[Tutorial] Viết Unit Test trong C# với NUnit

C# 24 C#.NET 4 code 39 unittest 3 unit test 6 Testing 26 test 8

Huy Hoàng Phạm viết ngày 12/10/2015

[Tutorial] Viết Unit Test trong C# với NUnit

Giới thiệu tổng quan về Unit Test

Ở trường đại học chắc các bạn đã được học khái niệm về Unit Test trong môn "Kiểm thử chất lượng phần mềm". Nói một cách dễ hiểu, unit test tức là **code dùng để test code ta đã viết**.

Một số đặc điểm của unit test:

1. Code unit test phải ngắn gọn, dễ hiểu, dễ đọc.
2. Mỗi unit test là 1 đơn vi riêng biệt, độc lập, không phụ thuộc vào unit khác.
3. Mỗi unit test là 1 method trong test class, tên method cũng là tên UnitTest. Do đó ta nên đặt tên hàm rõ ràng, nói rõ unit test này test cái gì (Test_A_Do_B), tên method có thể rất dàiii cũng không sao.
4. Unit Test phải nhanh, vì nó sẽ được chạy để kiểm định lỗi mỗi lần build. Do đó trong unit test nên hạn chế các task tốn thời gian như gọi I/O, database, network,…

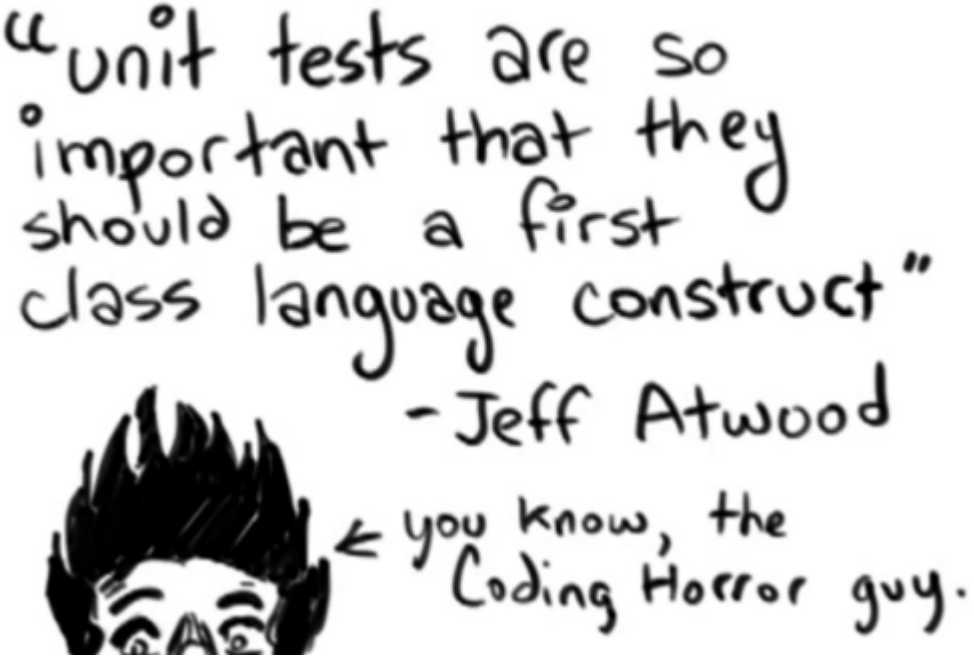
5. Unit Test nên test từng đối tượng riêng biệt. Vd: Unit Test cho Business Class thì chỉnh test chính BusinessClass đó, không nên dụng tới các class móc nối với nó (DataAccess Class chẳng hạn).

Một số bạn sẽ thắc mắc: "Ơ hay, trong business class thì phải gọi data access chứ?", và "code mình dính chùm lắm, làm sao test từng class riêng được".

- Để viết unit test, các class của bạn phải có quan hệ "lõng lẽo" với nhau (loose coupling), nếu không viết được unit test nghĩa là các class dính với nhau quá chặt, sẽ khó thay đổi sau này. Hãy áp dụng các nguyên lý SOLID vào code, viết code với tư tưởng "viết sao để unit test được" sẽ là code của bạn uyển chuyển, dễ test hơn.
- Về vấn đề Business Class và Data Access, các class không nên gọi trực tiếp lẫn nhau mà gọi thông qua interface. Khi unit test, ta sẽ thay các hiện thực interface này bằng các class giả, thay thế cho class data access thật. Mình sẽ nói rõ về vấn đề này ở những bài viết về ***Inversion of Control, Dependency Injection*** trong tương lai.

Một số lợi ích của Unit Test:

- Nếu viết Unit Test một cách cẩn thận, **code của bạn sẽ ít lỗi hơi**, vì Unit Test sẽ phát hiện lỗi cho bạn.
- Phát hiện những hàm chạy chậm và không hiệu quả thông qua thời gian chạy của Unit Test.
- Tăng sự tự tin khi code, vì đã có Unit Test phát hiện lỗi.
- Khi refactor code, sửa code hay thêm chức năng mới, **Unit Test đảm bảo chương trình chạy đúng**, phát hiện những lỗi tiềm tàng mà ta bỏ lỡ.


"unit tests are so important that they should be a first class language construct"
- Jeff Atwood
you know, the Coding Horror guy.

Gần đây, mô hình phát triển TDD (Test Driven Development) đang trở nên hot, được áp dụng nhiều. Mô hình này dựa trên khái niệm: Với mỗi chức năng, ta viết Unit Test trước, sau đó viết hàm hiện thực chức năng để unit test pass. Một số công ty ở Việt Nam cũng đang áp dụng mô hình này, trong khi phỏng vấn xin việc cũng có nhé :D.

Các bạn có thể xem thêm về Unit Test và TDD ở đây: http://www.pcworld.com.vn/articles/cong-nghe/cong-nghe/2005/12/1188434/unit-test-voi-phat-trien-phan-mem-hien-dai/

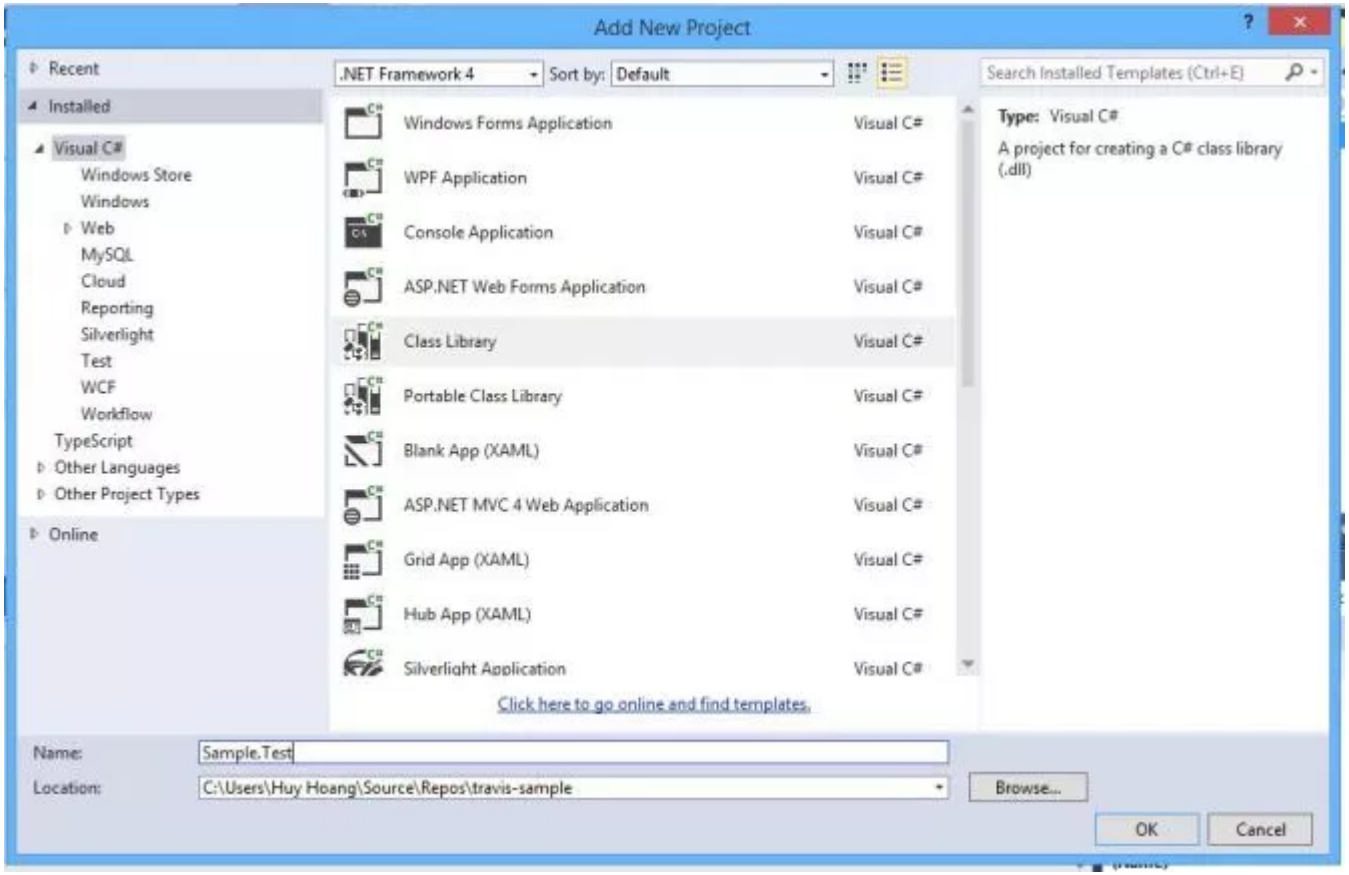
Unit Test trong C# với NUnit

Trước đây, để viết Unit Test trong C#, ta thường phải tạo một project test riêng, sử dụng thư viện MSTest của Microsoft. MSTest hỗ trợ khá nhiều chức năng: Test dữ liệu từ database, đo performance hệ thống, xuất dữ liệu report. Tuy nhiên, do MSTest đi kèm với Visual Studio, không thể chạy riêng rẽ, lại khá nặng nề, do đó NUnit được nhiều người ưa thích hơn. NUnit có 1 bộ runner riêng, có thể chạy UnitTest độc lập không cần VisualStudio, ngoài ra nó cũng hỗ trợ một số tính năng mà MSTest không có (parameter test, Assert Throw).

Nếu bạn chưa bao giờ viết Unit Test, bắt đầu với NUnit cũng là 1 lựa chọn không tồi. Đầu tiên, ta hãy tạo 1 project console trong Visual Studio. Ta viết 1 class Calculator với các 2 hàm Add và Subtract:

```
    public class Calculator

        ublic int Add(int x, int y)

            return x + y;
        }

        ublic int Subtract(int x, int y)

            return x - y;
        }

    }
```

Bây giờ, ta sẽ viết UnitTest để test 2 hàm này. Thông thường, code Unit Test sẽ được nằm ở một project Test riêng biệt, do đó ta sẽ tạo 1 project mới, thêm ".Test" đằng sau tên. Các bạn nhớ chọn kiểu project là Class Library nhé.


Add New Project
Recent
Installed
Visual C#
Windows Store
Windows
Web
MySQL
Cloud
Reporting
Silverlight
Test
WCF
Workflow
TypeScript
Other Languages
Other Project Types
Online
.NET Framework 4
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application
WPF Application
Console Application
ASP.NET Web Forms Application
Class Library
Portable Class Library
Blank App (XAML)
ASP.NET MVC 4 Web Application
Grid App (XAML)
Hub App (XAML)
Silverlight Application
Visual C#
Type: Visual C#
A project for creating a C# class library (.dll)
Click here to go online and find templates.
Name: Sample.Test
Location: C:\Users\Huy Hoang\Source\Repos\travis-sample
Browse...
OK
Cancel

Ở project mới, bấm vào Tools -> Library Package Manager -> Package Manager Console, gõ vào: `Install-Package NUnit -Version 2.6.4` để cài đặt NUnit nhé. Ta tạo 1 class mới có tên là CalculatorTest. Bắt đầu viết thôi:

```
     using NUnit.Framework;

    ublic class CalculatorTest
       private Calculator _cal;

        ublic void Setup()
           _cal = new Calculator();
       }

        ublic void OnePlusOneEqualTwo()
           Assert.AreEqual(2, _cal.Add(1, 1));
       }

        ublic void TwoPlusTwoEqualFour()
           Assert.AreEqual(4, _cal.Add(2, 2));
       }

        ublic void FourPlusOneEqualFive()
           Assert.AreEqual(5, _cal.Add(4, 1));
       }
   }
```

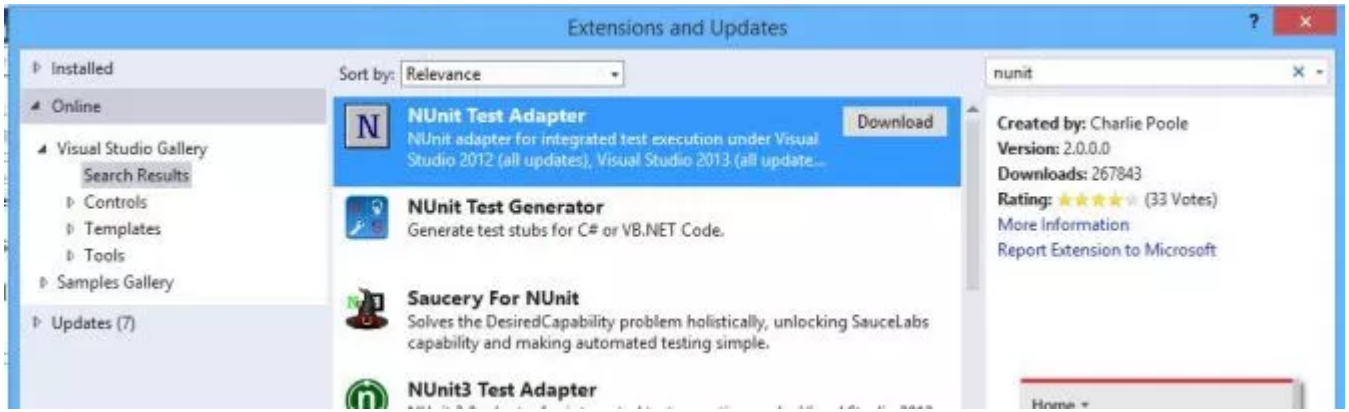
Giải thích một số annotation:

- Annotation [TextFixture] đặt vào đầu class chứa các unit test, đánh dấu đây là một bộ các unit test.
- Annotation [SetUp] để đánh dấu hàm setup. Hàm này sẽ được chạy vào đầu mỗi unit test.
- Annotation [Test] để đánh dấu hàm bên dưới là một unit test. Nhiều khi bạn viết unit test xong nhưng không thấy hiện unit test lên là do quên annotation này nhé.

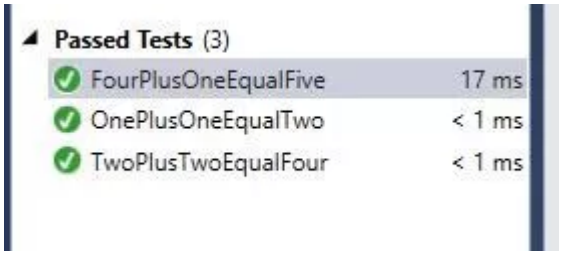
Để chạy Unit Test, ta cần cài *NUnit.TestAdapter*, đây là bộ runner cho phép chạy các Unit Test của NUnit trong Visual Studio. Bạn cũng có thể dùng 1 trong 2 cách sau:

1. Tải NUnit GuiRunner về, mở file dll của project test, các unit test sẽ hiện ra để chạy.
2. Tải Reshaper. Reshaper cũng tích hợp sẵn 1 bộ Runner cho phép chạy Unit Test của NUnit.

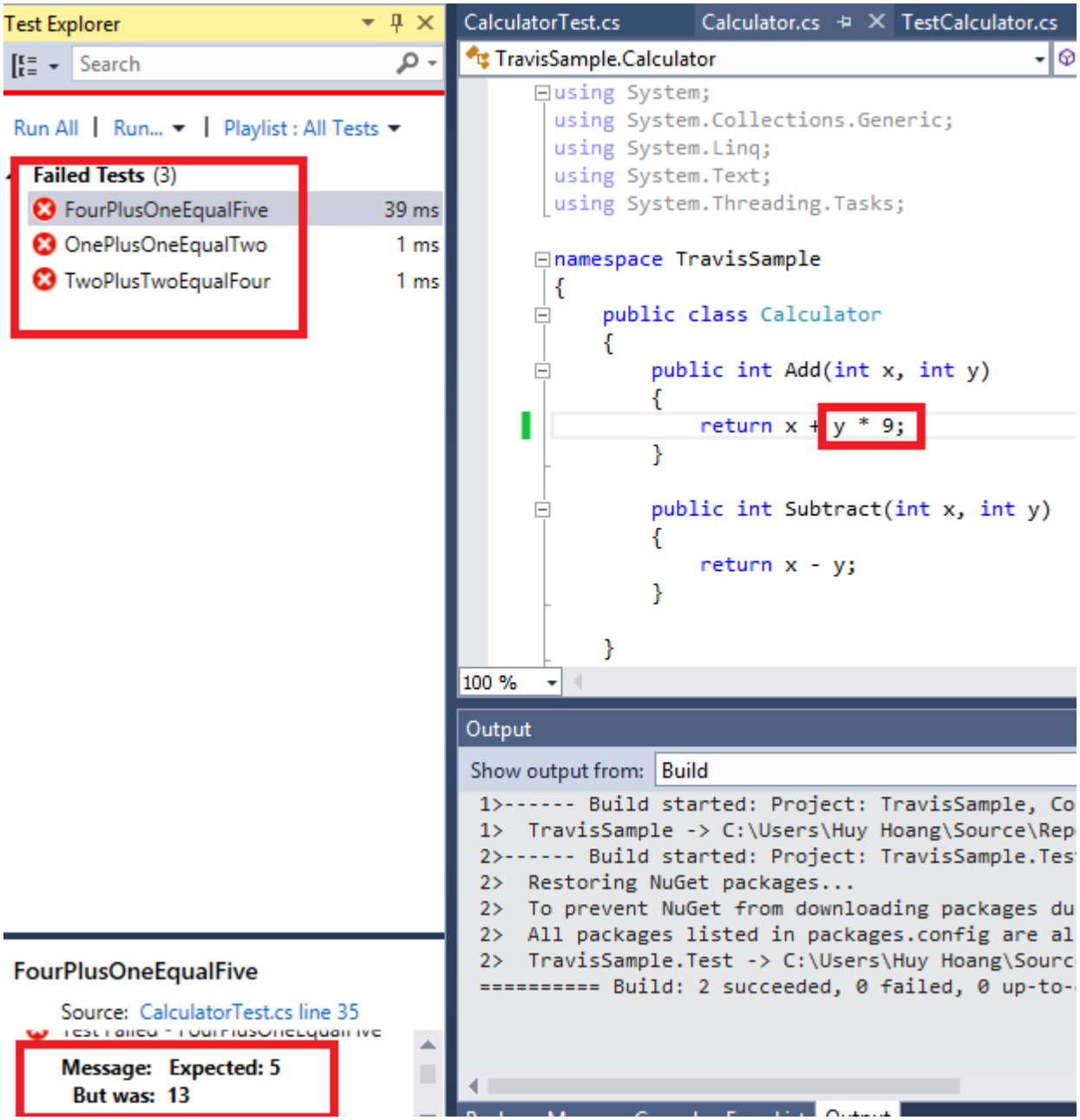
Để cài TestAdapter cho Visual Studio, ta vào Tool, Extentions and Updates, tìm NUnit và cài đặt. Cài xong nhớ Restart lại Visual Studio nhé.



Bấm Test -> Run -> All Tests để bắt đầu chạy. Ta sẽ thấy toàn bộ các case đã pass, màu xanh hiền hòa như trong ảnh.



Ta cố ý sửa lại code để code chạy sai. Chạy lại Unit Test, ta sẽ thấy các case đã fail. Unit Test đã tự động bắt lỗi chương trình cho bạn rồi đấy. Bấm vào từng test case, ta sẽ thấy kết quả mong muốn ở đoạn "**Excepted**", kết quả code chạy ở đoạn "**But was**".


Test Explorer
Search
Run All | Run... | Playlist : All Tests
Failed Tests (3)
FourPlusOneEqualFive 39 ms
OnePlusOneEqualTwo 1 ms
TwoPlusTwoEqualFour 1 ms
FourPlusOneEqualFive
Source: CalculatorTest.cs line 35
Message: Expected: 5
But was: 13
CalculatorTest.cs
Calculator.cs
TestCalculator.cs
TravisSample.Calculator
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace TravisSample
{
public class Calculator
{
public int Add(int x, int y)
{
return x + y * 9;
}
public int Subtract(int x, int y)
{
return x - y;
}
}
100 %
Output
Show output from: Build
1>------ Build started: Project: TravisSample, Co
1> TravisSample -> C:\Users\Huy Hoang\Source\Rep
2>------ Build started: Project: TravisSample.Tes
2> Restoring NuGet packages...
2> To prevent NuGet from downloading packages du
2> All packages listed in packages.config are al
2> TravisSample.Test -> C:\Users\Huy Hoang\Sourc
========== Build: 2 succeeded, 0 failed, 0 up-to-

Mục đích của bài viết là hướng dẫn các bạn nhập môn việc viết Unit Test cho C# bằng NUnit nên mình sẽ dừng ở đây. Nếu được ủng hộ, mình sẽ giới thiệu một số kĩ thuật phức tạp hơn như: Sử dụng Mock Object, Parameterlized Test, … ở những bài sau. Bạn nào muốn viết unit test cho javascript có thể thử tìm hiểu về Jasmine, mình đã có 2 bài giới thiệu tổng quát. Cảm ơn và hẹn gặp lại.

**Bản gốc: Blog tôi đi code dạo

Chia sẻ bài viết với bạn bè nữa nhé!

Huy Hoàng Phạm
54 bài viết. 1098 người follow

SOLID là gì – Áp dụng các nguyên lý SOLID để trở thành lập trình viên code “cứng”

solid lập trình code programming developer

SOLID là gì – Áp dụng các nguyên lý SOLID để trở thành lập trình viên code “cứng” Trong quá trình học, hầu như các bạn sinh viên đều được học một s...

Huy Hoàng Phạm viết hơn 3 năm trước

82 8

62 8

Nhập môn Design Pattern (Phong cách kiếm hiệp)

design pattern code programming nhập môn

Nhập đề Kinh thư ghi lại rằng, con đường tu chân có 3 cảnh giới: Luyện khí, Trúc cơ và Kết đan. Luyện khí là quá trình rèn thân luyện thể, cho phà...

Huy Hoàng Phạm viết 3 năm trước

62 8

59 8

Tăng sức mạnh cho javascript với lodash

Javascript lodash underscore

Tăng sức mạnh cho javascript với lodash Lần này mình sẽ giới thiệu 1 thư viện javascript vô cùng bá đạo có tên là "lodash]1]", có thể nói nó là LI...

Huy Hoàng Phạm viết hơn 3 năm trước

59 8

Bài viết liên quan

0 0

Reflection - Complicate Example in C#

C# Reflection Example

Mình khá là lười, nên mình sẽ không đưa định nghĩa hay usage của reflection vào đây. Vì dù sao mình cũng sẽ chỉ copy thôi :smile:. Vậy nên chúng ta...

Rice viết 8 tháng trước

0 0

25 4

Tạo dummy data với Faker và Mockaroo – Xa rồi những ngày nhập tay nhàm chán

Database data dummy data Javascript SQL C#

Tạo dummy data với Faker và Mockaroo – Xa rồi những ngày nhập tay nhàm chán Cuộc đời một thằng developer có rất nhiều việc rất chán nhưng phải làm...
Huy Hoàng Phạm viết hơn 3 năm trước
25 4

5 2

Sử dụng OpenCV function với C#.Net

C# #OpenCV

Sử dụng OpenCV function với C.Net Nếu bạn đã từng làm một số project có liên quan đến xử lý ảnh thì chắc hẳn sẽ biết thư viện OpenCV. Hiện tại đã ...
vankhangfet viết hơn 3 năm trước
5 2