

Упражнение №1 по ПС

C#

Среда за разработка Visual Studio.

Разлики между C++ и C#.

Целта на това упражнение

Запознаване с езика C# и средата, на която основно се използва – Visual Studio. Тях ще използваме на всички следващи упражнения.

Задачите в упражнението изграждат:

Малка студентска информационна система

В това упражнение:

Конзолно приложение илюстриращо MVVM структурата.

- Изброен тип, който да описва всички типове потребители
- Клас User, който представлява Модела на данните за всеки потребител.
- Клас UserView, който се използва за начина на репрезентиране на данните за потребителя.
- Клас UserViewModel, който отговаря за прехвърлянето на данните от Model-а, към View-то.

В края на упражнението:

Ще визуализираме информация относно потребител на системата посредством MVVM.

За домашно:

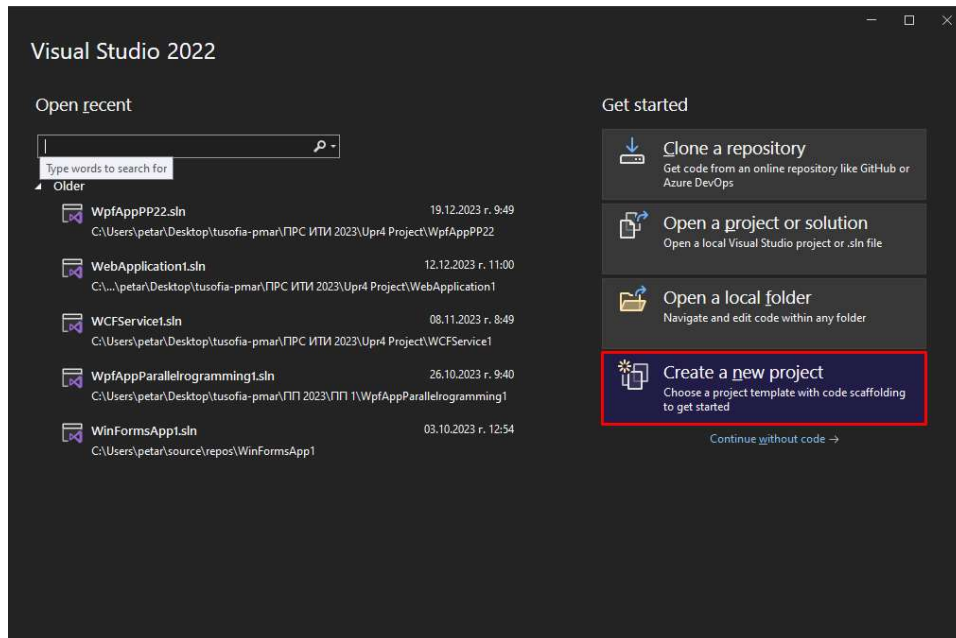
Да се промени кода така, че:

1. Да се добавят още данни за потребителите (пример: Факултетен номер, имейл)
2. Да се добавят различни визуализации на данните на потребителя.
3. Да се добави прост начин за криптиране/декриптиране на паролата в getter/setter на Password

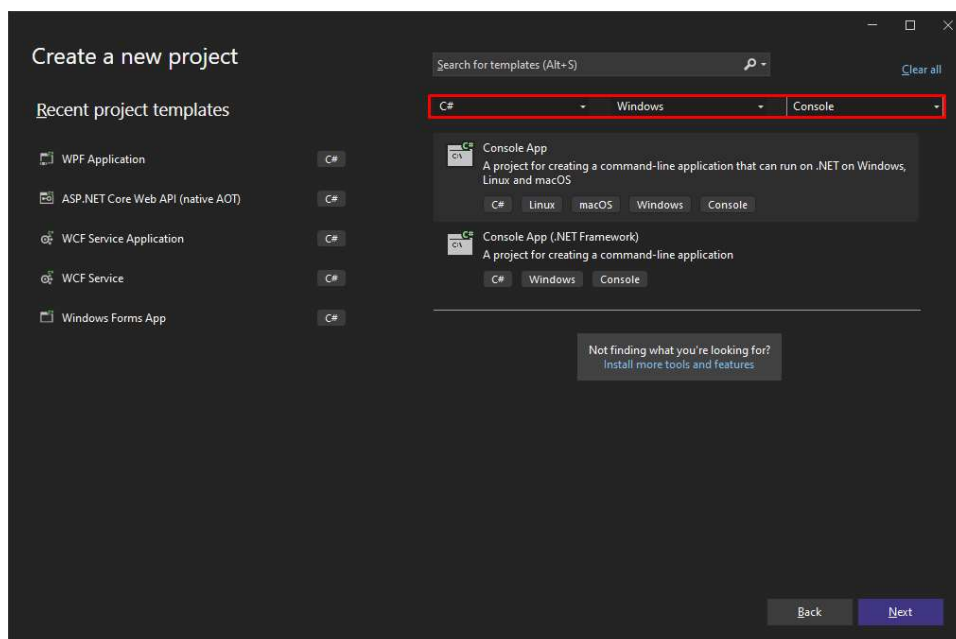
Важни знания от упражнението: Namespaces, Enums, MVVM, Property, Constructors, Class, Static Class, Method, Static Method

Създаване на проект

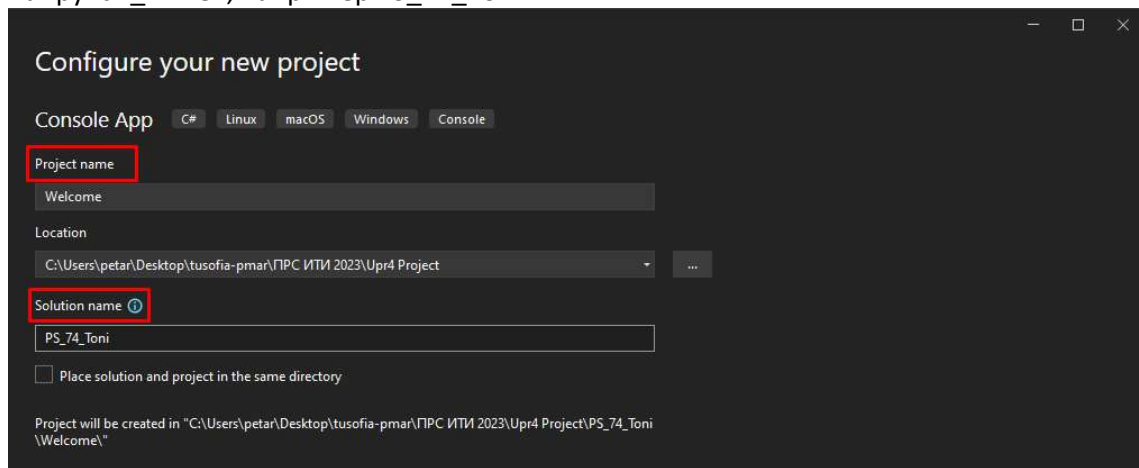
1. Отворете **Visual Studio 2022** (или по-ново)
2. От началния екран изберете **“Create a new project”**



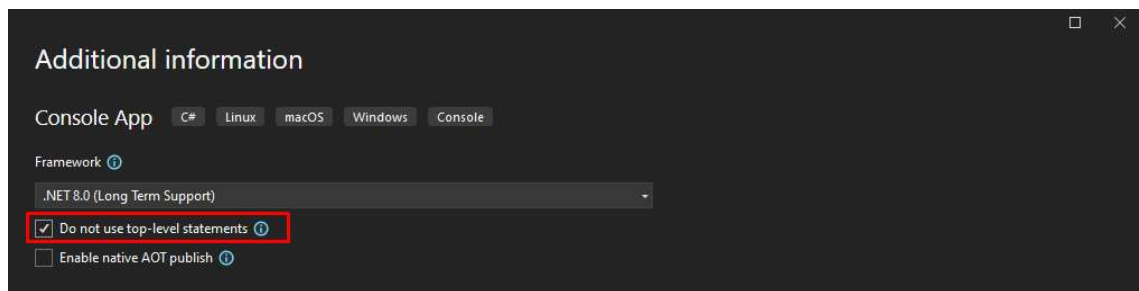
3. В диалоговия прозорец изберете **“C#”** → **“Windows”** → **“Console”** и изберете **“Console App”**



4. Кръстете вашия проект **Welcome**, но след това задайте име на **Solution**: PS_<No на група>_<име>, например PS_74_Toni.



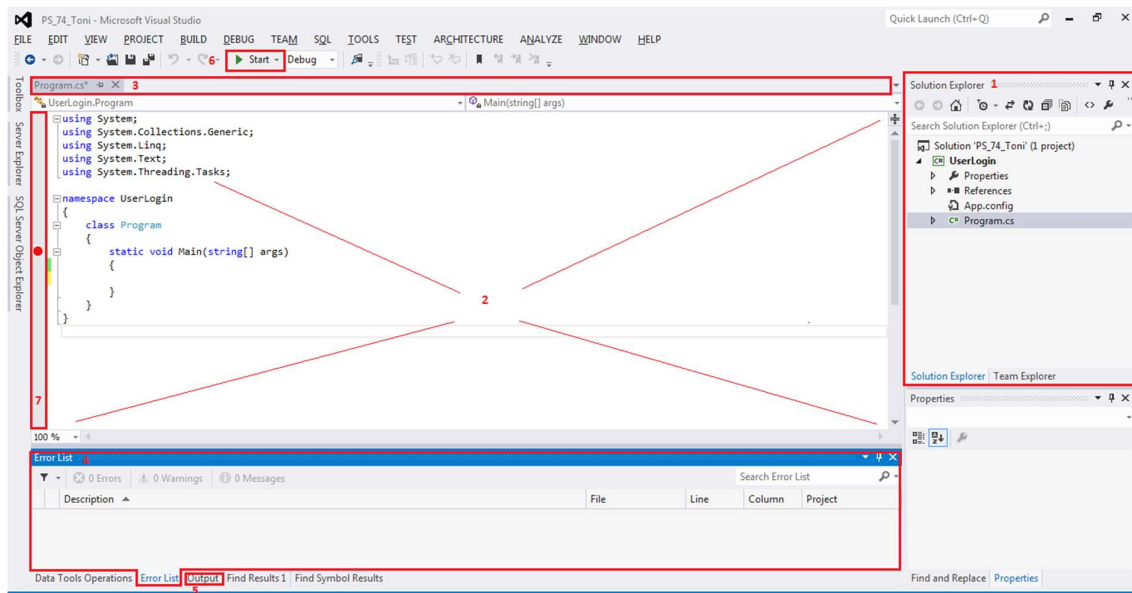
5. Изберете **“Do not use top-level statements”**, (това ще ни позволи по-лесно да се запознаем с *namespaces*).



Кратко обяснение:

След като е отворен един проект вашето Visual Studio 2022 трябва да има следните елементи:

1. **Solution Explorer** – списък на всички проекти и всички файлове на всеки проект. (Ако не се вижда: “View” → “Solution Explorer”)
2. Екран за редакция на отворен файл.
3. Всички отворени файлове.
4. **Error List** – списък на всички установени в кода грешки. (Ако не се вижда: “View” → “Error List”)
5. **Output** – стандартния изход на приложението. (Ако не се вижда: “View” → “Output”)
6. Бутон за стартиране на приложението. (Ако не се вижда: “View” → “Toolbars” “Standard”)
7. Лента за поставяне на Breakpoint за Debug.



Създаване на папки

1. От **Solution Explorer** (позиция.1) избираме проекта **Welcome**
2. Кликаме с десен бутон върху **Welcome**, след което избираме **Add → New Folder**
3. Създаваме следните папки: **Model, View, ViewModel, Others**

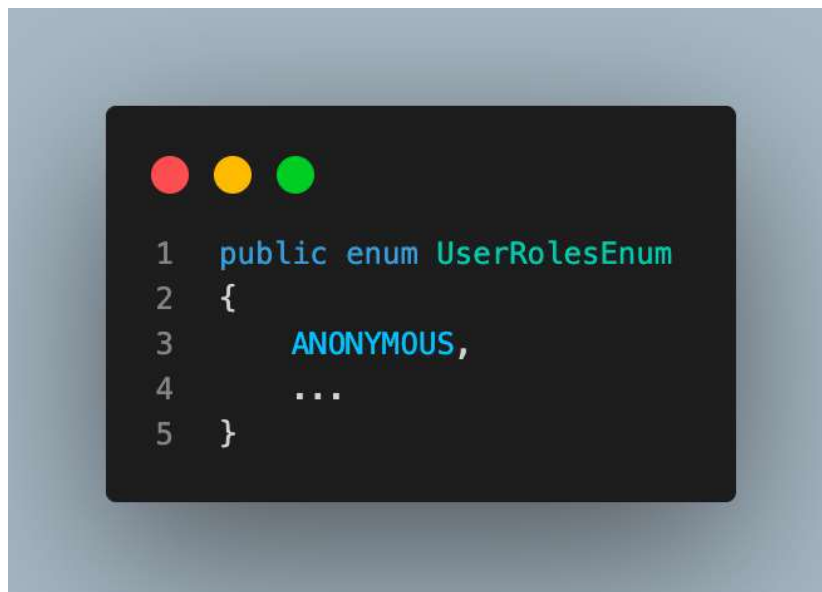
Създаването на папките има за цел, да ни помогне да разделим кода функционално, запознаване със структурата на MVVM, както и работа с namespaces.

Създаване на избран тип

1. Кликнете с десен бутон мишката на папката **Others**
2. От контекстното меню изберете **Add → New Item...**
3. От новоотвореният прозорец изберете **Code File**
4. В полето **Name**, въведете за име **UserRolesEnum.cs**
5. В ново създаденият файл добавяме първо, че ще работим в контекста на namespace **Welcome.Others**



6. Вътре в блока на namespace-а, добавете избран тип **UserRolesEnum** със следните стойности: **ANONYMOUS, ADMIN, INSPECTOR, PROFESSOR, STUDENT**



Създаване на Клас User

По аналог на горе-направеният изброен тип, създаваме клас **User**, в папката **Model**. За целта, обаче избираме **Class**, вместо **Code File**.

1. След като файлът е създаден започваме да добавяме неговите свойства. А именно **Names, Password, Role**. **Names** и **Password** са от тип **string**, а **Role** е от типа на създаденият по-горе изброен тип **UserRolesEnum**.

prop [Tab] [Tab] – може да се използва като темплейт за бързо създаване на свойство

2. Тъй като **UserRolesEnum** е в различен namespace, е възможно да се наложи да добавим **using** който да използва namespace-а му.

Свойство е:


- Член на клас
- Дава гъвкав достъп до *private* поле
- Използва се като публично поле
- Реализира се чрез функции (*get* и *set*)

Т.е. свойството изглежда като поле, но всъщност е функция (две функции).



Употребата на полето в случая е същата като при употребата на обикновена променлива.

Пример за класическа реализация:




```

1  private string _names;
2
3  public string GetNames() {
4      return _names;
5  }
6
7  public void SetNames(string value) {
8      _names = value;
9  }

```

За аналог пълният код на едно свойство би изглеждал по следния начин:




```

1  private string _names;
2
3  public string Names {
4      get { return _names; }
5      set { _names = value; }
6  }

```

В блока на get и set, могат да се извършват и трансформации или калкулации, ето и пример:



```


1  set {
2      var namesToUpperCase = value.ToUpper();
3      _names = namesToUpperCase;
4  }

```


Създаване на клас `UserViewModel`

Следвайки същите стъпки като до сега създаваме class **`UserViewModel`**, в папка **`ViewModel`**.

1. В блока на класа, създаваме частно поле **`_user`**, което трябва да е от типа **`User`**, създаден в предходната част.
2. Създаваме конструктор, в C# конструктора трябва да е с името на самият клас, за разлика от някои други езици. Като входящ параметър на конструктора, слагаме **`user`** от типа **`User`**, а вътре в блока на конструктора добавяме присвояване на **`user`**, върху частното поле **`_user`**.
3. Добавяме свойства за всяко едно свойство на **`User`** обекта. Пример:



```
1 public string Name
2 {
3     get { return _user.Name; }
4     set { _user.Name = value; }
5 }
```

Създаване на клас `UserView`

Отново по аналогичен начин създаваме клас **`UserView`**, в папката **`View`**.

1. Създаваме частно поле **`_viewModel`** от типа **`UserViewModel`**
2. Създаваме конструктор който приема като параметър **`viewModel`** от типа **`UserViewModel`** и отново в блока на конструктора добавяме присвояване на входящият параметър към частното поле.
3. Създаваме публичен метод **`Display`**, който ще принтира в конзолата следният текст:
`Welcome`
`User: {Имена на потребителя}`
`Role: {Типа на потребителя }`

Навързване на кода

Във файла Program.cs, в метода Main добавете следните неща:

1. Създайте обект от типа User, като му зададете Всички необходими параметри
2. Създайте обект от типа UserViewModel, като му подадете новосъздаденият обект от предходната точка
3. Създайте обект от типа UserView, като му подадете обекта от т. 2.
4. Извикайте Display метода на UserView обекта.
5. В случай, че приложението се затваря след стартиране, добавете Console.ReadKey(); на края на метода Main.