Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Webový informační systém na plánování a řízení agilního softwarového projektu

Karolína Pikorová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Karolína Pikorová

Systémové inženýrství a informatika
Informatika

Název práce

Webový informační systém na plánování a řízení agilního softwarového projektu.

Název anglicky

Web information system on agile software project planning and management.

Cíle práce

Cílem projektu je navrhnout a otestovat manažerský nástroj pro řízení agilních softwarových projektů kombinovanou metodou XP a Scrum. Výsledný program bude fungovat jako webová aplikace a bude mít různé úrovně přístupu pro různé role v projektu. Součástí řešení bude přehledný grafický panel zobrazující hotové a nehotové (naplánované) části projektu tak, aby z něj mohl vedoucí projektu odhadnout zbývající čas a potřebné lidské zdroje do dokončení projektu.

Metodika

Nejprve bude vypracována literární rešerše na metody Scrum a XP a jejich možné kombinace. Potom bude vypracována analýza a návrh podle standardů sw. inženýrství, především UML a vybrán programovací jazyk pro nejvhodnější implementaci. Součástí dokumentace bude malá uživatelská příručka.

Doporučený rozsah práce

80 – 100 stran

Klíčová slova

WWW; agile development; Scrum; XP; project management

Doporučené zdroje informací

BOULTON, Mark. Designing for the web. Penarth: Mark Boulton Design, [2009]. ISBN 978-0-9561740-7-9
ELLIOTT, Eric. Programming JavaScript applications. Sebastopol: O'Reilly, [2014]. ISBN 978-1491950296.
SUTHERLAND, Jeffrey Victor. Scrum: the art of doing twice the work in half the time. New York: Crown Business, [2014]. ISBN 978-0385346450.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 3. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 3. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 29. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci " Webový informační systém na plánování a řízení agilního softwarového projektu" jsem vypracovala samostatně pod vedením vedoucího diplomové 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 autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 6.4.2020 _____

Poděkování

Ráda bych touto cestou poděkovala doc. Ing. Vojtěchu Merunkovi, Ph.D. za odborné vedení a odbornou pomoc při zpracovávání této práce a blízkým přátelům a rodině za podporu, která mi při psaní práce pomáhala.

Webový informační systém na plánování a řízení agilního softwarového projektu

Abstrakt

Tato diplomová práce se zabývá problematikou vývoje softwaru. Analyzuje a popisuje jednotlivé metodiky jak pro tradiční, tak i agilní vývoj. Mezi tradiční metodiky zahrnuje vodopádovou metodiku, spirálovou metodiku a metodika Rational Unified Process. Jako agilní metodiky představuje extrémní programování, lean development, vývoj řízený testy a scrum.

Hlavním cílem této práce je navrhnout, vytvořit a vyzkoušet informační systém jako manažerský nástroj pro řízení agilních softwarových projektů. Návrh aplikace je vypracován podle standardů UML a dalších norem softwarového inženýrství. Pro implementaci byl zvolen framework Angular a ASP.NET s jazykem C#, jako webový server byl vybrán IIS od Microsoftu a Systém Řízení Báze Dat MS SQL server 2019.

Výsledně vytvořená aplikace podporuje a zjednodušuje vývoj softwaru pomocí agilních metodik.

Klíčová slova: WWW, agilní vývoj, Scrum, XP, projektový management

Web information system on agile software project planning and management

Abstract

This diploma thesis deals with software development. It analyses and describes individual methodologies for both traditional and agile development. Traditional methodologies include waterfall methodology, spiral methodology and Rational Unified Process methodology. As agile methodology represents extreme programming, lean development, test driven development and scrum.

The main aim of this work is to design, create and test an information system as a management tool for agile software projects management. The application design is based on UML standards and other software engineering standards. For implementation was chosen framework Angular and ASP.NET with C # language, as the web server was chosen IIS from Microsoft and data management system MS SQL server 2019.

The resulting application supports and simplifies software development using agile methodologies.

Keywords: WWW, agile development, Scrum, XP, project management

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Tato diplomová práce se zabývá problematikou vývoje webových aplikací použitých jako informační systém. Specifičtěji se bude zabývat vývojem webové aplikace, která bude podporou pro agilní vývoj software.

V první části práce jsou popsána teoretická východiska potřebná pro vývoj webových aplikací. Patří mezi ně webové služby, komunikace po internetu a technologie pro vývoj. Dále jsou sepsány metodiky pro vytváření software. Rozebrány jsou zde jak tradiční metodiky, tak i metodiky agilní. Mezi tradiční metodiky jsou zahrnuté vodopádová metodika, spirálová metodika a metodika Rational Unified Process. Mezi agilními metodikami jsou extrémní programování, lean development, vývoj řízený testy a scrum. Okrajově jsou zde také popsány databázové systémy a jejich komunikace s aplikacemi. Na závěr této části jsou vloženy standardy UML, jsou zde rozebrány jednotlivé diagramy a jejich rozdělení do základních skupin.

Druhá část obsahuje vlastní analýzu a implementaci webové aplikace následované jejím testováním. V analýze je zaobíráno požadavky a funkčními prvky, které má aplikace splňovat. Nechybí zde také výběr správného programovacího jazyka. Po důkladné analýze je vytvořen návrh systému. Dále jsou popsány technologie, které byly pro vývoj použity a následuje vlastní implementace a testování aplikace. Vytvořená aplikace byla veřejně nasazena jako prototyp. Na konec této části je připojena malá uživatelská příručka.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem projektu bude navrhnout a otestovat manažerský nástroj pro řízení agilních softwarových projektů kombinovanou metodou XP a Scrum. Výsledný program bude vytvořen jako webová aplikace a bude mít různé úrovně přístupu pro různé role v projektu. Součástí řešení bude přehledný grafický panel zobrazující hotové a nehotové (naplánované) části projektu tak, aby z něj mohl vedoucí projektu odhadnout zbývající čas a potřebné lidské zdroje do dokončení projektu.

2.2 Metodika

Nejprve bude vypracována literární rešerše na metody Scrum a XP a jejich možné kombinace. Potom bude vypracována analýza a návrh podle standardů sw. inženýrství, především UML a vybrán programovací jazyk pro nejvhodnější implementaci. Součástí dokumentace bude malá uživatelská příručka.

3 Teoretická východiska

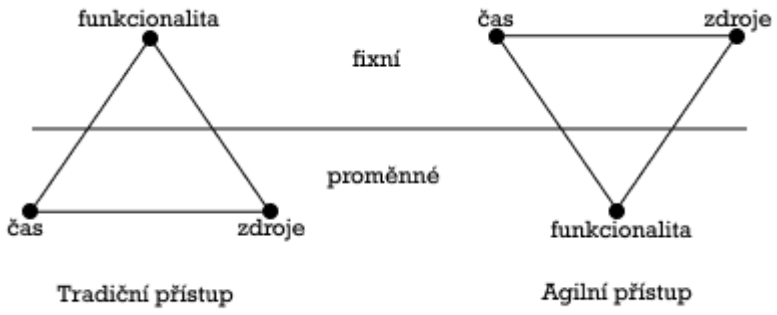
Kapitola rozebírá základní stanoviska pro vývoj webové aplikace, jsou zde rozepsány jednotlivé metodiky pro vývoj softwaru obecně. Jedná se jak o metodiky tradiční, tak i agilní. Dále jsou popsány technologie, které se při vytváření webových aplikací používají. A nedílnou součástí příprav vývoje softwaru jsou UML diagramy, které jsou zde v neposlední řadě také popsány.

3.1 Metodiky vývoje softwaru

Protože se dále v kapitole budeme zabývat především metodikami, musíme si pojem *metodika* blíže upřesnit.

Metodika představuje v obecném smyslu souhrn metod a postupů pro realizaci určitého úkolu. V souvislosti s vývojem softwaru nám metodika říká, jakým způsobem budeme při vývoji softwaru postupovat. Označuje komplexní postupy a návody, rozděluje jednotlivé role a definuje je ve smyslu jejich obsahu a činnosti. Zahrnuje veškeré etapy řešení vývoje a říká, jaká bude posloupnost jednotlivých činností, kdo a jak bude projekt řídit a jak bude probíhat jeho plánování (Bruckner, 2012).

Metodiky vývoje softwaru mohou být rozděleny do dvou hlavních skupin. Jedna skupina se bude nazývat *Tradiční metodiky vývoje software* a druhé se bude říkat *Agilní metodiky vývoje software*. Základní rozdíl mezi těmito dvěma skupinami je v tzv. trojimperativu, kterému se někdy říká železný trojúhelník. Imperativ projektu určuje omezení, tedy co je důležité v projektu dodržet a na co si dát pozor. Trojimperativ zohledňuje náklady na projekt, vymezený čas pro dokončení projektu a funkcionalitu projektu (Kadlec, 2004).



Obrázek 1 - Porovnání tradičního a agilního přístupu (Kadlec, 2004)

Tradiční metodiky definují funkcionalitu, tedy množinu požadavků, jako fixní. Dle ní se dále pak odhaduje čas pro dokončení projektu a náklady potřebné na realizaci, které jsou proměnnými veličinami. Naopak agilní metodiky vývoje považují za neměnné náklady na projekt a čas. Proměnná veličina je funkcionalita, která je přizpůsobována prioritám zákazníka (Kadlec, 2004).

Další rozdíly a základní charakteristiky těchto dvou skupin metodik budou popsány v následujících podkapitolách.

3.1.1 Tradiční metodiky

Tradiční metodiky jsou označovány jako tradiční nejen proto, že vznikly dříve než metodiky agilní, ale především proto, že tyto metodiky uplatňují tradiční přístupy k vývoji software (Myslín, 2016).

Tradiční přístupy považují vývoj za definovaný proces, který je možné přesně popsat. Snahou je co nejlépe určit jednotlivé termíny a jednotlivé požadavky. Je zde kladen poměrně velký důraz na to, aby bylo vše precizně dokumentováno. Od komunikace se zákazníkem, přes sběr veškerých požadavků a samotný vývoj, až po testování, předávání a následnou údržbu (Bruckner, 2012).

Role v tradičních metodikách bývají zcela přesně dány. Lidé mají svou specializaci a příliš se neangažují v těch fázích a činnostech vývoje softwaru, které jim nepřísluší. Proto pro splnění úkolů je často nutná kooperace a komunikace mnoha lidí, což může vývojový proces více prodlužovat a komplikovat (Myslín, 2016).

I kdyby se to nemuselo zdát, tradiční metodiky mají určité výhody, kterými jsou určitý řád, pořádek, jistota a předvídatelnost. Proto se spíše uplatňují v projektech většího rozsahu, v projektech, na kterých spolupracuje více týmů nebo v projektech, ve kterých se integruje více různých technologií a systémů (Myslín, 2016).

Dále budou popsány některé ze základních tradičních metodik.

3.1.1.1 Vodopádová metodika

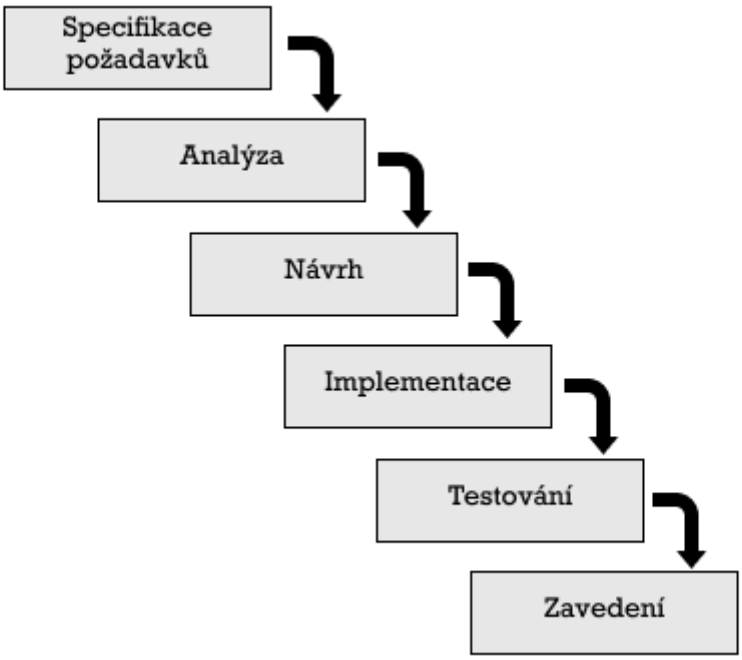
Vodopádová metodika, také někdy označována jako vodopádový model vývoje softwaru, patří mezi nejstarší metodiky vývoje softwaru. Vznikla a rozšířila se zejména v 70. a 80. letech minulého století. Inspirovala se postupy v průmyslu a rozdělila vývoj softwaru do postupně prováděných fází. Pojem vodopád se začal používat proto, že jednotlivé fáze následují po sobě a jejich grafické znázornění připomíná vodopád (Bruckner, 2012).

Vodopádový model je tedy sekvence všech podstatných fází, které jsou prováděny za sebou s žádnými nebo minimálními iteracemi. Díky sekvenci navazujících fází je vývoj postupný. Dokud není předešlá fáze hotová, tak nemůže začít fáze další, a proto není možné provádět více fází najednou (Kadlec, 2004).

Pro lepší pochopení lze vybranou vodopádovou metodiku charakterizovat několika body (Myslín, 2016):

1. Lineární průběh – každá fáze následuje po předchozí, nikdy se nevracíme zpět.
2. Jednoznačnost – vždy víme, v jaké fázi se nacházíme.
3. Úplné zadání – do další fáze vstoupíme, když je předchozí fáze se svými výstupy dokončena.

Vodopádový model odhaluje své nedostatky v těch případech, kdy není možné specifikovat všechny požadavky na začátku projektu, anebo je nutné během vývoje provádět časté změny. Jeho nevýhodou je také skutečnost, že zákazník je do vývoje zapojen jen na začátku a konci procesu a nemá tak možnost kontrolovat průběh projektu. Hlavním problémem modelu je pozdní zavedení programového systému, které se provádí až po naprogramování všech modulů. Při zavádění se často zjistí problémy, které vyžadují změny návrhu, přeprogramování a vedou k celkovému zpoždění projektu (Bruckner, 2012).



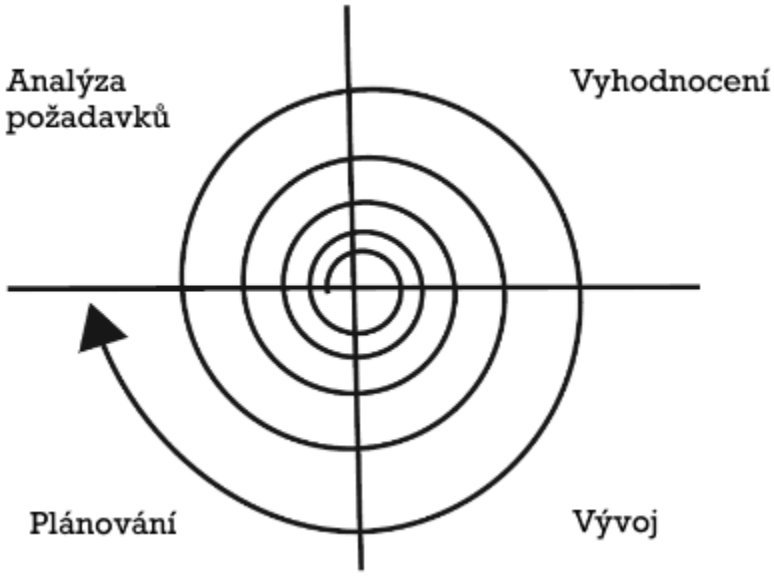
Obrázek 2 - Schéma vodopádového modelu (Bruckner, 2012)

3.1.1.2 Spirálová metodika

Metodika byla navržena tak, aby se pokusila odstranit některé nedostatky vodopádového modelu. Aby toho dosáhla, zavádí do procesu vývoje dva důležité koncepty, a to iterativní přístup a opakovanou, důslednou identifikaci rizik. Riziky se rozumí cokoli, co může mít negativní vliv na vyvíjený software (Myslín, 2016).

Iterativní přístup využívá ze schematického pohledu stejné fáze jako vodopádový model. Vše zde však probíhá v cyklech (iteracích). Mezi hlavní výhody patří výrazně kratší doba jednotlivých fází a tím je možné vydávat jednotlivá sestavení softwaru k validaci uživateli velmi často. V úvodních fázích existuje jen obecná dokumentace požadavků, která se postupně doplňuje o detaily v průběhu vývoje software (Šilhavý, 2013).

Vývojový proces je vyobrazen pomocí spirály, kde každá otáčka spirály znamená průběh jedné fáze vývoje daného softwarového produktu. Jedna otočka se zabývá proveditelností, druhá požadavky a podobně. V základu se spirála dělí na čtyři základní části. V první etapě se provede nastavení procesu, jsou nalezeny omezení pro proces. Definují se konkrétní cíle a provádí se plánování a identifikují se rizika. Ve druhé fázi se řeší nalezená rizika a přijímají se kroky pro minimalizaci jejich dopadu. Třetí fáze se zabývá vývojem a validací. Vybírá se zde vhodná metodika pro vývoj. Čtvrtá fáze je rozhodovací, rozhoduje se, zda pokračovat další otáčkou spirály (Sommerville, 2013).



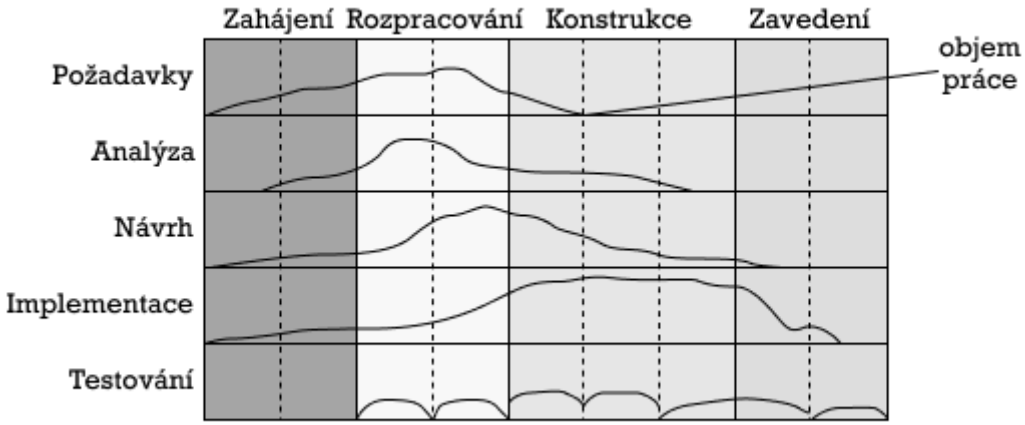
Obrázek 3 - Schéma spirálového modelu (Šilhavý, 2013)

3.1.1.3 Metodika Rational Unified Process

Metodika je deskriptivní (předepisuje procesy, činnosti, produkty a další prvky), dobře definovaný proces vývoje softwaru a je založena na iterativním a přírůstkovém procesu (Sommerville, 2013).

Proces vývoje software lze popsat v rámci dvou dimenzí. Tyto dimenze jsou znázorněny osami, což je dobře vidět na obrázku 4. Horizontální osa znázorňuje dynamický pohled na proces, který je vyjádřen pomocí cyklů, fází, iterací a milníků. Vertikální osa reprezentuje statické hledisko procesu, popis činností, artefaktů, pracovníků a pracovních toků (Buchalcevová, 2005).

V každé fázi, která se odehrává na horizontální ose, probíhá jedna nebo více iterací, která obsahuje veškeré prvky vývoje softwaru (požadavky, analýza, návrh, implementace, testování). Iterace vytvářejí základní linie, které obsahují fungující část softwaru a přidruženou dokumentaci. Základní linie jsou na sebe postupně vrstveny, dokud není dosaženo finální verze softwaru. Rozdílu, který se vytvoří mezi dvěma základními liniemi se říká přírůstek (Bruckner, 2012).



Obrázek 4 - Schéma metodiky RUP (Buchalcevová, 2005)

Životní cyklus software je rozdělen na cykly. Předmětem každého cyklu je nová verze produktu. Jeden vývojový cyklus se skládá ze čtyř po sobě jdoucích fází (Buchalcevová, 2005):

- Počáteční fáze
- Fáze rozpracování
- Konstrukční fáze

- Fáze zavedení

V počáteční fázi se realizují cíle projektu, požadavky, vytváří se harmonogram, odhadují se náklady a definují rizika. Součástí může také být vytvoření modelu nebo jednoduchého prototypu, na kterém se ověří, zda je možné splnit klíčové požadavky. Počáteční fáze končí rozhodnutím, jestli je projekt za daných požadavků, dostupných technologií, zdrojů a rozpočtu možné realizovat. V rámci fáze rozpracování je vytvořen prototyp, který má ověřit, všechny architektonické principy a umožní zpřesnění plánu realizace softwaru. Obsahem konstrukční fáze je návrh a realizace systému včetně testování. Prosazuje se zde pokud možno paralelní vývoj. Ve fázi nasazení se zjišťuje, zda mohou uživatelé systém používat. Je zde zahrnuto školení, předání dokumentace, vytvoření help-desk a podobně (Buchalcevová, 2005).

Metodika RUP se zaměřuje pouze na vývoj nového řešení realizovaný objektově orientovaným způsobem. Podstatným nedostatkem metodiky je její zaměření pouze na vývoj řešení (nezahrnuje provoz a údržbu) a pouze na softwarově inženýrské role. Silnou stránkou metodiky je její integrace s *CASE nástroji*, které podporují především oblast analýzy a návrhu, testování, řízení konfigurací a správu požadavků (Bruckner, 2012).

Nástroje typu CASE (Computer Aided Software Engineering – počítačem podporované softwarové inženýrství) umožňují využívat různé grafické modely, které výrazně usnadňují orientaci ve větších projektech a umožní pochopit velmi dobře jednotlivé vazby mezi jednotlivými strukturami v rámci projektu (Myslín, 2016).

3.1.2 Agilní metodiky

V nedávné době došlo ke změnám v technologiích a ekonomickém prostředí a vyvstaly požadavky na rychlé zavedení softwaru. To mělo za následek, že se začaly prosazovat metodiky, které umožňují vytvořit řešení velmi rychle a pružně jej přizpůsobovat měnícím se požadavkům. Tyto další metodiky jsou označovány jako agilní. Hlavní myšlenku, kterou prosazují je, že jediná cesta pro prověření správnosti navrženého softwaru je vyvinout jej co nejrychleji, předložit zákazníkovi a na základě zpětné vazby jej upravit (Buchalcevová, 2005).

Základní kámen agilním metodikám položila skupina inovativních lidí, kteří se sešli, aby se pokusili najít hodnoty, které spojují úspěšné týmy a jejich projekty a v únoru roku 2001 podepsali *Manifest agilního vývoje software*. Manifest na základě

zjištěných a popsaných hodnot deklaruje čtyři zásadní principy, které je třeba při vývoji software uplatňovat. Dáváme přednost (Stellman, 2014):

- Jednotlivci a interakci před procesy a nástroji
- Fungující software před vyčerpávající dokumentací
- Spolupráci se zákazníkem před vyjednáváním o smlouvě
- Reagování na změny před dodržováním plánu

Z principů popsaných výše, které je třeba při vývoji software uplatňovat, bylo definováno dvanáct hlavních principů agilních metodik (Agile Manifesto, 2001):

1. Nejvyšší prioritou je zákazník, časně a průběžně mu dodáváme hodnotný software.
2. Změny v požadavcích jsou vítané i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Fungující software je dodáván v kratších intervalech (týdny až měsíce).
4. Spolupráce lidí z byznysu a vývoje probíhá po celou dobu projektu.
5. Projekty jsou budovány kolem motivovaných jednotlivců. Je jim vytvářeno prostředí, které podporuje jejich potřeby a dostávají důvěru, že při vývoji odvedou dobrou práci.
6. Informace sdělujeme nejúčinnějším a nejefektnějším způsobem, a to osobní konverzací. Platí pro informace z vnějšku i uvnitř vývojového týmu.
7. Pokrok měříme podle správné funkčnosti softwaru.
8. Sponzoři, vývojáři i uživatelé stále udržují trvalé tempo, protože agilní procesy podporují udržitelný rozvoj.
9. Pozornost je věnována technické výjimečnosti a dobrému designu.
10. Je maximalizované množství práce, kterou není potřeba udělat.
11. Týmy jsou samo-organizující, protože z toho vzcházejí nejlepší architektury, požadavky a návrhy.
12. Tým si sám koriguje a přizpůsobuje své chování a zvyklosti, aby se stal efektivnějším.

V následujících podkapitolách budou charakterizovány některé z agilních metodik.

3.1.2.1 Extrémní programování

Tato metodika odvozuje svůj název od toho, že dovádí do extrému mnohé principy známé z jiných metodik. Je určená zejména pro malé až středně velké týmy (2 až 10

programátorů), které vyvíjejí software, jehož zadání není jasné nebo se mění. Extrémní programování je založeno na respektování čtyř základních hodnot, které zdůrazňuje (Myslín, 2016):

1. Jednoduchost – základem této hodnoty je vytvoření co nejjednodušší verze softwaru, která bude splňovat veškeré požadavky a zároveň bude fungovat.
2. Komunikace – všichni jsou součástí týmu a společně spolupracují od požadavků až po programování.
3. Zpětná vazba – po každé iteraci ukázat, co je hotové a brát vážně veškeré připomínky.
4. Odvaha – vždy je třeba mluvit pravdivě o průběhu vývoje a jeho odhadech.

Metodika vychází z principů a postupů běžných při vývoji software a dovádí je do extrémů (Buchalcevová, 2005):

- Párové programování – probíhá v párech u jednoho počítače, jeden přemýšlí o implementaci metod a druhý, jak strategicky napsat testy a metodu zjednodušit. Páry jsou dynamické a během dne se mění.
- Společné vlastnictví kódu – každý může provést jakoukoli změnu v softwaru. Na každou část kódu dohlíží mnoho lidí.
- Refaktorizace – změna struktury softwaru bez změny jeho funkčnosti. Pokud refaktorizujeme v průběhu celého životního cyklu projektu, šetříme čas a zvyšujeme kvalitu.
- Jednoduchý návrh – prosazuje se co nejjednodušší možné řešení, které vyhovuje požadovaným funkcím, každé složitější řešení je třeba ihned nahradit.
- Metafora – tým definuje společnou vizi fungování systému, chápe, jak systém pracuje a používá společnou terminologii.
- Nepřetržitá integrace – integrace a sestavení se provádí několikrát denně.
- Plánovací hra – řeší plánování dodávky (zákazník prezentuje požadavky a programátoři odhadují jejich náročnost) a plánování iterace (tým určuje, kolik úkolů bude vyvíjet na základě předchozích zkušeností).

3.1.2.2 Lean development

Lean znamená štíhlý a přesně to nám také ukazuje, jak metodika přistupuje k vývoji software. Zatímco většina agilních metodik se zabývá taktickou úrovní, lean development

se zaměřuje spíše na strategickou úroveň s vazbou na podnikovou strategii. Je zaměřena zejména na řízení vývoje software, méně pak na softwarově inženýrskou oblast. Omezuje rozdělanou práci a soustřeďuje se na to, aby byly jednotlivé požadavky dokončeny co nejrychleji (Kadlec, 2004).

Z popisu výše vyplývá, že lean development přímo nepopisuje, jak software vyvíjet, ale poskytuje řadu principů a pravidel, díky kterým je vývoj efektivnější (Buchalcevová, 2005):

- Odstraňuje zbytečné – znamená odstranit vše, co nepřináší hodnotu konečnému produktu.
- Minimalizovat zásoby – při vývoji software je zásobou dokumentace. Místo detailní specifikace stačí pár stran pravidel a dokumentace odchylek.
- Maximalizovat tok – zkrátit dobu vývoje (aplikujeme iterativní vývoj).
- Vývoj tažený poptávkou (rozhodovat se co nejpozději) – praktiky vývoje software, které dokážou přizpůsobit dodávku přímo požadavkům zákazníka představují v měnícím se prostředí konkurenční výhodu.
- Pracovníci s rozhodovací pravomocí (rozhodovat co nejníže) – programátoři chápou, jak jejich práce přispívá celkovému cíli a musí mít možnost rozhodovat.
- Uspokojovat požadavky zákazníků – časté důvody neúspěchu projektů jsou způsobeny chybějícími nebo nesprávnými požadavky.
- Zavést zpětnou vazbu – zavádíme zpětnou vazbu a doplňujeme požadavky postupně. Tím také provádíme změny v průběhu vývoje.
- Odstranit lokální optimalizaci – nemá smysl optimalizovat stávající řešení, když probíhají neustálé změny.
- Partnerství s dodavateli – pro zákazníka je důležitá hodnota, je umožněno nakupovat komponenty.
- Zavést kulturu pro neustálé zlepšování – vytváří se podmínky pro motivaci a zlepšování procesů při vývoji software.

3.1.2.3 Vývoj řízený testy (TDD)

V současné době se prosazuje myšlenka definovat nejprve sadu testů a teprve potom psát samotný software. Při psaní software se pak stačí soustředit pouze na to, aby výsledek prošel napsanými testy. Jakmile software testy projde, programátor se zamyslí, jak by jej

měl dále vylepšit, zase napíše testy a opět se snaží software upravit tak, aby testy prošel (Pecinovský, 2012).

Životní cyklus vývoje softwaru pomocí řízených testů je klasicky rozdělen do následujících kroků (Dvořák, 2017):

- Napsat test – celý proces začíná vytvořením testů.
- Spustit testy a přesvědčit se, že všechny neprojdou – potvrdí se, že aby testy prošly, musí se vyvinout zdrojový kód. Pokud by některý z testů prošel, aniž by se cokoli vyvinulo, pak se musí sada testů přepracovat.
- Napsat kód – na základě kritérii jednotlivých testů se provádí vlastní vývoj.
- Kód automatickými testy prochází – v průběhu vývoje dochází k jeho automatickému testování. Tento proces se opakuje do doby, než vývoj zajistí, že veškeré testy projdou.
- Refaktorizace – závěrečná optimalizace kódu. Je až zde, kdy je zajištěno, že kód splňuje očekávání zákazníka.
- Opakování – po refaktorizaci je nutné opět projít všemi testy.

Postup doporučovaný vývojem řízeným testy má řadu výhod. První z nich je, že si programátor ujasní, co přesně má vytvářený software dělat. Dále při psaní kódu může kdykoli spustit připravenou sadu testů a kontrolovat, kolik jich proběhlo správně a kolik práce mu ještě zbývá. Ve chvíli, kdy projdou všechny testy, je programátor s prací hotov a již nepřemýšlí, jestli jsou v softwaru chyby (Pecinovský, 2012).

3.1.2.4 Scrum

Pravděpodobně nejznámější agilní metodika. Její název byl vybrán podle skrumáže (mlýna) v rugby proto, aby zdůraznil, že metodika scrum je podobně jako hra rugby adaptivní, rychlá a samo-organizující. Scrum je založen na přesvědčení, že vývoj software je empirický proces. Dle empirismu vycházejí znalosti ze zkušeností a rozhodovat by se mělo na základě toho, co je známo (Buchalcevová, 2005).

Scrum je založen na neustálém učení a přizpůsobování se měnícím faktorům a je zaměřen především na řízení projektu. Vývojový tým se pomocí zkušeností, které nabírá během vývoje, vyvíjí. Metodika je strukturována tak, aby týmu pomohla přirozeně se měnícím podmínkám a požadavkům zákazníka přizpůsobovat. Samotný vývoj probíhá v pevně daných časových intervalech, které jsou nazývány *sprinty*. V rámci sprintu

probíhají *scrum meetingy*, které jsou považovány za klíčové praktiky scrumu. Jedná se o každodenní patnácti minutové porady, které slouží ke koordinaci a integraci práce (Sutherland, 2014).

Scrumový tým se skládá *z product ownera*, *vývojového týmu* a *scrum mastera*. Týmy jsou samo-organizující, což znamená, že si týmy samy volí způsob, jak nejlépe vykonávat svou práci, než aby byly řízeny někým mimo tým. Také mají veškeré kompetence potřebné k dokončení práce (Stellman, 2014):

- Product owner (vlastník produktu) – je jedinou osobou odpovědnou za správu produktového backlogu (bude popsán dále) a za maximalizaci hodnoty softwaru, který vytváří vývojový tým. Zajišťuje viditelnost, transparentnost a přehlednost toho, na čem bude tým pracovat dále. Dále řadí položky v backlogu tak, aby bylo co nejlépe dosaženo cíle.
- Vývojový tým – je strukturovaný a má oprávnění se sám organizovat a řídit si vlastní práci. Optimální velikost týmu je dostatečně malá tak, aby tým zůstal flexibilní a dostatečně velká, aby tým mohl v rámci sprintu dokončit významný kus práce. Scrum nerozlišuje názvy pozic jednotlivých členů vývojového týmu bez ohledu na to, jakou práci tato osoba vykonává.
- Scrum master – je zodpovědný za prosazování a podporu scrum metodiky, tím že pomáhá pochopit scrumovou teorii, praxi, pravidla a hodnoty. Pomáhá vývojovému týmu vytvářet produkty s vysokou hodnotou a odstraňuje překážky, které brání vývojovému týmu v postupu. V rámci celé organizace pomáhá zaměstnancům a všem zúčastněným pochopit a přijmout scrum. A při pomoci vlastníkovi projektu zajišťuje, aby věděl, jak uspořádat produktový backlog a aby byl maximalizován získaný přínos.

Předepsané scrumové ceremonie, nebo také události, zajišťují pravidelnost a tím minimalizují potřebu dalších scrumem nedefinovaných schůzek. Všechny události jsou časově ohraničené tím, že každá má definovanou svou maximální délku trvání. Jakmile začne sprint, jeho délka je neměnná, nemůže být ani zkrácena ani prodloužena. Ostatní aktivity mohou skončit, když je dosaženo jejich cíle, aby nedocházelo k plýtvání časem a zabíraly přiměřenou dobu (Schwaber, 2013):

- Sprint – délka trvání je jeden měsíc nebo méně. Součástí každého sprintu je popis toho, co má být vytvořeno, včetně návrhu a plánu, který slouží jako vodítko k tomu,

jak to vytvořit. Sprint může být zrušen v případě, že cíl sprintu zastará, tedy kdy cíl už nedává smysl.

- Sprint planning – v součinnosti celého scrumového týmu je dohodnuta práce, která má být vykonána během sprintu. Vývojový tým odhaduje, které všechny funkčnosti budou během sprintu vyvinuty.
- Scrum meeting – každodenní schůzka vývojového týmu, která není delší než patnáct minut. Synchronizují se na ní aktivity, probírá se vykonaná práce od poslední a vytvoří se plán na dalších 24 hodin. Scrum meeting zlepšuje komunikaci, eliminuje potřebu dalších schůzek a identifikuje překážky, které mají být odstraněny.
- Sprint review – probírají se zde výsledky sprintu. Zabývá se výsledným softwarovým přírůstkem a přizpůsobuje se produktový backlog. Konečným výsledkem je revidovaný produktový backlog, který ukazuje položky, které se pravděpodobně zařadí do dalšího sprintu.
- Sprintová retrospektiva – scrum tým zde prozkoumává sebe sama a řeší zde průběh posledního sprintu, co se lidí, vztahů, procesů a použitých nástrojů týče. Dále se plánují kroky, pomocí kterých má dojít během příštího sprintu ke zlepšení (efektivnější, příjemnější).

Nástrojům k řešení problémů se říká artefakty a metodika scrum obsahuje tři hlavní, které jsou navrženy speciálně tak, aby byly všechny klíčové informace maximálně transparentní a všichni tyto artefakty chápali stejně (Sutherland, 2014):

- Produktový backlog – je prioritizovaný seznam všeho, o čem se ví, že je v rámci softwaru potřeba, ale nikdy není úplný. V úvodní fázi obsahuje pouze známé a nejlépe pochopené požadavky. Dále se vyvíjí stejně jako se vyvíjí software a prostředí. Je dynamický, stále se mění, aby identifikoval vše, co zákazník potřebuje.
- Sprintový backlog – je množina položek, které byly vybrány vývojovým týmem z produktového backlogu a označují nezbytnou práci, kterou je potřeba k naplnění cíle sprintu. Sprintový backlog je upravován po celou dobu sprintu, čímž se neustále utváří.

- Přírůstek – je tvořen dokončenou prací, která je hmatatelná a dá se nějakým způsobem ukázat. Je to součet všech položek produktového backlogu dokončených v průběhu sprintu.

3.2 Webová aplikace

Jedná se o aplikaci, kterou není nutno instalovat přímo na zařízení uživatele a lze ji snadno spustit z jakéhokoli zařízení pomocí webového prohlížeče, protože je spuštěna na straně serveru a webový prohlížeč se postará o její zobrazení. Největší výhody webových aplikací jsou, že se nemusí instalovat, uživatel nemusí nic aktualizovat, protože potřebuje jen webový prohlížeč a data jsou uchovávána na serveru, takže jsou přístupná odkudkoli. Protože ke spuštění webové aplikace je zapotřebí pouze prohlížeč, je nazývána též jako lehký klient (Šilhavý, 2013).

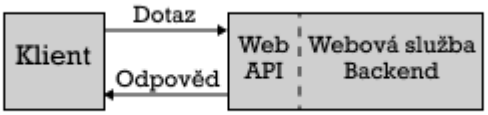
Přesněji je zde použita architektura klient / server, která je společná pro všechny služby na internetu. Program typu server obstarává samotnou službu a program typu klient zprostředkovává styk uživatele se serverem. Celá tato komunikace se odehrává pomocí formálních pravidel, kterým se říká protokol. Je to soubor daných pravidel, které musí program splňovat, aby byl schopen komunikovat s jiným odpovídajícím programem (Massé, 2012).

3.2.1 Hyper Text Transfer Protokol

Jedná se o jeden z nejdůležitějších protokolů internetu, jeho zkratka je HTTP a běžně se používá pro přenos dat mezi klientem a serverem. Pro lokalizaci prostředků používá URL (Uniform Resource Locator), který popisuje formalizovaný způsob vyjádření umístění nějakého dokumentu. URL se skládá ze způsobu přístupu k dokumentu, umístění serveru a cestě k dokumentu. Přístup k dokumentu může být http, nebo také https, který umožňuje přenášená data šifrovat a tím je chránit před odposlechy a jinými narušeními. Základní operace protokolu jsou načtení webové stránky, získání informací o webové stránce a odeslání dat z formuláře na server. Důležitá vlastnost, kterou HTTP protokol má je, že se po přenosu stránky spojení uzavře. Pro získání další stránky se musí mezi klientem a serverem vytvořit nové spojení (Pinkas, 2011).

3.2.2 REST API

Pod zkratkou API (Application Program Interface) se skrývá sada protokolů a nástrojů pro vytváření aplikací. V zásadě určuje, jak by měli dvě aplikace mezi sebou komunikovat a vyměňovat si data. Lze nastavit, zda se bude jednat o jednosměrnou či oboustrannou komunikaci. API pro komunikaci s webovými službami používají také programy typu klient. Webová služba je účelový webový server, který podporuje potřeby jiných aplikací, jedná se zde vždy o strojovou interakci. Vystavená webová služba má definované veřejné rozhraní a funkce, které jsou ostatním aplikacím předem známé. Při návrhu moderních webových služeb se používá architektura REST (Massé, 2012).



Obrázek 5 - Web API (Massé, 2012)

REST (Representational State Transfer) je orientován datově, a proto je používán pro jednotný a snadný přístup ke zdrojům (data, stavy aplikace). Všechny zdroje mají vlastní identifikátor URI (Uniform Resource Identifier – jednotný identifikátor zdroje) a REST definuje čtyři základní metody pro přístup k nim. URI je textový řetězec, který slouží k přesné identifikaci zdroje. Tyto čtyři základní metody jsou známé pod označením CRUD, což je vytvoření dat – Create, získání potřebných dat – Retrieve, změnu dat – Update a smazání dat – Delete. Tyto metody jsou implementovány pomocí odpovídajících metod protokolu HTTP (Elliott, 2014).

3.2.3 Webový server

Webový server je program, který nabízí služby ostatním programům (klientům). Klient pro získání určité webové stránky vytvoří spojení se serverem, svůj požadavek, který chce poslat přeloží do jazyka protokolu HTTP a odešle. Server trvale očekává požadavky na standartním portu 80 a hned jakmile nějaký požadavek přijde, ověří jej, najde příslušnou stránku a zašle ji klientovi. Portem se rozumí číslo, které je charakteristické pro určitou službu a slouží přímo k tomu, aby identifikovalo program, který danou službu bude vykonávat (Šilhavý, 2013).

Mezi dva nejběžněji používané webové servery, které dohromady ovládají přibližně sedmdesát procent trhu, patří webový server Apache a webový server Internet Information Services (zkratka IIS) (Up Guard, 2019):

- Webový server Apache – má velkou výhodu a tou je kompatibilita se všemi hlavními operačními systémy. Je to open-source aplikace a je spravována skupinou Apache Software Foundation. Kromě různých programovacích jazyků a databází, podporuje také nejrůznější formy autentizace a umí vytvářet logy návštěvnosti, které lze využít v pokročilé webové analytice.
- Webový server IIS – vyvinula a spravuje společnost Microsoft. Je přímo součástí operačního systému Windows a je kompatibilní pouze s ním. Pro vývojáře webových aplikací je připravena odlehčená verze IIS express.

3.2.4 JavaScript

Jazyk JavaScript je navržený pro skriptování v prostředí webového prohlížeče. Jeho syntaxe vychází z jazyka C a některé standartní rozhraní se podobají Javě, objektový a funkcionální model je inspirovaný jazyky Scheme a Self. Implementace JavaScriptu je dnes nedílnou součástí všech webových prohlížečů a jedná se o jednu z nejrozšířenějších technologií vůbec (Boulton, 2009).

V roce 2009 dochází k oživení myšlenky, že jazyk nemá důvod být používán jen v rámci klientského skriptování a lze jej použít pro programování serverového kódu. V témže roce se objevuje projekt Node.js (sada rozhraní a knihoven) dovolující spouštět JavaScriptový kód i mimo webový prohlížeč. Node.js se tak stal standartním řešením pro serverové vykonávání JavaScriptu (Žára, 2015).

Nejpoužívanější nástroj pro správu balíčku k serverovému JavaScriptu (Node.js) je nástroj npm. S npm se jednoduše pracuje přes příkazový řádek a všechny potřebné balíčky je možné nainstalovat příkazem `npm install`. Balíčky se v konkrétním projektu instalují do složky `node_modules`, která se sama nově vytvoří. Pokud se mají balíčky instalovat globálně, použije se parametr `-g` (Elliott, 2014).

Za svoji existenci si JavaScript vypěstoval poměrně velkou infrastrukturu různých knihoven, balíčků a dalších nástrojů (Žára, 2015):

- Preprocesor – je to nástroj, který umožňuje psát kód v upravené verzi jazyka. Ten je pak před spuštěním transformován do JavaScriptu. Nejčastěji transpilovaným

jazykem je TypeScript, který doplňuje informace o typech proměnných (typové anotace).

- Knihovna – je sbírka znovupoužitelných funkcí a objektů. Zpravidla mívá jedno konkrétní zaměření. Jedna z nejrozšířenějších knihoven je jQuery.
- Polyfill – je speciální kategorie knihoven. Přesněji se jedná o modul, který do prohlížeče doplňuje nějakou standardizovanou funkcionalitu, která povětšinou z historických důvodů chybí.
- Framework – jedná se o téměř hotová řešení pro celé třídy problémů, které si lze jen adekvátně upravit či nastavit podle konkrétních požadavků. Framework se zpravidla sestává z většího množství provázaných souborů. Nezřídka bývá objemnější než samotný aplikační kód. Při jeho používání následujeme konvence a postup, které stanovil jeho autor.

3.2.4.1 Vue.js

Lze říct, že se jedná o velmi přístupný, univerzální a výkonný open-source framework. Jedná se o progresivní framework, což znamená, že aplikaci jde rozdělit na části a ty vyvíjet nezávisle. Využívá se pro tvorbu uživatelských rozhraní. Je navržený tak, aby mohl být do již existujícího projektu začleněn okamžitě a jeho části přijímány postupně. Využívá se také v případě malých komponent na webu, které nemusí být přímo vázané na danou stránku (VueJS, 2019).

3.2.4.2 Angular

Frontendový framework pro tvorbu webových aplikací. Staví na komponentové architektuře se službami a využívá jazyk TypeScript namísto čistého JavaScriptu. Základem každé komponenty je programová třída, tím Angular staví na základech objektově orientovaného programování (OOP). K této třídě je pak navázána vlastní HTML šablona a i CSS. Díky tomuto způsobu je možné rozdělit webovou stránku na samostatné celky a poskládat ji právě pomocí OOP. Komponenty mají vlastní izolovanou funkcionalitu a snaží se ideálně být znovupoužitelné (Máca, 2019).

Pro samotnou logiku v aplikaci se používají služby, které slouží ke komunikaci a distribuci dat mezi klientskou aplikací a serverovým rozhraním API (rozhraní, kterým mohou komunikovat dvě aplikace mezi sebou) (Máca, 2019).

3.2.4.3 React

Jedná se o open-source knihovnu od společnosti Facebook pro tvorbu uživatelského rozhraní. Na rozdíl od různých jiných kompletních frameworků se soustředí přímo na jednu specifickou oblast. Základním stavebním kamenem zde tvoří komponenty, které jsou vlastně znovupoužitelné HTML elementy, jejichž skládáním vzniká komplexní uživatelské rozhraní aplikace. Tyto komponenty mají své vlastnosti a spravují svůj vnitřní stav (Máca, 2019).

Protože je React specificky zaměřenou knihovnou, používá se v praxi v kombinaci s dalšími knihovnami se značně rozdílným přístupem i různou architekturou. React se pak v rámci těchto použití stará o vykreslení uživatelského rozhraní (Máca, 2019).

3.3 Databáze

Databáze je místo, kam se ukládají všechny potřebné údaje. Přístup k potřebným údajům obstarává program, který se nazývá Systém Řízení Báze Dat (SŘBD) a pochází z anglického DataBase Management Systém (DBMS). Mezi SŘBD patří programy MS SQL Server, Oracle, Sybase, Informix, Progress a InterBase. Ceny takových programů se mohou pohybovat v desítkách až stovkách tisíců korun. Proto naštěstí existují i programy, které jsou šířené zdarma, jako MySql či PostgreSQL (Kosek, 1998).

Převážná většina dnes používaných SŘBD při uspořádání údajů v databázi vychází z relačního modelu dat. Název tohoto modelu vychází z relační algebry, což je matematický aparát, na kterém relační model staví. Podle modelu jsou údaje upořádané do tabulek, kde jedna tabulka shromažďuje data o jednom druhu objektů (Kosek, 1998).

Jednotlivé řádky představují objekty a sloupce pak obsahují informace o těchto objektech. Sloupcům se říká položky nebo atributy a jednotlivé řádky se pak nazývají záznamy. Aby se s tabulkami správně pracovalo, musí každý sloupec mít jednoznačný název, což pak umožňuje odvolávat se na obsah určitého atributu. Každá tabulka by také měla obsahovat primární klíč. Primární klíč je atribut, jehož hodnota je pro každý záznam jedinečná. Pro každý atribut tabulky musíme určit, jaký typ může obsahovat. Takový typ může být celé číslo, znakový řetězec nebo logické hodnoty. Databáze může obsahovat větší množství tabulek, proto i každá tabulka musí mít jednoznačné jméno (Kosek, 1998).

3.3.1 Komunikace

Aby mohly být údaje z databáze přístupné i ostatním aplikacím, musí SŘBD nabízet rozhraní pomocí kterého s ním mohou spolupracovat ostatní programy. Při komunikaci funguje model server / klient, kde SŘBD je nepřetržitě spuštěn nejčastěji jako démon nebo služba a očekává požadavky klientů. SŘBD je zde v roli serveru, a proto se mu také říká databázový server. Pro zadávání požadavků na databázový server aplikace nejčastěji používají jazyk SQL. Někdy se proto databázovým serverům zjednodušeně říká SQL-servery (Kosek, 1998):

- PostgreSQL – obsahuje některé pokročilé technologie, jako jsou objekty, dědičnost a možnost definice vlastních datových typů a funkcí. Původně byl vyvíjen na Kalifornské univerzitě profesorem informatiky Michaelem Stonebrakerem. Je zcela zdarma pro všechny verze Unixu, a i to je důvod proč je na mnoha serverech používán i přes svůj malý výkon.
- MySQL – je velice rychlý open-source databázový server, který běží na Unixu i na Windows. Jeho rychlost je jeho největší předností, vývojáři se na ni zaměřili již od začátku, a to i za cenu zjednodušení ostatních funkcí. Poprvé byl světu ukázán v roce 1996 a vyvinula ho švédská společnost MySQL AB. Nyní je vlastněn firmou Oracle.
- MS SQL Server – je běžnou volbou tam, kde se používají technologie Microsoftu. Jeho vývoj začal v roce 1995 firmami Microsoft a IBM. Celý jeho systém je navržen taky, aby mohl zpracovávat velké množství transakcí. Bývá vydáván v několika edicích.

SQL neboli Structured Query Language je specializovaný programovací jazyk, který je používán jako nástroj pro manipulaci, správu a organizování dat uložených v databázi. S jeho pomocí lze definovat strukturu tabulky, naplňovat sloupce tabulky daty nebo řídit přístup k datům pomocí přístupových práv. Ve většině případů je ale výsledkem úlohy popsané v jazyku SQL nějaká množina dat z jedné nebo více tabulek. Přesto jazyk není plnohodnotným programovacím jazykem, protože se v něm nenachází řídící programové konstrukce a další požadované prvky, které by měl obsahovat každý obecný programovací jazyk (Sadovski, 2010).

3.4 UML

Jazyk UML (Unified Modeling Language) je průmyslový standard. Je to modelovací jazyk vyvinutý pro systémové a softwarové vývojáře. Je používán pro specifikaci, vizualizaci, konstrukci a dokumentování systémů. Systémů, které nemusí být pouze softwarového charakteru. Přesto představuje soubor nejlepších inženýrských postupů, které pomáhají vývojářům specifikovat, vytvářet, vizualizovat a dokumentovat proces návrhu software. Osvědčil se při modelování velkých a složitých systémů (UML, 2005).

UML byl vytvořen firmou Rational Software a jejími partnery v průběhu 90. let. Jazyk je dnes standardizován a otevřen všem, což je jeho výhodou, a proto ho řada firem přijala jako svůj standard. Na specifikaci UML dnes dohlíží mezinárodní konsorcium OMG (Object Management Group), které je otevřenou institucí, ve které figurují firmy jako Microsoft, IBM, Rational Software a další (UML, 2005).

3.4.1 UML diagramy

Pro tvorbu modelu systému nebo pro tvorbu pohledů na systém definuje UML třináct typů diagramů. Diagramy lze rozdělit na dvě základní kategorie, a to na diagramy struktury (Structure diagrams), které zachycují statické strukturální aspekty, tedy z čeho je systém složený a na diagramy chování (Behaviour diagrams), které zachycují dynamické aspekty modelovaného systému, tedy jak systém funguje. V diagramech chování lze nalézt ještě samostatnou skupinu diagramů interakce (Interaction diagrams), které popisují interakce mezi jednotlivými částmi systému (Zelinka, 2009).

3.4.1.1 Diagramy Struktury

Základní popis diagramů z této kategorie. Obecně se zaměřují na popis struktury systému, tedy z čeho je systém složený (Zelinka, 2009), (Rejnková, 2009):

- Diagram tříd – je jedním z nejčastějších diagramů, je to graf s uzly a hranami zobrazuje statickou strukturu systému. Základními stavebními prvky jsou třídy a vztahy mezi nimi. Třídy představují uzly grafu a vztahy představují hrany grafu. Asociace je vztah mezi dvěma a více třídami. Říká, že objekty nejsou nezávislé a že lze mezi nimi poslat zprávu. Další vztah, který může mezi třídami existovat je kompozice. Označuje vztah, kdy odkazovaná třída je zásadní součást majitele a její existence bez majitele nemá smysl. Zánikem majitele tak zaniká i odkazovaná třída.

Vztah agregace říká, že je jedna třída součástí jiné třídy, tedy majitele. Tento vztah je volnější než kompozice, protože zánik majitele nezapříčiňuje zánik odkazované třídy. Generalizace je druh vztahu, když jedna třída je zobecněním vlastností jiné třídy. Jedná se tedy o vztah nadtřída a podtřída.

- Objektový diagram – popisuje objekty jako instance tříd a jejich vztahy. Je instancí diagramu tříd a zachycuje stav systému v jednom určitém časovém okamžiku. Používá se především pro znázornění určité konfigurace objektů ve speciálních situacích.
- Diagram komponent – popisuje strukturu realizace softwarového systému zobrazenou pomocí softwarových komponent, rozhraní a jejich vzájemných závislostí. Softwarovou komponentou se rozumí komponenta ve zdrojovém, binárním i spustitelném tvaru a dokumentace.
- Diagram balíčků – umožňuje sdružit elementy modelů UML do skupin a mezi těmito skupinami, které se nazývají balíčky znázornit závislost. Nejčastěji se balíčky používají pro sdružení souvisejících tříd. Balíčky také mohou být členy dalších balíčků.
- Diagram rozmístění zdrojů – popisuje topologii systému zachycenou pomocí softwarových a hardwarových prvků a vazeb mezi nimi. Hardwarovým prvkem se rozumí fyzicky existující zdroj nutný pro realizaci systému (počítač, lidský zdroj).
- Diagram vnitřní struktury – umožňuje znázornit interní strukturu komplexního prvku (třídy, komponenty) a zobrazit spolupráci tohoto prvku s ostatními prvky v systému. Poskytuje tak další informace, které se pomocí jiných diagramů hůře modelují.

3.4.1.2 Diagramy chování

Základní popis diagramů z této kategorie. Obecně popisují chování systému, tedy jak systém funguje (Zelinka, 2009), (Rejnková, 2009):

- Diagram případů užití – představuje základní funkční specifikaci systému. Popisuje chování z hlediska uživatele a zachycuje, které typy uživatelů se systémem pracují a jaké činnosti v rámci systému vykonávají. Definuje, co má systém dělat, ale už neříká, jak to bude dělat. Diagram se skládá z případů užití, aktérů a vztahů mezi nimi. Případ užití specifikuje logicky ucelenou část funkcionality systému, kterou

využívá aktér a která plní určitý cíl. Aktér vyjadřuje prvek okolí systému, který se systémem komunikuje, může to být uživatel, ale i externí systém. Vztahy mezi aktéry a případy užití jsou nazývány komunikační asociace či relace a znázorňují mezi nimi plynoucí tok informací

- Stavový diagram – zachycuje jednotlivé stavy objektu a přechody mezi nimi. Základními prvky diagramu jsou stavy, přechody a události. Stav objektu je situace, kdy modelovaný objekt splňuje nějakou podmínku, provádí nějakou operaci nebo čeká na událost. Přechody představují podmínky přechodu objektu z jednoho stavu do druhého a bývají zpravidla vyvolány určitou událostí. Pokud není v diagramu podmínka uvedena, znamená to, že přechod do dalšího stavu probíhá automaticky.
- Diagram aktivit – je speciálním případem stavového diagramu, kde všechny nebo alespoň většina stavů jsou aktivitami a zároveň všechny nebo většina přechodů jsou iniciovány dokončením aktivit. Chování systému je zde zachyceno pomocí sekvence činností řízených interními událostmi. Diagram aktivit se nejčastěji používá pro popis implementace operací.

4 Vlastní práce

Kapitola je věnována postupu tvorby zadané webové aplikace. V první podkapitole se postup tvorby aplikace rozebírá podrobněji a jsou zde vypsány jednotlivé kroky, pro dosažení požadovaného výsledku. Dále jsou analyzovány požadavky na funkčnost aplikace. V další podkapitole je rozebrán návrh systému aplikace a jsou zde rozebrány jednotlivé navržené modely. Pak jsou popsány použité technologie pro samotný vývoj a dále je popis implementace aplikace. Předposlední podkapitola se věnuje testování celé aplikace. Na závěr kapitoly je vložena uživatelské příručka, která má usnadnit práci s vytvořenou webovou aplikací.

4.1 Analýza

Samotná analýza všech požadavků a funkcionalit, které by měla aplikace splňovat je velmi důležitou součástí jejího vývoje. Slouží hlavně pro přípravu veškerých podkladů potřebných pro realizaci. Z těchto podkladů pak vychází celý návrh a implementace. Díky důkladnému naplánování celého vývoje aplikace lze větší problémy lehce rozložit na podproblémy.

Jedna z prvních důležitých věcí je naplánovat hlavní činnosti, ze kterých se bude vycházet při postupné tvorbě webové aplikace:

- Analýza požadavků a funkcionalit aplikace
- Návrh systému aplikace
- Vlastní implementace aplikace
- Testování aplikace
- Tvorba uživatelské příručky

4.1.1 Požadavky

Analýza požadavků je nezbytnou součástí procesu vývoje a následné implementace softwaru. Na základě analýzy požadavků lze určit a vyhodnotit, jaké řešení je potřebné a jaké funkce by měl požadovaný software umět. Nejdůležitějšími požadavky na vytvářenou aplikaci tedy jsou:

- Zobrazení projektů, správa projektů
- Zobrazení konkrétního projektu s veškerou naplánovanou prací

- Zobrazení jednotlivých User Stories, správa User Stories, přehledně je vidět, kolik práce je již v jednotlivých User Stories hotovo
- Zobrazení jednotlivých úkolů (Engineering Tasks) a správa těchto úkolů
- Různé stupně přístupu v aplikaci pro různé role uživatelů
- Správa uživatelů

K výše vypsanému požadavku, který se týká různého přístupu do různých částí aplikace v závislosti na roli, jakou uživatel má, je dobré tyto role přesně vypsat:

- Administrátor – má přístup do veškerých částí aplikace. Jeho nejdůležitější činnost je správa uživatelů. Může je přidávat, upravovat a mazat. Aplikace bude nastavena tak, že pouze administrátor může registrovat nového uživatele.
- Product Owner – může přidávat, spravovat a mazat projekty a vše co s projekty jakkoli souvisí (User Stories, Engineering Tasks).
- Scrum Master – stará se o práci uvnitř projektů, může User Stories a jednotlivé úkoly (Engineering Tasks) přidávat, spravovat a mazat.
- Člen vývojového týmu, označen jako User – projekty pouze zobrazuje. User Stories a úkoly v nich zadané může přidávat, spravovat a mazat.

4.1.2 Výběr vhodného programovacího jazyka

Součástí analýzy je také výběr vhodného programovacího jazyka. Protože se jedná o webovou aplikaci, která má řešit rozsáhlejší problém, bude rozdělena na dvě části, a to frontendovou a backendovou. Komunikace mezi těmito částmi bude realizována pomocí architektury REST. Vybrána byla z důvodu, že je nyní na vzestupu a používá se při řešení v moderně pojatých aplikacích.

Vybranou architekturu pro komunikaci mezi částmi aplikace lze jednoduše konstruovat pomocí ASP.NET s jazykem C# na backendu. Nejnovější verze ASP.NET a také verze, ve které bude tato část aplikace vyvíjena je 3.1.

Frontendová část bude řešena pomocí frameworku Angular, jelikož se jedná o nejrozumnější řešení pro větší aplikace, také už na trhu existuje delší dobu a jeho nejnovější verze (Angular 9) potvrdila, že se s frameworkem počítá i do budoucnosti a bude v něm psáno čím dál více aplikací. Na rozdíl framework React je používám spíše pro menší aplikace a framework Vue.js je poměrně nový, tedy není ještě tolik zavedený a rozšířený, navíc se také hodí spíše pro menší aplikace nebo jen pro části aplikací.

4.2 Návrh systému – modely

Když je analýza webové aplikace hotová, lze začít s návrhem systému. Data, se kterými aplikace bude pracovat budou uložena v databázi, proto se jedna podkapitola bude věnovat databázovému modelu. Aby bylo jasné, kteří uživatelé budou vykonávat které požadavky, bude v následující podkapitole namodelován diagram případů užití.

4.2.1 Diagram případů užití

Diagram byl vytvořen pomocí výsledků vycházejících z analýzy funkcionalit a požadavků na aplikaci. Jsou zde zobrazeni jednotliví uživatelé, kteří s aplikací interagují a činnosti jimi prováděné. Graficky znázorněný diagram je přiložen jako příloha 1 této práce.

Popis jednotlivých případů užití obsažených v diagramu:

Aktér	**Případ užití**	**Popis**
Administrátor	přidávat, spravovat, mazat uživatele	Zahrnuje úplné spravování uživatelů aplikace.
Administrátor	zobrazovat stránku pro správu uživatelů	Zobrazení stránky, kde lze manipulovat s registrovanými uživateli.
Administrátor, Product Owner	přidávat, spravovat, mazat projekty	Zahrnuje úplné spravování projektů.
Registrovaní uživatelé	přihlášení	Zaregistrovaný uživatel se může přihlásit.
Registrovaní uživatelé	přidávat, spravovat, mazat User Stories	Zahrnuje úplné spravování User Stroies.
Registrovaní uživatelé	zobrazovat produktový backlog	Registrovaní uživatelé si mohou zobrazovat produktový backlog.
Registrovaní uživatelé	přidávat, spravovat, mazat úkoly (Engineering Tasks)	Zahrnuje úplné spravování úkolů (Engineering Tasks).

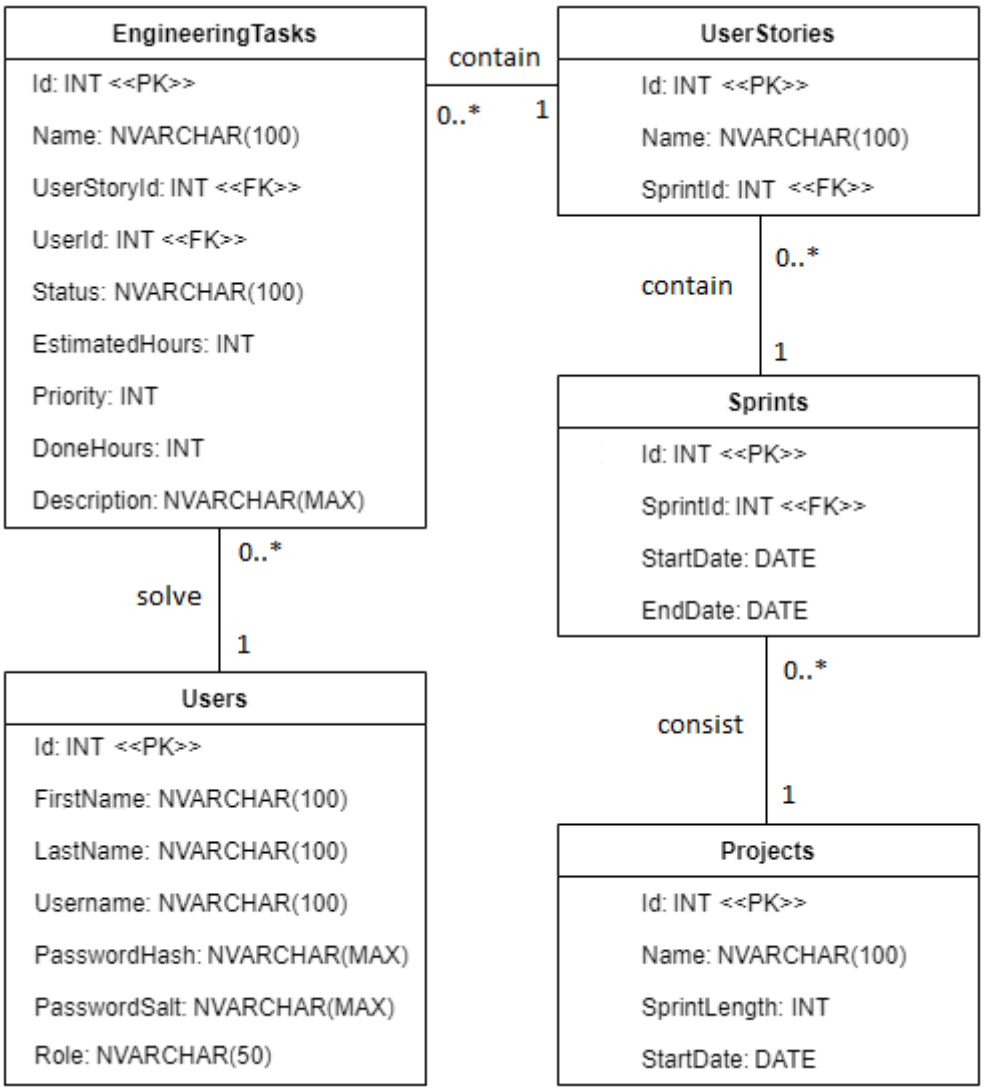
Tabulka 1 - Případy užití

4.2.2 Databázový model

Databáze je navržena tak, aby odpovídala analýze požadavků na aplikaci. Tedy kopíruje entity z reálného světa, které se v systému nachází. Vše je zobrazeno pomocí Entity Relationship diagramu, který je uveden na obrázku níže. Při následné implementaci bude databáze vycházet přímo z tohoto modelu.

Entity, které se nachází v diagramu jsou:

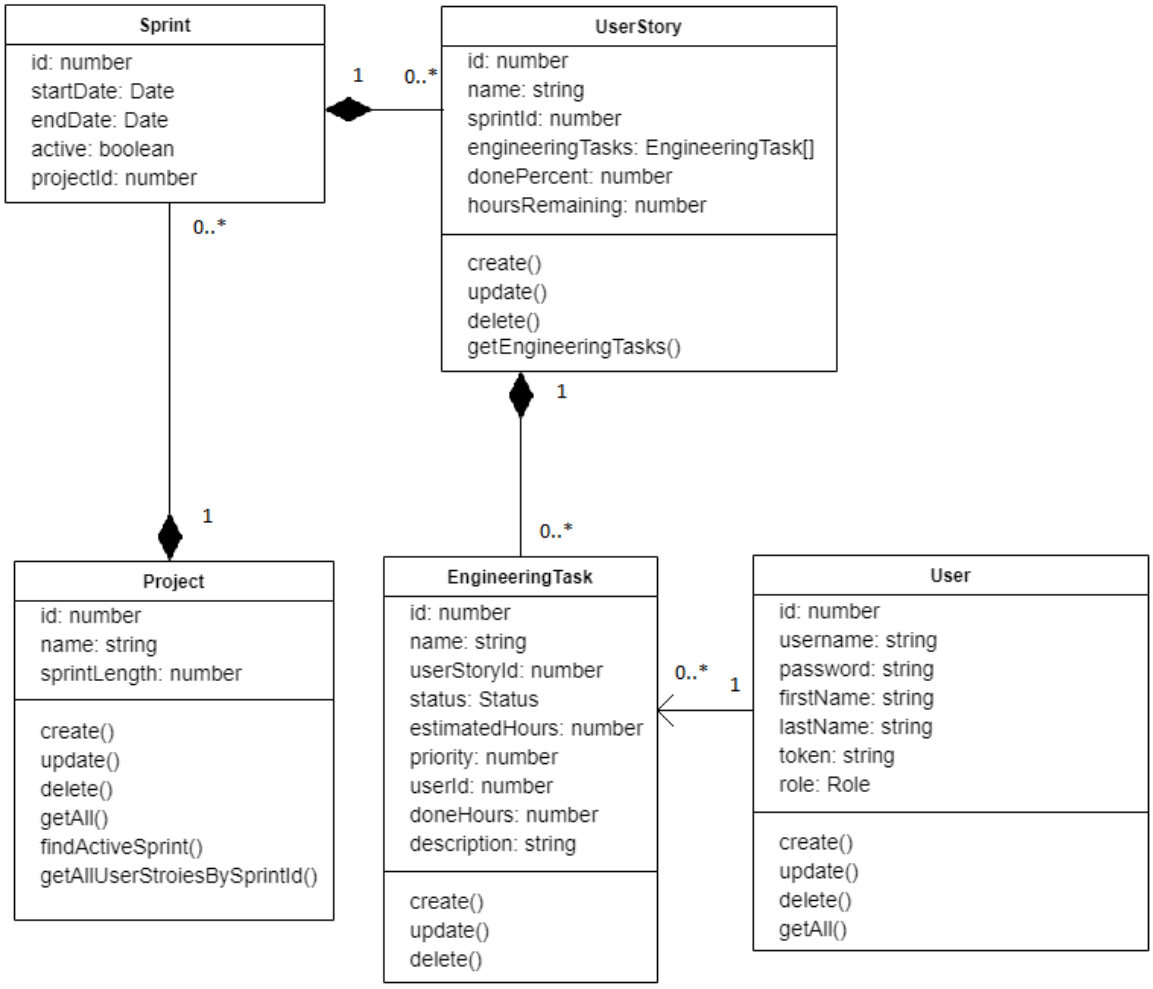
- Users – uživatelé, kteří aplikaci používají. Jedním z důležitých atributů je jejich role, která určuje, do kterých sekcí v aplikaci mají přístup.
- EngineeringTasks – úkol, který řeší člen vývojového týmu.
- UserStories – jsou to větší celky, které sdružují úkoly (Engineering Tasks). Každé User Story je součástí nějakého sprintu.
- Sprints – časově omezený úsek, který je součástí projektu. Všechny sprinty nacházející se ve stejném projektu mají shodnou délku.
- Projects – samostatné projekty, které jsou řešeny. Každý projekt má stanovenou délku na základě počtu sprintů.



Obrázek 6 - Datový model

4.2.3 Diagram tříd

Model byl navržen podle analýzy požadavků tak, aby objekty v něm se vyskytující reprezentovaly co nejlépe skutečné objekty a jejich chování.



Obrázek 7 – Diagram tříd

4.3 Použité technologie

S ohledem na analýzu veškerých požadavků na aplikaci byly vybrány technologie, které budou použity při následné implementaci. Jedná se jak o použité programovací jazyky, tak o Systém Řízení Báze Dat a také o vybraný webový server.

4.3.1 Programovací jazyk

Frontendová část aplikace byla tvořena pomocí frameworku Angular, který využívá jazyk TypeScript. Nejnovější verze je Angular 9, která vyšla v únoru tohoto roku. Tato část komunikuje se serverovým rozhraním API pomocí architektury REST. Backendová část aplikace je psána pomocí ASP.NET Core 3.1 s C#.

4.3.2 Systém řízení báze dat

Lokálně při vývoji aplikace byla použita SQL databáze od společnosti Microsoft. Přesněji se jedná o vyspělý databázový systém, který je vyvíjen již od roku 1992. Nejnovější verzí je MS SQL server 2019, která je dostupná od konce roku 2019 a při vývoji s ní bylo pracováno. Pro správu a vytváření databáze bylo použito Microsoft SQL Server Management Studio. Pro nasazení databáze a její ostrý provoz byla použita služba Azure SQL Database (škálovatelná cloudová databázová služba).

4.3.3 Webový server

Vybrané programovací jazyky jsou multiplatformní a aplikaci lze nasadit jak na operačním systému Linux s Apache serverem, tak i na kombinaci operačního systému Windows s IIS serverem. Přestože webový server Apache má výhodu, že je kompatibilní se všemi hlavními operačními systémy, byl k publikování aplikace vybrán IIS server od Microsoftu. Jeden z hlavních důvodů je ten, že pro nasazení aplikace byla vybrána služba Microsoft Azure.

4.4 Implementace

Po důkladné analýze vyvíjeného softwaru a jeho modelovém návrhu je dalším krokem k jeho dokončení implementace. Veškerý vývoj softwaru probíhal na notebooku s operačním systémem od společnosti Microsoft – Windows 10 Home. Jelikož aplikace je rozdělena na dvě části, a to frontendovou a backendovou, byly obě části vytvářené ve vlastním vývojovém prostředí. Frontendová část, která byla napsána pomocí frameworku Angular byla vyvíjena ve Visual Studio Code. Backendová část, která byla napsána pomocí ASP.NET byla vyvíjena ve vývojovém prostředí Visual Studio 2019.

4.4.1 Verzovací systém

Během vývoje je třeba aplikaci zálohovat a držet informace o jednotlivých verzích. Pro tento účel byl využit verzovací systém git, který byl v roce 2005 stvořen Linusem Torvaldsem. V současnosti se jedná o nejpoužívanější verzovací systém, který používají například společnosti Microsoft nebo Google. Aplikace byla během vývoje nahrávána na veřejný repozitář pomocí webové služby GitHub, který lze nalézt na internetové adrese https://github.com/KajuskaCZ/WebApplicationAG.

4.4.2 Nasazení

Pro nasazení aplikace byla vybrána služba Microsoft Azure. Jedná se o cloudovou platformu, která se využívá k vytváření, hostování a škálování webových aplikací. Pro vytvoření SQL databáze byla použita služba Azure SQL Database. Přístup k webovému serveru IIS byl zprostředkován pomocí virtuálního počítače, který byl ve službě Microsoft Azure vytvořen. K virtuálnímu počítači se přistupuje pomocí vzdálené plochy.

Aplikace byla během své implementace průběžně nahrávána na webový server, kde byla testována její funkčnost.

4.4.3 Struktura aplikace

Hlavní části bakcendové aplikace jsou třídy s názvy controller, service a model. Třídy controller se starají o příjem požadavků z frontendu. Podle svého názvu třída ví, který požadavek má zpracovat. Pokud se třída jmenuje `UserController`, potom cesta požadavku začínající `/user`, končí v této třídě. O samotnou logiku se starají třídy s názvem service. Poslední třídy models jsou pak přímo popisované objekty z reálného

světa. O připojení k databázi se stará třída `DataContext`, která dědí ze zavedené třídy `DbContext`, která je součástí `System.Data.Entity`.

Protože je frontendová část psaná ve frameworku Angular, je tvořena zejména pomocí komponent. Komponenta se skládá z TypeScriptového souboru, kde je psána její logika a dále z HTML a CSS souborů, které se starají o její vizuální stránku. Pro komunikaci s backendem se používají speciální třídy s názvem service. Jako příklad metody komunikující s backendovou částí je níže uvedena metoda pro získání určitého uživatele pomocí jeho identifikátoru. Metoda je součástí třídy `UserService`:

```
getById(id: number) {
  return this.http.get<User>(`${environment.apiUrl}/user/${id}`);
}
```

4.4.4 Autentizace

Slouží k jednoznačnému určení uživatele, který k systému přistupuje, cílem tedy je zjištění, zda daný uživatel je ten, za kterého se vydává. Autentizace patří mezi bezpečnostní opatření a zajišťuje ochranu před falešnou identitou (totožností). Způsoby autentizace se mohou rozdělit zhruba do tří základních skupin, a to na autentizaci dle toho, co uživatel má (identifikační karta, platební karta, klíč), autentizaci dle toho, co uživatel zná (PIN, heslo) a autentizaci dle toho, čím uživatel je (biometrické údaje). Potom podle míry důvěry se mohou způsoby autentizace kombinovat a místo jednofaktorové autentizace lze použít dvou faktorovou a tří faktorovou autentizaci.

Ve vytvářené webové aplikaci je autentizace implementována pomocí JWT (JSON Web Token). Jedná se o otevřený standard, který definuje kompaktní a nezávislý způsob pro bezpečný přenos informací mezi stranami pomocí JSON objektu. Přenášené informace jsou digitálně podepsány pomocí tajného hesla (HMAC) a proto jim lze důvěřovat a lze je lehce ověřit.

Token je vygenerován v backendové části aplikace pomocí třídy `JwtSecurityTokenHandler`. Jak lze vidět na ukázce kódu níže, token je nejprve digitálně podepsán pomocí tajného klíče, který je uložen v souboru `appSettings.json`. Poté je uložen do proměnné s názvem `token`, která je dále převedena na řetězec znaků a v tomto tvaru poslána na frontendovou část aplikace.

```
var tokenHandler = new JwtSecurityTokenHandler();
var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
var tokenDescriptor = new SecurityTokenDescriptor
{
  Subject = new ClaimsIdentity(new Claim[]
  {
    new Claim(ClaimTypes.Name, user.Id.ToString()),
    new Claim(ClaimTypes.Role, user.Role)
  }),
  Expires = DateTime.UtcNow.AddDays(7),
  SigningCredentials = new SigningCredentials(
    new SymmetricSecurityKey(key),
    SecurityAlgorithms.HmacSha256Signature)
};
var token = tokenHandler.CreateToken(tokenDescriptor);
var tokenString = tokenHandler.WriteToken(token);
```

Frontendová část aplikace se stará o zobrazování formulářů pro registraci a přihlášení uživatele. Při registraci uživatele je na backendovou část mimo jiné posláno i heslo. Heslo je nejprve osoleno, což znamená, že se při jeho hashování na vstup přidá ještě nějaký další řetězec, takzvaná sůl. Sůl je pro každého uživatele jiná, proto výsledný hash je i pro stejná hesla pokaždé jiný. Sůl nemusí být tajná, často jde o veřejnou hodnotu. Ve vyvíjené aplikaci je sůl náhodně generována a následně uložena do databáze, aby mohla být dále používána při opětovném přihlašování právě vytvořeného uživatele.

Při registraci uživatele se do databáze ukládají v souvislosti s jeho heslem dvě hodnoty, a to sůl a zahashované heslo. V následném vloženém kódu je vidět, jak tyto dvě hodnoty vznikají. Nejprve je použita funkce `HMACSHA512`, která vypočte kód HMAC (Hashed-based Message Authentication Code) pomocí funkce `SHA512`. Sůl vkládaná do databáze se rovná klíči vypočtené hodnotě (`hmac.Key`). Následný hash hesla je vytvořen pomocí funkce `ComputeHash`, která použije vygenerovanou sůl.

```
using (var hmac = new System.Security.Cryptography.HMACSHA512()) {
  passwordSalt = hmac.Key;
  passwordHash = hmac.ComputeHash(
    System.Text.Encoding.UTF8.GetBytes(password));
}
```

Při přihlašování již registrovaného uživatele jsou uložené hodnoty sůl a hash použity ke správné verifikaci zadávaného hesla. Následná ukázka kódu je vyjmuta z funkce, která zadávané heslo ověřuje. Nejprve je pomocí uložené soli vypočítán kód HMAC. Tento kód je poté použit k vypočítání hashe ze zadaného hesla pomocí funkce

`ComputeHash`. Když je znám hash vypočítaný ze zadaného hesla a hash, který je uložen v databázi, oba se spolu znak po znaku porovnají. Pokud se nějaký znak vypočítaného a uloženého hashe liší, je navrácena hodnota false, což znamená že heslo nebylo zadáno správně. V opačném případě proběhne přihlášení uživatele do aplikace.

```
using (var hmac =
  new System.Security.Cryptography.HMACSHA512(storedSalt))
{
  var computedHash =
   hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
  for (int i = 0; i < computedHash.Length; i++) {
    if (computedHash[i] != storedHash[i])
      return false;
  }
}
```

4.4.5 Autorizace

Pod pojmem autorizace se rozumí proces ověření přístupových oprávnění uživatele vstupujícího do aplikace. Tento proces ve většině případů navazuje na proces autentizace. Podstatou autorizace je ověřit, že daný uživatel má oprávnění příslušnou akci provést.

Každý uživatel má přiřazenou roli, ze které jsou odvozená jeho přístupová práva k různým částem aplikace. Role, které mohou být uživateli přiřazeny jsou:

- Administrátor
- Product Owner
- Scrum Master
- Člen vývojového týmu, označen jako User

Ve frontendové části aplikace, která je tvořena ve frameworku Angular, jsou některé části přístupné pouze pro určité role. Docílit toho, aby se tyto vybrané části zobrazovali pouze pokud je k nim umožněn přístup lze snadno pomocí modulu, který se nazývá `AppRoutingModule`. V tomto modulu jsou vypsány veškeré části nebo stránky aplikace a cesty k nim vedoucí. Jako příklad lze uvést stránku, která je určena pro správu uživatelů. K této stránce mají přístup pouze administrátoři a docíleno je toho pomocí proměnné `roles`, do které je přiřazena pouze role administrátora. Lze vidět na ukázce kódu níže.

```
{ path: 'userManager', component: UserManagerComponent,
  canActivate: [AuthGuardService], data: { roles: [Role.Admin] } }
```

4.5 Testování

Zjednodušeně lze říct, že testování je odhalování chyb v softwaru. Přesněji to můžeme definovat jako výzkum kvality softwaru, při kterém se ověřuje, jestli reálné vlastnosti softwaru odpovídají vlastnostem požadovaným a očekávaným. Vlastnosti, které lze testovat se mohou dělit ještě na vlastnosti funkční a nefunkční. Kde funkční vlastnosti jsou takové, které ověřují, že software správně vykonává úkoly, pro které byl stvořen a nefunkční vlastnosti jsou ty, které se týkají instalace, výkonu, přístupnosti a zabezpečení software.

4.5.1 Jednotkové testy

Tyto testy lze také nazvat jako unit testy. Slouží k testování tříd, přesněji k testování jejich metod. Testy předávají metodám různé vstupy a zkouší, zda jsou jejich výstupy korektní. Jsou psány vždy na základě návrhu softwaru a nikoli implementace. Což znamená, že jsou psány na základě očekávané funkčnosti.

Výhody unit testů v průběhu psaní software lze spatřit při refaktorizaci a při přidávání nových vylepšení, kdy díky správně proběhlým testům je jasné, že se software stále chová podle očekávání. Testy také mohou dopomoci k lepšímu návrhu software.

Ve vybraném frameworku Angular pro frontendovou část aplikace jsou jednotkové testy psány pomocí nástrojů Jasmine a Karma, které jsou automaticky nakonfigurovány při vytváření aplikace pomocí *Angular CLI*.

- Angular CLI – je nástroj příkazového řádku, který je používán pro správu Angular aplikací a lze ho jednoduše nainstalovat pomocí nástroje pro manipulaci s balíčky npm a to příkazem `npm install -g @angular/cli`, kde parametr `-g` znamená nainstalování balíčků globálně.
- Jasmine – je javasciptový framework pro vytváření testů, který obsahuje mnoho užitečných funkcí pro psaní různých testů, mezi které patří it (deklaruje určitý test), describe (tělo pro sadu testů) a expect (očekává nějakou hodnotu).
- Karma – je nástroj, který umožňuje testy spouštět. Výsledky testů se zobrazují ve webovém prohlížeči a při každé úpravě kódu jsou automaticky znovu spuštěny. Výhoda nástroje také spočívá v tom, že lze jednoduše ovládat pomocí příkazové řádky.

Framework Angular pro komunikaci s backendovým API používá takzvané service, což jsou specifické třídy. Metody v těchto třídách je také nutné otestovat, kvůli zjištění, zda navrací, vytváří, aktualizují a mažou správná data.

Na ukázku je zde vybrán test, jenž testuje metodu s názvem `getById`, která je obsažena ve třídě `EngineeringTaskService`. Metoda navrací určitý Enginnering Task, který je jednoznačně identifikován. Jelikož se při testování nepracuje s reálnými daty, je nutné Engineering Task, který chceme, aby byl navrácen, nejprve takto vytvořit:

```
const engineeringTask = [
  {
     "id": 1, "name": 'ET1', "userStoryId": 1, "status": 'New',
     "estimatedHours": 30, "priority": 3, "userId": 1,
     "doneHours": 0, "description": 'desc'
  },
];
```

Když je potřebný Enginnering Task vytvořen, pokračuje se psaním samotného testu. Nejprve jeho deklarací pomocí funkce it, kde první parametr je popisek testu, který se bude zobrazovat ve výpisech. Druhý parametr je funkce obsahující metodu, která bude testována. Také je potřeba použít funkci expect, jíž se předloží data, která jsou očekávaným návratem testované metody.

```
it('should get the correct engineering Task', () => {
  engineeringTaskService.getById(1).subscribe((data: any) => {
    expect(data).toBe(engineeringTask);
  });

  const req =
    httpMock.expectOne(`${environment.apiUrl}/engineeringtask/1`);
  expect(req.request.method).toBe('GET');
  req.flush(engineeringTask);
});
```

Pokud jsou testy hotové, spustí se pomocí příkazu `npm test`. Tím se otevře okno prohlížeče, kde je možné probíhající testy vidět. Po skončení se zobrazí výsledek. V tomto konkrétním případě všechny testy napsané pro třídu EngineeringTaskService, mezi kterými je i ukázkový test, proběhly bez chyb:

- should put the correct data
- should delete the correct data
- should get the correct Engineering Task
- getAllByUserStory should return data

4.5.2 Testování zobrazení

Protože existuje více klientů (webových prohlížečů), které jsou schopné aplikaci spustit, je nutné otestovat její správné zobrazení. Testování proběhlo na majoritních prohlížečích, které jsou dostupné pro počítače, a proběhlo úspěšně. Operační systém nacházející se na testovacím počítači byl Windows 10. Testované prohlížeče jsou:

- Google Chrome 80
- Mozilla Firefox 70
- Microsoft Edge 44

4.5.3 Testovací scénáře

Každý testovací scénář obnáší stanovení cíle, kterého má uživatel dosáhnout. Pokud jakýkoli krok při zkoušení testovacího scénáře se nepodaří provést, je test vyhodnocen jako neúspěšný. Výsledky všech následujících testovacích scénářů dopadly pozitivně.

- Testovací scénář 1: Přihlášení uživatele – Proces přihlášení zaregistrovaného uživatele. Uživatel zadal své přihlašovací údaje. Výsledkem je úspěšné přihlášení uživatele a jeho přístup do aplikace.

Předpokladem pro následující testovací scénáře je, že uživatel je úspěšně přihlášen v aplikaci.

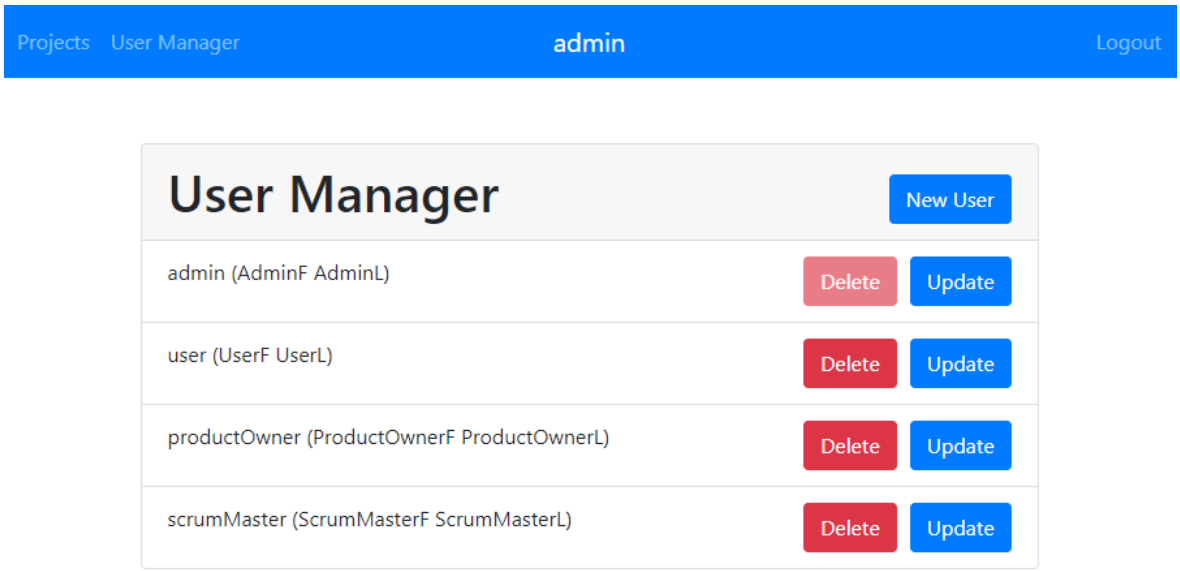
- Testovací scénář 2: Vytvoření nového projektu – Proces vytvoření nového projektu uživatelem, který má roli administrátora nebo Product Ownera. Výsledkem je nový projekt, se kterým se dá pracovat.
- Testovací scénář 3: Upravení statusu úkolu (Engineering Task) – Proces upravení statusu úkolu na hotový (done), když se uživatel nachází na stránce detailu projektu. Graficky se změní hotová část User Story, ve kterém se úkol nachází.

4.6 Uživatelská příručka

Pro usnadnění práce s aplikací slouží uživatelská příručka. V jejích kapitolách bude popsáno, jak aplikaci obsluhovat, aby bylo dosaženo žádoucích výsledků.

4.6.1 Práce s uživateli

Aplikace není volně přístupná a nové uživatele může registrovat pouze administrátor. Základní administrátor se může přihlásit pod přihlašovacími údaji admin/admin. Pro správu uživatelů se v horním menu vybere stránka User Manager, která je viditelná pouze pro administrátora. Na obrázku níže je vidět, jak User Manager může vypadat. Je zde seznam uživatelů a akce, které s nimi lze provádět. Za povšimnutí stojí, že administrátor nemůže smazat sám sebe, protože je žádoucí, aby aplikace nejméně jednoho měla.



Obrázek 8 – Správce uživatelů

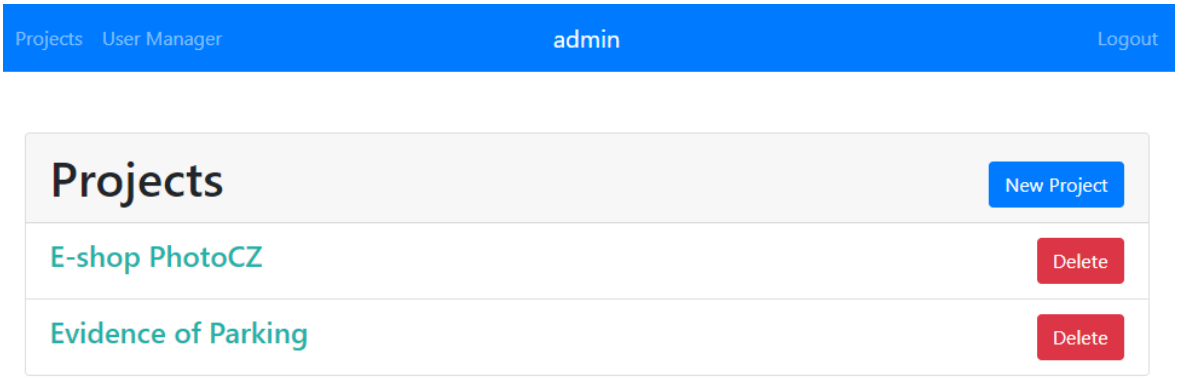
K registrování nových uživatelů slouží tlačítko New User. Zobrazí se jednoduchý formulář pro vytvoření nového uživatele, který je ke shlédnutí jako příloha 2 této práce. Všechna políčka k vyplnění jsou povinná, tedy pro založení nového uživatele je třeba uvést jeho jméno, příjmení, uživatelské jméno, roli, základní heslo a potvrdit heslo. Vytvořený uživatel lze poté upravit, anebo smazat.

4.6.2 Projekty

Pro správu jednotlivých projektů se vybere v horním levém menu stránka Projects. Zde může uživatel, kterému byla přiřazena role administrátora nebo Product Ownera jednotlivé

projekty spravovat. Stránka projektů je znázorněna na obrázku níže. Je zde vidět seznam projektů a akce, které s nimi lze provádět.

Pro vytvoření nového projektu slouží tlačítko New Project. Následně zobrazený formulář je ke shlédnutí jako příloha 3. Všechna jeho políčka jsou povinná a musí se tedy vyplnit jméno projektu, počet sprintů, ze kterých se projekt bude skládat, délka jednoho sprintu a datum začátku projektu.

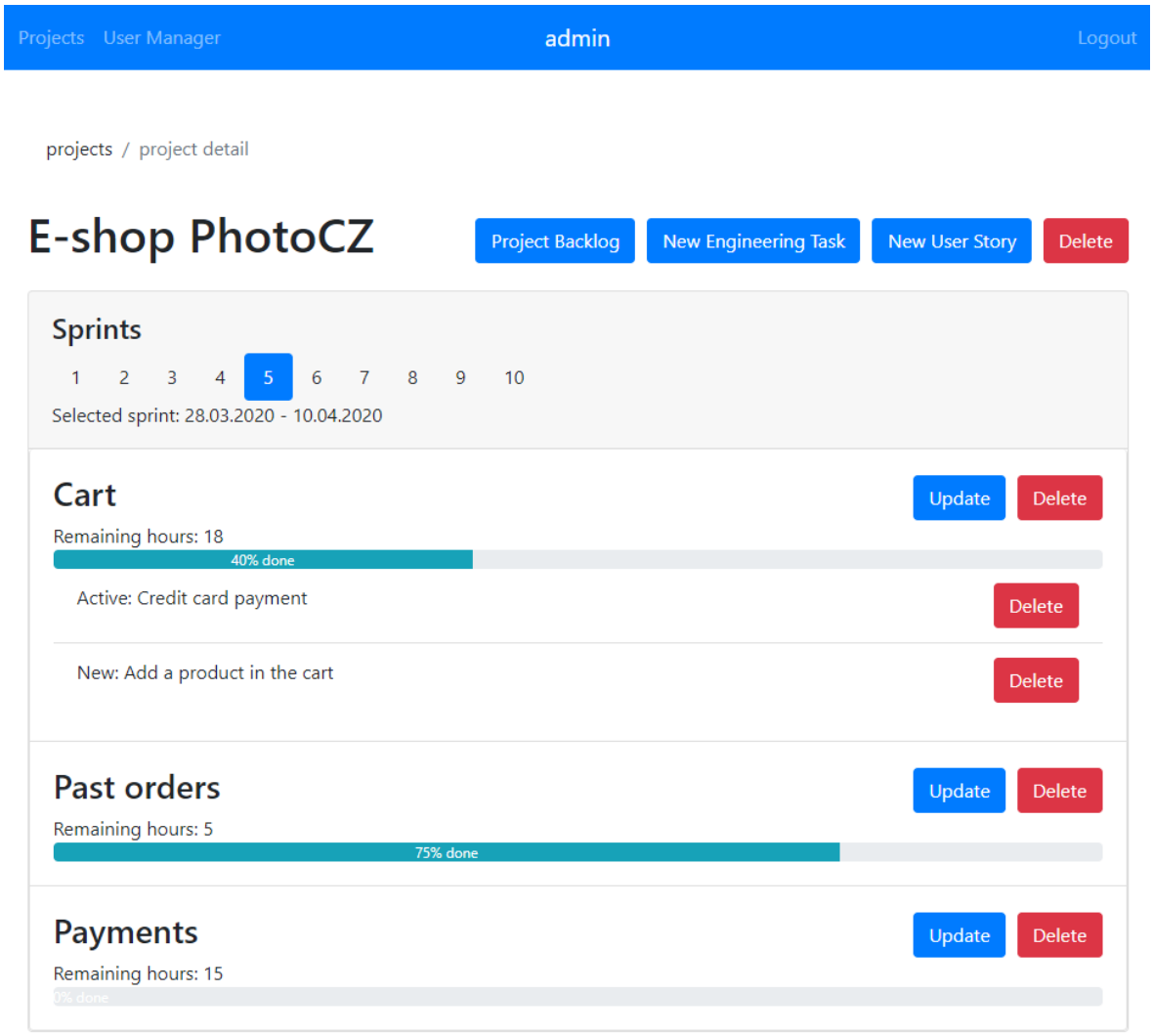


Obrázek 9 – Projekty

Detail projektu se zobrazí po kliknutí na něj a jak může vypadat je znázorněno na následujícím obrázku. Akce, které lze na projektu provádět jsou vidět v horní části a jsou udělané jako tlačítka. Po kliknutí na tlačítko Project Backlog se zobrazí detail projetku, kde je seznam všech User Stories bez ohledu na to, do kterého sprintu patří. Tlačítko Delete slouží pro smazání projektu. Tlačítka New Engineering Task a New User Story budou popsána dále.

Struktura detailu projektu je udělána tak, že jsou v horní části vidět jednotlivé sprinty a u právě vybraného sprintu jsou vidět data jeho začátku a konce. Níže jsou pak vylistována všechna User Stories, která se nacházejí v aktuálně vybraném sprintu.

Každé User Story má pod sebe přiřazené úkoly. Každý úkol má nastaveno, kolik je třeba hodin práce k jeho dokončení. Součet potřebných hodin k dokončení všech úkolů v jednom User Story se graficky promítá v podobě ukazatele průběhu. Lze na něm přehledně pozorovat, kolik je potřeba hodin k dokončení celého User Story.



Obrázek 10 - Detail projektu

Pro vytvoření nového User Story slouží tlačítko New User Story. Po kliknutí se otevře formulář, který je přiložen jako příloha 4. Obě políčka formuláře jsou povinná. K vyplnění je třeba zadat název a sprint, ve kterém je User Story chtěno. Jako výchozí hodnota pro sprint je nastaven právě vybraný sprint.

Engineering Tasks neboli úkoly jsou zobrazeny pod příslušným User Story. Viditelný je název, jemuž předchází aktuální status úkolu. Pokud se na Engineering Task klikne, otevře se jeho editovatelný detail. Nejprve je ale důležité sdělit, jak se úkoly vytváří. K tomu slouží formulář, který se zobrazí po kliknutí na tlačítko New Engineering Task a je k vidění jako příloha 5. Všechna políčka jsou povinná a musí se vyplnit název, User Story, do kterého úkol patří, odhadovaný počet hodin, priorita a popis. Automaticky je nově vytvořenému Engineeringu Tasku nastaven status na nový (New), počet

odpracovaných hodin (Done Hours) na nula a jako přiřazený uživatel (User), ten, kdo úkol zakládal.

Při editaci Engineeringu tasku je zobrazen formulář, který už políčka jako status, počet odpracovaných hodin a přiřazený uživatel obsahuje. Editační formulář je vložen jako příloha 6 této práce. Pokud je úkol označen jako hotov, tedy status je Done, musí se počet odhadovaných hodin a počet odpracovaných hodin rovnat. Také nelze nastavit počet odpracovaných hodin větší než počet odhadovaných hodin.

5 Výsledky a diskuse

Důsledná analýza a rozbor funkčních požadavků softwaru je základem jakékoli jeho realizace. Nejlepší možností, jak projít celým procesem vývoje je, si rozdělit vše na menší části, se kterými se snadněji pracuje. V průběhu práce je k dispozici celý návrh a lze ho jednoduše upravovat v malých detailech. Doporučené také je nezapomínat na důležité části vývoje a co nejpřesněji se držet plánu postupně od analýzy až po testování.

Při analýze webové aplikace bylo zjištěno, že nejlepším řešením bude ve frontendové části použít framework Angular, protože se jedná o již zavedený framework a jeho nejnovější verze přináší nezanedbatelné novinky. Backendová část je vytvořena pomocí ASP.NET s jazykem C#. Pro jejich vzájemnou komunikaci byla vybrána architektura REST, která je nyní nejpoužívanější při návrhu moderních webových aplikací. Samotnou výhodou webových aplikací je, že jsou multiplatformní, a tudíž se neváží na určitý operační systém. Tímto lze také snadno pokrýt co nejvíce uživatelů.

Webová aplikace pro podporu agilního programování uživatelům poskytuje zefektivnění práce a lepší orientaci v přidělování a uskutečňování jednotlivých úkolů. Manažerům se lépe práce sleduje a ulehčuje jejich budoucí rozhodování.

6 Závěr

V teoretické části diplomové práce byl čtenář seznámen s problematikou týkající se použití různých metodik při vývoji software, dozvěděl se, jaké jsou rozdíly mezi tradičními a agilními metodikami. S některými vybranými metodikami byl seznámen podrobněji. Dále bylo popsáno, jak slouží internetové technologie, jak se používají webové služby a jak spolu části webové aplikace mohou komunikovat. Na závěr teoretické části byly vypsány standardy UML a jednotlivé diagramy.

V praktické části byla čtenáři vysvětlena nezbytnost prvotní analýzy software při jeho vývoji a jak jsou důležité zadané funkční požadavky. Dále byl popsán návrh systému a následně samotná implementace. V závěru této kapitoly byla představena malá uživatelská příručka, která slouží pro lepší pochopení aplikace.

Výsledek, kterým je vytvořená webová aplikace ve framework Angular a ASP.NET s jazykem C#, byl veřejně nasazen.

7 Seznam použitých zdrojů

BOULTON, Mark. *A Practical Guide to Designing for the Web*. Penarth: Mark Boulton Design Ltd, 2009. ISBN 978-0-9561740-7-9.

BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury.* Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.

BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.

DVOŘÁK, Drahoslav a Martin MAREČEK. *Project Portfolio Management.* Brno: Computer Press, 2017. ISBN 978-80-251-4893-8.

ELLIOTT, Eric. *Programming JavaScript applications*. Sebastopol: O'Reilly, 2014. ISBN 978-1-491-95029-6.

IIS vs Apache. Up Guard [online]. 2019 [cit. 2020-03-08]. Dostupné z: https://www.upguard.com/articles/iis-apache

Introduction To OMG's Unified Modeling Language [online]. 2005 [cit. 2020-03-08]. Dostupné z: http://uml.org/what-is-uml.htm

Introduction: What is Vue.js. VueJS [online]. 2019 [cit. 2020-03-08]. Dostupné z: https://vuejs.org/v2/guide/

KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru.* Brno: Computer Press, 2004. ISBN 80-251-0342-0.

KOSEK, Jiří. *PHP – tvorba interaktivních internetových aplikací: podrobný průvodce.* Praha: Grada, 1998. Průvodce (Grada). ISBN 80-716-9373-1.

MÁCA, Jindřich. *Úvod do Angular*. IT Network [online]. 2019 [cit. 2020-03-08]. Dostupné z: https://www.itnetwork.cz/javascript/angular/zaklady/uvod-do-angular-frameworku

MÁCA, Jindřich. *Úvod do React.* IT network [online]. 2019 [cit. 2020-03-08]. Dostupné z: https://www.itnetwork.cz/javascript/react/uvod-do-react

Manifesto for Agile Software Development [online]. 2001 [cit. 2020-03-08]. Dostupné z: http://agilemanifesto.org/

MASSÉ, Mark. *REST API Design Rulebook.* Sebastopol: OŔeilly, 2012. ISBN: 978-1-449-31050-9.

MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.

PECINOVSKÝ, Rudolf. *Java 7: učebnice objektové architektury pro začátečníky.* Praha: Grada, 2012. Knihovna programátora (Grada). ISBN 978-80-247-3665-5.

PINKAS, Otakar. *Webové servery*. Ikaros [online]. 2011 [cit. 2020-03-08]. Dostupné z: https://ikaros.cz/webove-servery

REJNKOVÁ, Petra. *Příklady použití diagramů UML 2.0* [online]. 2009 [cit. 2020-03-08]. Dostupné z: http://uml.czweb.org

SADOVSKI. *Základy SQL* [online]. 2010 [cit. 2020-03-08]. Dostupné z: http://books.fs.vsb.cz/SQLReference/Sadovski/SQL-PRVN.HTM

SCHWABER, Ken a Jeff SUTHERLAND. *Průvodce Scrumem: Pravidla hry.* Scrum guides [online]. 2013 [cit. 2020-03-08]. Dostupné z: https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-CS.pdf

SOMMERVILLE, Ian. *Softwarové inženýrství.* Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

STELLMAN, Andrew a Jennifer GREENE. *Learning Agile.* Beijing: O'Reilly, 2014. ISBN 978-1-449-33192-4.

SUTHERLAND, Jeffrey Victor. *Scrum: the art of doing twice the work in half the time.* New York: Crown Business, 2014. ISBN 978-0-385-34645-0.

ŠILHAVÝ, Radek, Petr ŠILHAVÝ, Zdenka PROKOPOVÁ, Pavel POKORNÝ, Martin SYSEL, Miroslav MATÝSEK, Karel VLČEK a Libuše SVOBODOVÁ. *Vybrané aspekty návrhu webových informačních systémů. 2. vyd.* Vsetín: Scientific Press, 2013. ISBN 978-80-904741-3-0.

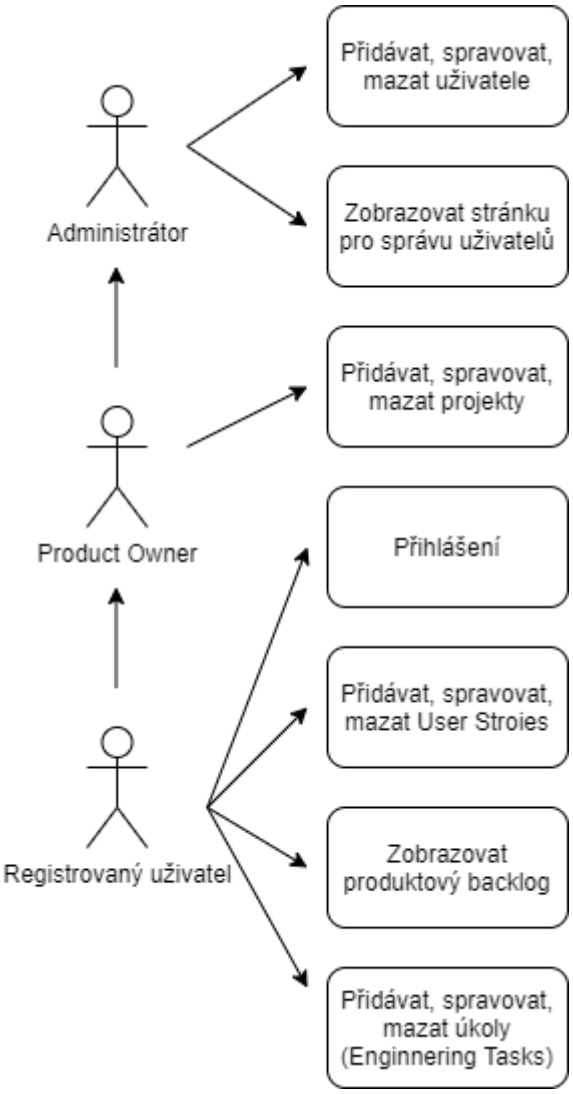
ZELINKA, Tomáš a Miroslav SVÍTEK. *Telekomunikační řešení pro informační systémy síťových odvětví.* Praha: Grada, 2009. Průvodce (Grada). ISBN 978-80-247-3232-9.

ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

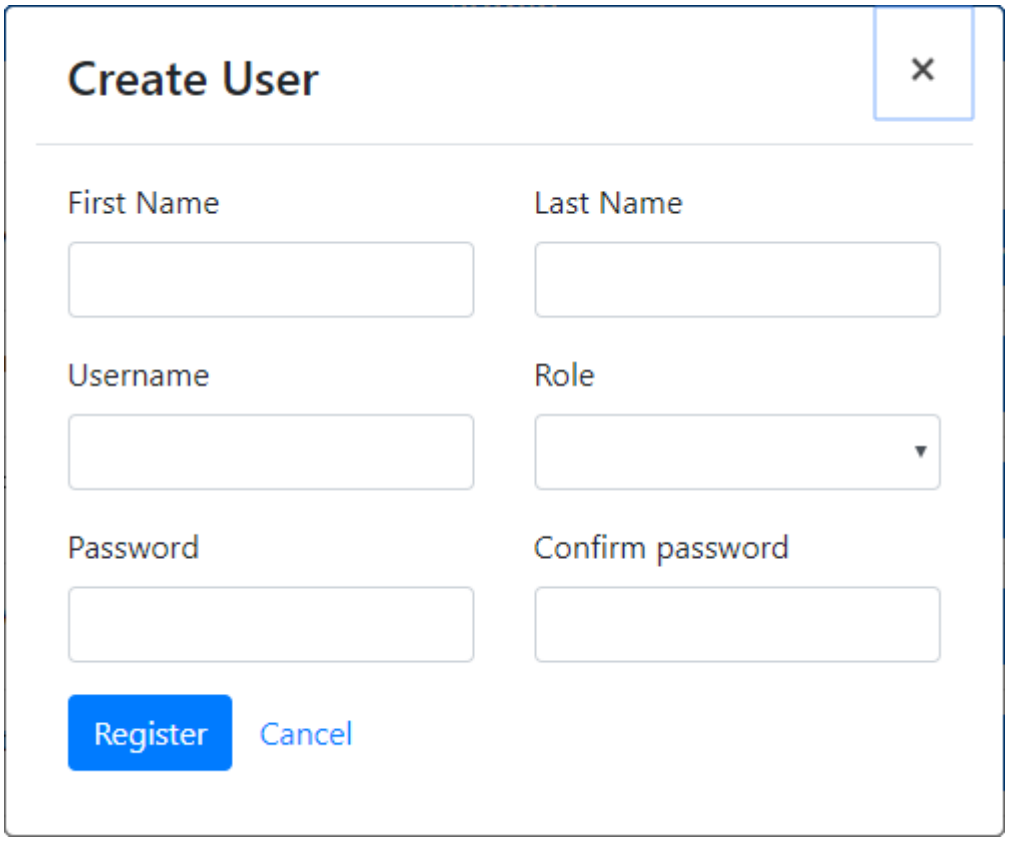
8 Přílohy

Příloha 1 – Diagram případů užití
Příloha 2 – Formulář pro vytvoření nového uživatele
Příloha 3 – Formulář pro vytvoření nového projektu
Příloha 4 – Formulář pro vytvoření nového User Story
Příloha 5 – Formulář pro vytvoření nového Engineering Tasku
Příloha 6 – Formulář pro editaci Engineering Tasku

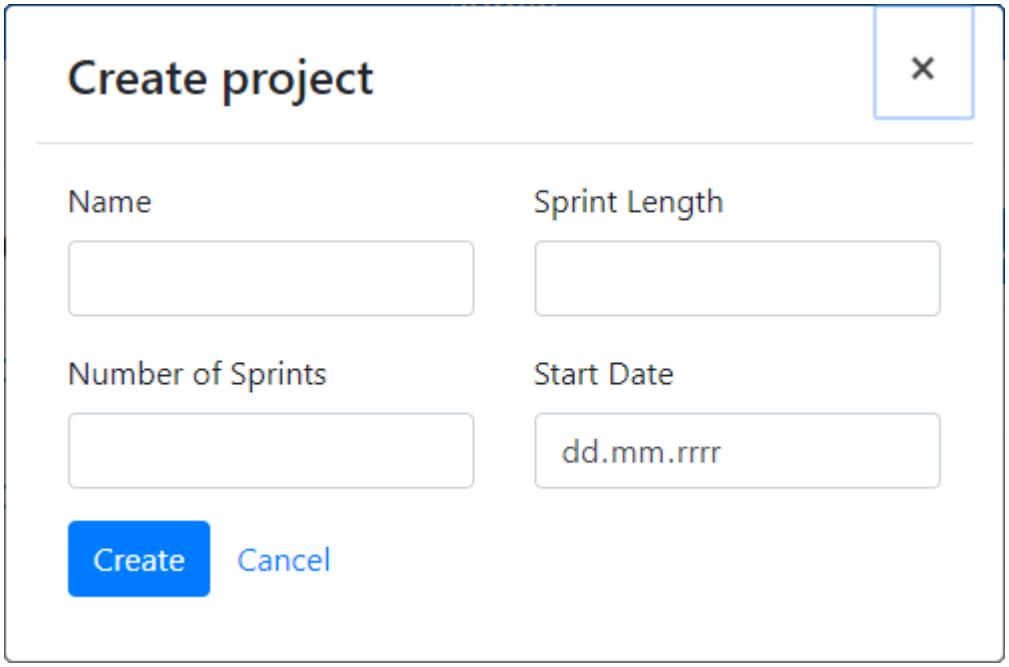
Příloha 1 – Diagram případů užití



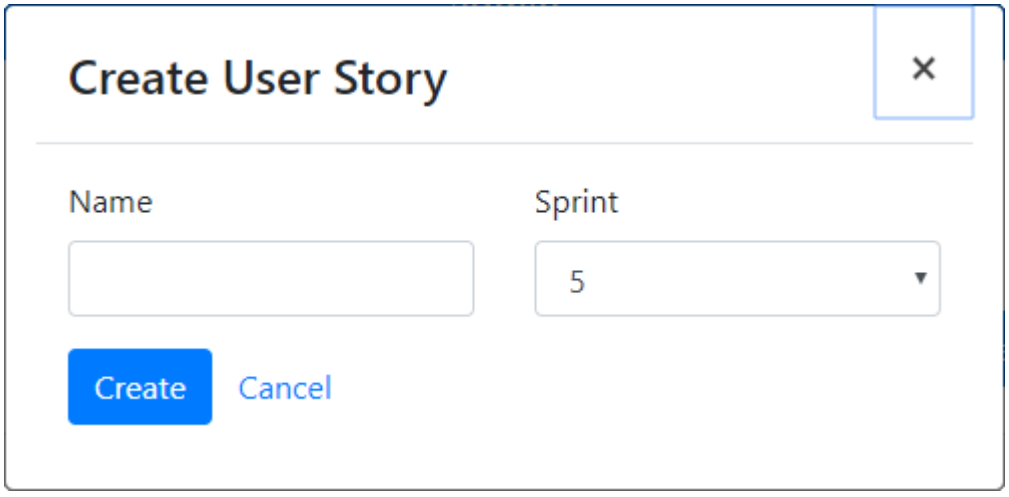
Příloha 2 – Formulář pro vytvoření nového uživatele



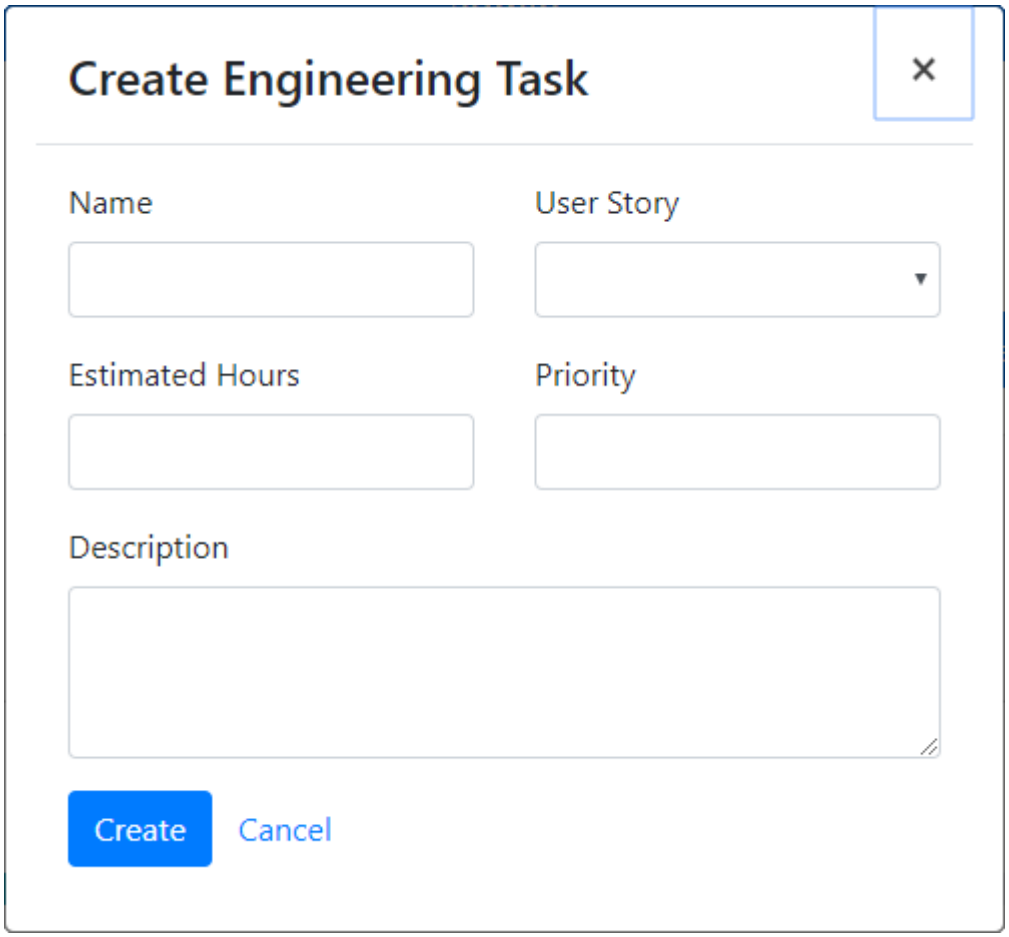
Příloha 3 – Formulář pro vytvoření nového projektu



Příloha 4 – Formulář pro vytvoření nového User Story



Příloha 5 – Formulář pro vytvoření nového Engineering Tasku



Příloha 6 – Formulář pro editaci Engineering Tasku