**Architectural design patterns**

**- MVC, Multitier, Monolithic, P2P, Client/Server**

**ARCHITEKTONICKÉ NÁVRHOVÝ VZOR**

**Co to vlastně je?**

Opakovatelně použitelný a ověřený způsob řešení obecného problému

Osvědčená šablona

Dokáže usnadnit vývoj a zlepšit kvalitu výsledného produktu – KÓDU

Není potřeba vymýšlet něco svého, co nejspíš nebude ani zdaleka tak dobře řešit potřebný problém jako lidmi ověřená metodika která se už nějakým způsobem standardizovala.

Ukazují vztahy a interakce mezi **třídami** a **objekty**

Dokáže zlepšit čitelnost, údržbu, rozšiřitelnost a znuvupoužitelnost

**! ALGORITMY –** nejsou považovány za návrhové vzory

Nejsou jen z programování, jsou také v každodenním životě

* Např. architektura (stavby)

**Výhody +**

* Znovupoužitelnost
* Ulehčení údržby a nasazování nebo šklálování

**Nevýhody -**

* Může vést k příliš složitému systému (kde to není potřeba)
* Limitovány na změnách v systému
* Je potřeba znát ty patterny (ne všichni programátoří)
* Chybové využití patternu

**MVC**

* Rozděluje datový **model aplikace / uživatelské rozhraní / řídící logiku**
* může být pojat jako i architektonický vzor nebo= agregační návrhový vzor
* Skládá se ze 3 komponent
* **CONTROLLER**
  + Reaguje na události (typicky od uživatele)
  + Manipuluje s MODELEM
* **MODEL**
  + Reprezentuje nějaké informace
  + S informacemi aplikace pracuje
  + Je řízen CONTROLLEREM
* **VIEW**
  + Převádí data z MODELU do vhodné podoby
  + Podoba pro **interakci/prezentaci uživateli**
* **Obecný princip:**
  + Uživatel provede akci
  + Controller obdrží oznámení o akci
  + Controller v modelu provede potřebné změny
  + View použije obnovený Model pro zobrazení dat uživateli

# Multitier / Vícevrstvá architektura

# Princip aplikace, která je jako celek rozdělena do více funkcí/vrstev

# Jeho vrstvy vzájemně spolupracují

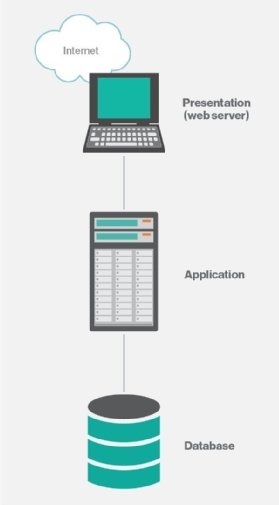
# PŘ: internetový obchod

# klientská část běží v prohlížeči uživatele

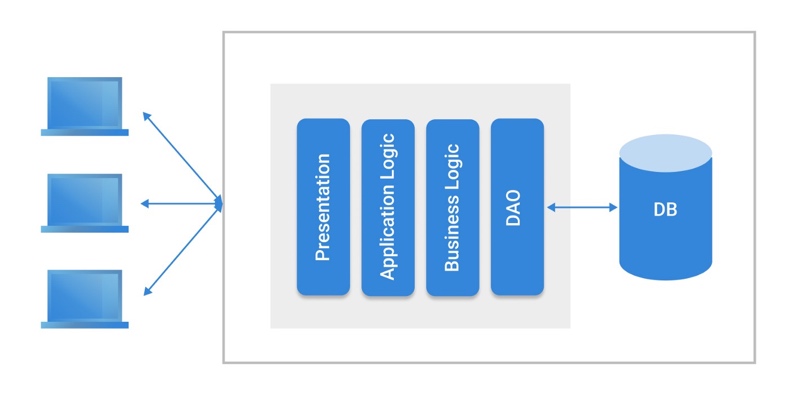
# aplikační logika na webovém či aplikačním serveru

# data sjsou uložena v databázovém serveru

# THREE TIER

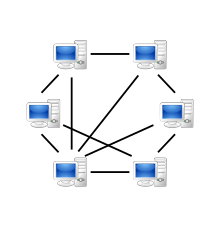
* Každý modul je alokovaný na jiný oddělený platformě
* Jednotlivý moduly mohou být vylepšovány nezávisle na sobě
* **Hlavní výhody** +
  + Dává svobodu pro vývojové týmy -> modifikování specifických částí změny celé aplikace
  + Aplikace je lehce rozšířitelná pomocí např. oddělení frontendu od databáze
  + Zamezení částí aplikace bez změn na běhu programu
  + Možnost adaptace k novým technologiím
  + Různé týmy na různých částí aplikace
* **Rozdělení**:
  + **Presentation**
    - Vrchní úroveň
    - Zobrazuje informace, které prezentují dostupné funkce
    - Tzv. **FRONTEND** pro interakci uživatele s aplikací
      * Např. **WEB**
  + **Application**
    - Prostřední úroveň
    - Logická, business úroveň
    - Ovládá hlavní funkce aplikace
    - Provádí operace
  + **Data**
    - Databázová úroveň
    - Komunikace s daty
    - Oracle, MySQL, MSSQL …

**MONOLITIC**

* Celá aplikace funguje jako jedna vrstva (jedna skupina objektů)
* Způsob se více hodí pro menší aplikace
* Spíše **ANTIPATTERN (špatný)**
* **Výhody +**
  + Lehký vývoj
  + Lehký nasazování
  + Lehký rozšíření
* **Nevýhody -**
  + Těžká na pochopení novými programátory
    - Zpomalování vývoje
  + Těžké provádění změn (k nepochopení)
  + Přetížení webu
    - zpomalení zapínání a nasazování
  + Pravidelné nasazování
    - Při malé funkci se nasadí celý program znova

**PEER-TO-PEER**

Síťová architektura

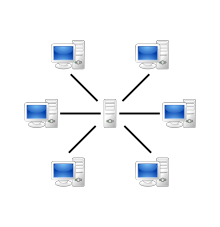
Také se nazývá **Klient-Klient**

Je to typ počítačové sítě, kde jsou klienti přímo připojený na sebe

* Opak je client-server (komunikace přes centrální server)
* Propojená zařízení působí současně jako **klienti i servery**
* V praxi se však objevují specializované servery, kteří dokáží zjednodušit návrh – Slouží **pro počáteční navázání**

Dnes se **P2P** využívá hlavně pro **výměnu dat**

* Základní **výhoda +**
  + S rostoucím množstvím uživatelů se přenosná kapacita roste
    - U client-server průměrná rychlost přenosu klesá
* Velká část obsahu šířená přes P2P je **PIRÁTSKÁ**

**KLIENT-SERVER**

Síťová architektura

Odděluje klienta a server (server je hlavní bod, který zařízení propojuje mezi sebou)

Princip:

* **Klient** žádá o služby **server** (prostřednictvím např. prohlížeče)
* Prohlížeč předá informaci serveru
* Server se dotáže na databázi
* Výsledek je zpět zaslán stejnou cestou

Je to **jedna z hlavních myšlenek síťové technologie**

Tento typ architektury je občas označován jako „**Two-tier**“

**Charakteristiky stran**:

* **Klient**
  + Je aktivní
  + Posílá žádosti na server
  + Po žádosti čeká a dostává odpovědi
  + Je většinou připojen k malému počtu serveru
* **Server**
  + Je pasivní
  + Naslouchá na síti a reaguje na žádosti
  + Při přijetí požadavku – obslouží

**Výhody +**

* Většinou rozděluje jednotlivé úkoly mezi komponenty systému
* Údaje uloženy na serveru – **zabezpečený**
  + Přímo přistupovat mohou pouze oprávněni uživatelé

**Nevýhody**

* S rostoucím množstvím uživatelů se přenosná kapacita klesá

**THIN**

* Funguje na straně serveru
* Klient používá pouze aplikaci
* Všechny operace a práce s daty je vždy pouze na serveru

**THICK**

* Zatěžuje počítač
* Ale data bere ze serveru
* Např. hra
  + Hra, která jde hrát na svém zařízení
* Používáme data ze serveru

**HYBRID**

* Skype
* Nebere data