.Bạn cần chuẩn bị những gì trước khi đi phỏng vấn vị trí .NET Developer?

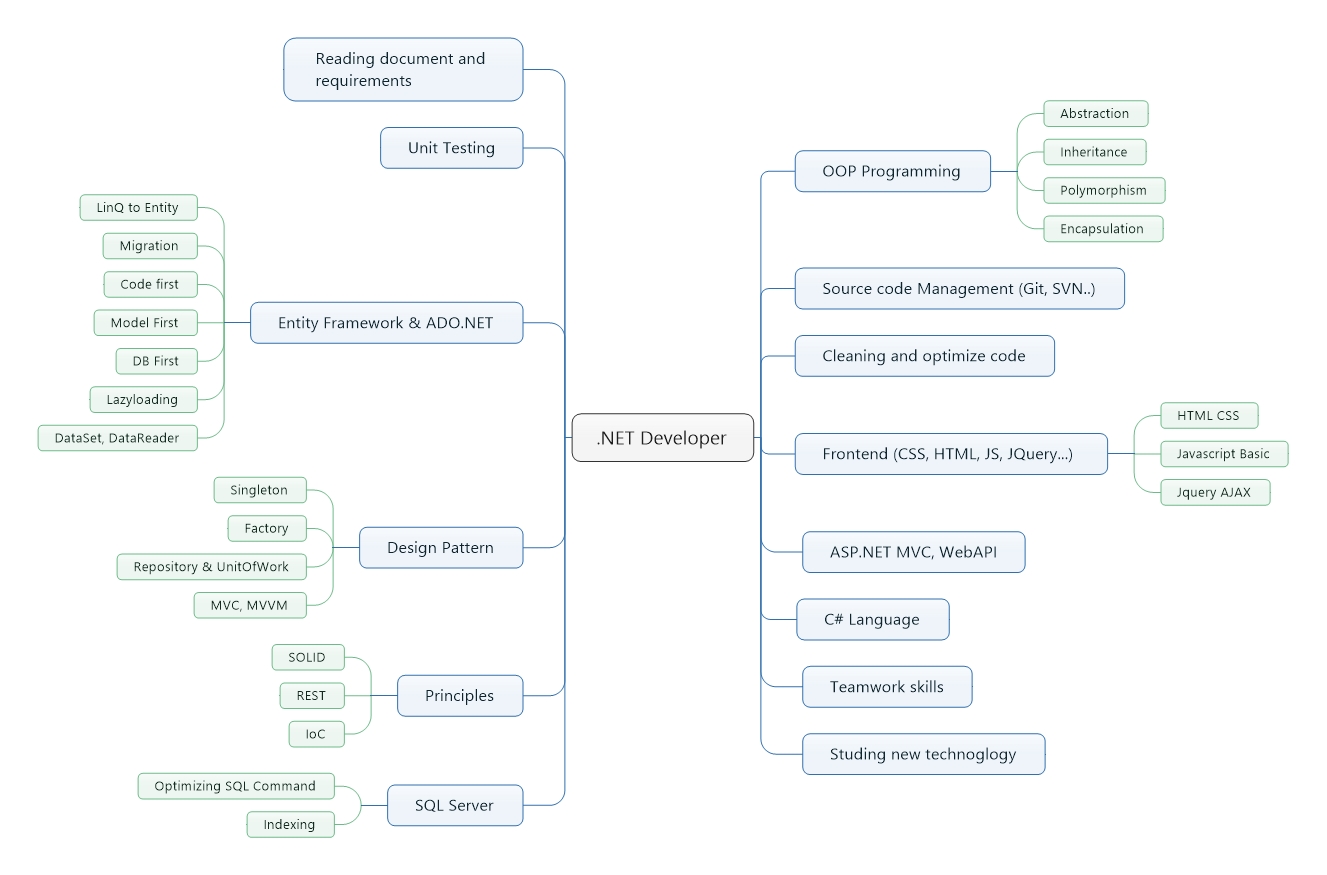
**Giới thiệu**

Một vài nhận xét về các ứng viên hiện nay mình đã phỏng vấn, nhìn chung thị trường rất nhiều người làm lập trình, và ở Việt Nam thì lượng lập trình viên .NET chiếm cũng phần lớn. Nhưng tại sao nhà tuyển dụng vẫn rất thiếu nguồn nhân lực chất lượng tốt? Mình không phủ nhận Việt Nam mình rất nhiều người làm tốt nhưng đến một thời điểm nào đó thì họ không muốn nhảy việc, hơn nữa họ đã được thăng tiến ở một công ty nào đó và nhận những chính sách đãi ngộ tốt. Nên những người đi phỏng vấn thường là những bạn mới ra trường hoặc ít kinh nghiệm, còn lại nhiều kinh nghiệm thì thường là muốn thay đổi môi trường, muốn tăng lương…tức là không được đãi ngộ tốt ở công ty hiện tại.

Đánh giá tổng quan như vậy để thấy rằng các bạn đang rất có nhiều cơ hội để tìm việc trong dòng .NET Developer. Vì thế các bạn cần tự tin nếu như mình có kiến thức và kỹ năng tốt. Vì tự tin các bạn sẽ thể hiện được năng lực tốt nhất có thể trước mắt nhà tuyển dụng đồng thời đàm phán được một mức lương xứng đáng với khả năng của mình.

1. **Vậy trước khi đi phỏng vấn chúng ta cần chuẩn bị những gì?**

Dựa vào cái mindmap mình đã xây dựng theo hình bên dưới để các bạn mường tượng ra tổng quan các bạn cần những gì? Sau đó mình sẽ giảng giải chi tiết hơn về từng phần một. Có những thứ đó các bạn sẽ tự tin và thấy mình còn thiếu những gì để có thể lấp vào đó.



1. **Cơ bản về lập trình OOP**

Đây là điều bắt buộc các bạn phải biết trong khi đi phỏng vấn, các bạn không nắm vững cái này thì sẽ có khả năng cao là bị loại vì đây là những kiến thức hết sức căn bản. Để nắm chắc OOP tức là lập trình hướng đối tượng, các bạn cần nắm chắc khái niệm và các tính chất của nó (có 4 tính chất). Mỗi tính chất cac bạn phải hiểu và lấy ví dụ được đồng thời các bạn cũng phải nêu ra được tác dụng của nó.

----------------------------------------------------& **Trả lời** &----------------------------------------------------------

**OOP:** là nghệ thuật quan sát các đối tượng trong tự nhiên rồi cố gắng nắm bắt những hành động cùng đặc tính của chúng và biểu diễn dưới dạng đối tượng ảo trong ngôn ngữ lập trình

Lập trình hướng đối tượng luôn đi kèm 4 đặc điểm chính

Abstraction : tính trừu tượng.  
Encapsulation : tính đóng gói.  
Inheritance : tính kế thừa.  
Polymorphism : tính đa hình.

**Tính đóng gói (encapsulation) và che giấu thông tin (information hiding)**

Như vậy tính đóng gói có thể hiểu: Gói dữ liệu (data, ~ biến, trạng thái) và mã chương trình (code, ~ phương thức) thành một cục gọi là lớp (class) để dễ quản lí. Trong cục này thường data rất rối rắm, không tiện cho người không có trách nhiệm truy cập trực tiếp, nên thường ta sẽ che dấu data đi, chỉ để lòi phương thức ra ngoài

**Tính kế thừa (inheritance) Và đa hình(Polymorphism)**

**-** Kế thừa là cách tạo lớp mới từ các lớp đã được định nghĩa từ trước

- Lớp cha có thể chia sẻ dữ liệu và phương thức cho các lớp con, các lớp con khỏi phải định nghĩa lại những logic chung, giúp chương trình ngắn gọn. Nếu lớp cha là interface, thì lớp con sẽ di truyền những contract trừu tượng từ lớp cha

**Chú ý**:   
- Một lớp cha có thể có nhiều lớp con  
- Đến lượt mình mỗi lớp con lại có thể có các con khác   
- Trong C++ cho phép đa kế thừa (một lớp con có thể nhận hơn 1 lớp cha)  
- Java chỉ cho phép mỗi lớp con kế thừa 1 và chỉ một lớp cha/ từ khóa implement sẽ thay thế cho đa kế thừa từ C++

* **Tính trừu tượng, phương thức ảo:**

**Định nghĩa**: Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết **(chỉ khai báo)**. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác **(sự override từ lớp con).** Tính chất này thường được gọi là sự trừu tượng của dữ liệu.

Phương thức ảo là phương thức được định nghĩa ở lớp cơ sở (lớp cha) mà các lớp dẫn xuất (lớp con) muốn sử dụng phải định nghĩa lại. Dùng từ khoá virtual (c++) hay abstract (java)để khai báo phương thức ảo:

* Trong Class ảo có thể có phương thức ảo hoặc không nhưng phương thức ảo bắt buộc phải ở trong class ảo
* phương thức ảo không chứa body
* Khi kế thừa class ảo, bắt buộc phải viết lại phương thức ảo của nó

1. [**Ngôn ngữ lập trình C#**](http://tedu.com.vn/khoa-hoc/lap-trinh-visual-c-can-ban-cho-nguoi-moi-bat-dau-2.html)

Về ngôn ngữ lập trình C#, các bạn không cần thiết phải cái gì cũng biết nhưng một số điểm các bạn nên biết về Interface và Abstract class hay hiểu biết về các collection trong C#.NET. Ngoài ra các bạn cũng cần hiểu về đa luồng, extension methods, delegate, xử lý bất đồng bộ, overriding method, và tổng quan các thư viện hay dùng nhất.

----------------------------------------------------& **Trả lời** &----------------------------------------------------------

**Interface và Abstract class**

Giống nhau:

- Đều không thể khởi tạo đối tượng bên trong được

- Đều có thể khai báo các phương thức nhưng không thực hiện chúng

- Đều bao gồm các phương thức Abstract

- Đều được thực thi từ các class con hay còn gọi kế thừa dẫn xuất

- Đều có thể kế thừa từ nhiều interface

Khác nhau

|  |  |
| --- | --- |
| Abstract class | Interface |
| Cho phép khai báo field | Không cho phép |
| Các phương thức có thể có thân hàm hoặc không có thân hàm | Chỉ khai báo không có thân hàm |
| Class dẫn xuất chỉ kế thừa được từ 1 Abstract class và nhiều interface | Class triển khai có thể triển khai nhiều interface |
| Có chứa constructor | Không có |
| Các phương thức có từ khóa access modifier | Không có |

**Collection**

- Các lớp collection là các đặc biệt dùng để lưu trữ và thu hồi dữ liệu

- Hỗ trợ Stack, Queue, List và Hash Table..

- Cấp phát bộ nhớ động cho các phần tử và truy cập một danh sách cách item dựa trên một chỉ mục

- Các lớp được sử dụng phổ biến của System.Collection namespace là: ArrayList, Hashtable, SortedList, Stack, Quenue, BitArray

**Đa luồng: (Multithread)**

- Mỗi thread định nghĩa một dòng điều khiển duy nhất, thực hiện một công việc duy nhất

- Vòng đời của một Thread bắt đầu khi một đối tượng của System.Threading.Thread được tạo và kết thúc khi Thread đó được kết thúc hoặc hoàn thành việc thực thi.

Xem thêm: http://vietjack.com/csharp/da\_luong\_trong\_csharp.jsp

**Delegate**

- Tương tự như con trỏ tới các hàm

- Một Delegate là một biến kiểu tham chiếu mà giữ tham chiếu tới một phương thức (Phương thức đó có thể thay đổi tại runtime)

- Các Delegate được sử dụng để triển khai các sự kiện và các phương thức call-back

Xem thêm: <http://vietjack.com/csharp/delegate_trong_csharp.jsp>

**Override và Overload**

|  |  |
| --- | --- |
| Override | Overload |
| Phương thức override và phương thức được override phải giống nhau | - Kiểu dữ liệu trả về của phương thức overload có thể giống nhau hoặc khác nhau  - Số lượng tham số hoặc kiểu dữ liệu của tham số ở các phương thức overload phải khác nhau |
| - Không thể thu hẹp phạm vi truy cập (access modifier) của phương thức được override | - Có thể mở rộng hoặc thu hẹp phạm vi truy cập của phương thức được overload |
| - Không thể overriding contructor method | - Overloading được contructor method |
| - Chỉ thực hiện được đối với các class có quan hệ kế thừa. | - Chỉ thực hiện trong nội bộ của class |
| - Là phương thức đa hình khi chạy runtime ( Tức là chỉ khi chương trình chạy, thì chúng ta mới biết phương thức được gọi là phương thức nào). | - Là phương thức đa hình khi biên dịch (compiler) (Khi biên dịch mới biết đang sử dụng phương thức ở trong lớp nào). |
| - Không cho phép tạo ra những ngoại lệ khác loại hoặc không phải đối tượng thuộc lớp con của lớp có thể hiện là ngoại lệ từ phương thức được override | - Cho phép tạo ra những ngoại lệ hoàn hảo mới so với những ngoại lệ từ phương thức được overload |

1. [**ASP.NET MVC**](http://tedu.com.vn/khoa-hoc/lap-trinh-du-an-website-ban-hang-aspnet-mvc-4-1.html)**và WebAPI**

Về ASP.NET MVC các bạn cần nắm rõ mô hình MVC là gì? Trong ASP.NET MVC có các thành phần gì? Cách truyền dữ liệu từ Controller về View có những cách nào? ViewBag, ViewData, TempData và  Session? Sự khác nhau giữa chúng như thế nào? Khi nào thì dùng cái nào?

Ngoài ra có cơ chế Routing, ASP.NET Identity, Antiforeign, WebAPI, kiến thức về REST và Restful API…

**V. Entity Framework và ADO.NET**

Câu hỏi hay hỏi nhất trong khi phỏng vấn .NET Developer là bạn đã làm việc với Entity Framework chưa? Có những cách làm việc nào? Vậy là các bạn phải trả lời được có 3 cách là DB First, Code First và Model First. Vậy là các bạn phải nêu được ưu nhược điểm của cả 3. Thường là DB First và Code First sẽ được hỏi nhiều nhất.

Một trong những câu hỏi hay được hỏi nhất về Entity Framework sẽ là phần Migration nếu bạn trả lời hay dùng Code First. Còn lại DB First thì các bạn cũng sẽ được hỏi những nội dung còn lại ngoài phần migration đó là phải hiểu về Linq to Entity tức là khi nào dữ liệu của bạn được load ra, dùng Iqueryable ra sao? Cách tối ưu hóa câu lệnh LinQ khi truy vấn database rồi cách phân trang bản ghi với LINQ.

Ngoài ra các bạn cần nắm được cách debug từ LINQ bản chất là gen ra câu lệnh SQL để execute trong SQL Server nên các bạn cần nắm được cách thức để debug, tracing và tối ưu hóa nó.

Đôi khi các dự án có làm ADO.NET có thể là call các store procedure trong SQL Server vì giờ không ai viết câu lệnh raw SQL trong code cả

----------------------------------------------------& **Trả lời** &----------------------------------------------------------

**Entity Framework**

- Là một bộ ánh xạ đối tượng – quan hệ

- Các thành phần chính: Code, Model, Database

+ Code: Là mã lệnh tạo thành các lớp đối tượng

+ Model: Là sơ đồ các thực thể và đường kết nối các quan hệ

+ Database: Là cơ sở dữ liệu ( SQL Server, MySQL, Oracle )

- Có 3 cách sử dụng Entity Framework: Code First, Model First, Database First

+ Code First: Nên dùng khi đã có mô hình cơ sở dữ liệu (không phải tạo)

+ Model First: Nên dùng khi ta bắt đầu thiết kế CSDL từ đầu ( từ chưa có gì)

+ Database First: Nên dùng khi đã có sẵn cơ sỡ dữ liệu

**Code First Migration**

- Là những vấn đề thường gặp phải khi cập nhật Database schema

**Entity Frame work và Linq to Sql**

|  |  |
| --- | --- |
| Linq to SQL | Entity Frame work |
| - Chỉ làm việc với CSDL của SQL Server | - Làm việc với CSDL: Oracle, MySQL, SQL Server |
| - Tạo ra file .dbml | - Tạo ra file .edmx, .csdl, .msl, .ssdl hoặc các file class .cs thông thường |
| - Không hỗ trợ các kiểu phức tạp | - Hỗ trợ các kiểu phức tạp |
| - Không thể tạo CSDL từ đối tượng | - Có thể tạo CSDL từ Model/ Code đã thiết kế |
| - Chỉ có kiểu ánh xạ 1 – 1 giữa các đối tượng class với Table/ View | - Có thể ánh xạ 1-1, 1-nhiều, nhiều-nhiều |
| - Sử dụng DataContext | - Sử dụng DataContext, Objectcontext, Entity SQL |

**Debug trong LINQ**

- Có thể ozCode

**VI. Kỹ năng Unit Test**

Đây là kỹ năng viết các đoạn code để kiểm thử ở mức đơn vị đối với các function code ra, trong .NET các bạn thường sử dụng MSUnit có sẵn của Visual Studio hoặc Nunit để viết các đoạn mã kiểm thử. Kỹ năng này không bắt buộc nhưng là điểm cộng của các bạn.

----------------------------------------------------& **Trả lời** &----------------------------------------------------------

- Một unit là một thành phần PM nhỏ nhất dùng để kiểm tra các function , procedure, class, method…

* Các khái niệm trong gặp:

1. Assertion: Các công việc kiểm thử cần kiểm tra
2. Test point: Là một đơn vị kiểm tra nhỏ nhất, chỉ chứa một Assertion.
3. Test case: Là tập hợp các test point
4. Test suite: Là tập hợp các test case định nghĩa cho từng module hoặc hệ thống con.
5. Regresstion Testing: (Hoặc automated testing) : Là một phương pháp kiểm nghiệm tự động
6. Production code: Phần mềm chính của ứng dụng được chuyển giao cho khách hàng.
7. Unit Testing code: Phần mã phụ để kiểm tra ứng dụng chính.

* Có 3 trạng thái : Lỗi, Tạm ngừng, làm việc

**VII. SQL Server**

Về SQL Server ngoài việc viết lệnh CRUD thì các bạn còn phải tìm hiểu về Store Procedure, Triggers, Funtions, View đồng thời quan  trọng hơn là cách viết lệnh tối ưu, đánh index để tăng tốc độ cho câu lệnh.

**VIII. Kỹ năng quản lý source code**

Các bạn cần có kỹ năng quản lý source code khi làm việc với các SCM như [Git](http://tedu.com.vn/khoa-hoc/quan-ly-source-code-trong-du-an-voi-git-8.html)hay SVN, TFS…để giúp làm việc nhanh và hiệu quả hơn.

----------------------------------------------------& **Trả lời** &-------------------------------------------------

* SVN (Subversion) là hệ thống quản lý version, hỗ trợ làm việc nhóm.
* TFC (Team foundation server) là công cụ quản lý vòng đời ứng dụng

**IX. Một số Design Pattern**

Các bạn làm dev có thể không phải sử dụng nhiều vì các SA họ đã áp dụng đối với core của công ty nhưng các bạn nên biết để hiểu và tốt cho việc phát triển sự nghiệp của mình. Hơn nữa lên senior thì phải biết Design Pattern chứ nhỉ.

----------------------------------------------------& **Trả lời** &---------------------------------------------------------

* Design Pattern: là kỹ thuật trong lập trình hướng đối tượng sử dụng được trong hầu hết các ngôn ngữ.
* Giúp dự án dễ bảo trì, nâng cấp và mở rộng
* Hạn chế được các lỗi tiềm ẩn
* Được chia làm 3 dạng chính:

+ Nhóm khởi tạo: Khởi tạo đối tượng

+ Nhóm cấu trúc: Thiết lập, định nghĩa quan hệ giữa các đối tượng

+ Nhóm ứng xử: Thực hiện các hành vi của đối tượng

**XII. Principles**

Các bạn cũng phải biết một số principle tiêu biểu là SOLID, cái này thì trong TEDU có bài viết về **SOLID**các bạn có thể [xem ở đây](http://tedu.com.vn/design-pattern/nguyen-tac-solid-va-vi-du-trong-c-49.html).

----------------------------------------------------& **Trả lời** &----------------------------------------------------------

SOLID: Là bộ 5 chỉ dẫn lập trình gồm:

1. Single Responsibility principle (SRP) ( Trách nhiệm duy nhất)

*VD: 1 class chỉ nên giữ một trách nhiệm*

1. Open/Closed principle ( OCP) (Đóng mở)

*VD: Có thể viết thêm một class mở rộng từ class cũ, không nên sửa class cũ*

1. Liskov substitution principle (LSP) (Thay thế Liskov)

*VD: Các object của class con có thể thay thế class của lớp cha mà không làm thay đổi tính đúng đắn của chương trình.*

1. Interface segregation principle (ISP) (Chia tách giao diện)

*VD: Thay vì dùng một interface lớn, ta nên tách thành nhiều interface nhỏ với nhiều mục đích cụ thể*

1. Dependency inversion principle (DIP) (Nghịch đảo phụ thuộc)

*VD: Các module cấp cao không nên phụ thuộc vào module cấp thấp. Cả hai nên phụ thuộc vào Abstraction*

* *Interface (Abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. ( Các class giao tiếp với nhau thông qua interface không phải thông qua implementation)*

**XIII. Kiến thức Frontend (**[**Javascript**](http://tedu.com.vn/khoa-hoc/khoa-hoc-java-script-can-ban-12.html)**,**[**Jquery**](http://tedu.com.vn/khoa-hoc/lap-trinh-jquery-tu-co-ban-den-nang-cao-4.html)**, Html, CSS,**[**Angular**](http://tedu.com.vn/khoa-hoc/lam-du-an-thuc-te-voi-webapi-angularjs-va-entity-framework-code-first-7.html)**…)**

Các bạn biết các kiến thức này mặc dù không phải là .NET nhưng nếu một công ty họ tuyển Fullstack developer thì những điều này giúp bạn tăng giá trị trong mắt nhà tuyển dụng.

----------------------------------------------------& **Trả lời** &---------------------------------------------------------

**XIV. Cách viết code sạch và tối ưu hóa code**

Cách viết code sạch và tối ưu này mình rất coi trọng vì nếu các bạn làm junior hay senior thì việc các bạn viết Code nói lên chất lượng thực sự của các bạn. Kỹ năng này bao gồm việc đặt tên biến, tên hàm, class, cách viết code sao cho dễ đọc dễ maintenance …Giúp việc sử dụng lại code dễ dàng hơn nhất là khi các bạn làm một sản phẩm lâu dài thì nó lại càng hữu ích. Đây là một kỹ năng rất cần thiết và rất được các bậc tiền bối coi trọng.

**Tóm lại**

Là một .NET developer không chỉ biết code tốt, mà cần các kỹ năng liên quan nữa như thế mới thực sự có value. Hơn nữa code  tốt không bằng thái độ tốt, nhiều khi kỹ năng của bạn chưa được như người ta mong muốn nhưng thái độ của bạn khi phỏng vấn cũng làm người ta thấy mình có tiềm năng phát triển và có thể đáp ứng được công việc. Kể cả khi mình giỏi thật thì thái độ cũng phải khiêm tốn và cầu thị vì không ai giỏi tất cả. Cuối cùng là tư duy về sản phẩm, một developer giỏi không phải chỉ giỏi kỹ thuật mà phải hiểu khách hàng cần gì cũng đừng giới hạn mình theo những gì mình nói trong bài viết này mà phải mở rộng hơn nữa? Làm khách hàng hài lòng và làm sao để lợi cho khách hàng, lợi cho người thuê mình tức là lợi cho chính mình.

***(Lưu ý: Các trang web đăng lại bài viết này đều phải dẫn nguồn từ TEDU.COM.VN)***