

C# 3

Zdolej pokročilé techniky programování

Martin Černil, Václav Kučera

28.9. 2024



Použití materiálů

Toto dílo je licencováno pod

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International
License



C#3

Co nás čeká a jak to bude probíhat 😊

OSNOVA

- 25. 9. Organizace, Visual Studio Code, založení GitHub repozitáře, Opakování
- 2. 10. Webové technologie a WebAPI
- 9. 10. Interakce s REST API
- 16. 10. Testování kódu
- 23. 10. Databáze
- 30. 10. Generika + Repository design pattern
- 6. 11. Opakování, Refactoring, Dependency Injection
- 13. 11. Blazor 1
- 20. 11. HTTP Klient, Asynchronní programování
- 27. 11. Blazor 2
- 4. 12. Refactoring, Vylepšení UX
- 11. 12. Opakování, závěr

STRUKTURA LEKCE

Časově

- Začínáme v 18:00, končíme ve 20:30
- Plánujeme vždy dvě přestávky – zhruba v 18:50 a 19:50

Obsahově

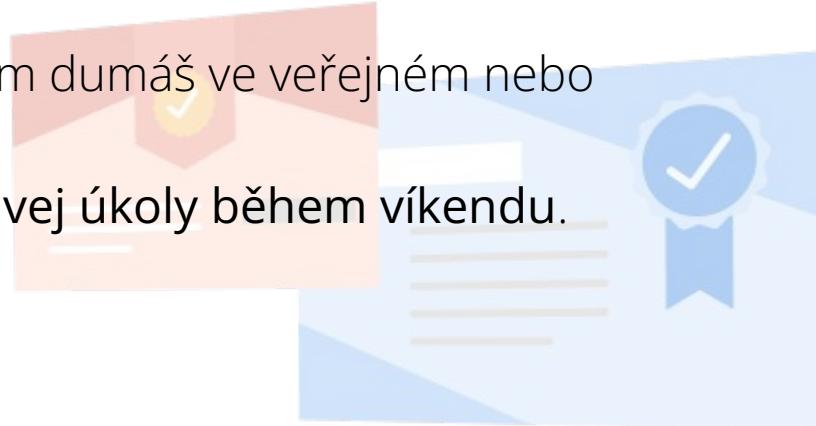
- Rekapitulace úkolů z minулé lekce
- Výklad nové látky
- Breakout rooms - samostatná práce na úkolech s kouči

ÚKOLY

Ale budou samé „jednoduché“, můžete nám věřit 😊

ÚKOLY

- Úkoly jsou **povinné** i nepovinné, nepovinné jsou silně doporučené.
- Odevzdávání bude probíhat do tvého vlastního GitHub repozitáře, kde dostaneš od svého kouče **individuální zpětnou vazbu**.
- Snažíme se opravit opravdu vše a odpovědět na všechny dotazy.
Využij toho.
- Připoj se na **Discord** a diskutuj nad čím dumáš ve veřejném nebo skupinkovém kanálu.
- Pro získání včasné odpovědi odevzdávej úkoly během víkendu.



BUDE TO BOLET?

- Doufáme, že nikoliv! Ale přece jen se tento kurz zabývá pokročilými tématy, takže buď na pozoru a připrav si dotazy.
- To, že jsi na úkolu strávila xy hodin, vůbec nevypovídá o tom, jestli na to máš nebo ne, jestli jsi chytrá, nebo ne, atd. Je to běžná součást učení, znamená to, že pracuješ opravdu intenzivně na tom, aby ses posunula dál a jde ti to!
- Neboj se tomu věnovat čas, je to zase úplně nová věc a na její vstřebání je potřeba zažít i „nepříjemné chvilky zoufalství“.
- Nezapomeň ale na pravidelný odpočinek.
- Odměnou ti bude funkční webová aplikace, pochopení, jak ji vytvořit a hlavně skvělý pocit!

DOTAZY

ŽÁDNÝ DOTAZ NENÍ HLOUPÝ

- Vždy se najde v místnosti někdo, kdo se na to chce také zeptat, ale nemá odvahu.
- Nezapomeň, že jsou tu i **koučové**, kteří sem přišli, aby ti byli k dispozici. Jsou to tví důležití **mentorové**, kteří tě budou vést správným směrem.
- Vyplňuj prosím zpětnou vazbu. Dává nám to hrozně moc!

DISCLAIMER

- Toto je **první běh** nového kurzu. Můžeme se setkat s mnohými obtížemi, technického či edukačního typu.
- Vaše **zpětná vazba je naprosto klíčová!** Potřebujeme vědět, kde přidat a kde ubrat, co se líbilo, co ne. Co bylo nejasné a nad čím jste dumaly.
- Časová dotace kurzu neumožňuje jít do přílišné hloubky některých technologií (vystačily by na samostatný kurz).
Důležité je si je vysvětlit a pochopit jejich princip.
- Očekávejte, že některé informace si budete dohledávat samy (z anglických zdrojů/dokumentace). Zdroje poskytneme a kouči vám budou k dispozici.
Vyhledávání informací je zásadní pro vaši budoucí pozici.

IT POZICE



IT POZICE

Computing-Core Disciplines	Computing-Intensive Fields	Computing-Infrastructure Occupations
Artificial intelligence	Aerospace engineering	Blockchain administrator
Cloud computing	Autonomous systems	Computer technician
Computer science	Bioinformatics	Data analyst
Computer engineering	Cognitive science	Data engineer
Computational science	Cryptography	Database administrator
Database engineering	Computational science	Help desk technician
Computer graphics	Data science	Identity theft recovery agent
Cyber security	Digital library science	Network technician
Human-computer interaction	E-commerce	Professional IT trainer
Network engineering	Genetic engineering	Reputation manager
Programming languages	Information science	Security specialist
Programming methods	Information systems	System administrator
Operating systems	Public Policy and Privacy	Web identity designer
Performance engineering	Instructional design	Web programmer
Robotics	Knowledge engineering	Web services designer
Scientific computing	Management information systems	
Software architecture	Network science	
Software engineering	Multimedia design	
	Telecommunications	

Backend vývojářka

Lekce 01

Intro

Lekce 01 - Intro

1. Rozjetí **IDE** (Integrated Development Environment)
 - a. .NET SDK 8.0
 - b. Visual Studio Code
 - i. Import nastavení
 - c. FizzBuzz projektík

Všichni jsme na jedné lodi, máme připravené vývojové prostředí, dokážeme se v něm orientovat a něco si zaprogramujeme ;)

S ČÍM BUDEME PRACOVAT

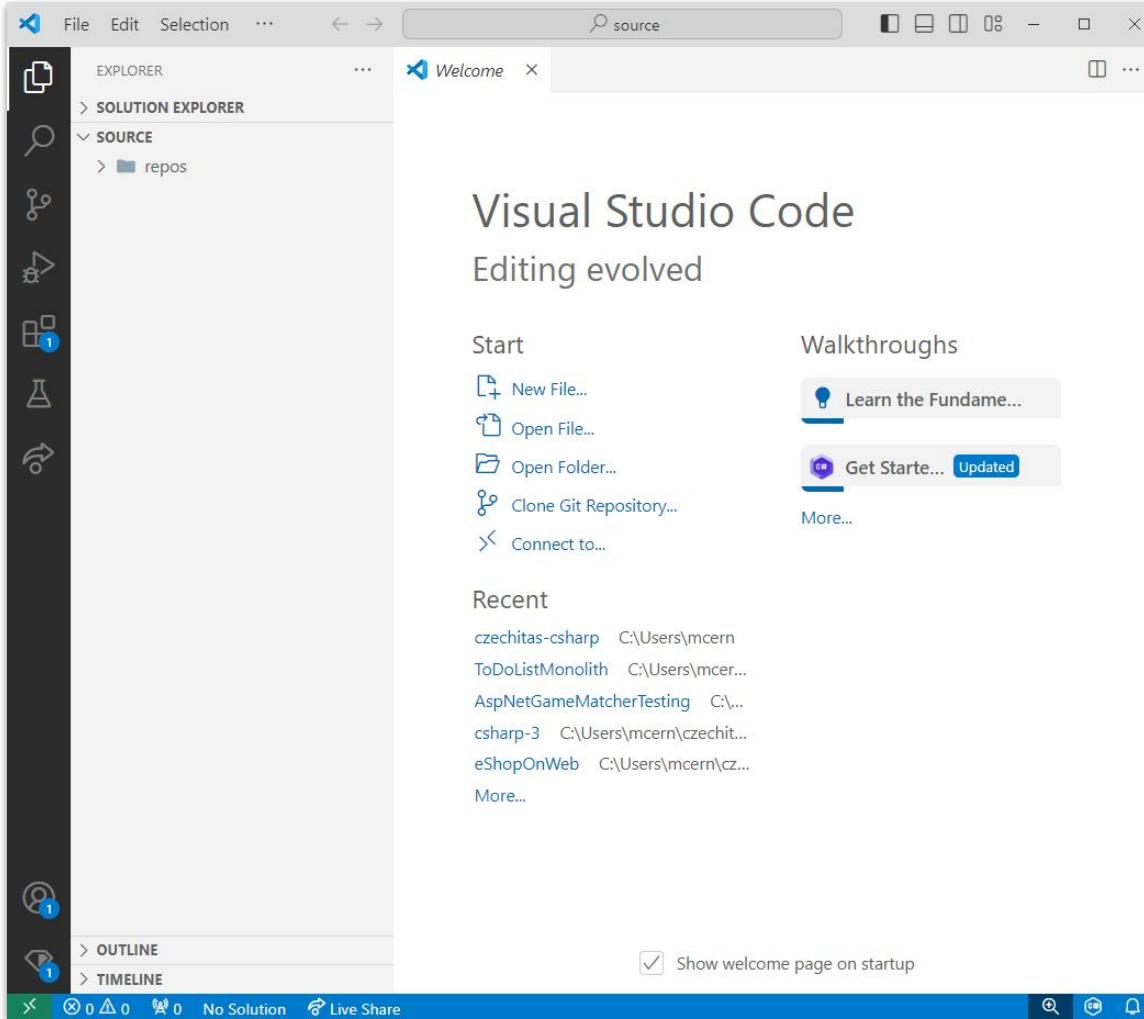


.NET 8.0
<https://dot.net/download>

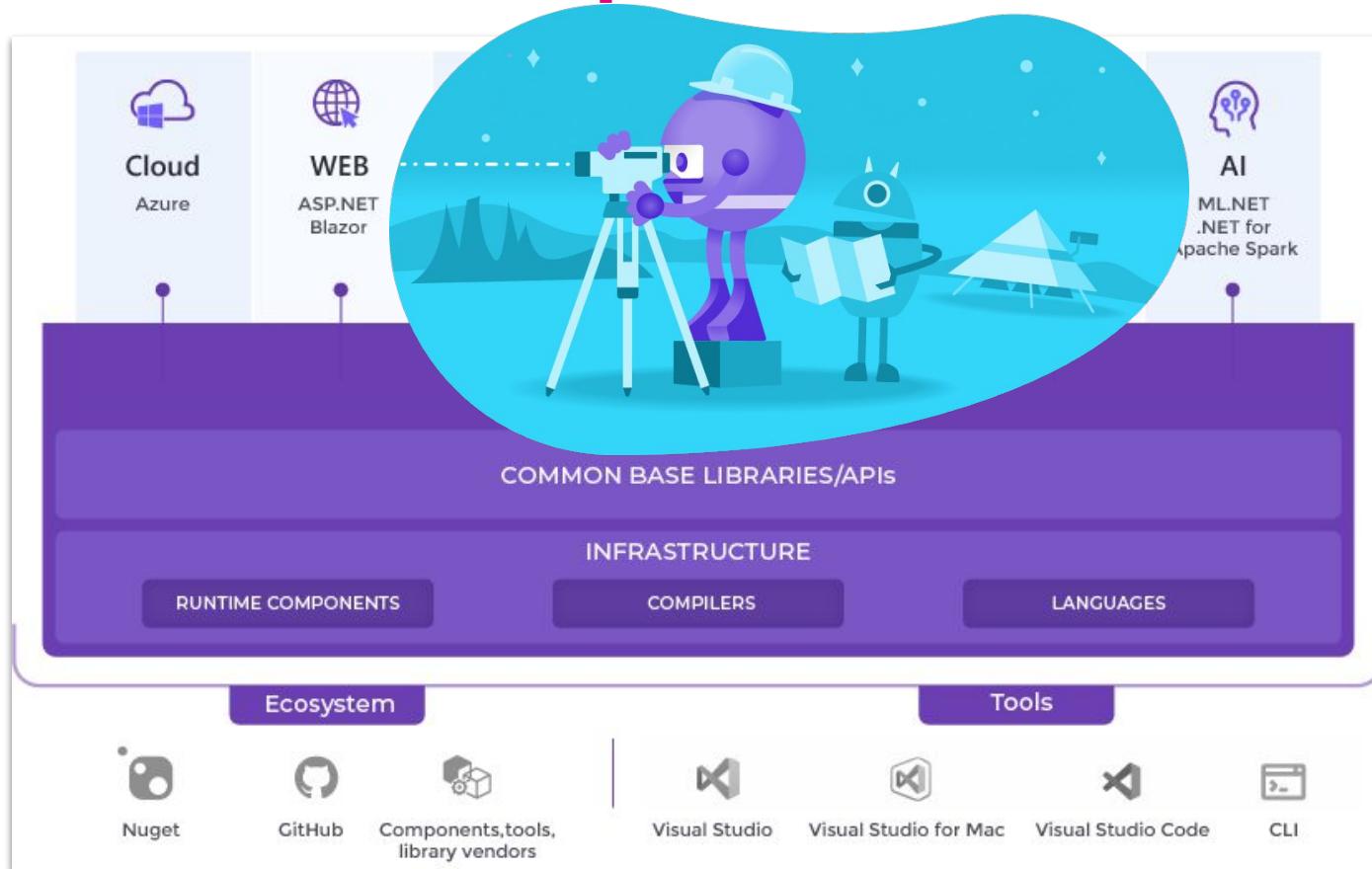


Visual Studio Code
<https://code.visualstudio.com>

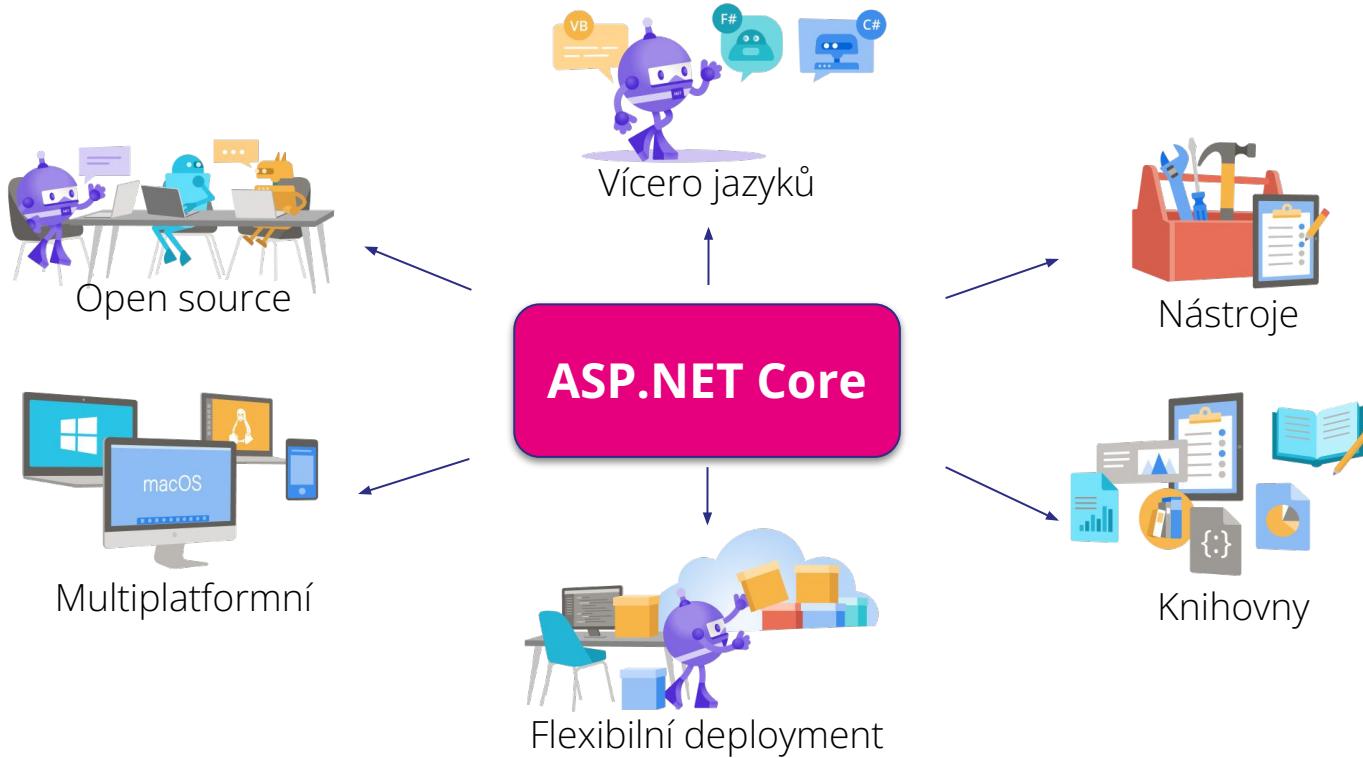




Na co vše se dá .NET použít



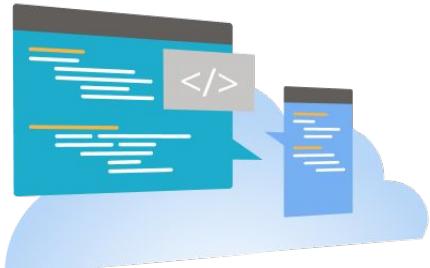
Co je to ASP.NET Core?



Co se dá s ASP.NET Core vytvořit?



Web UI



Backend Services

Server Rendered



Client Rendered (Single Page App)



Services

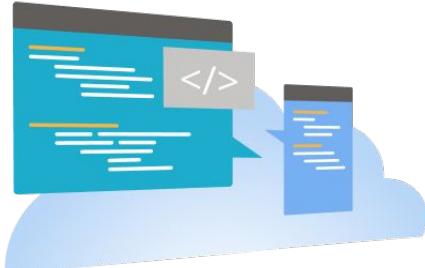
Web APIs



Co se dá s ASP.NET Core vytvořit?



Web UI



Backend Services

Server Rendered

MVC

Razor Pages

Client Rendered (Single Page App)

Blazor

Services

Web APIs

Controller based

Minimal

SignalR

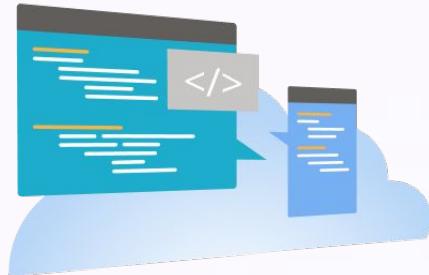
gRPC

Workers

Co se dá s ASP.NET Core vytvořit?

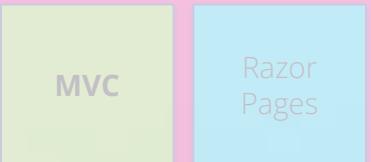


Web UI

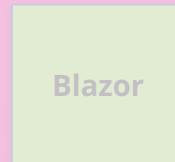


Backend Services

Server Rendered

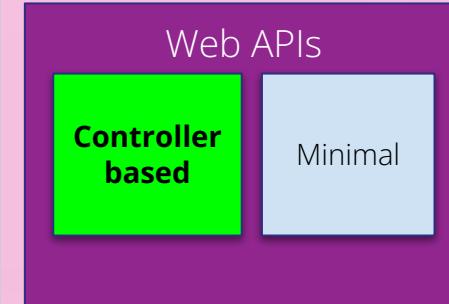


Client Rendered (Single Page App)



Services

Web APIs



SignalR

gRPC

Workers

No a co s tím?

Demo

- Faketsy
- EShopOnWeb
- GameStore
- GameMatcher

HOSPODA?

Aneb zpětnou vazbu si rádi vyslechneme i ~~u piva~~ ústně.



POJĎME PROGRAMOVAT 😊

ROZEHŘÁTÍ

FizzBuzz

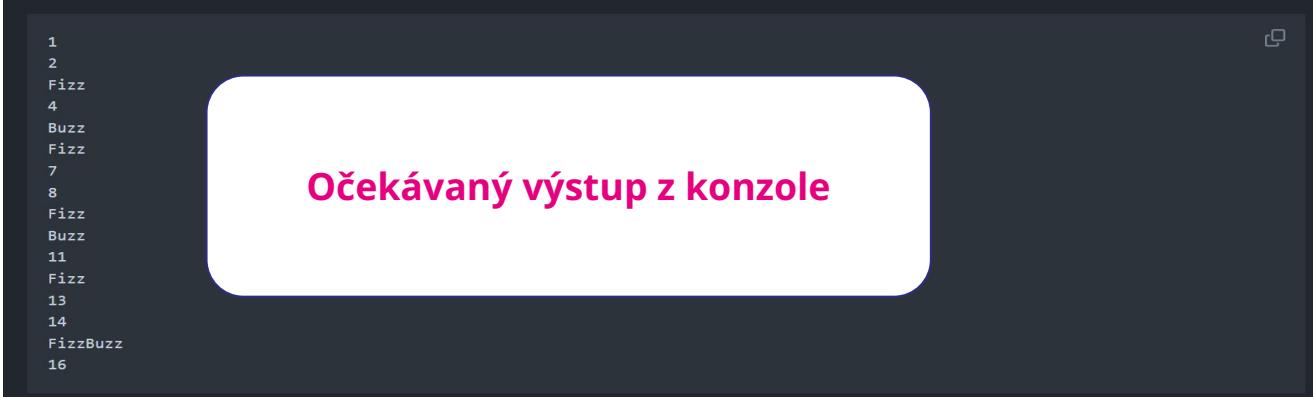
FizzBuzz - Zadání

FizzBuzz je jednoduchá číselná hra, ve které počítáš od 1 do X a nahrazuješ určitá čísla slovy "Fizz" a/nebo "Buzz" podle určitých pravidel.

Vytvoř třídu `FizzBuzz` s metodou `CountTo(int lastNumber)`, která vytiskne do konzole čísla od 1 do `lastNumber`, každé na nový řádek.

Místo čísel **dělitelných 3** by měla metoda vypsat do konzole "**Fizz**". Místo čísel **dělitelných 5** by měla vypsat do konzole "**Buzz**". A místo čísel **dělitelných 3 i 5** by měla vypsat do konzole "**FizzBuzz**".

Odkaz



The screenshot shows a terminal window with a dark background. On the left, there is some code. In the center, a white rounded rectangle contains the text "Očekávaný výstup z konzole". To the right of the white box, the terminal output is displayed:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
```

Začněte



Začněte?

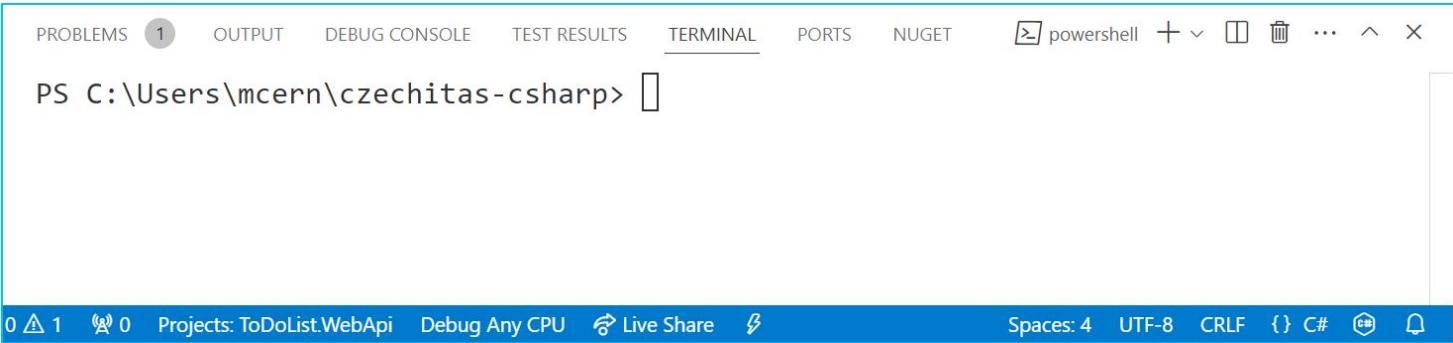


Začněme spolu!



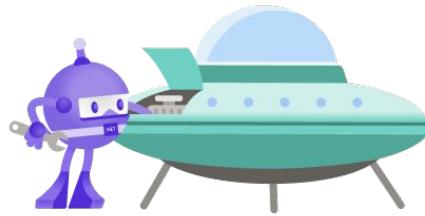
KONZOLE

- Slouží pro zadávání příkazů
a zobrazování výstupů (výpis hodnot) u konzolových aplikací
- Ve VSCode ji zapínáme pomocí **ctrl + ;** (**Cmd + `** pro Mac)



The screenshot shows the VSCode interface with the terminal tab selected. The terminal window displays a PowerShell prompt: "PS C:\Users\mcern\czechitas-csharp>". The status bar at the bottom shows project information ("Projects: ToDoList.WebApi"), build configuration ("Debug Any CPU"), and various code analysis tools like Live Share and Live Coverage.

dotnet --version
dotnet --info



KONZOLE - Příkazy Dotnet CLI

Pro práci v tomto kurzu budeme potřebovat taky pár příkazů.

Nebojte se, vždy je budete mít předem připravené na dosah ruky
a kláves **ctrl+c** ;)

dotnet new <template name>

(přidává nové projekty na základě šablony)

dotnet build

(přeložit program)

dotnet run

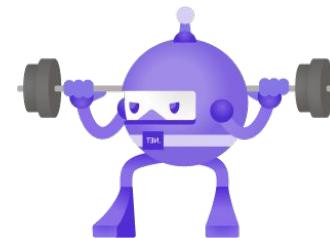
(spustit program)

dotnet watch

(interaktivní mód)

dotnet test

(spustí všechny testy)



KONZOLE - Příkazy Dotnet CLI

Aktuálně potřebujeme jen tyto

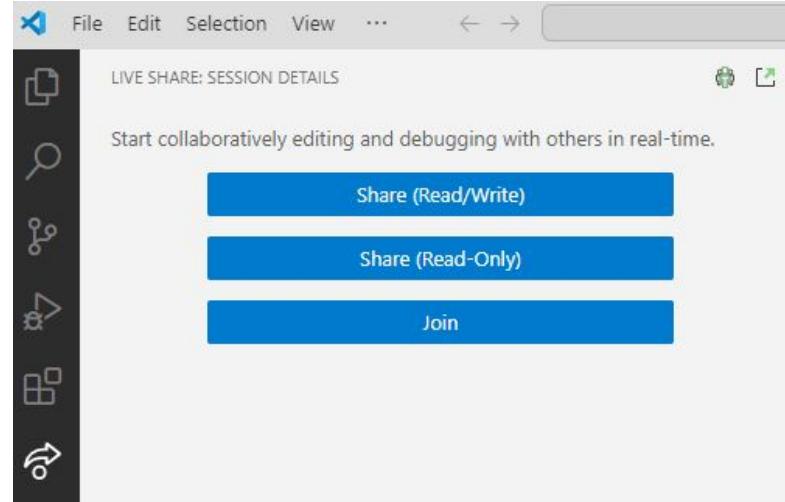
```
dotnet new solution --name FizzBuzz
```

```
dotnet new console --name FizzBuzz
```

```
dotnet sln add FizzBuzz (přidá projekt FizzBuzz do solution)
```

```
dotnet watch (interaktivní mód)
```

LiveShare



LEKCE 02

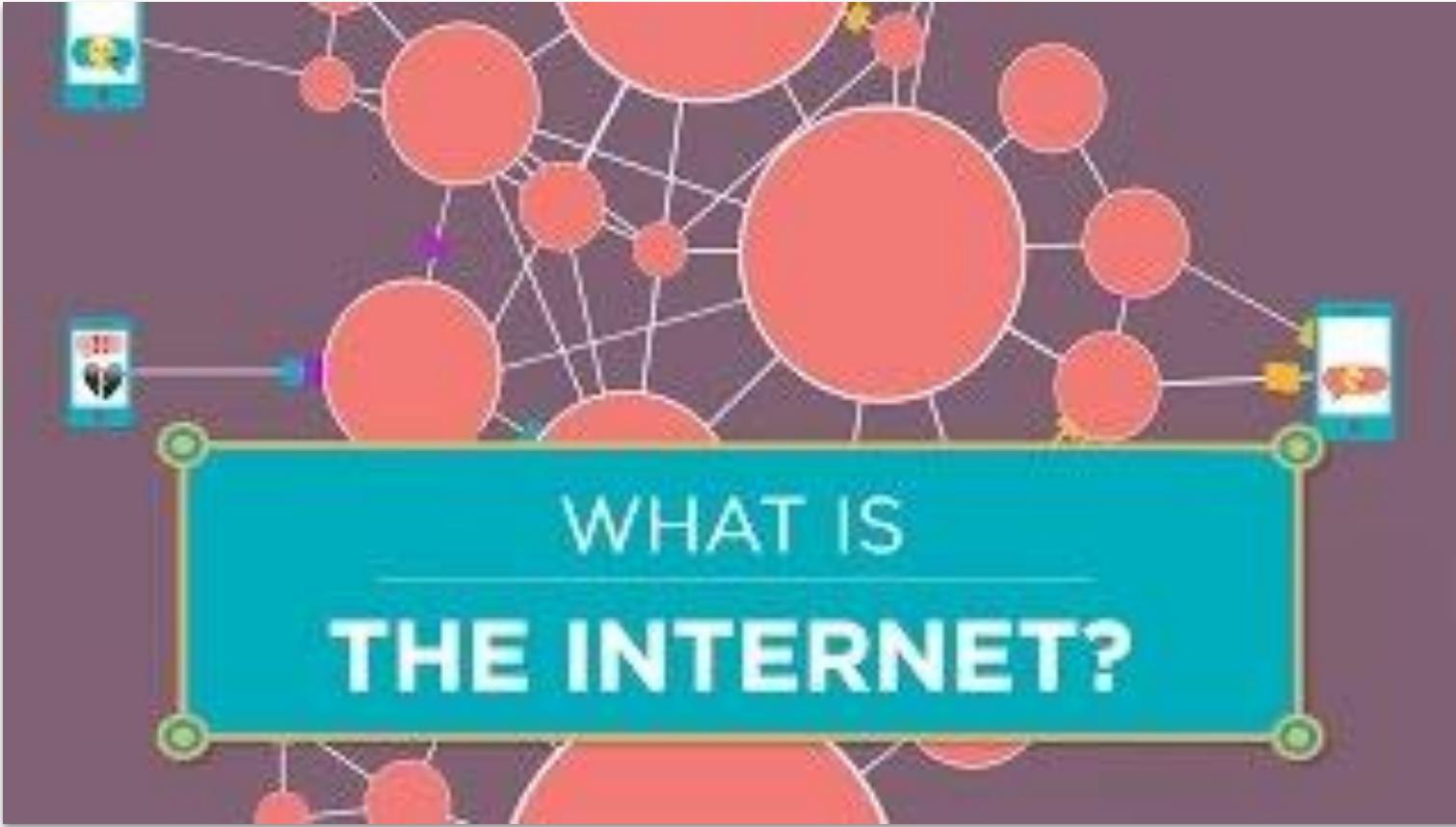
Webové technologie

Lekce 02 - Webové technologie

- Teorie
 - Internet
 - Odkazy **URL**
 - **Webové stránky**
 - Prohlížeče
 - **Webová API (REST, CRUD)**
- Založení projektu ToDoList.WebApi
 - Orientace v projektu
 - Význam

Máme povědomí o internetových technologiích, víme co jsou webová API a dokážeme se zorientovat v ASP.NET projektu ;)

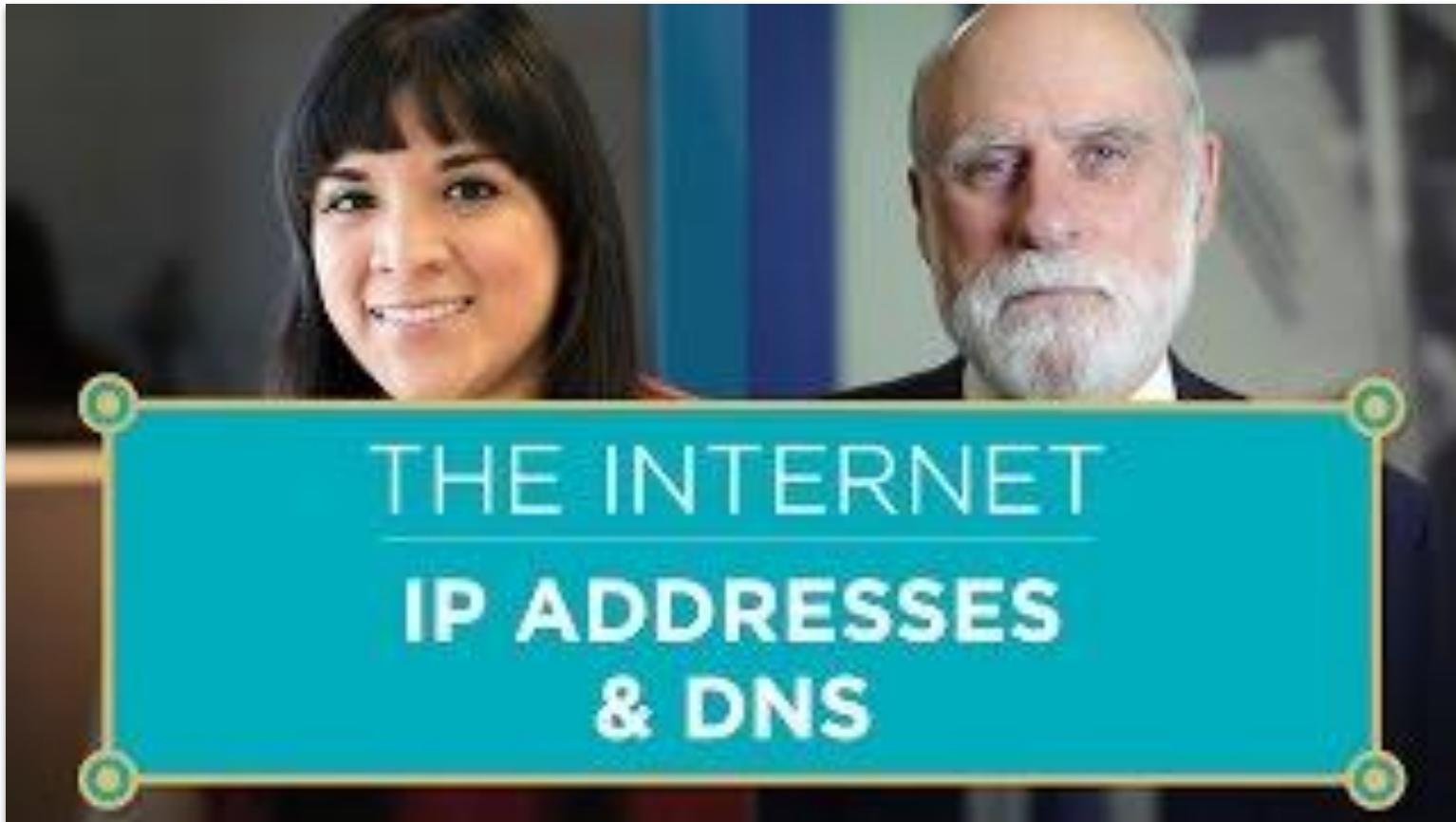
Internet



WHAT IS THE INTERNET?

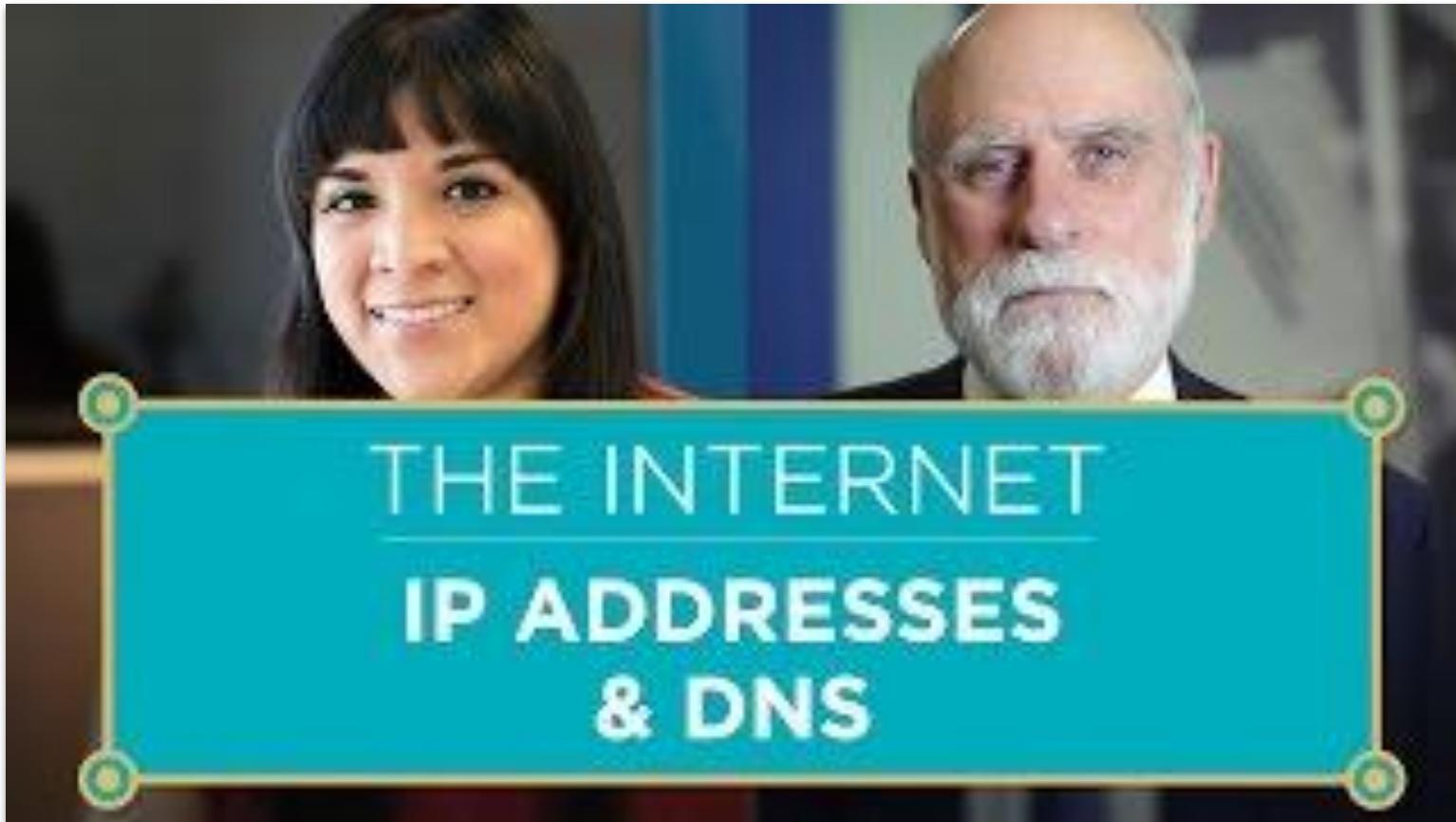
IP Adresa

Internet Protocol Address



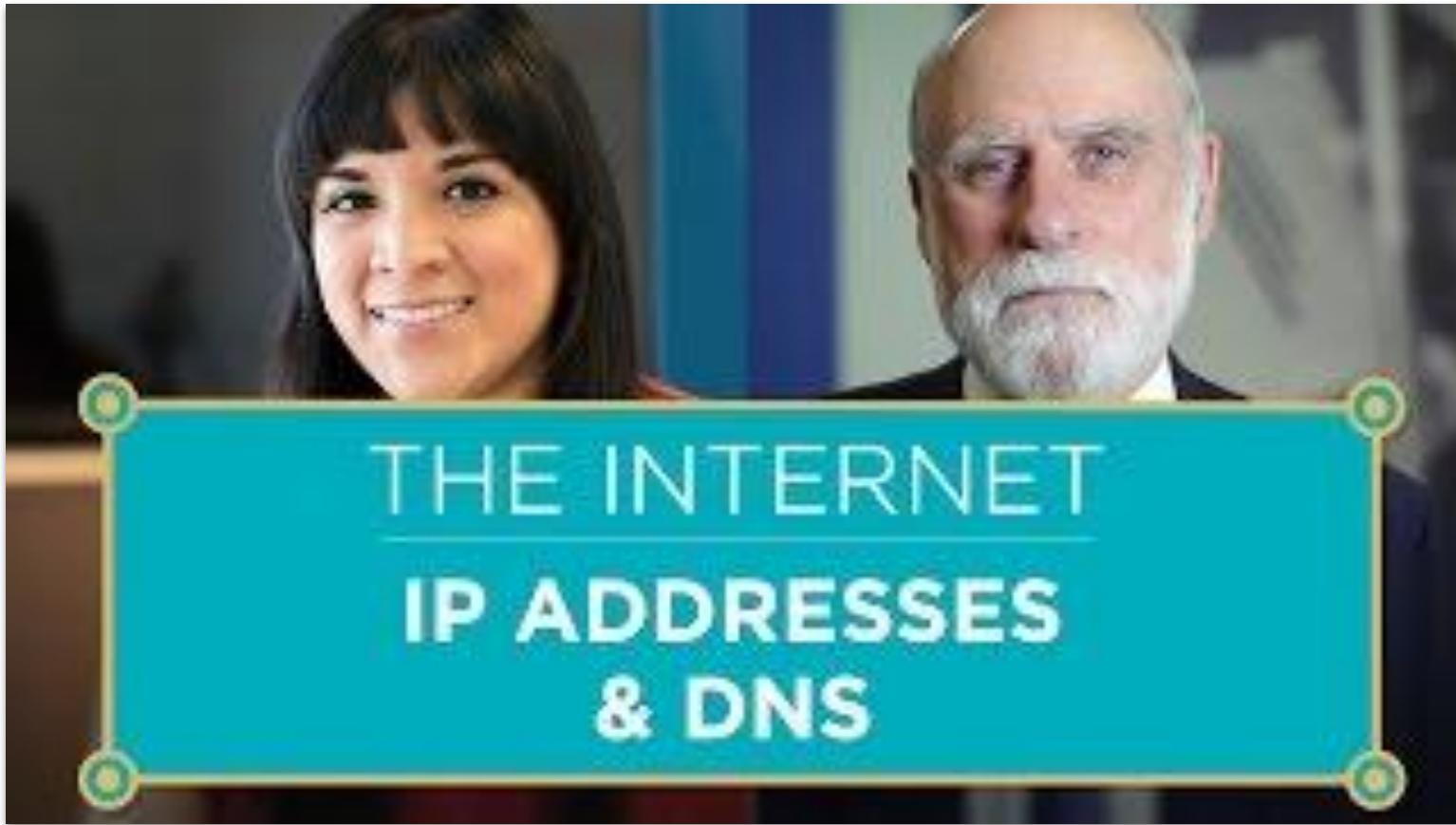
THE INTERNET

IP ADDRESSES & DNS



THE INTERNET

IP ADDRESSES & DNS



THE INTERNET

IP ADDRESSES & DNS

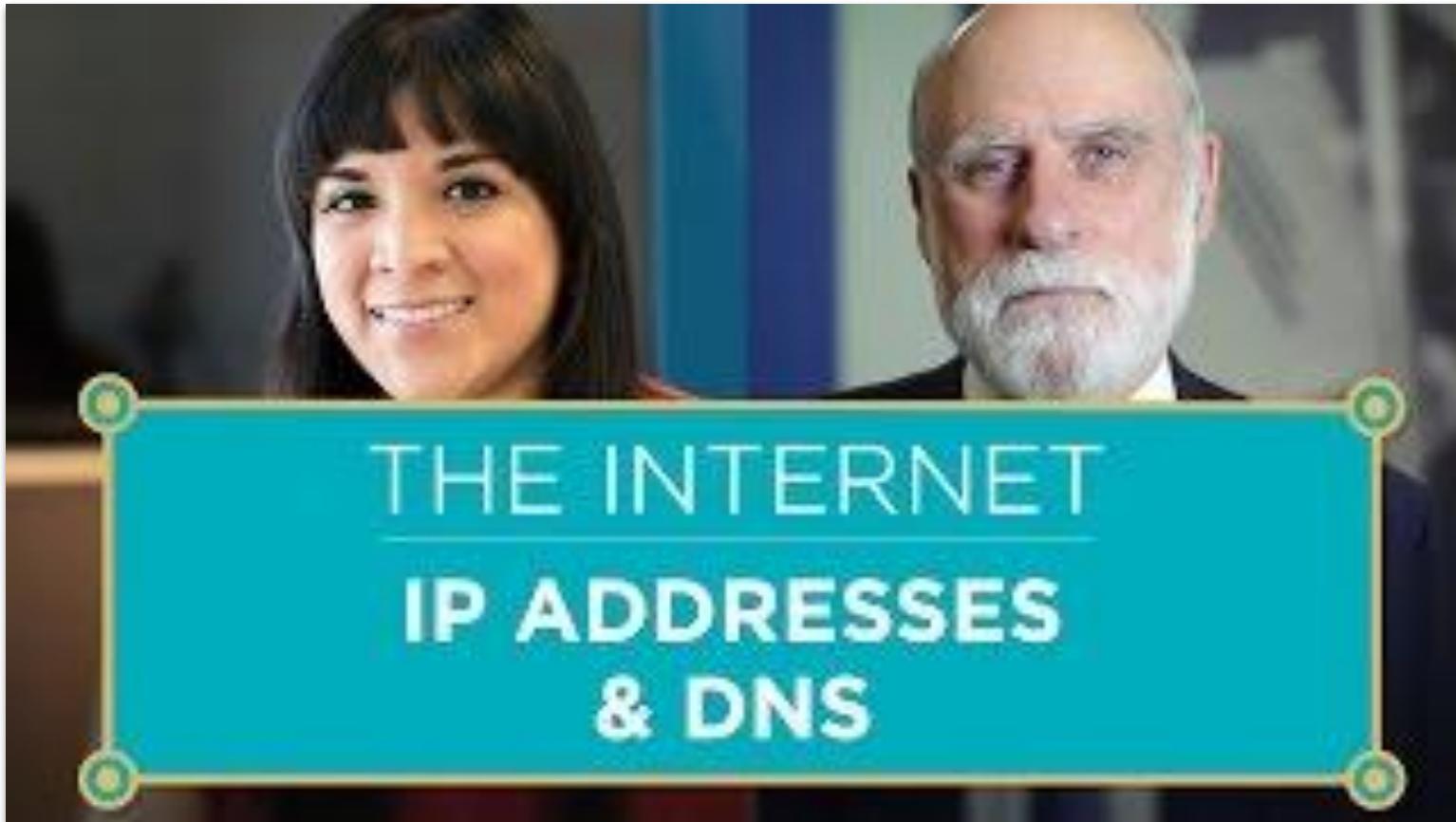
DNS

Domain Name System

I KNOW A GUY
WHO KNOWS A GUY
WHO KNOWS
ANOTHER GUY

Br
B
Bad

NETFLIX



THE INTERNET

IP ADDRESSES & DNS

Prohlížeče



THE INTERNET

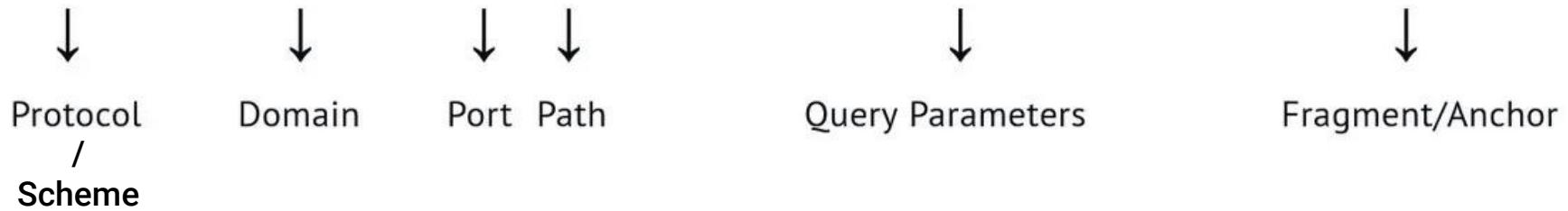
HTTP & HTML

URL

Uniform Resource Locator

URL Anatomy

https://example.com:80/blog?search=test&sort_by=created_at#header



HTTP

Hyper**T**ext **T**ransfer **P**rotocol



THE INTERNET

HTTP & HTML

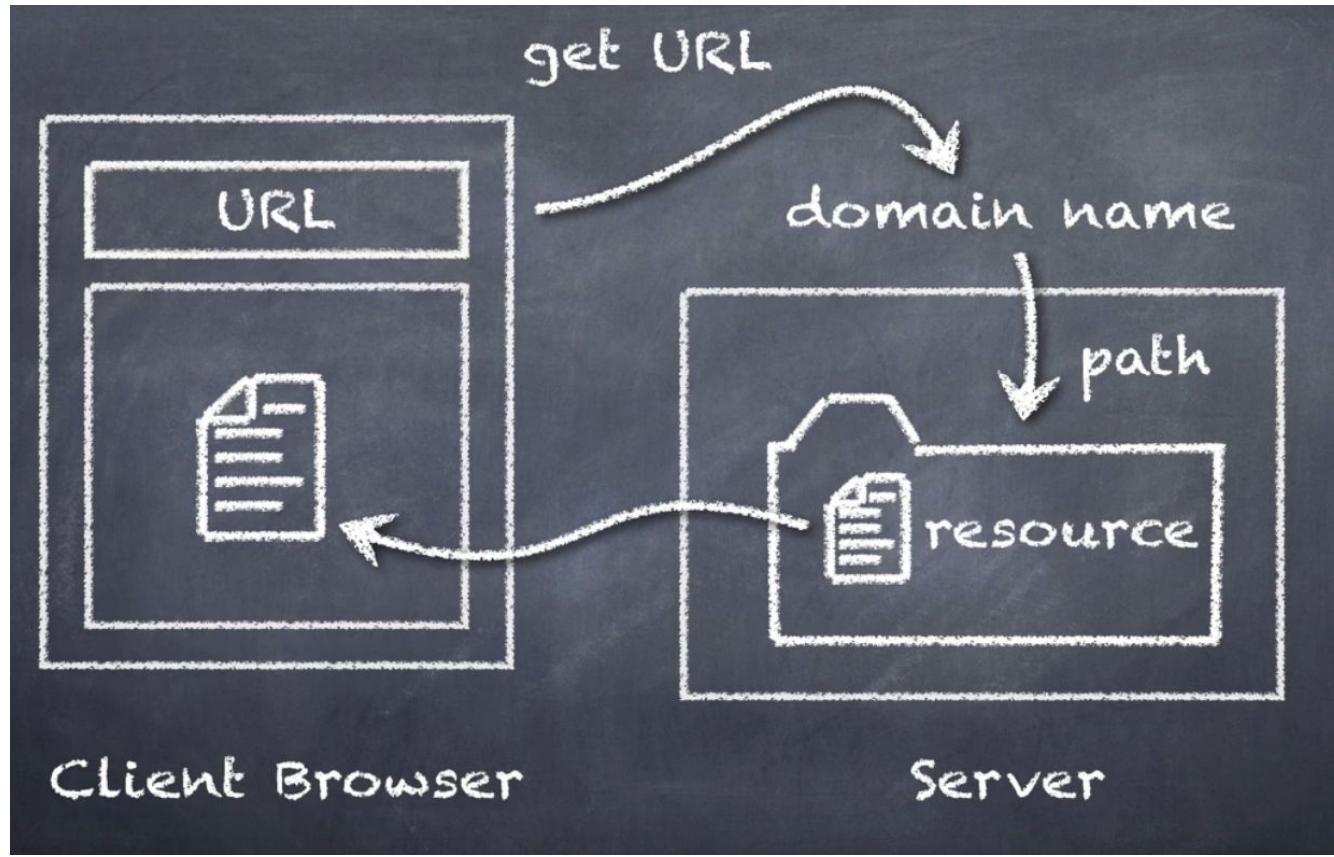
HTTP GET

DEMO V PROHLÍŽEČI

<https://www.google.com/search?client=chrome&q=dog>



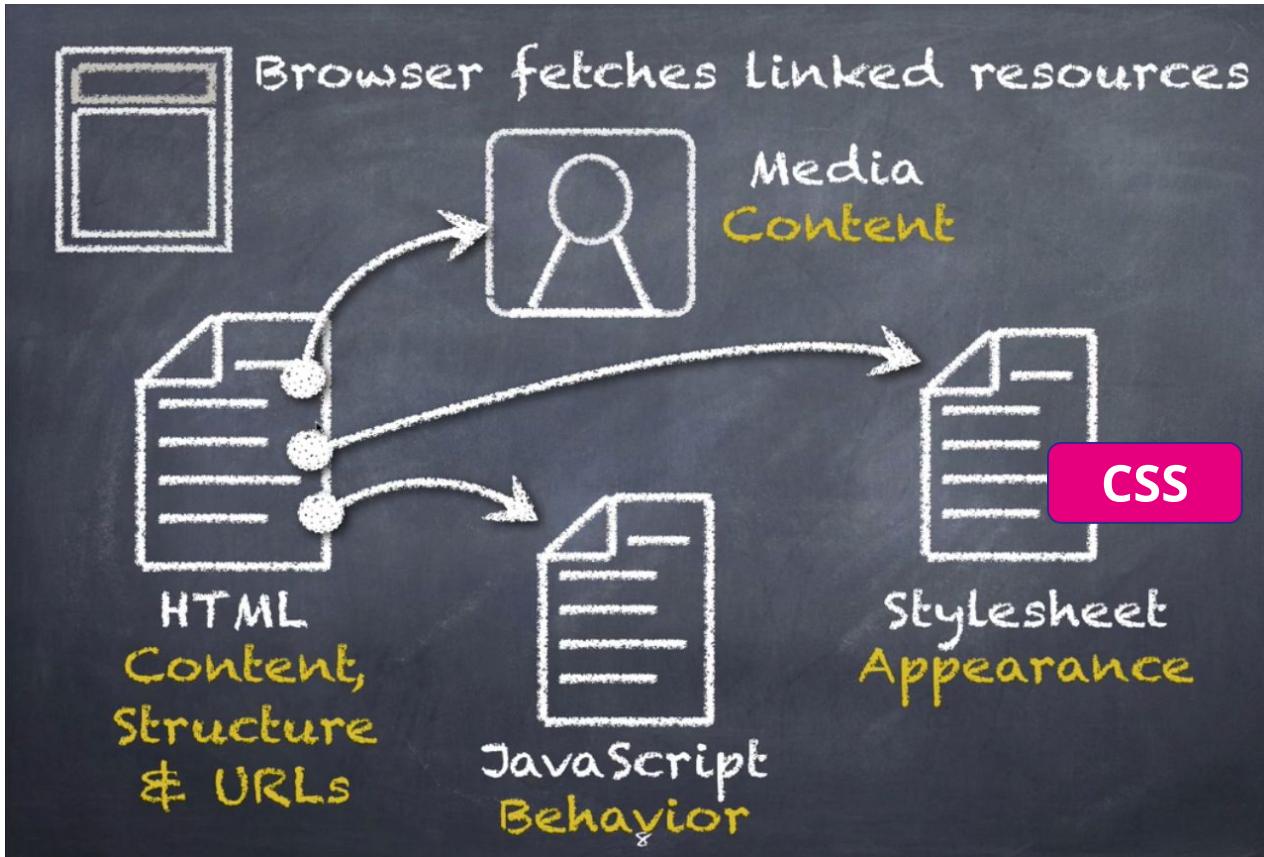
Zde se
schovává
port :80



HTML

Hyper**T**ext **M**arkup **L**anguage

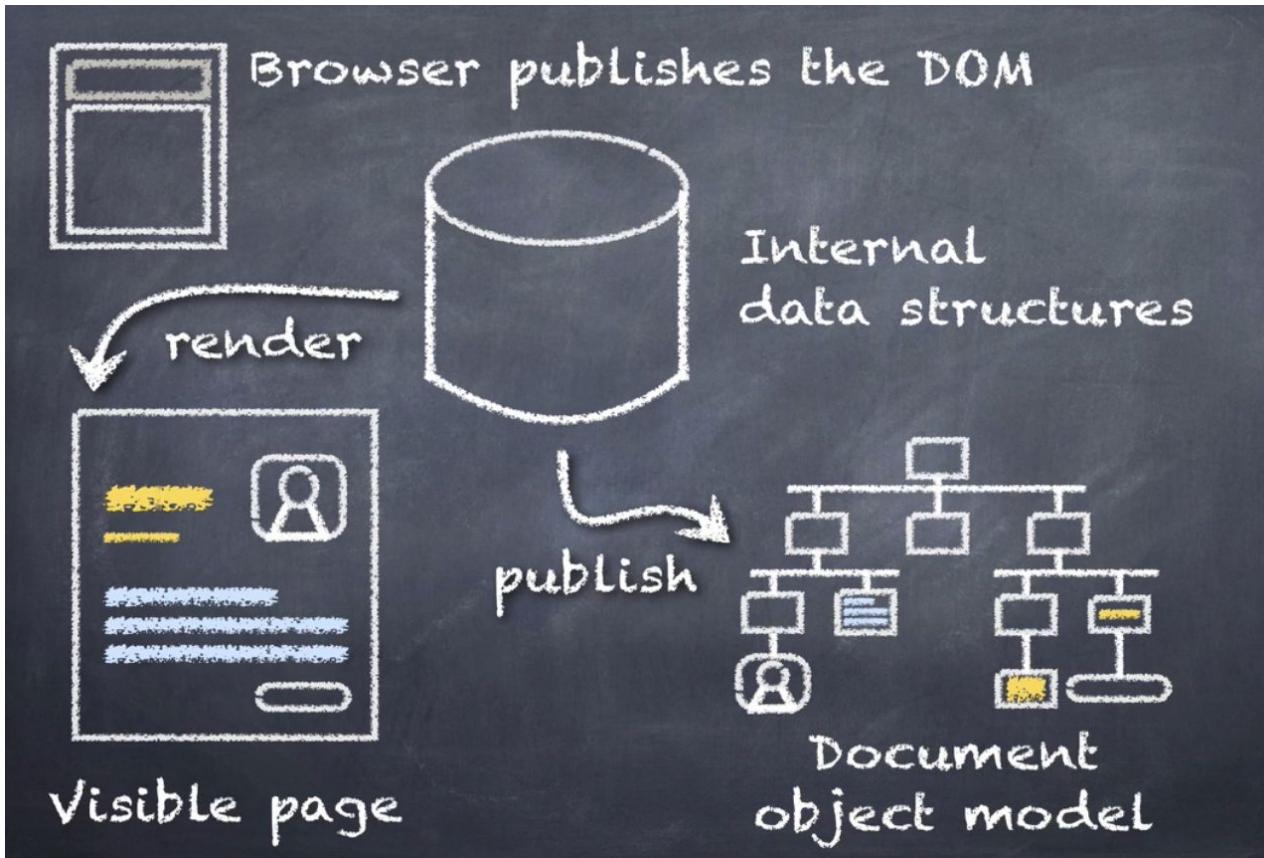




CSS

Cascading Style Sheets



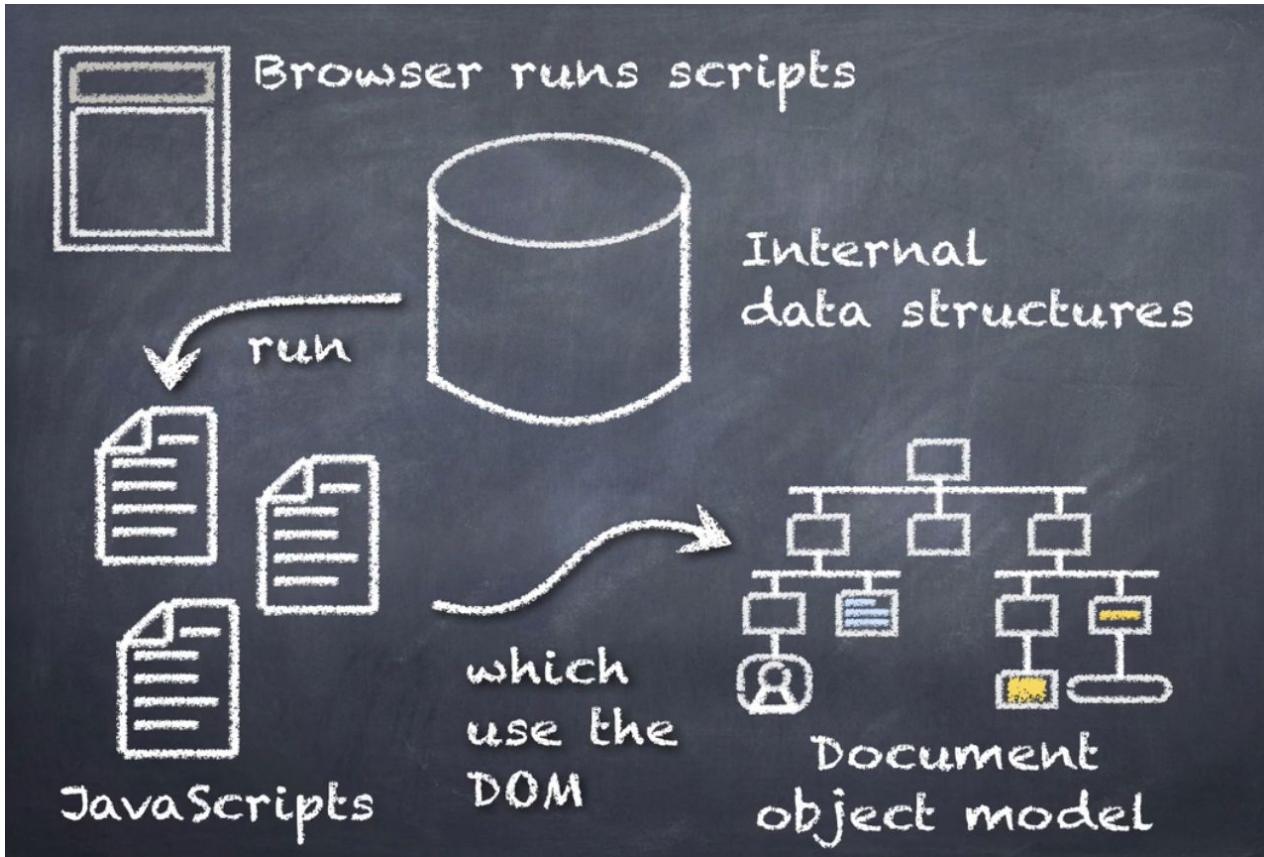


JavaScript

Interaktivní Weby



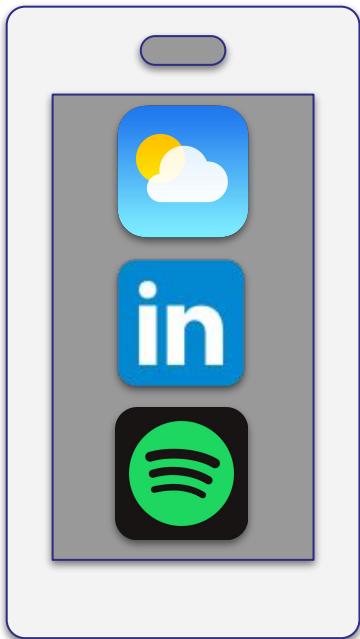
https://www.flaticon.com/free-sticker/disco_11449497





Co je to Web API?

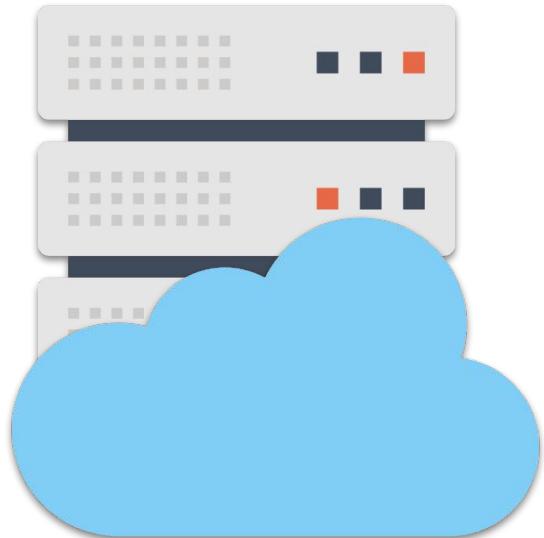
Klienti a Servery



Klient



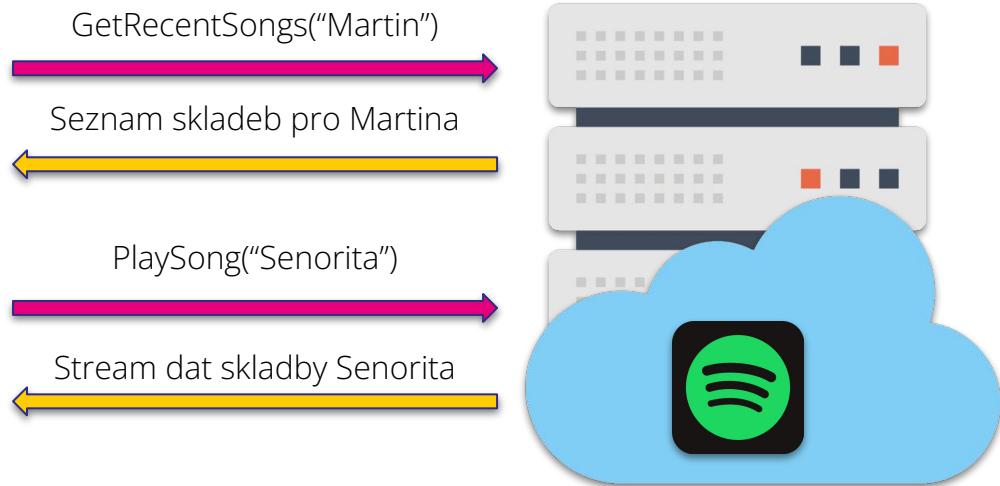
Jak získat data?



Server

Co je to API?

Application Programming Interface

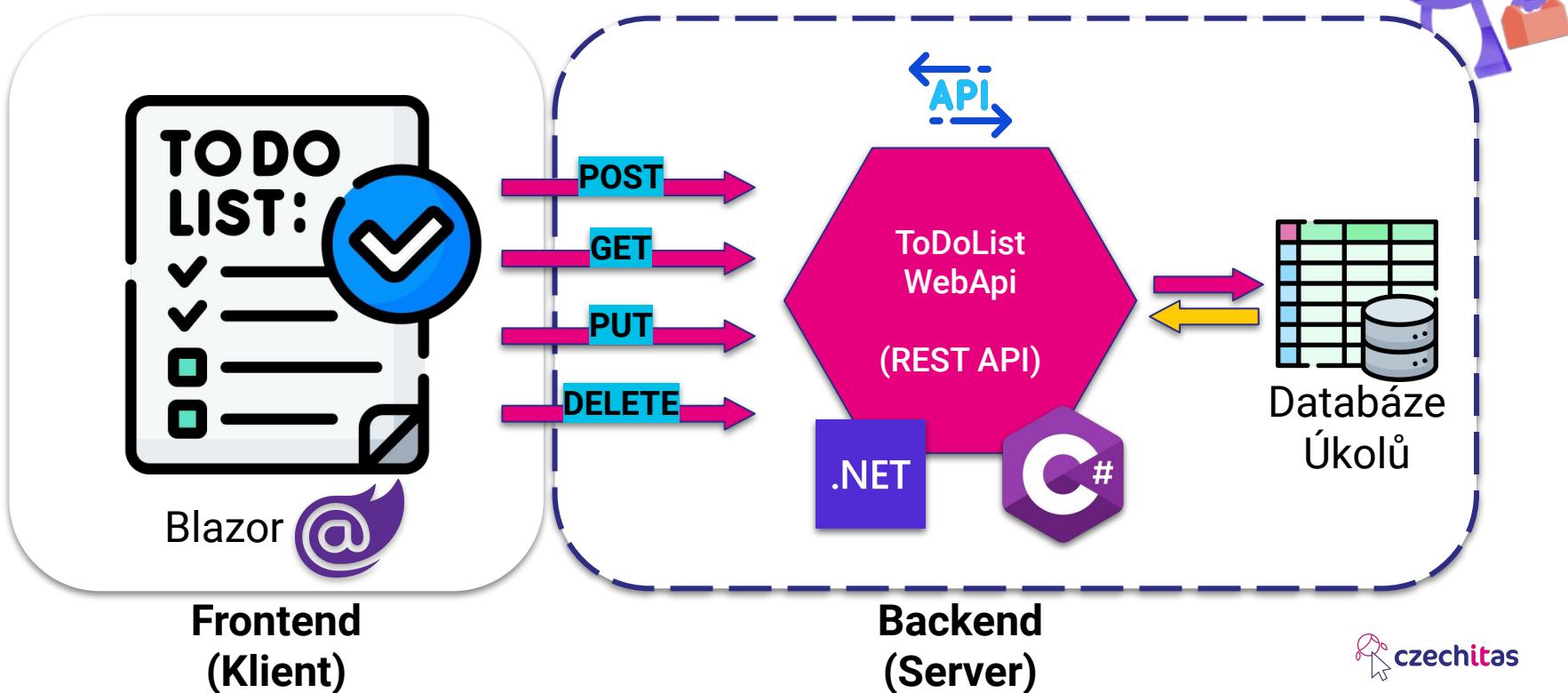


API pomáhá klientům komunikovat se službami takovým způsobem, aby služba rozuměla a splnila klientem zadaný požadavek (request).

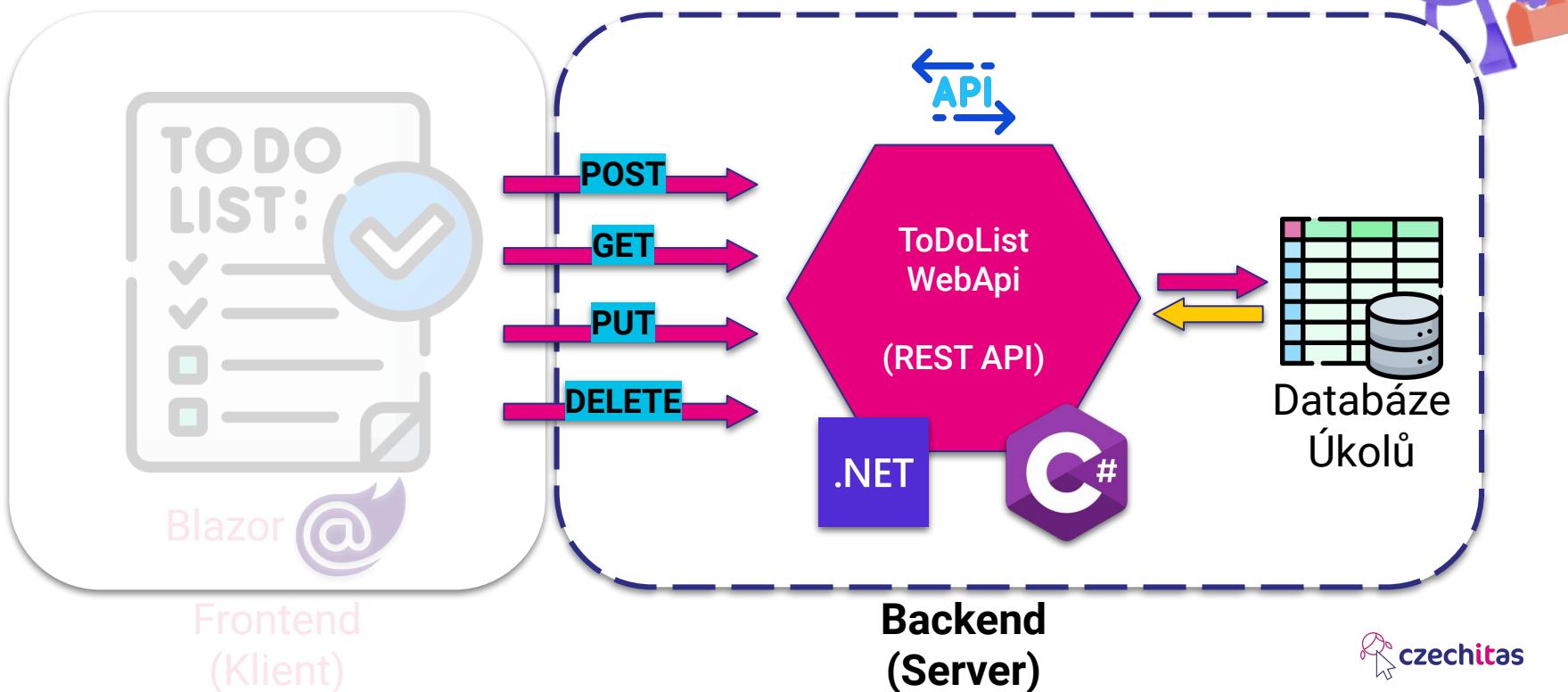
Co je to Web API?

Web API je takové API, které je dostupné na počítačové síti (např. internetu). Typicky využívá ke komunikaci protokol HTTP.

CO SPOLU POSTAVÍME



CO SPOLU POSTAVÍME



**Pojďme se
vrhnout do
programování**

Co je to REST API?

Co je to REST?

REpresentational

State

Transfer

Stateless

Klient-Server

Uniform
Interface

Layered
system

Cacheable

Code on
demand

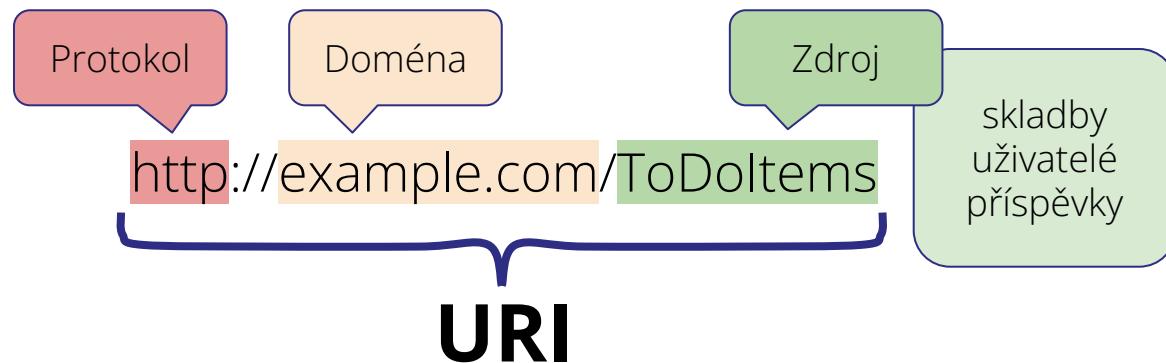
Soubor principů, které určují podmínky, jak má API fungovat.

Co je to REST API?

REST API nebo *RESTFUL API* je takové API,
které dodržuje *REST* architekturální styl.

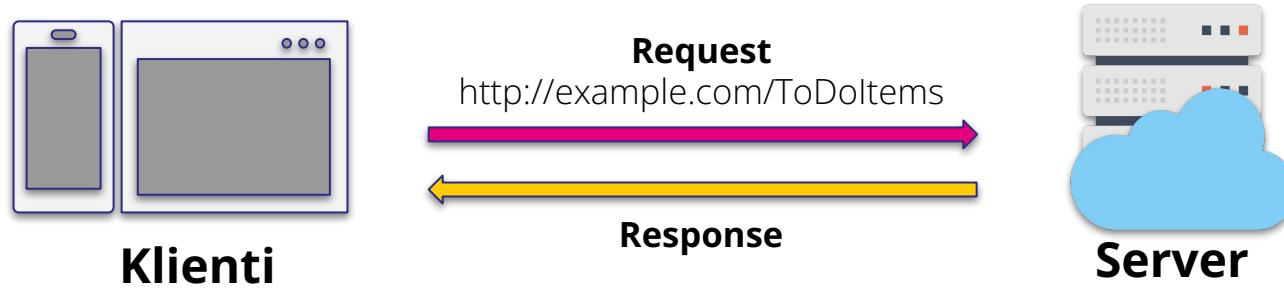
Jak identifikovat zdroje v REST API?

Zdroj je jakýkoliv objekt, dokument nebo věc, kterou API dokáže přijímat nebo posílat klientům



Uniform Resource Identifier

Jak interagovat s REST API?



CRUD operace

HTTP Metody

Create	POST	Vytvoří nový zdroj
Read	GET	Získá reprezentaci/stav zdroje
Update	PUT	Modifikace zdroje
Delete	DELETE	Smaže zdroj

LEKCE 03

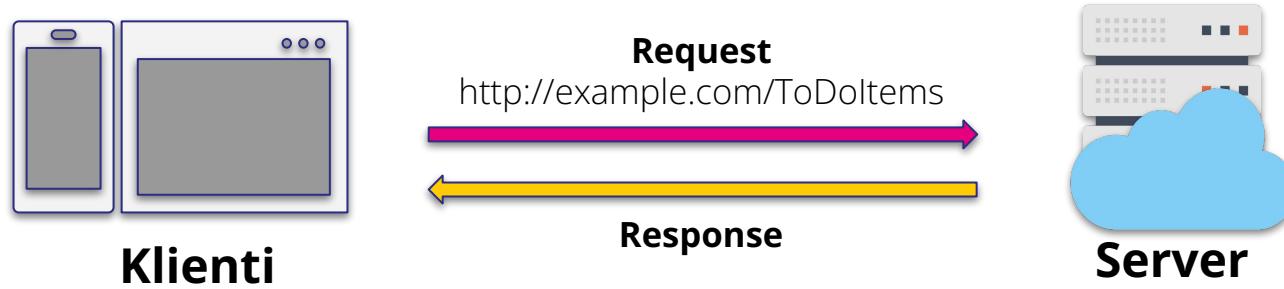
Interakce s REST API

Práce se šablonou

Lekce 03 - Interakce s REST API

- ToDoList.WebApi
 - **CRUD** metody
 - **Endpoints, Routing**
 - **WebApi Controllers**
- **DTOs**
 - Význam, Definice, Použití
- Testování volání API
 - **REST Client + .http soubory**
- **Implementace** API
 - Nejjednodušší možná

Jak interagovat s REST API?



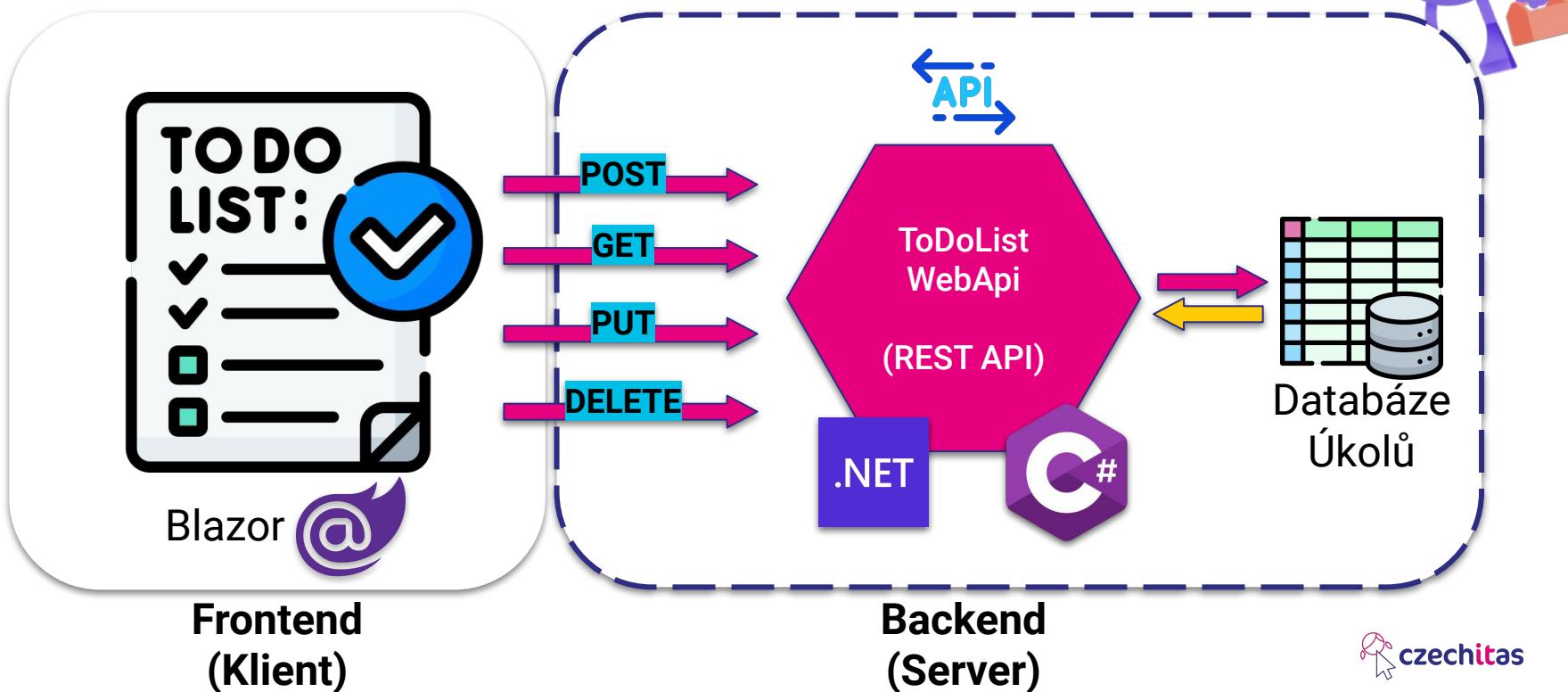
CRUD operace

HTTP Metody

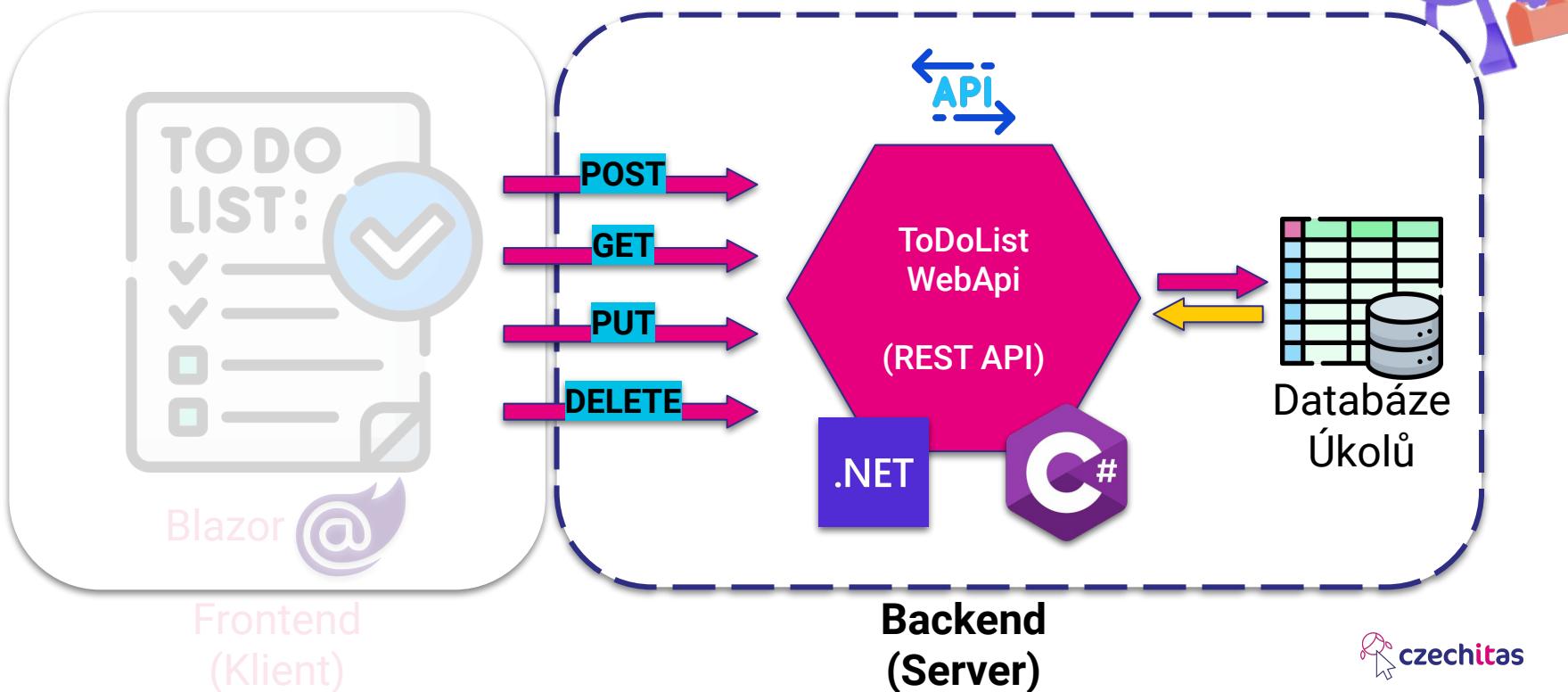
Create	POST	Vytvoří nový zdroj
Read	GET	Získá reprezentaci/stav zdroje
Update	PUT	Modifikace zdroje
Delete	DELETE	Smaže zdroj

**Pojďme si vyzkoušet
GET**

CO SPOLU POSTAVÍME



CO SPOLU POSTAVÍME



Web API Endpoint

Co je to Web API Endpoint?

URL, která reprezentuje konkrétní zdroj (data nebo službu). Každý zdroj má svůj unikátní endpoint, který umožňuje k němu přistupovat.

Web API Endpoints: EShop

<https://api.eshop.com/product>

<https://api.eshop.com/users>

<https://api.eshop.com/cart>

<https://api.eshop.com/payment>

Web API Endpoints: ToDoList

<https://localhost:<port>/ToDoItems>

REST API Endpoint pro Úkoly

Create	POST	Vytvoří nový zdroj
Read	GET	Získá reprezentaci/stav zdroje
Update	PUT	Modifikace zdroje
Delete	DELETE	Smaže zdroj

POST /ToDoItems
GET /ToDoItems
GET /ToDoItems/1
PUT /ToDoItems/1
DELETE /ToDoItems/1

REST API pro Úkoly

GET

/ToDoItems

GET

GET

/ToDoItems/1

GET

POST

/ToDoItems

POST

PUT

/ToDoItems/1

PUT

DELETE

/ToDoItems/1

DELETE

Read

ReadById

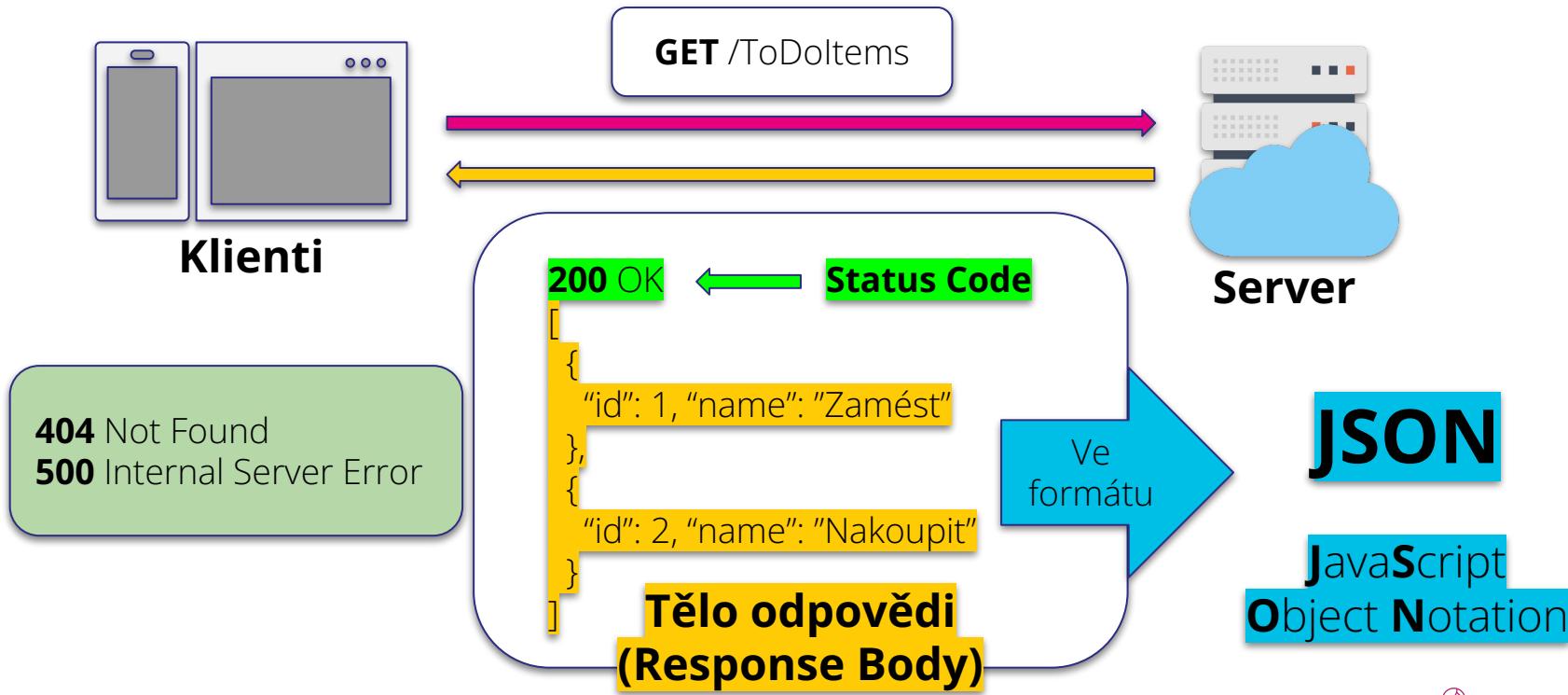
Create

UpdateById

DeleteById

ToDoList
WebApi
(REST API)

Získej všechny úkoly - HTTP GET



Jaké máme k dispozici Status Codes?

Informational (**100 - 199**)

Successful (**200 - 299**)

Redirection messages (**300 - 399**)

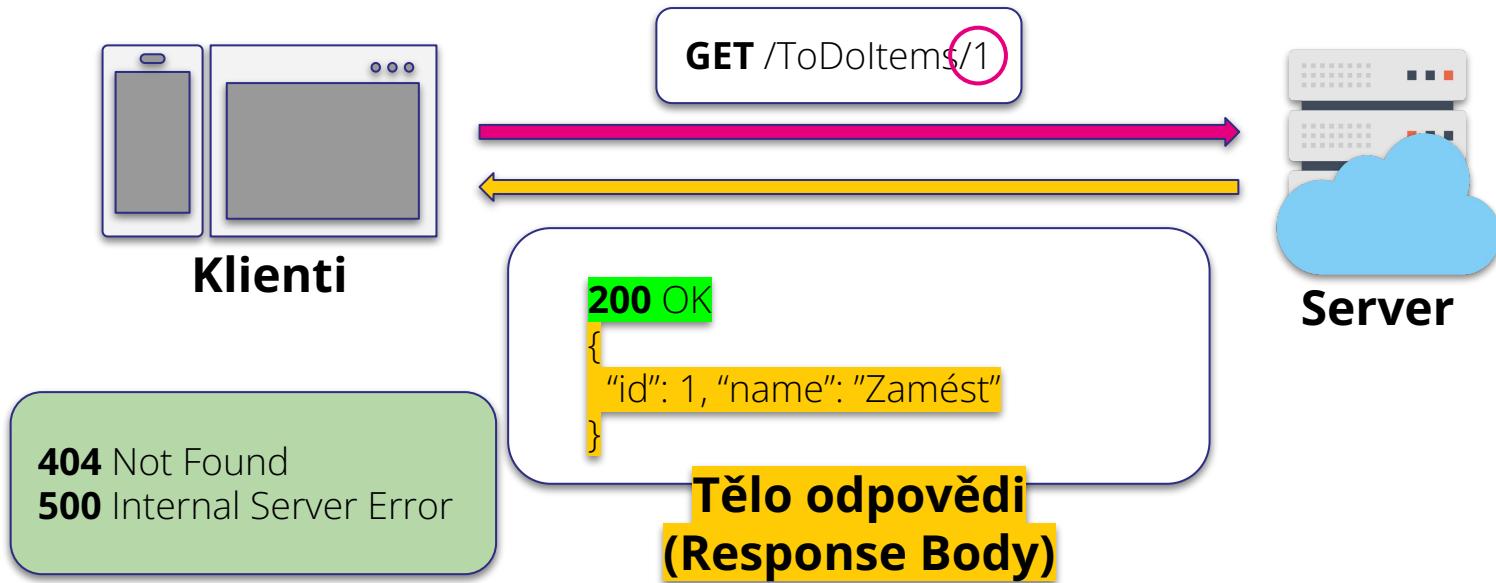
Client Error response (**400 - 499**)

Server Error responses (**500 - 599**)

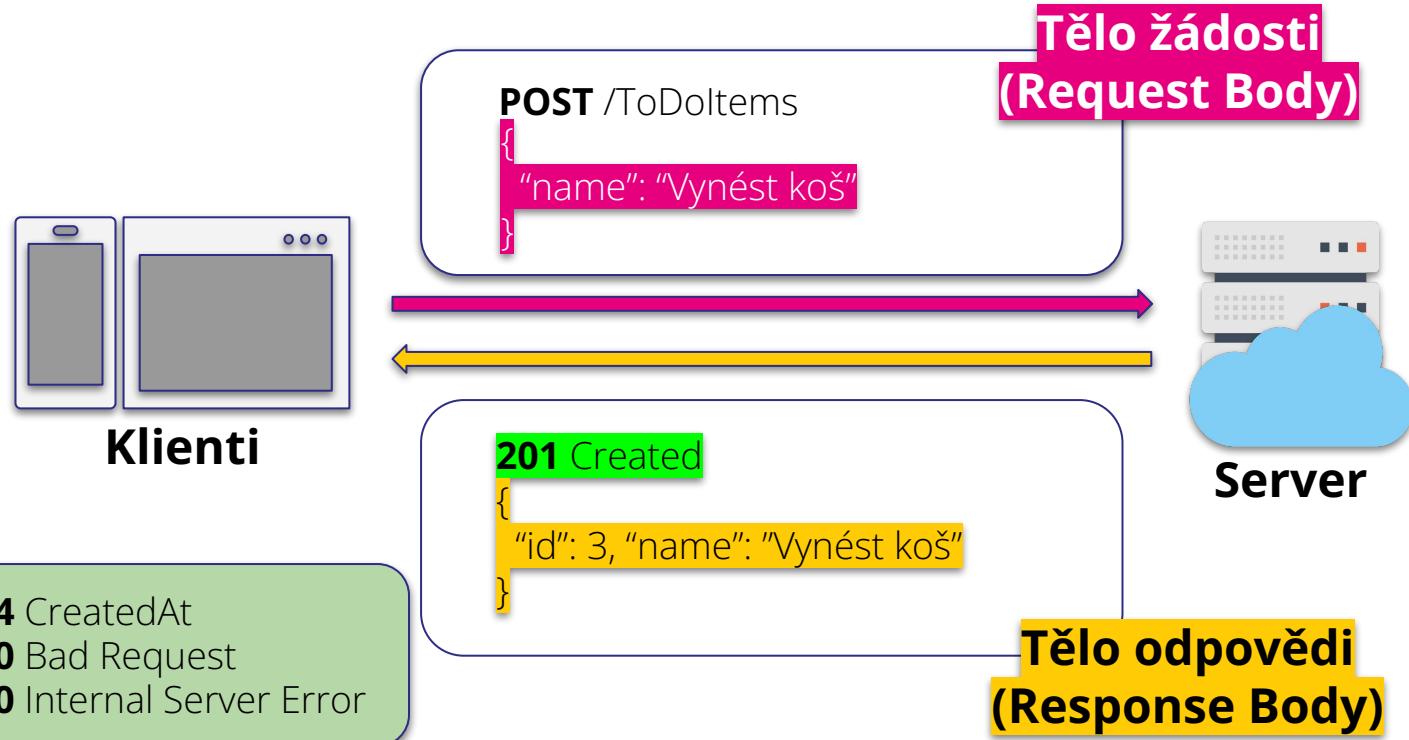
Link



Získej specifický úkol - HTTP GET



Vytvoř úkol - HTTP POST



Modifikuj úkol - HTTP PUT



404 Not Found
400 Bad Request
500 Internal Server Error

Smaž specifický úkol - HTTP DELETE



404 Not Found
400 Bad Request
500 Internal Server Error

REST API pro Úkoly

POST /ToDoItems
GET /ToDoItems
GET /ToDoItems/1
PUT /ToDoItems/1
DELETE /ToDoItems/1



Routing

Co je to Routing?

Routing rozhoduje, jaká akce nebo část kódu se má spustit na základě cesty zadané v URL a podle použité HTTP metody (GET, POST, PUT, DELETE apod.).

DTO

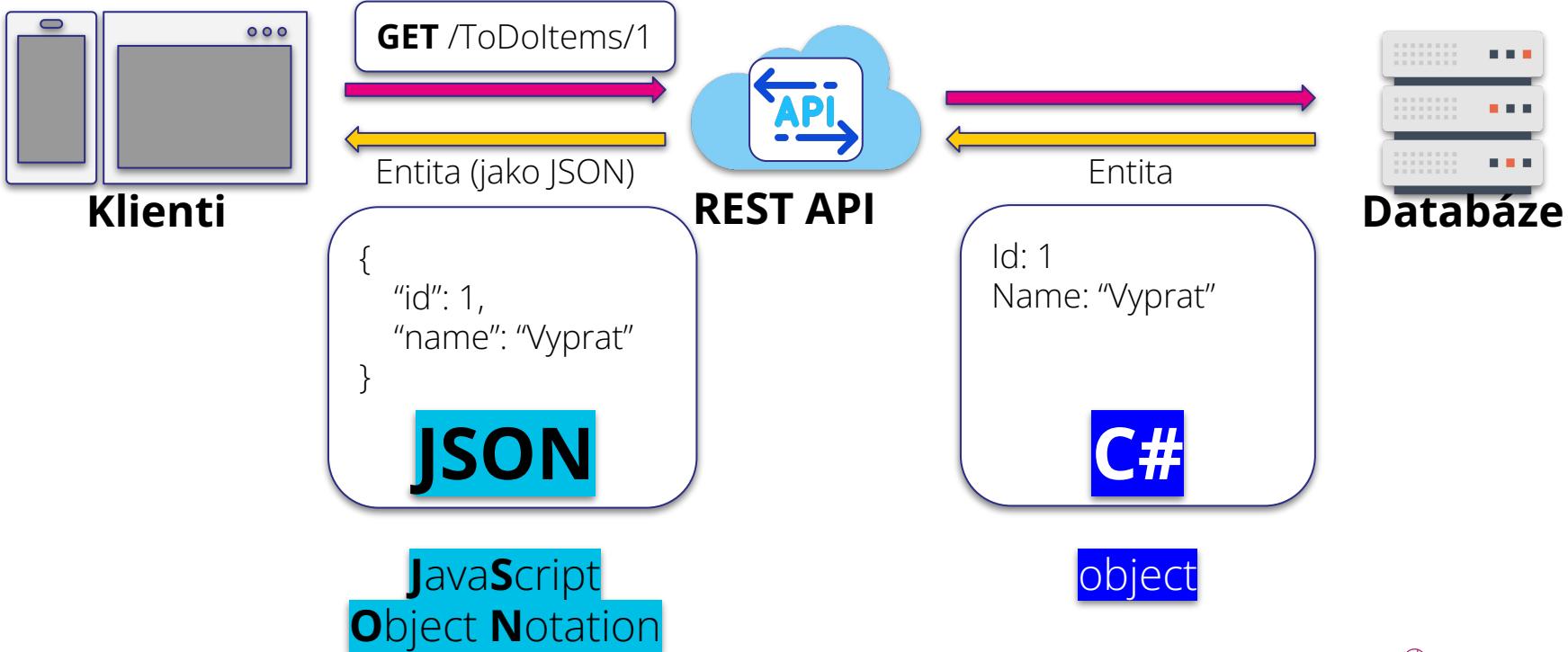
Data Transfer Object

Co je to DTO?

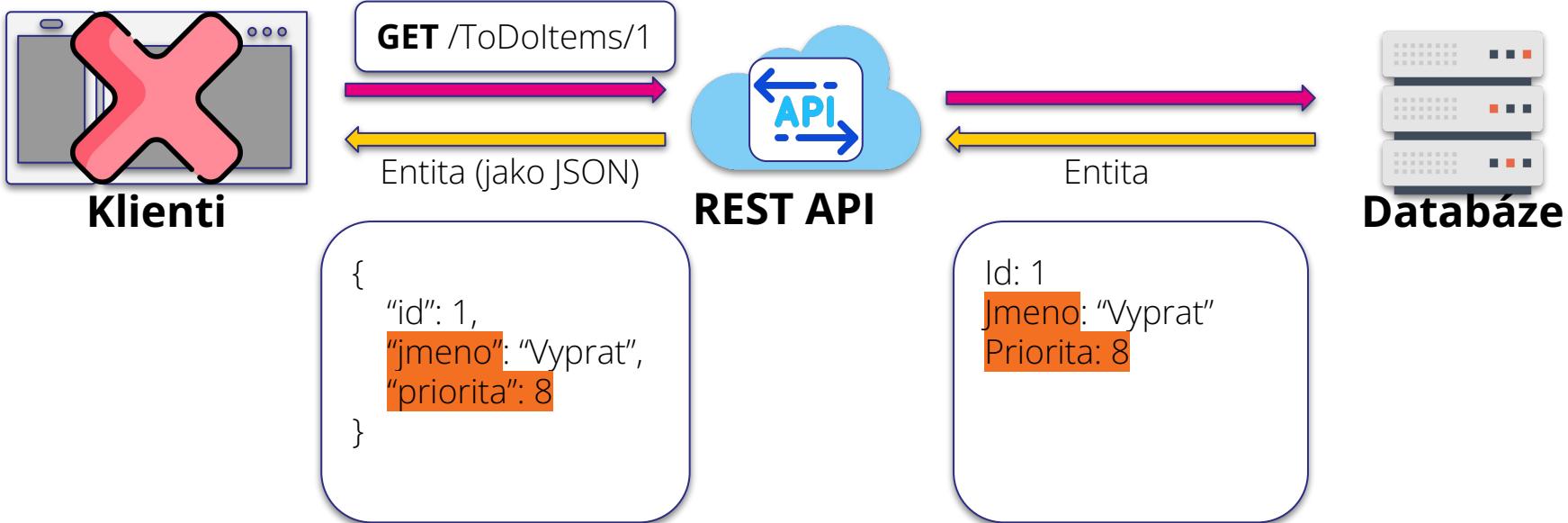
Data Transfer Object (DTO) je takový objekt, který přenáší data mezi procesy nebo aplikacemi.

V kontextu REST API, DTO může být považován jako kontrakt mezi klientem a serverem

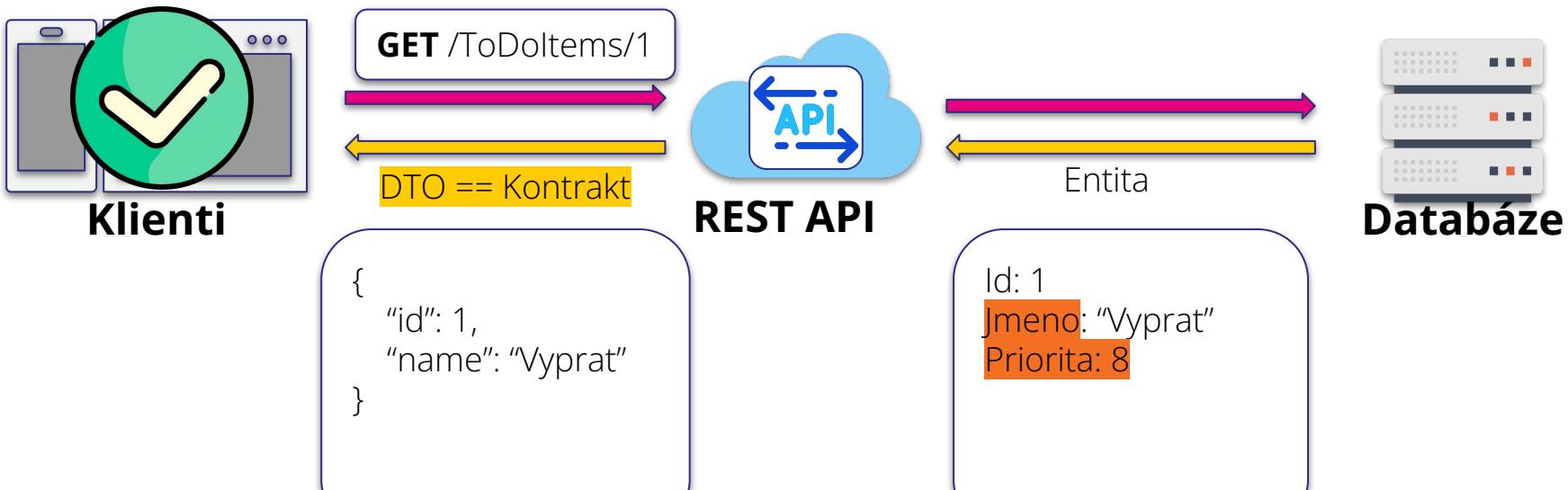
Proč používat Data Transfer Objects?



Proč používat Data Transfer Objects?



Proč používat Data Transfer Objects?



DTO se chová jako kontrakt, který definuje očekávání a požadavky na to, jak se budou posílat a přijímat data mezi klientem a serverem.

LEKCE 04

Testování kódu



```
git pull template main --allow-unrelated-histories
```

Lekce 04 - Testování kódu

0. Routing, DTO
1. Co je to **Unit Testing**
2. Testovací Frameworky
3. Jak psát testy?
4. Jak unit testovat REST API controller pomocí **xUnit**
5. Co je to Test Driven Development (**TDD**)
6. TDD v praxi
7. Mockování závislostí pomocí **NSubstitute**

Naučíme se testovat kód, odhalíme benefity testování
a zjistíme jak používat TDD metodiku testování ;)

Routing

Co je to Routing?

Routing rozhoduje, jaká akce nebo část kódu se má spustit na základě cesty zadané v URL a podle použité HTTP metody (GET, POST, PUT, DELETE apod.).

DTO

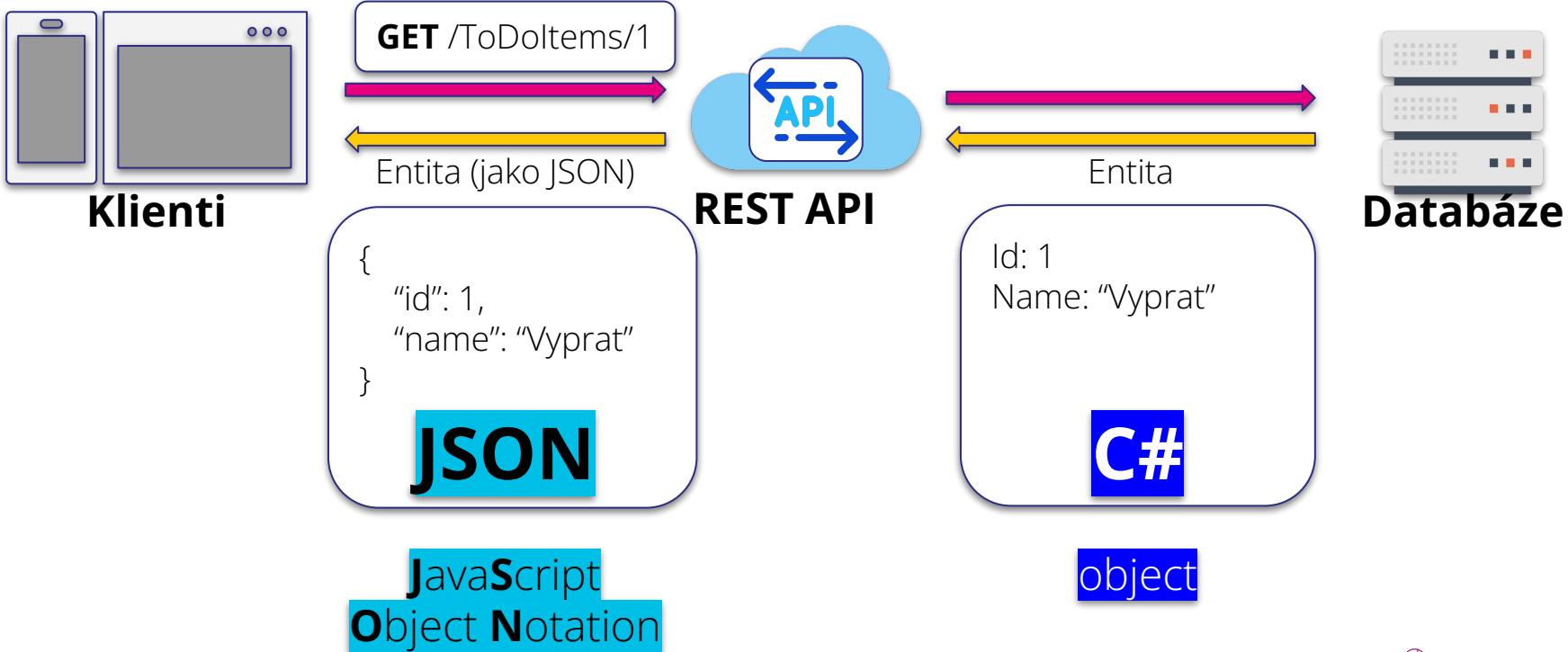
Data Transfer Object

Co je to DTO?

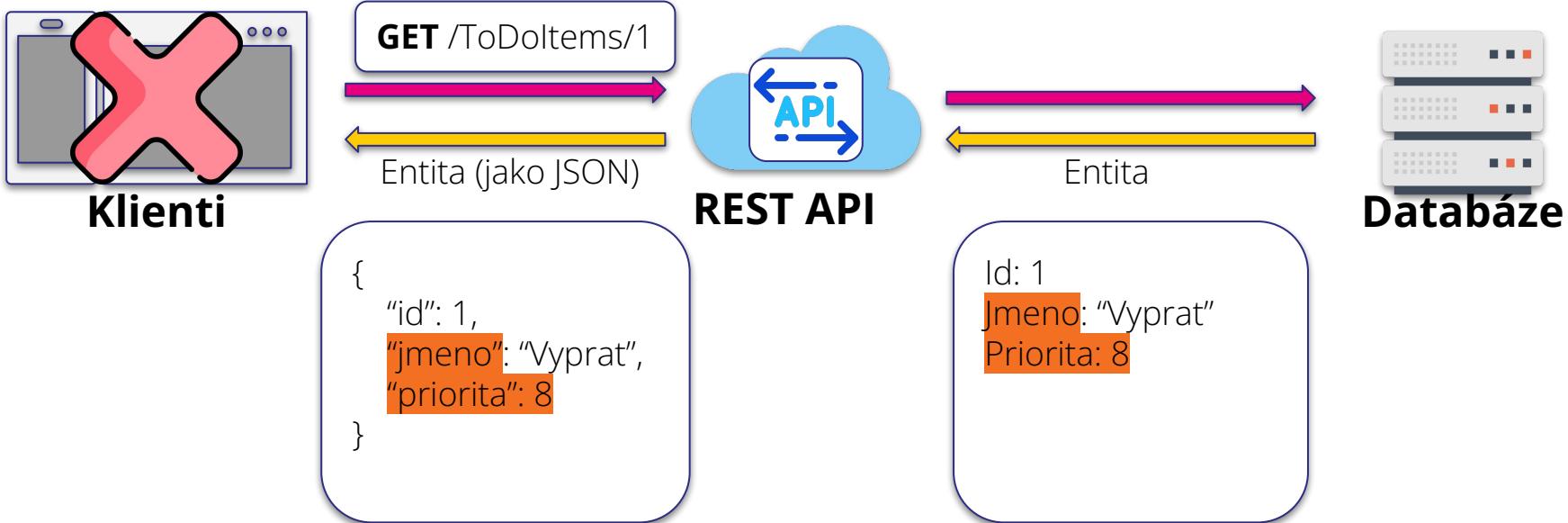
Data Transfer Object (DTO) je takový objekt, který přenáší data mezi procesy nebo aplikacemi.

V kontextu REST API, DTO může být považován jako kontrakt mezi klientem a serverem

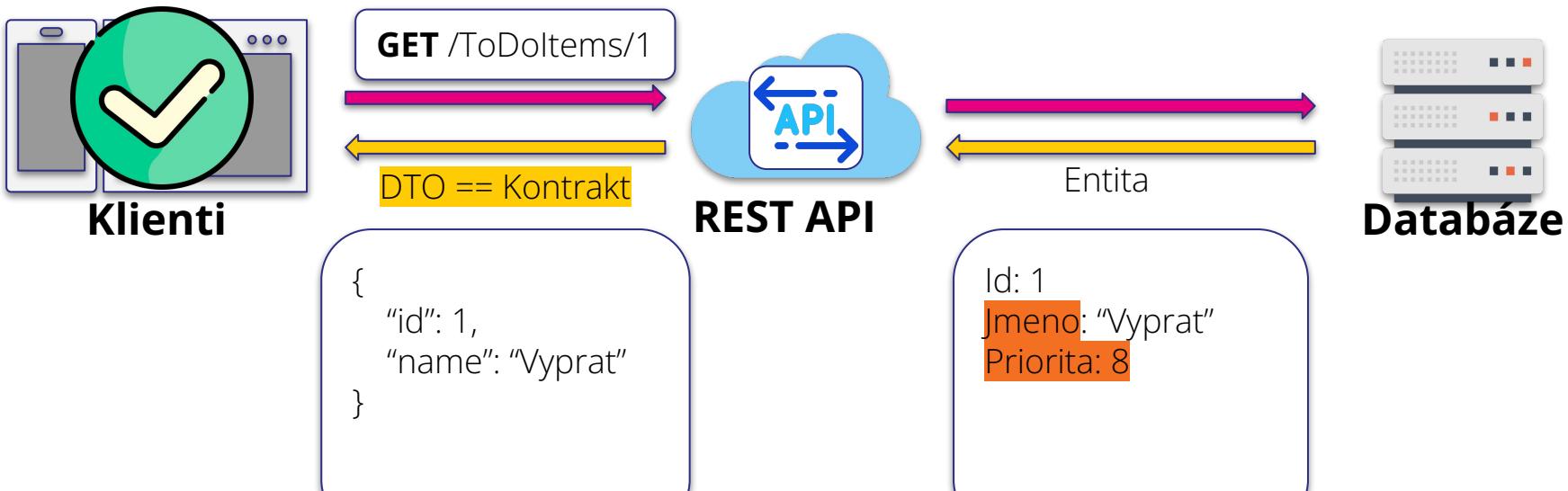
Proč používat Data Transfer Objects?



Proč používat Data Transfer Objects?



Proč používat Data Transfer Objects?



DTO se chová jako kontrakt, který definuje očekávání a požadavky na to, jak se budou posílat a přijímat data mezi klientem a serverem.

Testování

Pojďme otestovat tuto raketu

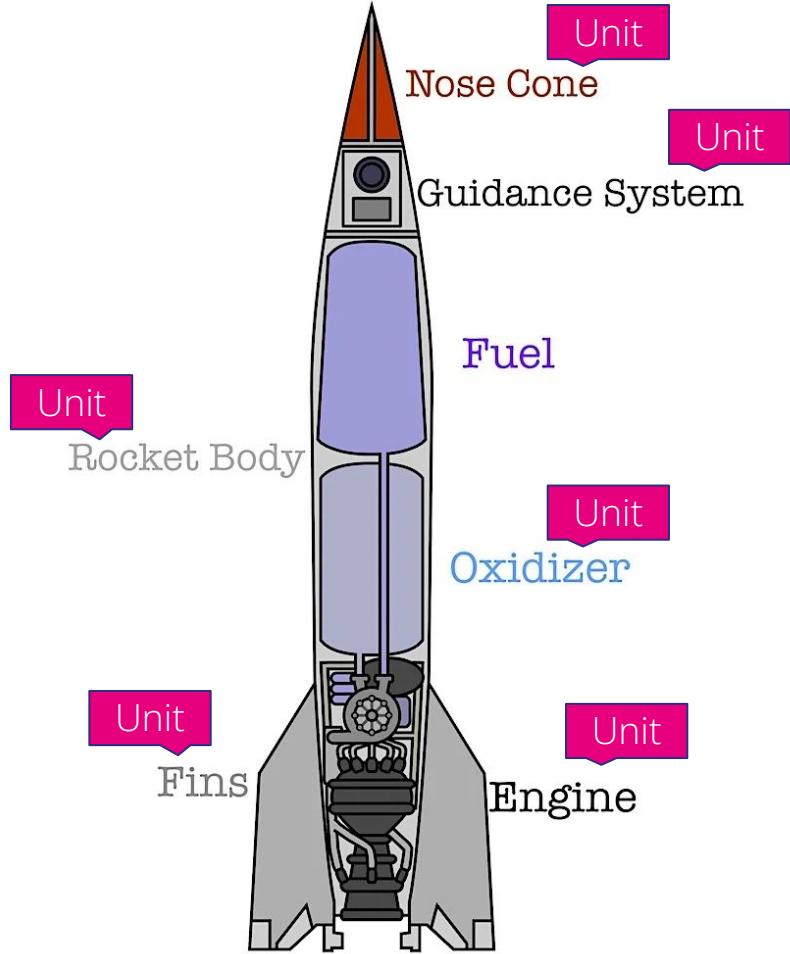


Co by se mohlo ukázat?

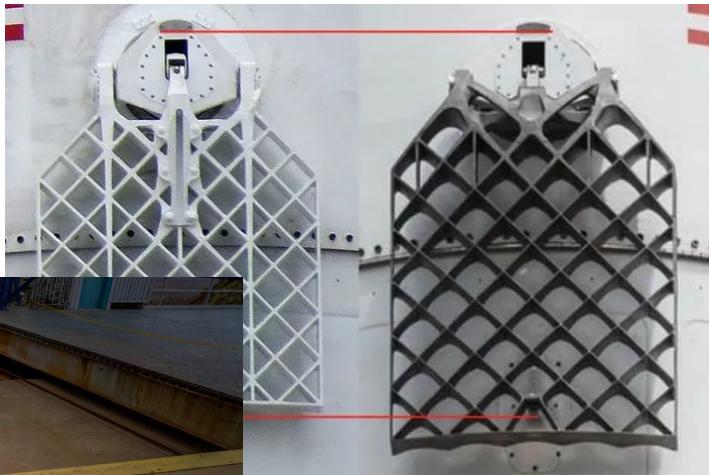




Unit Testing



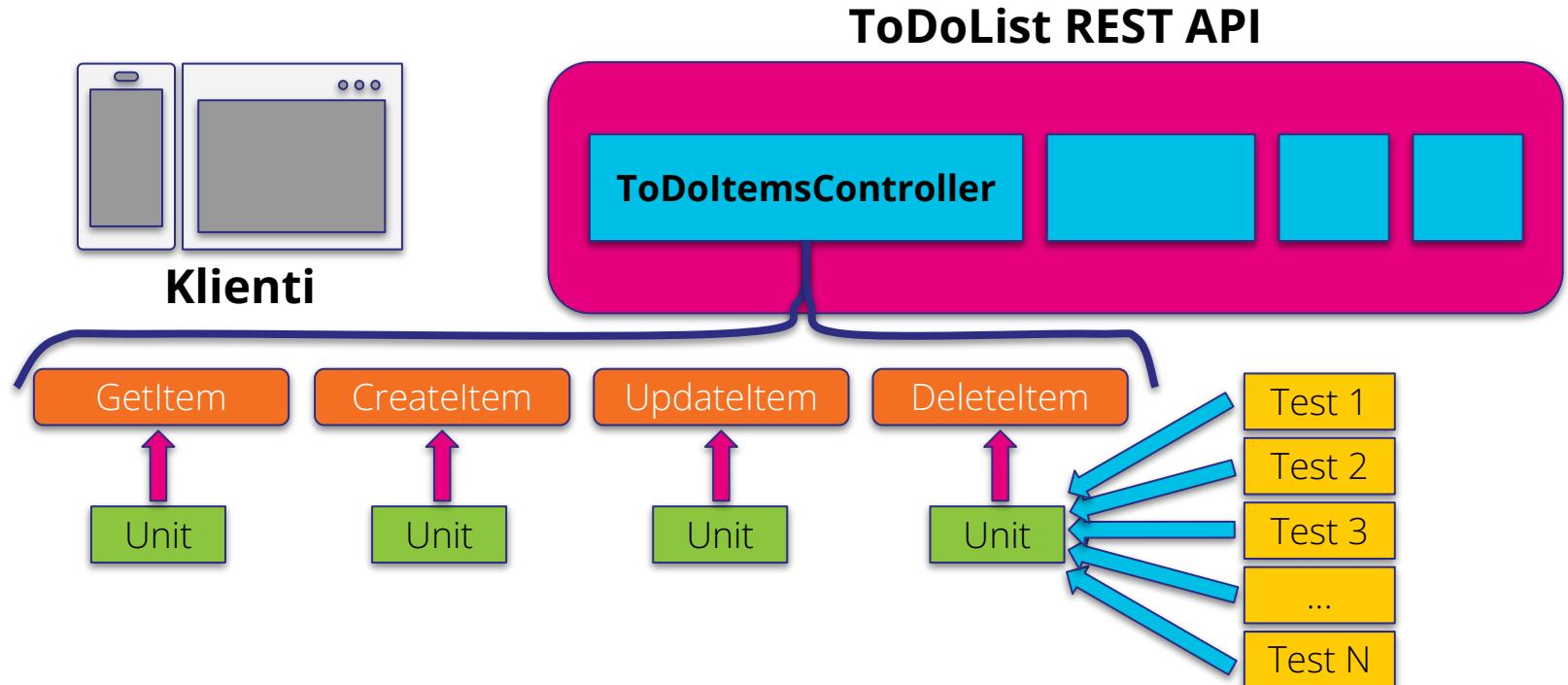
Units



Co je to Unit Testing?

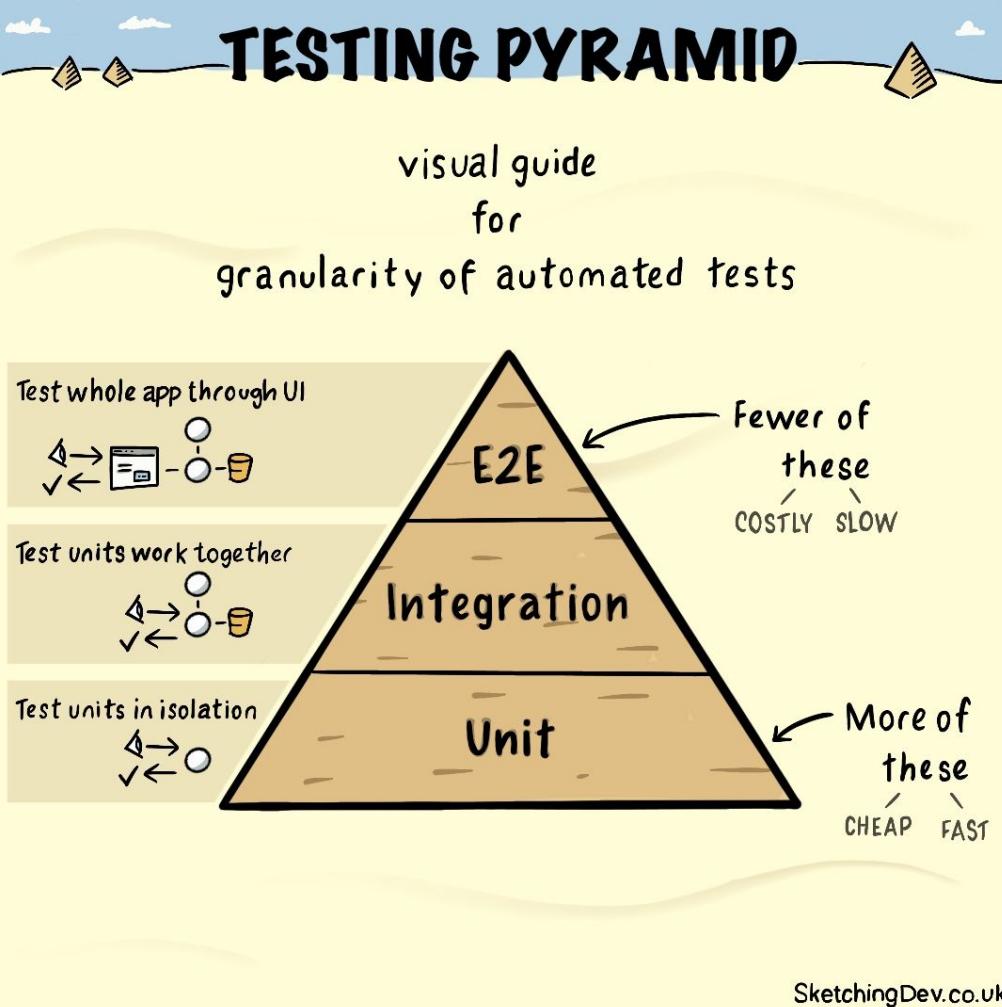
*Unit Testing je testování kódu,
kdy každá část kódu je testována v izolaci bez externích
závislostí*

Unit Testing REST API



Proč unit testovat?

- Rychlá kontrola správnosti kódu
- Možnost dělat změny v kódu bez obav
- Snížení dopadu chyb
- Živá dokumentace a specifikace



Testovací frameworky

Testovací frameworky v .NET



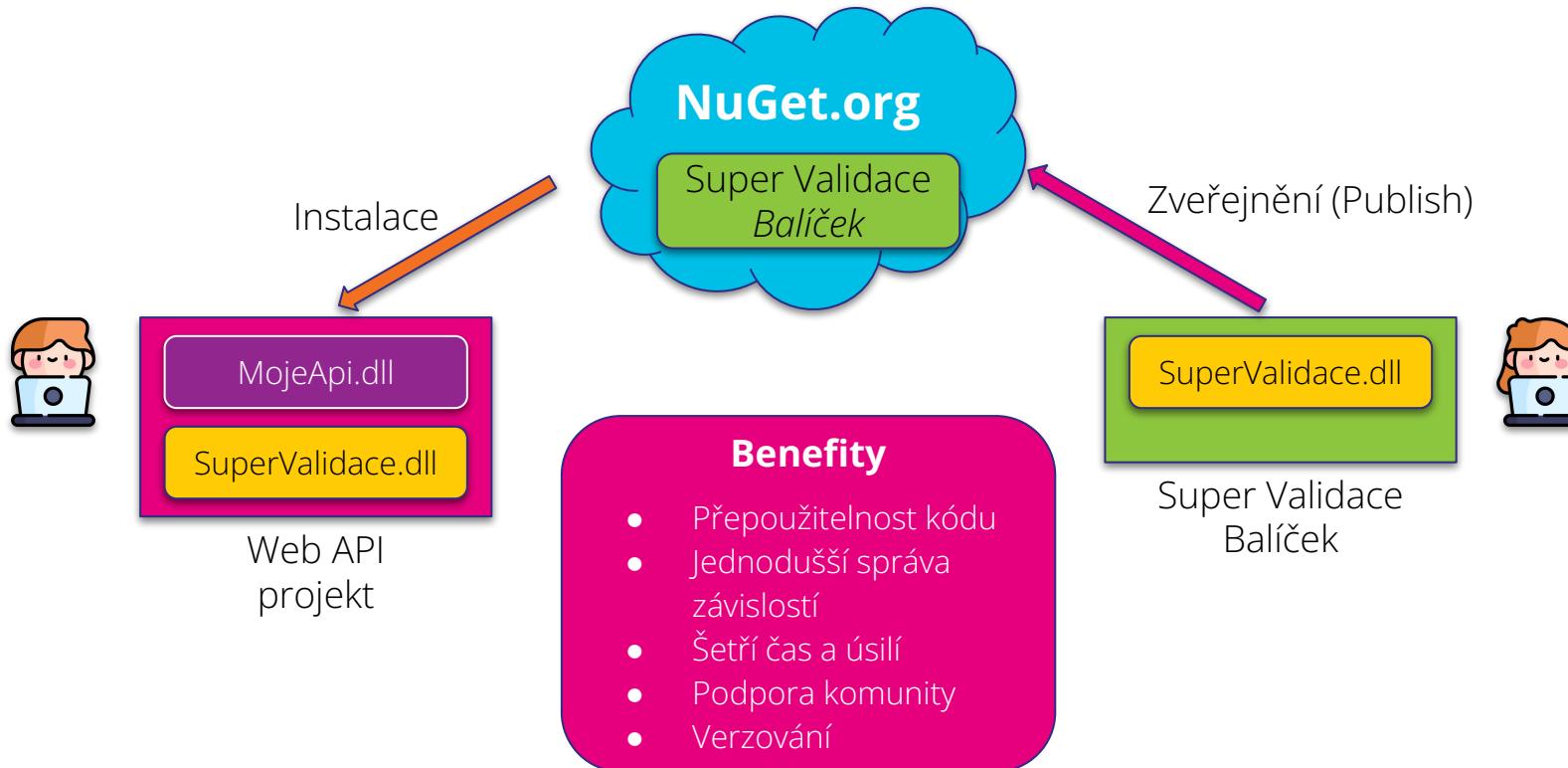
MSTest

xUnit.net

Má
preference

NuGet

Jak sdílet kód v .NET?



Jak psát testy?

AAA pattern

Arrange

Příprava kódu pro test

Act

Vykonání akce, kterou chceme testovat

Assert

Kontrola výstupu vykonané akce

AAA pattern - ukázka v xUnit

[Fact]

```
public CalculatorSumReturnsCorrectSumOfThreeNumbers()
{
    // Arrange
    var calculator = new Calculator();

    // Act
    var result = calculator.Sum(1,2,3);

    // Assert
    Assert.Equals(result, 6);
    //result.Should().Be(6); //FluentAssertions
}
```

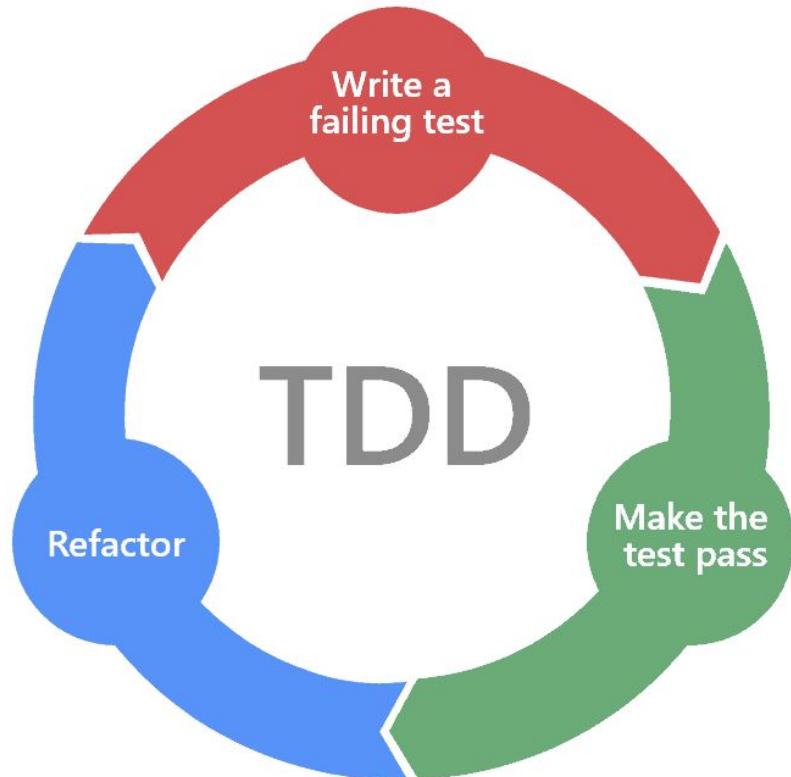
TDD

Test Driven Development

Co je to Test Driven Development (TDD)?

TDD je přístup k vývoji software, který je založen na psaní testů před napsáním logiky produkčního kódu.

Psaním testů definujeme funkcionality a chování systému, kterou následně implementujeme a napsanými testy ověřujeme.



Jak praktikovat TDD?

1. **Vytvoř nový test** reprezentující chtěnou funkčnosti/chování
2. **Spusť testy**, nový test by neměl procházet, všechny ostatní ano
3. Napiš **nejjednoduší možnou implementaci** tak, aby prošly všechny testy.
4. **Refactoring**

Dej kód kam logicky patří; Odstraňuj duplicity; Uprav názvy pokud neodpovídají kódu; Rozděl metody na menší;

1. **Opakuj znovu od kroku 1**

Proč praktikovat TDD?

- Vývoj je zaměřený na požadavky/funkcionalitu, né na implementaci
- Větší pokrytí produkčního kódu testy
- Chyby v kódu jsou díky testům odhaleny dříve
- Rovnou píšeme “testovatelný” kód
- Při změně požadavků se rychle dozvíme, co nefunguje.

Všeho s mírou!

Mocking

Co je to Mockování?

Mockování je proces vytváření objektů (Mocků), které simulují chování reálných objektů.

Tyto Mocky lze v testovacím prostředí upravovat a nastavovat jejich chování.

EXTRA: Typy zástupných objektů

- **Dummy**
- **Stub**
- **Spy**
- **Mock**
- **Fake**

Pro naše použití používáme Mocky!

Mockovací frameworky

Mocking frameworks v .NET



Tipy na závěr

Tipy

- Testujeme pouze jeden scénář
- Jméno testu mi přesně říká, co testuje (e.g. Metoda_Scenar_Vysledek)
- Nejjednodušší procházející test
- Konzistentní struktura
- Co nejméně logiky
- Debugger je kamarád
- [Theory]
- FluentAssertions

LEKCE 05

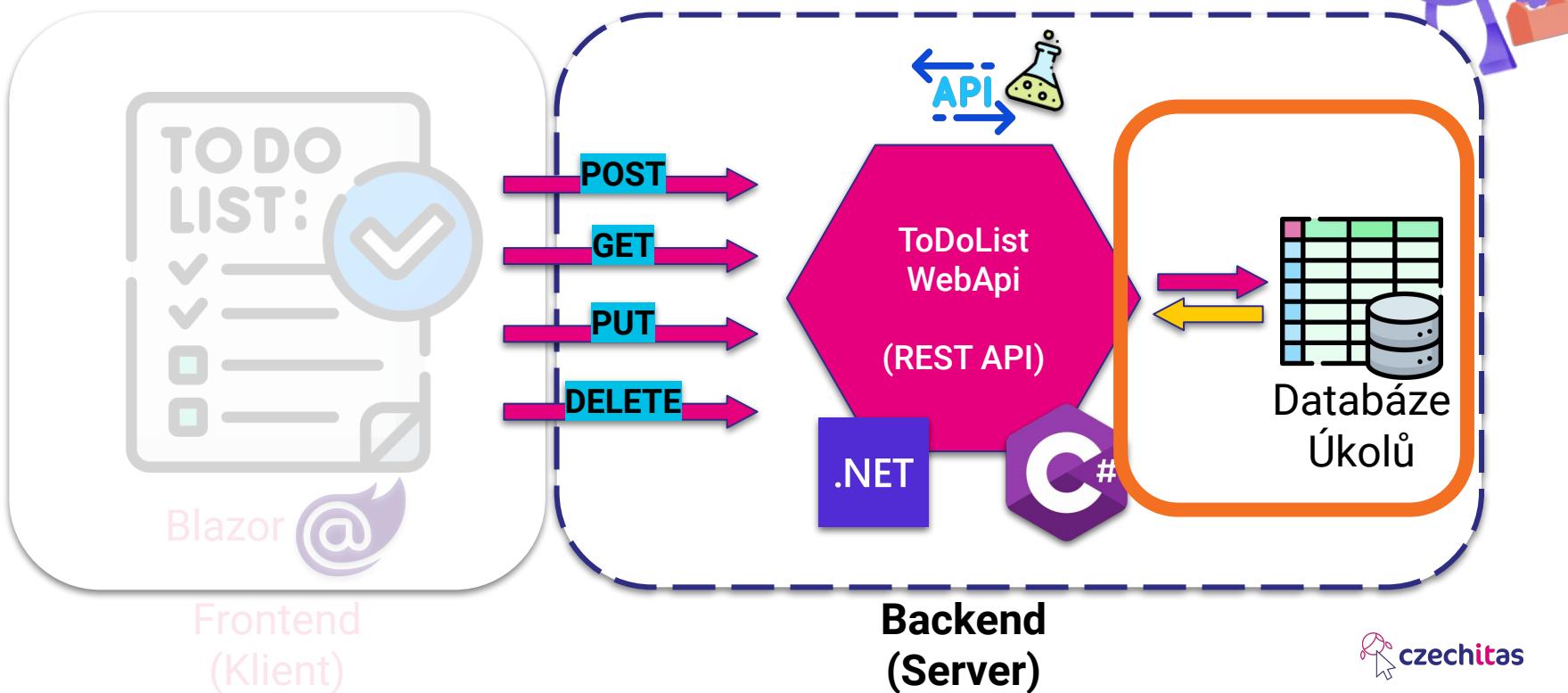
Databáze

Lekce 05 - Databáze

1. Co je to **databáze**?
2. **Tabulky, klíče, SQL**
3. Typy Databází
 - a. SQL
 - b. NoSQL
4. **O/RM**
 - a. **Entity Framework Core**

Seznámíme se s databázemi a SQL, víme co je to ORM a k čemu nám slouží.
Dokážeme se orientovat v základní práci s databázemi v projektu ;)

CO SPOLU POSTAVÍME



Database (DB)

Co je to databáze?

Databáze je organizovaná kolekce dat, která umožňuje efektivní uložení, načítání a správu informací.

Typické použití databází

- **Ukládání uživatelských dat** (např. e-shopy, webové aplikace)
- **Logistika** (sklady, přeprava)
- **Sledování transakcí** (banky, systémy e-commerce)
- **Skladování velkého množství dat** (analýzy, big data)
- **Blog** (články, komentáře, reakce)

Jak vypadá taková databáze?

TABLE

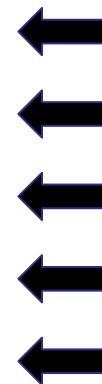
id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

COLUMNS



id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

ROWS



Keys

Primary & Foreign

Primární klíč

PRIMARY KEY



id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

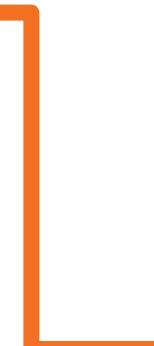
id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní

Cizí klíč

FOREIGN KEY



id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1



id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní

TASKS

PRIMARY

FOREIGN

id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

CATEGORIES

PRIMARY

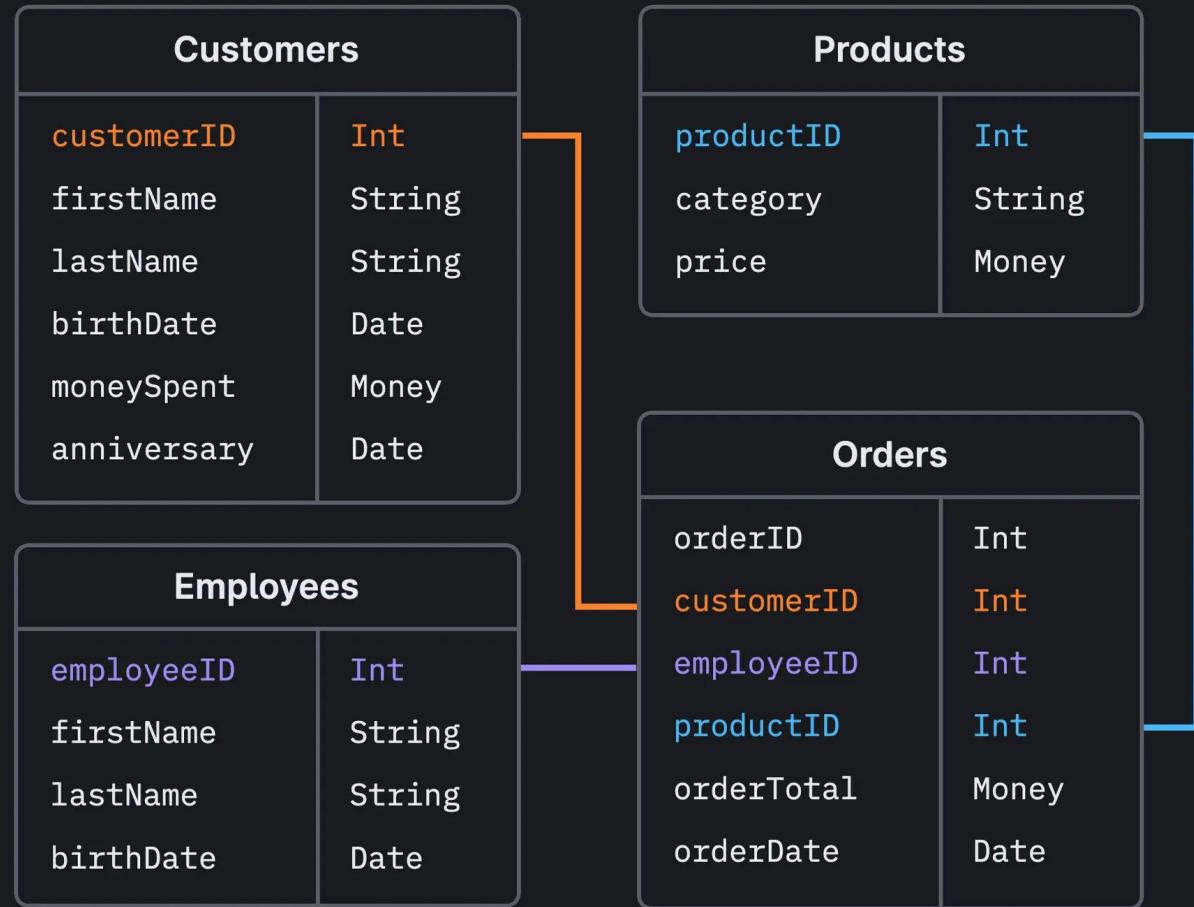
id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní



Jaké existují typy klíčů?

Primary Key: Unikátní identifikátor každého řádku v tabulce

Foreign Key: Sloupec, který odkazuje na primární klíč v jiné tabulce.

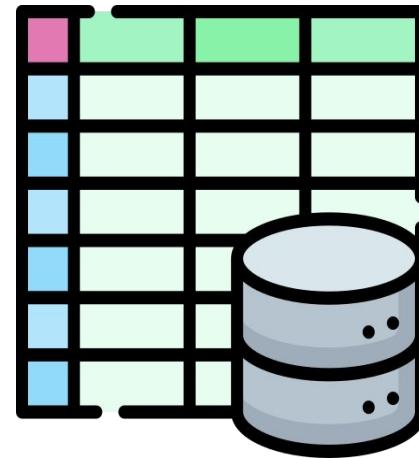
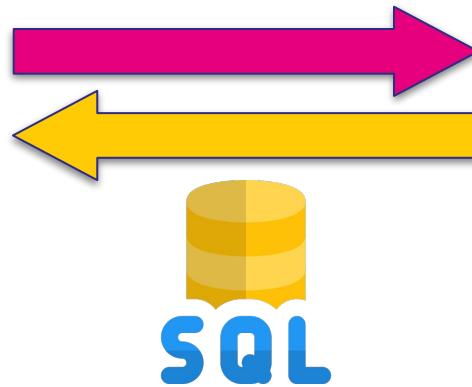
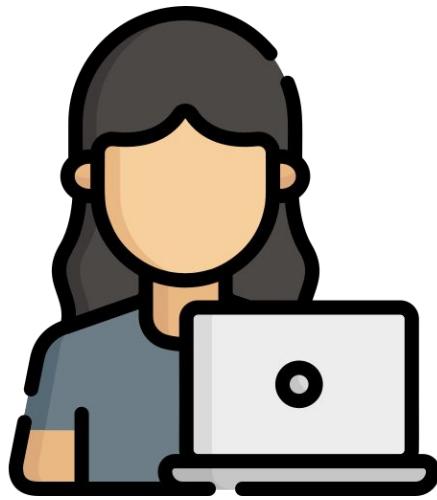


SQL

Structured Query Language

[sequel]

Jak interagovat s databází?



TASKS

FOREIGN		
id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

CATEGORIES

PRIMARY	
id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní



Ukázka

Vyber všechny úkoly, jejichž kategorie je Domácí práce

```
SELECT * FROM Tasks WHERE category_id = 1;
```

TASKS

FOREIGN		
id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

CATEGORIES

PRIMARY	
id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní



TASKS

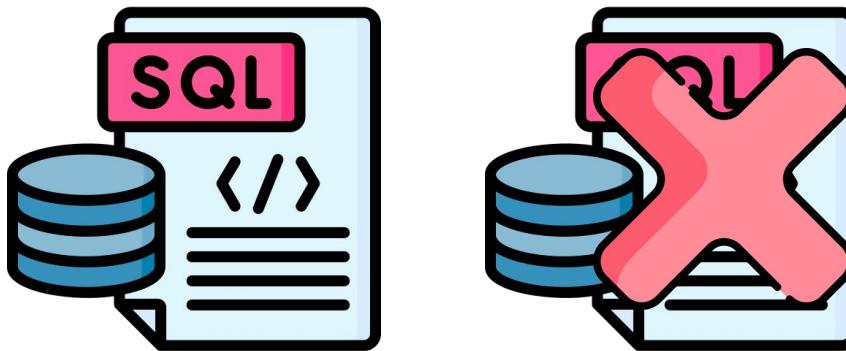
FOREIGN		
id	name	category_id
1	Vytisknout fakturu	5
2	Zubař	4
3	Vyzvednout balík	3
4	Nakoupit	2
5	Vysávat	1

CATEGORIES

PRIMARY	
id	name
1	Domácí práce
2	Jídlo
3	Pochůzky
4	Zdraví
5	Firemní

SQL vs NoSQL

V čem je ten rozdíl?



Zastoupení na trhu



Co použijeme v projektu?



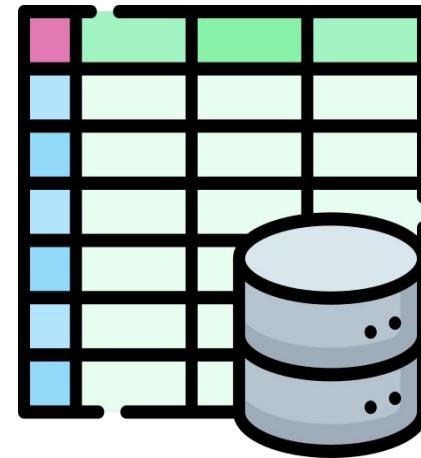
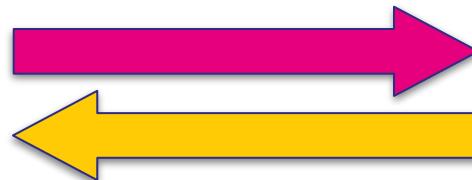
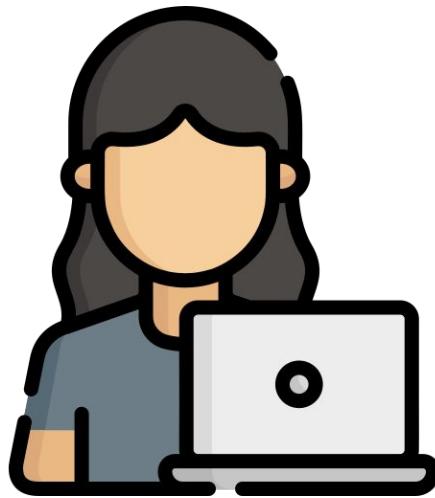
O/RM

Object-Relational Mapping

Co je to ORM?

*Technika, která propojuje objekty z objektově orientovaného programovacího jazyka s databázovými tabulkami**

Jak interagovat s databází??

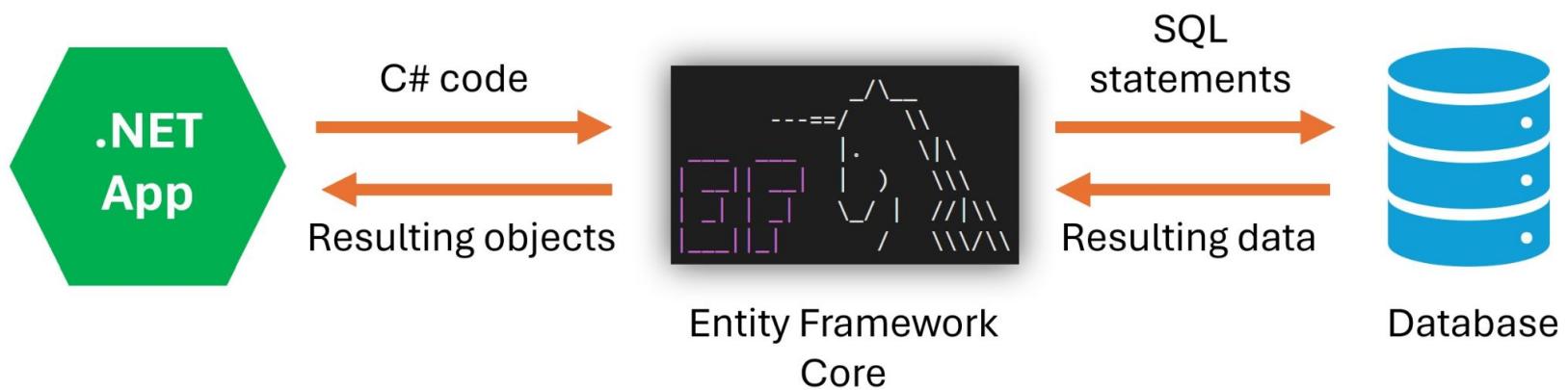


Co použijeme?

Entity Framework



Jak to vypadá podrobněji



LEKCE 06

Generika + Repository pattern

Zpětná vazba 🙏

Lekce 06

1. Generika
2. Repository pattern
3. Mockování v praxi

Generika

Což takhle rovnou příklad?

Definice generiky

- ≈ Generické typy nejsou specifické k určitému typu
- ≈ Parametrizované typy

Co je to generika?

- Generické mohou být třídy, metody, rozhraní...
- Bere ≥ 1 typových parametrů
- Nejlepší kombinace:
 - přepoužitelnosti - nemusíme duplikovat kód pro různé typy
 - silného typování - odlišné typy vyhodí kompilační chybu
 - výkonu - neboxujeme hodnotové typy

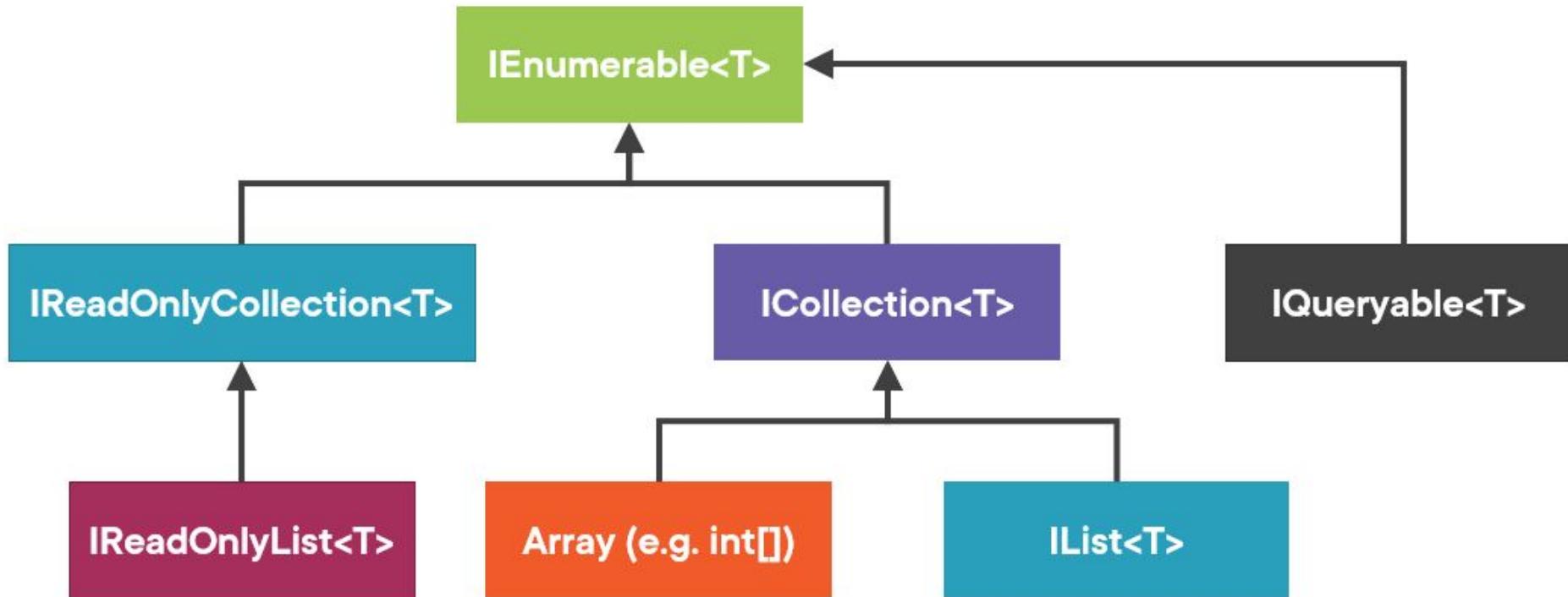
Where omezení

- Nechceme vždy povolovat všechny typy
- Silnější typování
- Můžeme využívat metody constrained typu

Where omezení

Constraint	Description
<code>where T : struct</code>	The type argument must be a non-nullable value type, which includes <code>record</code> <code>struct</code> types. For information about nullable value types, see Nullable value types . Because all value types have an accessible parameterless constructor, either declared or implicit, the <code>struct</code> constraint implies the <code>new()</code> constraint and can't be combined with the <code>new()</code> constraint. You can't combine the <code>struct</code> constraint with the <code>unmanaged</code> constraint.
<code>where T : class</code>	The type argument must be a reference type. This constraint applies also to any class, interface, delegate, or array type. In a nullable context, <code>T</code> must be a non-nullable reference type.
<code>where T : new()</code>	The type argument must have a public parameterless constructor. When used together with other constraints, the <code>new()</code> constraint must be specified last. The <code>new()</code> constraint can't be combined with the <code>struct</code> and <code>unmanaged</code> constraints.
<code>where T : <base class name></code>	The type argument must be or derive from the specified base class. In a nullable context, <code>T</code> must be a non-nullable reference type derived from the specified base class.
<code>where T : <interface name></code>	The type argument must be or implement the specified interface. Multiple interface constraints can be specified. The constraining interface can also be generic. In a nullable context, <code>T</code> must be a non-nullable type that implements the specified interface.
<code>where T : U</code>	The type argument supplied for <code>T</code> must be or derive from the argument supplied for <code>U</code> . In a nullable context, if <code>U</code> is a non-nullable reference type, <code>T</code> must be a non-nullable reference type. If <code>U</code> is a nullable reference type, <code>T</code> can be either nullable or non-nullable.
<code>where T : default</code>	This constraint resolves the ambiguity when you need to specify an unconstrained type parameter when you override a method or provide an explicit interface implementation. The <code>default</code> constraint implies the base method without either the <code>class</code> or <code>struct</code> constraint. For more information, see the default constraint spec proposal .

Generické kolekce

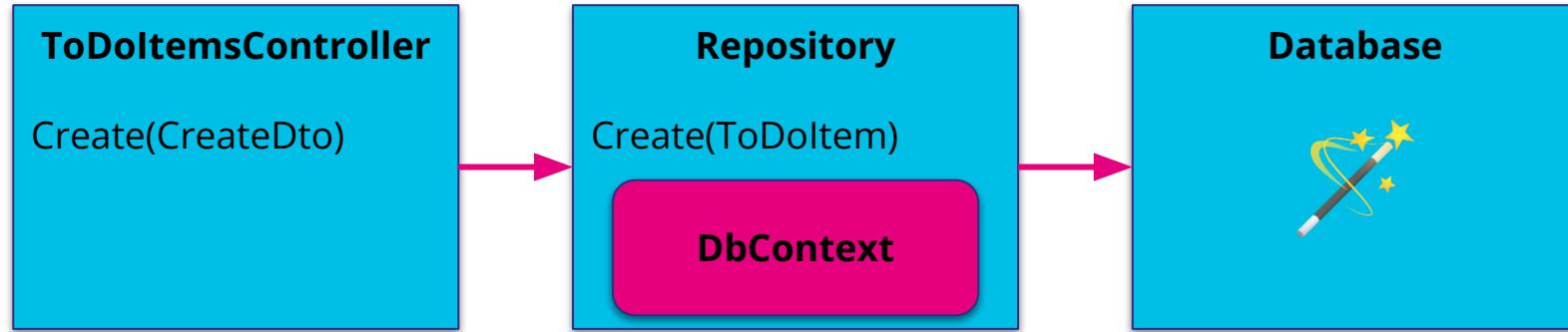
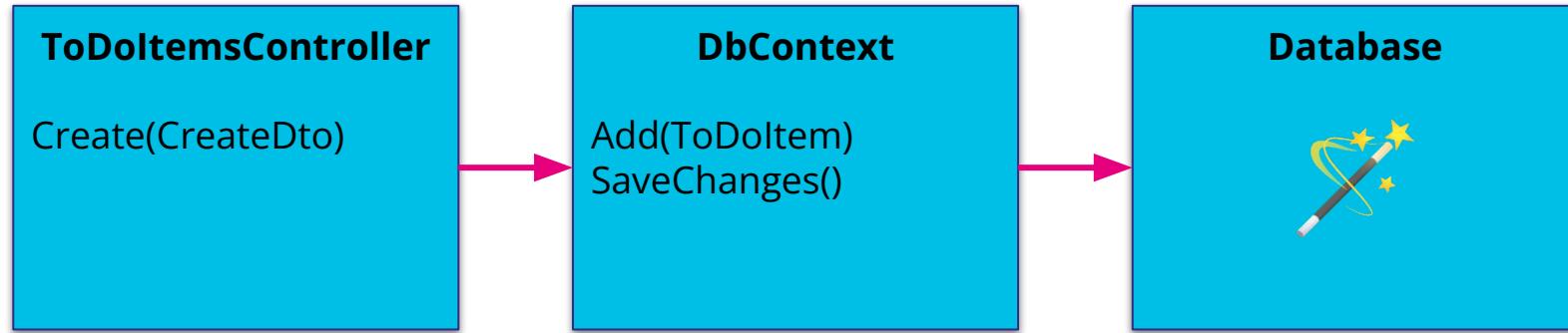


ArrayList

Všeho s mírou!

Repository

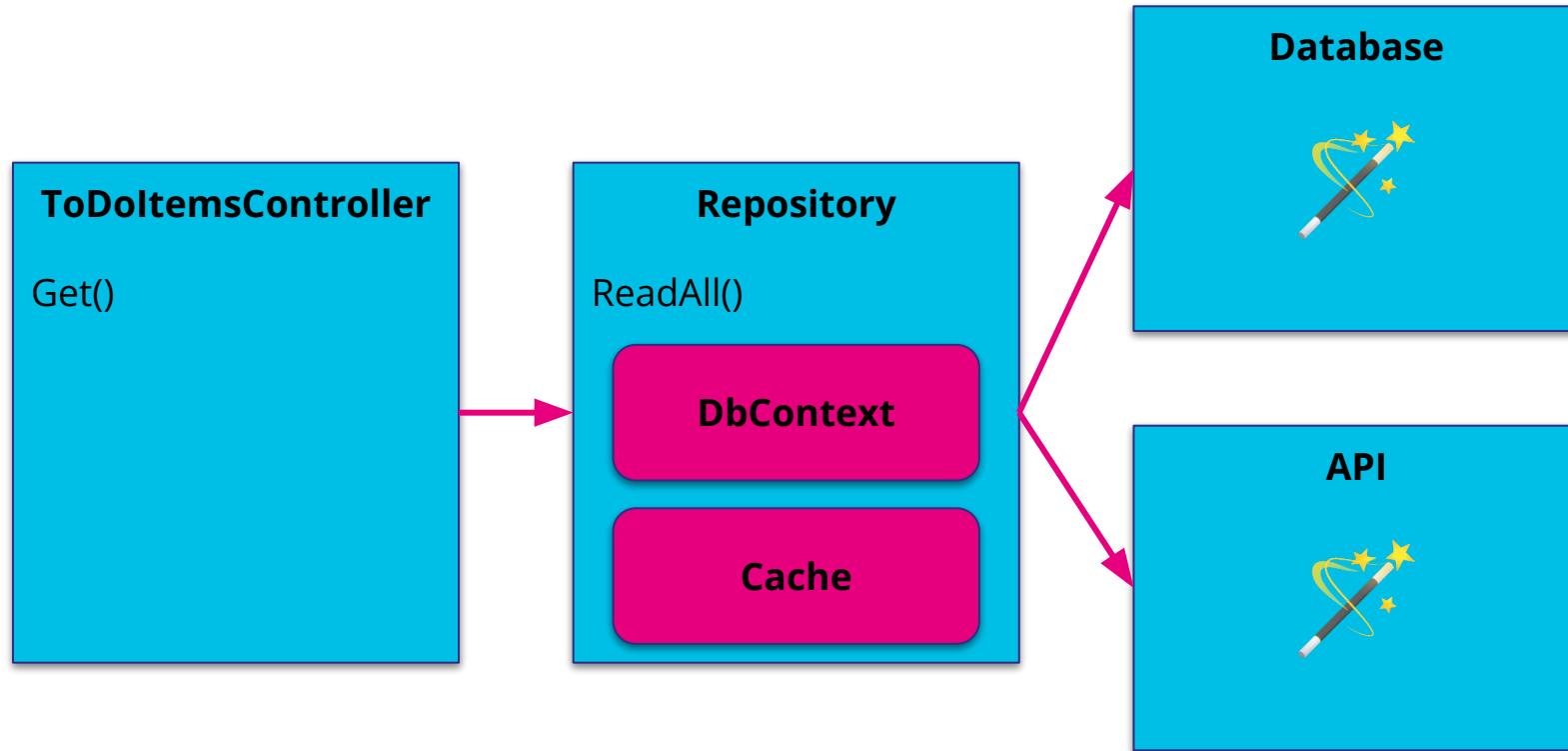
Repository pattern



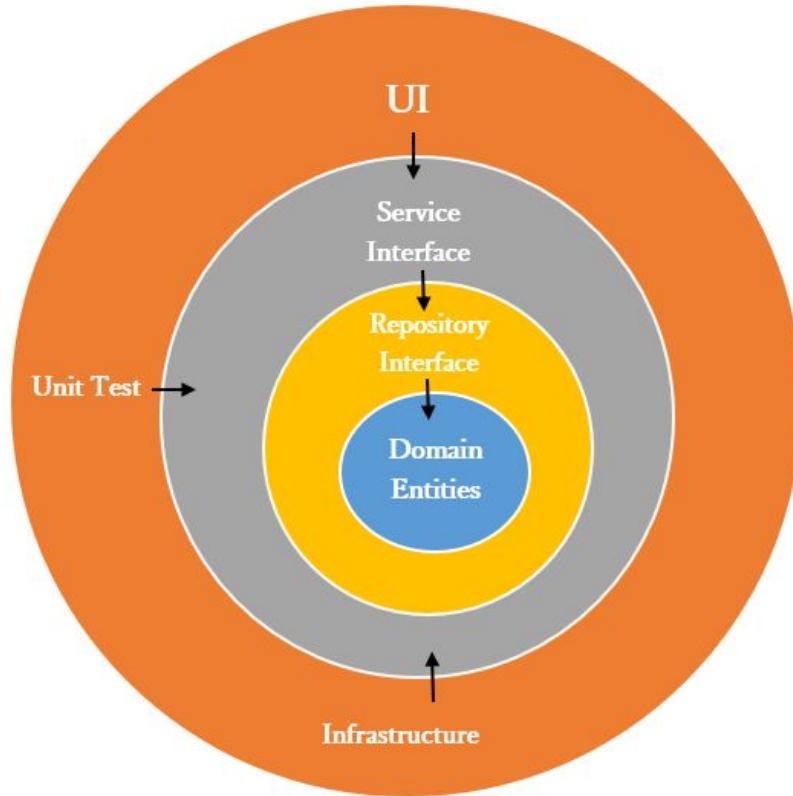
K čemu repository?

- Přepoužitelnost
- Centralizovaná logika
- Lepší testování
- “Oddělená” datová vrstva
- Může reprezentovat několik datových zdrojů
- Jednodušší vyměnit datový zdroj

Repository pro několik zdrojů



Cibulová architektura



IRepository<T>

Mocking

Co je to mockování?

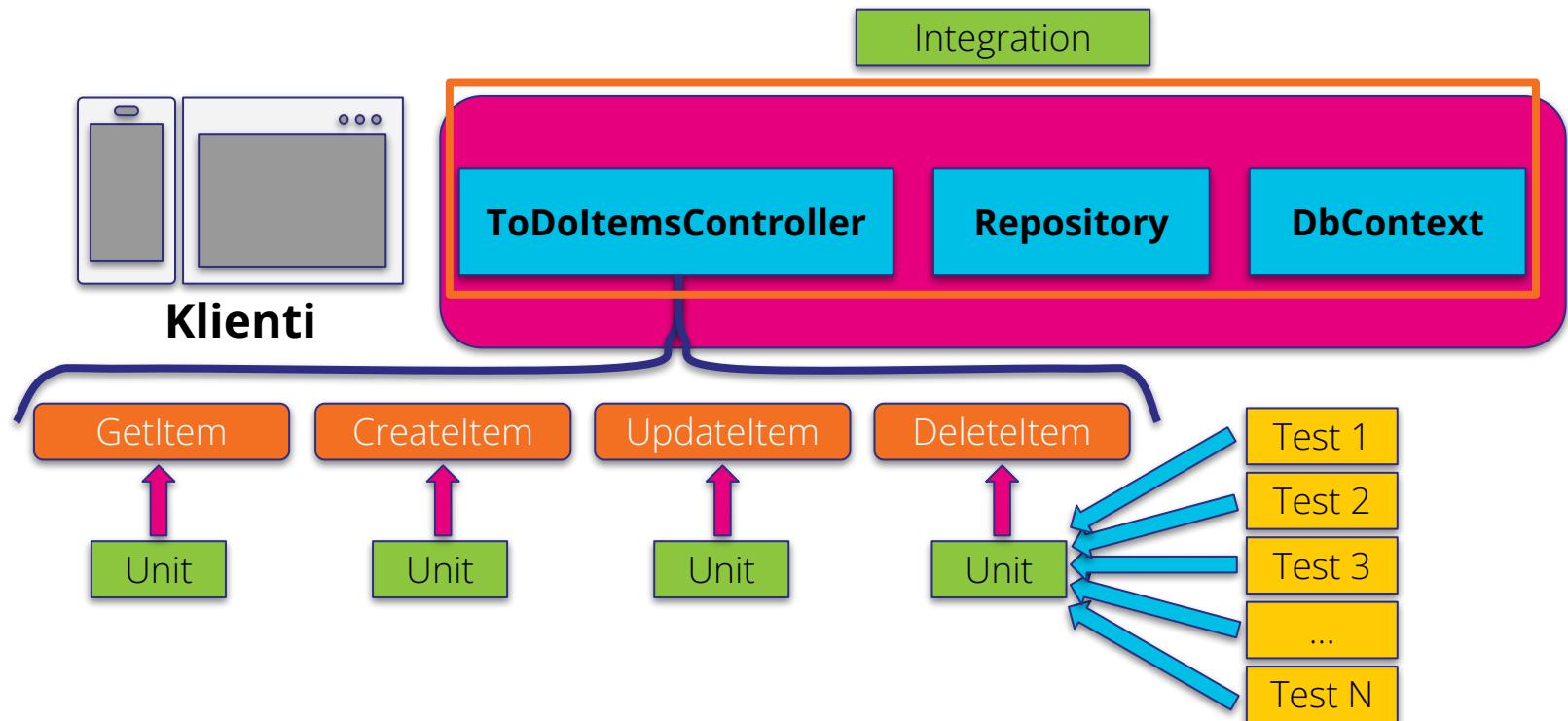
Mockování je proces vytváření objektů (Mocků), které simulují chování reálných objektů.

Tyto Mocky lze v testovacím prostředí upravovat a nastavovat jejich chování.

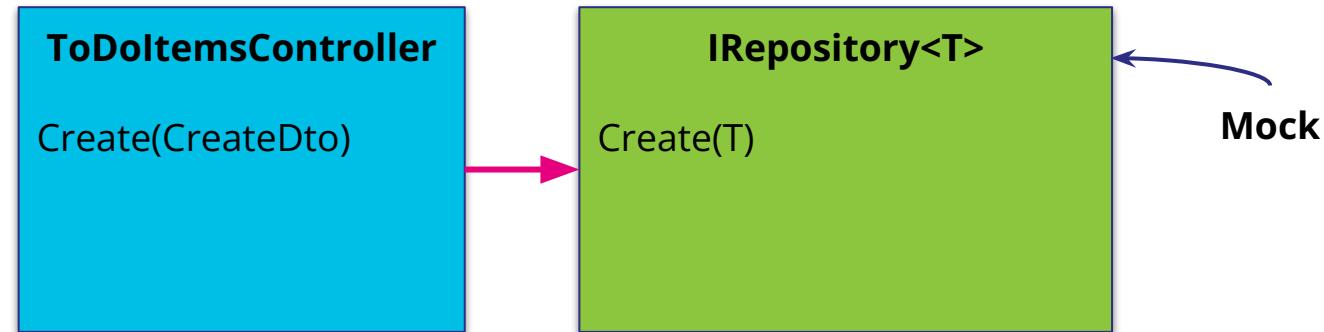
Výhody mockování

- Nemusíme testovat DB, jen naší logiku
- DB nám nerozbije testy
- Testy nespadnou při paralelním běhu

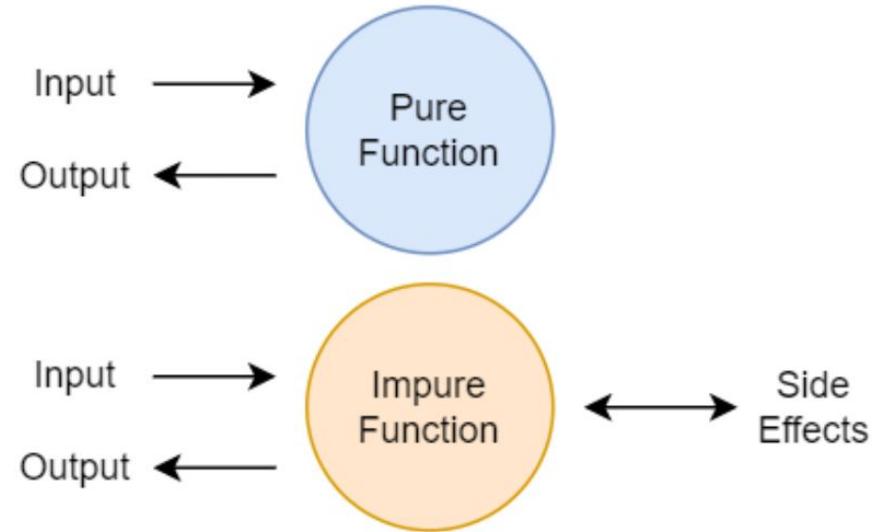
Unit Testing?



Mockování



Side effects



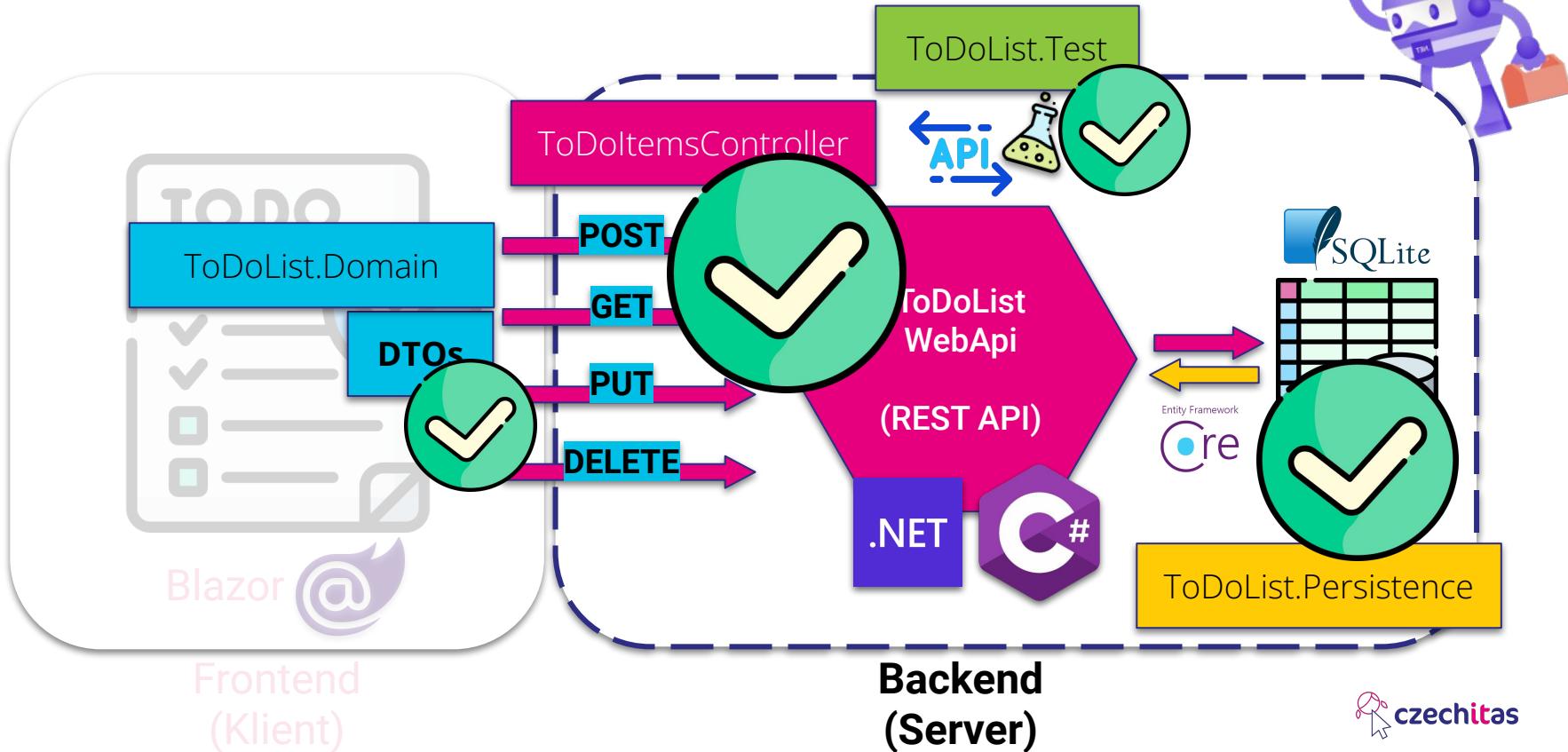
LEKCE 07

Opakování, Refactoring, Dependency Injection

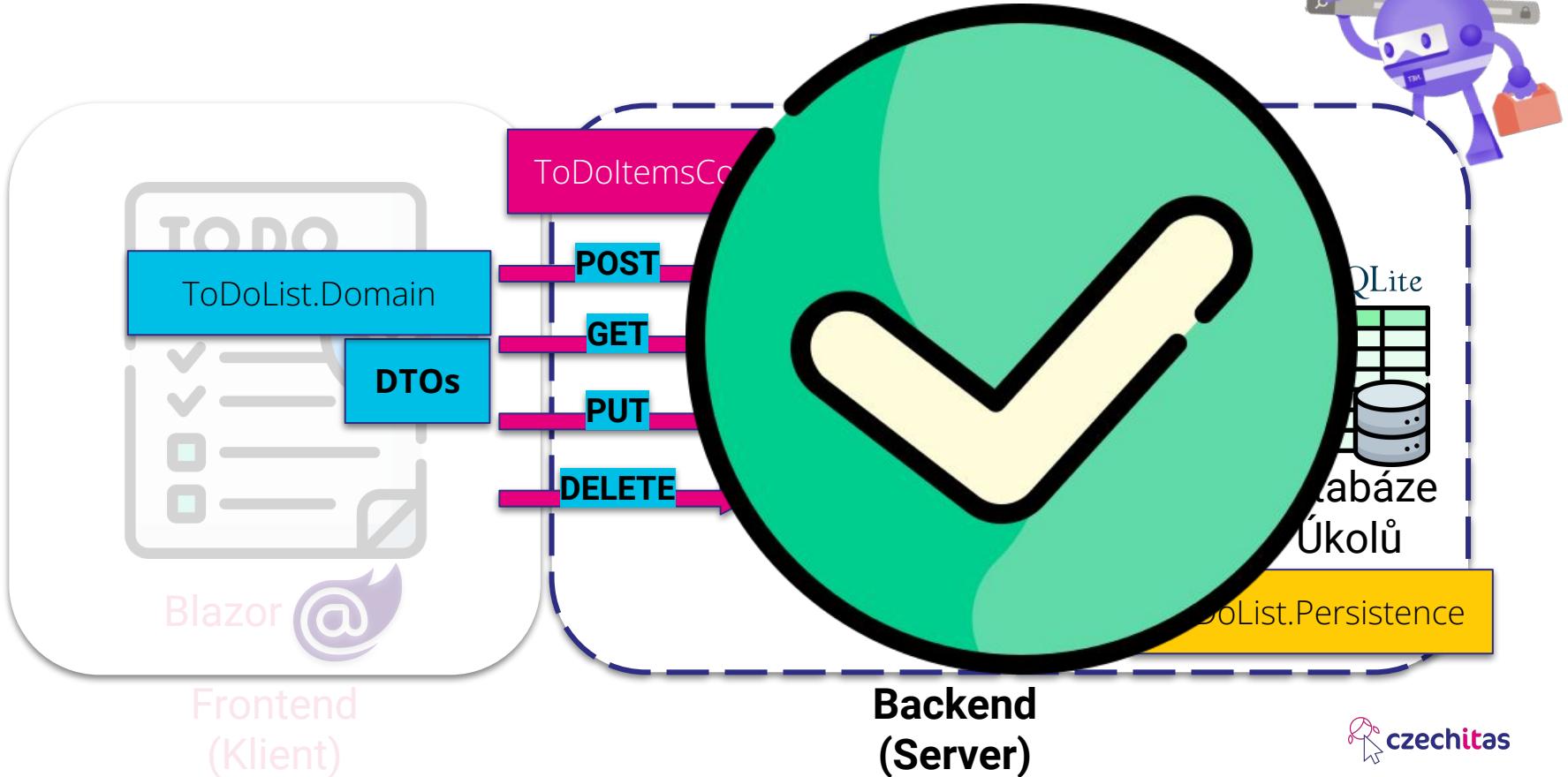
Lekce 07 - Opakování, Refactoring, Dependency Injection

1. Breakdown
2. Ukázka **řešení** domácího úkolu
3. Psaní unit testů s **NSubstitute**
 - a. Do().When()
 - b. Returns()
 - c. Received(1)
 - d. Throws()
4. **Dependencies**
5. **Dependency Injection**
6. **Dependency Inversion**
7. **Dependency Injection Containers** (`IServiceProvider`)

Breakdown



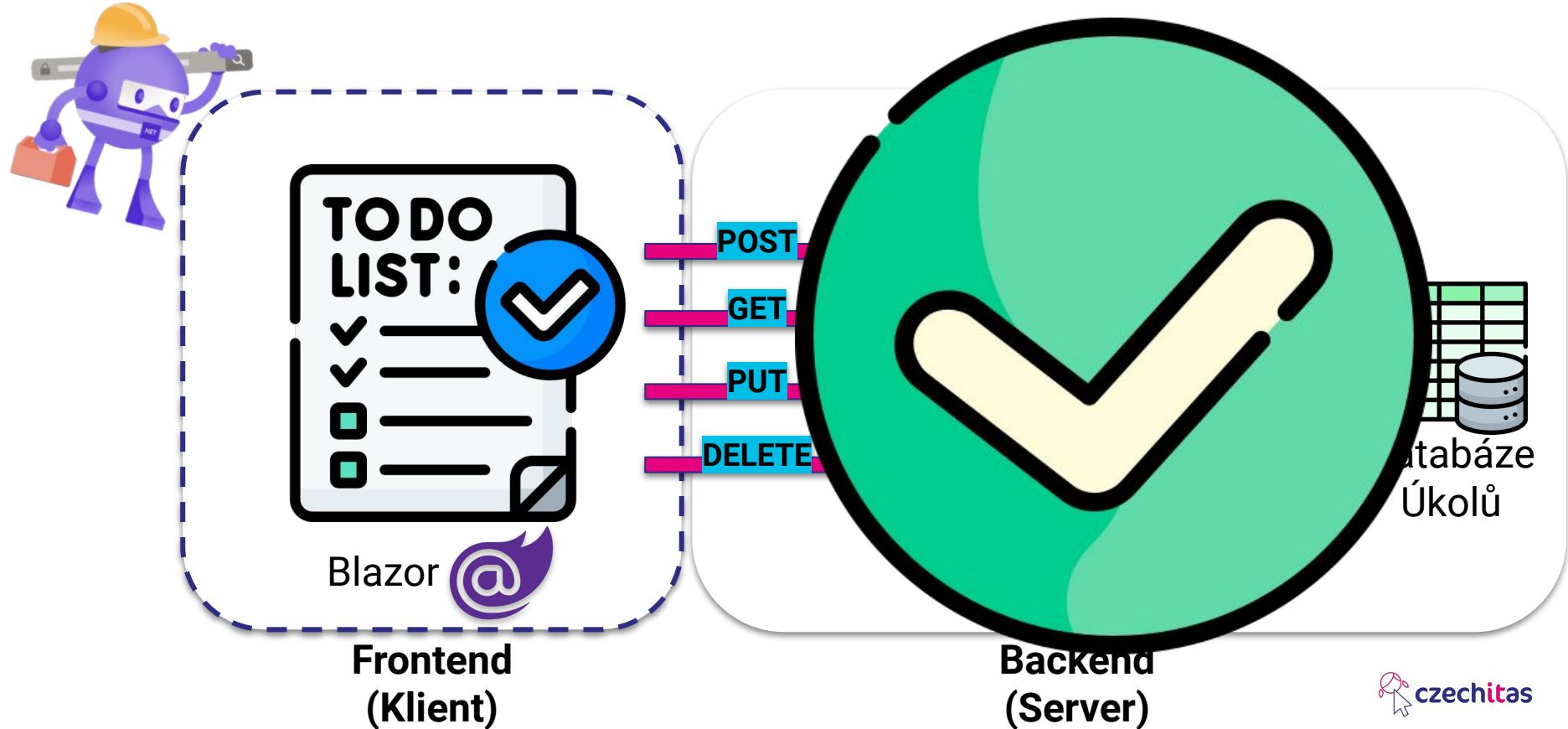
Breakdown



Frontend
(Klient)

Backend
(Server)

CO NÁS ČEKÁ



Co je to Mockování?

Mockování je proces vytváření objektů (Mocků), které simulují chování reálných objektů.

Tyto Mocky lze v testovacím prostředí upravovat a nastavovat jejich chování.

Závislosti

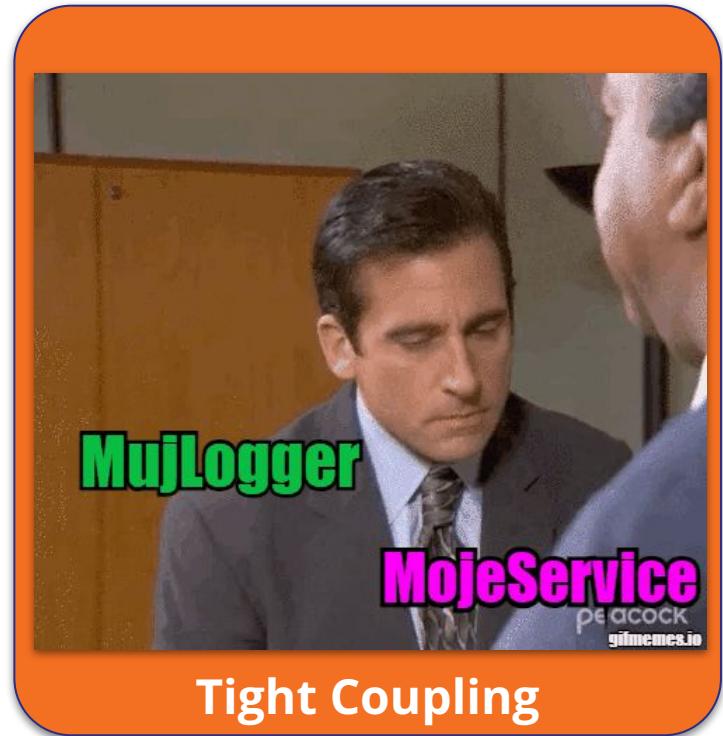
Dependencies

Co jsou to Dependencies?



```
public MojeService()
{
    var logger = new MujLogger();
    logger.Log("Jsem připraven!");
}
```

```
public MojeService()
{
    var writer = new MujFileWriter("output.log");
    var logger = new MujLogger(writer);
    logger.Log("Jsem připraven!");
}
```



Co jsou to Dependencies?



```
public ToDoltemsController()  
{  
    var repository = new ToDoltemsRepository(!!!);  
    repository.Create(item);  
}
```

Nevýhody

- ToDoltemsController je ovlivněna změnami v závislostech.
- ToDoltemsController musí vědět, jak se vytváří nebo konfigurují její závislosti.
- ToDoltemsController se nedá izolovaně testovat pomocí unit testů

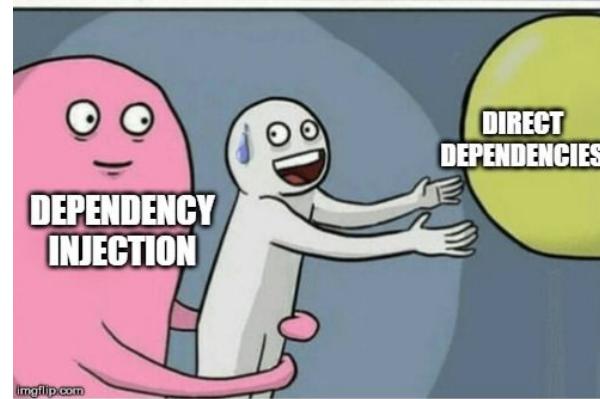
Co jsou Dependencies?

Na úrovni třídy

Závislost na úrovni třídy je vztah, kdy jedna třída potřebuje ke své funkčnosti instanci jiné třídy. Bez této třídy by nemohla správně fungovat nebo vykonávat určité úkoly.

Na úrovni projektu

Kus kódu nebo služba, na které aplikace spoléhá, aniž by ji musela sama implementovat (např. knihovny).



Dependency Injection

Co je to Dependency Injection?



```
public MojeService(MujLogger logger)
{
    logger.Log("Jsem připraven!");
}
```

ZÁVISLOST (DEPENDENCY)

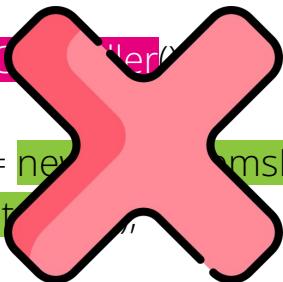
Benefity

- MojeService **není ovlivněna změnami na závislostech.**
- MojeService **nemusí vědět, jak se vytváří nebo konfigurují její závislosti.**
- Závislosti jde přidat, tzv. "**injectnout**" jako parametry konstruktoru.
- Umožňuje použití **Dependency Inversion.**

Použití Dependency Injection



```
public ToDoItemsController()
{
    var repository = new ToDoItemsRepository(!!!);
    repository.Create("!");
}
```

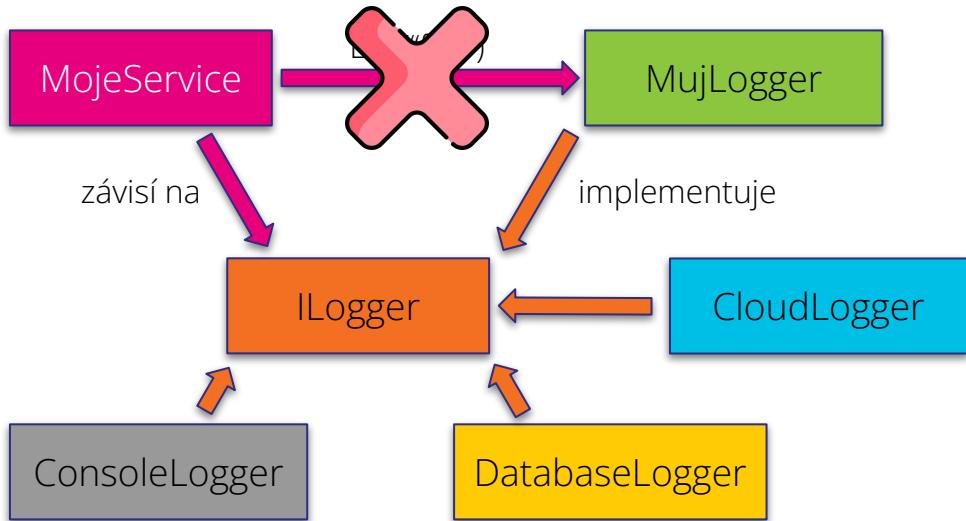


```
public ToDoItemsController(ToDoItemsRepository repository)
{
    repository.Create(item);
}
```



Dependency Inversion

Použití Dependency Inversion



```
public MojeService(ILogger logger)
{
    logger.Log("Jsem připraven!");
}
```

Princip Dependency Inversion

(Princip Inverze Závislostí)

"Kód by měl záviset na abstrakcích místo na konkrétních implementacích"

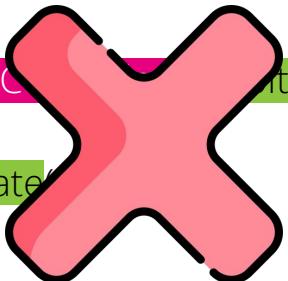
Benefity

- Závislost na loggeru může být nahrazena libovolnou jinou implementací, která implementuje interface **ILogger**, aniž bychom museli modifikovat MojeService.
- Je jednodušší testovat MojeService, jelikož závislost na **ILogger** je možno mockovat.
- Kód je čistější, jednodušší na úpravu a přepoužití

Použití Dependency Inversion



```
public ToDoItemsController(ToDoItemsRepository repository)  
{  
    repository.Create()  
}
```



A large red "X" is overlaid on the code, indicating that this implementation violates the principle of dependency inversion. The code is tightly coupled to a specific repository implementation.

```
public ToDoItemsController(IRepository<ToDoItems> repository)  
{  
    repository.Create(item);  
}
```

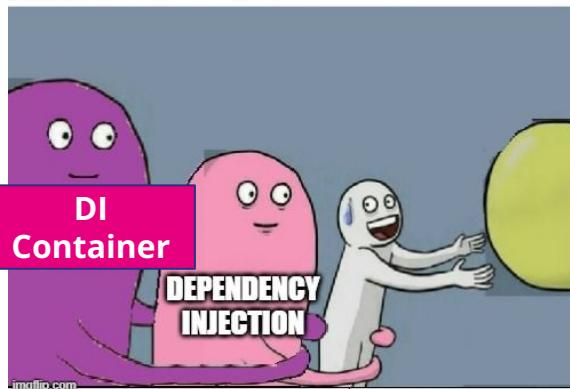
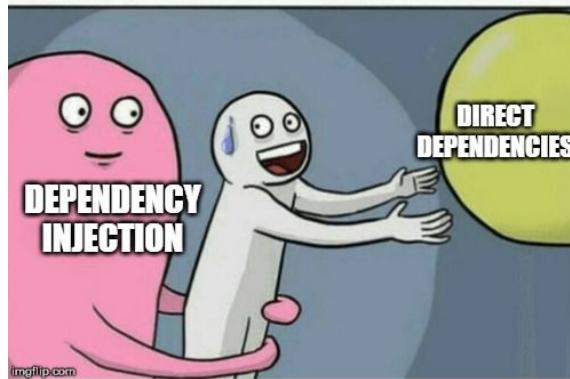
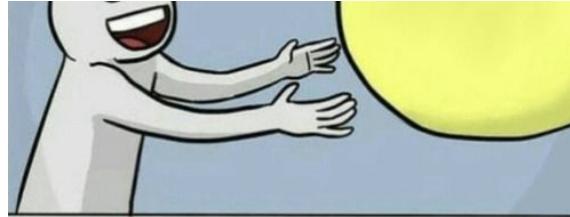
Princip Dependency Inversion (Inverze Závislostí)

"Kód by měl záviset na abstrakcích místo na konkrétních implementacích"

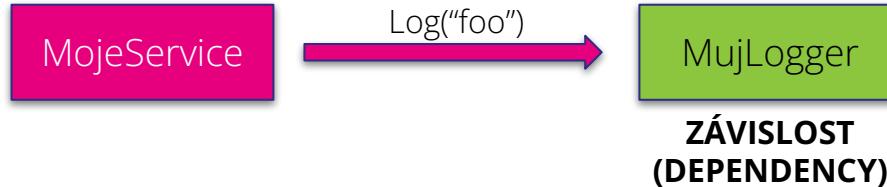


Dependency Injection Container (DI Container)

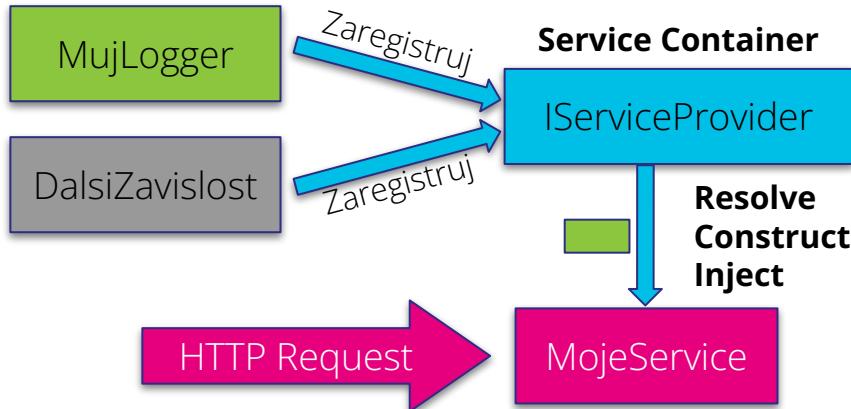
také IoC Container



Co je to DI Container?



```
public MojeService(MujLogger logger)
{
    logger.Log("Jsem připraven!");
}
```



Benefity

- Centralizace správy tříd, naši aplikaci řídíme z jednoho místa.
- IServiceProvider drží všechny závislosti.
- Lifecycle objektů je autonomně řízen kontejnerem (podle nastavení dané závislosti).
- To jaké třídy a závislosti se zaregistrují lze **konfigurovat**.

Použití DI Containeru

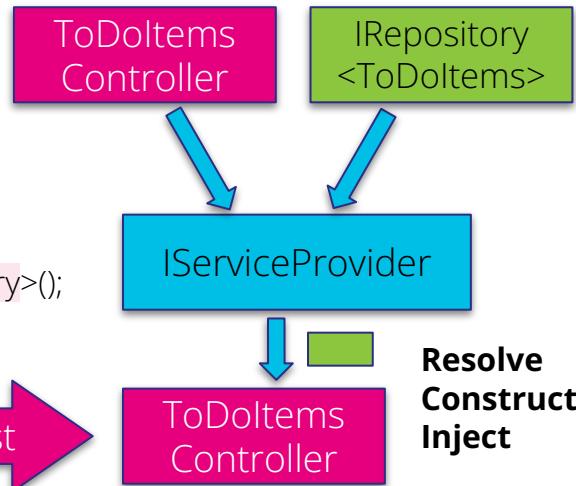


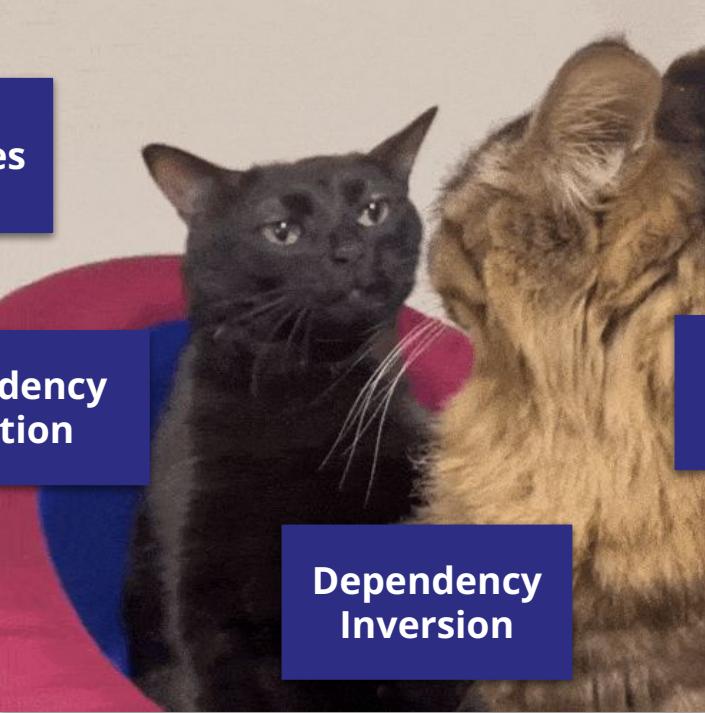
```
public ToDoItemsController(IRepository<ToDoItems> repository)
```

```
{  
    repository.Create(item);  
}
```

```
var builder = WebApplication.CreateBuilder(args); // exposes DI container
```

```
{  
    builder.Services.AddControllers();  
    builder.Services.AddScoped< IRepository<ToDoItem>, ToDoItemsRepository>();  
}  
var app = builder.Build();
```





Dependencies

**Dependency
Injection**

**Dependency
Inversion**

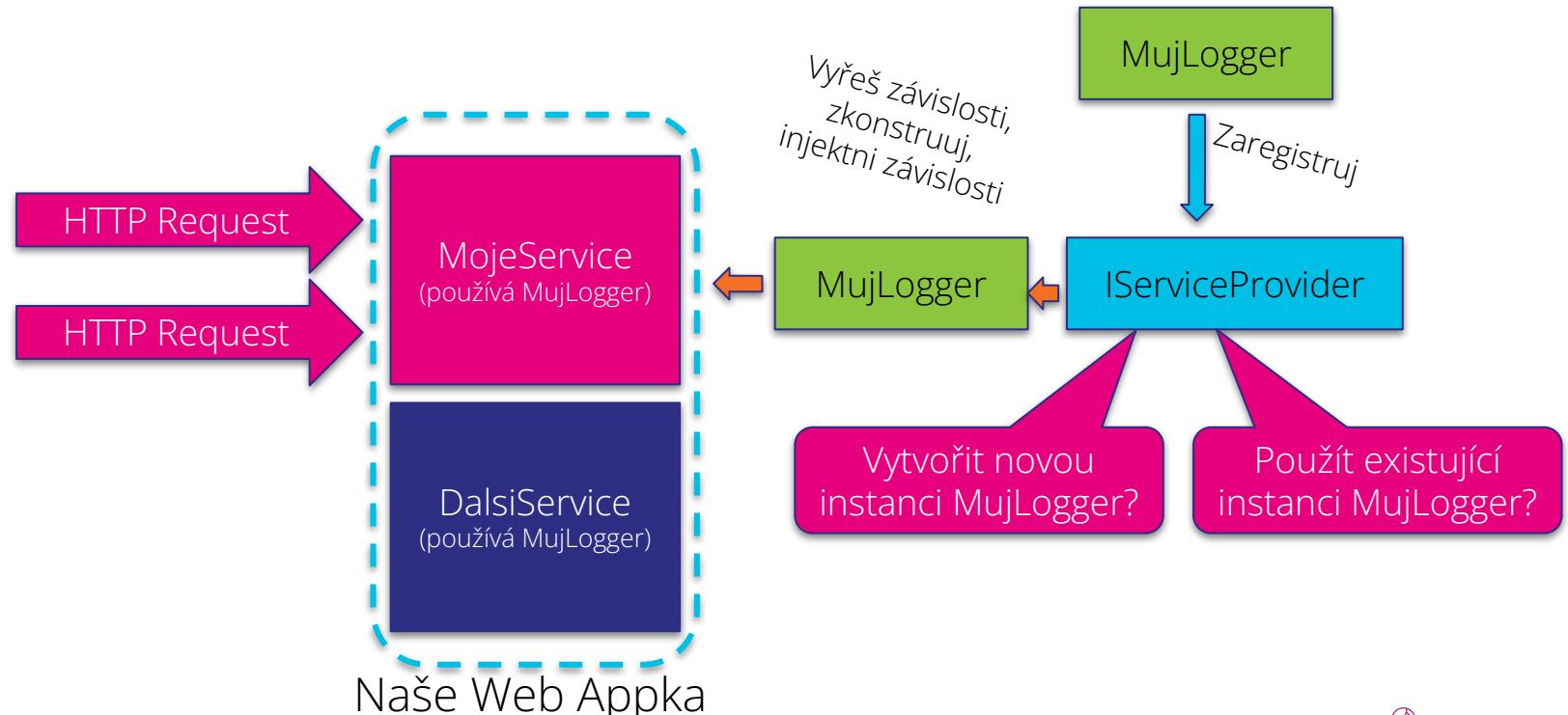
**DI
Containers**

**IoC
Containers**

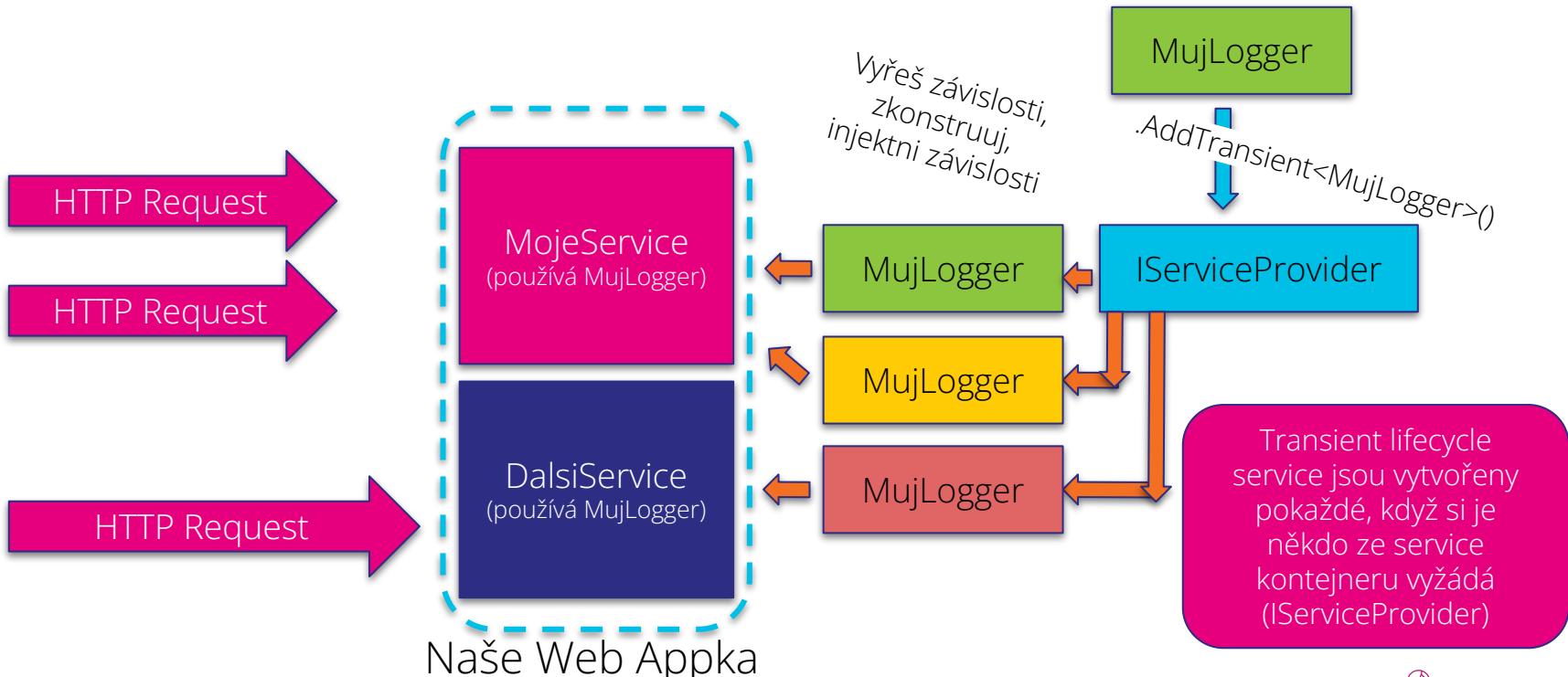
Pauza

Service Lifecycle

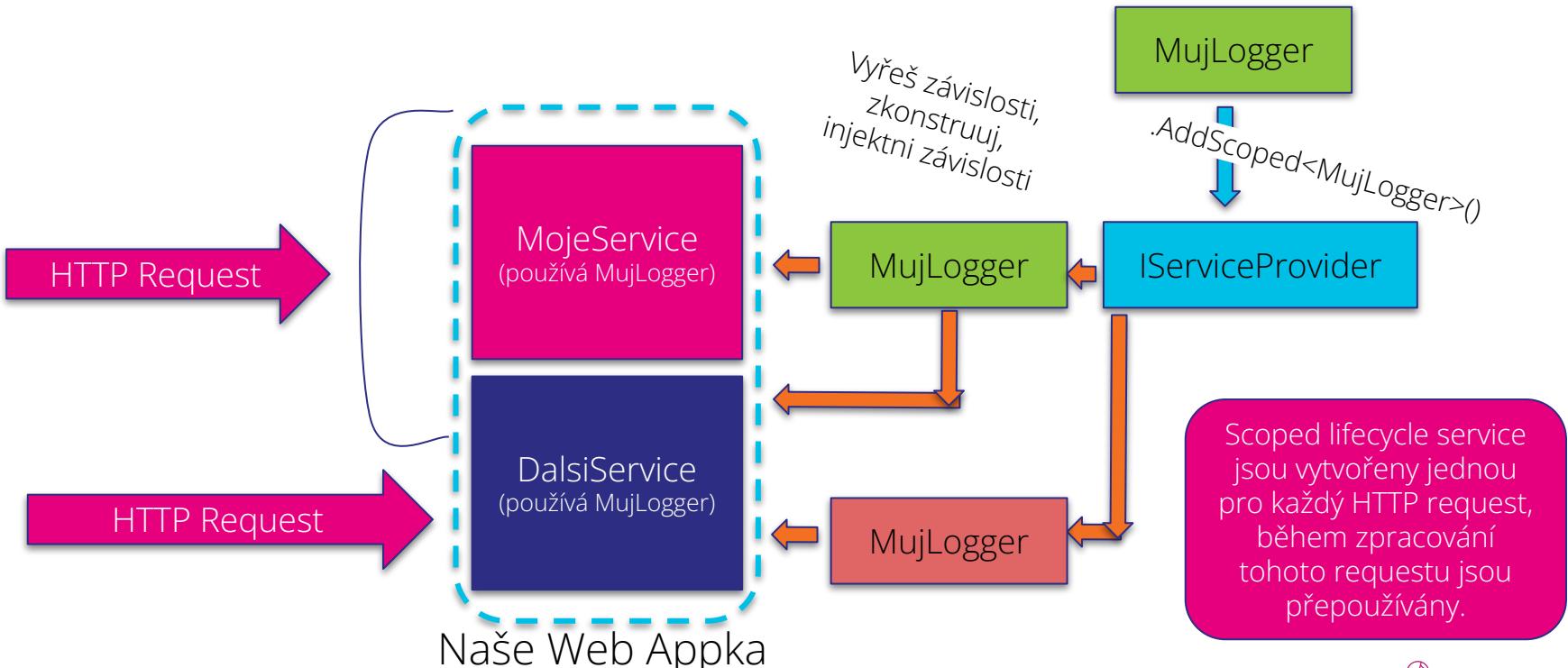
Kdy by se měly třídy instanciovat?



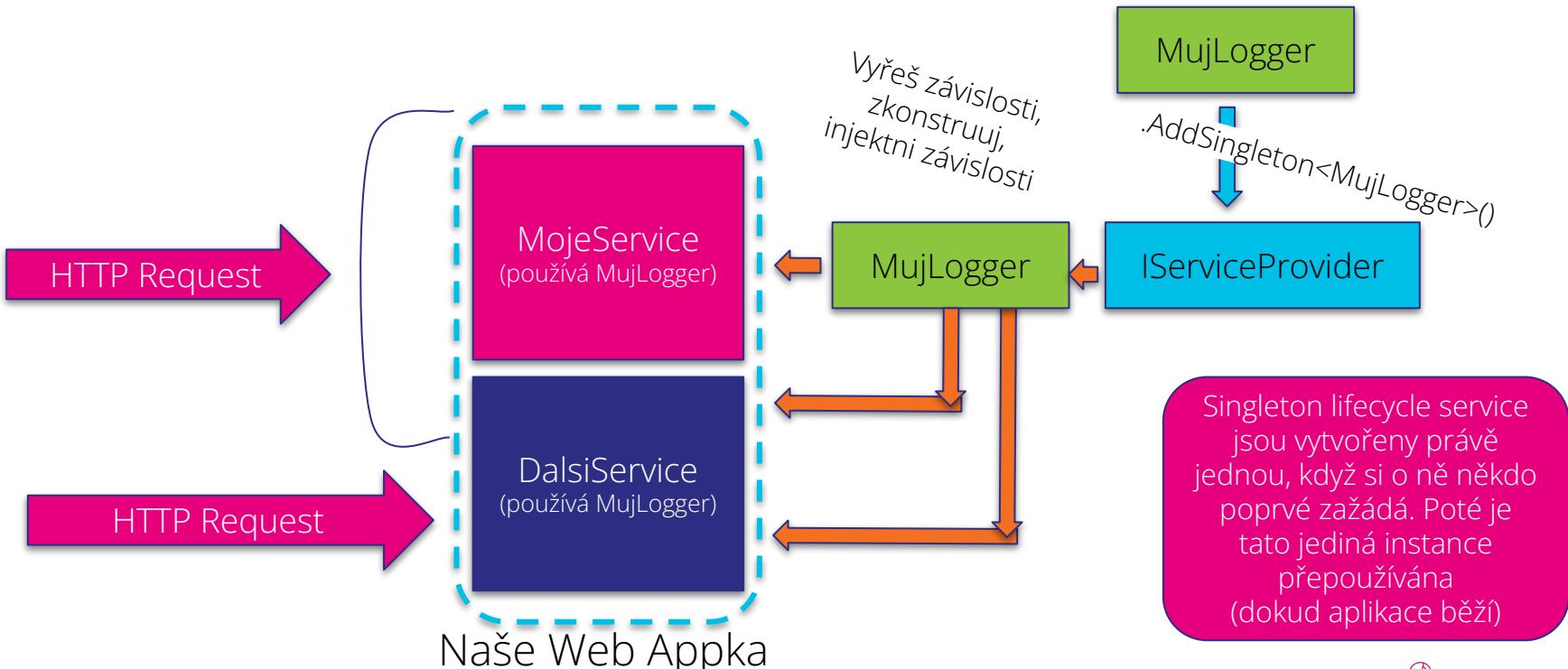
Transient Service Lifecycle



Scoped Service Lifecycle



Singleton Service Lifecycle



Swagger

Open API, Swagger UI

OpenAPI Specification - Version 3.1.0 | Swagger

Swagger Petstore

GitHub - swagger-api/swagger-ui: Swagger UI

LEKCE 08

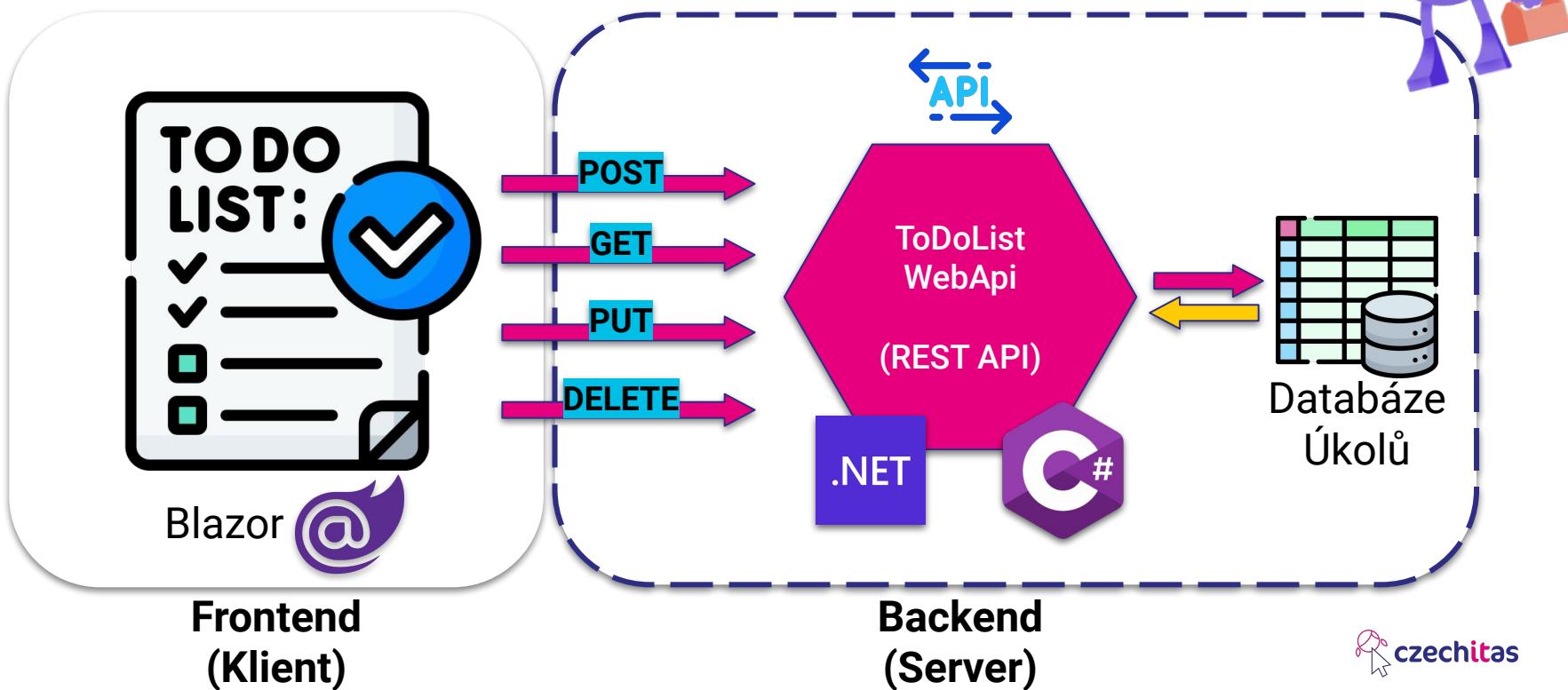
Blazor Introduction

ThermoFisher
SCIENTIFIC

Lekce 08

1. Frontend
2. Co je to Blazor?
3. Interaktivita
4. Komponenty

Co už máme



K čemu frontend

- Sami dokončíme celý projekt
- “Full-stack” porozumění
- Uvidíme, jak se používá naše API

Co je Blazor

- **Razor** ≈ syntax, který kombinuje HTML a C#
- **Blazor** ≈ framework na tvorbu webových aplikací (pomocí Razoru)
 - Browser + Razor
 - Běží na serveru i na clientovi
 - **S**ingle **P**age **A**pplication

SPA vs MPA



Proč Blazor

- Eliminuje nutnost používat JavaScript*
- Sdílení logiky a knihoven mezi BE a FE
- Open-source, podpora od Microsoftu
- .NET ecosystem (MAUI)

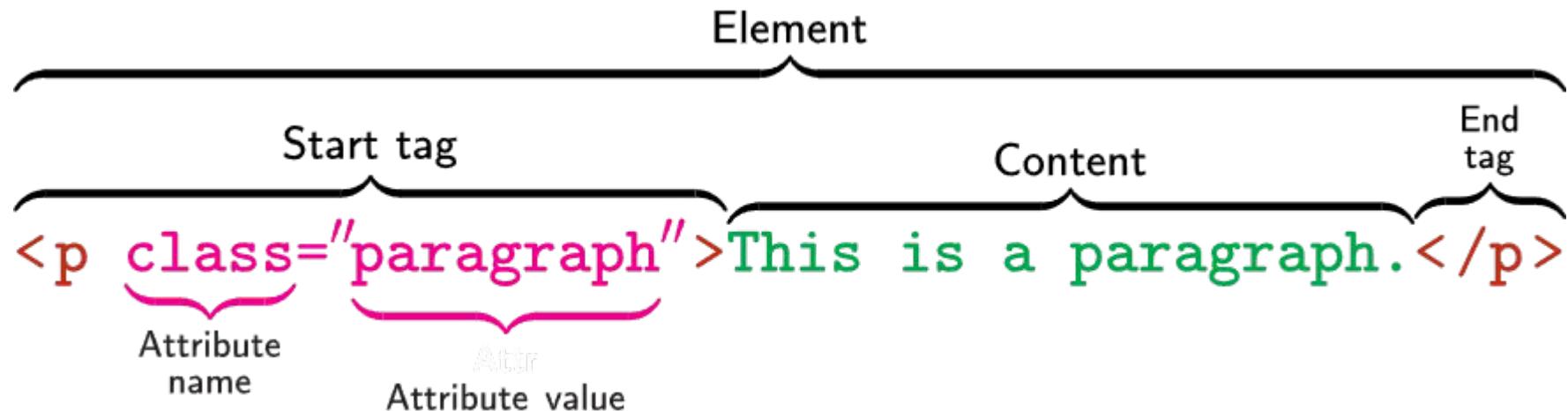
dotnet new blazor

HTML

HTML

- HyperText Markup Language
- Definuje obsah a strukturu
- <head> a <body>
- Skládá se z HTML elementů
- Elementy vytváříme pomocí otevíracích a zavíracích tagů

HTML syntax



Základní HTML tagy

- <**h1**>Titulek</**h1**>
- <**p**>Paragraf</**p**>
- <**br**> - Nový řádek



Interaktivita

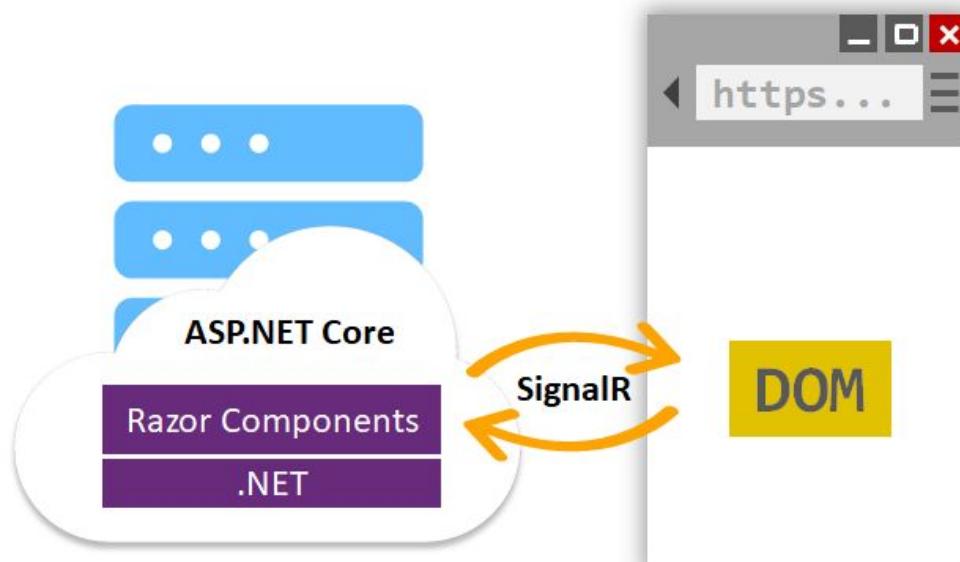


Renderovací typy

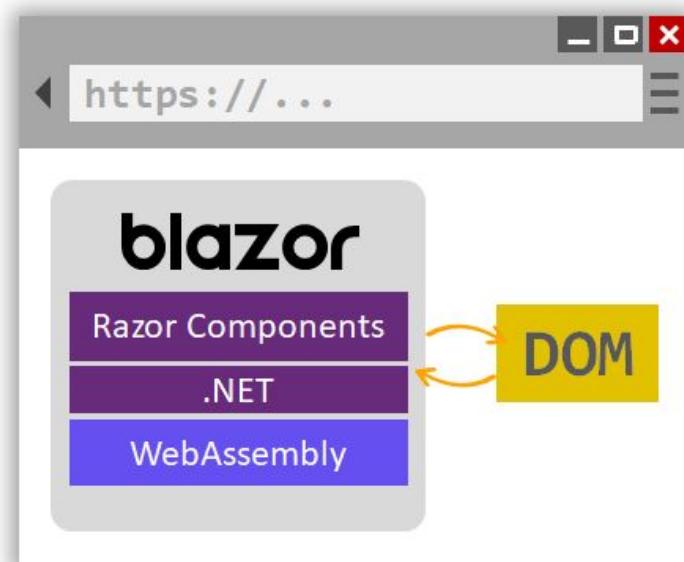
Name	Description	Render location	Interactive
Static Server	Static server-side rendering (static SSR)	Server	✗
Interactive Server	Interactive server-side rendering (interactive SSR) using Blazor Server.	Server	✓
Interactive WebAssembly	Client-side rendering (CSR) using Blazor WebAssembly†.	Client	✓
Interactive Auto	Interactive SSR using Blazor Server initially and then CSR on subsequent visits after the Blazor bundle is downloaded.	Server, then client	✓

Jak propíšeme změny?

Server-side Blazor



Client-Side Blazor



<button>

```
builder.Services.AddRazorComponents()  
    .AddInteractiveServerComponents();  
  
app.MapRazorComponents<App>()  
    .AddInteractiveServerRenderMode();
```

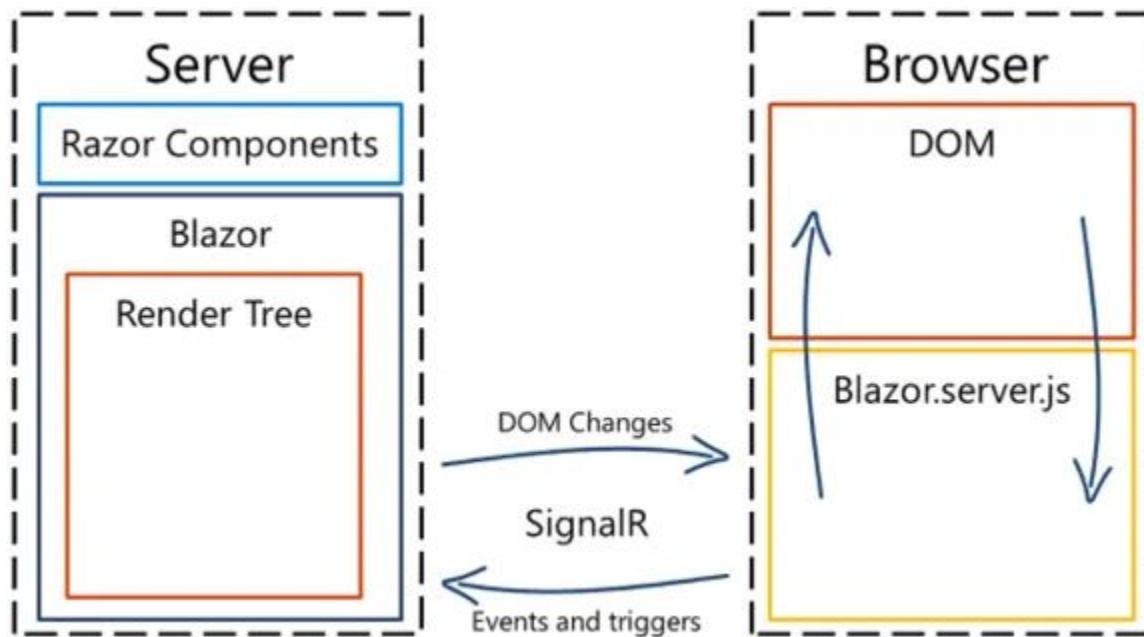
@rendermode InteractiveServer

<Komponenty>

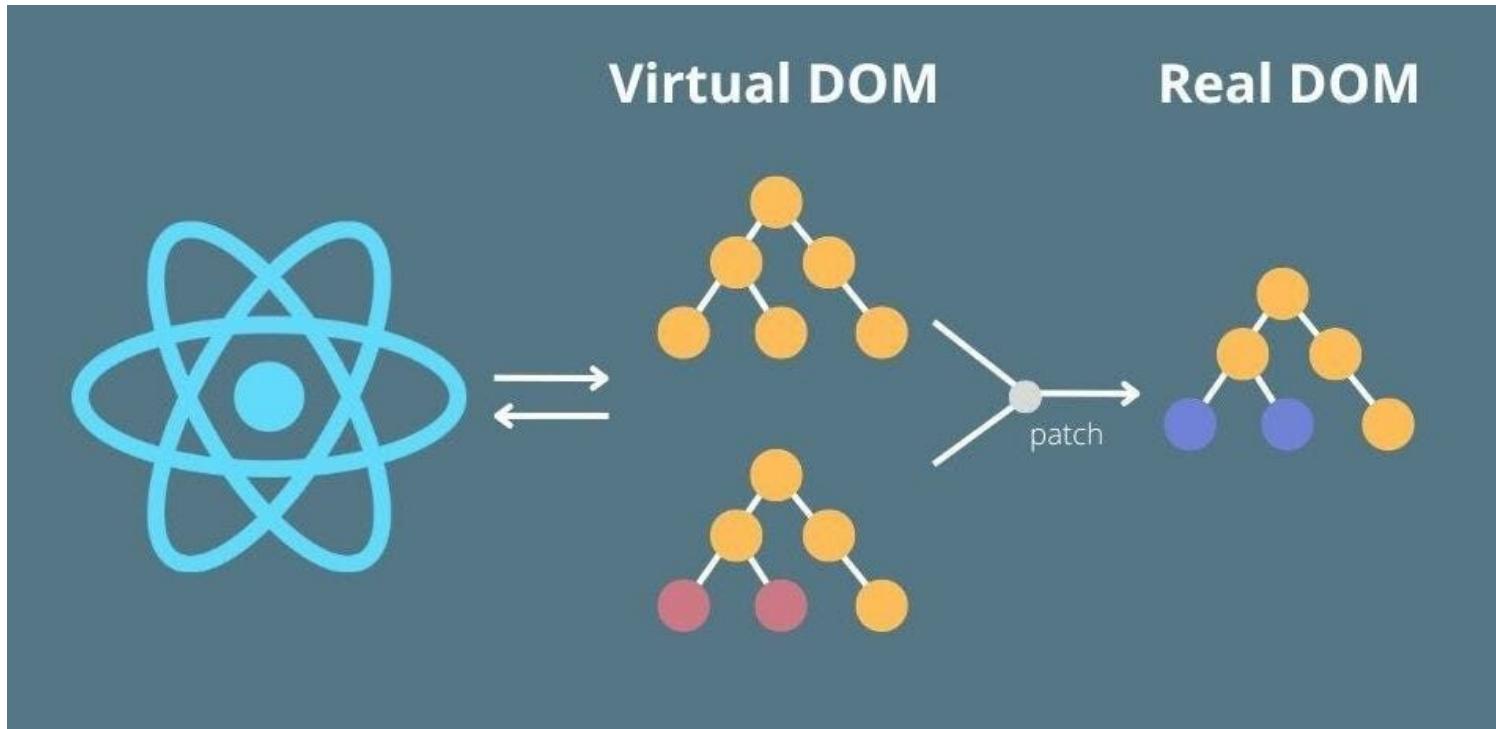
Komponenty

- Přepoužitelný kus UI
- Zahrnuje HTML, CSS i C# na jednom místě
- Nemusíme manuálně upravovat DOM
- Použití pomocí HTML syntaxi
- Můžeme nakoupit komponentové knihovny

Komponenty



Komponenty



Stránka vs komponenta

- Stránka je komponenta, která má svou cestu
- @page "/" direktiva zaregistrouje cestu ke komponentě

Další HTML tagy

- <button>Tlačítko</button>
- <table>Tabulka</table>
- <tr>Tabulkový řádek</tr>
- <td>Tabulková data</td>

Extra zdroje na závěr

- [Interaktivní HTML editor](#)
- [Awesome Blazor](#)
- [Blazor school](#)

LEKCE 09

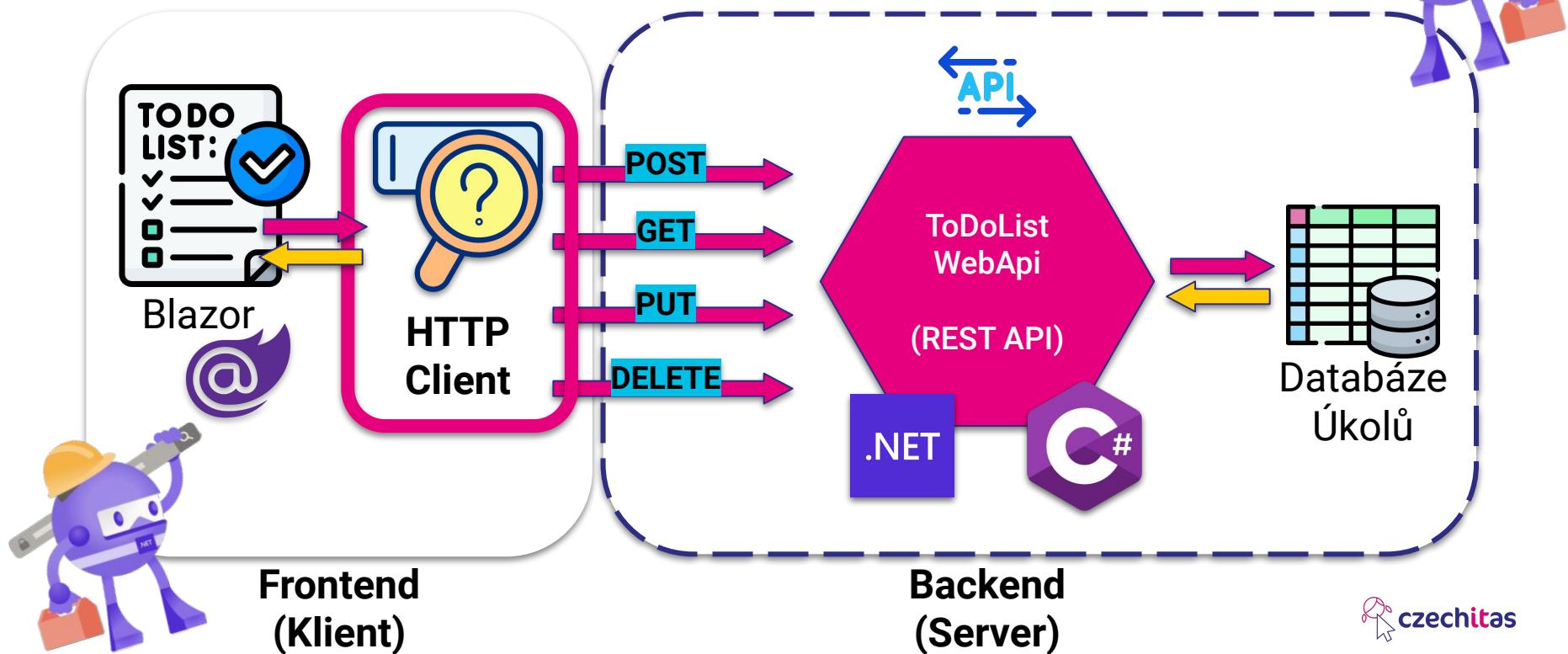
HTTP Klient, Asynchronní programování

Lekce 08

1. **HTTP Client**
2. **Asynchronní programování**
3. Závěrečný **projekt**

HTTP Client

Co už máme



Pojďme do kódu

Asynchronní Programování



Synchronní a Asynchronní práce



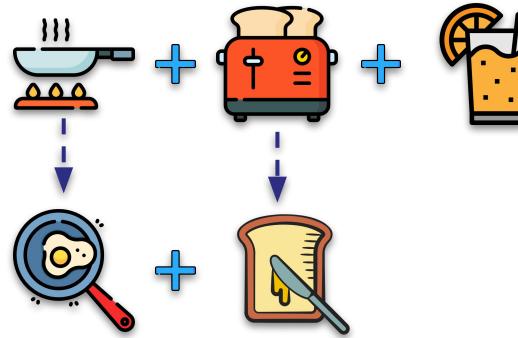
1 kuchařka (Thréda)

Synchronní



Asynchronní

20 minut

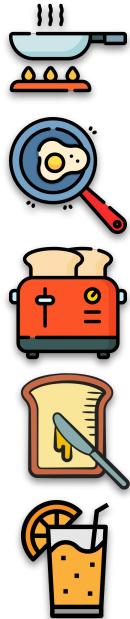


Další příklady asynchronní práce

- Zaslání emailu nebo pozvánky na meeting
- Veškeré rezervace
- Stahování souborů
- Nakupování na internetu
- Objednání jídla v restauraci
- Praní prádla v pračce, Vaření vody v konvici
- Nečekání na zrající ovoce

Paralelní práce

Synchronní

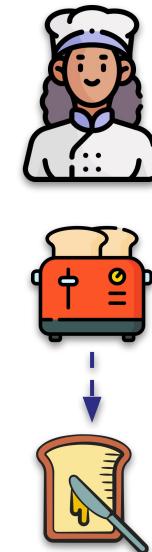


⌚ 30 minut



3 kuchařky (Thready)

Paralelní



⌚ 8 minut

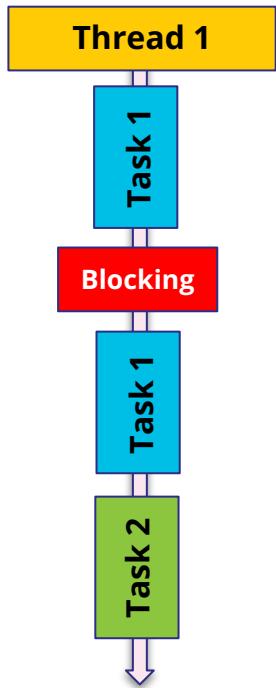
Multithréding

Další příklady paralelní práce

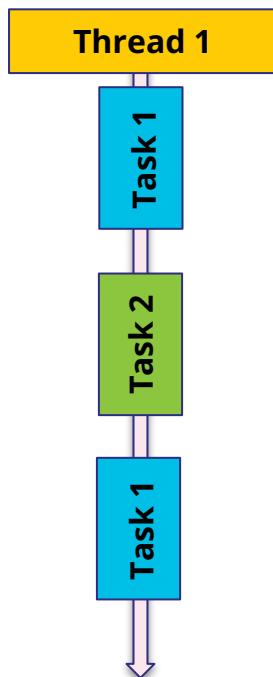
- Orchestr
- Kuchyně fast-foodového řetězce
- Natáčení filmu
- Výuka v celé škole dle rozvrhu
- Oprava silnice?



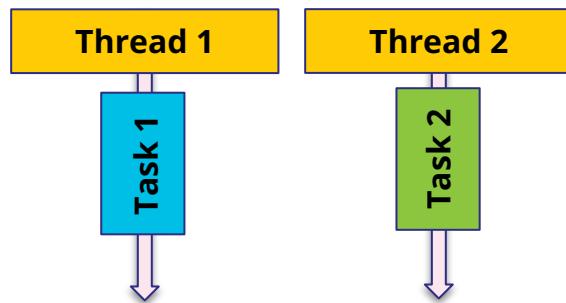
Synchronní vs Asynchronní vs Multithreading



Jeden kuchař,
čekající na každý
krok

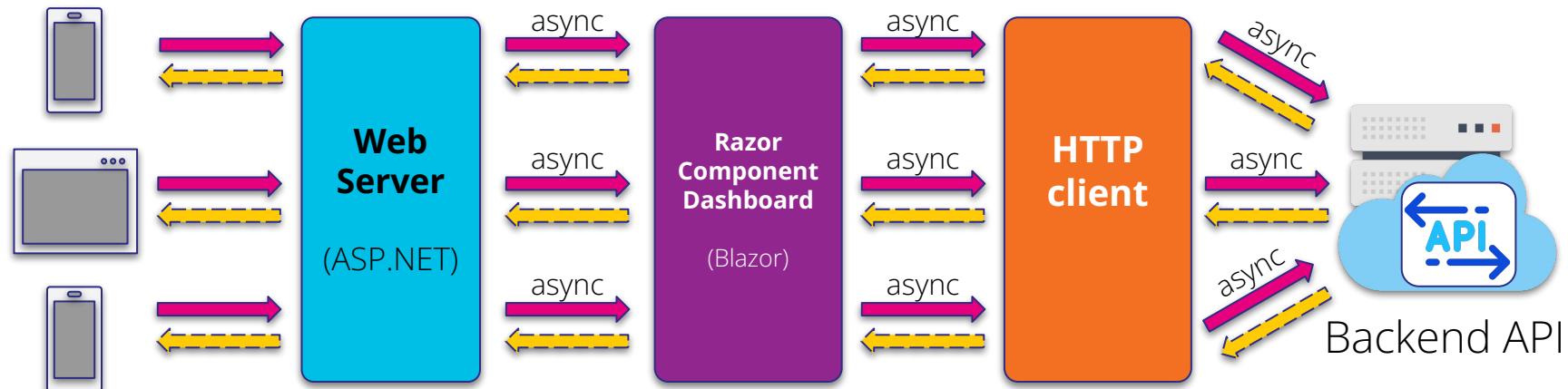


Jeden kuchař, **nečekající**
na dokončení každého
kroku. Místo aby čekal,
dělá něco dalšího



Dva kuchaři,
dokážeme jim rozdělit práci,
pracují paralelně.

Asynchronní programování v ASP.NET Core



Benefit

- **Zlepšení Performance**: Vyhneeme se blokování volání, thread ze které se metody volají se může věnovat dalším úkolům.
- **Zlepšení Scalability**: Naše aplikace takto může zvládat více requestů a uživatelů současně.
- **Zjednodušení Kódu**: Asynchronní kód je jednodušší na napsání díky Task objektům a async/await keywords

Nová klíčová slova

- **Task<ReturnType>**
- **Task** (*ReturnType* je *void*)
- **async**
- **await**

Závěrečný projekt

Závěrečný projekt - Co očekávat

ToDoList ToDoItem Categories

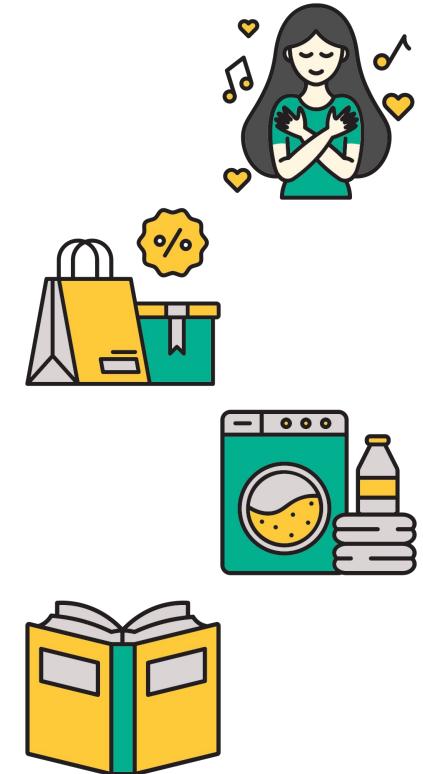
- Do ToDoList.Models.ToDoItem přidáme nový **string?** field
- **Category** je nový field, který musí být podporován
 - Na backendu (Databáze + EF + Repository + WebAPI + DTOs)
 - Na frontendu
 - Dashboard component
 - EditItem component (v příští lekci ;)



Závěrečný projekt - Motivace

ToDoList ToDoltem Categories

- Umíme samostatně pracovat s:
 - Modely
 - DTOs
 - WebAPI
 - Databází
 - Repository patternem
 - Testy
 - Blazorem



Závěrečný projekt - Flow Odevzdávání

ToDoList Categories - BE

27.11.

ToDoList Categories - FE

4.12.

ToDoList Categories - Final

11.12.

Úprava backendu,
aby podporoval kategorie

- Models
- Entity + Context
- Database Schema
 - Migrations
- DTOs
- Endpoint Controller
- Testy

Úprava frontendu,
aby podporoval kategorie

- Dashboard
- EditItem
- Opravy BE na základě
zpětné vazby

Prostor pro vychytání chyb a
vylepšení programu a kódu.

- Vylepšení vzhledu pro
uživatele frontendu
- Opravy FE na základě
zpětné vazby
- Refactoring
 - jednoduchost
 - přehlednost
 - pojmenování

Bootstrap CSS

Bootstrap

- Frontendový toolkit pro moderní styl stránek
- Responzivní, podporující grid systém
- CSS
- Cheatsheet · Bootstrap v5.2

<https://getbootstrap.com>

LEKCE 10

Blazor - EditToDoltem Component, EditForm

LEKCE 11

Refactoring

LEKCE 12

Opakování, Závěr