C#. Стиль кода, Generics, Lambda
ООП. Часть II

Лектор: Юрий Коноплев

31.07.19

ИНГОССТРАХ Ingosstrakh

PHOENIX EDUCATION

МФТИ Лаборатория Инноватики

ССЫЛКИ

- lambda выражения: Func, Action

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions

- https://www.learncs.org/
- https://www.tutorialsteacher.com/csharp/csharp-tutorials
- https://repl.it/languages/csharp
- Func and `delegate`

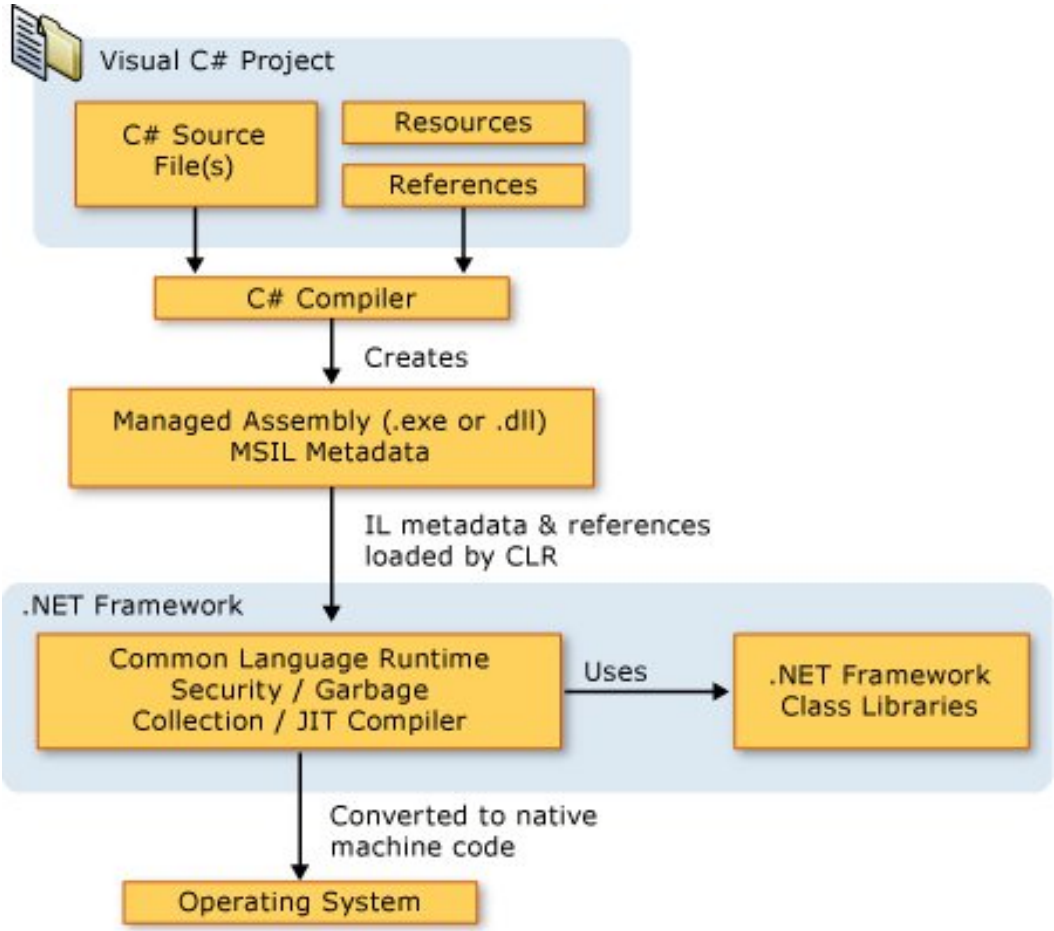
https://www.growingwiththeweb.com/2012/08/func-and-action-basics-in-c.html

- https://github.com/j0k/it_school_weeks

коммуникация

- telegram: @bimodaling
- tele-group: t.me/PIITSchool (Phoenix Ingostrah IT School)
- IRC chat.freenode.net (mIRC, HexChat): #itweeks
- mail: yuri@silkmind.com
- https://github.com/j0k/it_school_weeks

Архитектура платформы .NET Framework



https://docs.microsoft.com/ru-ru/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

план на день

- что было вчера и что узнали
- вопросы дня
- список ресурсов и задач на день
- немного про backend
- code style, немного синтаксиса
- quicksort с Generics и Lambda
- игра с ООП

backend

Front End

- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

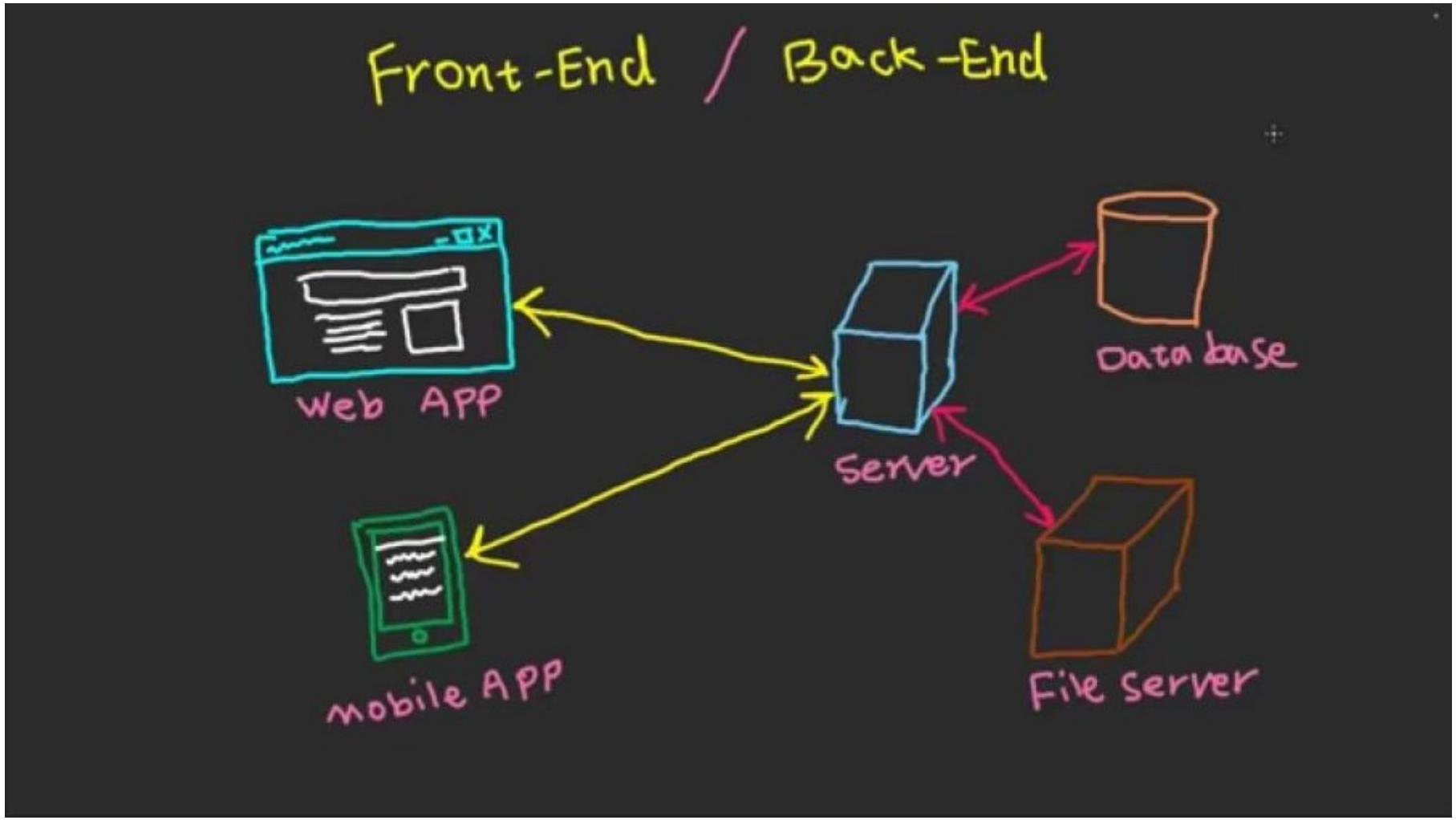
Back End

- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

and C# :)

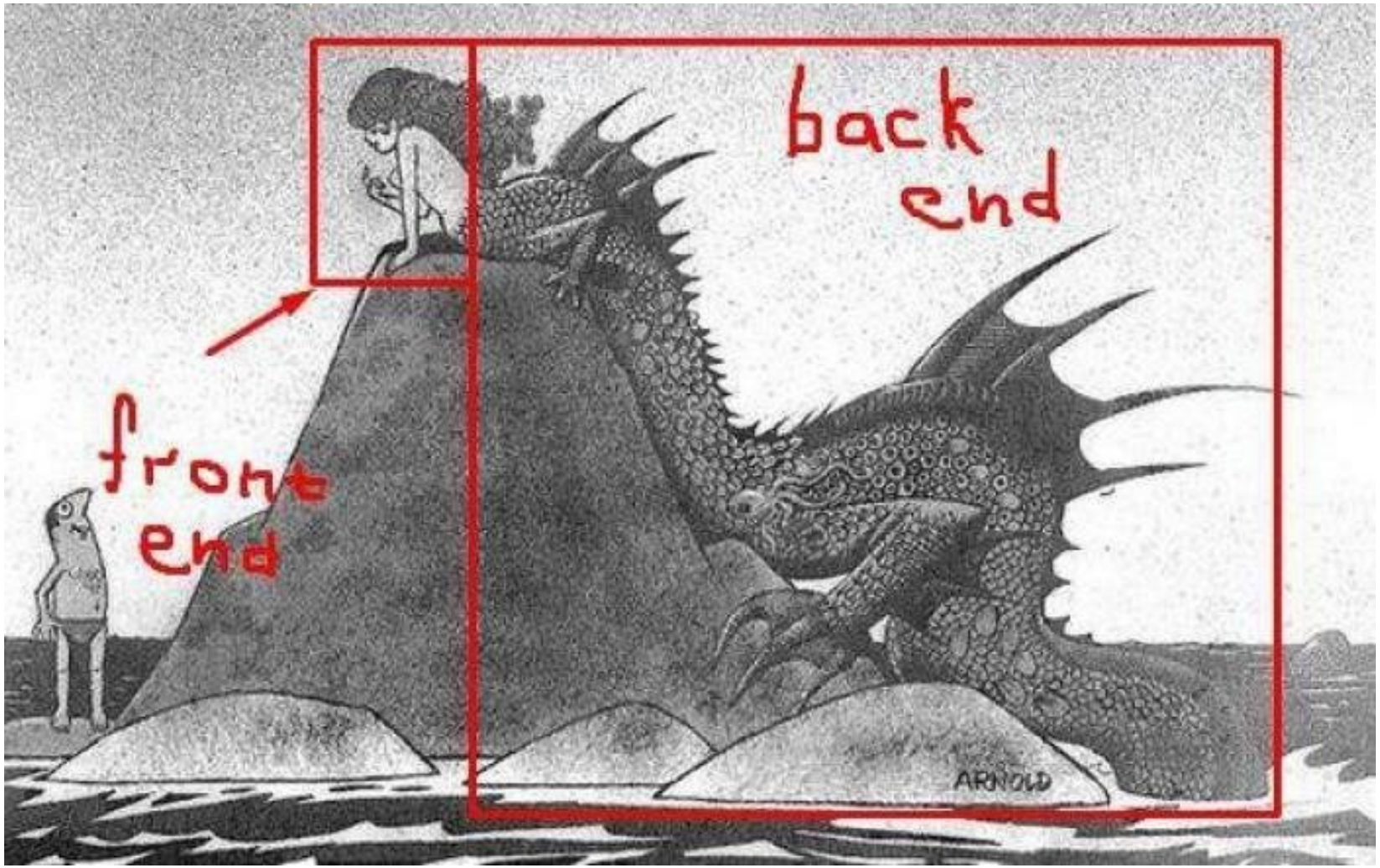
- https://www.quora.com/What-are-front-end-and-back-end-technologies

backend



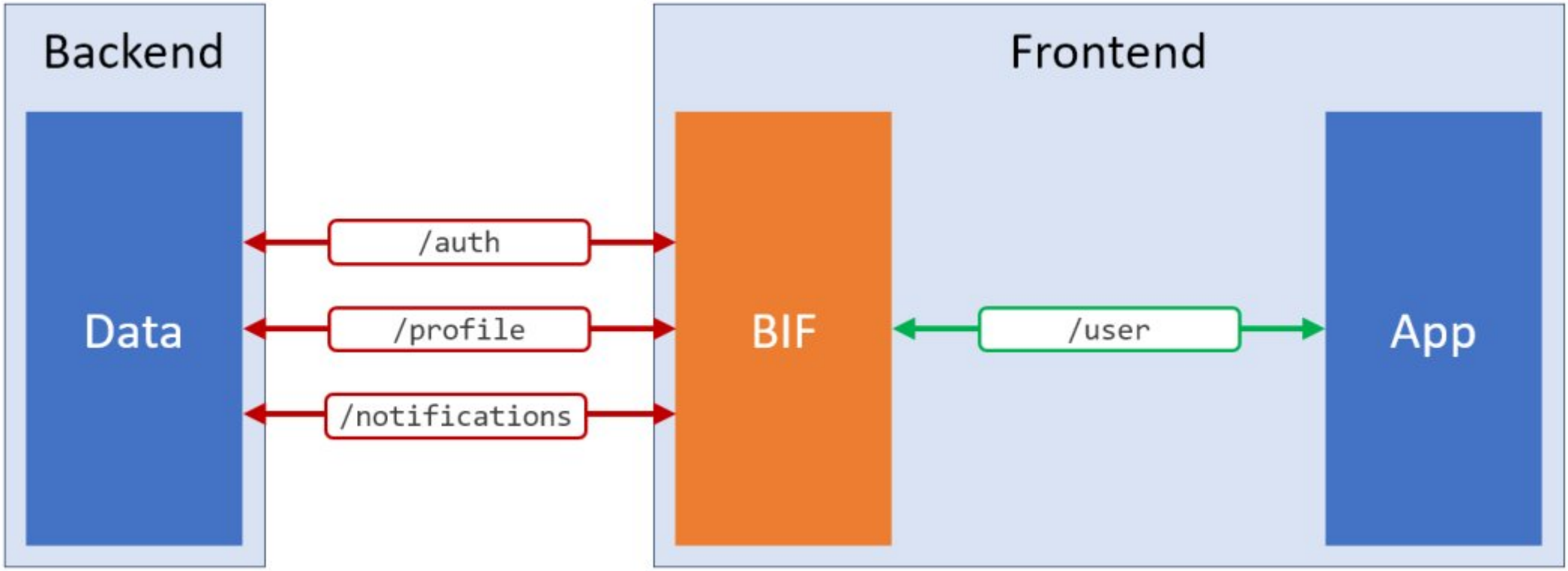
- https://www.youtube.com/watch?v=nMtgFZSdtwk

backend



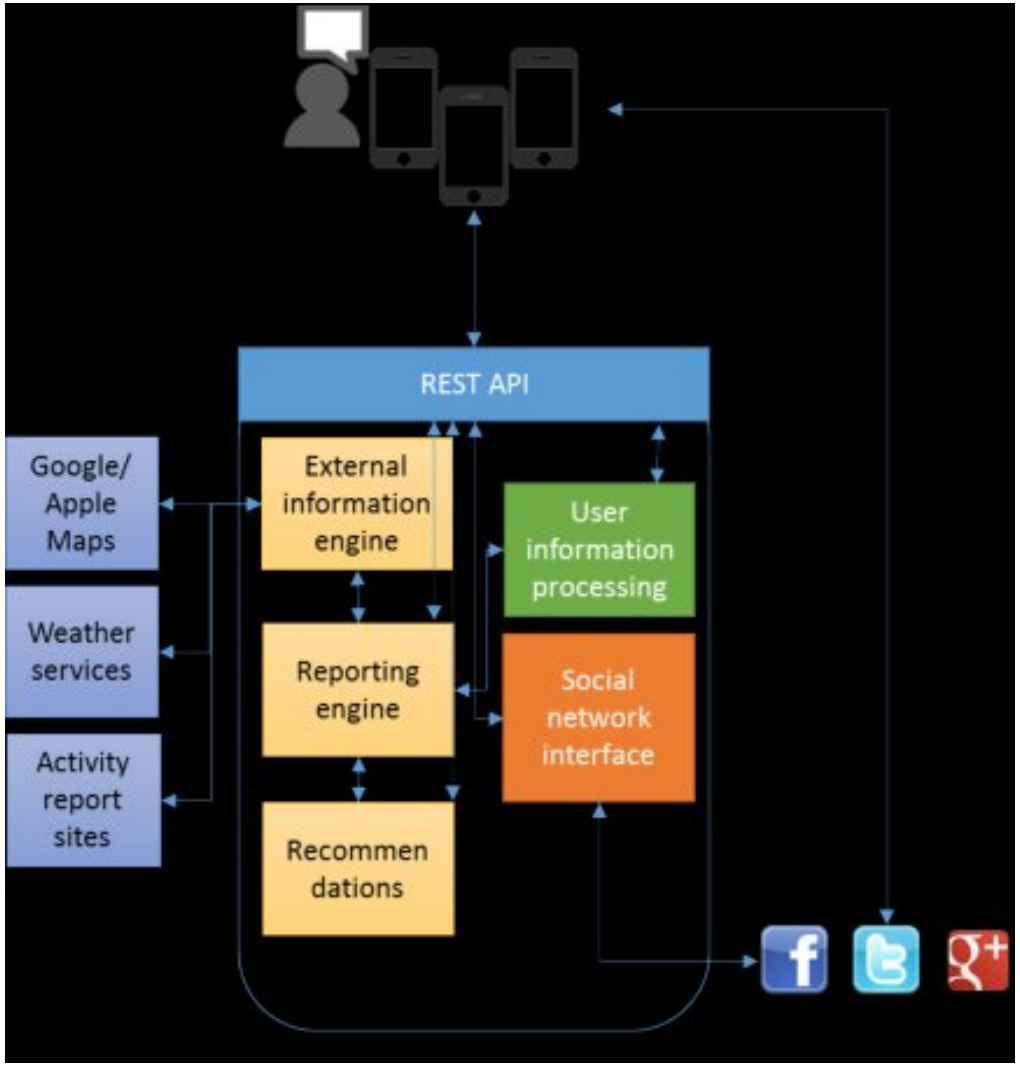
- https://ru.hexlet.io/courses/intro_to_web_development/lessons/backend/theory_unit

backend



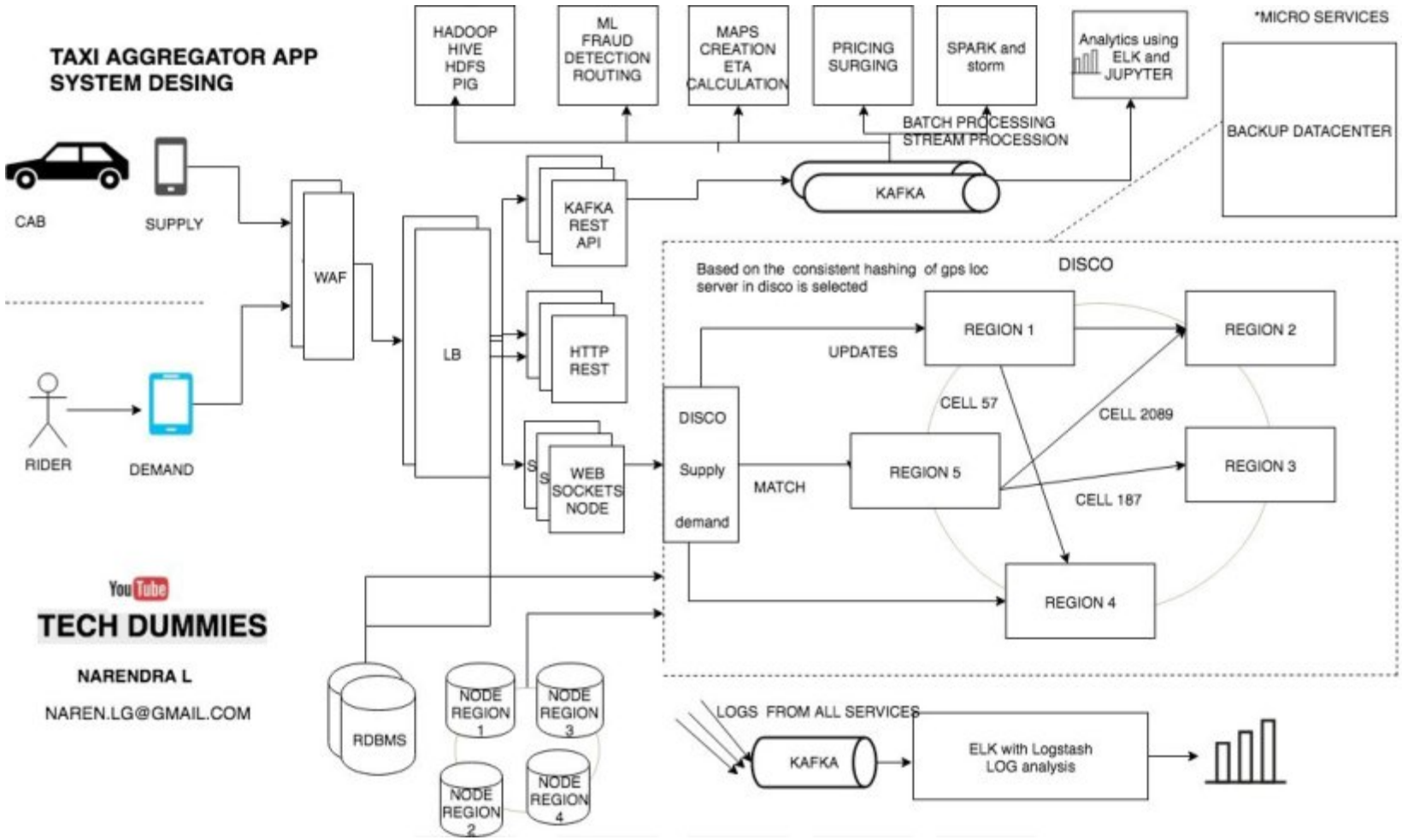
- https://www.quora.com/What-are-front-end-and-back-end-technologies

backend



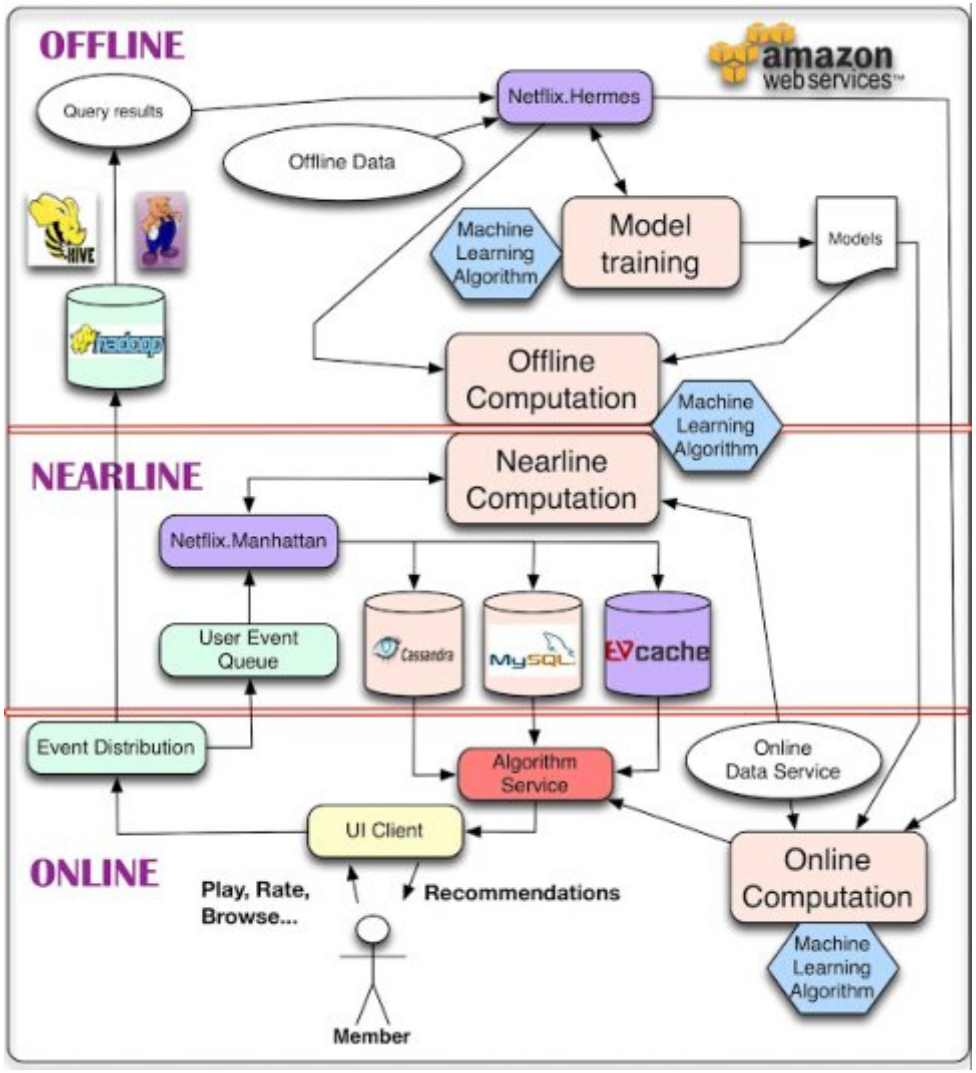
https://www.researchgate.net/figure/General-overview-of-the-SiUinde-backend-architecture-and-associated-services_fig4_282745067

Uber Backend



https://medium.com/@narengowda/uber-system-design-8b2bc95e2cfe

Amazon



https://www.kdnuggets.com/2015/12/xamat-20-lessons-building-machine-learning-systems.html/2

Waves Platform

https://blog.wavesplatform.com/web3-0-the-road-ahead-for-waves-9bd8a51f63ce

EEG Brain data

FRONT

NRBoom mobile app

TuSion Mobile app

games

back office

web page

BACKEND

signal processing

B2B accounts

EEG Data Server

Mobile app accounts

neurocommunication rooms for online sessions

https://silkmind.com/#s_tusion

план на день

- что было вчера и что узнали
- вопросы дня
- список ресурсов и задач на день
- немного про backend
- code style, немного синтаксиса
- quicksort с Generics и Lambda
- игра с ООП

Стандарты и соглашения по именованию в C#

1. Do use PascalCasing for class names and method names:

```
public class ClientActivity
{
  public void ClearStatistics()
  {
    //...
  }
  public void CalculateStatistics()
  {
    //...
  }
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

2. Do use camelCasing for method arguments and local variables:

```
public class UserLog
{
  public void Add(LogEvent logEvent)
  {
    int itemCount = logEvent.Items.Count;
    // ...
  }
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

3. Do not use Hungarian notation or any other type identification in identifiers

```
// Correct
int counter;
string name;
// Avoid
int iCounter;
string strName;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

4. Do not use Screaming Caps for constants or readonly variables:

```
// Correct
public const string ShippingType = "DropShip";
// Avoid
public const string SHIPPINGTYPE = "DropShip";
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

5. Use meaningful names for variables. The following example uses seattleCustomers for customers who are located in Seattle:

```
var seattleCustomers = from customer in customers
  where customer.City == "Seattle"
  select customer.Name;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

6. Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.

```
// Correct
UserGroup userGroup;
Assignment employeeAssignment;
// Avoid
UserGroup usrGrp;
Assignment empAssignment;
// Exceptions
CustomerId customerId;
XmlDocument xmlDocument;
FtpHelper ftpHelper;
UriPart uriPart;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

7. Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase):

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

7. Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase):

```
HtmlHelper htmlHelper;
FtpTransfer ftpTransfer;
UIControl uiControl;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

8. Do not use Underscores in identifiers. Exception: you can prefix private fields with an underscore:

```
// Correct
public DateTime clientAppointment;
public TimeSpan timeLeft;
// Avoid
public DateTime client_Appointment;
public TimeSpan time_Left;
// Exception (Class field)
private DateTime _registrationDate;
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

9. Do use predefined type names (C# aliases) like `int`, `float`, `string` for local, parameter and member declarations. Do use .NET Framework names like `Int32`, `Single`, `String` when accessing the type's static members like `Int32.TryParse` or `String.Join`.

```
// Correct
string firstName;
int lastIndex;
bool isSaved;
string commaSeparatedNames = String.Join(", ", names);
int index = Int32.Parse(input);
// Avoid
String firstName;
Int32 lastIndex;
Boolean isSaved;
string commaSeparatedNames = string.Join(", ", names);
int index = int.Parse(input);
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

10. Do use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.

```
var stream = File.Create(path);
var customers = new Dictionary();
// Exceptions
int index = 100;
string timeSheet;
bool isCompleted;
```

Why: removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

11. Do use noun or noun phrases to name a class.

```
public class Employee
{
}
public class BusinessLocation
{
}
public class DocumentCollection
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

12. Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.

```
public interface IShape
{
}
public interface IShapeCollection
{
}
public interface IGroupable
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

13. Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.

```
// Located in Task.cs
public partial class Task
{
}
// Located in Task.generated.cs
public partial class Task
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

13. Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.

```
// Located in Task.cs
public partial class Task
{
}
// Located in Task.generated.cs
public partial class Task
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

partial class

When we execute the above code, then compiler combines `Geeks1.cs` and `Geeks2.cs` into a single file

Geeks1.cs

```
public partial class Geeks {
    private string Author_name;
    private int Total_articles;

    public Geeks(string a, int t)
    {
        this.Authour_name = a;
        this.Total_articles = t;
    }
}
```

Geeks2.cs

```
public partial class Geeks {
    public void Display()
    {
        Console.WriteLine("Author's name is : " + Author_name);
        Console.WriteLine("Total number articles is : " + Total_articles);
    }
}
```

- https://www.geeksforgeeks.org/partial-classes-in-c-sharp/

Стандарты и соглашения по именованию в C#

14. Do organize namespaces with a clearly defined structure:

```
// Examples
namespace Company.Product.Module.SubModule
{
}
namespace Product.Module.Component
{
}
namespace Product.Layer.Module.Group
{
}
```

Why: consistent with the Microsoft's .NET Framework. Maintains good organization of your code base.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

15. Do vertically align curly brackets:

```
// Correct
class Program
{
  static void Main(string[] args)
  {
    //...
  }
}
```

Why: consistent with the Microsoft's .NET Framework. Maintains good organization of your code base.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

16. Do declare all member variables at the top of a class, with static variables at the very top.

```
// Correct
public class Account
{
  public static string BankName;
  public static decimal Reserves;
  public string Number { get; set; }
  public DateTime DateOpened { get; set; }
  public DateTime DateClosed { get; set; }
  public decimal Balance { get; set; }
  // Constructor
  public Account()
  {
    // ...
  }
}
```

Why: generally accepted practice that prevents the need to hunt for variable declarations.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Свойства

```
using System;

class TimePeriod
{
   private double _seconds;

   public double Hours
   {
       get { return _seconds / 3600; }
       set {
          if (value < 0 || value > 24)
             throw new ArgumentOutOfRangeException(
                   $"{nameof(value)} must be between 0 and 24.");

          _seconds = value * 3600;
       }
   }
}

class Program
{
   static void Main()
   {
       TimePeriod t = new TimePeriod();
       // The property assignment causes the 'set' accessor to be called.
       t.Hours = 24;

       // Retrieving the property causes the 'get' accessor to be called.
       Console.WriteLine($"Time in hours: {t.Hours}");
   }
}
// The example displays the following output:
//    Time in hours: 24
```

Свойство — это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля. Свойства можно использовать, как если бы они были членами общих данных, но фактически они представляют собой специальные методы, называемые методами доступа. Это позволяет легко получать доступ к данным и помогает повысить безопасность и гибкость методов

- https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/properties

Стандарты и соглашения по именованию в C#

17. Do use singular names for enums. Exception: bit field enums.

```
// Correct
public enum Color
{
  Red,
  Green,
  Blue,
  Yellow,
  Magenta,
  Cyan
}
// Exception
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

18. Do not explicitly specify a type of an enum or values of enums (except bit fields):

```
// Don't
public enum Direction : long
{
  North = 1,
  East = 2,
  South = 3,
  West = 4
}
// Correct
public enum Direction
{
  North,
  East,
  South,
  West
}
```

Why: can create confusion when relying on actual types and values.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

19. Do not use an "Enum" suffix in enum type names:

```
// Don't
public enum CoinEnum
{
  Penny,
  Nickel,
  Dime,
  Quarter,
  Dollar
}
// Correct
public enum Coin
{
  Penny,
  Nickel,
  Dime,
  Quarter,
  Dollar
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

20. Do not use "Flag" or "Flags" suffixes in enum type names:

```
// Don't
[Flags]
public enum DockingsFlags
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
// Correct
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
```

consistent with prior rule of no type indicators in identifiers.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

20. Do not use "Flag" or "Flags" suffixes in enum type names:

```
// Don't
[Flags]
public enum DockingsFlags
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
// Correct
[Flags]
public enum Dockings
{
  None = 0,
  Top = 1,
  Right = 2,
  Bottom = 4,
  Left = 8
}
```

consistent with prior rule of no type indicators in identifiers.

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

21. Do use suffix EventArgs at creation of the new classes comprising the information on event:

```
// Correct
public class BarcodeReadEventArgs : System.EventArgs
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

22. Do name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:

```
public delegate void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e);
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

23. Do not create names of parameters in methods (or constructors) which differ only by the register:

```
// Avoid
private void MyFunction(string name, string Name)
{
  //...
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

24. DO use two parameters named sender and e in event handlers. The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.

```
public void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e)
{
  //...
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

Стандарты и соглашения по именованию в C#

25. Do use suffix Exception at creation of the new classes comprising the information on exception:

```
// Correct
public class BarcodeReadException : System.Exception
{
}
```

- https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md

план на день

- что было вчера и что узнали
- вопросы дня
- список ресурсов и задач на день
- немного про backend
- code style, немного синтаксиса
- quicksort с Generics и Lambda
- ООП II

ref

```
void Method(ref int refArgument)
{
    refArgument = refArgument + 44;
}

int number = 1;
Method(ref number);
Console.WriteLine(number);
// Output: 45
```

out parameter modifier

```
int initializeInMethod;
OutArgExample(out initializeInMethod);
Console.WriteLine(initializeInMethod);     // value is now 44

void OutArgExample(out int number)
{
    number = 44;
}
```

out parameter modifier

```
class CS0663_Example
{
    // Compiler error CS0663: "Cannot define overloaded
    // methods that differ only on ref and out".
    public void SampleMethod(out int i) { }
    public void SampleMethod(ref int i) { }
}
```

```
class OutOverloadExample
{
    public void SampleMethod(int i) { }
    public void SampleMethod(out int i) => i = 5;
}
```

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/out-parameter-modifier

Func

```
int mul1(int x, int y)
{
    return x * y;
}
int mul1dbg(int x, int y)
{
    int xy = x * y;
    Console.WriteLine("{0} x {1} = {2}", x, y, xy);
    return xy;
}

Func<int, int, int> mul2 = (x, y) => x * y;
Func<int, int, int> mul3 = (x, y) =>mul1dbg(x,y);
int r1 = mul1(2, 2); // =4
int r2 = mul2(2, 2); // =4
int r3 = mul3(2, 2); // =4
```

https://docs.microsoft.com/ru-ru/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Func, примеры

Defining a **Func**

You can assign a **Func<>** type by providing a lambda expression, which is C#'s version of anonymous functions. Here is the most simple example of a lambda expression, returning the only parameter on the **Func<>**.

```
x => x
```

```
Func<object, int, string> function;
```

https://docs.microsoft.com/ru-ru/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Example: Generic class

```
class MyGenericClass<T>
{
    private T genericMemberVariable;

    public MyGenericClass(T value)
    {
        genericMemberVariable = value;
    }

    public T genericMethod(T genericParameter)
    {
        Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(),genericParameter);
        Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), genericMemberVariable);

        return genericMemberVariable;
    }

    public T genericProperty { get; set; }
}
```

https://www.tutorialsteacher.com/csharp/csharp-generics

Generics

Example: Instantiate Generic Class

```
MyGenericClass<int> intGenericClass = new MyGenericClass<int>(10);

int val = intGenericClass.genericMethod(200);
```

```
Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(),genericParameter);
Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), genericMemberVariable);
```

```
Parameter type: System.Int32, value: 200
Return type: System.Int32, value: 10
```

https://www.tutorialsteacher.com/csharp/csharp-generics

Реализуем алгоритм быстрой сортировки с использованием Generics

Google Scholar quicksort

Articles About 29,100 results (0.08 sec)

Any time
Since 2019
Since 2018
Since 2015
Custom range...

Quicksort [PDF] oup.com
CAR Hoare - The Computer Journal, 1962 - academic.oup.com
A description is given of a new method of sorting in the random-access store of a computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, which may be …
Cited by 1431 Related articles All 7 versions

By C. A. R. Hoare

A description is given of a new method of sorting in the random-access store of a computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, which may be useful in the optimization of inner loops, are described in the second part of the paper.

вчера закончили на :
https://github.com/j0k/it_school_weeks/tree/master/week3/quicksort

```
public static Random random = new Random();
2 references
private static int pivot(List<T> a)
{
    return random.Next(a.Count);
}
```

```
public static List<T> quickSort(List<T> a, Func<T, T, int> cmp)
{
    // cmp - comparator
    if (a.Count <= 1)
        return a;

    int ind = pivot(a);
    T elem = a[ind];

    List<T> left   = new List<T>();// < elem
    List<T> center = new List<T>();// = elem
    List<T> right  = new List<T>();// > elem
    for (int i=0; i < a.Count; i++)
    {
        T e = a[i];

        int r = cmp(e, elem); // result

        if (r == 0)
            center.Add(e);
        else if (r < 0)
            left.Add(e);
        else
            right.Add(e);
    }

    List<T> res = new List<T>();

    res.AddRange(quickSort(left, cmp));
    res.AddRange(center);
    res.AddRange(quickSort(right, cmp));
    return res;
}
```

.Sort

```
public class DoubleCmp : IComparer<double>
{
    0 references
    int IComparer<double>.Compare(double x, double y)
    {
        return x.CompareTo(y);
    }
}
```

```
List<double> A = new List<double>(){100.1, 1.23, -100, 3, 5};
A.Sort(new DoubleCmp());
// A.Sort(); also OK for that simple case

Console.WriteLine(string.Join(" ", A));
```

https://docs.microsoft.com/ru-ru/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

ВСЕГДА БРОСАЙТЕ ВЫЗОВ

старым методам.

ВСЕГДА ПРОВЕРЯЙТЕ

старые методы.

- Ховард Шульц (Sturbucks)

ALWAYS CHALLENGE the *old* ways.

- Howard Schultz, Starbucks

план на день

- что было вчера и что узнали
- вопросы дня
- список ресурсов и задач на день
- немного про backend
- code style, немного синтаксиса
- quicksort с Generics и Lambda
- ООП II

ООП II

- наследование
- полиморфизм
- инкапсуляция (модификаторы доступа и область видимости)
- абстрагирование

ООП II: инкапсуляция

Access Modifiers

public

The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private

The type or member can be accessed only by code in the same class or struct.

protected

The type or member can be accessed only by code in the same class, or in a class that is derived from that class.

ООП II: инкапсуляция

```
Этимология
Происходит от лат. incapsulatio, далее из лат. in- «в» + лат. capsula «ящичек, ларчик, шкатулка»,
уменьш. от capsa «вместилище, футляр, ящик» из capere «брать; получать»
(восходит к праиндоевр. *kap- «хватать»)..
```

ин-кап-су-ли́-ро-вать

Значение [править]

1. помещать в оболочку, капсулу ◆ Документы архивного хранения, единичные листы рукописей **инкапсулируют** — заключают в «капсулы» из прозрачной светостойкой прочной и долговечной плёнки. *С. А. Добрусина, Е. С. Чернина, «Научные основы консервации документов», 1993 г.* (цитата из библиотеки Google Книги)
2. *прогр.* использовать инкапсуляцию — сокрытие внутренней структуры данных и реализации методов объекта от остальной программы ◆ Мы видим три чётко выделенных уровня, каждый из которых **инкапсулирует** некоторые аспекты поведения программы и скрывает детали реализации от других уровней. *Брюс Е. Крелль, «Pocket PC. Руководство разработчика», 2010 г.* (цитата из библиотеки Google Книги)

- https://ru.wiktionary.org/wiki/%D0%B8%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F

ООП II: инкапсуляция

public
The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private
The type or member can be accessed only by code in the same class or struct.

protected
The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
internal
The type or member can be accessed by any code in the same assembly, but not from another assembly.

protected internal The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly.

private protected The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

ООП II: наследование

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

```
class Employee : Person
{

}
```

ООП II: полиморфизм

много форм

Виртуальный метод – это метод, который МОЖЕТ быть переопределен в классе-наследнике. Такой метод может иметь стандартную реализацию в базовом классе.

Абстрактный метод – это метод, который ДОЛЖЕН быть реализован в классе-наследнике. При этом, абстрактный метод не может иметь своей реализации в базовом классе (тело пустое), в отличии от виртуального.

Переопределение метода – это изменение реализации метода, установленного как виртуальный (в классе наследнике метод будет работать отлично от базового класса).

- http://mycsharp.ru/post/32/2013_08_27_polimorfizm_v_si-sharp_chto_eto_takoe_.html

ООП II: полиморфизм

много форм

Виртуальный метод – это метод, который МОЖЕТ быть переопределен в классе-наследнике. Такой метод может иметь стандартную реализацию в базовом классе.

Абстрактный метод – это метод, который ДОЛЖЕН быть реализован в классе-наследнике. При этом, абстрактный метод не может иметь своей реализации в базовом классе (тело пустое), в отличии от виртуального.

Переопределение метода – это изменение реализации метода, установленного как виртуальный (в классе наследнике метод будет работать отлично от базового класса).

- http://mycsharp.ru/post/32/2013_08_27_polimorfizm_v_si-sharp_chto_eto_takoe_.html

ООП II: полиморфизм

```
class MyBaseClass
{
    // virtual auto-implemented property. Overrides can only
    // provide specialized behavior if they implement get and set accessors.
    public virtual string Name { get; set; }

    // ordinary virtual property with backing field
    private int num;
    public virtual int Number
    {
        get { return num; }
        set { num = value; }
    }
}
```

- http://mycsharp.ru/post/32/2013_08_27_polimorfizm_v_si-sharp_chto_eto_takoe_.html

ООП II: полиморфизм

```
class MyBaseClass
{
    // virtual auto-implemented property. Overrides can only
    // provide specialized behavior if they implement get and set accessors.
    public virtual string Name { get; set; }

    // ordinary virtual property with backing field
    private int num;
    public virtual int Number
    {
        get { return num; }
        set { num = value; }
    }
}
```

- http://mycsharp.ru/post/32/2013_08_27_polimorfizm_v_si-sharp_chto_eto_takoe_.html

ООП II: полиморфизм

```
class MyDerivedClass : MyBaseClass
{
    private string name;

   // Override auto-implemented property with ordinary property
   // to provide specialized accessor behavior.
    public override string Name
    {
        get
        {
            return name;
        }
        set
        {
            if (!string.IsNullOrEmpty(value))
            {
                name = value;
            }
            else
            {
                name = "Unknown";
            }
        }
    }
}
```

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual