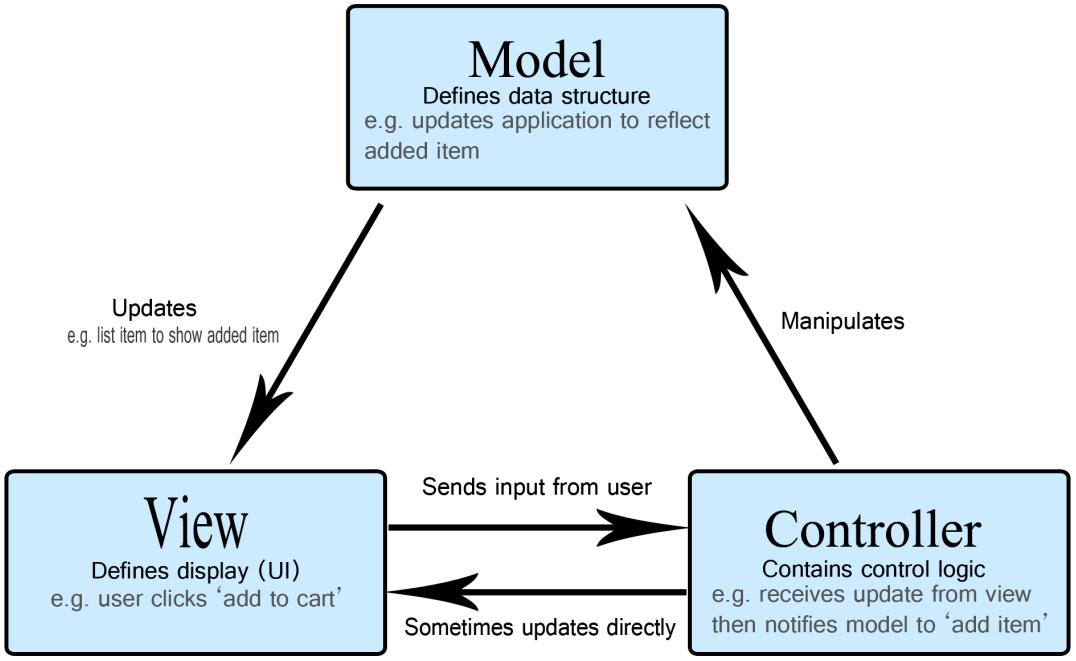
Architectural design patterns
Architektonické návrhové vzory jsou speciální typ návrhových vzorů, které se zaměřují na celkovou organizaci a strukturu softwarových systémů. Na rozdíl od klasických návrhových vzorů, které řeší konkrétní problémy na úrovni tříd a objektů, architektonické vzory definují základní strukturální organizaci celého systému. Slouží ke organizaci kódu a zajišťují škálovatelnost, bezpečnost atd…

MVC model-view controller
model rozděluje aplikaci na 3 hlavní komponenty, které navzájem komunikují

Model - obsahuje data a byznys logiku aplikace, komunikace s databází, výpočty, validace
View - Zobrazuje data a zachytává chyby, proste UI
Controller - Zpracovává požadavky - přijímá inputy uživatele, komunikace mezi view a model, řídí tok aplikace

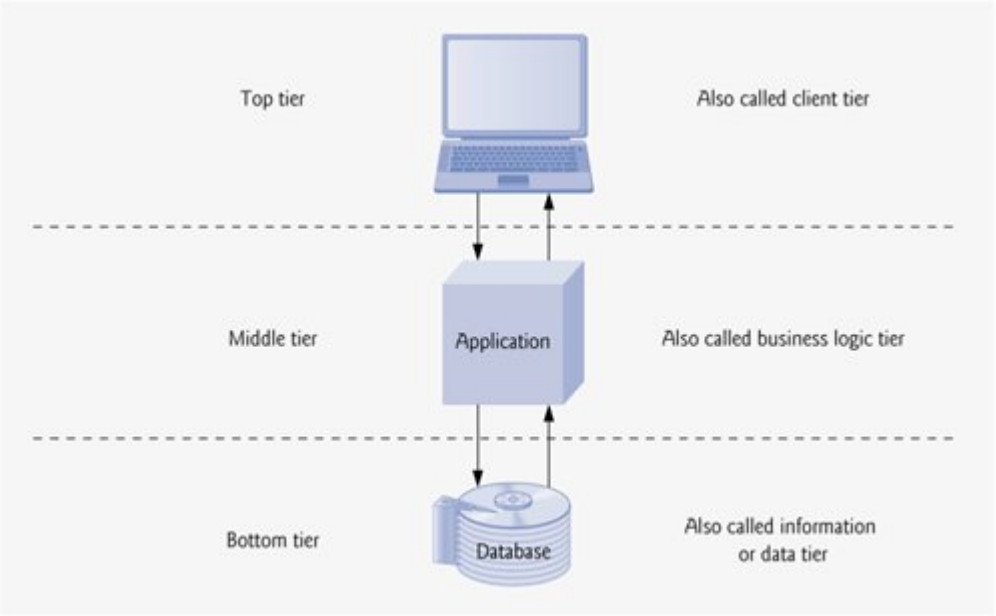


Multitier
Podobně jako mvc je rozdělen do 3 částí.

Prezentační část - uživatelské rozhraní, web/aplikace
Aplikační vrstva - řídí byznys logiku aplikace, zpracovává požadavky z frontendu, komunikuje s databází a jinými službami
Datová vrstva - uchovává a spravuje data aplikace, komunikuje s aplikační vrstvou pomocí příkazů (SQL…)

Jak to funguje:

1. Požadavek začíná v prezentační vrstvě (UI)
2. Prezentační vrstva předá požadavek aplikační vrstvě
3. Aplikační vrstva zpracuje požadavek pomocí byznys logiky
4. Pokud je potřeba, aplikační vrstva komunikuje s datovou vrstvou
5. Výsledky se postupně přidávají zpět nahoru až k uživateli



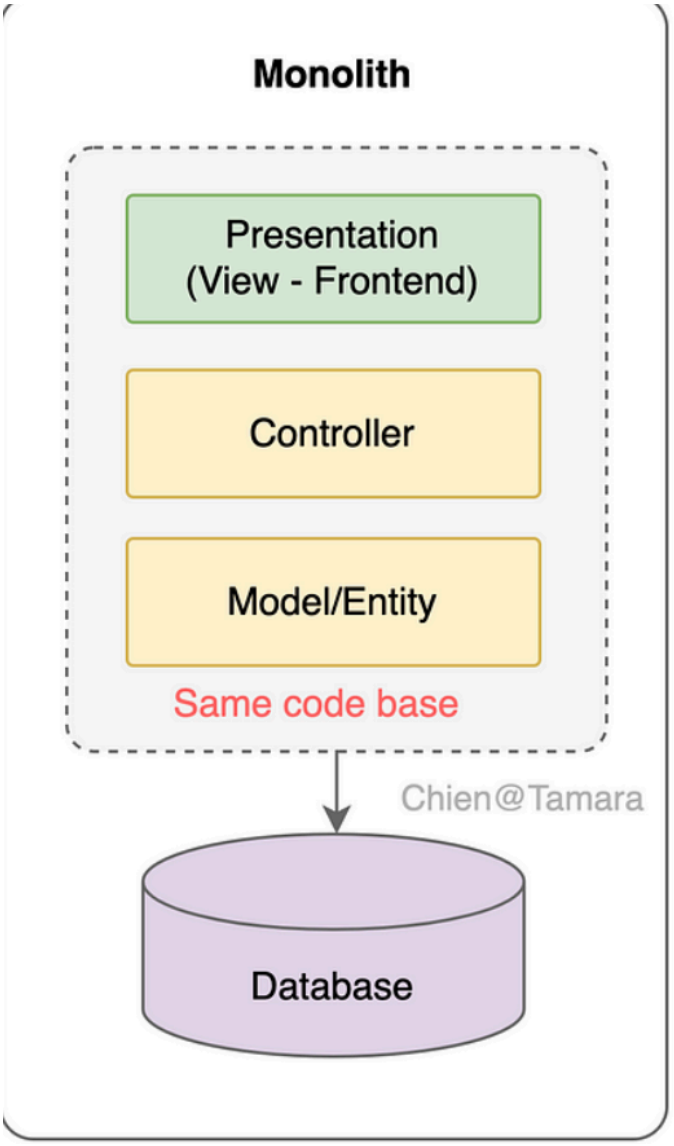
Monolithic

Architektura, která slouží jako jeden celek, kde jsou všechny komponenty spolu navzájem propojené a běží v rámci jednoho procesu

Jak to funguje:

- Veškerý kód aplikace je kompilován a spouštěn jako jeden celek
- Komponenty sdílejí paměť a zdroje
- Komunikace mezi komponenty probíhá přímo voláním funkcí nebo metod

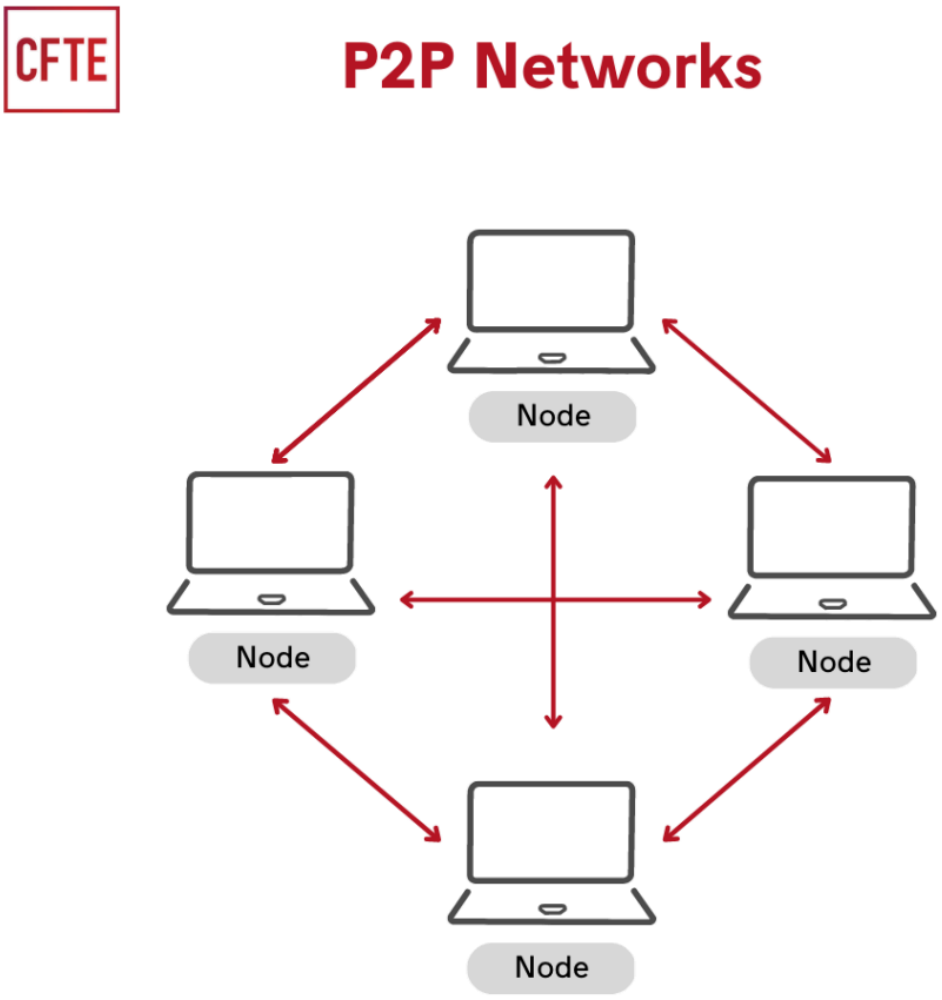
\+ snadné nasazení
\+ rychlost
\+ jednoduché testování
\- náročná údržba
\- špatná škálovatelnost



P2P
Architektura umožňující přímou komunikaci mezi jednotlivými uzly v síti bez potřeby centrálního serveru. Každý uzel může fungovat jako klient i server zároveň a může posílat i příjmat data. Například torrenty fungují na architektuře peer to peer, Mezibakovní systém (prostě ten jeden projekt na PV) a ještě třeba Bitcoin.

+ odolnost proti výpadkům = výpadek uzlu neovlivňuje celý systém
+ škálovatelnost = s počtem uzlů roste výkon i kapacita

- bezpečnostní rizika = více potencionálních bodů pro útok
- složitá implementace = řešení synchronizace, konzistence



Client - server TCP
Klient komunikuje se Serverem, který poskytuje služby, data a nebo výpočty, probíhá asymetrická komunikace. Účelem architektury je oddělení funkcí a odpovědností mezi klientskou a serverovou částí.

Hlavní dvě části

1. Klient - požaduje služby, data a výpočty,
   GUI, kontrola vstupů od uživatele, odesílání požadavků

2. Server - poskytuje služby, data a výpočty

Typy client-serverů

1. Thin client

Typ klienta, který má minimální funkcionalitu a především se spoléhá na server pro zpracovávání dat a aplikační logiku. Výhodu má v tom, že když se provádějí všechny funkce na serveru, tak se tolik nezatěžuje HW na straně klienta. Zase je nutné mít stabilní připojení k internetu

Prohlížeč načítající stránku ze serveru, vzdálené plochy….

2. Thick client

Typ klienta, který naopak od thin clienta, obsahuje významnou část aplikační logiky a může fungovat nezávisle i bez neustálého spojení se serverem. Často je poměrně náročný na HW, kvůli operacím.

Desktopové aplikace, Grafické editory, Hry

3. Hybrid client

Kombinuje vlastnosti thick a thin clienta. Plně využívá výhodu obou zařízení.

Mobilní aplikace, Email klienty