() เพื่อนำตัวในตะกร้าออกครั้งเดียว (และ qty เริ่มต้น > 1) เมื่อเข้าสู่ระบบแล้ว	flightID = 1 (เพิ่ม 2 ครั้ง ลบ 1 ครั้ง)	login ด้วย Component Account	สามารถลดจำนวนตัวที่มี flightID นั้นลง โดยมี qty น้อยลง 1	สามารถลดจำนวนตัวที่มี flightID นั้นลง โดยมี qty น้อยลง 1	ผ่าน
3	ใช้ removefromcart() เพื่อนำตัวในตะกร้าออกครั้งเดียว (และ qty เริ่มต้น = 1) เมื่อเข้าสู่ระบบแล้ว	flightID = 1 (เพิ่ม 1 ครั้ง ลบ 1 ครั้ง)	login ด้วย Component Account	ข้อมูลของตัวนั้นถูกนำออกจากตะกร้า	ข้อมูลของตัวนั้นถูกนำออกจากตะกร้า	ผ่าน
4	ใช้ removefromcart() เพื่อนำตัวที่ไม่มีในตะกร้า เมื่อเข้าสู่ระบบแล้ว	flightID = 1 (ลบ 1 ครั้ง)	login ด้วย Component Account	ไม่มีการเปลี่ยนแปลงในตะกร้า	ไม่มีการเปลี่ยนแปลงในตะกร้า	ผ่าน

ฟังก์ชัน gettotalfare()

No.	Description	Test Data	Pre-Condition	Expected Result	Actual Result	Status
1	ใช้ gettotalfare() เพื่อคำนวณราคารวมของตัวในตะกร้าเมื่อไม่ได้เข้าสู่ระบบ	-	-	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ไม่สามารถใช้ฟังก์ชันได้ (คืนค่า 0)	ผ่าน
2	ใช้ gettotalfare() เพื่อคำนวณราคารวมของตัวในตะกร้าที่มีตัวหนึ่งหน่วยเมื่อไม่ได้เข้าสู่ระบบ	flightID = 1 (เพิ่ม 1 ครั้ง)	login ด้วย Component Account	คำนวณราคารวมตัวได้ถูกต้อง	คำนวณราคารวมตัวได้ถูกต้อง	ผ่าน
3	ใช้ gettotalfare() เพื่อคำนวณราคารวมของตัวในตะกร้าที่มีตัวหนึ่งมากกว่าหน่วยที่เหมือนกันเมื่อไม่ได้เข้าสู่ระบบ	flightID = 1 (เพิ่ม 2 ครั้ง)	login ด้วย Component Account	คำนวณราคารวมตัวได้ถูกต้อง	คำนวณราคารวมตัวได้ถูกต้อง	ผ่าน
4	ใช้ gettotalfare() เพื่อคำนวณราคารวมของตัวในตะกร้าที่มีตัวหนึ่งมากกว่าหน่วยที่ต่างกันเมื่อไม่ได้เข้าสู่ระบบ	flightID = 1, flightID = 2 (เพิ่ม 1 ครั้ง/ 1 flightID)	login ด้วย Component Account	คำนวณราคารวมตัวได้ถูกต้อง	คำนวณราคารวมตัวได้ถูกต้อง	ผ่าน

## Screen Capture ของการทำ Unit Testing



```
Wiroons-MacBook-Pro:bloodymary whorangester$ php phpunit-5.3.4.phar --bootstrap component/Cart.php test/CartTester.php
PHPUnit 5.3.4 by Sebastian Bergmann and contributors.

.....
15 / 15 (100%)

Time: 244 ms, Memory: 12.50Mb

OK (15 tests, 31 assertions)
Wiroons-MacBook-Pro:bloodymary whorangester$
```

รูปที่ 3 ผลลัพธ์ของการทำ Unit Testing

### 4. Instruction to Run Unit Testing

1. เปิดใช้งานโปรแกรม xampp (หรือโปรแกรมอื่นเทียบเท่า)
2. เปิดใช้ใช้งาน php server และ mysql database บน xampp
3. import ไฟล์ bloodymary.sql (จาก github) ลงไปใน mysql database โดยควรตั้งชื่อ database ว่า bloodymary 3. ในกรณีที่ต้องการใช้ชื่อ database อื่นหรือมีการใช้ parameter ในการเข้าถึง database ต่างออกไป (username="root", password="") ต้องมีการไปแก้ไข parameter ใน Account.php, และ Ticket.php ให้ชื่อ database ตรงกัน
5. clone folder จาก github มาไว้ใน folder htdoc ของ xampp
6. (optional) ทำการลง phpunit (จาก <https://phpunit.de>) เพื่อใช้ทำ unit testing
7. ในกรณีที่ลง phpunit สามารถรัน unit testing โดยใช้คำสั่ง

```
phpunit --bootstrap [directory of component/Cart.php] [directory of test/CartTester.php]
```

8. ในกรณีที่ไม่ลง phpunit สามารถรันด้วยไฟล์ phpunit-5.3.4.phar ที่อยู่ใน folder ที่ clone ได้จาก github คำสั่งที่ใช้รัน (ตัวอย่างตามรูปที่ 3) คือ

```
php phpunit-5.3.4.phar --bootstrap [directory of component/Cart.php] [directory of test/CartTester.php]
```

5. GitHub Repository: <https://github.com/kamkanyawee/bloodymary>