Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj webové aplikace v jazyce Java

Vladimír Hynek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Vladimír Hynek

Informatika

Název práce

Vývoj webové aplikace v jazyce Java

Název anglicky

Web application development using Java

Cíle práce

Práce je zaměřena na problematiku vývoje webových aplikací s využitím jazyka Java. Hlavním cílem je analýza a implementace webové aplikace "Provozní deník radiolokátoru". Dílčím cílem je popis vývoje webových aplikací na této platformě.

Metodika

Práce sestává ze dvou částí, teoretické a praktické.

Teoretická východiska pro zpracování praktické části, představující první část práce, budou popsána na základě syntézy poznatků získaných studiem odborné literatury.

Praktická část práce spočívá v analýze, návrhu a implementaci webové aplikace určené pro vedení evidence provozního deníku radaru. Během analýzy a návrhu bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Webova aplikace; Java;

Doporučené zdroje informací

Boronczyk T. MySQL Okamžitě. ISBN 978-80-251-4737-5
oracle.com
SCHILDT, H. Mistrovství – Java. Comuter Press 2014. ISBN978-80-251-4145-8
spring.io
VIRIUS, M. *Java pro zelenáče.* Praha: Neocortex, 2001. ISBN 80-902230-9-5.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 23. 2. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 2. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 02. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj webové aplikace v jazyce Java" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14. 3. 2018 _____

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu bakalářské práce Ing. Jiřímu Brožkovi, Ph.D. za konzultace a odborný dohled, dále konzultantovi Ing. Michalu Wokounovi za cenné rady a nasměrování při samotné implementaci a své ženě za trpělivost a podporu.

Vývoj webové aplikace v jazyce Java

Abstrakt

Tato bakalářská práce se zabývá problematikou vývoje webové aplikace. V úvodní teoretické části jsou představeny technologie a nástroje, které jsou spjaty s vývojem na platformě Java. Jedná se o framework Spring, vývojové prostředí IntelliJ IDEA, nástroj pro řízení a automatizaci buildu Maven. Tyto nástroje jsou následně využity pro praktickou část vlastního vývoje.

Praktická část se zabývá analýzou, návrhem a následnou tvorbou webové aplikace Provozní deník radiolokátoru. V implementační části je popsán celý postup rozběhnutí, nastavení projektu a detailně popsány všechny důležité části projektu.

Klíčová slova: Webová aplikace, Java, Spring Boot, Maven, jQuery, DataTables, MySQL

Web application development in Java

Abstract

This bachelor thesis deals with the development of web application. The introductory theoretical part introduces technologies and tools, that are related to the Java platform development, such as the Spring framework, the IntelliJ IDEA development environment, the Maven building management and automation tool, which are then used for the practical part of their own development.

The practical part deals with the analysis, design and creation of a web application Provozní deník radiolokátoru (Radar logbook). The implementation part describes the entire process of starting and setting up the project and describing in detail all the important parts of the project.

Keywords: Web application, Java, Spring Boot, Maven, jQuery, DataTables, MySQL

Obsah

Seznam obrázků

Seznam kódů

1 Úvod

Bakalářská práce na téma „Vývoj webové aplikace v jazyce Java“ byla autorem vybrána z důvodu osobního zájmu o programování v jazyce Java. Jazyk Java je v dnešní době velmi rozšířen a žádán na pracovním trhu.

Webová aplikace s využitím platformy Java EE, frameworku SpringBoot a MVC architektury byla vybrána autorem na doporučení, jako vhodný způsob důkladného seznámení se světem Java programování.

Během práce budou představeny všechny technologie, které stojí za chodem webové aplikace a technologie, které velmi usnadňují samotný vývoj a implementaci. Během samotné implementace budou představeny všechny kroky nutné k rozběhnutí projektu a všechny klíčové části aplikace budou vysvětleny na příkladech s využitím Java kódu aplikace.

S prostředím provozu a oprav radiolokátoru má autor čtrnáctiletou pracovní zkušenost, proto cílem praktické práce je vývoj webové aplikace „Provozní deník radiolokátoru“. Deník bude sloužit k usnadnění práce autora s ukládáním záznamů o provozu radiolokátoru a jako základní analytický nástroj pro vyhodnocování provozu a poruch radiolokátoru.

Celý vývoj probíhal ve vývojovém prostředí IntelliJ Idea. Prostředí bylo vybráno s ohledem na usnadnění vývoje. Prostředí nabízí skvělé nástroje pro odhalování chyb a jejich odstranění. Další software, který byl použit je MySQL pro chod databáze a MySQL WorkBench pro přímou práci s databází během testování.

2 Cíl práce a metodika

2.1 Cíl práce

Práce je zaměřena na problematiku vývoje webových aplikací s využitím jazyka Java. Hlavním cílem je analýza a implementace webové aplikace "Provozní deník radiolokátoru". Dílčím cílem je popis vývoje webových aplikací na této platformě.

2.2 Metodika

Teoretická východiska pro zpracování praktické části představující první část práce, budou popsána na základě syntézy poznatků získaných studiem odborné literatury.

Praktická část práce spočívá v analýze, návrhu a implementaci webové aplikace určené pro vedení evidence provozního deníku radaru. Během analýzy a návrhu, bude využito standardních metod softwarového inženýrství, implementace bude provedena s využitím jazyka Java a souvisejících technologií.

Aplikace bude dále otestována, budou shrnuty poznatky získané během jejího vývoje a navrženy případné další možnosti jejího budoucího rozvoje.

3 Teoretická východiska

3.1 Java

Java je objektově orientovaný programovací jazyk a zároveň je to platforma či prostředí, na kterém běží Java programy. [1]

Javu byla vyvinuta roku 1991 firmou Sun Microsystems. Hlavní autoři byli James Gosling, Patrick Naughton, ChrisWarth, Ed Frank a Mike Sheridan. [2] První funkční verzi vyvinuli za 18 měsíců a do roku 1995 se jmenovala Oak (překlad dub). Javu vyvíjeli jako multiplatformní tzn. nezávislou na architektuře počítače. Toho bylo dosaženo tím, že zdrojový kód Javy se zkompiluje do bajtového kódu a následně je interpretován virtuálním strojem Javy (JVM Java Virtual Machine) přímo do strojového jazyka, který běží na daném počítači. „Virtuální stroj Javy má i několik dalších výhod. Zvyšuje bezpečnost programů, jelikož hlídá za programátora některé potenciálně nebezpečné operace a zabezpečuje bezpečnou správu paměti.“ [3]

„Platformy programovacího jazyka Java

- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)
- Java Micro Edition (Java ME)
- JavaFX“ [1]

Všechny Java platformy obsahují Java virtuální stroj (Java Virtual Machine) a API (application programming interface). API je rozhraní pro programování aplikací. Obsahuje kolekci procedur, tříd, funkcí, které usnadňují vývoj počítačových programů. [1]

Java SE poskytuje základní funkci programovacího jazyka. Je v ní definováno vše od základních metod programovacího jazyka až po složitější třídy, které se používají pro sítě, přístup k databázi, zabezpečení a grafického rozhraní GUI (Graphical User Interface).

Java EE je postavena na platformě Java SE. Poskytuje nástroje pro vývoj rozsáhlých, vícevrstvých, škálovatelných a zabezpečených síťových aplikací.

Java ME je podmnožinou Java SE. Jejím hlavním úkolem je zabezpečit vývoj aplikací pro malá zařízení jako například mobilní telefon. Poskytuje speciální třídy knihoven užitečné pro tato zařízení. Aplikace Java ME jsou často klienty služeb platformy Java EE.

JavaFX je platformou pro vývoj bohatých internetových aplikací RIA (Rich Internet Applications). Aplikace JavaFX využívají hardwarově akcelerované grafiky. Podporují přehrávání webového multimediálního obsahu. [1]

3.1.1 Java EE

Platforma Java EE je navržena tak, aby vývojářům pomáhala vytvářet takzvané podnikové aplikace (Enterprise Applications). Aplikace jsou nazývány podnikové, protože z počátku řešily hlavně problémy, které byli doménou velkých podniků a následně rozličných agentur a vlád. Dnes jsou tyto aplikace přínosné a někdy i nezbytné i pro daleko menší subjekty. [4]

„Funkce, které činí podnikové aplikace natolik zajímavé, jako jsou bezpečnost a spolehlivost, je také velmi často činí velice složitými. Platforma Java EE je navržena tak aby snižovala složitost vývoje podnikových aplikací tím, že poskytuje vývojový model, API a běhové prostředí, které umožní vývojářům soustředit se na požadovanou funkčnost.“ [4]

3.1.2 Vícevrstvá architektura

Vícevrstvé aplikace jsou rozděleny do izolovaných funkčních oblastí nazývaných vrstvy. Sousedící vrstvy spolu komunikují prostřednictvím rozhraní a povětšinou běží na odlišné výpočetní infrastruktuře. Komunikace přes definované rozhraní a různá infrastruktura umožňuje snadnější změny bez velkých dopadů na funkci celé aplikace. Vícevrstvé aplikace mají vrstvy klientskou, webovou, business (řídící) a vrstvu podnikových informačních systému EIS (Enterprise Information Systems). Vývoj aplikací Java EE se soustřeďuje na business vrstvu. [4]

Úkolem klientské vrstvy je zasílání požadavků webové vrstvy a přijímat odpovědi a korektně je zobrazovat. Aplikace klientské služby je webový prohlížeč či aplikace vytvořená pro komunikaci s danou službou, která ani nemusí být napsaná v Javě. [4]

„Druhou vrstvou je webová vrstva. Komponenty webové vrstvy zpracovávají interakci mezi klientskou a business vrstvou. Hlavním úkolem webové vrstvy je:

- Dynamicky generovat obsah v různých formátech.
- Sběr vstupů od uživatele a zobrazení výsledku od business vrstvy.
- Provádí základní logiku a dočasně ukládá data.

Základní technologie používané ve webové vrstvě:

- Servlety
- JSP JavaServer Pages
- JSF JavaServer Faces
- Java Beans Components“ [4]

Business vrstva se skládá z funkcí, které poskytují logiku aplikace. Jedná se o jádro celé aplikace, které vše řídí. [4]

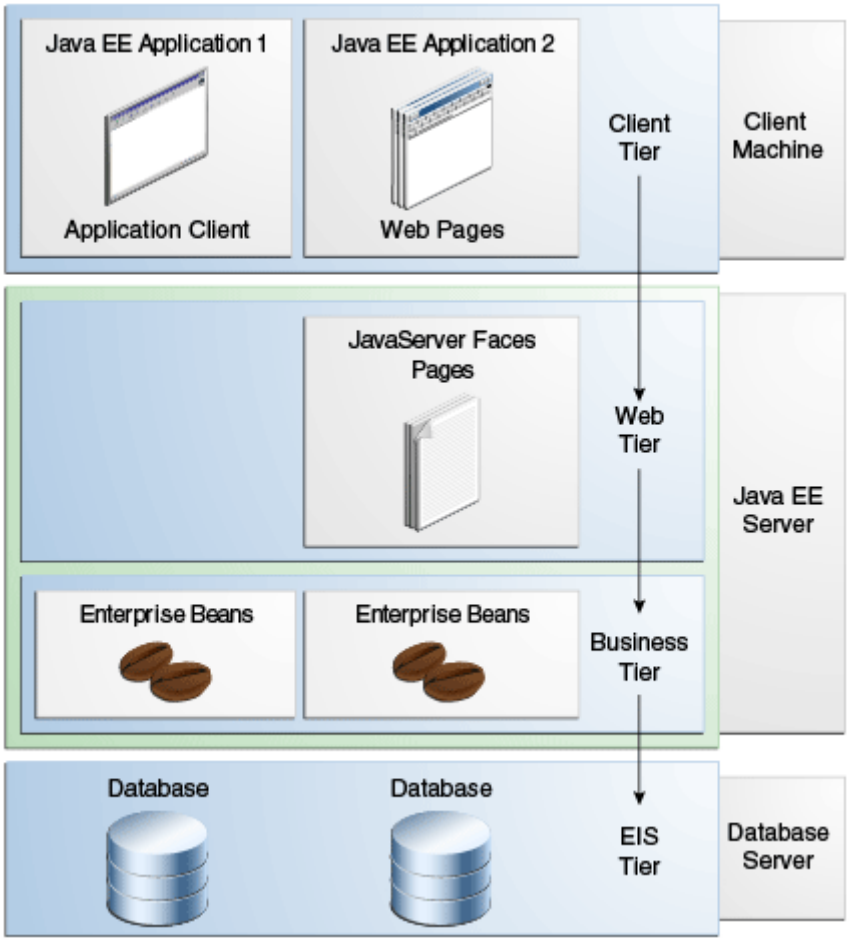
„Základní technologie užívané v podnikové vrstvě:

- Enterprise JavaBeans
- JAX-RS RESTful web services
- JAX-WS web service endpoints
- Java Persistence API entities
- Java EE managed beans“ [4]

Poslední vrstvou je vrstva podnikových informačních systému EIS (Enterprise Information Systems). Skládá se z databázových serverů, systémů pro řízení podnikových zdrojů a dalších zdrojů dat. [4]

„Technologie užívané v datové vrstvě:

- The Java Database Connectivity API (JDBC)
- The Java Persistence API
- The Java EE Connector Architecture
- The Java Transaction API (JTA)“ [4]



Obrázek 1- Java vícevrstvá architektura [1]

3.1.3 **Aplikační server**

Aplikační server je software, který zastřešuje všechny funkcionality námi vyvíjené aplikace a běží trvale. Jeho úkolem je přijímat požadavky a dle funkcionality systému na ně i odpovídat. Skládá se z několika komponent. Server poskytuje tyto komponenty ve formě kontejneru.

„**Kontejnery Java EE** jsou rozhraní mezi komponentou a funkcemi nižší úrovně poskytovanými platformou pro podporu této komponenty. Funkce kontejneru je definována platformou a je odlišná pro každý typ komponenty. Server však umožňuje, aby různé typy komponent spolupracovaly a poskytovaly funkce v podnikové aplikaci.“ [5]

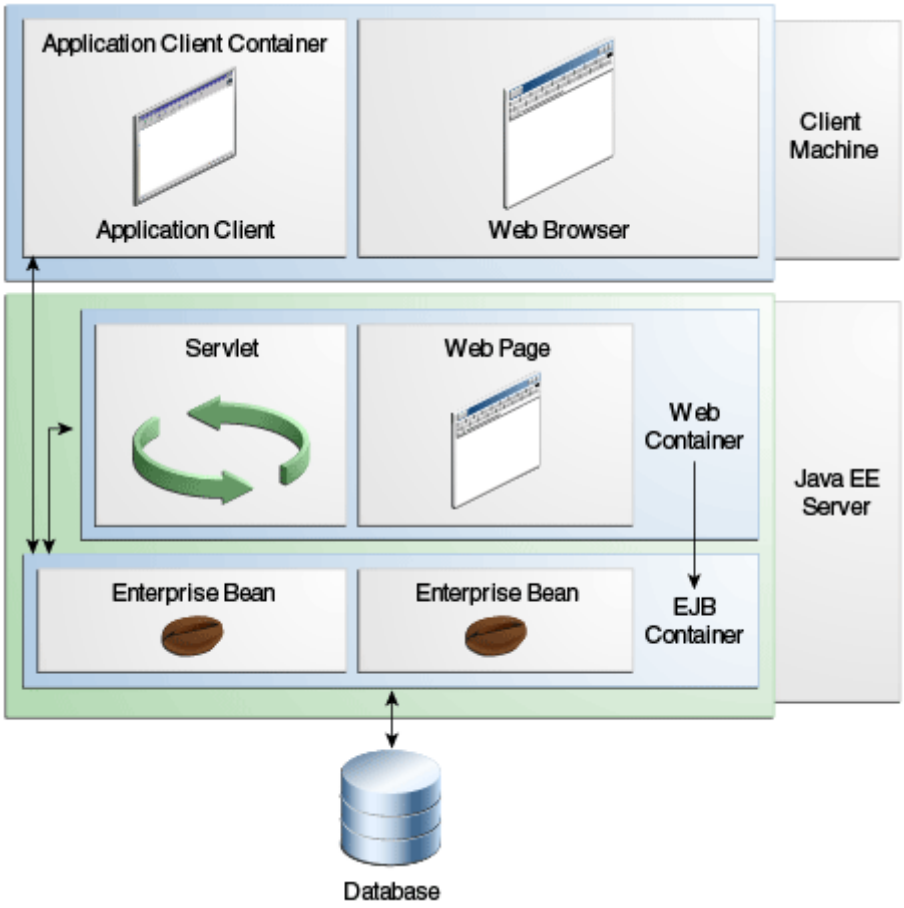
„Webový kontejner je rozhraní mezi webovými komponentami a webovým serverem. Webovými komponentami jsou Servlety, Facelety, JSP a další. Kontejner řídí životní cyklus

[1] Zdroj obrázku: https://javaee.github.io/tutorial/overview004.html

komponent, odesílá požadavky na ostatní součásti aplikace a poskytuje rozhraní kontextových dat, například informace o aktuální žádosti.“ [5]

„EJB kontejner je rozhraním mezi Enterprise Java Beans, které poskytují podnikovou logiku v aplikaci Java EE a server Java EE. Kontejner EJB běží na serveru Java EE a řídí provádění aplikací Enterprise Java Beans.“ [5]

Aplikační klient kontejner je rozhraním mezi jednotlivými komponentami serveru aplikace Java EE a klienty serveru, které jsou speciální aplikací Java SE. Běží na daném klientském zařízení. [5]



Obrázek 2 - Java EE Server a Kontejnery [2]

3.2 Framework Spring

„Tvorba jakékoliv netriviální aplikace obnáší nutnost zabývat se některými běžně se vyskytujícími aspekty návrhu, jakými jsou například autentizace, autorizace, správa transakcí, trvalé uložení dat v databázi apod. Dalším problémem běžných aplikací je slabá

[2] Zdroj obrázku: https://docs.oracle.com/cd/E19575-01/819-3669/6n5sg7as0/index.html

strukturovanost kódu. Tu lze řešit použitím některých návrhových vzorů, ovšem i takováto vlastní dekompozice kódu představuje zbytečné stále se opakující kódování.

Framework (někdy označován jako kontejner) tyto problémy řeší tak, že běžně se opakující funkce aplikací implementuje ve svých vlastních třídách a poskytuje tak hotovou infrastrukturu, kterou aplikace pouze využívají. Aplikační programátor se tak může soustředit jen na vlastní problém a zbytek nechat na frameworku, jenž je jeho aplikací využíván.

Typickou vlastností frameworků a la Spring je fakt, že životní cyklus aplikace má na starosti právě framework a kód aplikace je jím pouze v různých chvílích volán. Tím je zajištěn přesun kontroly nad během aplikace z aplikace do frameworku a vynucena vhodně dekomponovaná struktura dodržující rozšířené návrhové vzory.“ [6]

„Spring je primárně označován jako IoC kontejner, což znamená, že jedním z jeho účelů je aplikace konceptu Inversion of Control (přeložit to můžeme jako přesun kontroly). Návrhový vzor Inversion of Control, později překřtěný na Dependency Injection (tedy jakési vnucení závislostí) představuje možnost přesunout odpovědnost za vytvoření a provázání spolupracujících objektů z aplikace do frameworku.

Síť vytvářených objektů je definována např. v nějakém konfiguračním souboru, na jehož základě je frameworkem vytvořena a objekty jsou vzájemně provázány pomocí „set“ metod (Setter Injection), konstruktorů (Constructor Injection) či rozhraní (Interface Injection). Jádro Springu je na použití Dependency Injection vyloženě postaveno a podporuje první dva typy tohoto návrhového vzoru.“ [6]

3.2.1 **Spring Boot**

Spring Boot je další krok v evoluci frameworku Spring. Pomáhá vytvářet samostatné aplikace ve frameworku Spring s minimálním úsilím. Spring Boot odstranil všechny konfigurace založené na XML a poskytl anotace pro užívání Springu. Spring Boot obsahuje moduly, které se spustí při detekci komponenty ze Springu a danou komponentu rozumně nakonfigurují. Můžete rozběhnout vlastní aplikaci při využití Mavenu či Gradlu za několik minut. [7]

3.2.2 **JSP**

„JavaServer Pages (JSP) je technologie pro vývoj webových stránek, které podporují dynamický obsah. To pomáhá vývojářům vkládat kód Java do stránek HTML pomocí zvláštních značek JSP, z nichž většina začíná <% a končí s%>.

Komponenta JavaServer Pages je typ servletu Java, který je navržen tak, aby plnil roli uživatelského rozhraní webové aplikace Java. Weboví vývojáři zapisují JSP jako textové soubory, které kombinují HTML nebo XHTML kód, prvky XML a vložené akce a příkazy JSP.

Pomocí nástroje JSP můžete sbírat vstupy od uživatelů prostřednictvím webových formulářů, prezentovat záznamy z databáze nebo jiného zdroje a dynamicky vytvářet webové stránky.

Značky JSP mohou být použity pro různé účely, jako je získávání informací z databáze nebo registrace uživatelských preferencí, přístup ke komponentám JavaBeans, předávání informací mezi stránkami a sdílení informací mezi žádostmi, stránkami apod.“ [8]

3.2.3 **Hibernate**

Hibernate je framework pro objektově-relační mapování (ORM) pro Java aplikace. Byl vytvořen Gavinem Kingem v roce 2001. Hibernate je vloženo mezi Java objekty a databázový server a manipuluje s objekty na základě objektově-relačních mechanizmů a vzorů. [9]

Výhody využití Hibernate:

- Poskytuje jednoduché rozhraní API pro ukládání a načítání objektů Java přímo z databáze.
- Hibernate se stará o mapování tříd jazyka Java do databázových tabulek pomocí souborů XML a bez nutnosti napsání jediného řádku kódu.
- Nevyžaduje aplikační server pro svůj chod.
- Minimalizuje přístup k databázi pomocí strategií inteligentního načítání.
- Poskytuje jednoduché dotazování dat. [9]

Obrázek 3 - Java Hibernate [3]

3.3 MVC Architektura

„MVC je velmi oblíbený architektonický vzor, který se uchytil zejména na webu, ačkoli původně vznikl na desktopech. Je součástí populárních webových frameworků, jakými jsou např. Zend nebo Nette pro PHP, Ruby On Rail pro Ruby nebo MVC pro ASP .NET.“ [10]

„Základní myšlenkou MVC architektury je oddělení logiky od výstupu. Řeší tedy problém tzv. "špagetového kódu", kdy máme v jednom souboru (třídě) logické operace a zároveň renderování výstupu. Soubor tedy obsahuje databázové dotazy, logiku (např. PHP operace) a různě poházené HTML tagy. Vše je zamotané do sebe jako špagety. Kód se samozřejmě špatně udržuje, natož rozšiřuje.“ [10]

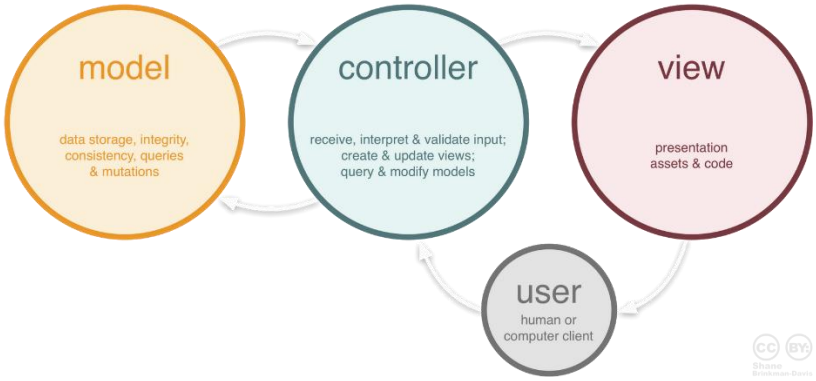
„Celá aplikace je rozdělena na komponenty 3 typů, hovoříme o Modelech, View (pohledech) a Controllerech (kontrolerech), od toho MVC. Neexistuje žádná striktní definice MVC architektury, a tak se můžete setkat s více výklady. Je dost možné, že to, co zde popisuji, je architektura MVP (nebudu rozebírat) nebo dokonce něco mezi. Co člověk, to názor, co framework, to jiný přístup. Důležitý je však základní princip, tedy rozdělení hlavních rolí těchto 3 komponent.“ [10]

„**Model** obsahuje logiku a vše, co do ní spadá. Mohou to být výpočty, databázové dotazy, validace a podobně. Model vůbec neví o výstupu. Jeho funkce spočívá v přijetí parametrů zvenku a vydání dat ven. Zdůrazním, že parametry nemyslím URL adresu ani žádné jiné parametry od uživatele. Model neví, odkud data v parametrech přišla a ani jak budou výstupní data zformátována a vypsána.“ [10]

[3] Zdroj obrázku https://www.tutorialspoint.com/hibernate/hibernate_overview.htm

„**Pohled (View)** se stará o zobrazení výstupu uživateli. Nejčastěji se jedná o phtml šablonu, obsahující HTML stránku a tagy nějakého značkovacího jazyka, který umožňuje do šablony vkládat proměnné, případně provádět iterace (cykly) a podmínky.“ [10]

„**Kontroler** je nyní onen chybějící prvek, který osvětlí funkčnost celého vzoru. Jedná se o jakéhosi prostředníka, se kterým komunikuje uživatel, model i pohled. Drží tedy celý systém pohromadě a komponenty propojuje.“ [10]



Obrázek 4 - Java MVC architektura [4]

3.4 Databáze

„Databázi si lze představit jako místo, kam ukládáme potřebná data. Velmi jednoduchým příkladem může být knihovna nebo kartotéka, kde jsou data uložena podle určitého systému. Pokud víme, jak tento systém použít, otevírají se nám možnosti, jak efektivně vyhledat data, jež zrovna požadujeme.

Databáze a její databázový systém slouží tedy k definici dat, která mají být ukládána. Řeší vztahy mezi těmito daty, způsob, jak je k nim přistupováno a jaké operace je s daty možné realizovat. V neposlední řadě pak funguje jako systém pro řízení přístupu k informacím – řeší oprávnění pro manipulaci s daty a správu uživatelů, jež mají k datům přístup.

Využití některého z databázových systémů jako úložiště dat a jeho propojení s programovanou aplikací je tak velmi nasnadě. Použití takového řešení je efektivnější, než

[4] Zdroj obrázku: https://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture

vytvářet vlastní způsob ukládání dat s dostatečným zabezpečením a snadným přístupem k požadovaným informacím.

Relační databáze se hojně využívají v aplikacích z důvodu své dobré pochopitelnosti a jednoduchosti. Mezi neznámější zástupce relačních databázových systémů patří MySQL, Oracle, Postgress, MS SQL a další. Pro webové aplikace menšího rozsahu se nejčastěji používá databáze MySQL ve spojení s PHP.

Základem relačních databází jsou databázové tabulky, které jsou na sebe určitým způsobem závislé – existuje mezi nimi jistá logická vazba (relace). Tabulky jsou též nazývány entitami – jsou chápány jako prvek reálného světa (např. zaměstnanec, školní předmět, vozidlo atd.).

Každá tabulka je tvořena sloupci a řádky, přičemž sloupce reprezentují vlastnosti této entity (pro příklad zaměstnance např. plat, datum nástupu, rodné číslo, …). Tyto vlastnosti se taktéž nazývají jako atributy. Každý sloupec musí mít jedinečný název a určený datový typ podle toho, jaká data jsou v něm uložena (číslo, text, logická hodnota atd.).

Řádky tabulky reprezentují samotné záznamy v databázové tabulce – jeden řádek reprezentuje např. jednoho zaměstnance s hodnotami daných atributů (výší platu, rodným číslem, …). Každý řádek by měl mít svůj jedinečný identifikátor, podle kterého bude možné určit příslušný záznam – klíč.“ [11]

3.5 JavaScript

Javascript je objektově orientovaný skriptovací jazyk. Běží ve webovém prohlížeči. Vznikl v roce 1995 ve firmě Netscape, kde řešili problém, jak do statických webových stránek vložit nějaký interaktivní obsah. [12]

„Netscape jde na větší interaktivnost jinak, vytváří jazyk LiveScript, který je v podstatě spojením jazyků Java (odtud syntaxe), Scheme (funkcionální paradigma, více dále) a Self (prototypové programování). Java + Scheme + Self = LiveScript. Nový jazyk byl navržen a naprogramován již za neuvěřitelné 3 týdny! Zde se zrodilo první z úskalí tohoto jazyka, problémem totiž je, že je šitý trochu horkou jehlou. Sun chtěl, aby se jazyk jmenoval JavaScript. Název je čistě obchodní tah a měl symbolizovat jakousi jednodušší verzi Javy. V

podstatě je to ale úplně nový jazyk, který přebírá jen část syntaxe Javy a její filozofii už vůbec ne. Zde je druhé úskalí, název je opravdu matoucí – JavaScript nebyl, není a nikdy nebude Java, má s ní jen pramálo společného.“ [12]

„Pomocí Javascriptu tedy můžeme měnit obsah webové stránky u uživatele, nabízí se tvorba dynamických menu, různých roletek a dalších kontejnerů, které umožňují ušetřit místo na stránce, když jsou zavřené a po najetí myší se otevřou. To je jistě šikovné a přehledné. JavaScript je skvělý k formátování textu, pomocí něj si můžete do napsané zprávy vkládat smajlíky nebo dokonce formátovat text jako ve Wordu. Když je na webové stránce nějaký editor, je to na 90 % JavaScript. Další využití nalezneme u ukazatelů času a data a dalších efektů na webových stránkách. Protože odeslání formuláře na server a čekání na následnou odpověď serveru je pomalé, používá se JavaScript také na validaci webových formulářů. Když například napíšete špatně email, webová stránka vás na to ještě před odesláním upozorní a není třeba stránku znovu načítat. Musíme si však uvědomit, že JavaScript běží na klientovi, může si ho uživatel vypnout nebo přepsat, proto nesmíme na takovouto validaci spoléhat a email musíme podruhé zkontrolovat i na serveru.“ [12]

3.5.1 **AJAX**

AJAX (Asynchronous JavaScript and XML) asynchronní Javascript a XML je soubor technologií, které umožňuje měnit obsah webových stránek bez nutnosti je znovu načítat. Ajax zahrnuje:

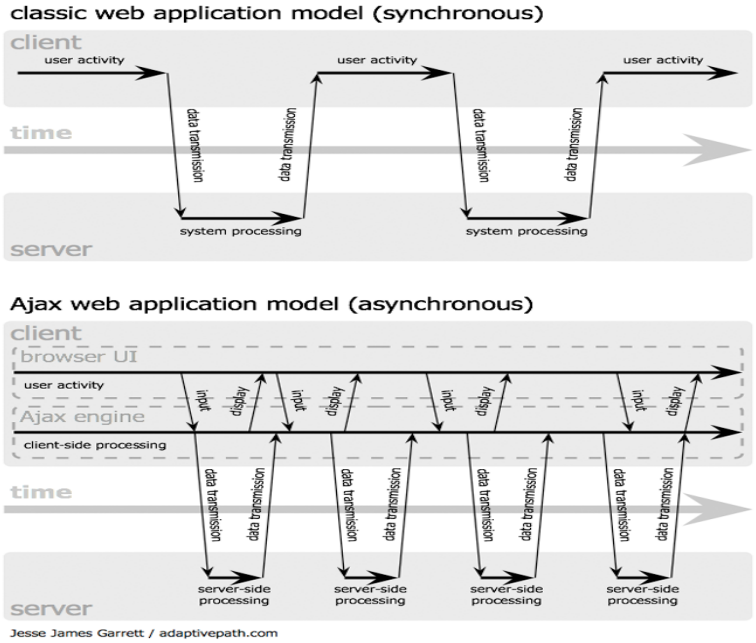
- Prezentace založená na standardech pomocí XMTML a CSS.
- Dynamické zobrazení a interakce použitím DOM (Document Object Model).
- Výměna dat a manipulace s nimi pomocí XML a XSLT.
- Asynchronní načítaní dat pomocí XMLHttpRequest.
- Vše spojené Javascriptem.

„Tradiční webové aplikace fungují tak, že uživatel spustí požadavek HTTP na webový server. Server provede zpracování (kontaktuje jiné systémy, získá data, upraví data) a poté vrátí jako HTML stránku.“ [13]

Ajax mění tuto start stop interakci přidáním mezivrstvy Ajax mezi uživatele a webserver. Na začátku prohlížeč načte Ajax a ten je zodpovědný za vykreslení rozhraní, které

uživatel vidí a komunikuje se serverem za uživatele. Ajax umožňuje interakci uživatele s aplikací, která se děje asynchronně – nezávisle na komunikaci se serverem. Takže uživatel nikdy nevidí na prázdnou stránku, když čeká na odpověď serveru. [13]

„Každá akce uživatele, která by normálně vygenerovala požadavek protokolu http, místo toho JavaScript zavolá Ajax a ten komunikuje asynchronně se serverem, kde spolupracuje s backendovým rozhraním. Tyto požadavky provádí obvykle pomocí XML (eXtensible Markup Language), aniž by zastavil interakci s uživatelem.“ [13]



Obrázek 5 - Normální a Ajax zpracování požadavků [5]

3.5.2 **JQuery**

JQuery je nejrozšířenější Javascriptová knihovna, která zrychluje a usnadňuje vytváření webových stránek a webových aplikací. Byla napsaná v roce 2006 Johnem Resigem s mottem „Pište méně, dělejte více“. jQuery umožňuje snadné psaní výkonných aplikací JavaScriptu a vytváření okouzlujících animovaných efektů. [14]

„Mezi jinými je jQuery skvělé pro:

- Přidání animovaných efektů do prvků. Funkce jQuery umožňuje snadné přidávání efektů, jako je vyblednutí, sklouzávání / vytahování a rozšiřování / zúžení.

[5] Zdroj obrázku: https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php

- Provádění XML (Ajax) požadavků. Ty používají JavaScript k vyžádání dodatečných dat z webového serveru, aniž by museli znovu načíst stránku.
- Manipulace s DOM. Můžete snadno přidávat, odstraňovat a upravovat obsah webové stránky pomocí několika řádků kódu.
- Vytváření obrázkových prezentací. Pomocí efektu jQuery můžete vytvářet hezké animované prezentace a lightboxy.
- Vytváření rozbalovacích nabídek. jQuery usnadňuje vytváření víceúrovňových dropdownů s animacemi.
- Vytváření rozhraní drag-and-drop. Pomocí jQuery můžete vytvořit stránku s prvky, které lze přesunout nebo přesunout jednoduše přetažením.
- Přidávání energie do formulářů. S nástrojem jQuery můžete snadno přidávat komplexní ověřování formulářů na straně klienta, vytvářet automatické textové pole Ajax, které stahují data z databáze na straně serveru a podobně.“ [15]

Další výhodou jQuery je kompatibilita mezi různými prohlížeči. S jQuery stačí zavolat příslušnou funkci a nechat jQuery vypořádat se s tím, aby kód běžel na různých prohlížečích stejně. [15]

3.5.3 **DataTables**

„DataTables je výkonný plugin jQuery pro vytváření seznamů tabulek a přidávání interakcí k nim. Poskytuje vyhledávání, třídění a stránkování bez jakékoliv konfigurace.“ [16] Chcete-li mít nástroje na třídění, vyhledávání a stránkování nad tabulkou, můžete si je naprogramovat nebo můžete použít DataTables které již toto vše řeší.

DataTables podporují jak zpracování na straně klienta, tak na straně serveru za pomoci Ajaxu.

3.6 **Integrované vývojové prostředí**

„Integrované vývojové prostředí (IDE) je aplikace, která usnadňuje vývoj aplikací. IDE jsou navrženy tak, aby zahrnovaly všechny programovací úlohy v jedné aplikaci. IDE proto nabízejí centrální rozhraní, které obsahuje všechny nástroje, které vývojář potřebuje, včetně následujících:

- **Editor kódu –** Tato funkce je textový editor určený pro psaní a úpravu zdrojového kódu. Editory zdrojových kódů se odlišují od textových editorů, protože zlepšují nebo zjednodušují psaní a úpravy kódu.
- **Kompilátor –** Tento nástroj transformuje zdrojový kód napsaný v jazyce, který je čitelný / zapisovatelný člověkem, do formátu spustitelného počítačem.
- **Debugger –** Tento nástroj se používá při testování, aby pomohl ladit aplikační programy.
- **Vytváření automatizačních nástrojů –** Tyto nástroje automatizují běžné úkoly vývojáře.

Navíc některé IDE mohou obsahovat také následující:

- **Prohlížeč třídy –** Tento nástroj se používá ke zkoumání a odkazování na vlastnosti objektově orientované třídy hierarchie.
- **Prohlížeč objektů –** Tato funkce se používá k prozkoumání objektů, které byly vytvořeny v běžícím aplikačním programu.
- **Hierarchický diagram třídy –** Tento nástroj umožňuje programátorovi vizualizovat strukturu objektově orientovaného programovacího kódu.

IDE může být samostatná aplikace nebo může být součástí jedné nebo více kompatibilních aplikací.“ [17]

„Celkovým cílem a hlavním přínosem integrovaného vývojového prostředí (IDE) je zlepšení produktivity vývojářů. IDE navyšují produktivitu tím, že zkracují dobu nastavování, zvyšují rychlost plnění vývojových úkolů, udržují vývojáře v obraze a standardizují vývojový proces.“ [17]

- **Rychlejší nastavení –** Bez IDE rozhraní by vývojáři museli trávit čas konfigurací několika vývojových nástrojů. Díky integraci aplikací IDE mají vývojáři na jednom místě stejnou sadu funkcí, aniž by bylo nutné neustále přepínat nástroje.
- **Rychlejší vývojové úkoly –** Přísnější integrace všech vývojových úkolů zvyšuje produktivitu vývojářů. Například kód lze analyzovat a syntakticky kontrolovat při psaní, což poskytuje okamžitou zpětnou vazbu a prevenci chyb. Navíc nástroje a funkce nástroje

IDE pomáhají vývojářům organizovat zdroje a zabraňovat chybám a snadněji je odhalovat.

- **Standardizace –** Rozhraní IDE standardizuje vývojový proces, který pomáhá vývojářům pracovat hladčeji a pomáhá novým zaměstnancům se zapracovat rychleji. [17]

3.6.1 Intellij Idea

„IntelliJ IDEA je inteligentní Java IDE, které poskytuje robustní kombinaci vývojových nástrojů. Klíčové funkce IntelliJ IDEA zahrnují: inteligentní pomoc při kódování, inteligentní navigaci a vyhledávání, četné refaktorování, analýzu kódu, podporu webového a podnikového vývoje, pokrytí unit testingu a usnadnění týmové práce.“ [18]

Hlavní přednosti IntelliJ IDEA:

Podpora pro vývoj webových a podnikových aplikací – Rychlý vývoj aplikací Java EE a webových aplikací s plnou podporou technologií a jazyků EJB, JSP, JSF, XML / XSL, HTML / XHTML / CSS, serverů JBoss, WebSphere, Glassfish, Geronimo, Tomcat a Weblogic, Rails, Struts a další rámce. Plus rozšířená podpora Javascript, AJAX a GWT. [18]

Inteligentní pomoc při kódování – Inteligentní editor, který rozpoznává jazyky Java, HTML / XHTML, XML / XSL, CSS, Ruby a JavaScript, podporuje frameworky jako Rails a GWT. Bez ohledu na jazyk, který je používán, jsou vždy po ruce pokročilé funkce dokončení kódu, ověřování, formátování a styling. Navíc zvýraznění chyb a syntaxe v kombinaci s inteligentními rychlými opravami umožňují efektivní vytváření bezchybného a srozumitelného kódu a zaměření na logiku projektu namísto běžné suché práce. [18]

Analýza kódu – Vestavěný vysokovýkonný analyzátor dynamických kódů s více než 600 kontrolami provádí analýzu kódu a detekuje všechny běžné syntaktické chyby a všechny další chyby, které by byly zjištěny během kompilace. Není potřeba kompilovat kód, aby bylo ověřeno, že je to bezchybné. IntelliJ IDEA poskytuje analýzu pro většinu podporovaných technologií. Statická analýza kódu detekuje překážky výkonu, nevyužitý kód, nesprávné závislosti a další problémy s kódem. IntelliJ IDEA poskytuje automatické řešení všech zjištěných chyb. [18]

3.7 Maven

„Maven je rozšířený systém pro správu a sestavování aplikací postavených nad platformou Java. Jeho využitím odpadá závislost na konkrétním IDE, protože všechny informace potřebné ke kompilaci a sestavení programu jsou přímo obsaženy ve speciálních souborech *pom.xml* (POM = project object model). Systém Maven je již plně integrován do všech velkých IDE (Eclipse, Netbeans, IDEA) a tak je práce s ním opravdu velmi snadná i přes uživatelské rozhraní.“ [19]

„Principem systému Maven je vytvoření objektového modelu nad zdrojovým kódem, se kterým lze provádět různé operace. Nejčastěji se jedná o kompilaci, kontrolu, vytvoření balíků atd. Model projektu je definován v souborech *pom.xml*, které se nachází v kořenovém adresáři každého projektu. V těchto adresářích lze spouštět příkazy *mvn* s parametry, které načtou model z odpovídajícího souboru *pom.xml* a provedou zadanou akci.

Klíčovou funkcí systému Maven je řešení závislostí. To znamená, že není nutné ručně kopírovat knihovny (JAR, WAR, EAR) a umisťovat je na classpath. Zvláště u velkých projektů složených z mnoha podprojektů je to neocenitelné zpřehlednění a zjednodušení vývoje.“ [19]

„Každý soubor *pom.xml* představuje jeden **projekt** (v hantýrce Maven **artefakt**). Artefakt je jednoznačně identifikován skupinou (groupId), názvem (artifactId) a verzí (version).“ [19]

„Repositář systému Maven obsahuje katalog artefaktů a systém pro jejich vyhledávání a stahování. Repositáře jsou **lokální** a **vzdálené**. Lokální repositář je vždy přítomen na vašem lokálním počítači a cachují se v něm použité závislosti. Není tedy nutné při každé kompilaci stahovat balíčky ze vzdáleného repositáře. Výchozím centrálním repositářem je http://search.maven.org/, ale konfigurací Mavenu či POM lze nastavit jiný (například firemní či týmový).

Závislostmi jsou myšlené jiné artefakty spravované systémem Maven, které nějaký artefakt vyžaduje ke své kompilaci či funkci. Závislosti mají tzv. *scope*, který specifikuje míru a okamžik potřeby dané závislosti. Scope může nabývat následujících hodnot:

- **compile** (výchozí) – artefakt je vyžadován pro kompilaci i běh aplikace (artefakt bude dostupný na všech classpath)
- **test** – artefakt je vyžadován pouze pro kompilaci a spuštění unit testů
- **runtime** – artefakt není vyžadován při komplikaci, ale je vyžadován za běhu aplikace (nejčastěji implementace různých API)
- **provided** – artefakt je vyžadován pro komplikaci i běh, ale bude poskytnut JVM za běhu (nejčastěji různé knihovny přibalené v aplikačním serveru)
- **system** – podobný jako *provided*, ale je nutné ručně uvést cestu k zadanému artefaktu“ [19]

3.8 Systémy správy verzí

K požadovanému cíli se dá dostat více než jednou cestou, ne všechny cesty jsou správné, i když jsou funkční. Novým vývojem můžeme rozbít již hotové a funkční celky. S velkou množinou těchto úskalí nám pomůže systém pro správu verzí.

Systémy správy verzí VCS (Version Control Systém) jsou softwarové nástroje, které sledují a ukládají veškeré změny prováděné na vybraných souborech. Software zaznamenává historii změn a umožňuje porovnat jednotlivé verze mezi sebou a vrátit zpět změny v souborech. Dokáže také sloučit dvě různé větve projektu. Tato schopnost velmi usnadňuje práci více lidí na jednom projektu. Systémy správy verzí mohou sloužit také jako systém projektového řízení a zálohovací systém. Systémy správy verzí se dělí na lokální, centralizované a distribuované. [20]

Hlavní výhody používání:

„Uchování historie verzí – Ještě včera váš program fungoval a dneska pořád padá? Honem ve stresu hledáte nějaké zálohy nebo nasazené kopie programu, abyste mohli zákazníkovi dodat alespoň poslední funkční verzi? S verzovacím systémem se vám tohle nestane – jednoduše se vrátíte k verzi, která ještě fungovala (nebo prošla testy) a postupně budete procházet změny a hledat, při které chyba vznikla. Ruční verzování (např. kopírováním do očíslovaných adresářů nebo tarů) je skutečně přežitek a verzovací systém oceníte i u těch nejmenších projektů.

Týmová spolupráce – verzovací systém je skvělý způsob, jak sdílet zdrojové kódy (resp. obecně soubory – třeba týmovou seminárku do školy) s ostatními kolegy. Odpadnou vám starosti se vzájemným přepisováním si souborů na sdíleném disku, nebo věčným otravným kopírováním souborů z poštovního klienta na disk a opětovným vkládáním do e-mailů.

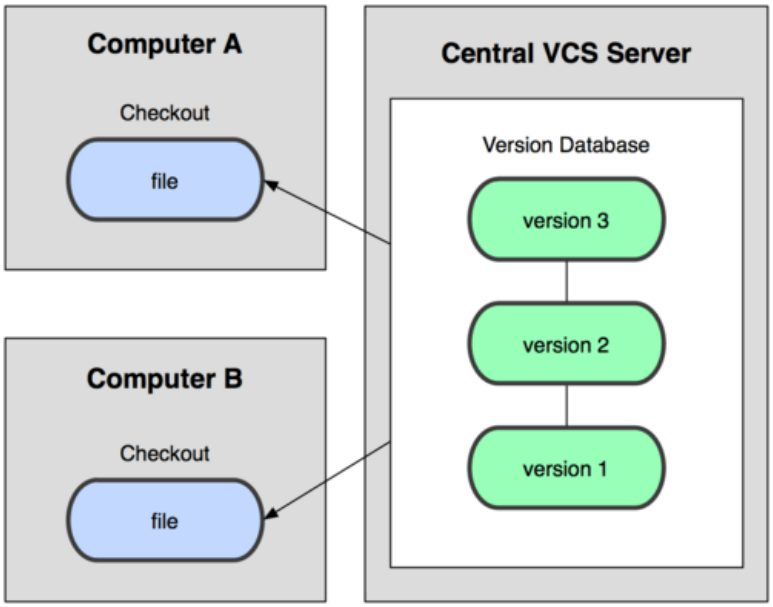
Práce ve více větvích – zejména při vývoji softwaru se vám bude hodit udržovat více větví téhož programu. Staré verze aplikace mnohdy nemůžete zavrhnout a nepodporovat – je třeba k nim vydávat různé opravy – takže zatímco ve větvi 2.0 budete pracovat na nové a úžasné (byť ještě ne zcela stabilní) verzi svého programu do původní, v produkci nasazené, větve 1.0 budete dopisovat drobné úpravy a opravy. Také můžete portovat nějakou funkcionalitu z nové větve do staré (sloučit některé změny)“ [21]

Dělení systému správy verzí:

Lokální systémy správy verzí slouží pro uživatele jednoho počítače. Systém ukládá změny do jednoduché databáze, dle stanovených kritérií ukládání. Nezálohují data mimo disk počítače. Mezi zástupce lokálních systémů správy verzí patří Revision Control System (RCS) [20]

Centralizované systémy správy verzí CVCS (Centralized Version Control System) Všechny soubory jsou uloženy v repozitáři (repozitory) na serveru. Jednotliví programátoři se přihlašují k systému přes klienta, stahují si potřebné soubory, vypracovávají zadané úkoly a odesílají zpět na centralizované místo. Výhodou tohoto uspořádání je přehlednost, kdy vedoucí projektu má souhrn, kdo na čem zrovna pracuje a jak dlouho mu to trvá. Nevýhodou je potřeba datového připojení. V případě nefunkčního připojení vznikají prostoje.

Hlavními zástupci jsou CVS, Subversion, Perforce. [20]

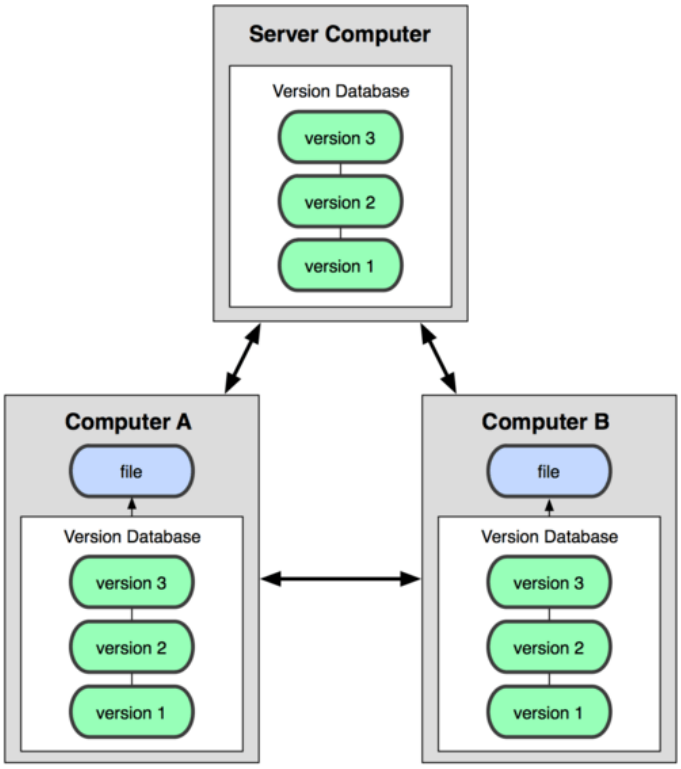


Obrázek 6 - Centralizované systémy správy verzí [6]

Distribuované systémy správy verzí BVCS (Distributed Version Control Systém) Mají uložena veškerá data na serveru, ale oproti centralizovaným systémům mají všichni uživatelé plnohodnotnou kompletní kopii projektu, čímž je zaručená jistá forma zálohy. Jako server může sloužit i jeden z uživatelů, ale tím ztrácíme některé výhody. [20]

„Distribuované systémy jsou velice užitečné při vývoji svobodného softwaru. Např. se nám líbí nějaký projekt a chceme v něm něco vylepšit. Jednoduše si naklonujeme úložiště a pracujeme nad touto kopií – nejsme ochuzeni o verzování, máme svoji vlastní historii změn. Odpadá tak administrativa a vyřizování práva zápisu do centrálního úložiště, nemusíme se ani nikoho ptát. Když uděláme kus práce, pošleme původním autorům odkaz na naše úložiště, a když se jim naše práce bude líbit, můžou ji přijmout a začlenit pomocí verzovacího systému do hlavní větve. Ani jedna strana nemusí mít k té druhé právo zápisu – každý si verzuje na svém vlastním písečku a v případě zájmu se změny sloučí.“ [21]

[6] Zdroj obrázku: https://git-scm.com/book/cs/v1/%C3%9Avod-Spr%C3%A1va-verz%C3%AD



Obrázek 7 - Distribuované systém správy verzí [7]

3.8.1 **Git**

Git vyvinula skupina vývojářů Linuxu v čele s Linusem Torvaldsem (tvůrce Linuxu) v roce 2005. Důvod vývoje byl v celku prozaický. Jednak dlouhodobé výhrady vůči používanému verzovacímu systému BitKeeper, který používali a jednak jeho následný přerod z volně použitelného na placený. [22]

„První důležitou věcí v Gitu je způsob ukládání souborů. Git každý soubor uloží pouze jednou a poté ukládá jen tzv. snapshoty. V každém commitu (pomyslný uzel systému) jsou uloženy všechny soubory jako snapshot. To dává Gitu výhody oproti ostatním verzovacím systémům (Perforce, CVS, Bazaar), které ukládají soubor vždy při jeho změně. Kromě toho Git ukládá všechny soubory binárně. Je tedy možné ukládat i netextové soubory (například obrázky) a Git poskytne stejné pohodlí. Velikost repositáře přitom zůstane prakticky stejná. Poté už závisí více na formátu souboru, zda se po malé změně změní celý binární soubor, nebo pouze část.

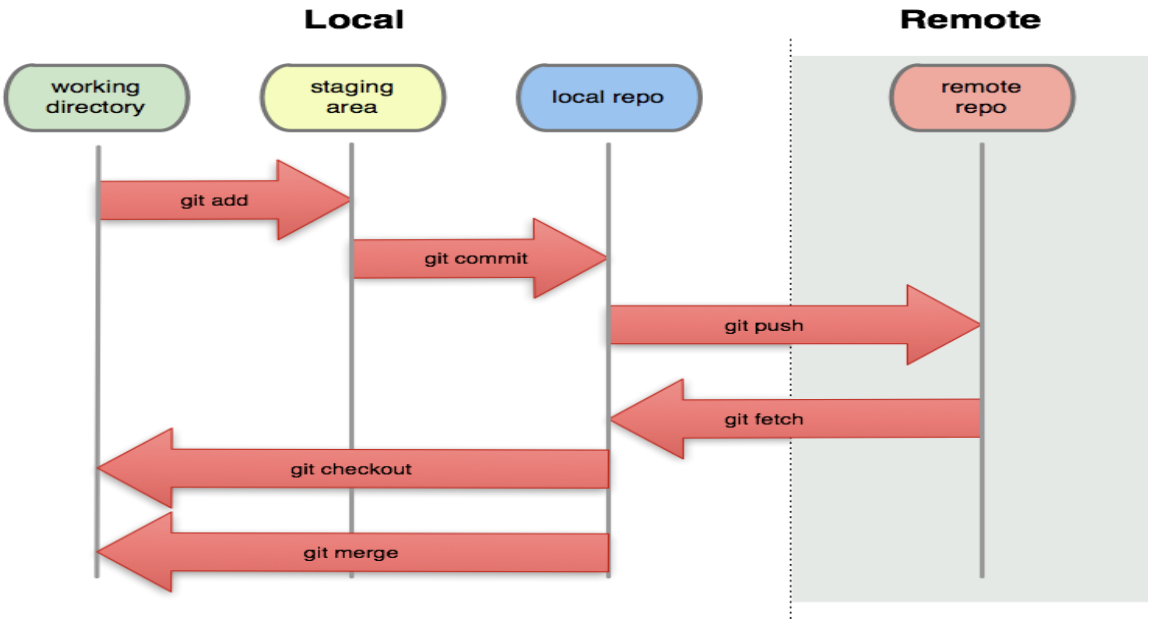
Další důležitou věcí je, že každá operace, kterou uděláte, je nejdříve lokální. Než řeknete, že je to nevýhoda, protože se mezi vás a server přidá další článek řetězce, pojďme se nad tím nejdříve zamyslet. V první řadě nám to dává možnost snadné opravy. Jakmile již něco

[7] Zdroj obrázku: https://git-scm.com/book/cs/v1/%C3%9Avod-Spr%C3%A1va-verz%C3%AD

publikujete, není možnost to změnit, protože někdo jiný již váš kód může mít stáhnutý a Git nenechává žádná zadní vrátka k ostatním uživatelům. Díky lokálnímu repositáři máte možnost po sobě kód dodatečně opravit, a teprve poté jej umístit na internet. Další výhoda, která z tohoto přístupu vyplývá, je, že není prakticky nic, co by vám Git nedovolil bez připojení k internetu. Můžete pracovat na svém lokálním repositáři a teprve po připojení můžete vše vzít a umístit na internet.

Do Gitu je také velmi silně zakomponována integrita. Pro každý soubor nebo složku Git spočítá kontrolní součet, který se nazývá SHA-1 hash (40-ti místné hexadecimální číslo). Nemůžete tedy změnit nebo poškodit soubor, aniž by o tom Git nevěděl. Git interně neukládá soubory podle jména, ale právě podle tohoto kontrolního součtu.

Git pouze přidává data. I když vymažete řádek, Git pouze dostane informaci o přidání „ničeho“ do souboru. Po provedení commitu už je velmi složité soubor poškodit do takového stavu, aby nešel obnovit. Nenastanou-li nějaké extrémní situace (poškození harddisku nebo kdyby se v repositáři přehraboval někdo neoprávněný) data jsou naprosto v bezpečí. Když navíc repositář nahrajete na nějaký server, ošetříte i tyto situace.“ [22]



Obrázek 8 - Práce v Git [8]

[8] Zdroj obrázku: https://www.quora.com/What-is-git-and-why-should-I-use-itv

4 Vlastní práce

V předcházející části byly představeny technologie a nástroje, které budou využity pro praktickou část. Zde bude popsán postup při tvorbě webové aplikace „Deník radiolokátoru“, která bude sloužit k evidenci a vyhodnocení provozních dat radiolokátoru P-37.

4.1 Analýza

Nejdříve je potřeba analyzovat současný stav. Co je to deník radiolokátoru? K čemu je využíván? Jaké technologie slouží k jeho funkčnosti, lze je nadále používat? Jaká data obsahuje a k čemu jsou využívána? Jaké jsou požadavky na bezpečnost? Jak lze rozšířit? Klady a zápory aktuální verze.

4.1.1 Co je deník radiolokátoru

Deník radiolokátoru slouží k zaznamenání a vyhodnocování veškerých provozních údajů. Mezi hlavní sledované údaje patří počet provozních hodin za den, který se dělí na technický provoz a bojový provoz. Bojový provoz je provoz, kdy radar předává data o zjištěných cílech do systému protivzdušné obrany. Technický provoz slouží k testování, ladění a nastavování radaru.

Dále se sleduje provedení údržby, počet hodin údržby, počet hodin v poruše, spotřebované náhradní díly a provoz a údržba elektrocentrál radaru. Jedním z hlavních hlídaných parametrů je provoz kritických dílů, u kterých se sleduje opracovaný počet hodin provozu. Z těchto dat lze následně vysledovat údaje důležité pro dlouhodobý provoz radiolokátoru a spolehlivost. Průměrná doba výdrže kritických součástek, kterou garantuje výrobce, ale i méně patrné údaje jako je častější poruchovost konkrétních částí radiolokátorů, za kterou může být zhoršení dílů od dodavatele či zhoršení kvality prací od externí servisní firmy. Velice často se využívají pro rozhodování o možnostech radiolokátoru. Z průměrné životnosti kritických dílu lze vyčíst, jak dlouho vydržíme se současným stavem náhradních dílu, či kolik dílů je nutné zabezpečit, pokud je potřeba zavést nepřetržitý provoz. Armáda plánuje nákupy dlouho dopředu a peníze pro nákup je nutné vyblokovat. Při cenách těchto dílů trvá jejich nákup i několik let. Data nejsou utajována, ale slouží jen pro vnitřní potřebu armády a neměla by být zveřejněna. Přístup k datům by měl mít jen ten, kdo je pro svoji práci potřebuje.

4.1.2 Stávající stav

V současné době se všechny tyto údaje zaznamenávají v papírové formě na stanovišti radiolokátoru. Následně jsou přepsány do excelového souboru a odesílány emailem na nadřízený stupeň.

Excelový soubor je pouhý digitální přepis, původního tištěného deníku. Liknavým přístupem řady uživatelů a vlivem mnoha okolností jako třeba výměna hardwaru, jsou současná data v elektronické podobě stará maximálně 3 roky a místy poněkud nevěrohodná.

V excelové verzi jsou implementovány vzorce, které upozorňují na očividnou chybnost dat (například provoz za den víc než 24 hodin). Vzorce ale nejsou implementovány úplně správně a někdy upozorní i na zápis, který je v pořádku. Pro tato falešná upozornění na chyby jsou vzorce často odstraňovány. Skutečnost, že někdo tyto vzorce odstranil, nelze na první pohled zjistit.

Velký problém je aktuálnost daného souboru. Když je po dvou opravách hlášení schváleno a přijato, následující měsíc přijde hlášení, které obsahuje původní chybná data z předchozího měsíce.

4.1.3 Nové požadavky

- oddělená data od programu
- přístup k informacím bude vyžadovat přihlášení
- rozlišit úrovně oprávnění přístupu
- data bude možno zálohovat
- zamezení úprav dat po schválení vyšší autoritou
- přístupnost ze všech stanovišť radiolokátorů a ze všech míst velení
- možnost filtrace a sumarizace filtrovaných dat

4.1.4 Výsledky analýzy

Na základě požadavků je zvoleno vytvoření webové aplikace, která všechna data bude ukládat v databázi.

Armáda provozuje intranetovou síť nepřipojenou k internetu. Na všech požadovaných stanovištích je k dispozici. Veškerý technický personál má na ni přístup.

Webové aplikace a databáze útvar již provozuje a nebude tak problém s jejich údržbou. Útvar zároveň provozuje zálohované uložiště, na kterém může běžet aplikace i databáze, čímž bude splněn požadavek na zálohování dat.

4.2 Návrh aplikace

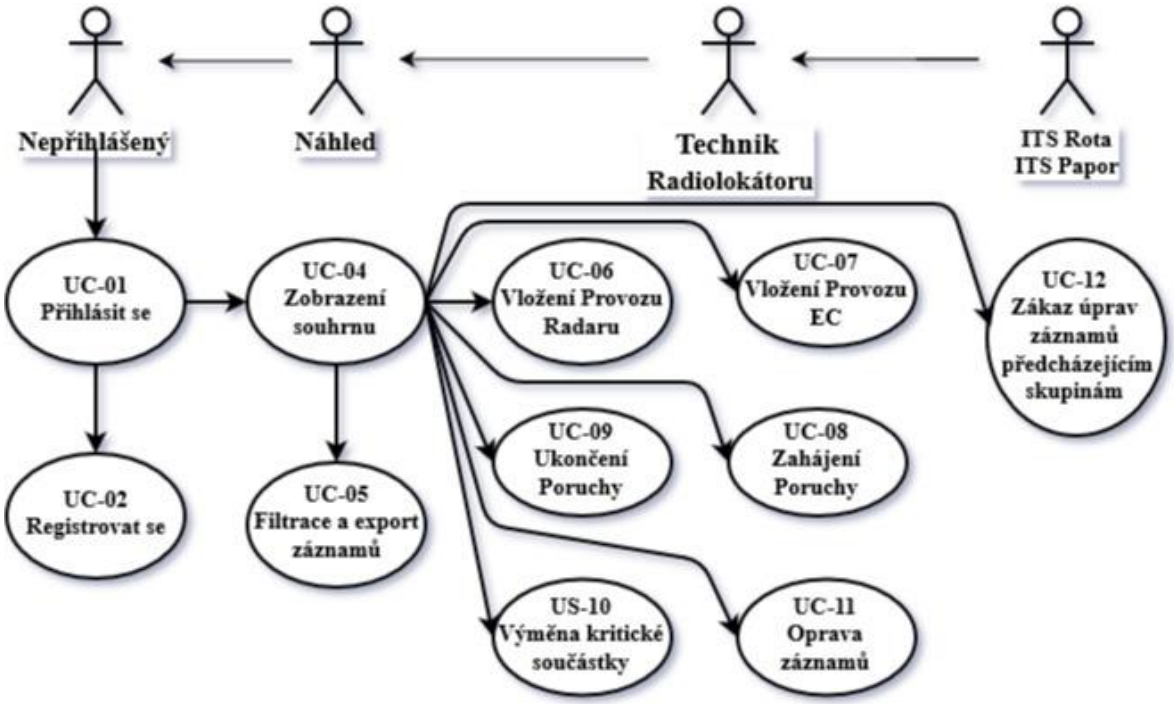
V následující části bude proveden návrh aplikace krok za krokem. Nejdříve bude navržen diagram případu užití, následně budou rozepsány samotné případy užití a bude navržena datová struktura.

4.2.1 Diagram případu užití

Use Case Diagram (česky diagram případů užití) je grafické zobrazení základní funkčnosti vyvíjeného systému. Účelem diagramu je popsat funkcionalitu systému, tedy co od něj uživatel očekává. „Diagram vypovídá o tom, co má systém umět, ale neříká, jak to bude dělat.“ [23]

Pro návrh funkčnosti aplikace bude využito jazyka UML. UML (Unified Modeling Language) je soubor grafických notací, který se používá pří vývoji softwaru.

V diagramu je vidět posloupnost uživatelských rolí. Po uživatelských rolích „Neregistrovaný“ a „Náhled“ následují role „Technik Radiolokátoru“, „ITS Rota“ (Inženýrsko Technická Služba) a „ITS Prapor“, které přesně kopírují organizační strukturu zadavatele. Neregistrovaný uživatel se může jen přihlásit, ale již nemůže zobrazit data. Uživatel s oprávněním „Náhled“ může nahlížet do dat filtrovat si je a exportovat. Uživatel s oprávněním „Technik Radiolokátoru“ je hlavním zdrojem dat pro aplikaci. Má všechna práva uživatele „Náhled“ a může všechna data zadávat a opravovat. Uživatelské role „ITS Rota“ a „ITS Prapor“ mají stejná oprávnění jako uživatelé v nižších uživatelských rolích. Jejich oprávnění je navíc rozšířeno o zákaz úpravy dat podřízeným uživatelským rolím. Role „ITS Prapor“ má možnost vkládat a zobrazovat položku cena opravy, jelikož jenom tito uživatelé vidí do rozpočtu oprav a jsou za něj odpovědní.



Obrázek 9 - Diagram případů užití [9]

4.2.2 Specifikace případů užití

V diagramu je vidět hierarchii uživatelských rolí. Jaké funkce bude daná role moci v aplikace spouštět, ale o samotných funkcích jsou známy jen jejich názvy. K podrobnějšímu popisu slouží Use Case Specifikace (Specifikace případů užití). Use Case Specifikace je dokument, ve kterém je specifikována funkcionalita daného systému a dělí se na jednotlivé Use Case (případy užití). Use Case nemá žádnou stanovenou podobu, ale měl by obsahovat několik základních bodů:

1. Krátký popis – k čemu má daná část aplikace sloužit
2. Aktéři – nejčastěji uživatel a systém
3. Podmínky spuštění – náležitosti, bez nichž nelze danou část spustit
4. Základní tok – scénář funkčnosti
5. Alternativní toky – scénáře pro chyby či odchylky
6. Podmínky dokončení – okolnosti vedoucí k úspěšnému ukončení [24]

Pro vývoj aplikace je třeba vypracovat všechny případy užití, ale v této práci budou představeny jenom některé.

[9] Zdroj obrázku: Vlastní zpracování

UC02: Registrovat se

Popis: Případ užití umožňuje se registrovat do aplikace

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel se musí nacházet na stránce pro registraci nového uživatele.

Základní tok – registrace nového uživatele.

1. **Systém** zobrazí registrační formulář s položkami login, heslo a potvrzení hesla.
2. **Uživatel** vyplní požadované údaje a odešle.
3. **Systém** provede validaci dat.
4. **Systém** registruje uživatele.
5. **Systém** zobrazí zprávu o úspěšné registraci.

Alternativní tok 1

3.1 Pokud **Uživatel** zadal chybné údaje Systém zobrazí registrační formulář dle kroku 1 základního toku s vyplněnými údaji a upozorní na chyby.

Obrázek 10 - Zobrazení formuláře registrace s upozorněním na chyby. [10]

UC06 Editace Provozu Radaru

Popis: Případ užití pro zápis denního provozu radiolokátoru

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel musí mít uživatelskou roli Technik Radiolokátoru nebo vyšší a musí se nacházet na stránce pro přehled provozu radiolokátoru.

Základní tok

1. **Systém** zobrazí přehled provozu radiolokátoru a dle výše oprávnění a role uživatele zobrazí tlačítko pro editaci záznamu provozu.
2. **Uživatel** stiskne tlačítko editace vybraného záznamu provozu.

[10] Zdroj obrázku: Vlastní vypracování

3. **Systém** ověří úroveň oprávnění editace záznamu a úroveň uživatele a zobrazí formulář s položkami Datum, Bojový provoz, Technický provoz, Údržba, Počet hodin údržby, Motohodiny a Počet hodin údržby Elektrocentrály.
4. **Uživatel** změní požadované údaje a odešle.
5. **Systém** provede validaci dat.
6. **Sytém** vyplní údaj o oprávnění dle role uživatele
7. **Systém** uloží údaje do databáze.
8. **Systém** zobrazí základní přehled radiolokátoru.

Alternativní tok 1

3.1 Pokud uživatel nemá dostatečné oprávnění, systém zobrazí zprávu o „nejste oprávněn měnit tento provoz“.

Alternativní tok 2

5.1 Pokud systém nevaliduje data, zobrazí formulář dle kroku 3 v základním toku se zadanými daty a upozorní na chyby.

UC-10 Vyjmutí kritické součástky

Popis: Případ užití pro zápis vyjmutí kritického dílu se sledovaným provozem

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel musí mít uživatelskou roli Technik Radiolokátoru nebo vyšší a musí se nacházet na stránce se souhrnem stavu radiolokátoru.

Základní tok

1. **Systém** zobrazí přehled osazených součástek a dle oprávnění uživatele tlačítko pro ukončení provozu.
2. **Uživatel** vybere, kterou kritickou součástku vyjmul a klikne na tlačítko end
3. **Systém** ověří oprávnění měnit tento záznam a zobrazí formulář pro výměnu součástky. Dle id předvyplní pole Id, Datum Osazení, Typ, Pozice a Výrobní číslo. Zobrazí pole Datum Vyjmutí k vyplnění.
4. **Uživatel** vyplní údaj.
5. **Systém** dle data výměny spočítá z dat provozu radiolokátoru, kolik hodin stará součástka odpracovala a uloží ji do databáze.
6. **Systém** zobrazí základní přehled radiolokátoru.

Alternativní tok 1

3.1 Pokud uživatel nemá dostatečné oprávnění systém zobrazí zprávu o „nejste oprávněn měnit tento díl“.

4.2.3 Datová struktura

Vytvoření správné struktury databáze je komplexní problém. Nesprávně navržená struktura může být příčinou budoucích potíží. Jedná se o problémy s rozšířením a vylepšením aplikace, kdy je žádoucí zachovat všechna stávající data v konzistentní podobě. Databáze by neměla být limitujícím faktorem při rozšiřování aplikace, kterému je nutno se přizpůsobit. Zároveň je nepraktická jakákoliv transformace dané databáze z důvodu příliš vysoké náročnosti. Proto při vytváření nové databáze je potřeba dodržovat několik základních pravidel:

- Nejdříve je potřeba se zamyslet nad všemi údaji, které je potřeba ukládat. Jednotlivé údaje by měly být nadále atomické (nedělitelné) a v databázi uloženy jen jednou.
- Nutno stanovit, jakých hodnot ukládané údaje mohou nabývat a přiřadit tomu odpovídající datové typy.
- Rozdělit vše do logicky ucelených skupin a přiřadit vazby.
- Určit primární klíč. Pokud žádný z údajů nemůže plnit funkci primárního klíče, pak doplnit vlastním.

V databázi budou vytvořeny tabulky, která odpovídají hlavním třídám aplikace: Provoz, Poruchy, Díly, User, Role.

Třída Provoz

Objekt třídy je záznam o denním provozu konkrétního radiolokátoru. Zaznamenává kolik hodin radar pracoval, jestli na něm byla provedena údržba a kolik hodin trvala.

Vlastnosti této třídy:

- Id – typ: Long
- Datum – typ: Date
- Bojovy – typ: Float
- Technicky – typ: Float
- Udrzba – typ: String
- UdrzbaHod – typ: Float
- MotohodinyEC – typ: Float

- UdržbaECHod– typ: Float
- Pristup – typ: Integer

Třída Porucha

Objekt třídy Porucha je záznam o poruše radiolokátoru. Zaznamenává, kdy začala, kdy skončila, celkovou dobu, kdy nebylo zařízení k dispozici, popis poruchy, kdo ji opravil a kolik oprava stála, pokud se prováděla externě.

Vlastnosti této třídy:

- Id – typ: Long
- Zahajeni – typ: Datetime
- Ukonceni – typ: Datetime
- HlavniDil – typ: String
- Prostoj – typ: Float
- Popis – typ: String, velikost 5000 znaku
- Opravil – typ: String
- Cena – Float
- Pristup – typ: Integer

Třída Díly

Objekt třídy je záznam o životním cyklu kritické součástky. Zaznamenává datum, kdy byla osazena, kdy byla vyjmuta, typ součástky, výrobní číslo, ve kterém kanálu radiolokátoru byla osazena a kolik hodin pracovala, než byla vyměněna.

Vlastnosti této třídy:

- Id – typ: Long
- Typ – typ: string
- Cislo – typ: String
- Osazeni– typ: Date
- Vyjmuti – typ: Date
- Pozice – typ: String
- OdpracovalBoj – typ: Float
- OdpracovalTech – typ: Float
- Pristup – typ: Integer

Třída User

Objekt třídy je záznam o uživateli aplikace a zaznamenává vlastnosti pro jeho identifikaci. Identifikace uživatele je pojata primitivněji, protože aplikace je provozována ve vnitřní síti, ve které je vyžadovaná důkladná autentizace.

- Id – typ: Long
- Username – typ: String
- Password – typ: String
- Roles – typ: Set<Role>

Třída Role

Objekt třídy Role je záznam o úrovni oprávnění pro manipulaci s daty v aplikaci

- Id – typ: Long
- Name – typ: String
- Users – typ: Set<User>
- Uroven – typ: Integer

4.3 Implementace

Po analýze požadavků a návrhu aplikace, přichází na řadu implementace. V této části budou popsány všechny kroky nutné pro rozběhnutí aplikace. Nebude popsána každá část aplikace, protože to není možné z důvodů rozsahu práce, ale budou názorně představeny všechny části důležité pro projekt.

4.3.1 Rozběhnutí projektu

Jak již bylo avizováno, bude tato aplikace vytvořena na platformě Java. K vývoji bude využito vývojového prostředí IntelliJ Idea, které již bylo popsáno v kapitole 2.5.1 a nástroj pro správu buildu Maven z kapitoly 2.6. IntelliJ Idea má open source verzi, ale pro vývoj aplikací JavaEE je výhodnější pořídit placenou verzi. Placená verze obsahuje moduly a nástroje, které usnadňují vývoj. Mimo jiné hledání a odstraňování chyb v projektu.

Příprava projektu zahrnuje stažení a nainstalovaní Java SDK, Intellij Idea, MySQL a Git. Java nepotřebuje žádné nastavení. MySQL stačí rozběhnout a nastavit přihlašovací údaje, kterými bude aplikace přistupovat do databáze.

V IntelliJi otevřít nový Maven projekt.

Nainstalovat Git a v IntelliJi nastavit, že ho má pro tento projekt používat jako VCS (version Control System). Soubory, u kterých je záhodno sledovat změny, musíme do Gitu připojit. Pokud ale se budou v projektu vytvářet další soubory, nebo se jen do projektu připojovat, zeptá se Git, jestli je požadováno připojení daných souborů do sledování.

Pro rozběhnutí projektu je dalším důležitým krokem nastavení Mavenu. Každý projekt má svůj konfigurační soubor „pom.xml“, který vygenerovala IntelliJi do kořenového adresáře při zahájení projektu. V tomto souboru se nacházejí obecné informace (název projektu, autor…), nastavení pluginů (verze jazyka Java atd.) a knihovny, na kterých je projekt vybudován. Pro tento projekt je klíčové nastavení, že projekt je založen na Spring Bootu, bez něhož by nic následně nefungovalo.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.6.RELEASE</version>
</parent>
```

Kód 1 - Parent Spring Boot

Knihovny potřebné pro projekt jsou anotovány závislostmi (dependency). Pokaždé, když při vývoji aplikace budeme přidávat další technologie, je nutné do souboru „pom.xml“ přidat další závislosti (dependency), které se nacházejí v dokumentaci výrobce dané technologie nebo na stránkách Mavenu.

Závislost, dle které Maven stáhne a nastaví v projektu webový server Tomcat:

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Kód 2 - Dependency MySQL

Pro připojení k databázi stačí v Mavenu přidat MySQL Conektor prostřednictvím další závislosti (dependency) do souboru „pom.xml“ a přidání souboru application.properties do projektu do složky src/main/resources. V souboru application.properties se nastaví způsob, jakým se bude komunikovat s databází. Nastaví se cesta k databázi a uvedou se přihlašovací jméno a heslo, které byli v databázi nastaveny pro komunikaci s aplikací.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/denik_radaru
spring.datasource.username=root
spring.datasource.password=Defiant1
```
Kód 3 - Nastavení přístupu do databáze

4.3.2 Vytvoření tříd

V dalším kroku se vytvoří třídy dle datového návrhu z kapitoly 3.9.3 každou jako samostatný soubor s příponou „.java“. Tyto třídy budou složit jako model dle MVC architektury z kapitoly 3.3. a jsou základní stavebním kamenem pro další práci. V třídě stačí vypsat proměnné a jejich typy. Gettery a settery se mohou vygenerovat za pomocí nástrojů vývojového prostředí.

Pro pojmenovávání všech prvků existuje v Javě konvence. Není sice povinná, ale pokud se vyvíjí aplikaci, kterou může někdo dále vyvíjet i po nás, měli bychom ji dodržovat.

- „Pro identifikátory (tříd, metod, proměnných, ...) používejte jména popisující jejich význam. Vyhýbejte se zkratkám. Jména by neměla být krátká (minimálně 3 znaky) ani příliš dlouhá (do 16 znaků).
- Jména tříd začínají velkým písmenem. Pokud se použije více slov, všechna začínají velkým písmenem. Pro označení třídy se obvykle používá podstatná jména v jednotném čísle, které bývá doplněno o přívlastky (např. třídy Student, SeznamStudentu).
- Jména metod, proměnných, formálních parametrů metod začínají malým písmenem. Pokud jméno obsahuje více slov, druhé a další začínají velkým písmenem.“ [25]

```
@IsCorrectProvoz
@Entity
public class Provoz {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Float bojovy;
    private Float technicky;
    private Date datum;
    private String udrzba;
    private Float udrzbaHod;
    private Float motohodiny;
    private Float udrzbaECHod;
    private int pristup;
}
```
Kód 4 - Třída provoz (bez getrů a setrů)

Anotace „@Entity“ (javax.persistence.Entity) deklaruje třídu jako entitu. Což umožňuje ukládat celou třídu jako jeden záznam databáze. Dále třída musí splňovat několik podmínek, aby se mohla státi entitou.

- „alespoň jeden veřejný či chráněný konstruktor bez parametrů

- třída ani její metody nesmí být „final“
- její atributy nejsou veřejné a ke každému existuje setter a getter¨ [26]

Anotace „@Id“ (javax.persistence) určuje, že následující proměnná bude v databázi fungovat jako primární klíč, a „@GeneratedValue(strategy = GenerationType.***AUTO***)“ (javax.persistence) ji zajistí automatické generování databází.

4.3.3 Vytvoření repozitářů

Pro každou třídu, která bude ukládat data do databáze, je nezbytné vytvořit rozhraní (interface). Toto rozhraní nám následně umožní práci s daty v databázi. Repozitář vytvoří potřebné tabulky v databázi podle daných tříd.

```
public interface ProvozRepository extends CrudRepository<Provoz, Long> {
}
```

Kód 5 - Repozitář Provoz

CRUD:

- Vytvoření (Create)
- Čtení (Read)
- Editace (Update)
- Smazání (Delete)

4.3.4 Zabezpečení aplikace

Se zabezpečením aplikace se musí potýkat všichni vývojáři aplikací. Proto byl vyvinut Framework Spring Security, který se pomalu stává standardem zabezpečení Spring aplikací. [27] Pro zabezpečení aplikace je zvolena implementace Spring Security. Implementace Spring Security není cílem této práce, tak se zaměříme na části, které byly upraveny nebo jsou významné pro projekt.

Jeden z hlavních požadavků na aplikaci je restrikce změny dat uživateli s nižším oprávněním. Třída Role je rozšířena o proměnou „uroven“ (Integer), které je určující při rozhodování, jestli uživatel může měnit konkrétní záznam v databázi. Ale to samotné je řešeno až v jiné části aplikace. Každé roli, vzestupně dle organizačního členění, je přiděleno číslo. Ve třídě uživatel se pak tato proměnná namapuje. Jelikož uživatel může mít více než jednu roli a tato role může být změněna, je nutné, aby se jeho proměnná nastavila na nejvyšší

z možných aktuálně přiřazených. Toho dosáhneme tím, že použijeme anotaci „@Transient“ (javax.persistence) a rozhraní proud (interface stream - java.util.stream). Z úrovní přiřazených rolí vybereme tu největší.

```
@Transient
public Integer getUroven(){
    if (roles != null){
        return roles.stream().filter(Objects::nonNull).mapToInt(role ->
role.getUroven()).max().orElse(0);
    }
    return 0;
}
```

Kód 6 - Výběr nejvyšší role

Další část, která je specifická pro projekt, je nastavení zabezpečení projektu. Nastavení se nachází v souboru „WebSecurityConfing.java“. Pro projekt je nastaveno:

- neověřený uživatel může navštívit jen registraci
- adresář „css“ je přístupný všem, aby nenastal problém se stylováním
- stránky s možností vkládat data jsou přístupné všem rolím kromě role „náhled“
- jakýkoliv požadavek (Request) na aplikaci musí být od ověřeného uživatele
- která stránka slouží k přihlášení

```
protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                    .antMatchers( "/registration","/css/**").permitAll()
                    .antMatchers("/provozAdd", "/provozDetail", "/poruchyAdd",
                          "/poruchyDetail", "/poruchyEnd", "/dilyAdd", "/dilyEnd",
                          "/dilyDetail").hasAnyAuthority("TECHNIK_ROTA", "ITS_ROTA",
                          "ITS_PRAPOR")
                    .anyRequest().authenticated()
                    .and()
                .formLogin()
                    .loginPage("/login")
                    .permitAll()
                    .and()
                .logout()
                    .permitAll();
    }
```

Kód 7 - Nastavení Spring Security

4.3.5 Zobrazení dat

Pro zobrazení(View) je zvolena technologie JavaServer Pages. Vstupní stránka do aplikace se jmenuje „index.jsp“. Na stránce se bude zobrazovat přehled základních sledovaných veličin, výpis posledních záznamů provozu, poruch a přehled osazených součástek se sledovaným provozem. Jakákoliv stránka, kterou má aplikace zobrazit, musí být namapovaná v kontroleru. Proměnné z databázových dotazů použitých v repozitářích, musí být přidány jako atribut daného mapování. A samotné repozitáře musí být připojeny ke kontroleru.

```
@Controller
public class MainController {

    @Autowired
    private ProvozRepository provozRepository;
    @Autowired
    private PoruchyRepository poruchyRepository;
    @Autowired
    private DilyRepository dilyRepository;

    @RequestMapping("/")
    public String index(Model model) {
       model.addAttribute("vypMag1k",
             dilyRepository.findAllByVyjmutiIsNullAndTypIs("Magnetron"));

       model.addAttribute("user", getCurrentUser());
       .
       .
       .
       return "index";
    }
}
```

Kód 8 - Namapování index.jsp – zkrácený

Načítání dat z databáze je možno provést několika způsoby. Pro načtení údaje o počtu poruchových hodin, které způsobil vysílač, je zvolena metoda dotazu použitím anotace „@Query“ (Spring Data JPA). V metodě je přesně specifikovaný databázový dotaz. Metoda vrací číslo, které je přiřazeno do proměnné.

```
@Query(value = "select SUM (prostoj) from Poruchy where hlavniDil ='Vysílač'")
public long getSumPoruchaVysilac();
```

Kód 9 - Dotaz počet poruchových hodin vysílače

Proměnou pak zobrazíme pomocí Expression Language, které je součástí JSP

```
${sumPoruchyVysilac}
```

Kód 10 - Zobrazení hodnoty počtu poruchových hodin vysílače

Pro načtení výpisu osazených dílů je využito rozhraní List<E> (java.util). Pro dotaz k naplnění listu dílu je použit pojmenovaný dotaz (named query - javax.persistence) . Samotný dotaz je včleněn do názvu dotazu. Samotný typ dílů, na který se dotazujeme, specifikujeme až v kontroleru. Tento jeden dotaz v repozitáři poslouží pro dotazy na všechny typ dílů.

```
public List<Dily> findAllByVyjmutiIsNullAndTypIs(String typ);
```

Kód 11 - List osazených dílu dle typu

Zobrazení listu dílů pomocí podmínky a cyklu knihovny JSTL Core, který je součást JavaServer Pages. Pro každou položku Listu vypíše námi stanovené proměnné.

```
<c:if test="${not empty vypMag1k}">
    <c:forEach var="dily" items="${vypMag1k}">
        <tr>
            <td>${dily.id}</td>
            <td>${dily.typ}</td>
            <td>${dily.pozice}</td>
            <td>${dily.cislo}</td>
            <td>${dily.osazeni}</td>
        </tr>
    </c:forEach>
```

```
</c:if>
```
Kód 12 - Zobrazení osazených dílů

4.3.6 Vkládání dat do databáze

Pro vkládání dat do databáze je nezbytné vytvořit stránku s formulářem pro vložení jednotlivých dat. Ukládání každého typy dat má svá specifika.

Databáze ukládá datum ve formátu 2018-01-02, který není uživatelsky příjemný. Proto je pro položku „Datum“ využito funkcí jQuery Datepicker. Při kliknutí do pole se spustí jQuery, které zobrazí kalendář s vyznačeným aktuálním datem, ze kterého jde kliknutím vybrat požadované datum, a jQuery vyplní pole ve správném formátu. Výstupní formát jQuery je nutno nastavit.

```
<td><form:label path="datum">Datum</form:label></td>
<td><form:input path="datum" id="datepicker"/>
    <form:errors path="datum"></form:errors>
</td>
<script>
    $(function () {
        $("#datepicker").datepicker({dateFormat: "yy-mm-dd"});
    });
</script>
```
Kód 13 - ProvozAdd.jsp datum

Typy údržby, které probíhají na radiolokátoru, jsou stanoveny výrobcem a upřesněny vnitřními předpisy armády. Pro toto pole je tedy nejvhodnější rozbalovací seznam s námi definovanými typy údržby. Seznam bude nadefinován v kontroleru.

```
@ModelAttribute("udrzbaList")
public Map<String, String> getUdrzbaList() {
    Map<String, String> udrzbaList = new HashMap<String, String>();
    udrzbaList.put("TO1", "TO1");
    udrzbaList.put("TO2", "TO2");
    udrzbaList.put("TO3", "TO3");
    udrzbaList.put("NO", "NO");
    udrzbaList.put("PTSP", "PTSP");
    return udrzbaList;
}
```
Kód 14 - List údržby kontroler.

A ve formuláři bude jen zobrazen.

```
<td><form:label path="udrzba">Údržba</form:label></td>
<td><form:select path="udrzba">
    <form:option value="" label=" "/>
    <form:options items="${udrzbaList}"/>
</form:select></td>
```
Kód 15 - ProvozAdd.jsp údržba list

Aby aplikace po zavolání „/provozAdd“ zobrazila provozAdd.jsp, musí být v kontroleru daný soubor namapován pomocí anotace „@RequestMapping“ (org.springframework.web.bind.annotation) Vyžijeme třídu ModelAndView (java.lang.Object). V ModelAndView bude nastaven, které „View“ a podle kterého „Modelu“ dle MVC architektury má zobrazit a že je vyžádán nový prázdný model.

```
@RequestMapping("/addProvoz")
public ModelAndView showProvozForm() {
    ModelAndView model = new ModelAndView("provozAdd", "provoz", new Provoz());
    return model;
}
```
Kód 16 - Namapování provozAdd.jsp

Po vyplnění hodnot ve formuláři a stisknutí tlačítka „Submit“ aplikace provede úkony dle hlavičky formuláře. Vyplněné údaje pošle metodou „POST“ na „/submitProvoz".

```
@RequestMapping("/submitProvoz")
public String submitProvoz(@Valid @ModelAttribute("provoz") Provoz provoz, BindingResult
result, ModelMap model) {
    if (result.hasErrors()) {
        return "/provozAdd";
    }
    User user = getCurrentUser();
    provoz.setPristup(user.getUroven());
    provozRepository.save(provoz);

    return "redirect:/";
}
```
Kód 17 - Vkládání provozu do databáze

Po odeslání je nutno zkontrolovat, jestli data odpovídají stanoveným parametrům. Pokud jsou data validována, aplikace nastaví přístup k datům dle role přihlášeného uživatele a data uloží.

4.3.7 **Validace dat**

K validaci použijeme anotaci „@Valid“ (javax.validation). Pro samotnou validaci vytvoříme rozhraní „IsCorectProvoz.java“ a samostatnou třídu „ProvozValidator.java“, ve které je samotná logika pro ověření vstupních dat. Rozhrání pro validaci je anotováno nad třídou „Provoz“. Anotaci „@IsCorectProvoz“ (viz Kód 4).

Spring nedopustí manipulaci s daty, které nekorespondují s datovými typy definovaných proměnných, a tak se zaměřujeme na data, která by zpracováním prošla a uložila by se do databáze. Provoz je sledován za jeden den v hodinách, takže žádná z hodnot nemůže nabývat větších hodnot než 24. Pole bojový a technický provoz sledují provoz radiolokátoru společně a jejich součet obou polí také nemůže nabývat vyšší hodnoty než 24 hodin. Dále je nutné zabránit vkládání záporných čísel.

```
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.FIELD,ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {ProvozValidator.class})
public @interface IsCorrectProvoz {
    String message() default "Provoz za den 0 - 24 hodin";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```
Kód 18 - Rozhraní IsCorectProvoz

```
public class ProvozValidator implements ConstraintValidator<IsCorrectProvoz, Provoz> {
    @Override
    public void initialize(IsCorrectProvoz constraintAnnotation) {}
    @Override
    public boolean isValid(Provoz provoz, ConstraintValidatorContext context) {
        if (provoz.getTechnicky() != null && provoz.getTechnicky() > 24.0) {
            return false;
        } else if (provoz.getTechnicky() != null && provoz.getTechnicky() < 0.0) {
            return false;
        } else if (provoz.getBojovy() != null && provoz.getBojovy() > 24.0) {
            return false;
        } else if (provoz.getBojovy() != null && provoz.getBojovy() < 0.0) {
            return false;
        } else if (provoz.getUdrzbaHod() != null && provoz.getUdrzbaHod() < 0.0) {
            return false;
        } else if (provoz.getUdrzbaHod() != null && provoz.getUdrzbaHod() > 24.0) {
            return false;
        } else if (provoz.getMotohodiny() != null && provoz.getMotohodiny() > 24.0) {
            return false;
        } else if (provoz.getMotohodiny() != null && provoz.getMotohodiny() < 0.0) {
            return false;
        } else if (provoz.getUdrzbaECHod() != null && provoz.getUdrzbaECHod() < 0.0) {
            return false;
        } else if (provoz.getUdrzbaECHod() != null && provoz.getUdrzbaECHod() > 24.0) {
            return false;
        } else if (provoz.getBojovy() != null && provoz.getTechnicky() != null &&
                   provoz.getBojovy() + provoz.getTechnicky() > 24.0) {
            return false;
        }
        return true;
    }
}
```

Kód 19 - ProvozValidator.java

Pokud se některá z podmínek vyhodnotí jako nepravda, kontroler následně vrátí zpátky „/provozAdd“, s uživatelem vyplněnými hodnotami. Do formuláře je vložena značka <Form:errors>,která se zobrazí, jen pokud validátor pošle námi definovanou zprávu.

4.3.8 Editace dat

Editovat záznamy provozu mohou jen uživatelé s rolí „technik-rota“ a vyšší, a to jen pokud příslušný záznam nezměnil již někdo s vyšším oprávněním. Záznamy provozu lze změnit tlačítkem „edit“ u vybraného zápisu. Tlačítka se zobrazují, jen pokud máte příslušná oprávnění měnit záznam. Při zobrazování výpisu provozu je přidána podmínka „if“, která pro každý zobrazovaný záznam provede porovnání proměnných úrovně uživatele a přístupu provozu.

```
<c:forEach var="provoz" items="${provozR}">
    <tr>
        <td>${provoz.id}</td>
        <td>${provoz.datum}</td>
        <td>${provoz.bojovy}</td>
        <td>${provoz.technicky}</td>
        <td>${provoz.udrzba}</td>
        <td>${provoz.udrzbaHod}</td>
        <td>${provoz.motohodiny}</td>
        <td>${provoz.udrzbaECHod}</td>
        <c:if test="${provoz.pristup <= user.uroven}">
            <td><a href="/provoz/edit/${provoz.id}">edit</a>
            </td>
        </c:if>
```

```
    </tr>
</c:forEach>
```

Kód 20 - Zobrazení tlačítka editace provozu

Pokud je podmínka splněna zobrazí se odkaz „edit“. Po kliknutí na odkaz zavolá aplikace /provoz/edit/ s číslem id provozu, které chce uživatel měnit. Kontroler načte příslušná data z databáze a ověří přístup k datům pro uživatele. Pokud má uživatel přístup k datům zobrazí aplikace, soubor „poruchyDetail.jsp“ s formulářem pro editaci provozu, ve kterém vyplní data dle záznamu v databázi.

```
@RequestMapping("/poruchy/edit/{id}")
public ModelAndView showPoruchyForm(@PathVariable("id") long id) {
    Poruchy poruchy = poruchyRepository.findOne(id);
    User user = getCurrentUser();
    if (poruchy.getPristup() > user.getUroven()) {
        throw new AccessDeniedException("Nejsi opravnen měnit tuto poruchu");
    }
    return new ModelAndView("poruchyDetail", "poruchy", poruchy);
}
```

Kód 21 - Poruchy edit kontroler

Položka, která nemá býti měněna jako „Id“, je ve formuláři jen zobrazena bez možnosti editace.

```
<tr>
    <td><form:label path="id">id</form:label></td>
    <td>${provoz.id}<form:hidden path="id"/></td>
</tr>
```

Kód 22 - Zobrazení Id bez možnosti editace

Validace dat a uložení probíhá obdobně jako ukládání nového záznamu provozu viz kapitola 4.3.6.

4.3.9 Ukončení poruchy

Při ukončování poruchy je potřeba sečíst celkový čas prostoje a uložit do databáze. Toho je docíleno tím, že když při volání „/updatePoruchy“ je vyplněn údaj o osazení i vyjmutí aplikace odečte od data a času vyjmutí datum a čas osazení. Výsledné číslo je ještě nutno vydělit číslem 3 600 000, protože výsledný rozdíl dvou hodnot Timestamp vyjde v milisekundách a je potřeba ho uložit v hodinách.

```
@RequestMapping("/updatePoruchy")
public String updatePoruchy(@Valid @ModelAttribute("poruchy") Poruchy poruchy,
BindingResult result, ModelMap model) {
    if (result.hasErrors()) {
        return "error";
    }
    if (poruchy.getZahajeni() != null && poruchy.getUkonceni() != null) {
        poruchy.setProstoj((float) (((float) (poruchy.getUkonceni().getTime() -
poruchy.getZahajeni().getTime())) / 3600000.0));
    }
    User user = getCurrentUser();
    poruchy.setPristup(user.getUroven());
    poruchyRepository.save(poruchy);
    return "redirect:/";
}
```

Kód 23 - Ukončení poruchy

4.3.10 Vyjmutí kritického dílu

Při vyjmutí kritického dílu je potřeba spočítat kolik hodin bojového a technického provozu odpracoval během svého osazení v radiolokátoru. Z repozitáře provozu se načtou všechny údaje za pomoci pojmenovaného dotazu (named query) a využitím rozhraní proud (Interface Stream - java.util.stream), které podporuje postupné a paralelní agregované operace. Výsledek se uloží do databáze.

```
@RequestMapping("/updateDily")
public String updateDily(@Valid @ModelAttribute("dily") Dily dily, BindingResult result,
ModelMap model) {
    if (result.hasErrors()) {
        return "error";
    }
    if (dily.getVyjmuti() != null) {
        List<Provoz> provozList = provozRepository.findByDatumBetween(dily.getOsazeni(),
           dily.getVyjmuti());
        dily.setOdpracovalBoj((float) provozList.stream()
                .filter(Objects::nonNull)
                .mapToDouble(provoz -> {
                    Float value = provoz.getBojovy();
                    if (value == null)
                        return 0.0;
                    return (double) value;
                })
                .sum()
        );
        dily.setOdpracovalTech((float) provozList.stream()
                .filter(Objects::nonNull)
                .mapToDouble(provoz -> {
                    Float value = provoz.getTechnicky();
                    if (value == null)
                        return 0.0;
                    return (double) value;
                })
                .sum()
        );
    }
```

Kód 24 - Výpočet odpracované doby dílu

4.3.11 Filtrování dat

Na radaru se ročně vede 365 záznamů provozu, do 20 poruch ročně a sledují se tři kritické součástky osazené v každém z pěti kanálů radiolokátoru s různou průměrnou životností, ale v řádech jednotek tisíců hodin. Při minimální plánované životnosti radiolokátoru 20 let bude jen za provoz v databázi uloženo 7 300 záznamů. Základní

sledované parametry se zobrazují v přehledu stavu radiolokátoru na úvodní stránce. Pro detailnější souhrny a procházení starších záznamů je vyžito technologie DataTables, která má v sobě spoustu funkcí pro práci s daty a není nutné je programovat.

Pro přidání zobrazení dat pomocí DataTables je nutno přidat před každou proměnou, která se má zobrazit, anotaci „@JsonView(DataTablesOutput.View.class)“ (com.fasterxml.jackson.annotation) a vytvořit pro každý model další repozitář.

```
public interface ProvozRestRepository extends DataTablesRepository<Provoz, Long> {
}
```

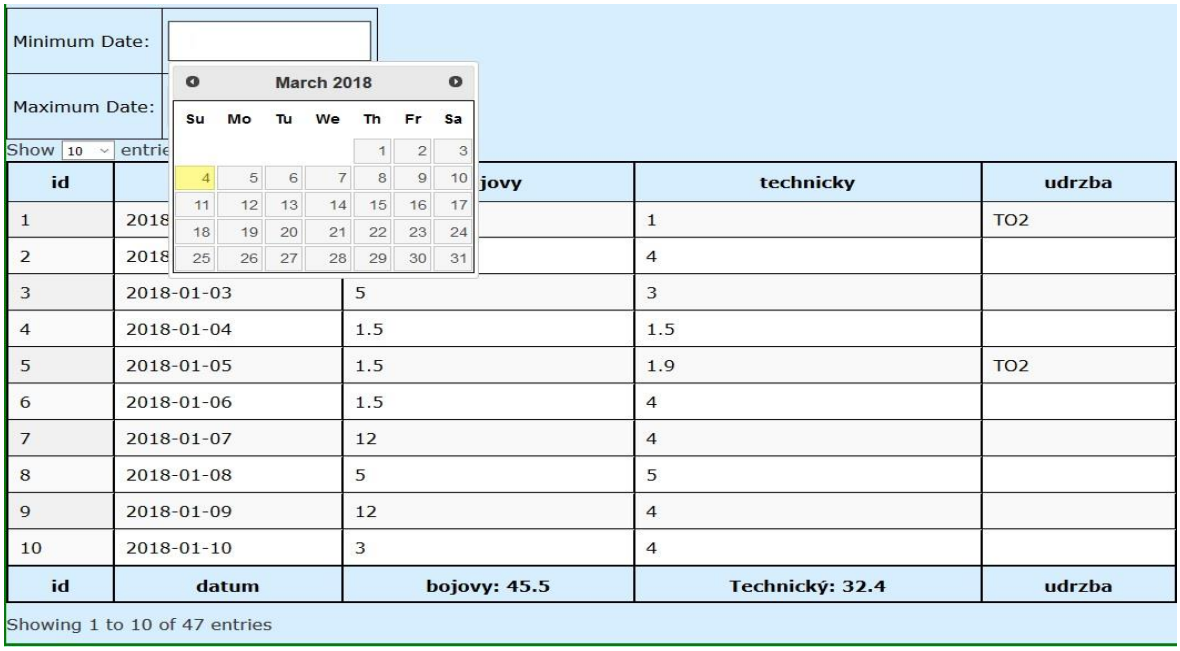
Kód 25 - ProvozRestRepository

Pro zobrazení dat pomocí DataTables je připravena stránka s tabulkou, která má jen záhlaví a zápatí. Obsah tabulky vyplňuje JavaScript. Načte data z databáze a zobrazí je do tabulky dle proměnných. Pro možnost filtrace záznamů dle dat jsou před tabulku přidány dvě pole pro vkládaní data „min“ a „max“ a script rozšířen o funkci, která si daná data načte pro potřeby kontroleru. Pro usnadnění vkládání dat do polí „min“ a „max“ je přidán DatePicker.

```
$(document).ready(function () {
    var table = $('#provoz').DataTable({
        "ajax": {
            "url": "data/provoz",
            "data": function (a) {
                return $.extend({}, a, {
                    "date_from": $('#min').val(),
                    "date_to": $('#max').val()
                }); }  },
        'serverSide': true,
        columns: [{
            data: 'id'
        }, {
            data: 'datum'
     …

$('#min, #max').datepicker({dateFormat:"yy-mm-dd"}).change(function () {
    table.draw();
});
```

Kód 26 - JavaScript pro zobrazení dat provozu zkrácený



Obrázek 11 - ProvozFilter.jsp [11]

Pro součet zobrazených údajů ve sloupci a zobrazení výsledku v zápatí tabulky se pro každý sloupec, který chceme sčítat přidá script.

```
pageTotal = api
    .column(3, {page: 'current'})
    .data()
    .reduce(function (a, b) {
        return intVal(a) + intVal(b);
    }, 0);
$(api.column(3).footer()).html(
    'Technický: ' + pageTotal
);
```

Kód 27 - JavaScript pro součet zobrazených hodnot

Kontroler pro zobrazení stránky „provozFilter.jsp“.

```
@RequestMapping("/provoz")
public ModelAndView showProvoz() {
    return new ModelAndView("provozFilter", "provoz", provozRepository.findAll());
}
```

Kód 28 - ProvozFilter.jsp kontroler

Pro samotný běh DataTables je vytvořen další kontroler „@RestControler“ (org.springframework.web.bind.annotation), který mapuje zdroj dat pro zobrazení. Pokud nejsou vyplněny „fromDate“ a „toDate“ tak naplní stránku všemi údaji z databáze, pokud vyplněny jsou pošle je jako dotaz do databáze a výsledek pošle pro zobrazení na stránku.

```
@RestController
public class ProvozRestController {
    @Autowired
    private ProvozRestRepository provozRestRepository;
 public static Specification<Provoz> getProvozSpecification(Date fromDate, Date toDate) {
    return new Specification<Provoz>() {
```

[11] Zdroj obrázku: Vlastní vypracování

```
        @Override
        public Predicate toPredicate(Root<Provoz> root, CriteriaQuery<?> query,
                CriteriaBuilder cb) {
            return cb.between(root.get("datum"), fromDate, toDate);
        }    };  }
    @JsonView(DataTablesOutput.View.class)
    @RequestMapping(value = "/data/provoz", method = RequestMethod.GET)
    public DataTablesOutput<Provoz> getProvoz(@RequestParam("date_from") String dateFrom,
             @RequestParam("date_to") String dateTo, DataTablesInput input) {
        Date fromDate = null;
        Date toDate = null;
        try {
            fromDate = Date.valueOf(dateFrom);
            toDate = Date.valueOf(dateTo);
        } catch (Exception e){}
        if (fromDate != null && toDate != null) {
            return provozRestRepository.findAll(input, getProvozSpecification(fromDate,
                    toDate));
        } else {
            return provozRestRepository.findAll(input);
        }     }     }
```

Kód 29 - DataTables kontroler

4.4 Testování

Testování přichází na řadu ihned po implementaci. Vyvinutý program patří do kategorie malých projektů, pro který není třeba vymýšlet složitější testovací scénáře či vyžívat testovací software. Dílčí testování probíhalo již během vývoje jednotlivých částí.

Testováno je zobrazení, ukládání dat, načítání dat.

Zobrazení je testováno se zaměřením, zdali se aplikace zobrazuje ve všech běžných prohlížečích stejně a jestli se vše zobrazuje správně dle přidělených rolí.

Ukládání dat do databáze je testováno zadávání různých vhodných a nevhodných hodnot. Mezi sledované funkcionality patřilo: ukládání nevhodných dat do databáze a případný pád aplikace, ukládání záznamů do správných tabulek a zároveň, jestli databáze neukládá záznamy, které tam nepatří. K přímému nahlížení do databáze a úprav jejich záznamů pro testování je využit nástroj MySQL Workbench.

Načítaná dat se ověřovala v porovnáním s přímím pohledem do databáze. Funkčnost jednotlivých vzorců se ověřila ručním spočítáním.

Finální verze aplikace je komplexně otestována. Testovala se celková funkčnost a zabezpečení.

Celková funkčnost je otestována postupně. Všechny prvky jsou otestovány dle jednotlivých Use Case po ukončení vývoje.

Testování bezpečnosti – jestli lze bez registrace zobrazit uložené záznamy (zobrazit se má pouze login a registrace). Otestována je možnost zobrazení stránek uživatelem či volání dat k editaci bez potřebného oprávnění včetně možnosti přímého zadání do adresního řádku.

Během testování není zjištěna žádná zjevná závada

5 Výsledky a diskuse

5.1 Zhodnocení vývoje

Aplikace funguje dle zadaných požadavků. Během vývoje bylo využito nabytých znalostí technologií a nástrojů uvedených v teoretické části. Velice bylo oceněno nástroje Maven pro zjednodušení sestavení a vývojové prostředí, jehož pluginy byly neocenitelnými pomocníky, při řešení všech zádrhelů během vývoje.

5.2 Možná vylepšení

Během plnění aplikace daty pro testování a samotném testování bylo navrhnuto několik možných vylepšení, která by stála za implementování.

- Mít možnost editovat seznam prováděné údržby, pokud by se časem ukázalo, že stárnoucí zařízení jich potřebuje víc.
- Možnost editace a schválení záznamů i DataTables
- Možnost hromadného schválení záznamů
- Zvážit ochranu před SQL injection

6 Závěr

Hlavním cílem práce byla analýza a implementace webové aplikace „Provozní deník radiolokátoru“ a dílčím cílem byl popis vývoje webových aplikací na platformě Java. Tyto cíle se v práci podařilo splnit.

V práci byla napřed představena teoretická východiska pro seznámení čtenáře s použitými technologiemi. Platforma Java se zaměřením na Java EE a s ní spojené technologie jako je Framework Spring Boot pro usnadnění nastavování projektu, JavaServer Pages pro jeho možnosti zobrazení a Hibernate jako nástroj objektově relačního mapování. Následně byla představena MVC architektura a MySQL databáze. Dále byly předestřeny možnosti JavaScriptu v podání AJAXu, JQuery a DataTables. Na konci teoretických východisek byly představeny nástroje podporující a usnadňující vývoj. Vývojové prostředí IntelliJ Idea, nástroj pro správu sestavení Maven a systém pro správu verzí Git.

Vlastní práce se zabývá analýzou stávající situace při evidenci provozních dat radiolokátoru a požadavků na novou aplikaci. Na základě analýzy bylo rozhodnuto o vývoji webové aplikace „Provozní deník radiolokátoru“. Dále byl proveden návrh nové aplikace s využitím jazyka UML a byl proveden datový návrh aplikace. Následně byla vyvinuta aplikace s využitím nabytých znalostí technologií a nástrojů z teoretické části. Na závěr byla aplikace otestována, byl zhodnocen vývoj a bylo navrhnuto několik vylepšení.

Silnou stránkou řešení je, že je napsané objektově. Je možno snadno dosáhnout rozšíření funkcionality v případě, že bylo potřeba rozšířit škálu dostupných funkčností, které v tomto okamžiku nebyly nezbytné. Další silnou stránkou je použití bezpečnostního standardu frameworku Spring, který se postupně stává standardem pro webové aplikace.

Slabou stránkou řešení je, že počítá pouze s nasazením v intranetové síti s uživateli, kteří se nebudou snažit o napadení aplikace nebo její databáze. Aplikace není chráněna před SQL injection.

Výsledkem práce je funkční webová aplikace „Deník radiolokátoru“.

Řádné prostudování použitých technologií a jejich následné využití ve vlastní práci autorovi velmi zjednodušilo vývoj a implementaci webové aplikace. Pro další vývoj se autor práce určitěš vrátí k nástrojům Maven a SpringBoot, jejichž přínos při programování a sestavovaní byl velký.

7 Seznam použitých zdrojů

[1] ORACLE. Your First Cup. *Oracle.com* [online]. 2012 [cit. 2017]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

[2] SCHILDT, Herbert. *Mistrovství - Java*. 1. Brno: Computer Press, 2014.

[3] ROUBALOVÁ, Eliška. *Java bez předchozích znalostí*. 1. Brno: Computer Press, 2015.

[4] ORACLE. Overview of Enterprise Applications. *Oracle.com* [online]. 2012 [cit. 2017]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html

[5] ORACLE. Java EE Servers. *Oracle.com* [online]. 2012 [cit. 2017]. Dostupné z: https://docs.oracle.com/javaee/6/firstcup/doc/gcrkq.html

[6] MATULÍK, Petr. Spring Framework I – úvodní pojmy. *Http://vsadnajavu.cz*. 2005. Dostupné také z: http://vsadnajavu.cz/2005-04/odborne/spring-framework/spring-framework-i-uvodni-pojmy/

[7] JANA, Abhisek. An Introduction to Spring Boot. *Http://www.adeveloperdiary.com*. 2016. Dostupné také z: http://www.adeveloperdiary.com/java/spring-boot/an-introduction-to-spring-boot/

[8] JSP - Overview. *Www.tutorialspoint.com* [online]. 2018 [cit. 2018]. Dostupné z: https://www.tutorialspoint.com/jsp/jsp_overview.htm

[9] Hibernate - Overview. *Www.tutorialspoint.com* [online]. 2017 [cit. 2017]. Dostupné z: https://www.tutorialspoint.com/hibernate/hibernate_overview.htm

[10] ČÁPKA, David. Popis MVC architektury. *Www.itnetwork.cz* [online]. 2012 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/php/mvc/objektovy-mvc-redakcni-system-v-php-popis-architektury

[11] ŠUBRTA, Václav. Základy relačních databází, jejich využití v programování webu. *Http://gml.vse.cz* [online]. 2014 [cit. 2017]. Dostupné z: http://gml.vse.cz/data/oppa-webdesign/zaklady-db.html

[12] ČÁPKA, David. 1. díl - Úvod do JavaScriptu. *Www.itnetwork.cz*. 2013. Dostupné také z: https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk/?all-comments#comments

[13] GARRETT, Jesse. Ajax: A New Approach to Web Applications. *Www.adaptivepath.com*. 2005. Dostupné také z: https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php

[14] JQuery - Overview. *Www.tutorialspoint.com* [online]. 2018 [cit. 2017]. Dostupné z: https://www.tutorialspoint.com/jquery/jquery-overview.htm

[15] DOYLE, Matt. What Is jQuery?. *Www.elated.com*. 2010. Dostupné také z: https://www.elated.com/articles/what-is-jquery/

[16] C, Shameer. Working with jQuery DataTables. *Www.sitepoint.com* [online]. 2013 [cit. 2018]. Dostupné z: https://www.sitepoint.com/working-jquery-datatables/

[17] VERACODE. What is an Integrated Development Environment (IDE)?. *Www.veracode.com/*. 2017. Dostupné také z: https://www.veracode.com/security/integrated-development-environments

[18] COMPONENTSOURCE.COM. About IntelliJ IDEA. *Www.componentsource.com* [online]. 2018 [cit. 2017]. Dostupné z: https://www.componentsource.com/product/intellij-idea/about

[19] HORDĚJČUK, Vojta. Maven. *Http://voho.eu*. 2018. Dostupné také z: http://voho.eu/wiki/maven/

[20] CHACON, Scot a Ben STRAUB. Pro Git. *Git-scm.com*. 2. Nevim: Apress, 2014. Dostupné také z: https://git-scm.com/book/cs/v1/%C3%9Avod-Spr%C3%A1va-verz%C3%AD

[21] KUČERA, František. Distribuované verzovací systémy – úvod. *Http://www.abclinuxu.cz* [online]. 2011 [cit. 2017]. Dostupné z: http://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1

[22] VALKOVIC, Patrik. Git - Historie a principy. *Www.itnetwork.cz/* [online]. 2014 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/software/git/git-tutorial-historie-a-principy

[23] ČÁPKA, David. UML - Use Case Diagram. *Www.itnetwork.cz* [online]. 2013 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram/?all-comments#comments

[24] ČÁPKA, David. UML - Use Case Specifikace. *Www.itnetwork.cz* [online]. 2014 [cit. 2017]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-specifikace-diagram

[25] Konvence pro psaní a odevzdávání programů v Javě. *Java.vse.cz* [online]. 2009 [cit. 2017]. Dostupné z: https://java.vse.cz/4it101/Konvence

[26] HORDĚJČUK, Vojta. JPA (Java Persistence API). *Http://voho.eu* [online]. 2017 [cit. 2017]. Dostupné z: http://voho.eu/wiki/java-jpa/

[27] ALEX, Ben, Luke TAYLOR, Rob WINCH, Gunnar HILLERT, Joe GRANDJA a Jay BRYANT. Spring Security Reference. *Docs.spring.io* [online]. 2017 [cit. 2017]. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.0.0.RELEASE/reference/htmlsingle/#preface

[28] MOROSYSTEMS. Spring Framework I – úvodní pojmy. *Http://vsadnajavu.cz*. 2005. Dostupné také z: http://vsadnajavu.cz/2005-04/odborne/spring-framework/spring-framework-i-uvodni-pojmy/

[29] ŠEDA, Jan. Úvod do JDBC. *Www.interval.cz* [online]. 2003 [cit. 2018]. Dostupné z: https://www.interval.cz/clanky/uvod-do-jdbc/

8 Přílohy

Seznam příloh:

- Příloha A: Soubor Pom.xml
- Příloha B:

Seznam příloh na CD:

- Aplikace Provozní deník radiolokátoru
- Databáze s testovacími daty
- Kopie bakalářské práce
- Kopie teze bakalářské práce

Příloha A

Soubor „pom.xml“

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>Roger</groupId>
    <artifactId>Prihlasovani</artifactId>
    <version>1.0-SNAPSHOT</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.6.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>

        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>jquery</artifactId>
```

```
            <version>3.2.1</version>
        </dependency>

        <dependency>
            <groupId>com.github.darrachequesne</groupId>
            <artifactId>spring-data-jpa-datatables</artifactId>
            <version>4.1</version>
        </dependency>

    </dependencies>

    <properties>
        <project></project>
        <java.version>1.8</java.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```