53.) LOGICKÁ ARCHITEKTURA SW

Softwarová architektura

- uspořádání a organizace jednotlivých SW částí a modulů včetně určení jejich rolí a definování jejich rozhraní
- určuje jak jsou tyto prvky organizovány do skupin, nebo to, jak se jeden prvek či modul sestává z jiných
- jak jsou tyto komponenty provázané, jaké mají závislosti a jak spolu komunikují,
- organizace systémů, jejich podsystémů a modulů a komponent a jejich vzájemná komunikace
- je to nedílnou součástí fáze návrhu při vývoji SW či informačního systému
- může být v UML reprezentována jak diagramem tříd tak i diagramem architektury (není součástí UML ale v AIS jsme ho na projektu použili), diagramem nasazení, komponent a balíčků ...
- třeba SW architektura MVC, MVVM, SOA a s ními WS, ...
  - XML
  - SOAP
  - WSDL

Architektura logická

- udává rozřazení tříd do balíčků a vrstev architektury a udává jim namespace a také samozřejmě ty třídy identifikuje
- používá se diagram balíčků a může i tříd a komponent (UML)

Architektura nasazení

- opak logické architektury
- udává kde budou jaké části, moduly a komponenty konkrétně nasazeny a třeba i na jakém potřebí HW
- konkrétní rozložení na klientské PC, co bude na serveru a jaký server se použije - SOLARIS + OS UBUNTU + WEBSERVER NGINX, co bude v DB a jaká to bude - MARIADB ...
- v UML se používá diagram nasazení

Vrstvená architektura

- rozdělení architektury SW do vrstev pomáhá lepšímu pochopení ze strany vývojářů, dodává přehlednost, větší srozumitelnost, clean code, udržovatelnost, refactoring a rozšiřitelnost
- je možné vidět jednotlivé komponenty, jejich logické rozdělení do vrstev a komunikaci
- hierarchické rozdělení
- vyšší vrstva architektury využívá nižší - nemělo by být naopak (viz TCP/IP stack)
- komunikace komponent v rámci vrstvy za použití komponent nižších vrstev
- podporuje rozdělení práce a práci v týmu
- třeba 3-vrstvá architektura - dole datová/technická vrstva + log atd, nad ní řídící a logická vrstva a poté nejvýš prezentační - uživatelské rozhraní

- doménový objekt – reprezentuje doménu, část dat
- striktní vrstvená arch. – vrstva může komunikovat jen s bezprostředně nižší
- uvolněná vrstvená arch. – vrstva může komunikovat se všemi nižšími
- mělo by platit: – že čím nižší vrstvy tím jsou méně závislé na zbytku systému a jsou téměř samostatné a často se neupravují
  - vyšší vrstvy mohou záviset na nižších a upravují se častěji
  - třeba uživatelské rozhraní (nejvyšší vrstva) se upravují nejčastěji

Vrstvy, balíčky, třídy a jejich závislosti

- balíček představuje několik elementů, entit a tříd v jednom jmenném prostoru, které spolu nějak logicky souvisí a utváří celek či jeden z modulů či komponent SW
- balíčky mohou být i zanořeny
- nějaké balíčky mohou patřit pouze do jedné vrstvy architektury, jiné mohou být napříč vrstvami
- závislosti:

tímto směrem mohou závislosti vznikat

- metod – jedna přímo volá druhou
- tříd – třída jednoho balíčku například vyžaduje nebo třeba jako parametr přijímá nebo jako atribut má třídu jiného balíčku (nebo vlastního balíčku)
- balíčků – obsahuje třeba třídy které závisí na třídách jiných balíčků
  - jestliže balíček A závisí na B tak úpravy v B si vyžádají úpravy v A většinou
- vrstev – pokud jsou závislé balíčky jedné vrstvy na balíčkách druhé
  - měly by být jen vyšší vrstvy na nižších
- závislosti mohou vznikat vlivem: atributem jiné třídy, parametrem metody, voláním jiné metody, dědičností, ...
- cyklická závislost: A ⟺ B problém
- řešení – vyjme se část A co vyžaduje B a vloží se do třídy A2 a A se stane závislé na A2 ale pořád i na B a B jen na A2

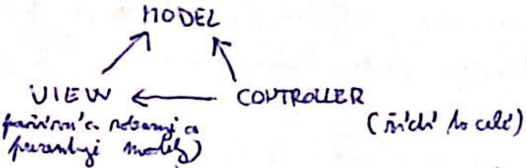
$$\begin{tikzcd} A \arrow[r, leftrightarrow] & B & \Rightarrow & A \arrow[rr] \arrow[dr] & & B \arrow[dl] \\ & & & & A2 & \end{tikzcd}$$

– už není cyklická závislost

Oddělení pohledu (view) a modelu

- v pohledu, tedy UI nebo také uživatelském rozhraní se odehrává nejvíce změn a úprav – je tedy dobré aby žádné nižší vrstvy nebyly na pohledech závislé (jinak by se musely také měnit) a pohledy mohou být závislé na nich
- není vhodné vkládat aplikační logiku, řešení ani rozhodování do UI ale raději ho držet na nižších vrstvách
- objekty nižších vrstev by neměly znát objekty UI
- můžeme pak třeba vyměnit celé UI a na nižších vrstvách nemusíme nic upravit
- na webech třeba z Tailwind CSS frameworku na Bootstrap 4 FW

Architektonické vzory



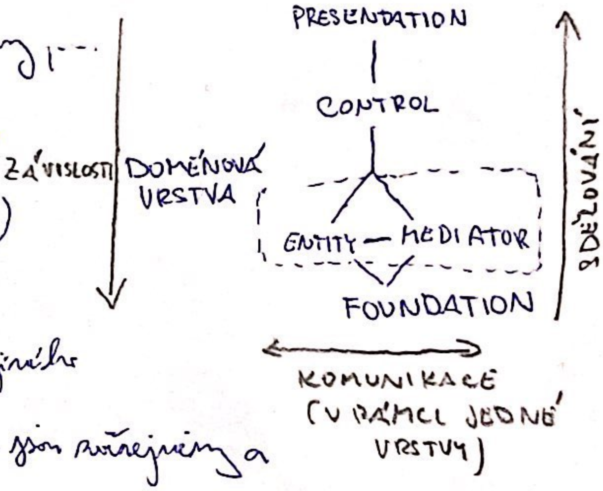
MVC (Model-View-Controller)

- rozděluje architekturu na tři více méně (do jisté míry) samostatné části
- model – datová vrstva, doména, objekty domény, dokonce přidává trochu aplik. logiku
- view – uživatelské rozhraní, zobrazení dat, prezentace modelu
- controller – řídící logika celé aplikace / architektury, komunikuje, reaguje na události
- známe -> Laravel

PCMEF (Presentation + controll, mediator, entity, foundation)

- hierarchická struktura obsahující 5 vrstev do kterých se třídy rozřadí
- presentation – prostě UI (nejvyšší) (Blade)
- control – obsluhuje požadavky na řízení a reakci na události atd.. (Controller)
- mediator – práce s daty, prostředník mezi DB a vyšší vrstvou
- entity – uchovávání dat
- foundation – přímá DB vrstva, DB dotazy, ...

(entity + mediator tvoří doménovou vrstvu
(migrace a query builder) (model eloquent, atd)



SOA (Service-Oriented Architecture)

- architektura orientovaná na služby
- služby jsou distribuované a každá může mít jiného vývojáře a vlastníka
- je dán standard jak jsou služby popsány a kde jsou zveřejněny a jak jsou invokovány
- služby dostupné přes rozhraní, jsou heterogenní, každá může být jinak implementovaná
- služby jsou bezstavové – vstup vrátí výstup a je to
- jsou nezávislé mezi sebou a nezávislé na platformě
- jsou standardizované !
- SOA podporuje business proces – služby realizují jeden či více procesů, něco vypočítají, zpracují data, reagují na vstup, zpracují dokumenty...

poskytovatel služeb – tvoří a poskytuje služby spotřebitelům (service provider)
⇓
registr služeb – centrální standardizovaný registr služeb SOA (service registry)
⇑
spotřebitel služeb (service consumer)

- služby mohou komunikovat nejen se spotřebitelem ale i mezi sebou a poskytovat si navzájem vyžádaná data či funkčnost a společně tvořit business proces
  ↳ choreografie
- kooperace
- agregace – jedna je z více služeb
- orchestrace – jedna řídí ostatní za účelem vykonání procesu

-> dělá se i servisně orientovaná analýza a návrh

Web Services (WS – webové služby)

- konkrétní webová implementace SOA / volá se služba a její metody s parametry atd.
- protokol pro komunikaci je SOAP – komunikace v XML
- UDDI – adresář služeb
- WSDL – web services description language – jazyk pro popis WS, co service umí a poskytuje a kde je a kdo je vlastní, jak komunikuje
  - jaké má metody a jaké ty metody mají parametry