# LEKCE 07

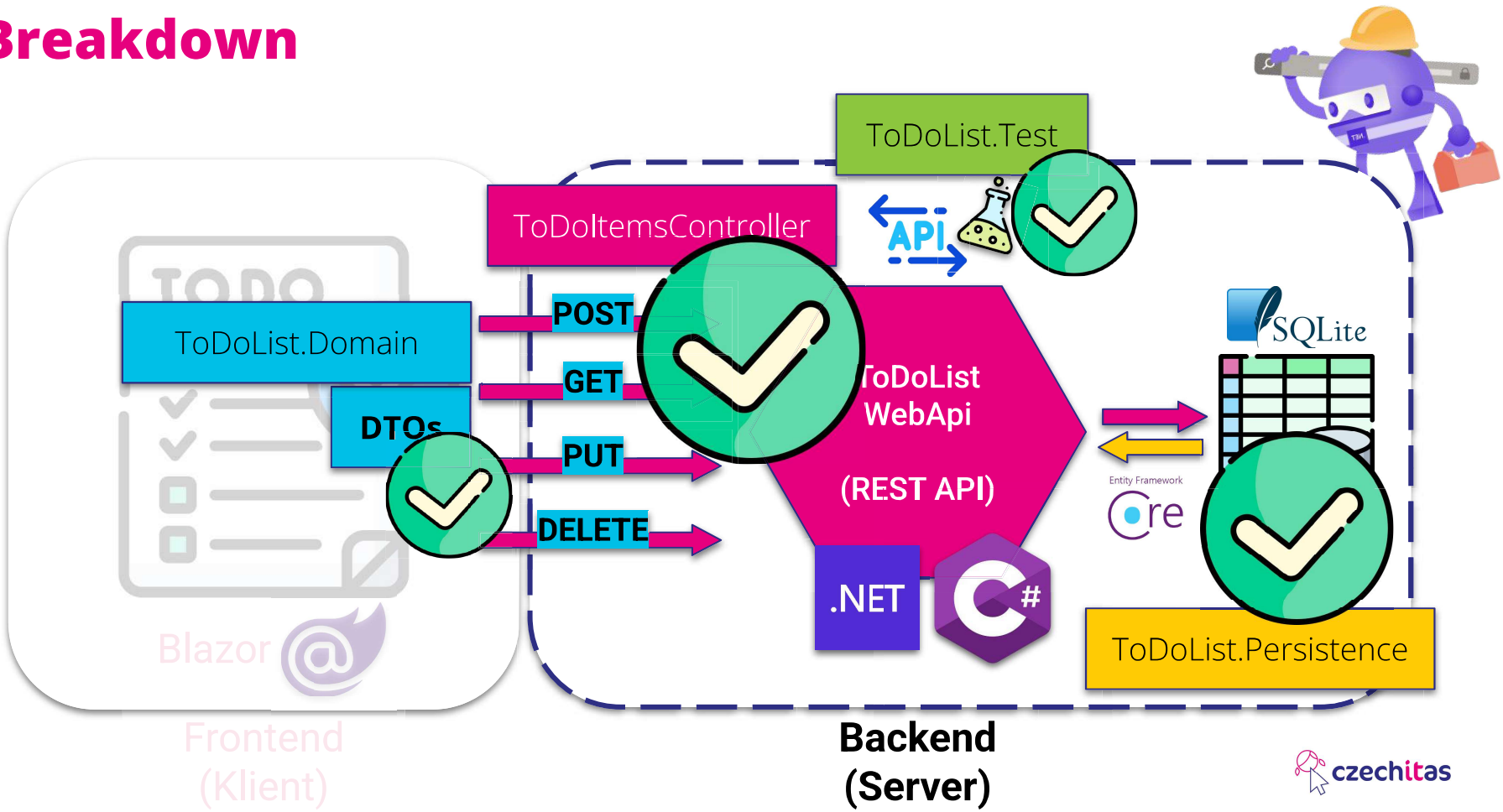
Opakování, Refactoring, Dependency Injection

# 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)

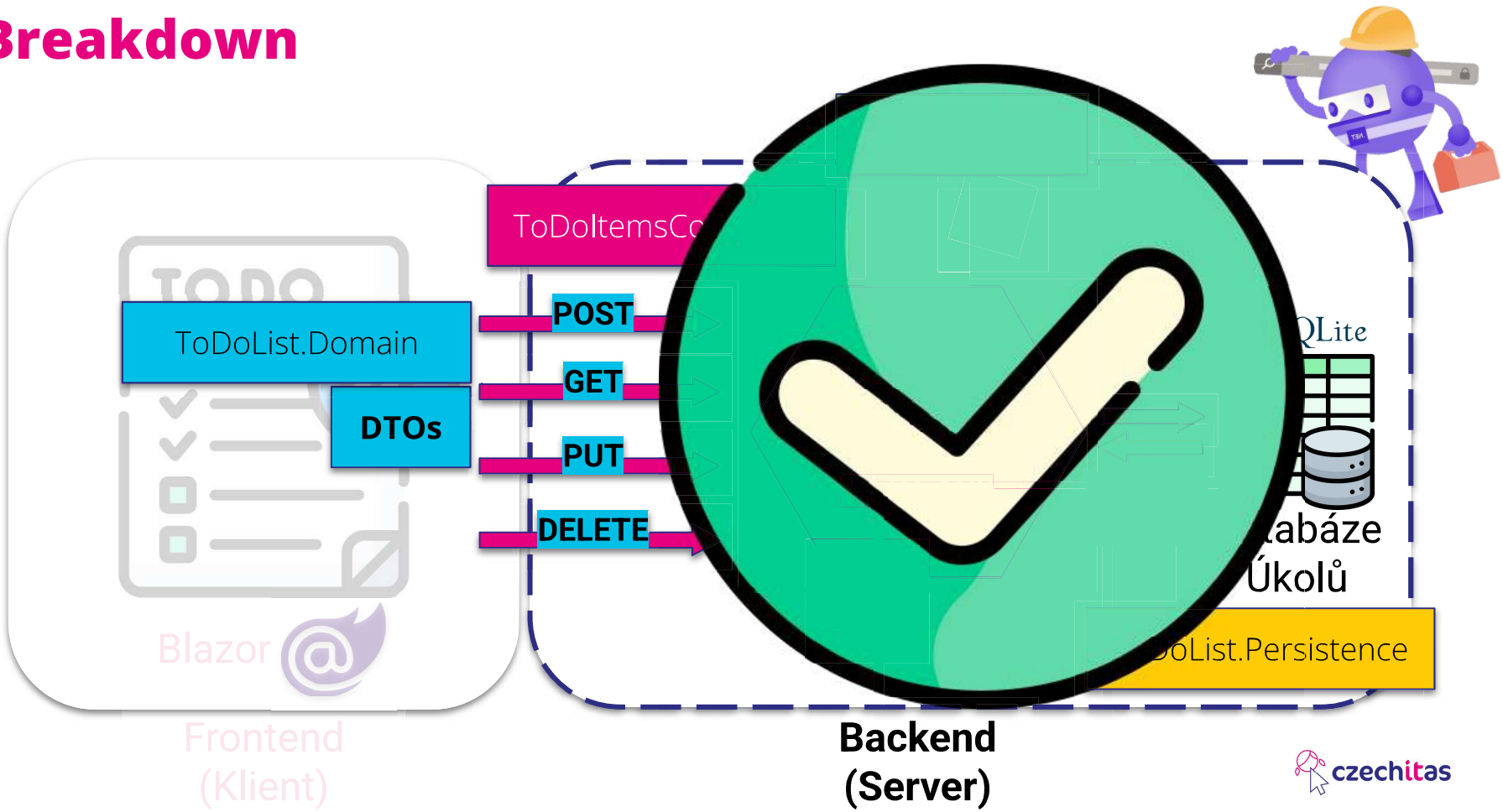
czechitas

# Breakdown

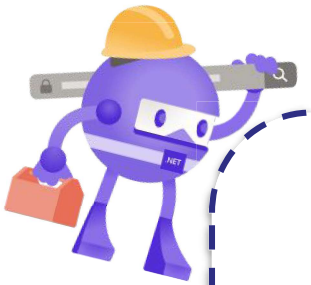


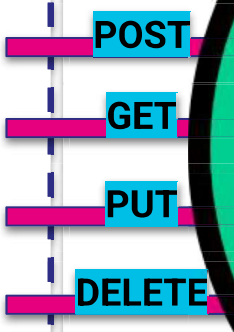

czechitas

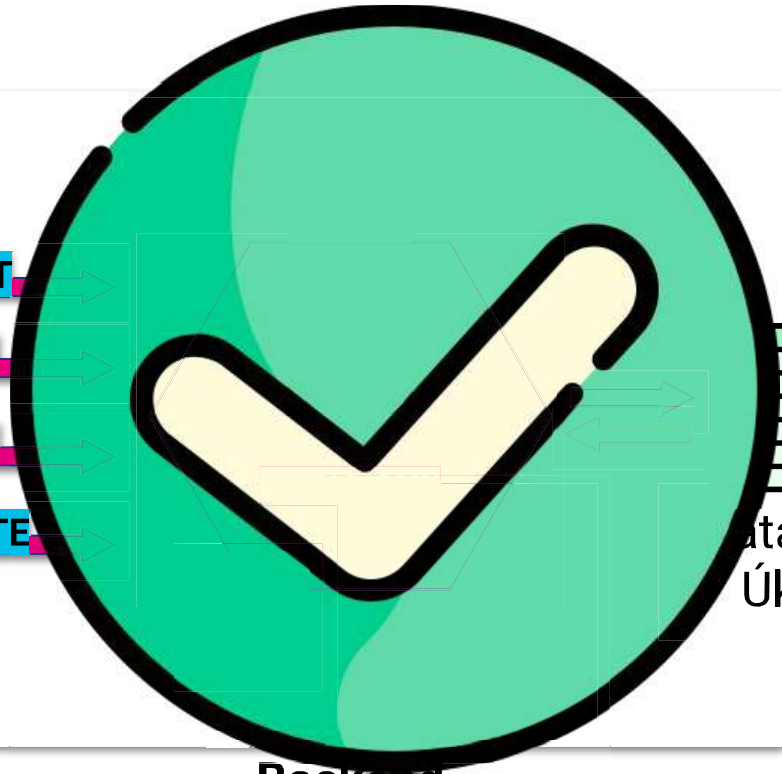
# Breakdown

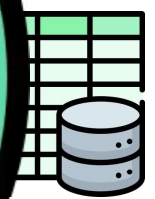

ToDoItemsCo
POST
GET
PUT
DELETE
ToDoList.Domain
DTOs
Blazor
Frontend
(Klient)
QLite
abáze
Úkolů
oList.Persistence
Backend
(Server)
czechitas

# CO NÁS ČEKÁ





tabáze
Úkolů

**Frontend**
**(Klient)**

**Backend**
**(Server)**

czechitas

# 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!**

# Závislosti

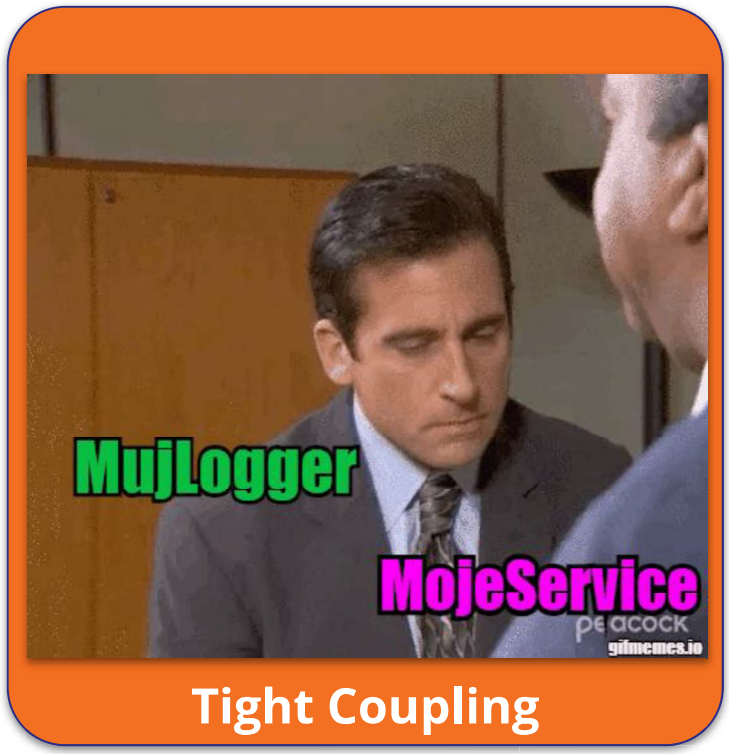
# Dependencies

# Co jsou to Dependencies?


**ZÁVISLOST (DEPENDENCY)**

```
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!");
}
```




czechitas

# Co jsou to Dependencies?


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

**Nevýhody**

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

czechitas

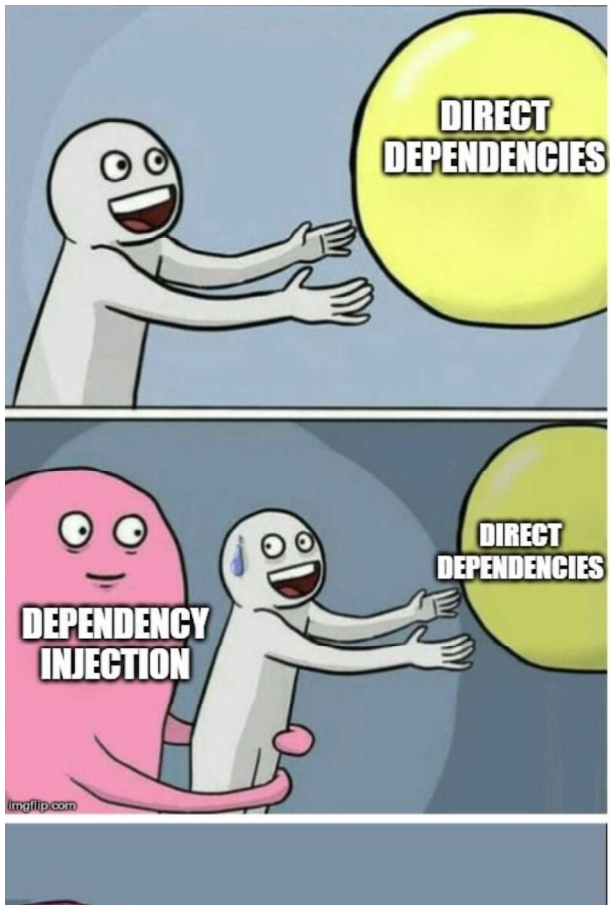
## Co jsou Dependencies?

### *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).*


DIRECT
DEPENDENCIES
DIRECT
DEPENDENCIES
DEPENDENCY
INJECTION
imgflip.com

# Dependency Injection

# Co je to Dependency Injection?


**ZÁVISLOST (DEPENDENCY)**

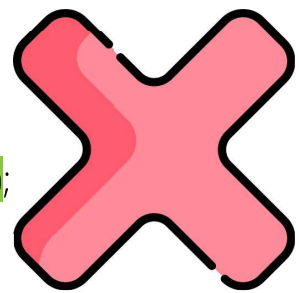
```
public MojeService(MujLogger logger)
{
    logger.Log("Jsem připraven!");
}
```

**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

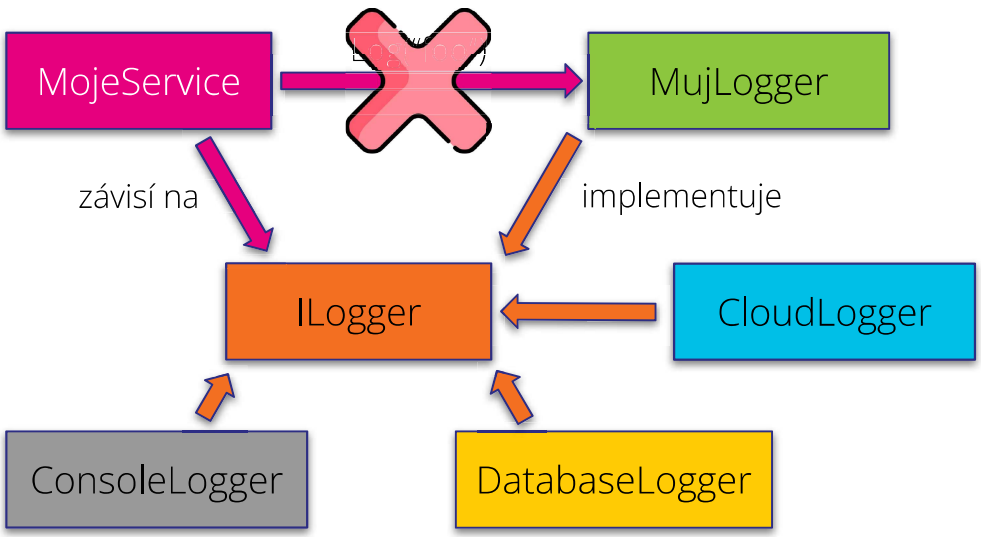

```
public ToDoItemsController()
{
    var repository = new ToDoItemsRepository(!!!);
    repository.Create(item);
}
```



```
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)

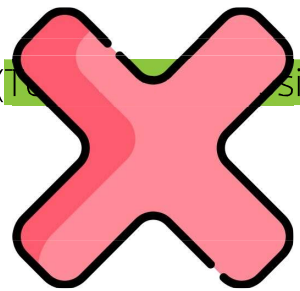
"Kód by měl záviset na abstrakcích místo na konkrétních implementacích"

**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** jde mockovat.
- **Kód je čistější, jednodušší na úpravu a přepoužití**

# Použití Dependency Inversion


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



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

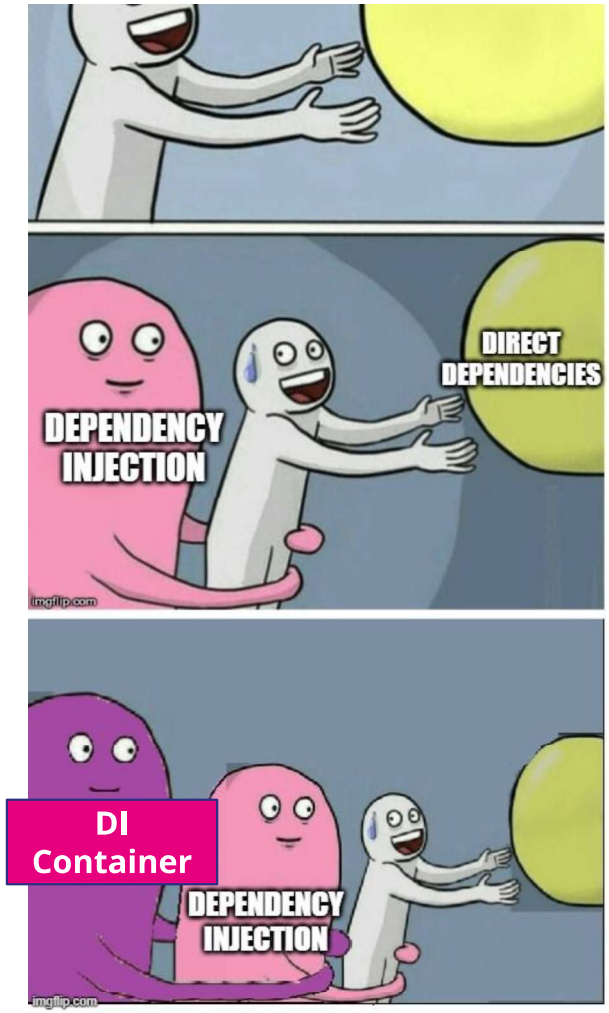
**Princip Dependency Inversion**
(Inverze Závislostí)

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

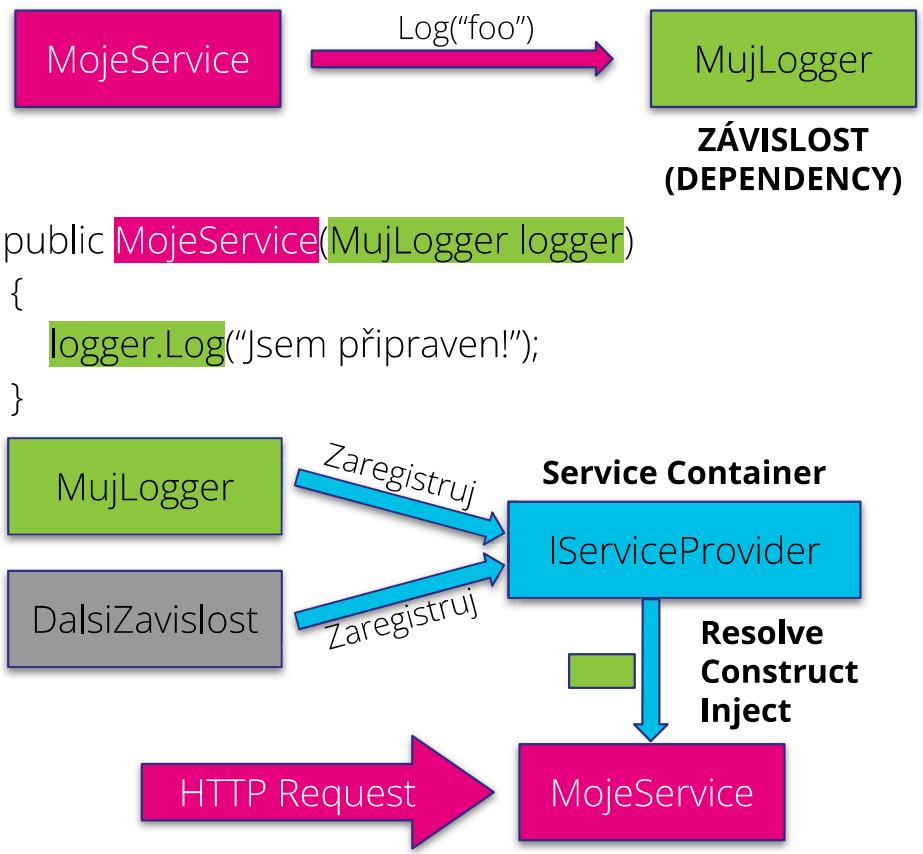
czechitas

# Dependency Injection Container
# (DI Container)

*také IoC Container*


DIRECT
DEPENDENCIES
DEPENDENCY
INJECTION
imgflip.com
DI
Container
DEPENDENCY
INJECTION
imgflip.com

# Co je to DI Container?




**Benefity**

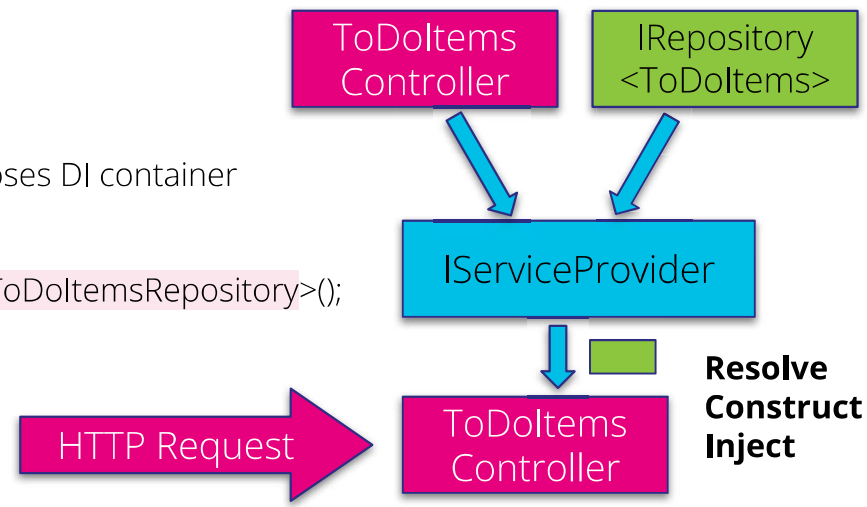
- Centralizace správy tříd, naši aplikaci řídíme z jednoho místa.
- IServiceProvider drží všechny závislosti.
- Lifecycle objektů je autonomně řízen kontejnerem (podle nastavení dané závislosti).
- To jaké třídy a závislosti se zaregistrují lze **konfigurovat**.

czechitas

# Použití DI Containeru


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

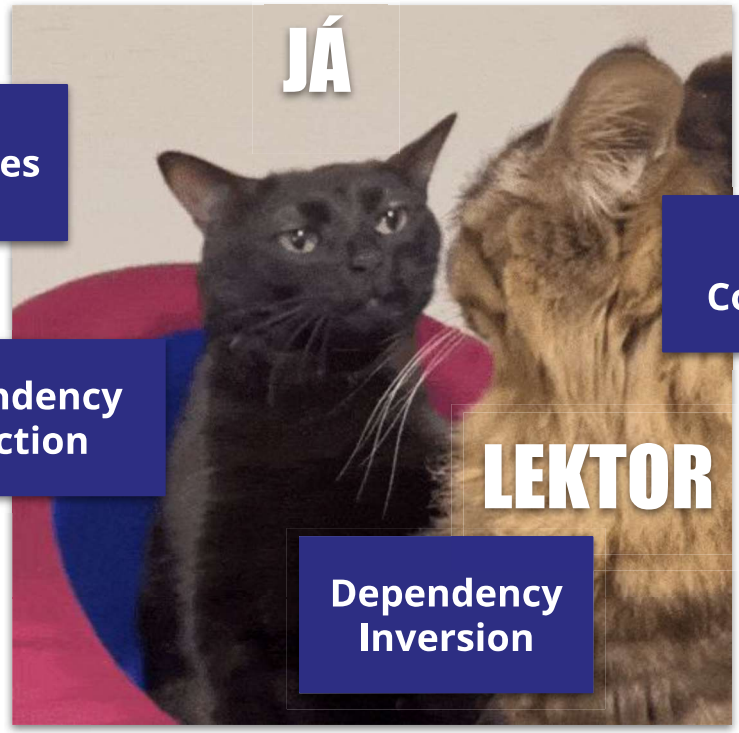
```
var builder  = WebApplication.CreateBuilder(args); // exposes DI container
{
    builder.Services.AddControllers();
    builder.Services.AddScoped<IRepository<ToDoItem>, ToDoItemsRepository>();
}
var app = builder.Build();
```




czechitas

Dependencies

Dependency Injection




Dependency Inversion

DI Containers

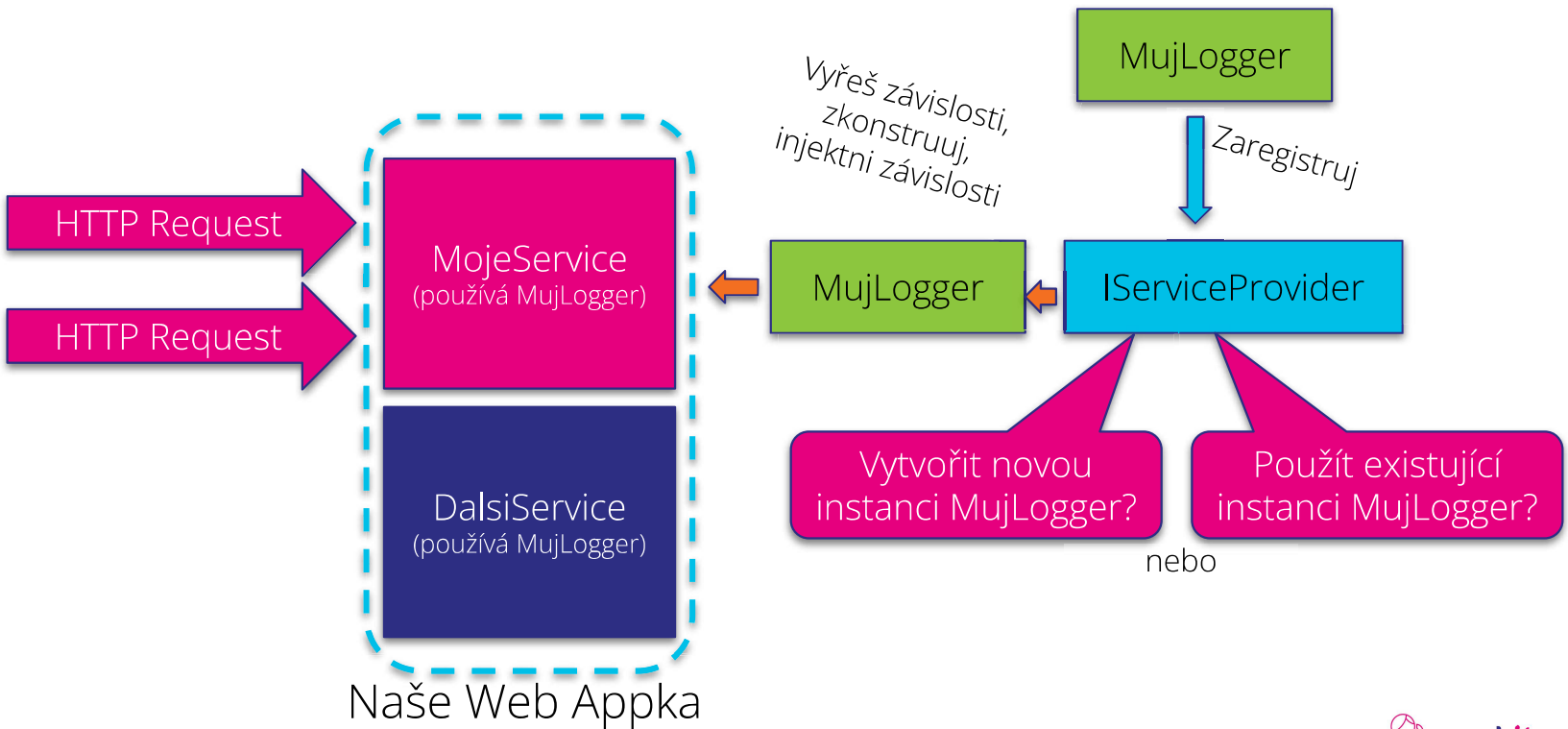
IoC Containers

**Pauza**

czechitas

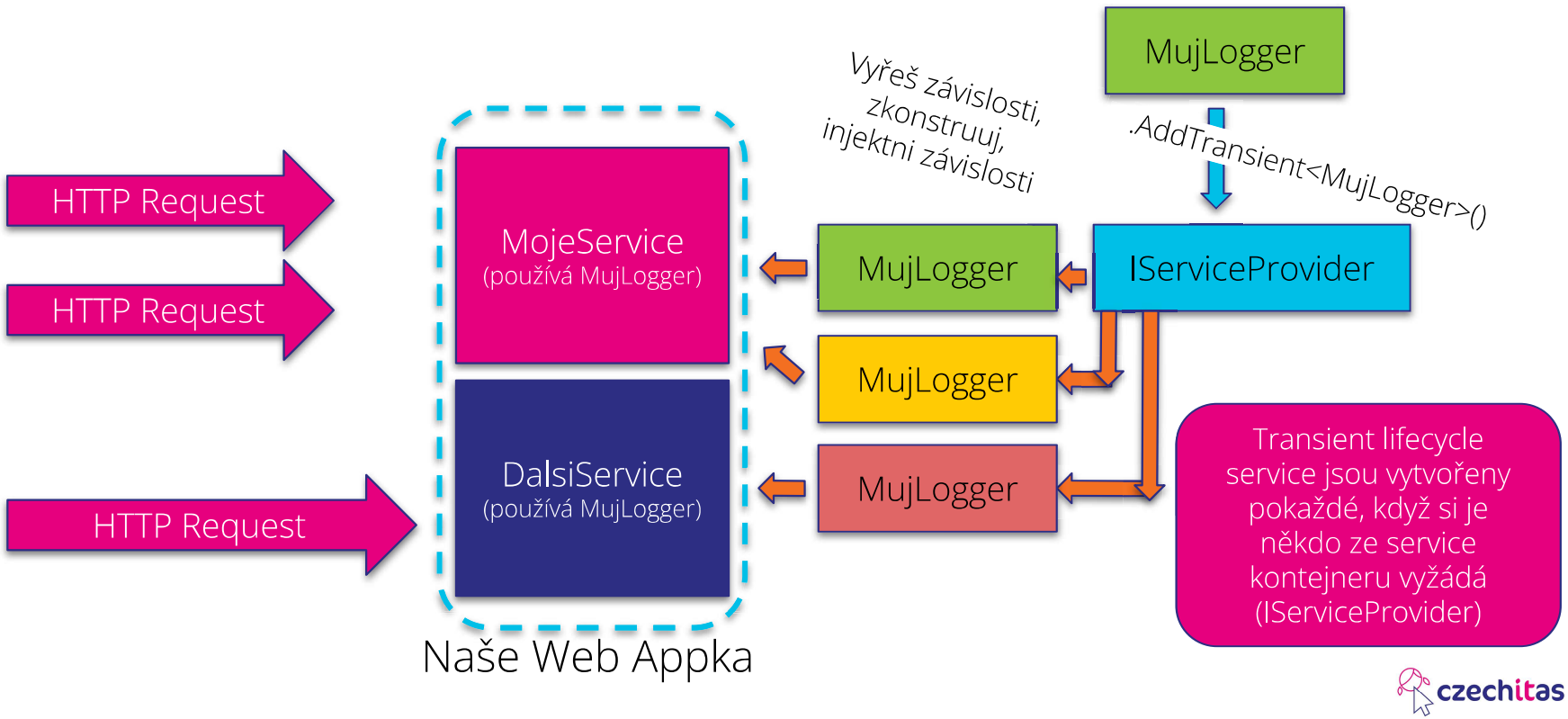
# Service Lifecycle

# Kdy by se měly třídy instanciovat?

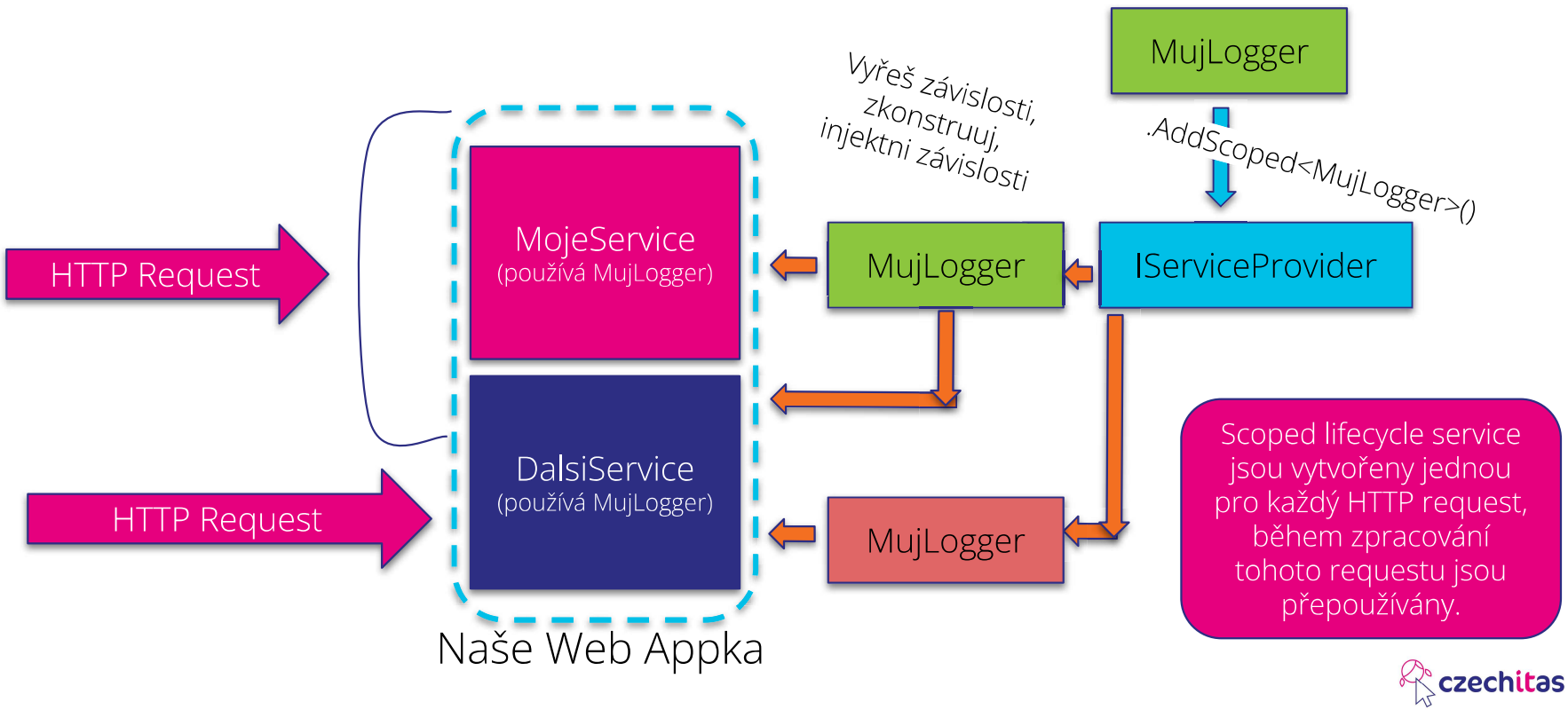



czechitas

# Transient Service Lifecycle

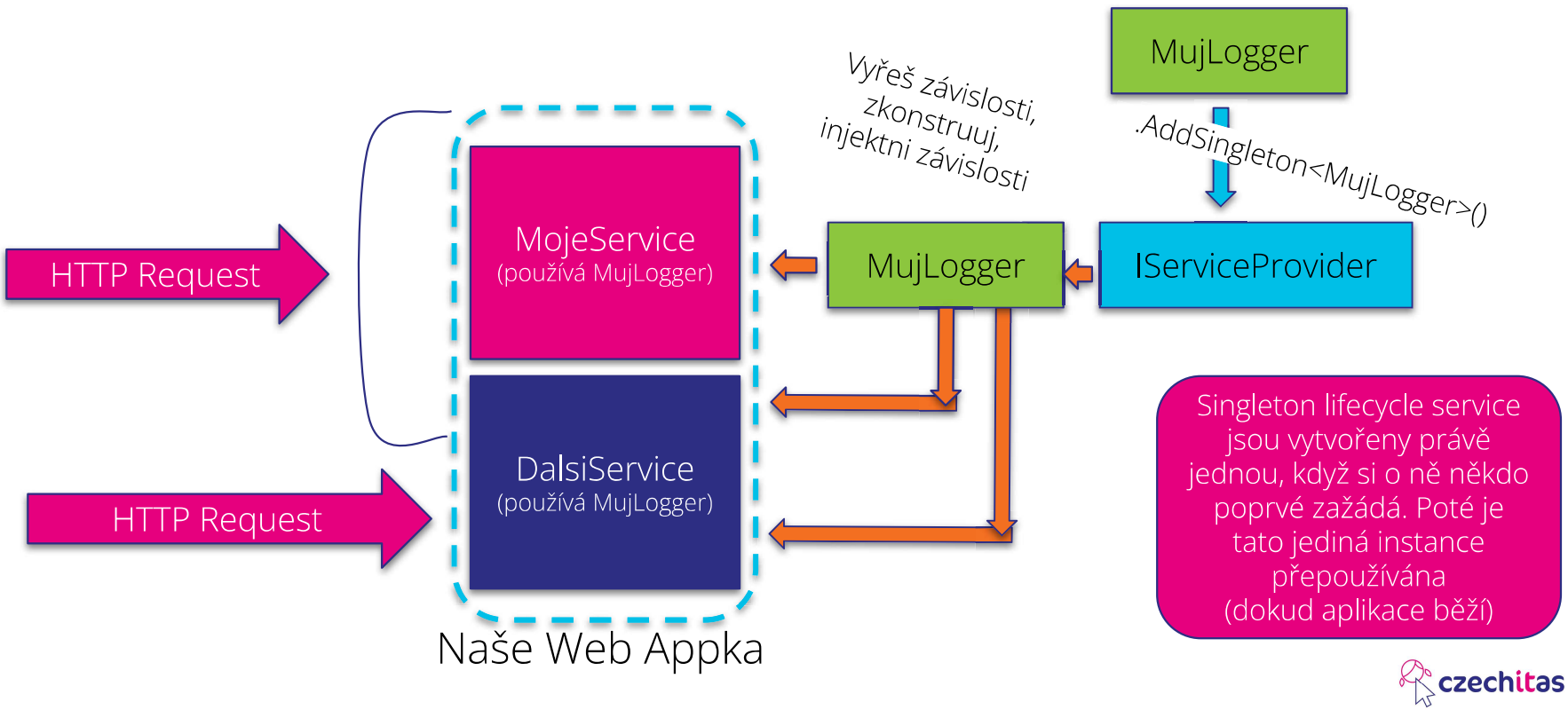


# Scoped Service Lifecycle




czechitas

# Singleton Service Lifecycle




czechitas

# Swagger

# Open API, Swagger UI

OpenAPI Specification - Version 3.1.0 | Swagger

Swagger Petstore

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