# Analýza

## Popis navrhovaného systému.

Jak bylo uvedeno výše, systém se bude zaobírat monitorováním činnosti práce veřejných činitelů. Systém bude veřejně přístupný, pro běžné uživatele, a pokud bude mít uživatel práva administrátora, bude mít možnost do systému vkládat komentáře, informace a dokumenty.

Jelikož systém budou obsluhovat uživatelé bez speciálního IT vzdělání, je potřeba aby systém měl kromě stability taky jednoduché, intuitivní ovládání.

Systém by měl být dostatečně adaptabilní, aby pokryl rozmanitost skutečného stavu. (Tj. umožnil pokrýt rozmanitost typů veřejných orgánů, a všech situací, které mohou nastat.)

Nicméně návrh kompletního systému je na absolventskou práci příliš rozsáhlý, je tedy potřebné přistoupit k zjednodušení, tedy vytvořit jenom základní funkční kostru, která bude otevřená dalšímu rozvoji.

## 0. Všeobecné požadavky na systém.

- Systém je přístupný skrz běžný web. prohlížeč, ve formě zadané internetové stránky (např. www.kosvopo.sk).

- Systém bude běžet na serveru v režimu 24/7. Měl by být dostupný všem uživatelům dobré vůle, z jakéhokoli místa sítě.

- Ukládání dat v perzistentní podobě.

- Uchovávání vnitřní historie, tj. historie akcí v systému.

- Systém musí všechna relevantní data ukládat permanentně, tj. události se nesmí vázat jenom na operační paměť.

- Bezpečnost

## A. Úroveň běžného uživatele

- bezpečná registrace a přihlášení

- přístup k datům a informacím odpovídajícím úrovni oprávnění (čtení obsahu a stahování uložených dokumentů)

- možnost vkládat komentáře k diskusi

- možnost dostávat e-mailem upozornění na sledované události

## B. Úroveň uživatele s vyššími právy

- úroveň A +

- vkládání / odstraňování dokumentů a hodnocení

- vkládání / odstraňování v systému figurujících entit (veřejně činných osob, veřejných orgánů, atd..)

- Přístup k statistikám a možnost jejich vyhodnocování, tvorba „reportů“

- Přístup k historii akcí.

## C. Úroveň programátora, který bude systém spravovat.

- Systém je schopen fungovat i bez soustavné odborné podpory. Tj. je stabilní

- Snadná údržba kódu

- Možnost relativně snadně měnit funkcionalitu a implementační detaily systému (typ úložiště dat, nezávislost na výstupním médiu) snadná rozšiřitelnost.

- nezávislost na výstupním médiu (PC, Tablet,... )

# Analýza řešení požadavků

## 0. Všeobecné požadavky na systém.

Úkol je možné řešit různými způsoby, jedním z cílů této práce je však vyhnout se „klasickým“ spůsobům tvorby webových stránek, tj. za použití kombinace PHP, JavaScript a HTML a vyzkoušet některý z frameworků, které se v poslední době hojně rozmohly. Jejich hlavní výhodou je, že umožňují programátorům, zvyklým na vyvinutější programovací jazyky (Java) tvorbu webových aplikací, přičemž je odstiňují od základnějších technologií na kterých jsou ve skutečnosti postaveny. Programátor se tak nemusí rozptylovat obeznamováním se s detaily Javascriptu a HTML a zároveň je mu umožněno využívat silných stránek jemu známého jazyka, který tvoří vrstvu ve které jsou tyto základnější technologie obaleny. Tímto způsobem může poměrně snadno tvořit i stránky s komplikovanou funkcionalitou, co by bylo kupříkladu v prostším PHP problematické.

## Ukládání dat.

Standardním řešením je relační databáze založena na jazyku SQL, případně jeho dialektu. Vzhledem k tomu jak byla úloha zadaná, tj. nejsou kladeny speciální nároky jak na objem ukládaných / zpracovávaných dat a nejsou kladeny speciální nároky ani co se týče stupně bezpečnosti práce s daty. Nevidím smysl spekulovat nad jinými možnostmi (např. XML databáze), každé další řešení je za této konstelace zbytečná komplikace. Nad jinými typy ukládání dat, se nebudu ani zamýšlet. Data tedy budou ukládány ve formě databázových tabulek.

## Uchovávání vnitřní historie, tj. historie akcí apod.

V daném systému bude vhodné, aby se relevantní neaktuální záznamy nemazaly, ale byly potenciálně stále k dispozici. Na druhé straně nemohou být pro běžného uživatele viditelná. Tento problém budeme řešit zavedením pomocné proměnné isVisible, která bude v databázi uchovávat stav viditelnosti dané entity pro běžného uživatele.

## Spravovatelnost systému

S postupující komplikovaností softwarů dochází k otázce rozvržení projektu tak, aby bylo co nejpřehlednější, snadno udržovatelný a přitom co nejfunkčnější. Jádrem tohoto dění je izolace a separace podobných procesů, tj. funkcionalit. Pokud danou funkcionalitu v systému izoluju, mohu ji, jako celek vyměnit za jinou, bez toho, aby to zasáhlo další části programu. Takový kód je potom daleko přehlednější a jeho udržování je mnohem snadnější. V programátorské praxi dominuje izolační architektura (resp. návrhový model, což je častější označení) MVP (Model - View - Presenter).

Návrhový model MVP (Model-View-Presenter) vychází z modelu MVC (C=controller)

„Model“ v zjednodušení část Modelu by měla zahrnovat-popisovat základní vztahy vnitřní logiky programu, tj. vztahy mezi DB entitami, a tzv. bussiness logiku, tj. jak se má s vnitřními datami při té - které příležitosti zacházet, definovat mapu vnitřních stavů systému a přechodů mezi mini.

„Presenter“ – zahrnuje vrstvu, která se stará jak o ovládání změn dat v modelu, povětšinou na základě požadavků uživatele, tak i o přenos, resp. “zadrátování” změn hodnot dat v modelu tak, aby se projevili do změn ve „View“, který představuje zobrazovací i ovládací prvek, většinou s výstupem na monitor.

Ve všeobecnosti možno hovořit on MVP v případe „těžkých“ View, tj. kdy stav View závisí na stavu Modelu netriviálně, kde nelze jednoznačně rozlišit mezi vstupem a výstupem. Např. ukázkovým případem jsou Web stránky, např. tlačítko na web stránce může sloužit jak na výstup (objeví se, nebo zmizne) jako i vstup (stlačením dává uživatel pokyn). Naopak MVC je aplikovatelný v případě odlehčených“ View, kde jsou vstupy a výstupy striktně odděleny. Jako příklad bych uvedl fyzický knoflík jako vstup a diodu jako výstup – indikuje změnu stavu tak-řečeno napřímo.

Model v našem případě bude představovat kód řídící komunikaci s databází a business logiku systému.

Presenter se bude starat o přenos dat z Modelu do View, který představuje samotnou zobrazovací / ovládací část. Tento model se budu snažit zachovat i já.

### Přihlašování

Přihlašování musí splňovat několik všeobecných podmínek, které jsou na přihlašování kladeny. Jsou to:

- bezpečnost, rozuměj odesílání hesla z formuláře zabezpečenou linkou.

- bezpečnost při ukládání hesla do DB.

Opět, jelikož ukládané data nejsou zvlášť choulostivé, budu vybírat jenom ze standardních řešení v rámci frameworku.

## Závěr

Všechny tyto požadavky v dostatečné míře splňuje framework Vaadin, který byl právě pro tento typ aplikací navržen. Konkrétně je to pro následovní jeho vlastnosti.

* Komplexní kompatibilita s ostatními webovými technologiemi. Jako např. HTML.
* Modularita. Tj. vlastnost umožňující přidávat a odebírat již hotové celky - moduly, s minimální přidanou námahou na jejich „zadrátování“. Tato vlastnost umožňuje lehkou rozšiřitelnost (extensibility)
* Podporuje MVP model.
* Prakticky nevyžaduje znalost žádných jiných web technologií, mimo rozsah Javy.
* Nezávislost na platformě.
* Mnoho nejčastějších úkonů a témat (přihlašování, registrace, propojení dat s grafickými prvky) je již vyřešeno do standardní podoby, není tedy třeba znovu objevit kolo..

Nevýhody:

* Ukrývání technologií, na kterých systém běží před zraky programátora. Tj. všechno je perfektní až do doby, když to přestane fungovat a hlavní výhoda se zvrátí v hlavní nevýhodu. (Celkově však možno na adresu Vaadinu říci, že byl napsán vcelku solidně.)
* Propojenost s IDE a dalšími technologiemi (Maven, Ivy, Spring, etc.), beze kterých to de facto nejde zesiluje efekt „stavby babylonské věže“ tj. nevýhody používání černých skřínek. A prodlužuje učící se křivku. Hodně času člověku zabere, než mu vývojové prostředí funguje tak, jak má.

O nevýhodách se ovšem člověk dozví dodatečně po vnoření se do systému.