PV179 – Ako písať pekný, udržovateľný a testovateľný kód?

Martin Macák

12. 10. 2021

Fakulta informatiky, Masarykova univerzita, Brno

Osnova prednášky

1. Reklama
2. Warmup
3. Query
4. Princípy písania kódu
5. Zvyšok architektúry

Reklama

- Najväčšia česká .NET konferencia **Update Conference 2021**
- 18 - 19 November 2021
- https://www.updateconference.net/en/2021/schedule
- .NET, Web, AI/ML, Architecture, Cloud, Database, Future & Inspiration, IoT, Mobile Development, Security, . . .

Warmup

V minulej časti ste videli...

- Repository
  - Create, Read, Update, Delete
- Query
  - dotazovanie
  - napr:
    SELECT TOP 10 * FROM Customers
    WHERE Country='USA' AND Age>=18
    ORDER BY Name DESC
- Unit of Work
  - sleduje zmeny v entitách v priebehu danej transakcie

Query

- Reprezentuje dotazovanie.
- Chceme ho nezávislé na ORM frameworku
  - Použité ORM v DAL nemusí vyhovovať po celú dobu.
  - Najčastejšie sa mení kvôli výkonu.
- Naša reprezentácia dotazu by mala podporovať:
  - komplexné WHERE filtrovanie,
  - radenie,
  - stránkovanie.

Ako implementovať Query?

- Expression trees
- Zliepanie SQL dotazov
- Rozhranie pre konkrétne situácie
- ...

Maj sa, DAL!

Pravidlá písania kódu

- DRY
- KISS
- SOLID
- ...

Pravidlá pre písanie kódu

- DRY – Don't Repeat Yourself
- KISS – Keep It Simple, Silly
- SOLID
- ...

SOLID: Single responsibility

- The Single Responsibility Principle states that every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.

SOLID: Single responsibility

- Chceme vysokú súdržnosť zodpovedností jedného modulu.
- Chceme málo väzieb závislostí medzi modulmi.

(high cohesion, low coupling)

- **N** zodpovedností = **N** dôvodov zmeny
- Čím viac modulov zmena ovplyvňuje, tým väčšia šanca chyby.

SOLID: Open / Closed

- The Open / Closed Principle states that software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

SOLID: Open / Closed

- chceme jednoducho pridávať novú funkcionalitu
- nechcem sa hrabať vo "vnútornostiach"
- rozhrania! :)

SOLID: Liskov Substitution

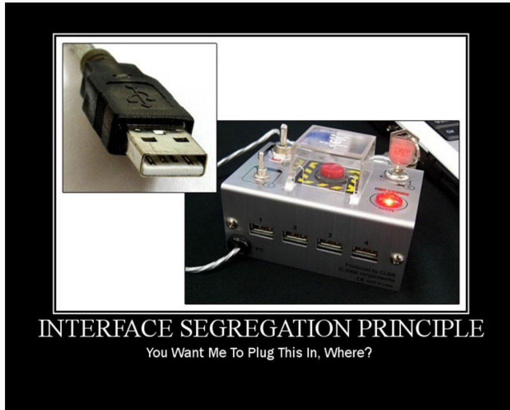
- The Liskov Substitution Principle states that if S is a subtype of T, then objects of type T may be replaced with objects of type S.

SOLID: Liskov Substitution

- potomkovia by nemali meniť chovanie ani porušovať invarianty rodičovskej triedy
- chovania: kváka to, pláva to, žere to chleba
- invarianty: po plávaní sa usuší, keď ju pohladím, usmeje sa

SOLID: Interface Segregation

- The Interface Segregation Principle states that no client should be forced to depend on methods it does not use.



SOLID: Interface Segregation

- rozhranie sú v tomto prípade myslené aj public metódy
- chceme zmenšiť počet závislostí menšími rozhraniami

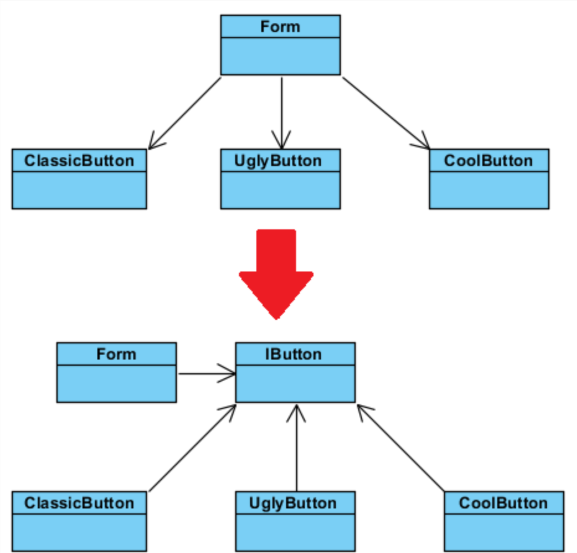
SOLID: Dependency Inversion

- The Dependency Inversion Principle states that high-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

SOLID: Dependency Inversion

- high-level trieda je istým spôsobom nadradená
- nechceme vo vnútri takejto triedy volať *new* a starať sa o životnosť low-level objektov
- tieto objekty by mala dostať zvonku

Dependency Inversion



Inversion of Control

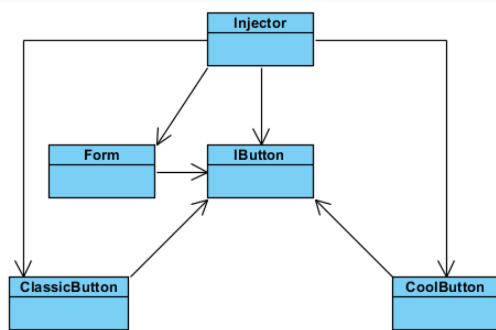
- **"Pattern"**, ktorý sa dá použiť na implementáciu Dependency Inversion
- Rieši **AKO** predať inštanciu nižšieho modulu do vyššieho
- Interface Inversion, Flow Inversion, **Creation Inversion**

Creation Inversion

- Factory
- Service Locator
- **Dependency Injection**
- ...

Dependency Injection

- Zoberiem všetky závislosti modulu, vytvorím konkrétne objekty a zviažem ich k inštancií modulu zvonku



Dependency Injection

- Constructor Injection

```
IButton button = new UglyButton();
var form = new Form(button);
```

```
public class Form
{
    private readonly IButton button;

    public Form(IButton button)
    {
        this.button = button;
    }
}
```

- Setter Injection

```
IButton button = new UglyButton();
var form = new Form {Button = button};
```

```
public class Form
{
    public IButton Button { get; set; }
}
```

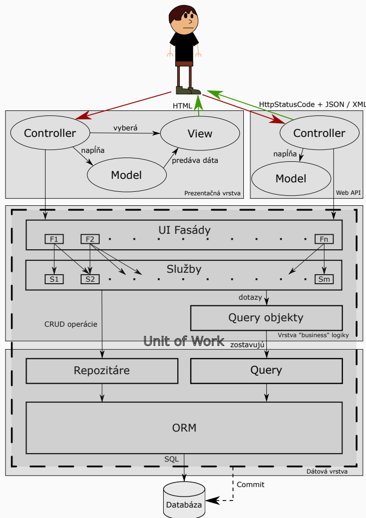
Dependency Injection Container

- Jednoduchá realizácia DI
- Umožňuje konfigurovať závislosti pre daný modul
- Následne ich dokáže riešiť (rekurzívne) podľa toho, ako je kontainer nakonfigurovaný
- Castle Windsor, Autofac, Unity, ...

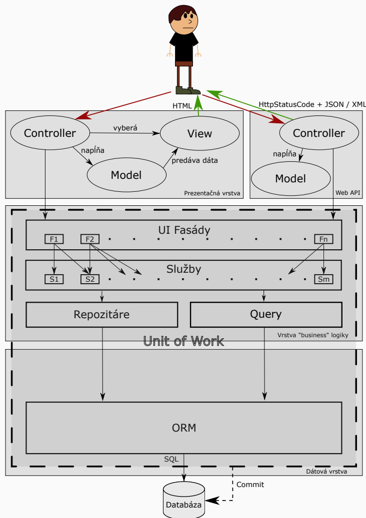
Architektúra aplikácie

- Na základe požiadaviek vyberieme princípy, podľa ktorých zostavíme architektúru aplikácie.
- V projektoch môžu byť rôzne architektúry.

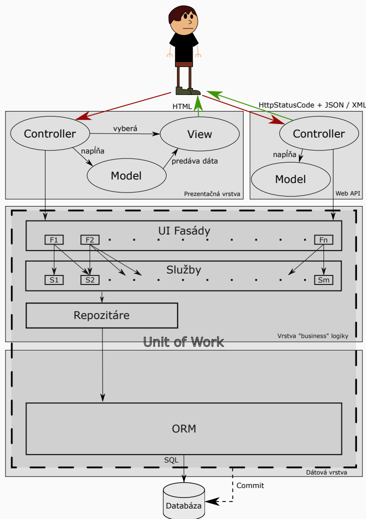
Ukážka A: Naša architektúra



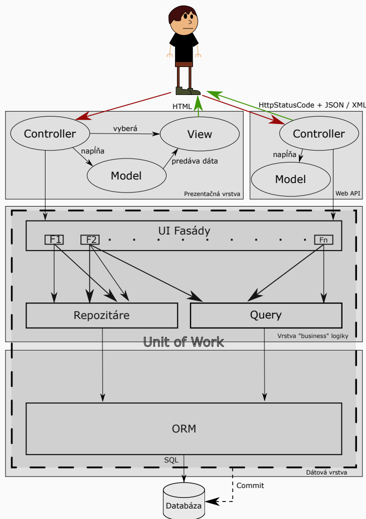
Ukážka B



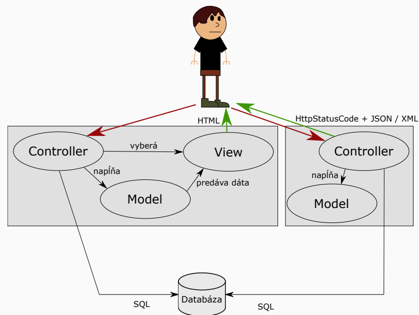
Ukážka C



Ukážka D



Ukážka E



Ukážka F

Clean architecture

Rekapitulácia

1. Query
2. princípy DRY, KISS, SOLID
   - Single Responsibility
   - Open / Closed
   - Liskov Substitution
   - Interface Segregation
   - Dependency Inversion
3. zvyšok architektúry

   architektúra BL:
   - DTOs
   - Query objekty
   - Služby
   - Fasády

macak@mail.muni.cz