VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO PODPORU ŘÍZENÍ AGILNÍCH VÝVOJOVÝCH TÝMŮ

AGILE DEVELOPMENT TEAMS MANAGEMENT SUPPORT TOOL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ALBERT UCHYTIL**
AUTHOR

VEDOUCÍ PRÁCE — **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Uchytil Albert, Bc.**
Program: Informační technologie Obor: Management a informační technologie
Název: **Nástroj pro podporu řízení agilních vývojových týmů**
Agile Development Teams Management Support Tool
Kategorie: Softwarové inženýrství

Zadání:

1. Seznamte se s agilními metodami vývoje softwarových produktů.
2. Seznamte se se znalostními oblastmi managementu projektů dle aktuálního standardu PMI včetně jeho aktuálního rozšíření pro agilní vývoj aplikací.
3. Specifikujte požadavky na systém pro podporu managementu v softwarových projektech vyvíjených agilně. Systém navrhněte.
4. Zvolte vhodné vývojové prostředí a implementujte prototyp funkcí navrženého systému, vybraných po dohodě s vedoucí.
5. Na vzorku dat, vybraném po dohodě s vedoucí, ověřte funkčnost vytvořeného prototypu nástroje.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti jeho dalšího rozšíření.

Literatura:

- *A Guide To The Project Management Body Of Knowledge:* Sixth Edition, Project Management Institute, 2017. ISBN 978-1-62825-184-5.
- *Agile Practice Guide*: global standard of Project Management Institute, 2017. ISBN 978-1-62825-199-9.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kreslíková Jitka, doc. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 16. října 2018

Abstrakt

Tato práce řeší tvorbu prototypu nástroje pro řízení agilních vývojových týmů. Nejdříve se věnuje teoretickým základům pro budovaný nástroj. Dále se věnuje analýze požadavků na daný informační systém. Poté probírá výběr implementačních technologií, následovaný návrhem technického řešení včetně samotného architektonického návrhu daného informačního systému. V práci je probíráno i uživatelské testování a jeho zhodnocení. Tvorba systému byla inspirována i poslední verzí PMI standardu. V závěru je shrnuta práce na projektu a jsou nastíněny další možnosti rozšíření vytvořeného prototypu.

Abstract

This thesis describes an implementation of a support tool for an agile development team management. In the beginning, it describes a theoretical background for the tool. The thesis also focuses on the selection of the implementation technology, after that it focuses on designing a technical solution and provides an architectural design of the information system itself. It also describes a user testing process and reviews it afterwards. The system is inspired by the latest version of the PMI standard. The conclusion summarizes the work and discusses possible improvements of the created prototype.

Klíčová slova

Agilní metodiky, Informační systém, Projektové řízení

Keywords

Agile methodologies, Information system, Project Management

Citace

UCHYTIL, Albert. *Nástroj pro podporu řízení agilních vývojových týmů.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

Nástroj pro podporu řízení agilních vývojových týmů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Albert Uchytil
20. května 2019

Poděkování

Chtěl bych poděkovat své vedoucí diplomové práce paní doc. RNDr. Jitce Kreslíkové, CSc. za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych chtěl poděkovat Jakubu Vostrčilovi a Pavlu Tobiášovi za poskytnuté rady, které mi pomohly při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Dle každoročních zpráv *State of Agile* vydávaných společností *VersionOne* dochází k zvětšování počtu společností využívajících agilní metodiky. Což vytváří prostor pro softwarovou podporu příslušných procesů. Podle zprávy z roku 2017, kde odpovídalo 1492 respondentů pohybujících se v softwarovém vývoji po celém světě, označilo 46% respondentů, že mezi nástroji používanými k agilnímu řízení využívají *MS Excel*. Jedná se o velice silný nástroj, ale na druhou stranu je obecný, což přináší větší nároky na jeho obsluhu. Proto jsem se rozhodl v rámci této práce vytvořit snadno ovladatelný nástroj, který poskytne menším a středním týmům přínosnou softwarovou podporu.

Tato práce se zabývá tvorbou nástroje pro podporu řízení agilních vývojových týmů. V rámci práce jsou v kapitole 2 představeny teoretické základy čerpající z *A Guide to the PROJECT MANAGEMENT BODY OF KNOWLEDGE (Sixth edition)* včetně jejího agilního rozšíření *AGILE PRACTICE GUIDE* publikované organizací *Project Management Insititute* (dále jen PMI) (převzato z [48]). Dále je v kapitole 3 provedena analýza a shrnutí požadavků na tento nástroj (částečně převzato z [48]). Následně je v kapitole 4 popsán výběr a analýza vhodných technologií pro tvorbu realizovaného systému. Na základě definovaných požadavků a zvolených technologií je proveden návrh systému, tento návrh včetně implementace je popsán v kapitole 5. Testování a průběžné ověřování konceptu je popsáno v rámci kapitoly 6. V závěru jsou shrnuty dosažené výsledky a možnosti dalšího rozšíření.

Kapitola 2

Teoretický úvod

Dle definice PMI je projekt časově ohraničeným usilím, vyvinutým za účelem vytvoření jedinečného výrobku nebo služby [4, s. 711].

Můžeme si také uvést definici podle standardu *ISO 10006*, tedy *"Projekt je jedinečný proces sestávající z řady koordinovaných a řízených činností s daty zahájení a ukončení, prováděný pro dosažení cíle, který vyhovuje specifickým požadavkům, včetně omezení daných časem, náklady a zdroji."* [32]

2.1 Etapy projektu

Projekt prochází v rámci svého životního cyklu několika etapami. Dle *Project Management Book of Knowledge* (dále jen PMBOK) jde o následující:

- Inicializace projektu
- Plánování
- Realizace
- Monitorování a kontrola
- Ukončení projektu

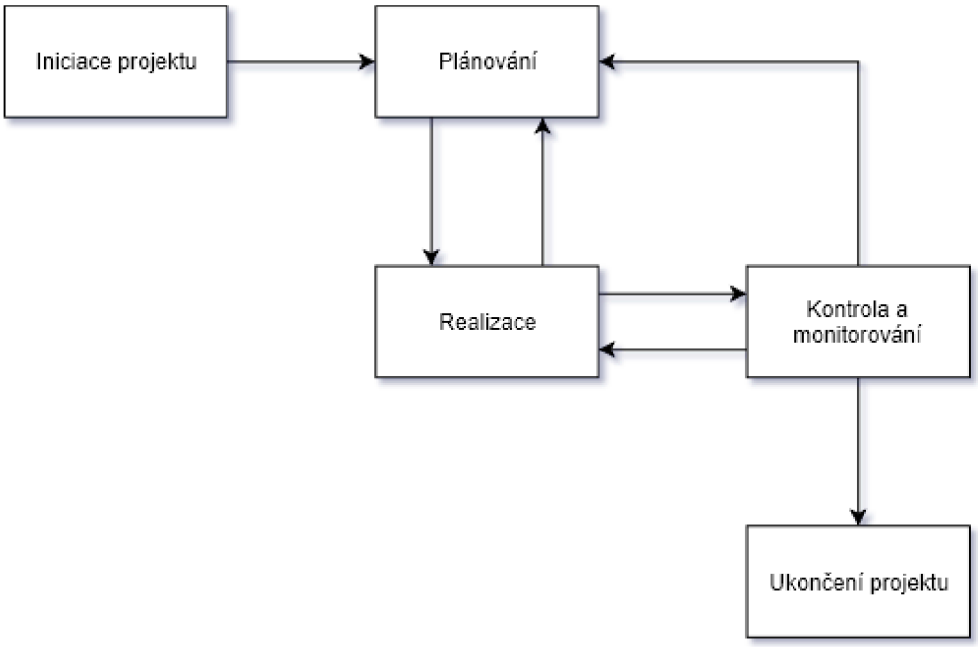
Jak můžeme vidět na Obrázku 2.1, etapy nejsou lineárně za sebou, ale spíše se mezi sebou prolínají a dochází k postupnému iterování, než je projekt dokončen. Část Realizace jde ruku v ruce s Monitorováním a kontrolou.

2.1.1 Inicializace projektu

V rámci inicializace projektu je cílem vytvořit zakládací listinu projektu a identifikovat všechny zapojené strany [50].

Zakládací listina je dokument poskytující formální základ projektu. Na základě tohoto dokumentu, který je formálně odsouhlasen sponzorem projektu, mohou být započaty samotné práce. Poskytuje základní rámec vysvětlující, jaké jsou cíle projektu, proč projekt existuje, jaká jsou kritéria pro úspěch projektu a kdo bude zapojený do jeho realizace. [4, s. 81]

Identifikace zapojených stran je velice důležitá pro bezproblémový chod projektu. Z vlastní zkušenosti můžeme říci, že při neprovedení identifikace zapojených stran, může mít projektový manažer problémy s celkovým nastavením komunikace. Předejde se tak problémům,



Obrázek 2.1: Diagram znázorňující propojení jednotlivých projektových etap podle PMI (inspirováno [50])

průtahům a nedorozumněním, které bývají způsobeny když jsou informace a dotazy směřovány na nerelevatní nebo méně prioritní zainteresovanou stranu.

2.1.2 Plánování

Plánování je další etapou. Hlavním cílem je vytvoření plánu řízení projektu. Během plánování jsou určeny nejen vlastní plány pro relevantní oblasti, ale také pravidla pro práci projektového týmu a řízení daného projektu jako takového [39].

Dle PMBOK: "Plán projektu je formální, schválený dokument, který se používá jako vodítko pro realizaci projektu a projektového řízení. Primárně se plán projektu používá na zdokumentování předpokladů a rozhodnutí, usnadnění komunikace mezi zúčastněnými stranami, a zdokumentování schváleného rozsahu, ceny a harmonogramu. Plán projektu může být pouze souhrnný nebo velmi podrobný."[31]. Ve své podstatě jde o "nastavení pravidel hry".

2.1.3 Realizace

Intenzivní fáze projektu, kdy probíhá hlavní část aktivit [29]. Probíhá paralelně s částí Monitorování a kontrola. Jde o časově nejdelší část projektu [33].

V rámci této části probíhá, mimo samotných pracovních aktivit realizačního týmu, i řízení komunikace, řízení zapojení zainteresovaných stran, řízení projektových znalostí, řízení kvality, zavedení rizikových opatření a případné požadavky na nákup [50].

2.1.4 Monitorování a kontrola

V rámci této části dochází celkově ke kontrole průběhu realizace projektu s následnými změnovými řízeními pro odstranění objevených nedostatků. Kontroluje se rozsah projektu,

zdroje, náklady, kvalita výstupů. Dále se sledují rizika, komunikace a zapojení zainteresovaných stran. [50]

2.1.5 Ukončení projektu

V rámci ukončení projektu by mělo dojít k předávce výstupů projektu jeho zákazníkovi včetně jejich akceptace. Toto předání by mělo proběhnout na základě postupů a podmínek stanovených ve smlouvě nebo interních směrnicích. Při ukončení projektu by také mělo dojít k vyhodnocení úspěšnosti projektu. Vyhodnotit by se měl cíl projektu, stanovené metriky s následným návrhem opatření pro příští projekty. Projektový tým by neměl v této etapě zapomenou na úklid, tedy utřídit dokumenty, potřebné archivovat, uklidit přebývající materiál, tedy celkově uklidit po sobě pracoviště [13].

Tato etapa by měla být silně provázána s projektovou iniciací, jelikož dostatečně definovaný cíl a správně zvolené metriky jsou důležité pro možnosti vyhodnocení projektu a předcházení následných sporů mezi zainteresovanými stranami [13].

2.2 Znalostní oblasti managementu projektů podle PMI

Aktuálně poslední verze PMBOK definuje deset znalostních oblastí, obsahující celkově 49 procesů pro projektové řízení. Na rozdíl od předchozí verze PMBOK 5, přibyly tři nové procesy a jeden proces byl sloučen do jiného. Každá znalostní oblast seskupuje procesy, nástroje a techniky vztahující se k určité projektové problematice. Jednotlivé oblasti jsou hlouběji probrány níže.

2.2.1 Řízení integrace projektu

Projektová integrace je důležitou znalostní oblastí, která provází projekt po celou dobu jeho životního cyklu a je zastoupena ve všech jeho etapách. Obsahuje procesy k identifikaci, definici, kombinaci, sjednocení a koordinaci různých procesů a činností v rámci projektového řízení [4, s. 69]. Samotné procesy z této znalostní oblasti jsou popsány níže.

Vytvoření zákládací listiny projektu

Zakládací listina je dokument, který formálně potvrzuje existenci projektu a poskytuje projektovému manažerovi pravomoci k využití organizačních zdrojů. Poskytuje formální záznam o projektu a ukazuje k němu závazek organizace [4, s. 75]. Jde o dokument poskytující rámcové informace k projektu a jeho výstupům. Rámcově zajišťuje zakládací listina společné porozumění mezi zainteresovanými stranami ohledně klíčových výstupů, milníků a rolí zapojených do projektu [4, s. 81]. Tento proces probíhá v rámci *Inicializace projektu*.

Vytvoření projektového plánu

Jde o proces definice, přípravy a koordinování všech komponent plánování a jejich následné spojení v integrovaný projektový plán. Hlavním přínosem, a zároveň výstupem, tohoto procesu je *Projektový plán*, což je dokument definující základ veškeré projektové práce a toho, jak bude prováděna [4, s. 82]. Tento dokument definuje jak bude projekt realizován, jak bude probíhat kontrola a jak bude uzavřen. Detailnost plánu se odvíjí od potřeb projektu [4, s. 83]. Vytváření projektového plánu navazuje na *Zakládací listinu projektu*.

Projektový plán by měl obsahovat vymezení rozsahu, nákladů a času na realizaci projektu. Toto je nutné, aby šlo řídit výkonnost projektu [4, s. 83].

Řízení práce na projektu

Řízení práce na projektu je proces vedení, vykonávání práce definované v *Projektovém plánu* a implementace schválených změn, to vše k dosažení projektových cílů. Celkově se jedná o řízení projektových prací a výstupů, což má za důsledek zvýšení pravděpodobnosti úspěšného dokončení [4, s. 90].

Řízení projektových znalostí

Řízení projektových znalostí je proces využívání stávajících znalostí a vytváření nových k dosáhnutí projektových cílů a k rozšíření znalostí organizace. Velkým přínosem je, že takto nabyté znalosti je možné využít i v dalších projektech nebo projektových fázích [4, s. 98]. Nejdůležitější součástí *Řízení projektových znalostí* je vytváření atmosféry důvěry, která podporuje sdílení znalostí mezi lidmi [4, s. 100].

Důležitou komponentou je *Registr získaných znalostí*, který obsahuje informace ohledně doporučených postupů v určitých situacích [4, s. 101]. Může obsahovat kategorizované situace s jejich popisy. Může také obsahovat dopady daných situací, doporučená a navrhovaná řešení daných situací. Tento registr také může zaznamenávat projektové výzvy, problémy, zjištěná rizika a příležitosti, případně další relevantní informace [4, s. 104].

Kontrola a monitorování projektové práce

Tento proces je zaměřen na měření, vyhodnocování a hlášení celkového postupu k dosáhnutí výkonnostních cílů definovaných v rámci *Projektového plánu*. Umožňuje zainteresovaným stranám porozumět aktuálnímu stavu projektu a mít vhled do očekávaného dalšího vývoje včetně nákladů a časování [4, s. 105]. Tento proces probíhá paralelně s prováděním projektových prací.

Na základě zjištěných informací můžou vzniknou změnové návrhy, které budou následně provedeny v rámci *Provedení integrovaného změnového řízení* [4, s. 112].

Provedení integrovaného změnového řízení

Provedení integrovaného změnového řízení je proces vyhodnocování všech změnových návrhů, jejich schvalování a řízení změn výstupů, projektových dokumentů a *Projektového plánu*, včetně komunikování daného rozhodnutí [4, s. 113].

Změnové návrhy mohou přijít od kterékoliv zainteresované strany v rámci celého životního cyklu projektu. Ačkoliv mohou být změny iniciovány ústně, nemělo by se zapomenout je zaznamenat písemně. Změny mohou ovlivnit všechny komponenty *Projektového plánu*. V případě, že by změna ovlivnila nějakou ze základních veličin čas-náklady-rozsah, tak musí být takový změnový návrh formálně odsouhlasen od zodpovědného jedince, nejčastěji projektového sponzora [4, s. 115].

Ukončení projektu nebo jeho etapy

Ukončení projektu popisuje ukončení všech prací na projektu [4, s. 112]. Jsou provedeny činnosti nutné k formálnímu uzavření projektu [4, s. 123]. Všechny projektové dokumenty jsou aktualizovány a označeny za finální. Důležitým dokumentem je *Registr získaných znalostí*,

který může dále posloužit zbytku organizace. Je vytvořeno finální vyhodnocení projektu [4, s. 127].

2.2.2 Řízení rozsahu projektu

Řízení rozsahu projektu obsahuje procesy potřebné k ujištění, že projekt obsahuje veškerou potřebnou práci, a jenom tu, která je nutná k úspěšnému dokončení projektu. *Řízení projektového rozsahu* se primárně pohybuje okolo definice a kontroly, co má být v projektu zahrnuto a co ne [4, s. 129].

2.2.3 Řízení časování projektu

Řízení časování projektu obsahuje šest procesů potřebných k včasnému dokončení projektu [4, s. 173]. Poskytuje detailní plán znázorňující, jak a kdy by měly být hotovy jednotlivé body definované v rozsahu projektu. Slouží i jako nástroj pro komunikaci a řízení očekávání zapojených stran [4, s. 175].

2.2.4 Řízení nákladů projektu

Řízení nákladů projektu obsahuje procesy dotýkající se plánování, odhadování, rozpočtování, financování a kontrolování nákladů, aby byl projekt dokončen v rámci schváleného rozpočtu. V rámci menších projektů jsou mnohdy tyto procesy uvažovány jako jeden [4, s. 231].

Tato znalostní oblast se zabývá hlavně náklady na zdroje, které jsou nutné k provedení projektových činností. Nemělo by se zapomenout na následné opakující se náklady spojené se správou a podporou výstupů projektu [4, s. 233].

2.2.5 Řízení kvality projektu

Řízení kvality projektu zahrnuje procesy začleňující organizační politiky kvality, týkající se plánování, řízení a kontroly projektu, včetně kvality výstupů tak, aby došlo k naplnění cílů zainteresovaných stran [4, s. 271]. Jedná se o kvalitu řízení projektu a jeho výstupů [4, s. 273].

2.2.6 Řízení zdrojů projektu

Řízení zdrojů projektu zaštiťuje procesy k identifikaci, získání a řízení zdrojů nutných k úspěšnému dokončení projektu. Tyto procesy pomáhají zajistit, aby byly projektovému manažerovi a projektovému týmu dostupné správné zdroje ve správný čas a na správném místě. [4, s. 307]

Důležitou součástí realizace projektu je projektový tým, ten se skládá z jedinců, s přiřazenými kompetencemi a odpovědnostmi, kteří dohromady pracují na dosažení společného projektového cíle. Projektový manažer by měl proto investovat dostatečné úsilí do řízení, motivace a posilování týmu. [4, s. 309]

2.2.7 Řízení komunikace projektu

Řízení komunikace projektu obsahuje procesy nutné k zajištění informačních požadavků projektu a zainteresovaných stran, pomocí vytváření informačních artefaktů a implementace činností navržených k dosáhnutí efektivní výměny informací. Celkově se dělí na dvě části.

První je příprava strategie k dosažení efektivní komunikace mezi zainteresovanými stranami. Druhou je provádění činností k zavedení dané strategie. [4, s. 359]

Komunikace popisuje možné způsoby, jak předávat nebo získávat informace. Buď komunikačními činnostmi jako jsou schůzky a prezentace nebo pomocí artefaktů, jako například emaily, projektová hlášení a dokumentace [4, s. 361].

Plánování řízení komunikace

Plánování řízení komunikace je proces vytváření vhodného přístupu a plánu pro komunikační činnosti v rámci projektu. Toto probíhá na základě informačních potřeb každé zainteresované strany, projektu a dostupných prostředků. Hlavním přínosem je, že tento proces poskytuje dokumentovaný přístup k efektivnímu a účinnému zapojení zainteresovaných stran, pomocí včasného prezentování relevantních informací. [4, s. 366]

Řízení komunikace

Řízení komunikace je proces zajištění včasného a vhodného sběru, vytváření, distribuce a ukládání projektových infromací. Tento proces probíhá v rámci celé realizace projektu a umožňuje efektivní proud informací mezi projektovým týmem a zainteresovanými stranami. [4, s. 379]

Sledování komunikace

Sledování komunikace je proces kontrolující, zda jsou informační potřeby všech stran dostatečně naplněny [4, s. 388]. Zjišťuje, zda mají plánované činnosti a artefakty kýžený efekt zvyšování nebo alespoň udržování podpory mezi zainteresovanými stranami [4, s. 389]. Na základě zjištěných nedostaků mohou vzniknout změnové požadavky pro jejich nápravu [4, s. 393].

2.2.8 Řízení rizik projektu

Řízení rizik projektu obsahuje procesy k vykonávání plánování řízení rizik, identifikaci, analýzu a jejich monitorování, dále také plánování a implementaci reakcí na nastalá rizika. Cílem této oblasti je zvýšení pozitivních dopadů příležitostí, a zároveň snížení dopadů negativních rizik. To vše k zvýšení pravděpodobnosti úspěšného dokončení projektu [4, s. 395]. Všechny projekty se potýkají s určitými riziky a pokud tato rizika nejsou řízena, tak může dojít až k selhání projektu, podle závažnosti daného rizika [4, s. 397].

2.2.9 Řízení obstarávání projektu

Řízení obstarávání projektu zahrnuje procesy nutné k nákupu nebo zajištění produktů nebo služeb mimo projektový tým. Tato oblast obsahuje řízení a kontrolu procesů nutných k vytvoření a administraci smluv, objednávek a interních dohod o úrovni služeb. Členové oprávnění obstarávat zboží nebo služby mohou, ale nemusí být přímo členy projektového týmu. [4, s. 459]

V rámci procesu obstarávání, na rozdíl od ostatních částí, se mohou vyskytovat asi největší právní závazky a postihy. I když většinou není projektový manažer oprávněn vstupovat za organizaci do právních závazků, což mají na starosti většinou jiní členové organizace, měl by mít obecný přehled k tomu, aby činil rozumná rozhodnutí ohledně smluv a smluvních

vztahů [4, s. 460]. Doporučení projektového manažera můžou mít totiž silný vliv na oprávněné osoby.

Samotné procesy by se měly přizpůsobit velikosti a komplexnosti poptávaného plnění. Úsilí vložené do tvorby smluv by mělo být přiměřené. Smlouvy by měly dostatečně jasně popisovat objednané plnění [4, s. 460-461].

2.2.10 Řízení zainteresovaných stran projektu

Řízení zainteresovaných stran projektu zahrnuje procesy potřebné k identifikaci lidí, skupin nebo organizací, které mouhou mít vliv na projekt, nebo jím mohou být ovlivněny. Dále osahuje procesy k analýze jejich očekávání a celkově velikosti jejich vlivu. To vše s cílem efektivního zapojení zainteresovaných stran do relevantních projektových rozhodnutí a samotné realizace. [4, s. 503]

Určité strany mohou mít na projekt velice silný vliv, proto by měla být jejich spokojenost identifikována a řízena jako jeden z projektových cílů. Ideálně by měla probíhat průběžná komunikace k zjištění jejich potřeb a očekávání [4, s. 505].

V rámci této znalostní oblasti se nachází celkově čtyři procesy.

Identifikace zainteresovaných stran

Identifikace zainteresovaných stran je proces při kterém se průběžně identifikují, analyzují a dokumentují relevantní informace ohledně zajmů, zapojení a vlivu jednotlivých stran. Tento proces se provádí opakovaně, podle potřeb, v průběhu projektu. [4, s. 507]

Klíčovou částí tohoto procesu je *Analýza zainteresovaných stran*, kde se sbírají a vyhodnocují informace o zájmech, pravomocích, znalostech, přínosech a dalších relevantních informacích o jednotlivých zainteresovaných stranách [4, s. 512]. Výstupem tohoto procesu je *Registr zainteresovaných stran*, kde jsou jednotlivé strany identifikovány a klasifikovány podle potřeb projektu [4, s. 514].

Plánování zapojení zainteresovaných stran

Plánování zapojení zainteresovaných stran je proces stanovování přístupu k zapojování jednotlivých zainteresovaných stran na základě jejich potřeb, očekávání, zájmů a potenciálního vlivu na projekt. Plán by měl být průběžně aktualizován na základě postupu projektu, případně změn v rámci zainteresovaných stran. První verze je vytvořena po úvodní *Identifikaci zainteresovaných stran.* [4, s. 516]

Řízení zapojení zainteresovaných stran

Řízení zapojení zainteresovaných stran je proces komunikace a práce se zainteresovanými stranami s cílem uspokojit jejich potřeby a naplnit očekávání a podpořit jejich vhodné zapojení. Hlavním přínosem je, že umožňuje projektovému manažerovi zvyšovat podporu projektu. [4, s. 523]

Monitorování zapojení zainteresovaných stran

Monitorování zapojení zainteresovaných stran je proces sledování vztahů se zainteresovanými stranami a příprava strategií jak dále zlepšovat jejich zapojení. Hlavním přínosem je, že pomáhá udržet efektivitu a účinnost nastavených činností v celém průběhu projektu [4, s. 530]. Probíhá společně s ostaními procesy v celém průběhu realizace projektu. Na základě

zjištěných nedostatků nebo možností k zlepšení vznikají *změnové návrhy*, které jsou dále zpracovávány procesem *Provedení integrovaného změnového řízení* [4, s. 535].

2.3 Agilní metodiky ve vývoji software

Roku 2001 došlo k historicky klíčovému setkání sedmnácti lidí, z nichž následně vznikla organizace *Agile Alliance*. Toto setkání bylo klíčové, jelikož dalo za vznik manifestu definující podstatu agilních metodik [24]. *Manifest agilního vývoje software* byl důsledkem zvyšování administrativní náročnosti řízení softwarových projektů a přidávání dalších vrstev managementu [41].

> Jednotlivci a interakce před procesy a nástroji
>
> Fungující software před vyčerpávající dokumentací
>
> Spolupráce se zákazníkem před vyjednáváním o smlouvě
>
> Reagování na změny před dodržováním plánu

Citát 1: Agile manifesto [9]

Ze samotného manifestu definuje *Agile Alliance* dvanáct upřesňujících principů agilního řízení, znění jednotlivých bodů lze vidět v Citátu 2. Z těchto principů vycházejí postupy, které jsou vybírány podle konkrétních potřeb projektu [3, s. 10].

> - Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
> - Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
> - Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
> - Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
> - Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
> - Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
> - Hlavním měřítkem pokroku je fungující software.
> - Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
> - Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
> - Jednoduchost–umění maximalizovat množství nevykonané práce–je klíčová.
> - Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
> - Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

Citát 2: Agilní principy rozvíjející Manifest agilního vývoje software [10]

Se zvyšující se nejistotou v rámci projektu se zvyšuje pravděpodobnost změn, zbytečné práce a nutnosti přepracovávat výstupy projektu, což je značně nákladné a časově náročné [3, s. 14]. Agilní přístupy se snaží takovým situacím předcházet a co nejvíce je eliminovat. Pojmy *Agilní přístupy* a *metodiky* zastřešují různorodé metodiky naplňující podstatu *Agilního manifestu* [3, s. 13].

Agilní projektový životní cyklus přináší týmům možnost získávat dřívější zpětnou vazbu, také s sebou přináší větší zapojení zákazníka a jeho vliv na projekt. Díky kratším vývojovým cyklům a častějšímu vydávání verzí může projekt začít generovat výnosy dříve. Tým se navíc může soustředit na části s nejvyšší přidanou hodnotou nejdříve, což dané zisky umocňuje. [3, s. 19]

Role projektového manažera v rámci agilního prostředí není úplně definována, jelikož většina agilních přístupů s touto rolí nepočítá. Existují názory, že tato role není potřeba, jelikož veškeré kompetence zvládá zajistit samoorganizovaný tým. Každopádně více pragmatický názor tvrdí, že projektoví manažeři mohou v určitých situacích přinést značnou přidanou hodnotu. Hlavním rozdílem je, že se liší jejich povinnosti a odpovědnosti. [3, s. 37]

Agilní týmy se soustředí na rychlý produktový vývoj, z důvodů již zmiňovaného získávání zpětné vazby. Nejefektivnější počet členů v týmu je tři až devět, s tím, že ideálně sedí fyzicky blízko sebe. Každý člen je dedikován svou plnou kapacitou na jeden konkrétní projekt. Rozdělování práce probíhá uvnitř týmu, kde se členové dohadují na tom, kdo bude pracovat na jaké části projektu [3, s. 39]. Pokud by totiž nebyl kompletně dedikován, tak by docházelo k multitaskování, což by vedlo k významně omezené produktivitě [3, s. 44].

Dle PMI figurují v rámci agilního vývoje primárně tři role. *Členové týmu* jsou jedinci se schopnostmi potřebnými k vytvoření funkčního produktu. V softwarovém vývoji mezi ně většinou patří vývojáři, testeři, designéři, případně další potřební jedinci. *Product Owner* je zodpovědný za správné směřování produktu. Má na starost prioritizaci jednotlivých částí projektu na základě obchodní hodnoty. Komunikuje se zainteresovanými stranami s cílem definovat a udržet produktovou vizi. *Týmový facilitátor* je člověk, který podporuje tým a pomáhá mu k dosažení cílů projektu, v metodice *Scrum* bývá označen jako *Scrum Master*. [3, s. 41]

Níže jsou popsány vybrané agilní metodiky.

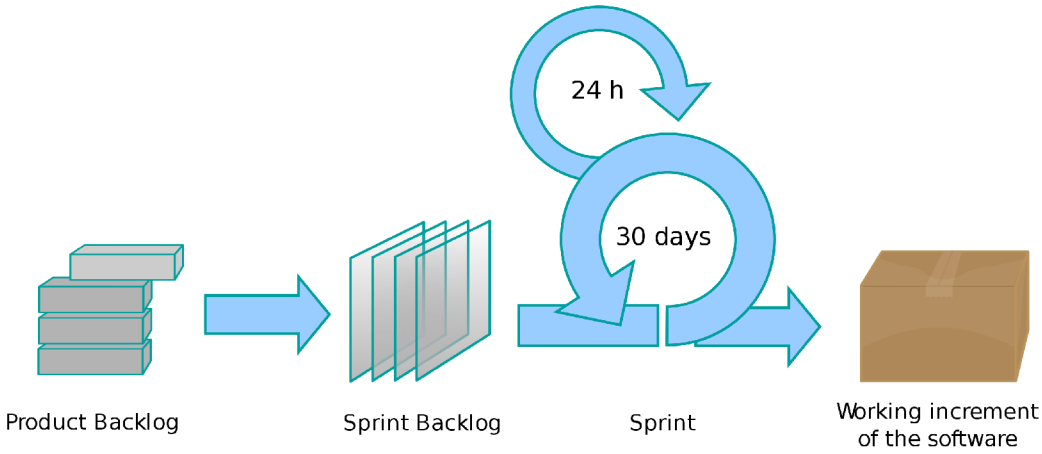
2.3.1 Scrum

Scrum je agilní metodika poskytující procesní rámec pro realizaci projektů. Dává silný důraz na produkt a procesy pro neustálé zlepšování tohoto produktu [44, s. 3]. Vývoj probíhá v několikatýdenních iteracích (Sprintech), se kterými se pojí určité ceremoniály. Podoba samotného procesu je znázorněna na Obrázku 2.2

Scrum procesy stojí na třech základních pilířích: *transparentnost*, *kontrola* a *přizpůsobivost* [44, s. 4].

Transparentnost - podstatné části procesů musí být přístupné všem stranám odpovědným za výstup. Přičemž tyto strany sdílí stejné porozumění dané problematice. *Kontrola* nese průběžné revidování projektových artefaktů a směru k cíli, což slouží k vyhnutí se nechtěným odbočkám. Přítomnost elementu *kontroly* je důležitá, ale měla by být tak častá, aby neomezovala samotnou práci na projektu. *Přizpůsobivost* je klíčová v případě, že se zjistí nějaká výraznější deviace od cílů a procesů. Důležité je přizpůsobit se situaci v co nejkratším čase, aby se tak minimalizovali případné ztráty. [44, s. 5]

V *Scrum* týmech je důležité, aby figurovaly hodnoty jako odpovědnost k závazku, odvaha, soustředěnost, otevřenost a respekt. Teprve takto začnou základní pilíře metodiky



Obrázek 2.2: Diagram ukazující iterativní proces vývoje v rámci metodiky Scrum (převzato [26])

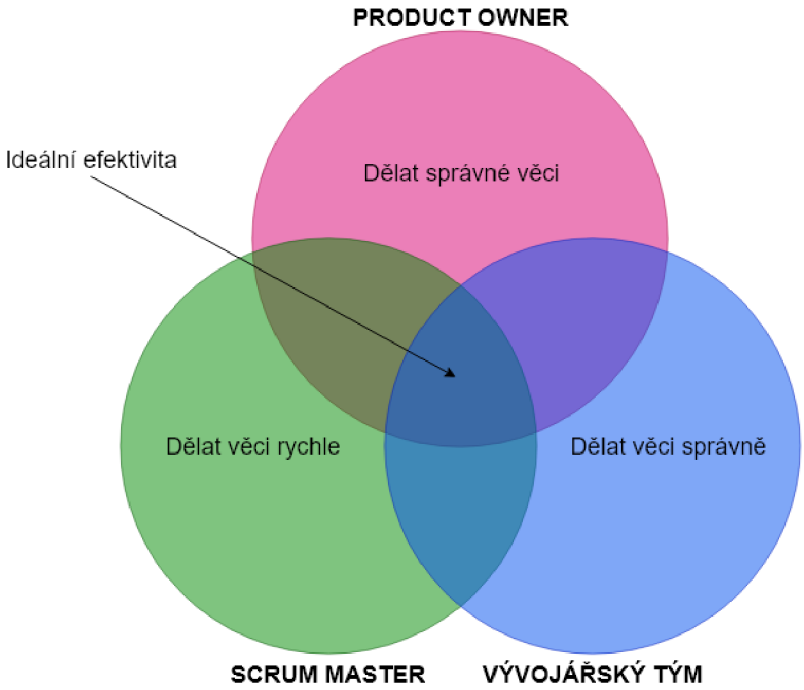
dávat pořádný smysl. Úspěch metodiky *Scrum* je závislý na tom, aby zmiňované hodnoty byly přirozenou součástí práce každého jedince zapojeného v projektu. Jedinci se osobně *zavážou* k dosažení projektového cíle. Členové týmu mají *odvahu* volit správná rozhodnutí a pracovat na obtížných problémech. Všichni se *soustředí* na práci ve daném Sprintu. Tým, spolu s ostatními zainteresovanými stranami, vede *otevřenou* komunikaci o všech aspektech projektu. Všichni zapojení jedinci se navzájem *respektují* jako schopní samostatní lidé. [44, s. 5]

V rámci této metodiky figurují tři primární role. *Product Owner* je jedinec zodpovědný za zvyšování hodnoty produktu, spravuje produktový plán a určuje priority pro vývojový tým. Komunikuje se zainteresovanými stranami jako jsou uživatelé, sponzor projektu a podobně. *Scrum Master* je týmový facilitátor, jehož odpovědností je, aby byly jednotlivé procesy dodržovány správně. Identifikuje problematická místa a navrhuje zlepšující opatření. *Vývojový tým* je skupina jedinců s potřebnými dovednostmi k realizaci daného projektu. [44, s. 6-7] Vztah mezi těmito rolemi je ukázán na Obrázku 2.3.

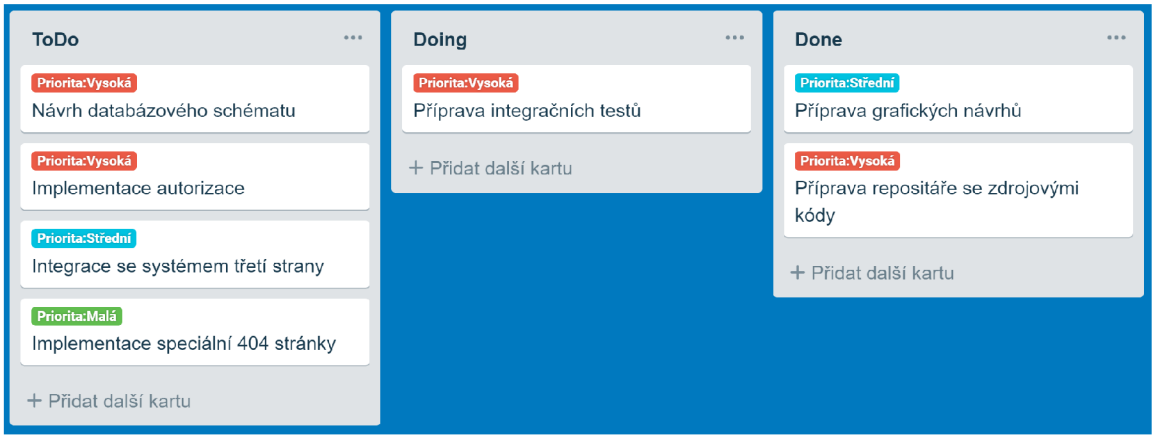
2.3.2 Kanban

Kanban je metodika, která má své kořeny v japonském automobilovém průmyslu. Tato metodika využívá jednoduchý systém kartiček [30]. Tyto kartičky se přesouvají mezi několika sloupci. Každý sloupec znázorňuje určitý stav, ve kterém se daná kartička nachází. Základní rozložení *Kanban* tabule jsou tři sloupce *ToDo*, *Doing*, *Done*. Toto rozložení lze vidět na Obrázku 2.4.

Praktické využití může zjednodušeně vypadat následujícně. Vydefinují se činnosti nutné k dokončení projektu. Pro každou činnost se vytvoří kartička a ta se umístí do sloupce *ToDo*. V případě začátku provádění určité činnosti se odpovídající kartička přesune do *Doing*. Po dokončení se umístí do *Done*. Jakmile jsou všechny kartičky v posledním sloupci, projekt je hotov.



Obrázek 2.3: Diagram znázorňující spolupráci v rámci *Scrum* metodiky (inspirováno [23])



Obrázek 2.4: Možný příklad Kanban tabule - vytvořeno v systému Trello.com (snímek obrazovky)

2.3.3 Extrémní Programování (XP)

Extremní Programování dává důraz na zákaznickou spokojenost. Namísto dodávání celého projektu někdy daleko v budoucnosti, se vývojáři soustředí na průběžné dodávání. *XP* podporuje spolupráci. Týmy jsou samoorganizované a soustředěné na řešení daného problému. Funkcionality jsou dodávány jakmile jsou potřeba. [52]

Testování je v rámci *XP* klíčové. Veškerý kód musí být pokryt pomocí jednotkových testů. Všechny tyto testy musí být před nasazením verze úspěšné. Jakmile je nalezena chyba, tak je testovací sada rozšířena o příslušné případy užití. Akceptační testování probíhá průběžně a výsledky jsou sledovány. [51]

2.3.4 Feature-driven Development (FDD)

FDD je agilní metodika založená na iterativním a inkrementálním vývoji. Nepokrývá celý vývojový proces, ale zaměřuje se na návrh a vývojovou etapu [35]. Jádrem jsou *Užitné vlastnosti (features)*, což jsou výsledky užitečné z pohledu zákazníka. Tyto výsledky jsou srozumitelné, měřitelné a realizovatelné v rámci vývojové iterace [37].

2.4 Spojení PMI znalostních oblastí a Agile

V rámci této sekce je popsáno, jak podle *Agile Practice Guide* jsou jednotlivé PMI znalostní oblasti přizpůsobeny na jejich využití v rámci agilního prostředí.

2.4.1 Řízení integrace projektu

Agilní přístupy zvětšují zapojení týmových členů do celkového průběhu procesu. Tým určuje, jak by jednotlivé plány a procesy měly do sebe zapadat. Zodpovědnost za plánování a dodávání je přesunuta na projektový tým. Hlavní rolí projektového manažera je podpora kooperativního prostředí a ujištění, že je tým schopen reagovat na změny. [3, s. 91]

2.4.2 Řízení rozsahu projektu

V projektech s postupně se vyvíjejícími požadavky není na začátku celkový rozsah mnohdy znám. Proto se v rámci agilních metodik tráví na začátku méně času definováním a domluvou rozsahu projektu. Čas se dedikuje spíše na nastavení správných procesů pro práci s nově nabytými poznatky a následným upřesňováním a uprávě stávajících požadavků. V průběhu vývoje se také pracuje s vytvářením a ověřováním prototypů a vydávaných verzí systému, toto vše k získání informací, které jsou využity k upřesnění požadavků. Důsledkem je, že se rozsah projektu definuje a mění v průběhu projektu samotného. [3, s. 91]

2.4.3 Řízení časování projektu

Jsou využívány krátké vývojové cykly, čímž tým získává rychle zpětnou vazbu a může se tak přizpůsobovat podle potřeb. Tyto cykly poskytují zpětnou vazbu na postupy a udržitelnost výstupů. V důsledku se celkově uplatňuje iterativní plánování. [3, s. 92]

2.4.4 Řízení nákladů projektu

Pro projekty s vysokou mírou nejistoty nebo s nedostatečně definovaným rozsahem nemají významnou hodnotu detailní finanční kalkulace. Místo toho se využívají spíše přibližné

odhady pracnosti podle kterých se dají odhadovat náklady na pracovní sílu. K detailním kalkulacím dochází v případě potřeby a jedná se o kalkulace krátkodobého horizontu. Pokud existuje pevný rozpočtový strop, tak se pracuje s úpravou rozsahu a časového plánu projektu, aby nedošlo k překročení finančních limitů. [3, s. 92]

2.4.5 Řízení kvality projektu

Při využívání agilních metodik je potřeba kontrolovat kvalitu pravidelně v průběhu celého projektu. Použití retrospektiv přináší průběžný náhled na efektivitu a kvalitu procesů. Z těchto retrospektiv také často vyplynou návrhy na zlepšení a po zavedení následně na dalších retrospektivách mohou být hodnoceny. Podle daného hodnocení může tým udělat rozhodnutí, zda pokračovat s danou změnou, udělat nějaké úpravy a nebo změnu zrušit. Cílem této průběžné kontroly je odhalit nedostatky v počátku, kdy jsou náklady na změnu menší. [3, s. 93]

2.4.6 Řízení zdrojů projektu

Samoorganizované týmy, které se snaží maximalizovat spolupráci a společný směr, jsou pro agilní projekty velkým přínosem. Spolupráce zvyšuje produktivitu a usnadňuje inovativní řešení problémů. Týmy, u nichž funguje spolupráce na dobré úrovni, jsou schopny usnadnit sdílení znalostí, spojení různých pracovních aktivit a flexibilně reagovat na změny. Spolupráce je v agilních projektech mnohdy klíčovým kritériem k úspěchu. V agilním prostředí totiž bývá méně času a prostoru pro centralizované rozhodování. [3, s. 93]

2.4.7 Řízení komunikace projektu

V prostředí agilních projektů dochází průběžně ke změnám a nejasnostem, u nichž je potřeba tyto informace častěji a rychleji komunikovat. Toto vede k zefektivňování přístupů k informacím jednotlivých členů týmu a větší synchronizaci týmu. Transparentní sdílení projektových artefaktů a validace ze strany zapojených stran vede k zlepšení komunikace s vedením. [3, s. 94]

2.4.8 Řízení rizik projektu

V prostředí s větším výskytem změn se přirozeně vyskytuje více nejistoty a rizik s tím spojených. V projektech řízených agilně proto dochází k pravidelnému přezkoumávání přírůstků. Podstatné je také urychlování sdílení znalostí, aby docházelo k porozumnění možným rizikům a také k jejich řízení. Navíc jsou požadavky ukládány jako živý dokument, který je průběžně aktualizován a jednotlivým úkonům tak může být změněna priorita v průběhu projektu podle aktuální úrovně porozumnění rizikům. [3, s. 94]

2.4.9 Řízení obstarávání projektu

V agilním prostředí mohou být vybráni určití dodavatelé k rozšíření týmu. Takováto forma spolupráce tak může vést k tomu, že obě strany budou společně sdílet rizika a zisky spojené s projektem. Větší projekty mohou pro nějaké výstupy použít adaptivní přístup a pro jiné jiný více konzervativní přístup. [3, s. 95]

2.4.10 Řízení zainteresovaných stran projektu

Projekty u nichž se vyskytuje velká míra změn potřebují aktivní zapojení a participaci zainteresovaných stran. Ke zvýšení efektivity komunikace tým komunikuje napřímo namísto několika vrstev managementu. Mnohdy zákazník, uživatel a vývojář sdílí informace v rámci určitého společného kreativního procesu, což vede k většimu zapojení zainteresovaných stran a také vyšší spokojenosti. Průběžné interakce se zainteresovanými stranami vedou ke snižování rizik, budování důvěry a podporují změny v rámci projektu. Agilní metodiky silně vyzdvihují velkou otevřenost. Přítomnost zainteresovaných stran na projektových schůzkách a sdílení projektových artefaktů na volně dostupná místa k jejich posouzení, může napomoct k rychlému odstranění nepřesností směřování týmu. [3, s. 95]

Kapitola 3

Specifikace požadavků na nástroj pro podporu řízení agilních vývojových týmů

Dle zkušeností z opakovaných navštěv událostí zaměřených na projektové řízení, konkrétně třeba série setkání *Agilní sdílení zkušeností*, pořádané v Brně *Jihomoravským Inovačním Centrem*, mívají společnosti největší mezery právě ve fázích projektové iniciace a úvodního plánování. Dle zmiňovaných setkání bývá tento problém často umocněn tím, že společnosti, které nemají dostatečně podchyceny základy projektového řízení se snaží zavádět agilní metodiky, protože slepě věří, že je takový přístup spasí.

Bez správně zavedných základů projektového řízení je značně ztíženo zavádění agilních přístupů. Je potřeba si uvědomit, že agilní přístup ve vývoji **není**:

- vývoj bez produktového plánu
- vývoj bez specifikace
- vývoj bez plánování [40]

Z pohledu aktuálně běžně dostupných nástrojů pro řízení agilních týmů, se tyto nástroje zaměřují převážně na projektové etapy *Plánování*, *Realizace* a *Monitorování a kontrola*. Proto je cílem tuto práci nejvíce soustředit na podporu projektové iniciace a úvodního plánování. Primárně se zaměřuje na metodiku *Scrum*, přičemž by měla najít své využití i v rámci jiných metodik. Podstatou systému je poskytnutí opory k operativními a taktickému řízení vývoje v mikro a malých společnostech.

Definice požadavků vycházela z diskuzí s potenciálními uživateli z šesti různých jihomoravských společností. Podstatné bylo, že tyto společnosti se zabývají vývojem software a mají již zkušenosti s agilním vývojem, případně alespoň jeho zaváděním. Role, které byly zastupovány mezi zmíněnými potenciálními uživateli byly *vedoucí vývoje*, *projektový manažer*, *Scrum Master*, *Product Owner* a *vývojář*.

Nejčastěji používanými nástroji pro řízení projektů, vedených agilní metodikou, byly *GitHub* (`https://github.com/`), *JIRA Software* (`https://www.atlassian.com/software/jira`) a *Trello* (`https://trello.com/`). Dané nástroje poskytují v určité míře podporu pro *Řízení rozsahu a časování projektu*. V rámci *Atlassian* produktů lze integrovat *JIRA Software* spolu s *Confluence* (`https://www.atlassian.com/software/confluence`). Primárním účelem *Confluence* je budování a sdílení znalostní báze v rámci organizace. Z tohoto

pohledu lze zaznamenávat projektové artefakty v rámci budované znalostní báze. S přihlédnutím k PMI lze přiřadit primární využití *Confluence* k procesu *4.4 Řízení projektových znalostí*. Přičemž umožňuje i pokrytí procesů ze skupin *Iniciace projektu*.

Dle produktové dokumentace lze integrovat s *Confluence* i *Trello* [6]. Tuto možnost nikdo z dotazovaných nepraktikoval, takže této kombinaci nebyla dále dána žádná důležitost.

Na základě zjištěných skutečností byly diskutovány požadavky na tvořený systém. Vznikl cíl vytvořit nástroj, který bude sloužit k podpoře PMBOK procesů *4.1 Vytvoření zakládací listiny* a *13.1 Identifikace zapojených stran*, obojí s vědomím doporučení obsažených v *Agile practice guide*. Následně byla definována potřeba podpory skupiny procesů z oblasti *Řízení zainteresovaných stran projektu*. Právě *Řízení zainteresovaných stran projektu* patřilo k nejvíce zanedbávaným oblastem. Následně by systém měl podporovat i *Řízení zdrojů projektu*, hlavně se zaměřením na plánování kapacit. V případě přidání *Řízení obstarávání projektu* by měl systém umožňovat predikci nákladů projektu a tím i určitou formu rozpočtování.

Dále vzešly i funkční požadavky. Nástroj by měl umožňovat kooperaci a sdílení informací mezi více členy projektového týmu. Důležité datové změny by měly být v systému implicitně verzovány. Důvodem je předcházení případnému řešení zbytečných chyb v komunikaci. Systém byl při vývoji průběžně konzultován i s potenciálními uživateli a požadavky byly průběžně validovány a upravovány. Preferovaná možnost je podpora *Single Sign-On*. Toto by přispělo k zvýšení uživatelského pohodlí díky možnosti využití již existujícího firemního účtu pro autentizaci do systému.

Z analýzy a komunikace lze vydefinovat klíčové požadavky následně:

- Vytvoření agilní zakládací listiny projektu dle PMI.
- Nástroje pro analýzu zainteresovaných stran.
- Registr zainteresovaných stran.
- Určení rolí pro jednotlivé projektové členy.
- Správu uživatelů v rámci organizace i samotných projektů.
- Poskytovat informace k dostupnosti vývojářů při plánování práce pro další *Sprint*.
- Reporting dostupných kapacit.
- Predikci a vyhodnocení nákladů na projekt.
- Historii změn v rámci projektových dokumentů.
- Žádosti o absenci jednotlivých členů realizačního týmu.
- Požadavky k nákupu ze strany projektového týmu.

Kapitola 4

Výběr implementačního prostředí a realizačních technologií

V rámci analýzy plánovaného řešení bylo nutné zvolit cílovou platformu, která bude definovat kde a jak bude systém provozován, a také jak bude fungovat jeho distribuce. Po volbě cílové platformy bylo následně potřeba zvolit technologii pro databázovou vrstvu a poté technologie pro implementaci systému jako celku.

4.1 Volba cílové platformy

Ze zkušenosti není softwarový vývoj s ohledem na výběr používaných operačních systémů mezi uživateli homogenním prostředím. Uživatelé při práci používají, jak různé verze *Windows*, tak *Mac OS X*, mezi vývojáři je také často běžné využití různých distribucí *Linuxu*. Z tohoto důvodu by měl být při tvorbě systému dáván důraz na více-platformní funkčnost systému. Další důležitou otázkou je, jak bude probíhat distribuce aplikace mezi samotné uživatele. S přihlédnutím na toto hledisko se jeví jako nejvhodnější možnost použití webového rozhraní. Uživatelé tak budou relativně snadno mít přístup k aplikaci bez nutnosti instalace.

Tvrdit, že volbou této platformy budou odstraněny veškeré problémy s distribucí by bylo velmi naivní. Každopádně se bude jednat převážně o problémy na straně nasazení aplikace nikoliv přímo u uživatelů. Jako příklad přetrvávajících problémů lze uvést nastavení *cache* zdrojových kódů spouštěných na straně klienta, dále verzování *REST API* rozhraní, atp. Samozřejmě nesmíme opomenout obvyklé problémy spojené s podporou většího množství prohlížečů. Volba podporovaných prohlížečů by měla být předmětem výstupů analytiky uživatelských přístupů. Pro potřeby prototypu bude řešení optimalizováno k použití na klasických počítačích s prohlížečem *Google Chrome* ve verzi 72 a vyšší.

Další zajímavostí mluvící pro volbu této platformy je možnost úpravy řešení do formy *Progresivní Webové Aplikace*. Jedná se o webové aplikace, které lze "nainstalovat", poté se pro uživatele jeví jako běžný aplikační program [27]. Tato funkcionalita je podporována napříč všemi čtyřmi nejpoužívanějšími operačními systémy od *Chrome* verze 73 [27].

4.2 Klientská aplikace

Volba cílové platformy určuje, že uživatelé budou k systému přistupovat skrz webový prohlížeč. První otázkou při volbě implementačních technologií je místo, kde se bude vykonávat

logika a vykreslování uživatelského rozhraní. V zásadě jsou k dispozici dvě možnosti. Vykreslování na straně serveru, kde na požadavky od klienta přichází odpovědi v podobě hotového HTML souboru, který je prohlížečem rovnou zpracován a zobrazen uživateli. V opačném případě se aplikace běžící v prohlížeči datazuje serveru na potřebná data, nejčastěji v *JSON* formátu, která zpracuje a dynamicky vykreslí na straně uživatele. S příhlédnutím k cílové skupině uživatelů lze předpokládat, že se v jejich řadách nebudou příliš vyskutovat jedinci se stroji s nízkým výpočetním výkonem. Spolu s možnostmi lepšího uživatelského zážitku a potenciálu rozšíření v *Progresivní Webovou Aplikaci* byla vybrána možnost druhá. To i přes zvýšené nároky na vývoj samotný.

V oblasti klientského webového vývoje je standardem použití jazyku *JavaScript*. Avšak v poslední době získává na čím dál větší popularitě i jazyk *TypeScript*. Tento jazyk je typovanou nadmnožinou jazyka *JavaScript* a umožňuje typovou kontrolu tvořených zdrojových kódů [34]. Typová kontrola umožňuje zachytit množství problémů a ulehčit programátorovi práci při orientaci v projektu [12]. Z těchto důvodů padlo rozhodnutí, že by *TypeScript* byl preferovanou možností před využitím samotného jazyka *JavaScript*.

Otázka, která následovala toto rozhodnutí byla, jakou technologii použít pro tvorbu samotné aplikace. Tvořit projekt čistě za použití jazyka *JavaScript* případně *TypeScript* by bylo zbytečně pracné a časově neefektivní. Na výběr bylo ze tří aktuálně předních frameworků *Angular*, *React.js* a *Vue.js* [47]. Při jejich srovnávání nebylo nalezeno nic zásadního, co by jasně určilo výběr jednoho z nich. Mnoho názorů bylo spíše na "náboženské" úrovni a tedy nepřílis relevantní. Samozřejmě mezi srovnávanými možnostmi existují rozdíly, avšak tyto rozdíly jsou v případě vyvíjeného systému zanedbatelné. Jako příklad může posloužit rozdíl mezi *Angular* a *React.js* ve vykreslování HTML, tento rozdíl je pozorovatelný na velmi slabých strojích. V *React.js* jsou změny v HTML překreslovány postupně, oproti druhé možnosti, kde změny jsou vykreslovány až v celku.

Kvůli malé zkušenosti v této oblasti vývoje byly hlavními důvody výběru rychlost orientace při tvorbě menšího prototypu a kvalita dostupné dokumentace. Pro všechny tři možnosti existují rozsáhlé dokumentace a úvodní návody, avšak při experimentování byl pocitově *Angular* nejpohodlnější. Samozřejmě lze najít i pragmatické důvody, například to, že jako u jediného je u něj poskytován *Long Time Support*. Jako další výhoda může být uvažována prokazatelně nejlepší integrace s jazykem *TypeScript*, jako jediný je v něm totiž přímo vyvíjen [17]. U *Vue.js* je naplánováno, že nová velká verze bude psána v jazyce *TypeScript* také. Při výběru technologií bohužel nebyl znám konečný termín vydání a stavět na tom projekt by bylo rizikové.

4.2.1 Grafické předpisy a komponenty

Při vývoji uživatelského rozhraní je vhodné následovat nějakou grafickou předlohu. První možností je realizace vlastních grafických návrhů a tvorba vizuálních komponent, ze kterých se bude následně budovat uživatelské rozhraní. Opačnou alternativou je následovat standardní grafické předpisy nějaké metodiky a využít již existující komponenty na kterých se bude dále stavět. Pro tvorbu prototypu a ověření konceptu s potenciální uživatelskou základnou je časově efektivnější možnost druhá.

Z důvodů osobních estetických preferencí byl pro tvorbu návrhu uživatelského rozhraní vybrán *Material Design*. Jedná se grafickou metodiku předepisující doporučené postupy při tvorbě grafického rozhraní [19]. Výhodou této volby je, že ruku v ruce s frameworkem *Angular* je vyvíjena knihovna *Angular Material* obsahující komponenty následující *Material Design* a jeho principy.

4.3 API server

Výběr technologie pro tvorbu *REST API* byla významně ovlivněna volbou technologie pro realizaci klientské aplikace. Jako implementační jazyk byl zvolen *TypeScript*, jelikož na teoretické úrovni by mělo být určitou výhodou používat stejný implementační jazyk jak na klientské straně, tak na straně serveru. Skutečným přínosem je možnost sdílení datových modelů mezi oběma vrstvami. Další původně nedoceněnou výhodou je sdílení *NPM* ekosystému pro tyto dvě vrstvy. V rámci obou je při vývoji použit *Node Package Manager* (dále jen NPM), což usnadňuje vývojáři mírně práci, jelikož nemusí přepínat technologický kontext. *NPM* je také využito pro orchestraci kompilace, testování a spouštění. Co tímto rozhodnutím bohužel není ovlivněno je rozdílnost domén klientského a serverového vývoje.

Pro samotnou realizaci byla vybrána knihovna *Express.js*. Důvodem výběru této knihovny byla určitá vývojářská zkušenost a také její jednoduchost. Toto rozhodnutí bylo podpořenou i její vysokou popularitou [14].

4.4 Databázová vrstva

Závěrečným klíčovým rozhodnutím byla volba perzistentního úložiště dat. Volba databáze silně ovlivní návrh datového modelu a operací nad ním. Srovnávané alternativy byly podle typu databáze uvažovány PostgreSQL jako reprezentant relačních databází a MongoDB jakožto zástupce databází dokumentových. PostgreSQL bylo vybráno, jelikož by podle jejich tvůrců mělo být nejpokročilejší open-source databází [22]. Navíc je poskytováno v rámci určitých cloudových platforem jako služba, což by mohlo v případě přesunu aplikace na cloud znamenat snažší migraci dat a změnu připojovacího řetězce. *MongoDB* je nejpopulárnější NoSQL databází na světě [36].

4.4.1 Srovnání relačních (SQL) a nerelačních (NoSQL) databází

Pro výběr vhodného typu databáze je nutno provést srovnání diskutovaných variant a vyhodnotit vhodnost jejich použití pro budovaný systém. V rámci relačních databází jsou data reprezentována tabulkami, které se skládají z datových řádků. Schéma těchto tabulek je předdefinováno narozdíl od dokumentových databází, které zpravidla nelimitují podobu uložených dat. Dokumentové databáze poskytují snažší práci s nestrukturovanými daty. Použití SQL databází je vhodnější v případě prostředí s množstvím komplikovaných dotazů. [2]

Nevýhodou nerelačních databází je, že se nelze spolehnout, že čtený dokument obsahuje konkrétní atribut. Toto je způsobeno právě zmiňovanou neexistencí schématu, které by vynucovalo určitý formát dat. Jelikož v nerelačních databázích, jak již jméno napovídá, nejsou pevné relace mezi daty, tak je častá úprava na sobě závislých dat problematická. Je totiž nutné při vývoji neopomenout, kde všude se daná data musí změnit.

Při experimentování se zvolenými kandidáty je možnost říct, že dokumentové databáze jsou vhodné pro tvorbu rychlého prototypu, který poslouží k ověření požadavků. Navrhnout schéma relační databáze je časově náročnější. V případě změn je nutno psát migrační skripty, které uzpůsobují schéma tabulek těmto změnám. Na druhou stranu toto schéma poskytne vývojáři pevnou oporu a část datové validace může být přesunuta na databázovou vrstvu.

Po přihlédnutí k výše zmíněnému srovnání bylo rozhodnuto použít relační databázi PostgreSQL. Hlavními důvody byla provázanost dat mezi entitami a jasná definice datového modelu.

4.4.2 Komunikace s databází

Pro komunikaci se zvolenou databází *PostgreSQL* byla zpočátku zvolena *npm* knihovna *pg*. Tato knihovna poskytuje minimum abstrakce a umožňuje přímou komunikaci s databází [11]. Při vývoji to nebylo vývojářsky komfortní a pro každý dotaz byla nutnost napsat samostatný textový řetězec. Do určité úrovně šlo tyto řětězce parametrizovat, avšak problém nastával v případě nutnosti podpory více permutací skupiny parametrů. Pro každou jednu permutaci bylo potřeba připravit odpovídající dotazový řetězec. Zásadní výhodou je to, že v jaké formě je dotaz napsán, tak v takové formě je databázi odeslán. Při práci s databází tedy není v tomto případě nutnost orientace ve vyšší úrovni abstrakce.

Po chvíli využívání knihovny *pg* začalo být naráženo na pracnost změn formátu SQL dotazů při změnách v systému. Tato nepružnost zpomalovala provádění změn a díky tomu protahovala dobu mezi získáváním zpětné vazby od potenciálních uživatelů. Proto byl proveden průzkum alternativních možností. Vybranou alternativou byla knihovna *sequelize-typescript*, která zapouzdřuje mapovač objektových relací (ORM) *sequelize* pro využití v kombinaci s jazykem *TypeScript*. Hlavní motivací bylo usnadnění práce s databázovou vrstvou. V tomto případě přidáním další úrovně abstrakce. Začátek s tímto ORM byl relativně rychlý, prvním krokem byla definice datového modelu. Toto bylo provedeno přímo v **.ts* zdrojových kódech. Již při definování datového modelu a relací mezi entitami přišla první komplikace, dokumentace *typescript-sequelize* nebyla příliš rozsáhlá. Na jednu stranu to dává smysl, jelikož dokumentace samotného *sequelize* je objemná a detailní. Na druhou stranu definice modelů probíhá při použití varianty pro *TypeScript* pomocí dekorátorů a je syntakticky velmi odlišná. Naštěstí je správce této knihovny aktivní a průběžně odpovídá na otázky přímo u repositáře se zdrojovými kódy, který je umístěn na adrese `https://github.com/RobinBuschmann/sequelize-typescript`.

Přidání ORM abstrakce snižuje flexibilitu práce s databází. Skládání komplexnějších dotazů probíhá komplikovaně a výstupy je nutno významně přemapovávat, aby byly snáze stravitelné pro klientskou aplikaci. Tyto problémy by mohla řešit metoda *raw*, která umožňuje komplexnější dotazy psát přímo jako samostatné SQL. Bohužel není v rámci *sequelize-typescript* typově podporována [5]. Tímto zjistěním bylo ukončeno experimentování s danou alternativou. Postavit další vývoj nad touto možností by bylo rizikové z pohledu komplikací v pozdější fázi vývoje. Vyměnit snažší a rychlejší start za potenciální průtahy později za to nestojí.

Naštěstí existuje mezikrok mezi ORM a SQL řetězci v podobě tvůrců SQL dotazů. Jedná se o knihovny, které poskytují určitou úroveň abstrakce pro tvorbu dotazů. Avšak tato abstrakce není příliš vzdálená samotné syntaxi jazyka SQL. Vývojář je schopen kód psaný za pomocí takovéto knihovny číst téměř jako SQL a dokáže si představit, jak bude vypadat vytvořený SQL řetězec. S tím, že automaticky generuje dotazy podle dané parametrové permutace.

Po analýze možností pro tvorbu SQL dotazů byla zvolena knihovna *Knex.js*, která podporuje *TypeScript*. Výhodou je možnost snadného sdílení poddotazů a jejich parametrizace. *Knex.js* poskytuje i rozhraní pro příkazovou řádku, pomocí které lze vytvářet a spravovat databázové migrace. Tvorba a úpravy databázového schématu lze psát v jazyce *JavaScript* namísto nutnosti psaní přímo SQL dotazů. V případě, že by se vykytnul nějaký problém, tak je možnost použít metodu *raw* a napsat potřebný dotaz rovnou jako SQL řetězec.

Na základě experimentů provedených s knihovnami *pg*, *sequelize-typescript* a *Knex.js* byla vybrána poslední možnost. Nevyžaduje nutnost znalostí speciálního rozhraní ORM, která není přenositelná napříč jednotlivými ORM knihovnami. Při tvorbě SQL dotazů po-

skytuje určitý vývojářský komfort, který se pojí se snadnější znovupoužitelností. Dalším důvodem je to, že při generování dotazu se jedná o rozumný kompromis mezi efektivitou generovaného dotazu a poskytovanou abstrakcí. Ukázka rozdílu mezi výstupy jednotlivých možností lze vidět ve Výpisech 4.1, 4.2 a 4.3. Zmíněné výpisy byly upraveny pro lepší čitelnost.

```
SELECT
    project_goals.*,
    projects.name AS project_name,
    projects.deadline AS project_deadline
FROM
    project_goals
LEFT JOIN
    projects ON projects.id = project_goals.project_id
WHERE
    projects.organization_id = ?;
```

Výpis 4.1: Ručně psaný SQL dotaz

```
select
  "project_goals".*,
  "projects"."name" as "project_name",
  "projects"."deadline" as "project_deadline"
from
  "project_goals"
left join
  "projects" on "projects"."id" = "project_goals"."project_id"
where
  "organization_id" = ?;
```

Výpis 4.2: SQL dotaz generovaný s pomocí *Knex.js*

```
SELECT
  "projects"."id" AS "projects.id",
  "projects"."name" AS "projects.name",
  "projects"."deadline" AS "projects.deadline",
  "projects"."start_date" AS "projects.start_date",
  "projects"."project_vision" AS "projects.project_vision",
  "projects"."organization_id" AS "projects.organization_id",
  "projects"."definition_of_done" AS "projects.definition_of_done",
  "projects"."notes" AS "projects.notes",
  "projects"."created_at" AS "projects.created_at",
  "projects"."updated_at" AS "projects.updated_at",
  "projects->project_goals"."id" AS "projects.project_goals.id",
  "projects->project_goals"."description"
    AS "projects.project_goals.description",
  "projects->project_goals"."unit" AS "projects.project_goals.unit",
  "projects->project_goals"."current_state"
    AS "projects.project_goals.current_state",
  "projects->project_goals"."project_id"
```

```
    AS "projects.project_goals.project_id"
FROM
  "project_goals" AS "projects->project_goals"
INNER JOIN
  "projects" AS "projects"
ON
  "projects"."id" = "projects->project_goals"."project_id"
WHERE
  "projects"."organization_id" = ?;
```

Výpis 4.3: SQL dotaz generovaný s pomocí *Sequelize*

4.5 Možnosti využití dalších existujících řešení

Při analýze byly identifikovány funkcionality, které by mohly být nahrazeny využitím externí služby nebo knihovny. Použití existujících řešení umožní větší možnost soustředění se na implementaci a rozvíjení klíčových částí systému s největší přidanou hodnotou pro zákazníka. Mimo jiné má také dlouhodobý potenciál ušetřit náklady na vývoj a údržbu systému. Samostatně vyvíjet snadno substituovatelné části by nebylo efektivní vzhledem k poměru mezi přidanou hodnotou a náklady na jejich zhotovení.

Při rozhodování se, zda na danou funkcionalitu použít existující knihovnu nebo ji implementovat samostatně, je nutné pohlížet i na jiná kritéria než čistě vývojový čas. Důležité je brát ohled i na udržitelnost a rozšiřitelnost, případně náklady na přepis. Také je vhodné posoudit i smysluplnost použití. Když se to vyžene do extrému, tak nedává smysl integrovat knihovnu o velikosti několika megabytů, aby se následně použila jedna funkce, která jde samostatně napsat pár řádky.

4.5.1 Autentizace a autorizace

Při návrhu architektury aplikace byla jednou z důležitých otázek autentizace a autorizace uživatelů. V rámci analýzy byly srovnávány tři možnosti. První možností bylo implementovat autentizaci a autorizaci vlastnoručně, další možností bylo integrovat existující knihovnu nebo použít existující službu.

Možnost vlastní implementace byla vyloučena z důvodu nutnosti pokrytí velkého množství funkcionality, která má velmi velký vliv na celkovou bezpečnost systému. Jako příklady můžou být zmíněny: správa a ukládání hesel, obnova hesel, řízení uživatelských relací a integrace *OAuth* autorizačních poskytovatelů. Tato funkcionalita by navíc měla být silně testována a celkově udržována. Tato možnost byla z výběru vyřazena nejdříve, jelikož přidaná hodnota vzhledem k pracnosti byla nejnižší.

Při srovnání zbývajících dvou alternativ byly zvažovány již konkrétní možnosti. Jako potenciální knihovny se nabízely *Passport* a *Permit*. Jejich výběr byl zvolen na základě množství vývojářů, kteří jej využívají dle statistiky stažení na npmjs.com. Obě knihovny jsou psány v jazyce *JavaScript*, avšak existují typové definice, takže by se nemělo jednat o technickou překážku. Uvažované služby byly *Auth0* a *Firebase Authentication*.

Permit je z výše zmíněných nejvíce jednoduchou variantou. Zaměřuje se čistě na API autorizaci [46]. Komplexita poskytovaného rozhraní není vysoká, což souvisí s úzkým zaměřením této knihovny. V případě jejího využití bude nutné samostatně implementovat správu a ukládání hesel, včetně uživatelské registrace.

Knihovna *Passport* patří k nejvíce používaným autentizačním knihovnám v rámci *Node.js* ekosystému [38]. Tato knihovna umožňuje zvolit několik autentizačních strategii, což se dá popsat jako autentizační přístupy. Kromě autentizace s kombinací jméno-heslo, může jít například také o integraci různých *OAuth* poskytovatelů jako je Google a Facebook. Dokonce existují strategie pro integraci porovnávaných služeb *Auth0* [28] a *Firebase Auth* [42]. Navíc zapadá do zvolených technologií, jelikož je koncipována jako *Express.js* mezivrsta [1]. Což vyhovuje našim potřebám. V případě volby lokální strategie jsou přihlašovací údaje uloženy v lokální databázi a je potřeba se starat o správnou koncepci jejich uložení, zejména šifrování hesel.

Auth0 je služba specializovaná právě na autentizaci a autorizaci aplikací. Poskytuje velké množství možností autentifikačních metod [7]. Do počtu 7000 uživatelů je služba poskytována bezplatně [8]. Z pohledu integrace s již zvolenými technologiemi jsou poskytovány knihovny jak pro serverovou část, tak i pro web. Tyto knihovny jsou vyvíjeny v jazyce *JavaScript*, avšak existují pro ně typové definice, tudíž by neměl být problém je využít. Potenciální nevýhodou, která však může být i výhodou, je přenesení autentizace a uložení dat s tím spojených mimo naši infrastrukturu. V případě, že bychom uvažovali o nasazení aplikace do intranetového prostředí s omezeným přístupem k internetu, tak bychom narazili na problém v nedostupnosti autentifikační služby. Tomuto problému by se nepředešlo ani v případě využití *Firebase Authentication*.

Poslední srovnávanou variantou byla služba *Firebase Authentication*, která je poskytována společností Google v rámci *Firebase Platform*, jenž je součástí *Google Cloud Platform* [18]. Zmiňovaná služba je bez omezení poskytována bezplatně. V základu umožňuje registraci kombinací email a heslo, případně přihlášení přes běžně používané *OAuth* poskytovatele. Narozdíl od všech předešlých variant je SDK vyvíjeno rovnou v jazyce *TypeScript*, tedy nemůže nastat situace, kdy by typová definice neodpovídala skutečné implementaci knihovny. Množství funkcionality oproti *Auth0* je menší, například chybí kompletně podpora technologie *Touch ID*. Na druhou stranu je součástí *Firebase Platform*, což by do budoucna mohlo přinést snažší integraci dalších služeb ze zmíněné platformy.

Po srovnání možností byla zvolena autentizační služba *Firebase Authentication*. Externí služba byla zvolena s přihlédnutím na budoucí udržitelnost a rozšiřitelnost aplikace. Na jednu stranu to vytváří projektovou závislost, na druhou stranu to přináší jistotu dodržení aktuálních bezpečnostních standardů. *Firebase Authentication* byl preferován před *Auth0* hlavně z důvodu poskytovaného SDK a potenciálních dalších možností rozšíření aplikace s rámci *Google Cloud Platform*. Funkcionalita poskytovaná vybranou variantou je plně dostačující pro potřeby projektu. Dále se ještě vedla diskuze, zda zvolenou autentizační službu nepoužít v kombinaci se zvažovanou knihovnou *Passport*. Přinosem by byla snažší možnost změny z *Firebase Auth* na něco jiného. Nevýhodou by bylo přidání robustní projektové závislostí. Z tohoto důvodu bylo rozhodnuto autorizační kontroly implementovat přímo v projektu několika funkcemi.

Další následně objevenou výhodou bylo, že pro *Firebase Authentication* existuje UI knihovna *FirebaseUI*, která lze využít pro vytvoření přihlašovacích a registračních formulářů. Zmíněná knihovna existuje v několika variantách, včetně webu. Problematické je, že pro integraci do *Angular* projektu je nutné využít speciální knihovnu, která *FirebaseUI for Web* pro toto použití obalí [25].

4.5.2 Odesílání e-mailových zpráv

V rámci řešení bude v případě potřeby odesílání emailových zpráv integrována služba sendgrid.com. Kromě poskytovaného REST API rozhraní je možnost využití SDK pro snažší integraci do našeho řešení. Toto, v kontrastu s nutností nasazení a udržování vlastního SMTP serveru, je při tvorbě prototypu aplikace značnou výhodou.

Hlavním případem užití v projektu je odesílání pozvánek neregistrovaným uživatelům. Obnova hesla je v případě registrace s využitím emailové adresy pokryta autentizační službou. V budoucnosti je zde potenciál k zasílání různých notifikací a měsíčních reportů.

Kapitola 5

Návrh a implementace

Po specifikaci požadavků a výběru technologií, následuje návrh řešení spolu se samotnou implementací. Cílem bylo vytvořit systém pro podporu řízení agilních vývojových týmu, zaměřený na podporu operativního a taktického rozhodování. Po volbě implementačních technologií bylo rozhodnuto koncipovat informační systém jako webovou aplikaci s uživatelským rozhraním přístupným přes webový prohlížeč. Funkcionalita implementovaná v rámci prototypu lze vidět v diagramu na Obrázku 5.1.

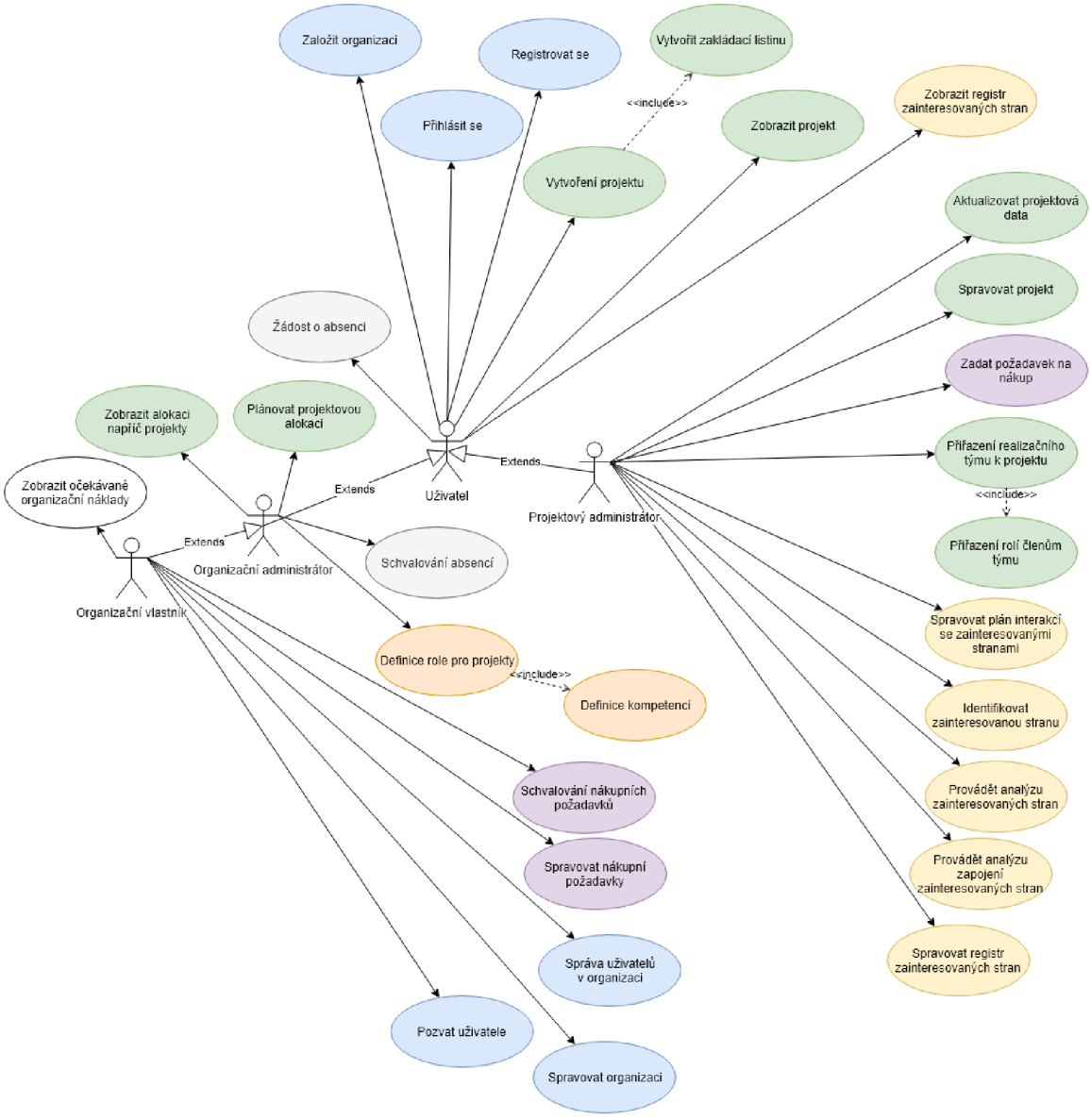
Systém samotný se skládá z více vrstev. Vrstvou nejbližší k uživateli je klientská webová *single-page* aplikace, implementovaná s využitím frameworku *Angular* a jazyka *TypeScript*. Serverová část je postavena nad běhovým prostředím *Node.js* s využitím stejného programovacího jazyka jako v případě webového rozhraní. Serverová část poskytuje REST API rozhraní, které je využíváno webovým klientem pro komunikaci za pomoci HTTP protokolu. Formát dat využívaný v komunikaci mezi těmito dvěma vrstvami je JSON.

Serverová část zapouzdřuje databázovou vrstvu. Komunikace s databází probíhá s využitím knihovny *Knex.js*. Databázová vrstva je postavena nad relační databází *PostgeSQL*.

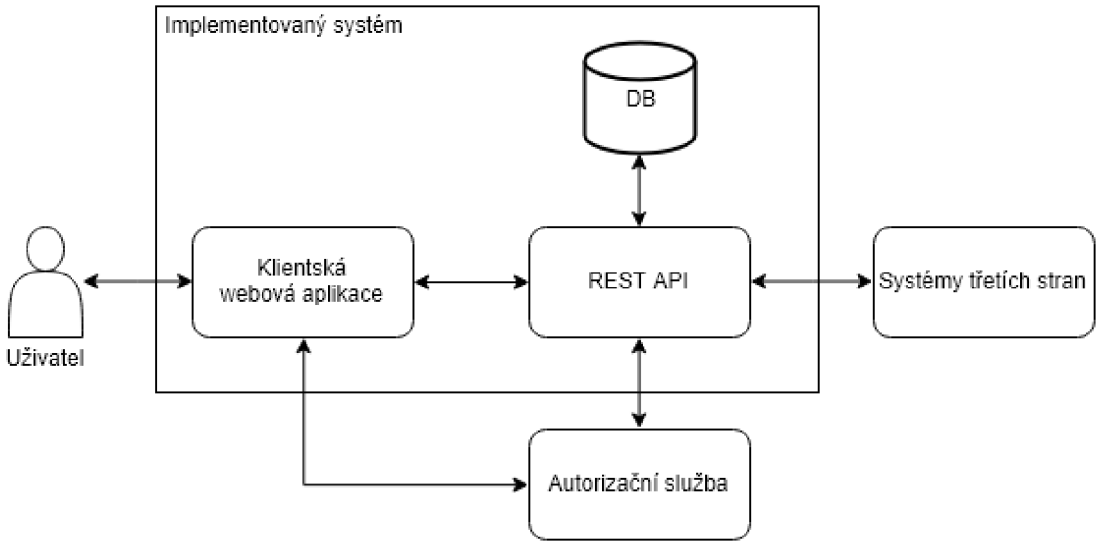
V rámci systému jsou integrovány také externí služby. K autentizaci a autorizaci uživatelů je využita služba *Firebase Authentication*. Pro odesílání emailových pozvánek je použita služba *SendGrid*. Znázornění finálního vysokoúrovňového návrhu architektury systému lze vidět na Obrázku 5.2

5.1 První iterace

Prvním krokem bylo vytvoření grafického prototypu pomocí nástroje *Adobe XD*. Prototyp znázorňoval, jak by přibližně mohlo vypadat uživatelské rozhraní včetně přibližného vzhledu jednotlivých funkcí dle kapitoly 3. Takto realizovaný prototyp umožňoval určitou formu navigace mezi jednotlivými obrazovkami a posloužil k validaci konceptu s potenciálními uživateli. V principu se jednalo o obrázky, které mezi sebou byly propojeny tak, že v případě kliknutí do určité definované oblasti došlo ke změně obrázku na jiný. Tvorba tohoto prototypu byla relativně časově nenáročná a umožnila rychlou zpětnou vazbu, která byla zapracována do dalšího návrhu. Po získání první zpětné vazby se přistoupilo dále k návrhu databázového schématu.



Obrázek 5.1: Diagram zachycující většinu implementovaných případů užití



Obrázek 5.2: Vysokoúrovňový návrh architektury systému

5.2 Databáze

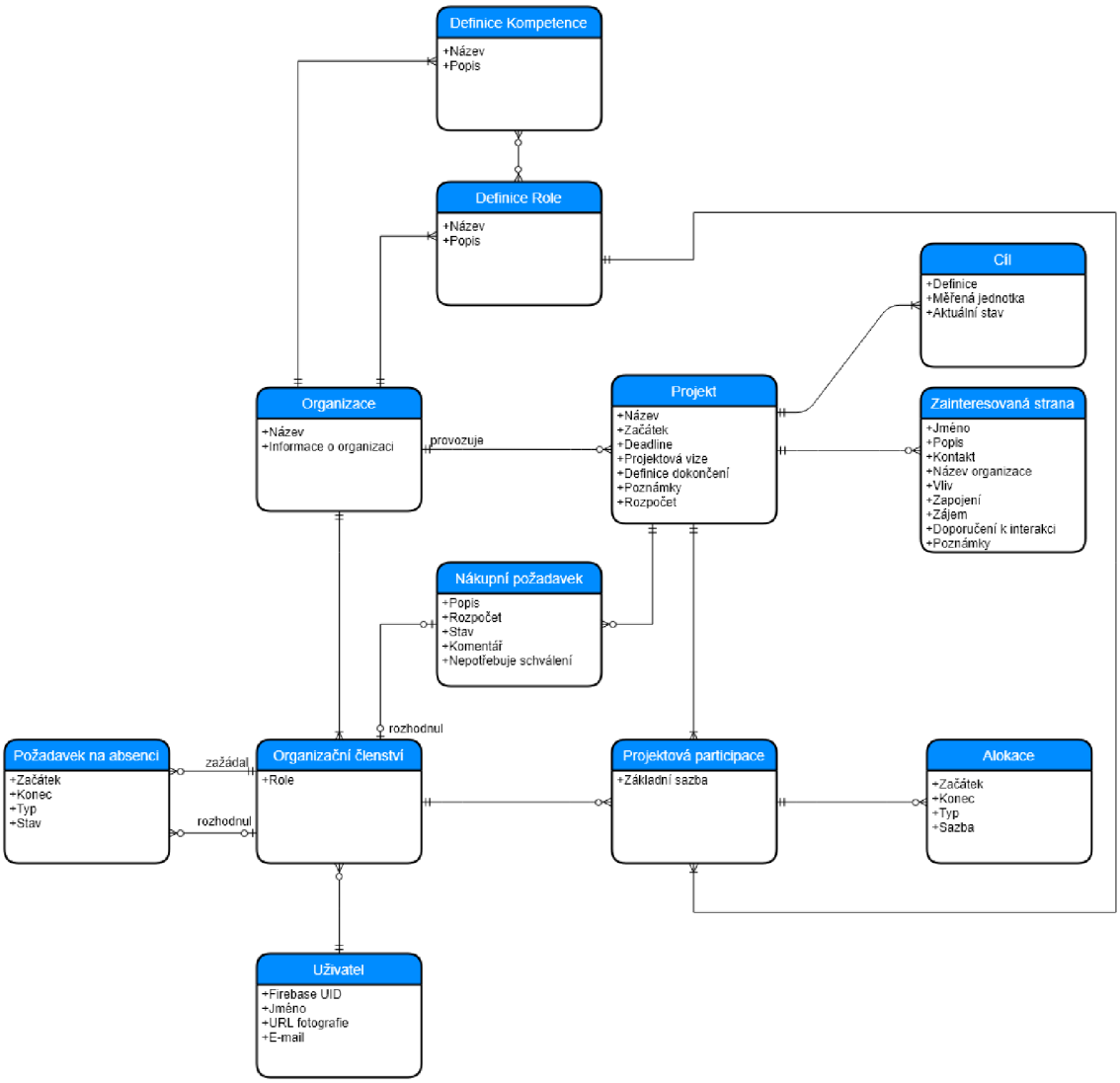
Při tvorbě databázového schématu se stavělo na aktualizovaném ER diagramu, který lze vidět na Obrázku 5.3.

5.2.1 Návrh

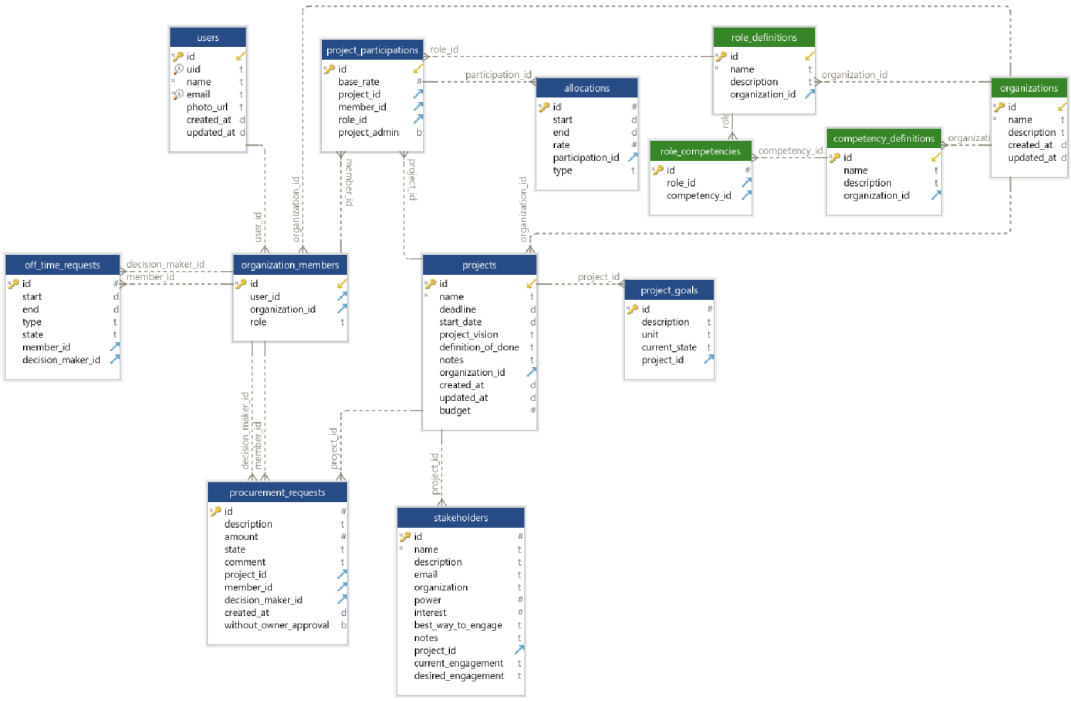
Systém podporuje existenci více na sobě nezávislých organizací zároveň. Uživatel může být v jednu chvíli součástí více organizací a v každé figurovat s jinou rolí. Tato příslušnost je vyjádřena vztahem organizačního členství. Každý jeden člen může participovat na více projektech patřících do jejich organizace. Dokonce je teoretická možnost, že organizační člen bude participovat v projektu pod více rolemi, například jako vývojář a zároveň grafický designér. V návrhu databázové struktury i serverové části je tato funkcionalita dostupná. Avšak v uživatelském rozhraní je záměrně omezena. Tento záměr je podložen diskuzemi s potenciálními uživateli, kde tato možnost nebyla využívána a došlo by tak ke zbytečnému přidání funkcionality, která by mohla mást uživatele. V praxi byly zmiňovány možnosti, že jeden člověk aktivně participuje na více různých projektech zároveň.

Pro každého organizačního člena mohou existovat požadavky na absenci, které lze agregovat s daty o alokaci a predikovat tak náklady projektu. Časová alokace jedinců se datově řeší na úrovni projektů. Neexistence dat o časových alokacích nelimituje využití pro správu. Avšak jeho nepoužití ovlivní neexistenci určitých informací, které mohou být pro společnost klíčové jako je třeba predikce nákladů. Další případ, kdy není potřeba vytvářet alokaci je když daný člen projektového týmu je do projektu zapojen pasivně. Není nutno, aby aktivně participoval, ale je vhodné aby byl součástí týmu. Z těchto důvodů není na databázové vrstvě vyžadována existence alokačního záznamu a ani se při připojení k projektovému týmu nevytváří.

Každá organizace může mít definovány vlastní projektové role a k daným projektovým rolím vlastní kompetence. Po vytvoření nové organizace se automaticky vytvoří nové výchozí záznamy pro definice rolí spolu s jejich výchozími kompetencemi.



Obrázek 5.3: ER diagram systému



Obrázek 5.4: Databázové schéma

Existující ER diagram se převedl do databázového schématu, které lze vidět na Obrázku 5.4.

PostgreSQL poskytuje datový typ *ENUM*, který funguje obdobně jako výčtové datové typy v programovacích jazycích [21]. Tento datový typ umožňuje staticky specifikovat uspořádaný výčet hodnot, které může daný sloupec obsahovat. V případě skupin hodnot, které by se neměly v systému měnit byla zvolena toto možnost namísto číselníků. Tímto je při konkrétních dotazech ušetřena jedna operace *JOIN*. Toto rozhodnutí vede k omezení možností pro případnou změnu databáze. Jedná se totiž o datový typ specifický pro *PostgreSQL* a například v *MySQL* není podporován [15]. V případě potřeby by byla možnost tento typ částečně simulovat pomocí databázových omezení.

ENUM byl využit pro definici stavů žádostí, organizačních rolí a také druhů pracovních absencí. U zmíněných možností se neuvažuje nutnost jejich pravidlených změn. V případě potřeby je potřeba vytvořit a vykonat databázovou migraci.

5.2.2 Realizace

Pro tvorbu fyzického schématu byla využita knihovna *Knex.js* umožňující generování SQL dotazů. Databázové migrace byly psány v jazyce *JavaScript* podle výše zmíněného databázového návrhu a jejich orchestrace je řešena nástrojem *Knex CLI*. Tento nástroj umožňuje řešit správu migrací včetně přechodů mezi nimi. Také umožňuje generování šablon migrací a cvičných dat. V průběhu vývoje byla databázová struktura postupně upravována, s čímž se pojí vytvoření a aplikace několika migračních souborů. Samotné práci s migracemi je věnována sekce v Příloze C.

5.3 Uživatelské role

V systému existují tři druhy rolí, přičemž pouze dva mají vliv na uživatelská oprávnění. Role v rámci organizace a role u projektové participace jsou popsány níže. Aktuální návrh uživatelských rolí vychází z diskuzí s potenciálními uživateli. Je značně pravděpodobné, že při delším provozu dojde k požadavkům na změny, možná dokonce i na nějakou formu parametrizace.

5.3.1 Organizační role

Návrh organizačních rolí byl inspirován aktuálním systémem uživatelských rolí v rámci komunikačního nástroje *Slack*. Existují tři úrovně uživatelských oprávnění v rámci organizací a to Vlastník (*Owner*), Administrátor (*Admin*) a Člen (*Member*). Uživatel může být současně součástí více organizací zároveň, avšak v každé má právě jednu roli.

Člen může participovat v rámci organizačních projektu a vidí projekty, ve kterých se nachází jako člen. Administrátor může vše co člen s tím rozdílem, že vidí všechny organizační projekty. Dále má také schopnost schvalovat žádosti o abscenci jednotlivých členů. Vlastník má kompletní oprávnění pro činnosti spojených s organizací, schvalovat všechny požadavky v rámci organizace. Navíc ke všemu může vidět celkové a predikované organizační náklady.

Návrh organizačních rolí není dokonalý, ale pro potřeby prototypu, který cílí na společnosti klasifikované jako malé je to po dohodě s vedoucí dostatečné. V případě používání ve větších organizacích by bylo vhodné přidat další role, např. nákupčí, případně umožnit definici rolí vlastních.

5.3.2 Role u projektové participace

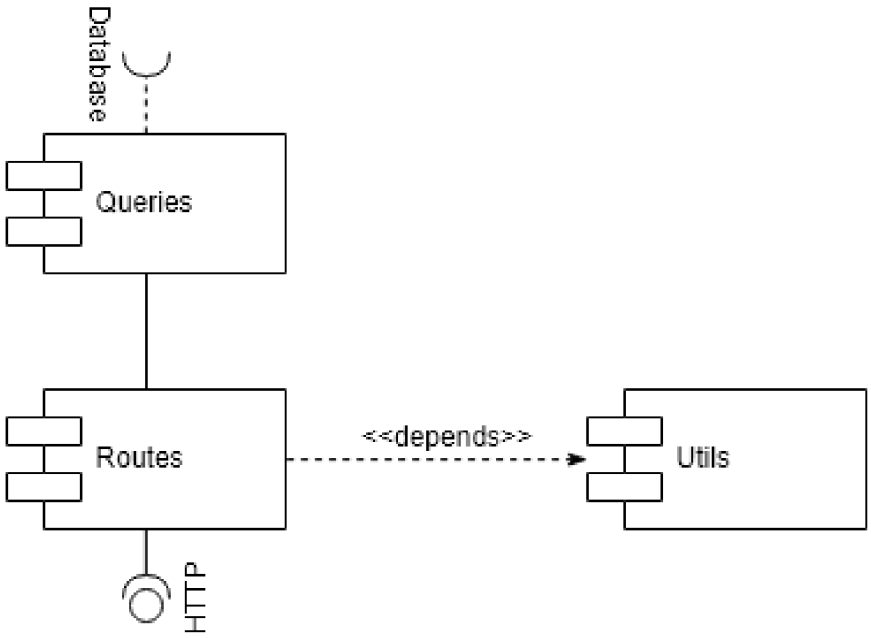
V rámci projektu existují dvě skupiny rolí, projektová a administrační. Projektová role má informativní charakter a určuje kompetence daného člena projektového týmu při realizaci projektu. Tato informace slouží k řízení a organizaci týmu.

Administrační role určuje, zda může daný člen projektového týmu editovat projektové informace, přidávat nové týmové členy, žádat o nákup a provádět analýzu zainteresovaných stran. Možnost vidět finanční informace je odvozeno od organizační role, v případě nedostatečných oprávnění v organizaci jsou zobrazeny pouze časové údaje. Po založení projektu v rámci organizace se zakládající jedinec stává automaticky projektovým administrátorem nezávisle na zvolené projektové nebo organizační roli.

Pokud uživatel participuje na projektu s více rolemi, tak stačí aby měl administrační oprávnění alespoň u jednoho projektového zapojení.

5.4 Sdílení zdrojových kódů mezi vrstvami

Veškeré zdrojové kódy jsou uloženy v rámci jednoho repositáře. Díky zvoleným technologiím usnadňuje tato volba možnosti sdílení datových modelů a pomocných funkcí mezi serverovou a klientskou vrstvou. Pro samotnou správu a verzování zdrojových kódů je použit nástroj *git*.



Obrázek 5.5: Vysokoúrovňová architektura serverové části

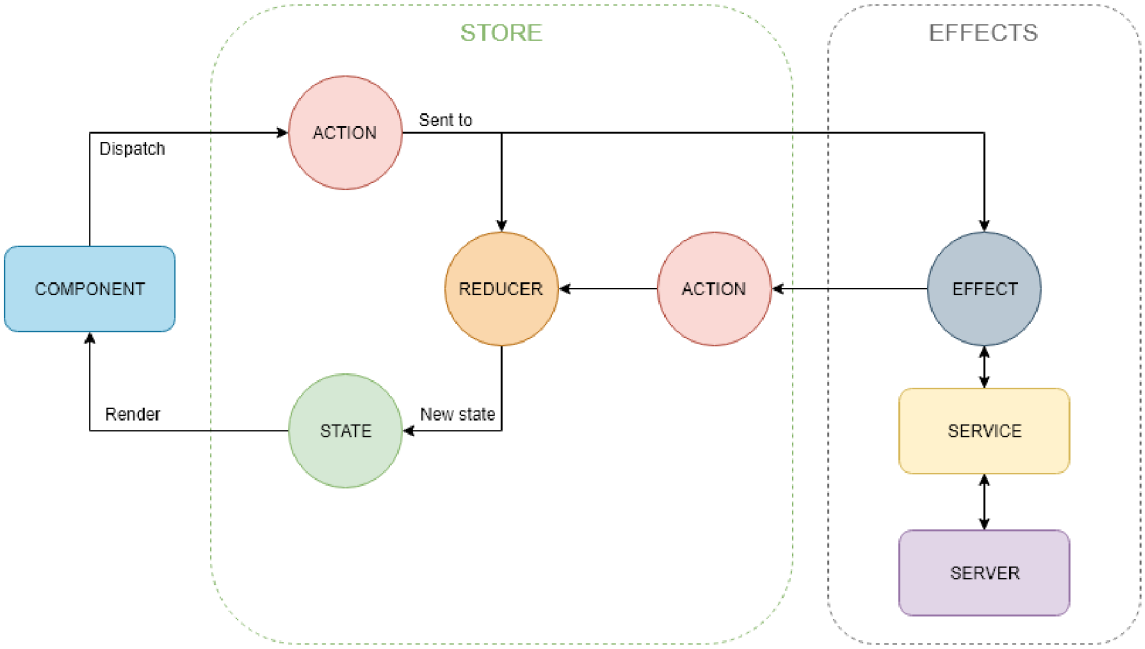
5.5 Serverová část

Serverová část je koncipována jako monolit a využívá k běhu *Node.js*. Poskytované REST API je postaveno s využitím knihovny *Express.js*. Hlavními účely této vrsty jsou poskytování operací nad daty, jejich validace při vkládání a úpravách, zabezpečení přístupů k jednotlivým datům a tvorba datových agregací. Server obsahuje tři hlavní moduly. Modul *queries*, který zaobstarává tvorbu dotazů na databázovou vrstvu. Dále *routes*, ten implementuje funkcionalitu jednotlivých koncových bodů v rámci API rozhraní. Třetím modulem je *utils*, který sdružuje pomocné funkce používané napříč celým projektem. Poslední zmiňovaný modul je kompletně pokryt automatizovanými jednotkovými testy. Architektura serverové části je vyobrazena v Obrázku 5.5. Dokumentace rozhraní ve *swagger* formátu lze vygenerovat pomocí návodu v Příloze C.

5.6 Klientská část

Klientská část je koncipována jako webová *single-page* aplikace, která je implementována s využitím frameworku *Angular* v kombinaci s komponentovou knihovnou *Angular Material*. Komunikace se serverovou části probíhá přes HTTP protokol. Při implementaci klientské aplikace je využito reaktivní programování a také *NgRx Store*, který poskytuje funkcionalitu pro řízení stavu aplikace [43]. Tato knihovna je postavena nad *RxJS*, což je knihovna pro reaktivní programování s využitím *observables*. Toto umožňuje snažší tvorbu asynchronního kódu [20].

Aplikace má jeden globální stav. V rámci aplikace lze vyvolat akce (*Actions*). Tyto akce na základě svého typu a obsahu upravují globální stav. Akce může způsobit i nějaký efekt (*Effect*), což se obvykle využívá k zavolání funkcí s vedlejšími efekty, například HTTP dotaz na API. Koncept je graficky vyobrazen na Obrázku 5.6.



Obrázek 5.6: NgRx Store (inspirováno [49])

Aplikace má připraveny základy pro podporu více jazykových mutací, k dokončení je potřeba provést překlady a ty přidat do překladových souborů. Tyto soubory již existují a část textů je v nich již umístěna.

Komponentizace klientské aplikace je vyobrazena na Obrázku 5.7. Většina těchto komponent je propojena s *NgRx Store*, způsob propojení je zobrazen na Obrázku 5.6

5.7 Shrnutí klíčových funkcionalit projektu

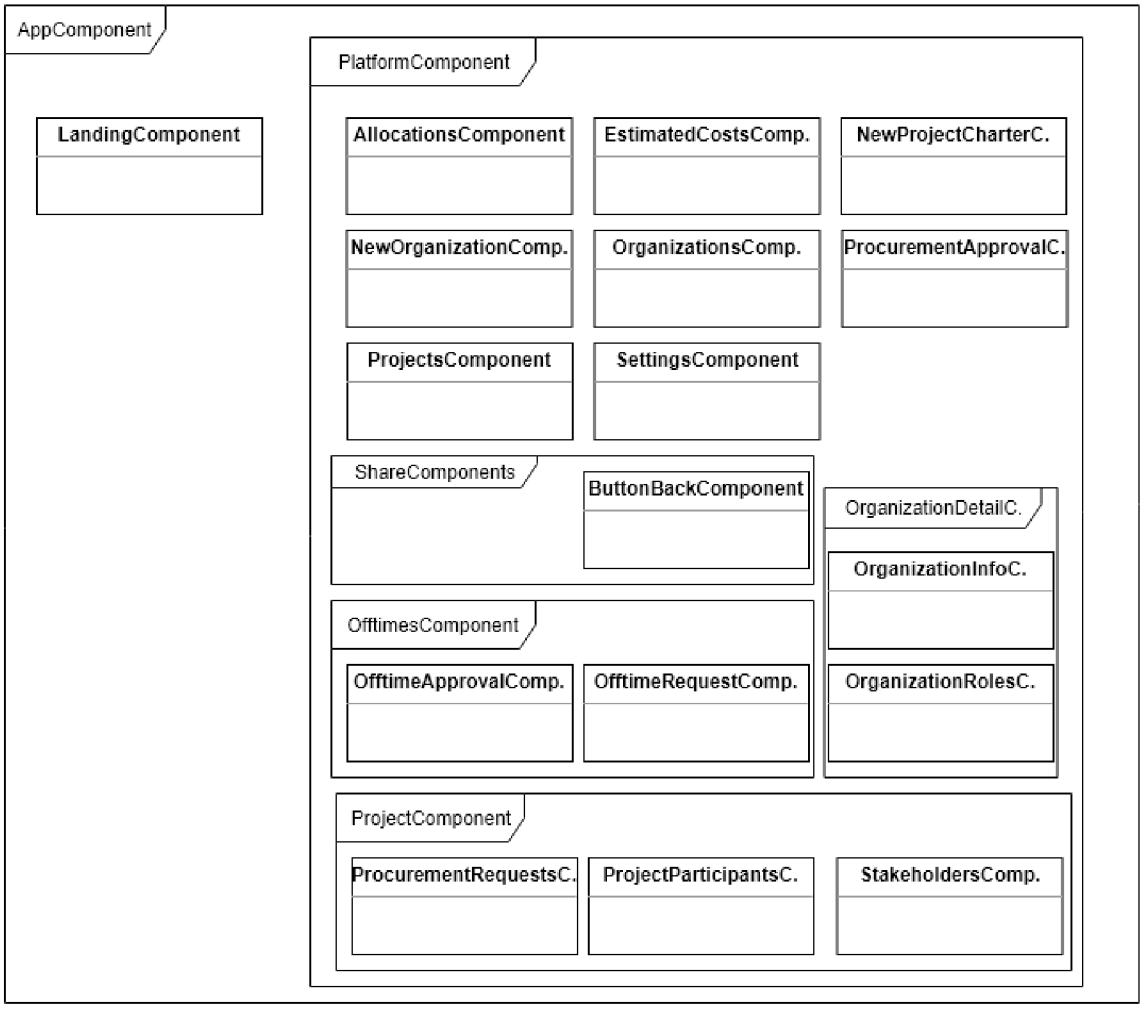
Tato sekce se věnuje popisu klíčových funkcionalit systému, nepopisuje kompletně všechny případy užití.

5.7.1 Přihlášení a registrace

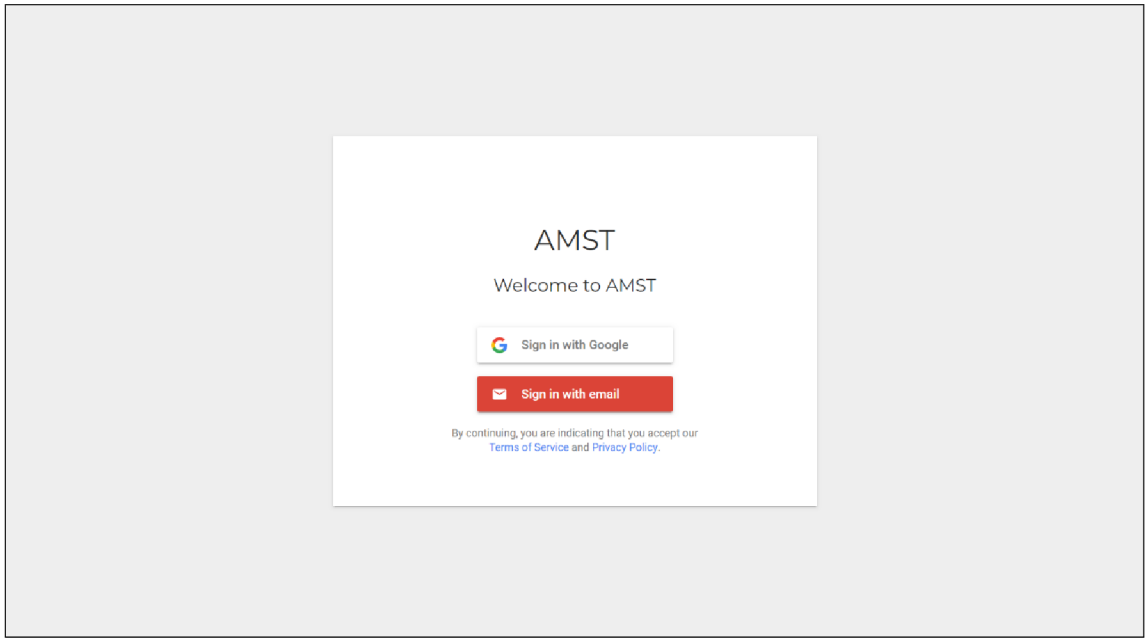
Přihlášení a registrace jsou vstupní bránou do systému, vzhled této části systému lze vidět na Obrázku 5.8.

Realizace této části probíhá napojením na službu *Firebase Auth*. Pro přihlašovací a registrační formulář byla využita komponenta *FirebaseUI-Angular*. Přihlášení do systému je umožněno i s již existujícím *Google* účtem. Tato metoda byla zvolena z důvodu narůstajícího využití *G Suite* napříč společnostmi [16]. Touto volbou bylo umožněno jednotného přihlášení přes již existující firemní účet.

Výzvou bylo problematické napojení přihlašovací komponenty na celkové řízení stavu aplikace. Toto bylo problematické z důvodu nemožnosti využití dynamického definování funkce zpětného volání pro uspěšné přihlášení. Použitá knihovna zapouzdřovala knihovnu *FirebaseUI-web* pro využití v rámci frameworku *Angular*. V důsledku tohoto zapouzdření přestala fungovat možnost dynamické definice potřebné funkce a šlo ji definovat pouze staticky. Tento problém se naštěstí podařilo vyřešit namapováním statického světa do "reaktivního" světa tvořeného systému využitím *RxJS*.



Obrázek 5.7: Komponentizace klientské aplikace



Obrázek 5.8: Přihlašovací obrazovka (snímek obrazovky)

Uživatel se registruje vůči *Firebase Auth*. Po úspěšné registraci je na základě požadavku z klientské aplikace vytvořen záznam v tabulce uživatelů. V případě selhání daného požadavku tento záznam není vytvořen. Uživatelský záznam spojující autentizační službu se systémem, tak v databázi nevznikne. Tento problém je řešen automatickým vytvořením uživatelského záznamu v případě selhání jeho nalezení u právě autentizovaného uživatele. V případě že registrace probíhá na základě pozvánky, tak dochází pouze ke spárování již existujícího záznamu s autentizační službou.

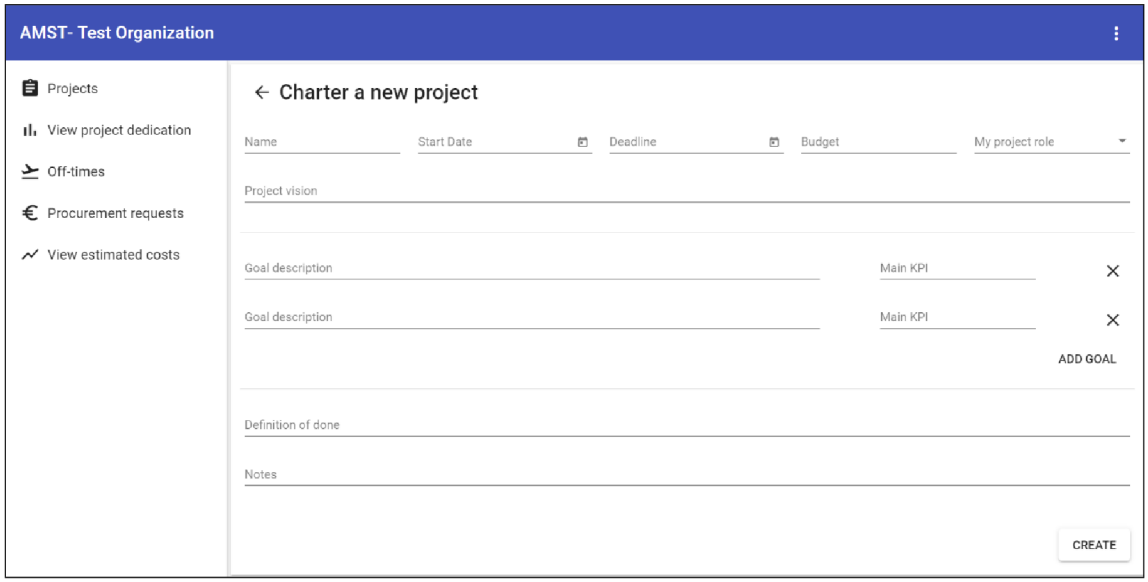
5.7.2 Organizace a práce s ní

Organizace je centrální entitou celého systému. Každý uživatel má oprávnění si založit v systému vlastní organizaci. Do této organizace může přidávat další uživatele podle e-mailové adresy. V případě, že uživatel s touto adresou není ještě registrován, tak se mu odešle e-mail s pozvánkou a také se vytvoří neaktivovaný účet. Tento účet se zaktivní prvním přihlášením/registrací s daným emailem. V ostrém provozu by zde měla být registrace pozvaného uživatele omezená nějakým tokenem, aby nemohlo dojít k tomu, že by se k organizaci přihlásila nějaká neoprávněná třetí strana. S neaktivním účtem lze již dále plánovat a přidávat jej do projektů.

V rámci organizace mají vlastníci možnost definice organizačních kompetencí a ty skládat do organizačních rolí. Tyto role jsou čistě informativního charakteru a usnadňují orientaci vymezením povinností v projektovém týmu. Ke každé organizaci mohou její členové vytvářet projekty.

5.7.3 Inicializace projektu

Při vytváření nového projektu v rámci systému je nutno vytvořit jeho zakládací listinu. Obsah zakládací listiny je přizpůsoben agilním projektům jeho podoba je zobrazena na Obrázku 5.9. Slouží primárně k definici vize, časového rámce a cílů projektu. Takto vytvořený



Obrázek 5.9: Tvorba agilní zakládací listiny (snímek obrazovky)

projekt se následně důležitou částí systému ke které se pojí další funkcionality. Jak plánování kapacit, plánování nákladů, tak celkové řízení zainteresovaných stran.

5.7.4 Analýza zainteresovaných stran

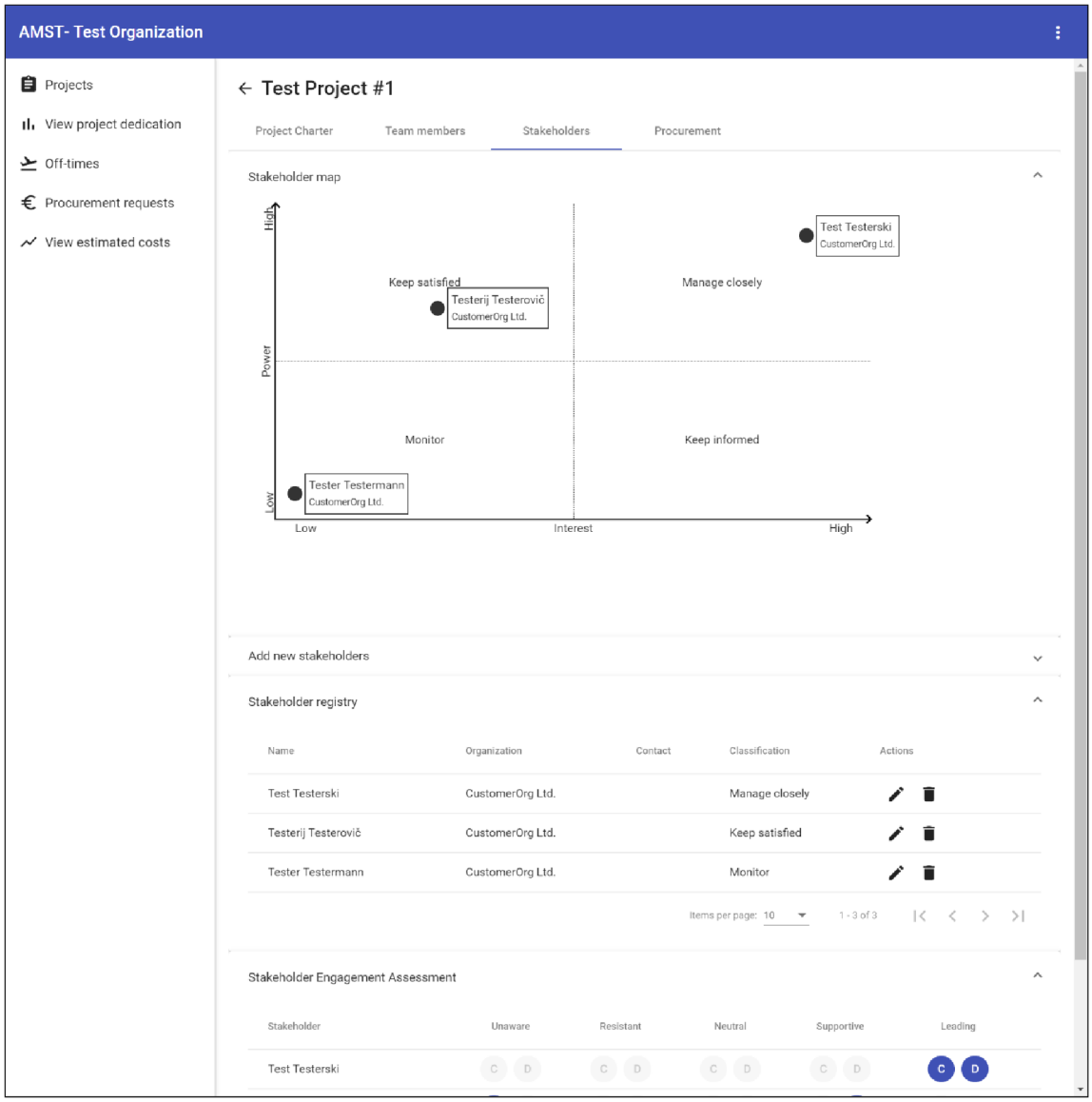
Analýza zainteresovaných stran je nástroj pro relativní porovnání zainteresovaných stran projektu mezi sebou. Tento nástroj je inspirován PMI. Výstupem analýzy zainteresovaných stran je dvojrozměrné uspořádání daných zainteresovaných stran na základě jejich zájmu a vlivu na projekt. Ukázka použití je na Obrázku 5.10.

5.7.5 Proces žádání a schvalování absencí

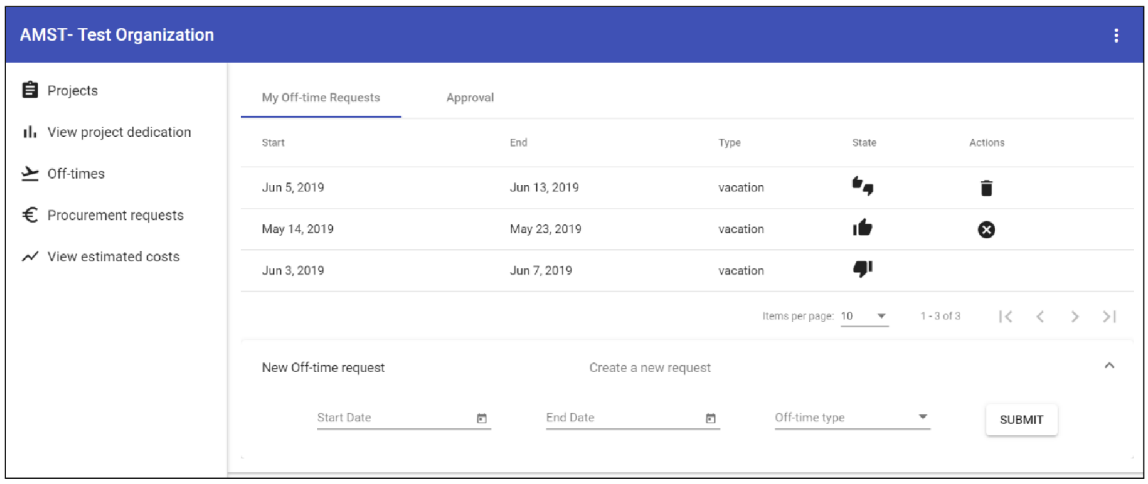
Žádosti o absence a jejich schvalování jsou důležitým doplňkem pro plánování týmových kapacit. Žádosti jsou vždy spojeny s konkrétní organizací a možnosti jejich schvalování se odvíjí od role v rámci dané organizace. Každá žádost má konkrétní typ. Podle typu se mohou promítnout i do očekávaných nákladů, například v případě neplaceného volna se dané období se nebude započítávat do nákladů k danému projektu. Proces schvalování je navržen, tak aby omezoval možnosti zneužití a šikany ze strany vedoucích pracovníků. Stavový diagram popisující změny stavů žádosti o absenci lze vidět na Obrázku 5.12.

5.7.6 Plánování kapacit

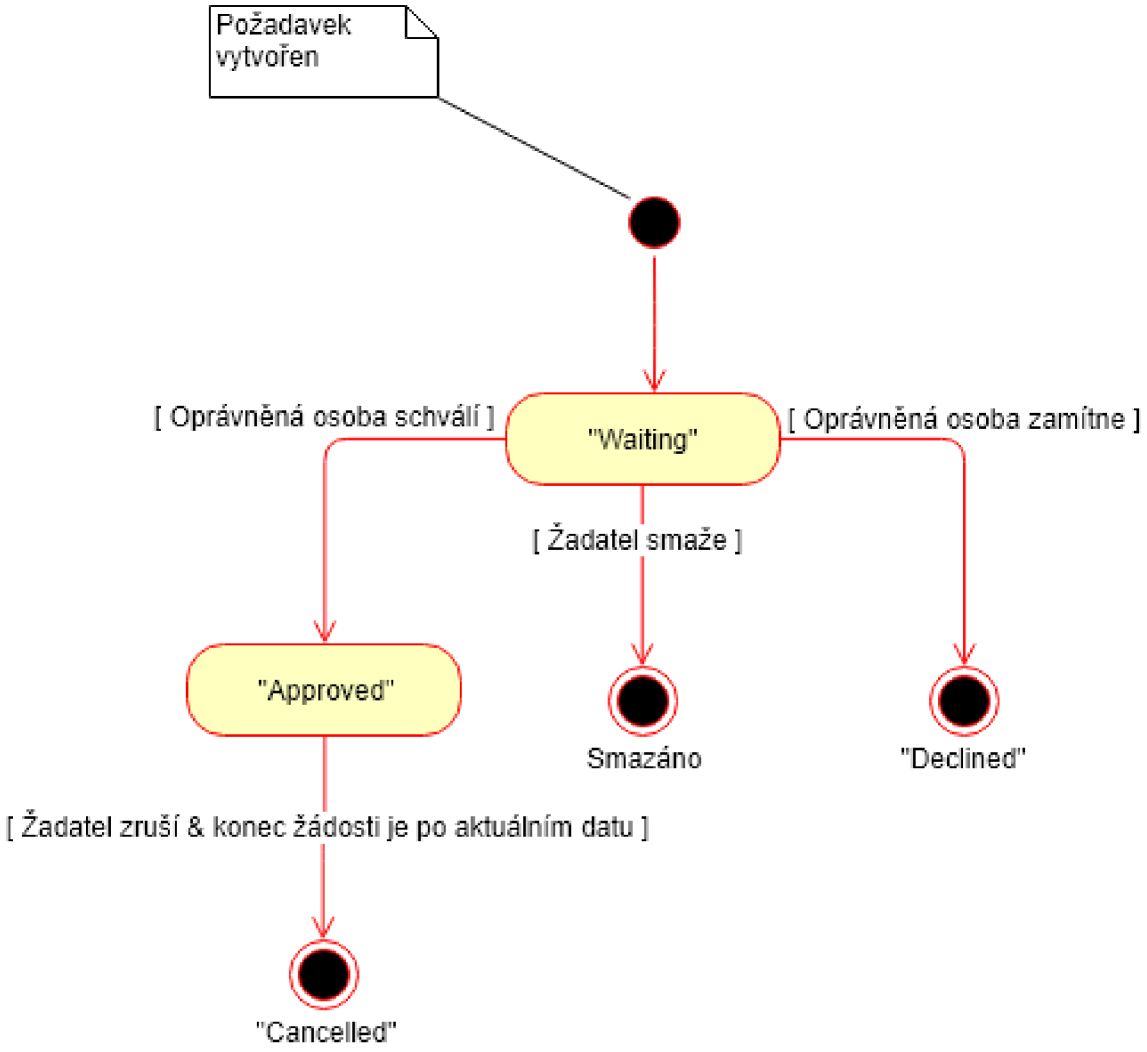
Tato část systému umožňuje přehledný způsob sledování stavu alokovaných vývojových kapacit. Jsou zde agregovány jak plánované alokace, tak i schválené absence. Dohromady poskytují informace k dalšímu plánování. Odpovědné osoby zde mohou plánovat alokaci projektových kapacit. Příklad naplánovaného měsíce v menší organizaci se dvěma paralelními projekty lze vidět na Obrázku 5.13.



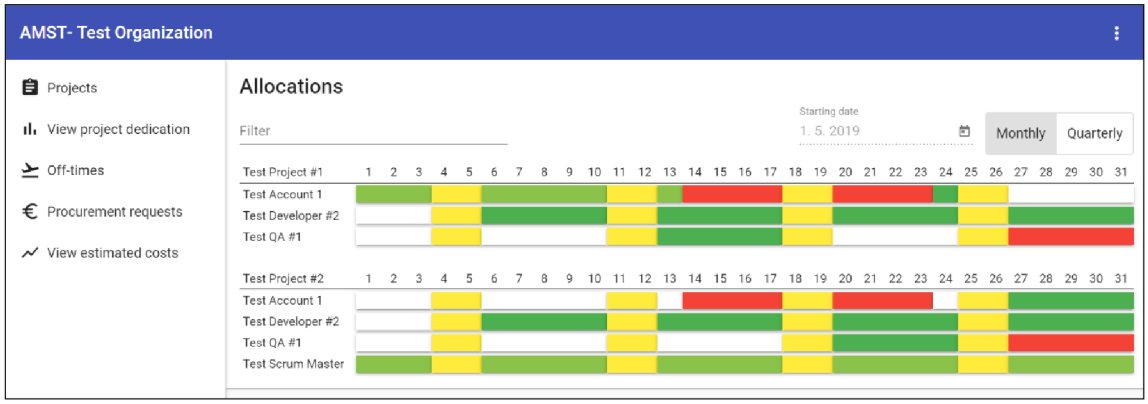
Obrázek 5.10: Nástroje pro práci se zainteresovanými stranami (snímek obrazovky)



Obrázek 5.11: Žádosti o absence (snímek obrazovky)



Obrázek 5.12: Stavový diagram žádosti o absenci



Obrázek 5.13: Příklad měsíčního plánu menší organizace (snímek obrazovky)

5.7.7 Predikce nákladů

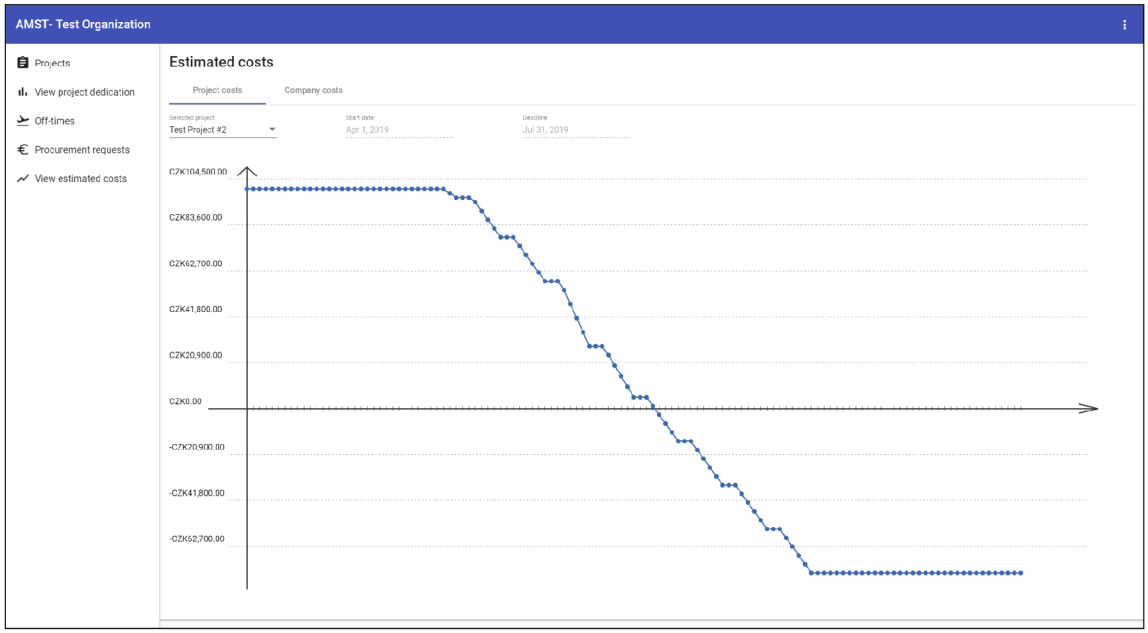
Na základě informací o plánovaných alokacích a případných nákladech v rámci procesu nákupu, lze predikovat očekávané přibližné náklady na daný projekt. Projektové náklady lze společně agregovat do celkových organizačních nákladů na vývoj. Tato část systému poskytuje podporu pro finanční řízení projektu a organizace jako celku. V případě pevně omezeného rozpočtu poskytuje nástroje k jeho hlídání. V rámci budovaného systému není ambice tvořit přesný účetní software, ale spíše nástroj poskytující informace v dostatečné přesnosti k podpoře informovaného manažerského rozhodování.

5.7.8 Autentizace a autorizace

Pro autentizaci a autorizaci je využita služba *Firebase Authentication*. Klientská aplikace se podle zvolené metody autentizuje vůči zmiňované službě. Po úspěšné autentizaci obdrží klient JWT token, který se poté využívá k autorizaci požadavků na REST API.

Autorizaci požadavků řeší *middleware* funkce, která je předřazena všem API koncovým bodům s vyjímkou odmítnutí pozvánky do systému. U všech zpracovávaných požadavků, s vyjímkou HTTP požadavků typu *OPTIONS*, dochází k validaci autorizačního tokenu. Tento token se podle standardu bere z HTTP hlavičky *Authorization* a následně je pomocí *Firebase Admin SDK* validován. V případě úspěchu se autorizační proces posune dál na další autorizační funkci, případně pokud není potřeba u daného koncového bodu kontrolovat další oprávnění, tak se vykoná logika odpovídající danému požadavku. Z autorizačního tokenu lze získat uživatelské *uid*, které je využíváno k identifikaci odpovídajícího uživatelského záznamu v databázi a od toho se odvíjejících uživatelských oprávnění. Pokud je autorizace neúspěšná z důvodu nevalidního tokenu či nedostatečných oprávnění, tak je navrácena odpověď *401 UNAUTHORIZED*, v případě absence tokenu *403 FORBIDDEN*. Diagram popisující proces autorizace lze vidět na Obrázku 5.15.

V rámci prototypu je většina uživatelských omezení řešena na klientské straně a do budoucna je nutnost zabezpečit úplně všechny API koncové body. Ne všechny koncové body pracují se senzitivními daty a z tohoto důvodu to vzhledem k formě testování a testovacího provozu nebylo problematické.



Obrázek 5.14: Nástroj pro predikci nákladů projektu, u zobrazeného projektu lze vidět jasná nutnost navýšení rozpočtu (snímek obrazovky)

5.8 Práce s osobními údaji

V této sekci je nastíněna práce s osobními údaji a jsou zdůrazněny body, které by bylo potřeba vyřešit před ostrým provozem. Komplexní právní analýza a tvorba smluvních podkladů pro budované řešení nebyly předmětem této práce.

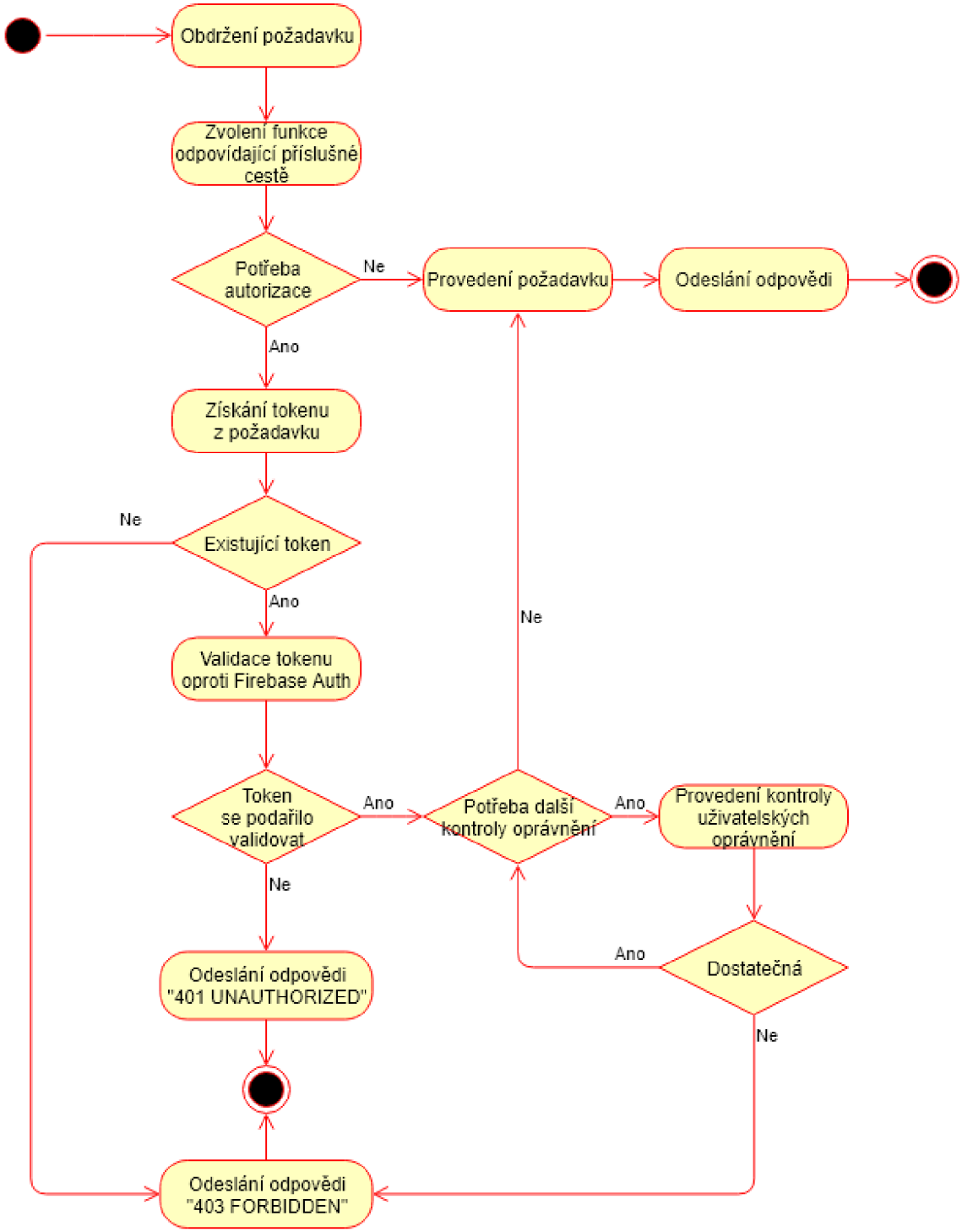
V květnu 2018 vešlo v platnost *Obecné nařízení na ochranu osobních údajů* (dále jen GDPR). Toto nařízení nahradilo v České republice stávající právní úpravu ochrany osobních údajů [53]. GDPR udává povinnosti právnickým osobám bez ohledu na jejich velikost a vztahuje se pouze na ochranu osobních údajů fyzických osob [54]. Dle GDPR se osobním údajem rozumí takový údaj, který může vést k jednoznačné identifikaci osoby.

V systému jsou dvě skupiny osobních údajů. První skupinou jsou údaje patřící přímo uživatelům a druhou skupinu tvoří údaje třetích stran.

V případě ostrého veřejného provozu by bylo nutno v aplikaci doplnit dokumenty *Všeobecné obchodní podmínky* a *Zásady ochrany osobních údajů*, které poskytnou uživatelům bližší informace o jejich právech a poslouží jako základ k poskytnutí souhlasu ke zpracování uživatelských osobních údajů.

5.8.1 Uživatelské osobní údaje

Uchovávané uživatelské údaje jsou jméno, e-mailová adresa a volitelně profilová fotka. Tyto údaje jsou společně uloženy v samostatné tabulce *users* a pro každou organizaci vzniká záznam v tabulce *organization_members* s cizím klíčem daného uživatele. V případě vymazání uživatele ze systému, tak organizace nemusí přijít o historická data spojená s tímto uživatelem. Toto vymazání by mělo z důvodu zachování datové integrity proběhnout vymazáním všech položek z uživatelského záznamu s vyjímkou kromě interního primárního klíče. Alokační intervaly a intervaly absencí by neměly umožnit jednoznačnou identifikaci



Obrázek 5.15: Proces autorizace

fyzické osoby. Z tohoto důvodu jsou daná data v systému ponechána, i když by se v tomto případě hodil i podložený expertní názor.

Při pozvání jiného uživatele do systému je zadávána jeho emailová adresa, která je ve většině případů osobním údajem. V případě obdržení e-mailové pozvánky uživatelem je možnost tuto pozvánku jedním klikem zrušit a údaje týkající se sebe odstranit.

5.8.2 Osobní údaje třetích stran

Osobní údaje třetích stran, které mohou být v systému evidovány jsou jména a kontaktní údaje zainteresovaných stran. V tomto případě je to odlišné, odpovědnost vlastnit souhlas s evidencí těchto údajů je na straně organizace, která systém používá. Na provozovatele budovaného řešení by se pravděpodobně vztahovaly povinnosti *Zpracovatele osobních údajů*. V případě potřeby lze zpracovatelskou smlouvu zakomponovat do *Všeobecných obchodních podmínek*.

5.9 Napojení externích systémů

Díky rozdělení vícevrstevné architektuře je možnost pracovat se systémem bez využití uživatelského rozhraní. Tímto se odemykají možnosti automatizovaného napojení systému do již existujících organizačních procesů. Například lze využívat data o dovolených pro přípravu mzdových podkladů.

V rámci prototypu je nutno pro integraci vytvořit normální uživatelský účet s potřebnými oprávněními a autentizovat jej oproti *Firebase Authetication*. Případně je možnost využít již existující autorizační token jiného uživatele. Do budoucna by šlo uvažovat o implementaci podpory API klíčů. Tato funkcionalita nebyla zmiňována žádným z dotazovaných jedinců a proto nebyla v rámci prototypu implementována.

Kapitola 6

Testování

V průběhu vývoje byly využívány dva rozdílné způsoby testování. Prvním bylo testování automatizované, které sloužilo k ověření správnosti systému. Druhým způsobem bylo testování uživatelské, které sloužilo i k ověření konceptu a jeho případným úpravám.

6.1 Automatizované testování

Cílem automatizovaného testování projektu je zajištění jeho správnosti. Při vývoji byly využívány jednotkové testy a také integrační testy. Do budoucna se uvažuje o rozšíření testovací sady i o automatizované testování ve webovém prohlížeči.

6.1.1 Jednotkové testy

Tento typ testů je v projektu využit na testování pomocných čistých funkcí. Čisté funkce nemají žádné vedlejší efekty a jejich výstup je přímo závislý na jejich vstupu. Tyto funkce byly vyvíjeny přístupem programování řízeného testy. Nejdříve byla vytvořena testovací sada s definovanými vstupy a výstupy na jejichž základě probíhala samotná implementace. V případě nalezení chyby v již implementované funkci došlo k rozšíření testovací sady a následné opravě řešené funkcionality.

K implementaci jednotkových testů byla využita kombinace nástrojů *mocha* a *chai*.

6.1.2 Integrační testy

Integrační testy jsou v případě vyvíjeného systému zaměřeny na serverovou část. Pro jejich tvorbu byl využit nástroj *Postman*. Tento nástroj umožnuje definovat HTTP požadavky a uspořádávat je do kolekcí, které lze spouštět. Také umožňuje definovat testovací kritéria a to s využitím jazyka *JavaScript*.

V případě řešeného projektu se jednalo o pokrytí základních funkcionalit a to vytvoření organizace, přidání organizačního člena, vytvoření projektu, přidání člena projektu a další činnosti spojené s projekty.

6.1.3 Automatizované testování ve webovém prohlížeči

Údržba prohlížečových testů je problematická, jelikož bývají silně závislé na HTML struktuře testované aplikace. Automatizované prohlížečové testy je vhodné používat až v určité vyspělosti projektu [45]. Vytvářený systém je stále ve fázi prototypu a jsou očekávány netriviální změny v rámci uživatelského rozhraní, z tohoto důvodu by investice času do této

formy automatizace nejspíše nepřinesla dostatek hodnoty. Pravděpodobně spíše naopak, jelikož by došlo ke zpomalení dalšího vývoje.

6.2 Uživatelské testování

Cílem tohoto projektu bylo vytvořit nástroj pro podporu řízení agilních vývojových týmů, který opravdu poskytne uživatelům přidanou hodnotu. Vytvořit něco, co bude odpovídat skutečným potřebám uživatelů a bude se jim pohodlně ovládat, není možné bez komunikace a zpětné vazby právě s touto cílovou skupinou. Z tohoto důvodu byla získávaná uživatelská zpětná vazba klíčovým bodem při definici požadavků a validaci konceptu.

Nejdříve probíhaly demonstrace již zmiňovaného *Adobe XD* prototypu, kde byla získávána první zpětná vazba. Následně docházelo průběžně k demonstraci nově přidávaných funkcí. Při větším množstvím implementované funkcionality bylo na malém vzorku provedeno uživatelské testování. Byla sledována uživatelská schopnost orientace na zadané úlohy. Potenciální uživatelé byly schopni se relativně dobře orientovat v systému. Tato skutečnost mohla být způsobena tím, že byli již dříve zapojeni do diskuze projektu. Získané výstupy posloužili k drobným zlepšením uživatelského ovládání. Jako příklad lze uvést přemístění zadávání alokace lidských zdrojů z projektové obrazovky do obrazovky se přehledem alokací. Dále může být zmíněno přemístění přehledu organizací z nastavení do vlastní obrazovky.

6.2.1 Testovací případy

V rámci uživatelského testování byly opakovaně testovány určité testovací scénáře. Ze všech testovacích scénářů jsou jako příklady uvedeny scénáře týkající se přihlašování s registrací (Tabulka 6.1), správy organizace (Tabulka 6.2) a práce se zainteresovanými stranami (Tabulka 6.3).

Testovací případ	**Podmínky**	**Očekávané výstupy**
Registrace nového uživatele	Uživatel není registrován	Uživateli je vytvořen záznam v databázi a je přesměrován do systému
Registrace pozvaného uživatele	Uživatel není registrován a přišla mu pozvánka	Uživatelské *uid* je přiřazeno existujícímu záznamu, jméno s obrázkem jsou aktualizovány a uživatel je přesměrován do systému
Pokus o registraci již existujícího uživatele pomocí emailu	Uživatel se zadanou adresou již existuje	Uživateli není umožněna registrace
Přihlášení pomocí libovolné metody	Uživatel je již registrován	Uživatel je přesměrován do systému
Odmítnutí pozvání do systému	Uživatel obdržel pozvánku	Uživateli je potvrzeno smazání neaktivovaného účtu

Tabulka 6.1: Testovací scénáře pro registraci a přihlášení

Testovací případ	Podmínky	Očekávané výstupy
Vytvoření organizace	Uživatel je přihlášen	Vytvořena nová organizace a uživatel v ní má roli *Owner*
Přidání uživatele do organizace	Uživatel se v organizaci nachází jako *Owner* a zvaný nemá účet	Je odeslána emailová pozvánka a vytvořen neaktivovaný účet, který je přidán do organizace
Změna uživatelské role v organizaci	Uživatel se v organizaci nachází jako *Owner*	Zvolenému členovi je změněna role
Změna vybrané organizace	Uživatel je součástí více organizací	V uživatelském rozhraní se projeví změna a veškerá data odpovídají zvolené organizaci

Tabulka 6.2: Testovací scénáře týkající se správy organizace

Vypsané testovací případy jsou podmnožinou všech uživatelských testů napříč systémem. Tyto testovací scénáře sloužily zpočátku pro zjišťování zpětné vazby, ale v průběhu vývoje se postupně přetransformovaly do seznamu popisujícího korektní funkcionalitu systému. V závěrečných etapách vývoje prototypu sloužily jako kontrolní seznam, zda se při vývoji již implementované funkcionality nerozbily.

Testovací případ	Podmínky	Očekávané výstupy
Přídání zainteresované strany	Uživatel je u projektu evidován jako projektový administrátor	K projektu je přidána nová zainteresovaná strana a automaticky je přidána i do analytických nástrojů
Úprava analýzy zainteresovaných stran	Uživatel je u projektu evidován jako projektový administrátor	Po jakékoliv změně je tato změna perzistentně uložena
Úprava analýzy zapojení zainteresovaných stran	Uživatel je u projektu evidován jako projektový administrátor	Po jakékoliv změně je tato změna automaticky perzistentně uložena
Odstranění zainteresované strany	Uživatel je u projektu evidován jako projektový administrátor	Zainteresovaná strana je odstraněna
Editace zainteresované strany	Uživatel je u projektu evidován jako projektový administrátor	Upravené údaje jsou uloženy

Tabulka 6.3: Testovací scénáře týkající se práce se zainteresovanými stranami

6.3 Testovací provoz

V rámci jedné softwarové společnosti sídlící v Brně proběhlo u jednoho z jejich týmů provozní testování realizovaného řešení. Velikostně je tato společnost klasifikovatelná jako malá, tedy do 49 zaměstnanců včetně. Tato společnost si bohužel nepřála být uvedena konkrétně.

6.3.1 Cíle testovacího provozu

Cílem testovacího provozu bylo ověřit budovaný systém v praktickém použití. Při testování vzniklého nástroje byla pozorována subjektivní použitelnost tvořeného systému. Pro ověření snížení výskytu problémů při interakci se zainteresovanými stranami by bylo potřeba realizovat delší testovací období.

6.3.2 Výstupy testovacího provozu

Získané výstupy byly pozitivní i negativní. Pozitivní bylo, že projektový manažer ocenil specializovanost řešení oproti obecnosti již zmiňovaných běžně používaných systémů. Pozitivní zpětná vazba byla také na přehled alokovaných organizačních kapacit. V rámci testovaného týmu přispěl nástroj k vyřešení neočekávané situace týkající se koordinace absencí značného počtu členů a schůzek s externími stranami. Hlavním přínosem nástroje byla přehlednost prezentovaných informací. Zazněla tu i poznámka, že výrazně větší hodnota by byla v tomto přehledu za předpokladu celoorganizačního nasazazení. V tomto případě by nástroj poskytnul odpovědi i na dostupnost členů jiných týmů, čímž by umožnil sledovat termínový stav jiných projektů a jejich kapacitní alokaci. Na základě těchto informací by bylo snažší pozorovat volné kapacity v rámci organizace jako celku a s tímto se pojící informovanost při komunikaci požadavků na rozšíření kapacity týmu.

U funkcionality věnované řízení zainteresovaných stran byla zpětná vazba vesměs pozitivní. Nástroj pro analýzu zainteresovaných stran byl hodnocen pozitivně, s tím že se jedná o uživatelsky přívětivou možnost provedení této analýzy se snadnou možností aktualizace. U hodnocení zapojení zainteresovaných stran byla poznámka, že ačkoliv v rámci PMI předlohy se jedná čistě o matici s hodnotami 'C' (*current*, v překladu aktuální) a 'D' (*desired*, v překladu žádoucí), tak by možná stálo za zvážení přidání položky ke komentování. Tento podnět by bylo vhodné diskutovat i s jinými testovacími uživateli, jelikož možnost komentování a poznámek již existuje u samotné zainteresované strany. Možnost zobrazení těchto dvou informací společně je však krkolomná.

Při tomto testování se přišlo i na chybu, která z vývojářského hlediska byla zanedbána a tím byla možnost zobrazení všech informací k zainteresované straně. V rámci registru nebyly zobrazeny všechny pořízené informace a ani nebyla možnost je nějak zobrazit.

Problematickým bodem bylo, že při tomto testování vývojáři neviděli v systému přidanou hodnotu a fakticky jej vůbec nepoužívali. V organizaci nyní probíhá schvalování absencí málo organizovanou formou a relativně nahodile, tento proces není v organizaci sjednocen a není zaveden jeden konkrétní proces nebo softwarové řešení. Funkcionalita schvalování absencí nebyla vývojáři využívána. Pravděpodobným důvodem bylo to, že v rámci organizace to aktuálně funguje jinak. Přínosnou zpětnou vazbou byl požadavek na synchronizaci s firemními kalendáři. Oceněnou funkcionalitou by bylo kdyby v případě schválení dané absence existovala možnost automatického přidání události "mimo kancelář"do kalendáře daného jedince. Tímto by odpadla nutnost manuálního přidávání dané absence do kalendáře. Dalším podnětem byla možnost rozšíření systému o export dat dovolených za určité

období. Toto by za předpokladu využití vytvořeného nástroje pomohlo při přípravě mzdových podkladů.

S přihlédnutím k nevyužívání schvalování absencí bylo doporučeno přidat možnost manuálního zadávání absencí pro jednotlivé členy, případně synchronizace s kalendářem i z opačné strany.

Řešení nákupu bylo použito k evidenci několika výdajů vztažených k danému projektu, které byly schváleny mimo systém.

Vytvořené řešení má podle zpětné vazby potenciál a bylo doporučeno jej dál rozvíjet.

Kapitola 7

Závěr

Zpočátku byla práce zaměřena na studium problematiky projektového řízení a agilních metodik. Byly prozkoumány běžně používané podpůrné nástroje. Nálsedně byly definovány požadavky na systém pro podporu řízení agilních vývojových týmů. Tyto požadavky byly na základě uživatelské zpětné vazby postupně upravovány.

Před samotnou implementací byla provedena analýza a volba implementačních technologií. V rámci této analýzy bylo experimentováno s několika různými technologiemi. Na základě jejich srovnání byla zvolena základní skladba technologií. V případě volby implementačních technologií nebyl brán ohled na vývojářskou zkušenost. Toto v úvodu mírně zpomalilo vývoj, jelikož bylo nejdříve potřeba si zvolené technologie osvojit. Zvolená skladba byla využita k implementaci konečného řešení. Následně byl proveden architektonický návrh systému a zahájena implementace.

Vývoj byl řízen metodikou *Kanban*. Tato metodika byla zvolena, jelikož ji lze využít i v jednočlenném týmu. Zpočátku byla jako podpůrný nástroj použita aplikace *Trello*, v průběhu vývoje ji nahradily fyzické papírky na zdi. Zajímavým osobním poznatkem byl celkově lepší a příjemnější subjektivní pocit z práce s fyzickými papírky. Kumulující se množství papírků na zdi ve sloupci *Hotovo* působilo motivačně k dokončování ještě většího množství. Fyzické papírky také způsobovaly menší vyrušení při vývoji, jelikož pro práci s nimi nebylo nutné přepínat okna operačního sytému, ale stačilo se ohlednout za sebe.

Při vývoji byla průběžně získávána zpětná vazba na budovaný systém od potenciálních uživatelů. Takto získané znalosti byly využity k průběžnému upřesňování požadavků a jejich reflektování do tvořeného prototypu. Při diskuzích s potenciálními uživateli byl pozorován poznatek, že v malých společnostech je projektové řízení spíše intuitivní a s velikostí roste formalizace přístupu. Dělat nad touto informací závěry bez hlubšího výzkumu by nebylo rozumné, jelikož právě formalizace přístupu by mohla způsobovat možnost snažšího růstu. Tyto hypotézy je potřeba prozkoumat samostatně.

V průběhu implementace probíhala tvorba testovací sady pro automatizované jednotkové a integrační testování. Po dokončení implementace funkcionality byl proveden testovací provoz, kde byly oveřovány přínosy tvořeného systému. Systém byl v testovacím provozu označen jako přínosný. Zároveň také vzniklo množství podnětů pro rozšíření a další rozvoj systému. Na základě uživatelské zpětné vazby jsou vhodnými možnostmi dalšího rozvoje systému integrace s kalendáři, přístup přes API klíče a větší možnost datových exportů. Další možností by bylo upravit aplikaci pro běh v cloudu. Tato úprava by mohla být značně velká a tedy by bylo nutno ještě zvážit přínos této volby. Před ostrým provozem bude vhodné upravit minoritní detaily pro zlepšení uživatelské zkušenosti a připravit nutné právní podklady.

Vzniklý prototyp je schopnou předlohou nového softwarového produktu, který má potenciál zaujmout uživatele a řešit jejich existující problémy.

Literatura

[1] Passport Documentation. [Online; navštíveno 08.05.2019].
URL http://www.passportjs.org/docs/authenticate/

[2] SQL vs NoSQL: From Oracle to MongoDB, which database is right for your business. Březen 2016, [Online; navštíveno 08.05.2019].
URL https://search.proquest.com/docview/1776617488?accountid=17115

[3] *Agile practice guide*. Newton Square, Pennsylvania: Project Management Institute, první vydání, [2017], ISBN 978-1-62825-199-9.

[4] *A guide to the project management body of knowledge / Project Management Institute*. Newtown Square, PA: Project Management Institute, sixth edition vydání, [2017], ISBN 978-1-62825-184-5.

[5] Raw queries. Leden 2018, [Online; navštíveno 08.05.2019].
URL https://github.com/RobinBuschmann/sequelize-typescript/issues/281

[6] Atlassian: Use Trello and Confluence together. Březen 2018, [Online; navštíveno 08.05.2019].
URL https://confluence.atlassian.com/confcloud/use-trello-and-confluence-together-943978256.html

[7] Auth0: Auth0. [Online; navštíveno 08.05.2019].
URL https://auth0.com/

[8] Auth0: Auth0 Pricing. [Online; navštíveno 08.05.2019].
URL https://auth0.com/pricing

[9] Beck, K.; Beedle, M.; van Bennekum, A.; aj.: Manifest Agilního vývoje software. 2001, [Online; navštíveno 14.01.2019].
URL https://agilemanifesto.org/iso/cs/manifesto.html

[10] Beck, K.; Beedle, M.; van Bennekum, A.; aj.: Principy stojící za Agilním Manifestem. 2001, [Online; navštíveno 14.01.2019].
URL https://agilemanifesto.org/iso/cs/principles.html

[11] Carlson, B.: node-postgres. Březen 2019, [Online; navštíveno 08.05.2019].
URL https://github.com/brianc/node-postgres/blob/master/README.md

[12] Chiusano, P.: The advantages of static typing, simply stated. Září 2016, [Online; navštíveno 08.05.2019].
URL http://pchiusano.github.io/2016-09-15/static-vs-dynamic.html

[13] Doležal, J.: **D10** Vyhodnocování a ukončování projektu. [Online; navštíveno 14.01.2019].
URL http://www.projektmanazer.cz/kurz/soubory/modul-d/d10.pdf

[14] Express: express. [Online; navštíveno 08.05.2019].
URL https://github.com/expressjs/express

[15] Fontaine, D.: PostgreSQL Data Types: ENUM. Květen 2018, [Online; navštíveno 08.05.2019].
URL https://tapoueh.org/blog/2018/05/postgresql-data-types-enum/

[16] Google: 5 Million and counting: how G Suite is transforming work. [Online; navštíveno 08.05.2019].
URL https://cloud.google.com/blog/products/g-suite/5-million-and-counting-how-g-suite-is-transforming-work

[17] Google: Architecture overview. [Online; navštíveno 08.05.2019].
URL https://angular.io/guide/architecture

[18] Google: Firebase. [Online; navštíveno 08.05.2019].
URL https://firebase.google.com/

[19] Google: MATERIAL DESIGN. [Online; navštíveno 08.05.2019].
URL https://material.io/

[20] Google: The RxJS library. [Online; navštíveno 08.05.2019].
URL https://angular.io/guide/rx-library

[21] Group, T. P. G. D.: 8.7. Enumerated Types. [Online; navštíveno 08.05.2019].
URL https://www.postgresql.org/docs/11/datatype-enum.html

[22] Group, T. P. G. D.: PostgreSQL: The World's Most Advanced Open Source Relational Database. [Online; navštíveno 08.05.2019].
URL https://www.postgresql.org/

[23] Haywood, R.: Product Owner – Build the right thing; Development team – Build the thing right; Scrum Master – Build it fast. Everyone is responsible for the product. Srpen 2016, [Online; navštíveno 16.01.2019].
URL https://i2.wp.com/rhaywood.com/wp-content/uploads/2016/01/Scrum-responsibilities.png?w=409

[24] Highsmith, J.: History: The Agile Manifesto. 2001, [Online; navštíveno 14.01.2019].
URL https://agilemanifesto.org/history.html

[25] Jenni, R.: FirebaseUi-Angular. Leden 2019, [Online; navštíveno 08.05.2019].
URL
https://github.com/RaphaelJenni/FirebaseUI-Angular/blob/master/README.MD

[26] Lakeworks: The scrum process. Leden 2009, [Online; navštíveno 16.01.2019].
URL https://cs.wikipedia.org/wiki/Scrum#/media/File:Scrum_process.svg

[27] LePage, P.: Desktop Progressive Web Apps. Květen 2019, [Online; navštíveno 08.05.2019].
URL https://developers.google.com/web/progressive-web-apps/desktop

[28] Machuga, M.: passport-auth0. Říjen 2018, [Online; navštíveno 08.05.2019].
URL https://github.com/auth0/passport-auth0/blob/master/README.md

[29] ManagementMania.com: Fáze projektu (Project phase). [Online; navštíveno 14.01.2019].
URL https://managementmania.com/cs/faze-projektu-project-phase

[30] ManagementMania.com: KANBAN. [Online; navštíveno 14.01.2019].
URL https://managementmania.com/cs/kanban

[31] ManagementMania.com: Plán projektu (Project Plan). [Online; navštíveno 14.01.2019].
URL https://managementmania.com/cs/plan-projektu

[32] ManagementMania.com: Projekt. [Online; navštíveno 14.01.2019].
URL https://managementmania.com/cs/projekt

[33] Method123®: Project Management Guidebook 2018. [Online; navštíveno 14.01.2019].
URL https://www.method123.com/free-project-management-book.php

[34] Microsoft: TypeScript. 2019, [Online; navštíveno 08.05.2019].
URL https://www.typescriptlang.org/index.html

[35] Mishra, D.; Mishra, A.: Complex software project development: agile methods adoption. *Journal of Software Maintenance and Evolution: Research and Practice*, ročník 23, č. 8, 2011: s. 549–564, doi:10.1002/smr.528, https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.528.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.528

[36] MongoDB, I.: Most Popular NoSQL Database. 2019, [Online; navštíveno 08.05.2019].
URL https://www.mongodb.com/scale/most-popular-nosql-database

[37] Palmer, S. R.; Felsing, J. M.: *A Practical Guide to Feature-Driven Development*. Prentice Hall, 2002, ISBN 0130676152.
URL https://www.amazon.com/Practical-Guide-Feature-Driven-Development/dp/0130676152?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0130676152

[38] Patel, J.: Top JavaScript User Authentication Libraries for 2019. Leden 2019, [Online; navštíveno 08.05.2019].
URL http://www.advanceidea.co.in/javascript-authentication-libraries

[39] PMConsulting®: Plánování (projektu). [Online; navštíveno 14.01.2019].
URL https://www.pmconsulting.cz/slovnikovy-pojem/planovani-projektu/

[40] PraizonMedia: Agile Basics for PMBOK Guide 6 Students (PART 1). 2018, [Online; navštíveno 14.01.2019].
URL https://www.youtube.com/watch?v=5PqvbpyAEV0

[41] Prathan, D.: History: Some pictures and PDFs of the Agile Manifesto meeting on 2001. Březen 2018, [Online; navštíveno 14.01.2019].
URL https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a33c40bcc2b

[42] Quan, N. T.: passport-firebase-auth. Říjen 2016, [Online; navštíveno 08.05.2019].
URL https://github.com/bambusoideae/passport-firebase-auth/blob/master/README.md

[43] Roberts, B.: Store. [Online; navštíveno 08.05.2019].
URL https://ngrx.io/guide/store

[44] Schwaber, K.; Sutherland, J.: The Definitive Guide to Scrum: The Rules of the Game. Listopad 2017, [Online; navštíveno 14.01.2019].
URL https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

[45] ScienceSoft: Automated web application testing: PM's guide. [Online; navštíveno 08.05.2019].
URL https://www.scnsoft.com/blog/automated-web-app-testing-pm-guide

[46] Taylor, I. S.: Permit. Duben 2018, [Online; navštíveno 08.05.2019].
URL https://github.com/ianstormtaylor/permit/blob/master/Readme.md

[47] Technologies, C.: Top 3 Best JavaScript Frameworks for 2019. Prosinec 2018, [Online; navštíveno 08.05.2019].
URL https://medium.com/cuelogic-technologies/top-3-best-javascript-frameworks-for-2019-3e6d21eff3d0

[48] Uchytil, A.: *Nástroj pro podporu řízení agilních vývojových týmů*. Brno, 2019. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií.

[49] Vanapalli, S.: Ngrx Introduction and its basic setup with Angular. Srpen 2018, [Online; navštíveno 12.05.2019].
URL https://blog.imaginea.com/ngrx-introduction-and-its-basic-setup-with-angular/

[50] Vargas, R.: PMBOK® Guide Processes Flow – 6th Edition. [Online; navštíveno 14.01.2019].
URL https://ricardo-vargas.com/downloads/download-file/15087/15092

[51] Wells, D.: The Rules of Extreme Programming. 1999, [Online; navštíveno 14.01.2019].
URL http://www.extremeprogramming.org/rules.html

[52] Wells, D.: Extreme Programming: A gentle introduction. Říjen 2013, [Online; navštíveno 14.01.2019].
URL http://www.extremeprogramming.org/

[53] Škorničková, M. E.: Co je GDPR a jak bude aplikováno v Česku. [Online; navštíveno 08.05.2019].
URL https://www.gdpr.cz/gdpr/co-je-gdpr/

[54] Škorničková, M. E.: Jaké povinnosti ukládá GDPR institucím a firmám. [Online; navštíveno 08.05.2019].
URL https://www.gdpr.cz/gdpr/povinnosti/

Příloha A

Obsah přiloženého paměťového média

- /DP.pdf
  - Text diplomové práce
- /tex/
  - LaTeX zdrojové kódy
- /code/
  - zdrojové kódy implementovaného prototypu
- /code/models
  - sdílené aplikační modely
- /code/server
  - serverová část včetně databázové vrstvy
- /code/ui
  - klientská část
- /code/README.md
  - README projektu
- /data/
  - testovací data
- /postman.json
  - integrační testy importovatelné do aplikace *Postman* (www.getpostman.com)

Příloha B

Aplikační požadavky

Vyjma závislostí zmíněných v souborech *package.json*, jsou k zprovoznění systému potřeba následující systémové závislosti. U každé je také zmíněna verze využívaná při vývoji samotném. Samotné zprovoznění projektu je popsáno v Příloze C.

- Node.js - v8.11.3 - https://nodejs.org/en/download/
- NPM - 6.4.1 - zahrnuto v předchozí závislosti
- PostgreSQL - 11.1 - https://www.postgresql.org/download/

Příloha C

Manuál

C.1 Úvodní instalace závislostí

Pro instalaci všech potřebných závislostí je potřeba v příkazové řádce v kořenovém adresáři a následně ve složkách *server* a *ui* zavolat tento příkaz:

```
$ npm install
```

C.2 Konfigurace

Pro správnou funkcionalitu systému je potřeba konfigurovat napojení na externí systémy, konkrétně *Firebase* a *Sendgrid*.

C.2.1 Konfigurae napojení na *Firebase*

V rámci konfigurace je potřeba poskytnout serverové části přístupová oprávnění. V tomto případě je ideální postupovat podle oficiálního návodu https://firebase.google.com/docs/admin/setup. Vygenerované přístupové údaje je nutno umístit do souboru */server/-serviceAccountKey.json*.

V případě konfigurace klientské části je potřeba do konfiguračních souborů */ui/environments/environment.ts* a */ui/environments/environment.prod.ts* umístit odpovídající přístupové klíče. Ideální je postupovat pomocí následujícího návodu https://firebase.google.com/docs/web/setup.

C.2.2 Konfigurace napojení na *Sendgrid*

V souboru */server/sendgrid.json* je potřeba nahradit položku *key* validním API klíčem. Pro jeho vygenerování lze postupovat následovně: https://sendgrid.com/docs/ui/account-and-settings/api-keys/#creating-an-api-key.

C.3 Databázové migrace

K práci s databázovými migracemi je využito rozhraní *Knex.js* pro příkazovou řádku. Tento nástroj je instalován pomocí *npm* globálně. Nejdříve je potřeba v souboru */server/knexfile.js* definovat připojovací řetězec k databázi. Poté je nutno v adresáři */server* použít příkaz:

```
$ knex migrate:latest
```

C.4 Kompilace projektu

Projekt lze kompilovat pomocí příkazu:

```
$ npm run build
```

Buď centrálně z kořenového adresáře nebo každou část zvlášť, tedy ve složkách */server* a */ui*.

C.5 Spouštění projektu

V případě spouštění projektu lze postupovat obdobně jako v předchozím případě, s tím rozdílem, že použitý příkaz je:

```
$ npm run start
```

C.6 Spouštění testů

V případě automatizovaného testování projektu lze postupovat obdobně jako v předchozích případech, s rozdílem, že použitý příkaz je:

```
$ npm run test
```

C.7 Generování *swagger* dokumentace

Dokumentaci API rozhraní lze vygenerovat pomocí použití příkazu:

```
$ npm run swagger
```

Tento příkaz lze použít v kořenovém adresáři a také ve složce *server*. Následně je spuštěna na určitém portu webová služba, která zobrazuje danou dokumentaci. Detaily k připojení jsou vypsány v příkazové řádce.