LEKCE 06

Generika + Repository pattern

Zpětná vazba 🙏

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

czechitas

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

czechitas

Where omezení

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

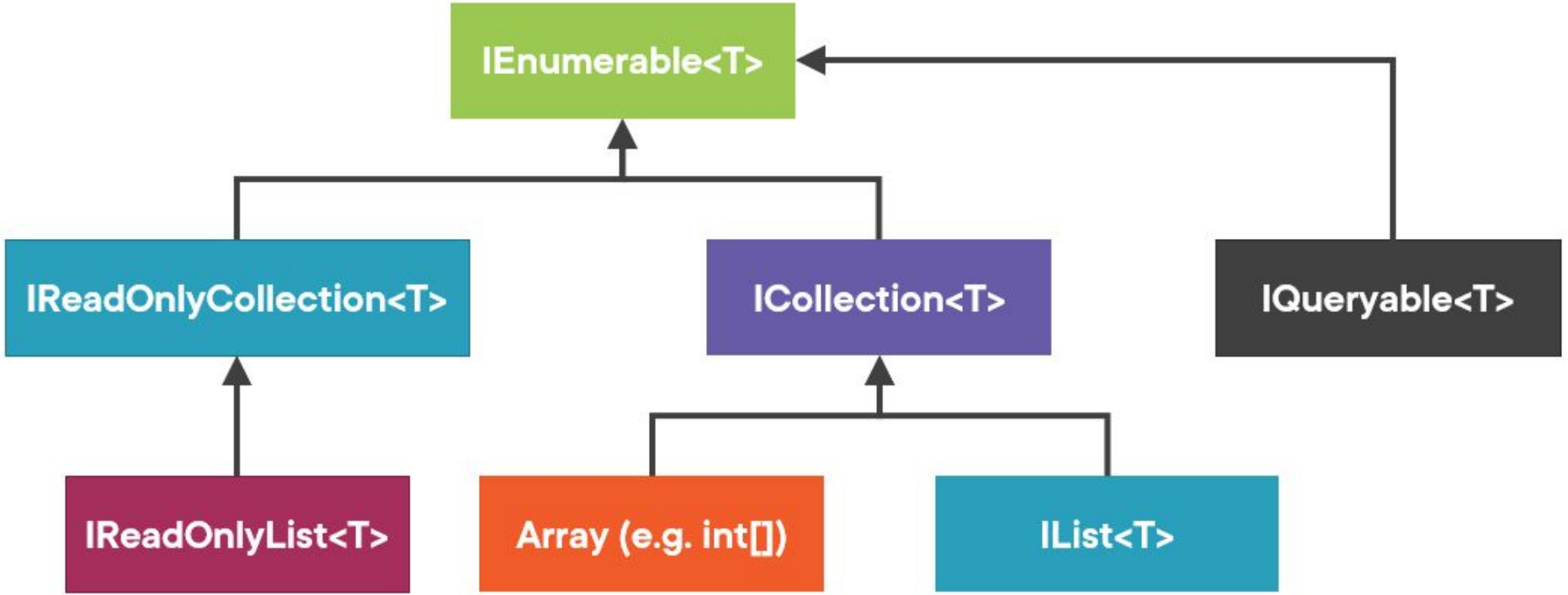
czechitas

Where omezení

Constraint	Description
`where T : struct`	The type argument must be a non-nullable value type, which includes `record struct` 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 `struct` constraint implies the `new()` constraint and can't be combined with the `new()` constraint. You can't combine the `struct` constraint with the `unmanaged` constraint.
`where T : class`	The type argument must be a reference type. This constraint applies also to any class, interface, delegate, or array type. In a nullable context, `T` must be a non-nullable reference type.
`where T : new()`	The type argument must have a public parameterless constructor. When used together with other constraints, the `new()` constraint must be specified last. The `new()` constraint can't be combined with the `struct` and `unmanaged` constraints.
`where T :` *<base class name>*	The type argument must be or derive from the specified base class. In a nullable context, `T` must be a non-nullable reference type derived from the specified base class.
`where T :` *<interface name>*	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, `T` must be a non-nullable type that implements the specified interface.
`where T : U`	The type argument supplied for `T` must be or derive from the argument supplied for `U`. In a nullable context, if `U` is a non-nullable reference type, `T` must be a non-nullable reference type. If `U` is a nullable reference type, `T` can be either nullable or non-nullable.
`where T : default`	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 `default` constraint implies the base method without either the `class` or `struct` constraint. For more information, see the default constraint spec proposal.

czechitas

Generické kolekce

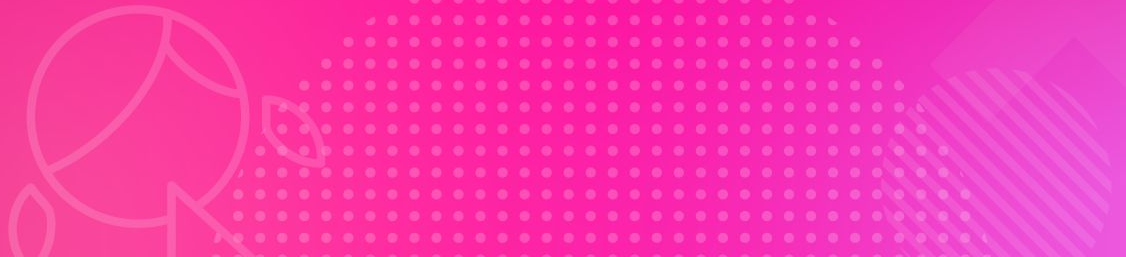


czechitas

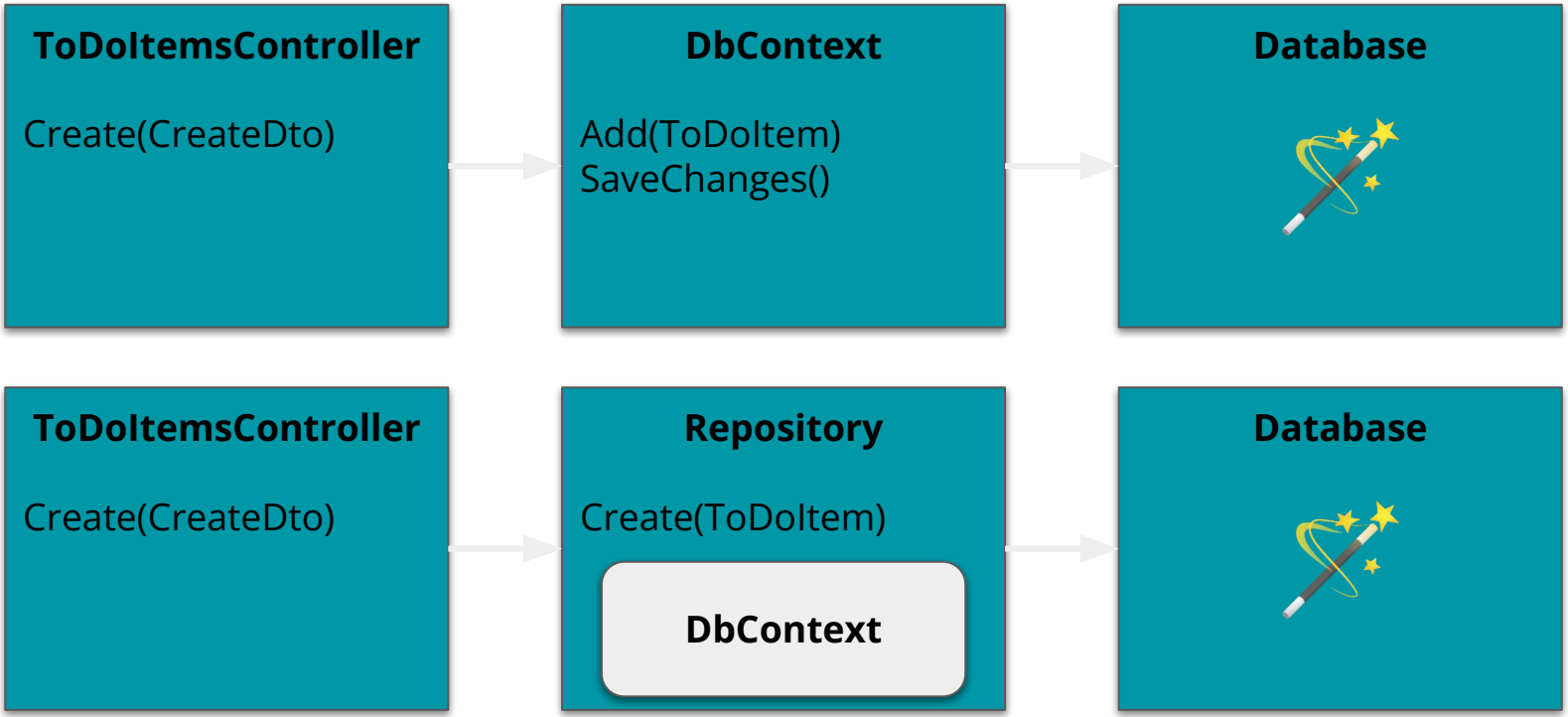
~~ArrayList~~

Všeho s mírou!

Repository



Repository pattern

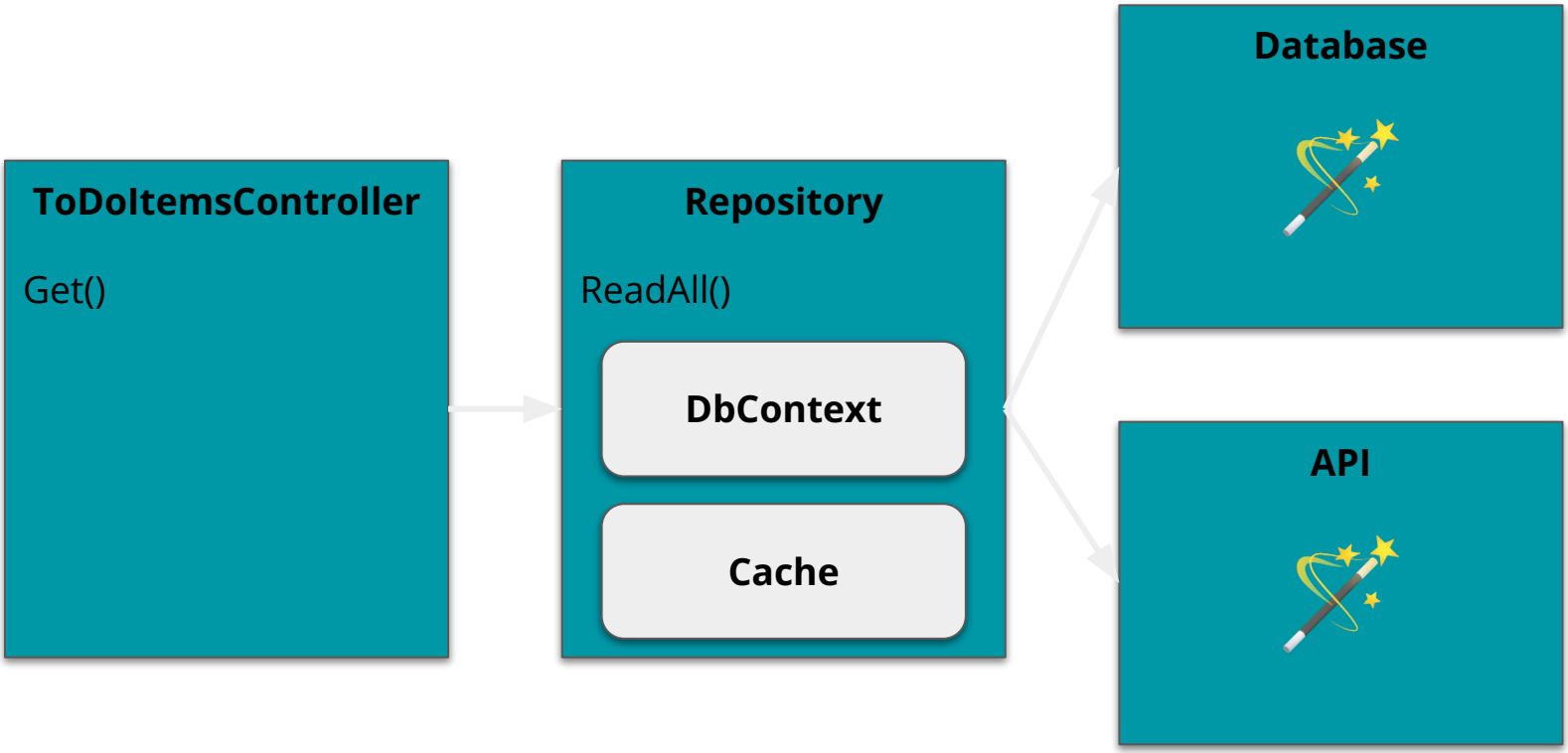


czechitas

K čemu Repository pattern?

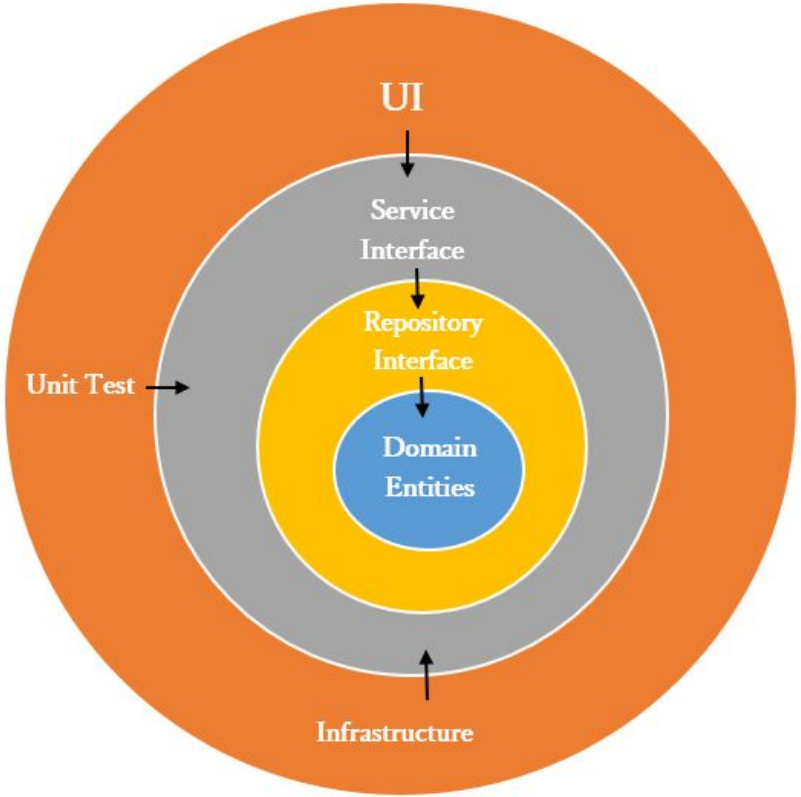
- Přepoužitelnost
- Centralizovaná logika
- Lepší testování
- “Oddělená” datová vrstva
- Může reprezentovat několik datových zdrojů
- Jednodušší vyměnit datový zdroj

Repository pro několik zdrojů



czechitas

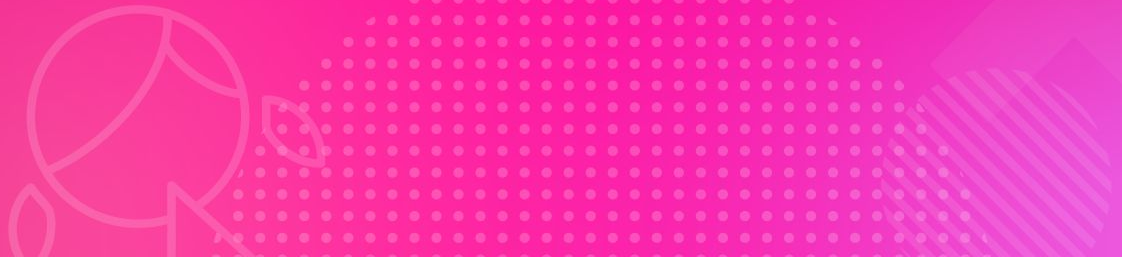
Cibulová architektura



czechitas

IRepository<T>

Mocking



Co je to mockování?

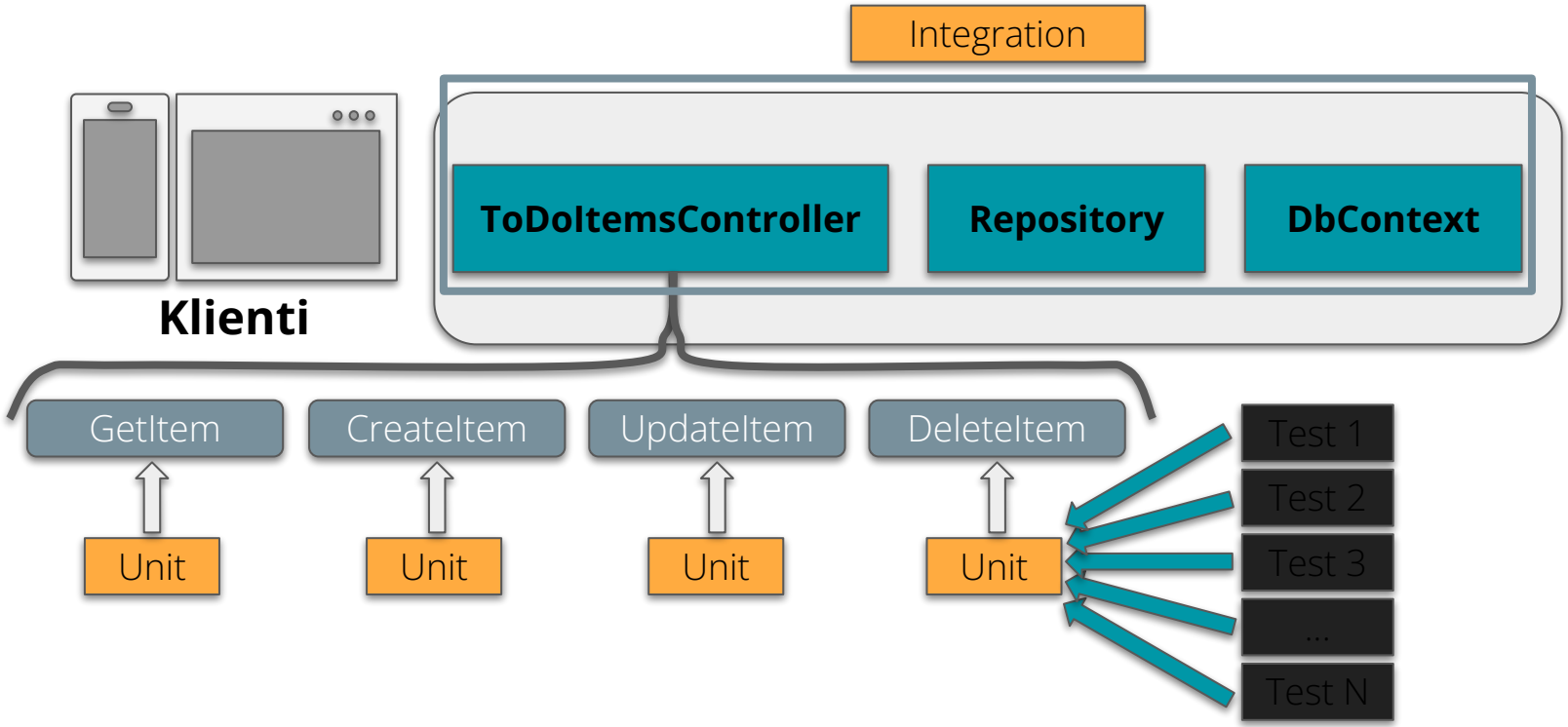
Mockování je proces vytváření objektů (Mocků), které simulují chování reálných objektů.

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

Výhody mockování

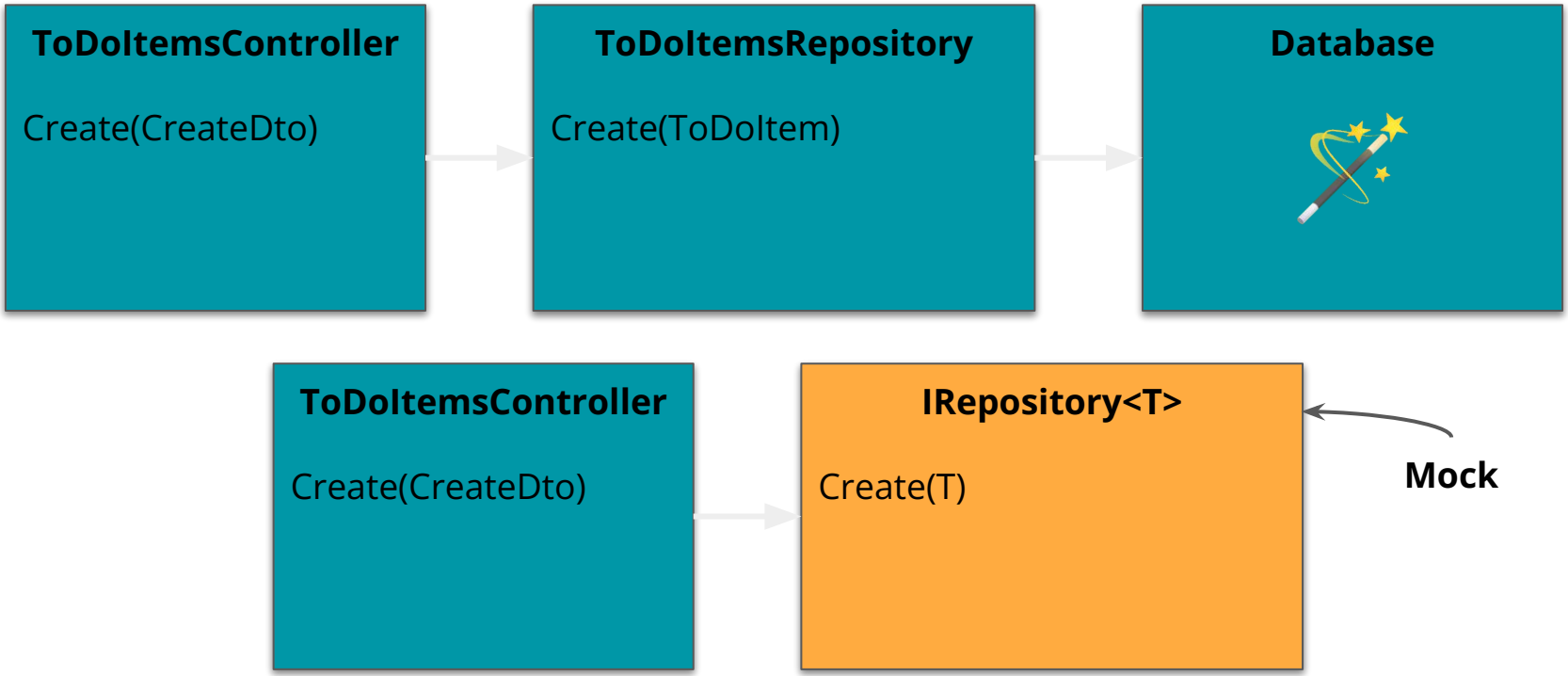
- Nemusíme testovat DB, jen naší logiku
- DB nám nerozbije testy
- Testy nespadnou při paralelním běhu

Unit Testing?



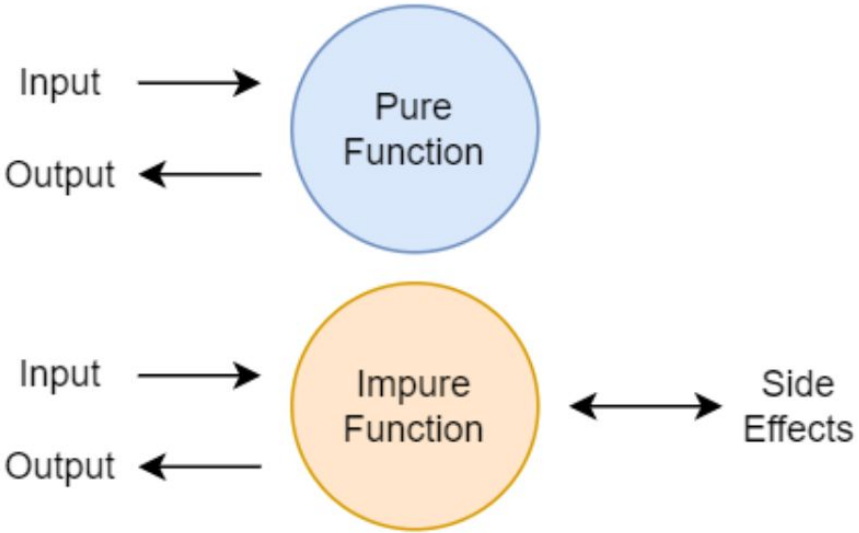
czechitas

Mockování



czechitas

Side effects



czechitas