

ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Študijný odbor: **Informačné systémy - spracovanie dát**

Bc. Maroš Novák

**Integrácia softvérových nástrojov na podporu
riadenia projektov pomocou internetových robotov**

Vedúci: Ing. Roman Hauptvogel, PhD.

Registračné číslo: 396/2016

Žilina 2017

Abstrakt

BC. NOVÁK MAROŠ: *Integrácia softvérových nástrojov na podporu riadenia projektov pomocou internetových robotov* [Diplomová práca]

Žilinská Univerzita v Žiline, Fakulta riadenia a informatiky, Katedra softvérových technológií.

Vedúci: Ing. Roman Hauptvogel, PhD.

Stupeň odbornej kvalifikácie: Inžinier v študijnom programe Informačné systémy - spracovanie dát.

Žilina: FRI ŽU v Žiline, 2017 — 96 s.

Cieľom diplomovej práce bolo vytvoriť internetového robota, ktorý integruje komunikačný nástroj Slack a softvérový produkt na podporu riadenia agilných projektov Jira. Práca sa zameriava na identifikáciu a riešenie problémov agilných (distribovaných) tímov. V úvode sa práca zaoberá problémovými oblasťami riadenia tímu v rámci projektu Fri-Skill. Súčasťou kapitoly je aj definovanie cieľov práce. Druhá časť je venovaná analýze agilného vývoja, popisu metodiky Scrum a termínom, ktoré sa v rámci metodiky vyskytujú. Záver kapitoly sa venuje definovaniu problémových oblastí agilného vývoja. Tretia časť sa zameriava na hodnotenie a analýzu súčasných softvérových riešení, ktoré by mohli pokrývať problémové oblasti. Po vyhodnotení poskytuje informácie o teoretickom pozadí použitých technológií a služieb, ktoré sú využívané pri riešení. Štvrtá kapitola opisuje návrh a implementáciu internetového robota. Zaoberá sa špecifikáciou, návrhom architektúry, popisom technológií a realizáciou až po nasadenie. Posledná časť je venovaná zhodnoteniu cieľov, prínosu pre projekt Fri-Skill a možným rozšíreniam chatbota do budúcnosti.

Kľúčové slová: Slack, Slackbot, Chatbot, Jira, agilný vývoj, metodika Scrum

Abstract

BC. NOVÁK MAROŠ: *The integration of software tools to support project management by web robots* [Diploma thesis]

University of Žilina, Faculty of Management Science and Informatics, Department of software technologies.

Tutor: Ing. Roman Hauptvogel, PhD.

Qualification level: Master of Science in program Information systems - data processing
Žilina: FRI ŽU in Žilina, 2016 — 96 p.

The aim of the thesis was to develop a web robot, which integrates communication tool Slack with the software product to support agile project management Jira. The thesis is focused on identifying and solving problems in agile (distributed) teams. In the introduction, complication areas of team management within university project Fri-Skill are described. This chapter contains the definition of thesis goals as well. The second chapter is devoted to analysis of agile development, description of the Scrum methodology, its concepts and identification of problem areas. The third part contains an evaluation and analysis of existing software tools, which could cover problem areas and theoretical background of technologies and services, which are used in development. The fourth chapter describes the design and implementation of web robot. It contains specification, architecture design, technology stack, implementation and deployment. The last part is devoted to evaluation of goals, resulting in benefits to the project and possible extensions of the chatbot in the future.

Key words: Slack, Slackbot, Chatbot, Jira, agile development, Scrum methodology

Prehlásenie

Vyhlasujem, že som zadanú diplomovú prácu vypracoval samostatne, pod odborným vedením vedúceho diplomovej práce Ing. Romana Hauptvogela, PhD. a použil som len literatúru uvedenú v práci.

V Žiline, dňa 25.4.2017

Bc. Maroš Novák

Pod'akovanie

Týmto by som sa chcel poďakovať môjmu vedúcemu diplomovej práce Ing. Romanovi Hauptvogelovi, PhD., ktorý mi pomáhal pri vypracovaní diplomovej práce jeho odbornými znalosťami. Ďalej by som sa chcel poďakovať Ing. Marekovi Tavačovi, PhD. za jeho konzultačný prínos a pomoc so štruktúrou práce. Za technické rady ďakujem Ing. Dušanovi Dragulovi, ktorý ma zasvätil do tajov node.js.

V Žiline, dňa 25.4.2017

Bc. Maroš Novák

Obsah

Zoznam obrázkov	10
Zoznam tabuliek	12
Zoznam skratiek	13
1 Úvod	14
1.1 Ciele práce	14
2 Analýza agilného riadenia	16
2.1 Agilné metodiky	17
2.2 Princípy metodiky Scrum	18
2.2.1 Roly	18
2.2.2 Artefakty	19
2.2.3 Stretnutia	21
2.3 Nástroje na podporu riadenia agilných tímov	22
2.3.1 Tabuľa úloh	23
2.3.2 Atlassian Jira	26
2.3.3 Slack	27
2.4 Identifikácia problémových oblastí agilného vývoja	28
2.4.1 Komunikácia	28
2.4.2 Daily standup	29
2.4.3 Estimácia	29
2.4.4 Monitorovanie	29
3 Analýza riešenia	30
3.1 Definovanie hodnotiacich kritérií riešení	30
3.2 Existujúce riešenia	32
3.2.1 Samostatné softvérové riešenia	32
3.2.2 Doplnkové moduly	37
3.2.3 Zhrnutie analýzy existujúcich riešení	40
3.3 Technológie a služby	41

3.3.1	Chatbot	41
3.3.2	Parser - Analýza textu	45
3.3.3	Slack API	48
3.3.4	Jira Cloud API	53
3.3.5	Node.js	54
3.3.6	Heroku	55
4	Návrh a implementácia	56
4.1	Definovanie cieľov a podmienok	56
4.2	Návrh funkcií a príkazov	57
4.2.1	Jazyk a identita bota	57
4.2.2	Monitorovanie	58
4.2.3	Estimácia	60
4.2.4	Denné stretnutia - standup	62
4.2.5	Nastavenia systému	66
4.3	Popis technológií	68
4.3.1	Klientská časť	69
4.3.2	Serverová časť	69
4.3.3	Databáza	70
4.3.4	Vývojárske prostredie	71
4.3.5	Verziovací systém	71
4.4	Konceptuálny návrh architektúry	72
4.5	Implementácia komponentov	73
4.5.1	Štruktúra projektu	73
4.5.2	Bot	74
4.5.3	Parser	75
4.5.4	Manager	76
4.5.5	Servisná vrstva	77
4.5.6	Actions	81
4.5.7	Responder	82
4.6	Nasadenie	83
4.7	Overenie riešenia	83

5 Záver	84
5.1 Možnosti rozšírenia	85
Literatúra	86
Prílohy	88
Príloha A	88
Príloha B	90
Príloha C	93
Príloha D	94
Príloha E	95

Zoznam obrázkov

2.1	Manifesto agilného vývoja softvéru	16
2.2	Percentuálne vyjadrenie podielu využívaných agilných metodík podľa prieskumu <i>VersionOne, 2016</i> [3]	17
2.3	Cyklus šprintu	18
2.4	Príklad Burndown grafu zobrazujúci ideálne a reálne ubúdanie story pointov	21
2.5	Papierová verzia tabule úloh	24
2.6	Softvérová verzia tabule úloh v nástroji Trello	25
2.7	Štandardný pracovný postup úlohy v Jire (<i>JIRA User's Guide, 2017</i>) [8]	26
2.8	Multiplatformová chat aplikácia Slack	28
3.1	Konceptuálny diagram prípadov použitia	30
3.2	Webová aplikácia Status Hero (<i>statushero, 2017</i>) [10]	33
3.3	Webová aplikácia Jell (<i>jell, 2017</i>) [11]	34
3.4	Webová aplikácia PlanITpoker (<i>planitpoker, 2017</i>) [12]	35
3.5	Webová aplikácia Team O'clock (<i>teamoclock, 2017</i>) [13]	36
3.6	Slack integrácia GeekBot	37
3.7	Slack integrácia Jirio	38
3.8	Slack integrácia nextup	39
3.9	Príklad komunikácie s prvým botom Eliza	41
3.10	Klasický textový chatbot a digitálny asistent Apple Siri	43
3.11	Všeobecná architektúra chatbota	44
3.12	Príklad spracovania textu z prirodzeného jazyka na štruktúru dát	45
3.13	Príklad entity pre hodnoty zapnutia/vypnutia svetla v platforme api.ai	47
3.14	Príklad zámeru s vlastnou a systémovou entitou v platforme api.ai	47
3.15	Príklad komunikácie so Slack botom	49
3.16	Zobrazenie formátovania jednoduchých správ v aplikácii Slack	51
3.17	Zobrazenie formátovania správy v aplikácii Slack s prílohou aj s prislúchajúcimi atribútmi	52
4.1	Identita komunikačného bota Alica v aplikácii Slack a príklad jej správy	57

4.2	Doménový model monitorovania úloh a členov tímu	58
4.3	Monitorovanie - konverzácia v kanáli	59
4.4	Doménový model estimácie používateľských príbehov	60
4.5	Spustenie hlasovania o používateľskom príbehu - konverzácia v kanáli .	60
4.6	Hlasovanie - súkromná konverzácia	61
4.7	Výsledky hlasovania a nastavenie bodov - konverzácia v kanáli	61
4.8	Doménový model denného standupu	62
4.9	Riadenie denného standupu - súkromná konverzácia	63
4.10	Priebeh denného standupu - súkromná konverzácia	64
4.11	Report člena tímu - konverzácia v kanáli	65
4.12	Sumár denného standupu - súkromná konverzácia	65
4.13	Doménový model bota Alica	66
4.14	Priradenie práv a vypnutie modulu - súkromná konverzácia	67
4.15	Prehľad modulov - súkromná konverzácia	67
4.16	Popis technológií	68
4.17	Konceptuálny návrh architektúry riešenia	72
4.18	Štruktúra projektu	73
4.19	Komunikácia medzi aplikáciou Alica a serverovým komponentom Bot. .	74
4.20	Komunikácia medzi komponentom Parser a službou api.ai	75
4.21	Príklad objektu intent pre denný standup a monitorovanie úloh	76
4.22	Komunikácia komponentu Manager	77
4.23	Komunikácia medzi komponentami servisnej vrstvy	77
4.24	REST komunikácia medzi rozhraním Actions a Slack aplikáciou	81

Zoznam tabuliek

3.1	Hodnotiace kritéria existujúcich riešení	31
3.2	Hodnotenie existujúcich riešení	40

Zoznam skratiek

XP	Extreme programming (Extrémne programovanie)
JQL	JIRA Query Language (Jira dopytovací jazyk)
ALICE	Artificial Linguistic Internet Computer Entity
AIML	Artificial Intelligence Markup Language (Značkovací jazyk pre umelú inteligenciu)
API	Application programming interface (Rozhranie pre programovanie aplikácií)
RegEx	Regular expression (Regulárny výraz)
ML	Machine learning (Strojové učenie)
NLP	Natural language processing (Spracovanie prirodzeného jazyka)
RTM	Real Time Messaging (Komunikácia v reálnom čase)
SDK	Software development kit (Sada pre vývoj softvéru)
JSON	JavaScript Object Notation (Javascript objektový zápis)
URL	Uniform Resource Locator (Jednotný vyhľadávač prostriedku)
RPC	Remote Procedure Call (Vzdialené volanie procedúr)
REST	Representational state transfer (Reprezentačný prenos stavu)
HTTP	Hypertext Transfer Protocol (Hypertextový prenosový protokol)
HTTPS	Hypertext Transfer Protocol Secure (Zabezpečený HTTP)
SSL	Secure Socket Layer
JWT	JSON Web Token
URI	Uniform Resource Identifier (Jednotný identifikátor prostriedku)
I/O API	Input/Output Applications Programming Interface (Vstupno-výstupné API)
PaaS	Platform-as-a-Service (Platforma ako služba)

1. Úvod

V dnešnej digitálnej dobe sa stal softvér neodlučiteľnou súčasťou našich životov. Denne ho využívame v počítačoch, inteligentných telefónoch, hodinkách, autách či domácnostiach. Cesta od návrhu až po nasadenie do obchodu s aplikáciami alebo internetovú doménu však nemusí byť taká priamočiara, ako sa môže na začiatku zdať. Vývoj sa môže zastaviť alebo zaseknúť na prekážkach, o ktorých by si nikto nemyslel, že mohli nastať. Pri návrhu, analýze či implementácii tak často dochádza k situáciám, kedy produkt nie je hotový na čas, nie je kvalitný alebo neobsahuje dohodnuté funkcie. Vývojové tímy sa preto snažia minimalizovať takéto výsledky zavedením pravidiel, nástrojov a postupov. Softvérové spoločnosti si adaptujú rôzne metodiky pri vývoji ich softvéru. Vodopádový či iteratívny spôsob riadenia je bežnou súčasťou veľkých korporátnych firiem ako aj malých tímov. Vznikajú rôzne kombinácie či modifikácie a každá z nich má svoje výhody a nevýhody.

Istou modifikáciou prešiel aj agilný spôsob riadenia, presnejšie jeho metodika Scrum, ktorý je používaný aj na univerzitnom projekte Fri-Skill. Webová aplikácia, na ktorej pracujú študenti fakulty riadenia a informatiky v spolupráci so spoločnosťou GlobalLogic s.r.o. využíva metodiku Scrum v školskom prostredí. Počas jeden a pol roka vývoja na produkte sa objavili rôzne problémy a nekvalitné výsledky. Scrum, ako metodika sa osvedčila práve preto, že sa smerovanie produktu neustále mení a tím musí vedieť flexibilne reagovať na zmeny. Na druhej strane nie je možné pracovať na projekte každý deň alebo v rovnaký čas, čím sa značne narušuje dodržiavanie pravidiel metodiky Scrum. Niekoľko nedostatkov sa podarilo odstrániť komunikačným nástrojom Slack. Pri distribuovaných tímoch, ktoré nepracujú každý deň pod jednou strechou je elektronická komunikácia kľúčová. Členovia tímu tak rýchlejšie odhalili problémové oblasti.

1.1 Ciele práce

Cieľom práce je identifikovať kľúčové a problémové procesy agilného riadenia softvérových projektov a následne ich automatizovať pomocou internetových robotov. Medzi takéto procesy patria napríklad informovanie o kritickom bode alebo automatizované denné standupy priamo v online komunikácii pri distribuovaných tímoch. Využitie tzv.

chatbotov pri spojení komunikačných a agilných nástrojov je motiváciou tejto práce. Podrobnejšie pod-ciele práce sú nasledovné:

- Zadefinovať pojmy súvisiace s agilným vývojom. Popísať jednotlivé súčasti metodiky Scrum ako sú stretnutia, roly a artefakty
- popísať nástroje využívané pri agilnom vývoji
- identifikovať kľúčové udalosti riadenia, možné problémy, ktoré môžu nastať pri jednotlivých procesoch riadenia
- analyzovať existujúce riešenia a nástroje, ktoré môžu riešiť čiastkové problémy
- definovať riešenia jednotlivých kľúčových problémov, za pomoci využitia chatbotov
- navrhnuť funkcie internetového robota, ktoré budú vyplývať z návrhu riešenia
- pri vývoji využiť služby na spracovanie prirodzeného jazyka
- implementovať navrhnutú aplikáciu podľa špecifikácie, ktorá bude spĺňať prípady použitia
- nasadiť riešenie do cloudovej služby
- overiť funkčnosť vytvorenej aplikácie na projekte Fri-Skill

2. Analýza agilného riadenia

Agilné riadenie je sada prístupov a metodík, ktoré pomáhajú organizáciám pri lepšom rozhodovaní tým, že poskytujú interné prostredie pre tímy na podporu ich myslenia a efektivity (*Stellman & Greene, 2014*) [1].

Začiatky agilného prístupu sú často chápané ako reakcia na nevýhody tradičných metód vývoja produktov, ako sú nepružnosť a neschopnosť riešiť nové zmeny a požiadavky zákazníka. Nespokojnosť s tradičnými metodikami viedol rad softvérových vývojárov v roku 1990 k návrhu agilných metodík, ktoré posunuli pozornosť vývoja viac na softvér, ako na jeho návrh, dizajn alebo dokumentáciu.

Vo februári 2001 bola zadaná filozofia agilného vývoja a podpísaná skupinou 17 prispievateľov v oblasti alternatívneho vývoja softvéru. Túto filozofiu dnes poznáme ako 'Manifesto agilného vývoja softvéru' a jej hlavná myšlienka je popísaná na obrázku 2.1 (*Ashmore & Runyan, 2014*) [2].

Manifesto agilného vývoja softvéru

Ľudia a komunikácia sú viac než *procesy a nástroje*.
Funkčný softvér je viac než *vyčerpávajúca dokumentácia*.
Spolupráca so zákazníkom je viac než *dojednanie zmluvy*.
Radšej reagovať na zmenu než *sa držať plánu*.

Aj keď časť napravo je dôležitá,
viac si ceníme ľavú časť.

Kent Beck	James Grenning	Brian Marick
Mike Beedle	Jim Highsmith	Robert C. Martin
Arie van Bennekum	Andrew Hunt	Steve Mellor
Alistair Cockburn	Ron Jeffries	Ken Schwaber
Ward Cunningham	Jon Kern	Jeff Sutherland
Martin Fowler		Dave Thomas

Obr. 2.1: Manifesto agilného vývoja softvéru

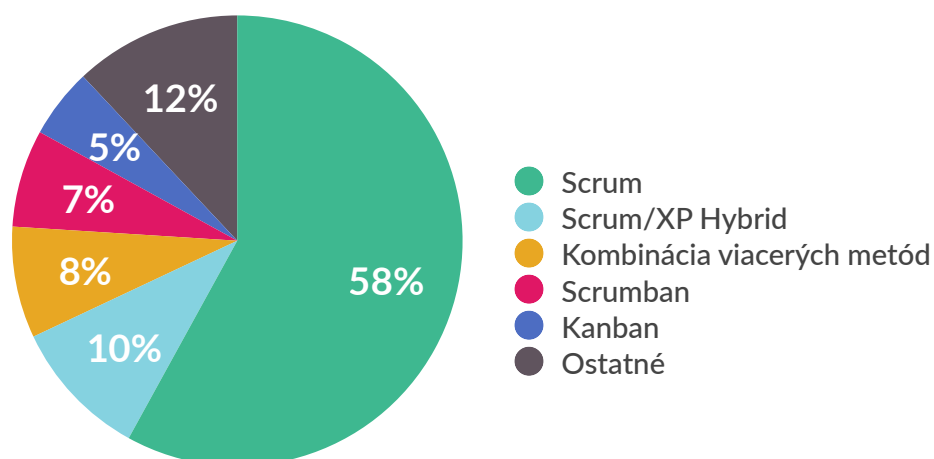
Od tej doby, vzbudzuje agilný prístup veľký záujem medzi technologickými firmami. V súčasnosti je veľmi populárnym iteratívnym prístupom k vývoju softvéru. Snaží sa totiž o zavedenie väčšej flexibility k vyhoveniu požiadaviek projektu a sľubuje určité

riešenia problémov, ktoré sa vyskytujú pri vývoji softvéru pri tradičných prístupoch, ako napríklad skrátenie času a nákladov, kvalitnejší softvér alebo zvýšená spokojnosť používateľov (*Ashmore & Runyan, 2014*) [2].

2.1 Agilné metodiky

Agilné metodiky sú procesy, ktoré nasledujú filozofiu agilného vývoja. Agilné prístupy sa skladajú z rôznych postupov. Tieto postupy existovali mnoho rokov, ale agilný princíp ich kombinuje jedinečným spôsobom do ucelených skupín. Rôzne agilné organizácie používajú rôzne agilné nástroje, metodiky alebo techniky, aby sa viac prispôbili zmenám. *Ashmore & Runyan* [2] veria, že neexistuje univerzálny agilný prístup, ktorý poskytuje úspešne riešenia každého projektu. Preto organizácie používajú kombináciu agilných nástrojov a metodík, ktoré najlepšie vyhovujú ich cieľom a zámerom. Hlavnou výhodou agilného vývoja je, že klient dokáže sledovať prírastok po kratších časových obdobiach a tým tak dokáže lepšie reagovať na jeho zmeny priorít a požiadaviek.

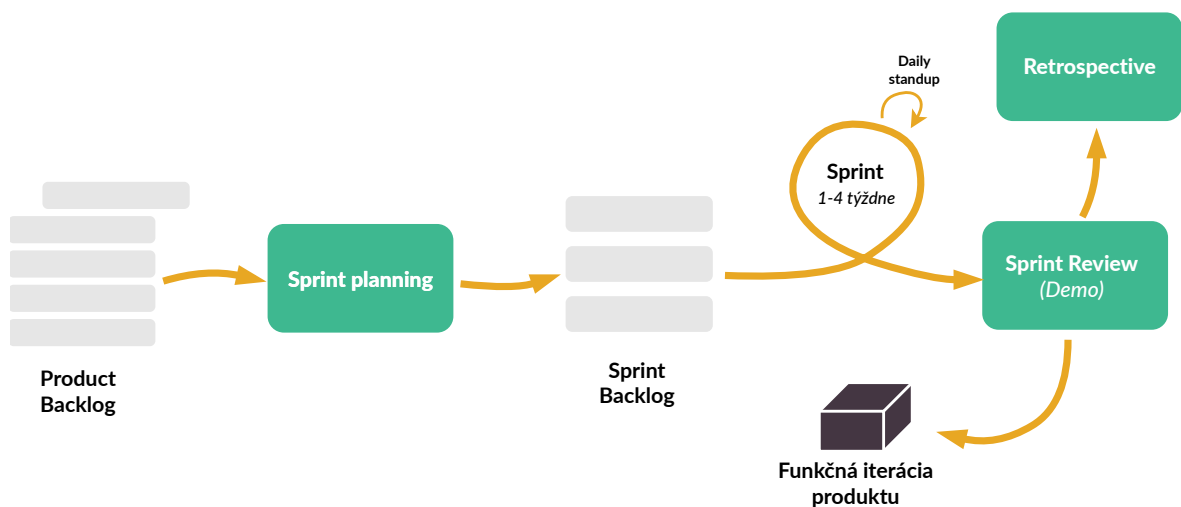
Extrémne programovanie (XP) a **Scrum** sú dve najbežnejšie agilné metodiky. Riadenie projektu s využitím rámca Scrum sú často kombinované s viac praktickými inžinierskymi technikami XP, ktoré môžu fungovať bok po boku (*Stellman & Greene, 2014*) [1]. Graf na obrázku 2.2 ukazuje, aký veľký je podiel firiem, ktoré využívajú Scrum v porovnaní s inými agilnými metodikami.



Obr. 2.2: Percentuálne vyjadrenie podielu využívaných agilných metodík podľa prieskumu *VersionOne, 2016* [3]

2.2 Princípy metodiky Scrum

Pre tímy, ktoré využívajú klasické princípy riadenia, ako napríklad vodopádový prístup, môže predstavovať Scrum zásadnú zmenu vo vývoji. Scrum je založený na samo-organizovaných tímoch, ktoré sú vedené k viac neformálnej komunikácii a spoločnej zodpovednosti za produkt. Je dôležité zvoliť správnych členov tímu, ktorí budú udržiavať tímovú prácu dostatočne efektívnu. Obrázok 2.3 popisuje proces vývoja, ktorý sa bežne označuje ako **cyklus šprintu**.



Obr. 2.3: Cyklus šprintu

Šprint (Sprint) je označenie používané v metodike Scrum na popísanie vývojového obdobia. Vo väčšine prípadov je toto obdobie ohraničené od jedného týždňa po jeden mesiac. Vývoj projektu sa tak skladá z viacerých kratších šprintov, počas ktorých sa pracuje na vyčlenených používateľských príbehoch.

2.2.1 Roly

Metodika Scrum je postavená na troch pilieroch: rolách (roles), stretnutiach (ceremonies) a artefaktoch (artifacts). Táto kapitola objasňuje rozdieli medzi rolami metodiky ako sú tím, vlastník produktu, Scrum master a zákazník. Roly v metodike Scrum nie sú chápané ako pozície, ale sú skôr previazané so zodpovednosťami jednotlivých rolí vo vývoji.

Tím

V rámci metodiky Scrum sa usiluje o vytvorenie tímu, v ktorom majú členovia rôznu kvalifikáciu a sú zameraní na rozličné oblasti. V kontexte vývoja to znamená, že tester pracujú v tíme s dizajnérmi a vývojármi bez hierarchickej štruktúry, kde spoločne plnia cieľ: dodať kvalitný produkt, napr. softvér. Tím sa zaväzuje, že v rámci sprintu dodá vybrané používateľské príbehy. V Scrum platí záväzok pre celý tím. Ak sa tímu nepodarí udržať záväzok, každý člen tímu je zodpovedný, to znamená, že jednotlivých členov tímu nemožno viniť.

Vlastník produktu

Vlastník produktu predstavuje most medzi zainteresovanými stranami (stakeholders) a balansuje ich očakávania. Zastupuje obchodnú časť a jeho hlavnou úlohou je prinášať biznis hodnotu písaním používateľských príbehov a udržiavať product backlog aktuálny. Aj keď sa často považuje za člena tímu, nemal by zasahovať do technických otázok vývoja.

Scrum master

Človek, ktorý predstavuje rolu Scrum mastra je zodpovedný za pomoc pri plnení cieľa. Odstraňuje problémy, ktoré sa vyskytujú počas vývoja u jednotlivých členov tímu, pomáha pri ich rozhodnutiach a podporuje pri vývoji.

Zákazník

Zákazník je osoba, ktorá financuje projekt. To predstavuje jeden z kľúčových rozdielov medzi jeho úlohami a úlohami vlastníka produktu. Na rozdiel od vlastníka produktu, zákazník často nevie, čo chce a vlastník produktu mu pomáha s definovaním požiadaviek.

2.2.2 Artefakty

Druhým pilierom Scrum-u sú artefakty. Artefakty sú prostriedky informácií. Sú to používané pojmy a definície jednotlivých kľúčových prvkov metodiky Scrum.

Vízia

Vízia je východiskovým bodom všetkých projektov. Vlastník produktu vytvára víziu a jeho hlavným cieľom je objasniť tímu, čo má byť výsledkom a ako to má byť dosiahnuté. Kvalita vízie rozhoduje, či sa projekt môže stať úspešným, alebo nie. Dobrá vízia je krátka a výstižná.

Používateľský príbeh (User story)

Každý používateľský príbeh je reprezentácia funkcie produktu. V podstate možno povedať, že používateľský príbeh je jedna veta, ktorá má vopred stanovenú štruktúru: *Ako **používateľská rola**, potrebujem **funkciu**, ktorá mi prinesie **biznis hodnotu**.* napríklad:

*Ako **čakník**, potrebujem **prehľad objednávok**, vďaka ktorému **môžem efektívnejšie obsluhovať zákazníkov**.*

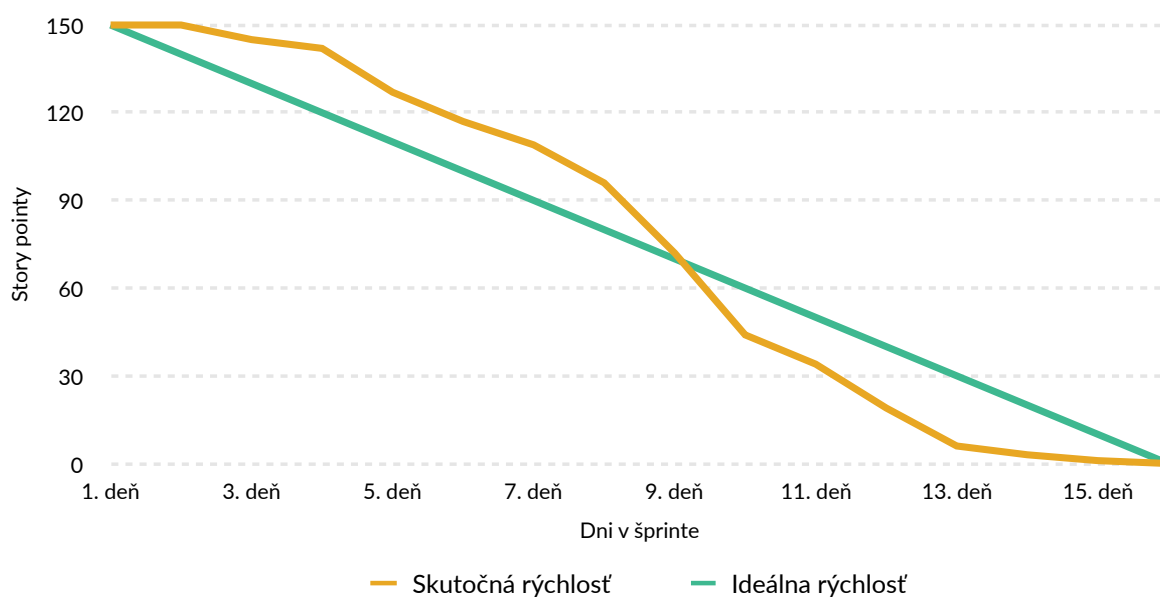
Vlastník produktu využitím tejto štruktúry hovorí tímu príbeh, ktorý im pomáha pochopiť, čo daná funkcia presne vyjadruje a aký je jej prínos.

Product backlog

Mnoho spoločností začína vývojový proces tak, že zhromaždia všetky vlastnosti, ktoré má ich aplikácia alebo softvér obsahovať. Neuvedomia si však, že veľký počet funkcií aplikácie nikdy nebude využitých, nebude fungovať správne alebo budú len zbytočným vyhodnotením zdrojov. Backlog je prioritizovaný zoznam používateľských príbehov. Tento zoznam a priority sa môžu meniť počas celého vývoja. Pri vytváraní backlogu je podstatné sa zamyslieť, čo prináša produktu hodnotu a čo nie.

Burndown graf

Burndown graf (obr. 2.4) je hlavnou metrikou merania, ktorá sa používa v rámci metodiky Scrum. Je dvojrozmerná a zobrazuje množstvo **bodov používateľských príbehov** (story point) aktuálneho šprintu na osi y a množstvo dní súčasného šprintu na osi x. Po každom dni je krivka horizontálne. Akonáhle sa používateľský príbeh dokončí, krivka začína klesať. Táto čiara je taká dlhá, ako množstvo bodov príbehov, ktoré boli ukončené. V ideálnom prípade krivka dosiahne nulu na konci šprintu, čo znamená, že všetky story pointy boli dokončené v predpokladanom čase.



Obr. 2.4: Príklad Burndown grafu zobrazujúci ideálne a reálne ubúdanie story pointov

Burndown graf môže byť použitý na určenie rýchlosti tímu (tzv. rýchlosť v metodike Scrum) napr. koľko používateľských príbehov je možné splniť v rámci jedného šprintu.

2.2.3 Stretnutia

Stretnutia sú posledným pilierom metodiky. Pomáhajú posilniť tím a efektívne implementovať Scrum.

Sprint planning

Pred začiatkom každého šprintu sa tím stretne a spoločne plánuje, čo bude obsahom ďalšej iterácie. Princíp plánovania spočíva v hodnotení používateľských príbehov bodmi, ktoré určujú ich komplexnosť (zložitosť, časovú náročnosť, ..). Podľa počtu bodov, dokážu spoločne odhadnúť trvanie implementácie, množstvo práce a môžu lepšie plánovať zaradenie používateľských príbehov do nasledujúceho šprintu. Jednou z metód plánovania je aj tzv. **planning poker**. Scrum master vyberá postupne používateľské príbehy, ktoré jednotliví členovia tímu hodnotia bodmi. Každý príbeh sa pred hodnotením analyzuje, aby sa predišlo prípadným nedorozumeniam, alebo skrytým problémom, ktoré môžu pri implementácii nastať. Po analýze dá Scrum master pokyn pre spustenie hlasovania. Na hlasovanie slúžia kartičky, ktoré majú hodnoty napríklad 0, 1, 2, 3, 5,

8, 13, 20, 40, 100. Každý člen ukáže svoj odhad bodovej náročnosti. Tím má možnosť ukázať aj kartičku s otáznikom, pokiaľ sa nevie rozhodnúť, alebo kartičku s kávou, ak už potrebuje prestávku. Cieľom tejto 'hry' je, aby sa všetci zhodli na jednej hodnote a tá bude priradená používateľskému príbehu. Používateľské príbehy sú vyberané podľa najväčšej priority, aby klient obdržal najväčšiu hodnotu produktu čo najskôr. Na konci stretnutia sú príbehy zhromaždené v sprint backlogu.

Denné stretnutie (Daily standup)

Každý deň sa počas šprintu stretnú členovia tímu na 10 až 15 minút. Počas stretnutia každý člen povie ostatným čo dosiahol od posledného stretnutia, na čom bude pracovať a či nemá pripomienky, ktoré by ho mohli blokovať. Výhodou stretnutí je rýchle odhalenie problémov a následná reakcia na zabezpečenie úspešného šprintu.

Sprint review (Demo session)

Sprint review je dôležitou súčasťou z viacerých dôvodov. Prezентuje vlastníčkovi produktu a klientom, čo tím dosiahol za posledný šprint. Vlastník produktu vedel, na čom sa bude pracovať, ale až na tomto stretnutí vidí prezentované výsledky. Klient môže predniesť svoju spätnú väzbu, ktorá často vedie k definovaniu nových používateľských príbehov.

Sprint retrospective

Poslednou súčasťou šprint cyklu je retrospektíva. Narozdiel od sprint review, ktorého účelom je zlepšovať produkt, je pri retrospektíve kľúčové zlepšovať tím a jeho procesy. Členovia tímu diskutujú o slabých a silných stránkach v predchádzajúcom šprinte a snažia sa zdefinovať, v čom by sa mali zlepšiť a v čom sú naopak ich kvality.

2.3 Nástroje na podporu riadenia agilných tímov

V tejto časti sa zameriame na nástroje využívané počas vývoja v agilných tímoch. Prehľad o úlohách a aktuálny stav vývoja je kľúčový pre úspešné ukončenie šprintu. Na tento účel existuje v metodike Scrum **tabuľa úloh** (task board).

2.3.1 Tabuľa úloh

Úloha tabule je reprezentácia používateľských príbehov s ich zodpovedajúcimi úlohami v súčasnom šprinte. Scrum definuje tri stĺpce (*Wirdemann, 2011*) [4].:

1. **To Do**, kde sa nachádzajú všetky úlohy patriace do určitého používateľského príbehu, ktoré sú naplánované, ale ešte sa nezačali.
2. **In Progress**, zahŕňa všetky úlohy, na ktorých sa aktuálne pracuje.
3. **Done**, ktorý obsahuje úlohy ukončené v šprinte.

Na základe tejto štruktúry, bežnejší názov pre šprint backlog je Scrum tabuľa úloh. Za ideálnych okolností by mal tím dokončiť prvý používateľsky príbeh pred začatím práce na druhom. Použitím tabule úloh, je jednoduchšie sledovať stav aktuálneho šprintu. Členovia tímu sú schopní samostatne rozhodovať o tom, na ktorých úlohách chcú pracovať - nikto nepriradí úlohu konkrétnej osobe. Táto metóda sa nazýva aj pull a je protikladom push metóde, pri ktorej sú úlohy priradované. Akonáhle sa rozhodnú pracovať na úlohe, musia ju presunúť do stĺpca 'In Progress'. Ak je úloha dokončená, znova sa presúva, tentokrát do stĺpca 'Done'. V prípade, že sú všetky úlohy hotové, mal by byť hotový aj používateľsky príbeh. Ak tomu tak nie je, tím zabudol pridať niektoré úlohy. Hotové používateľské príbehy by mali byť pripravené na predvádzanie a plne funkčné. Podľa potrieb tímu môžu vznikať aj rôzne alternatívy stĺpcov, napríklad 'Testing' alebo 'In review'.

V súčasnej dobe existujú dva typy reprezentácie tabule úloh. V časoch keď vznikol Scrum, jediným médiom bol papier. Avšak časom sa tabuľa úloh nevyhla digitalizácii a dnes existuje veľký počet softvérových nástrojov.

Papierová verzia

Papierový variant tabule úloh je reprezentovaný na obrázku 2.5. Samotná tabuľa je často reprezentovaná prázdnu stenou, sklom alebo pomocou veľkého papiera. Zvyčajne sa nachádza v miestnosti spolu s členmi tímu, aby každý mohol vidieť aktuálny stav. Pri každodennom stretnutí sa členovia zhromaždia okolo tabule úloh a ich cieľom je popísať aktuálny stav. Ak chce člen tímu priradiť úlohu sebe, napíše svoje meno na papierik, aby každý člen vedel, kto má úlohu priradenú a presunie ho do 'In Progress'

stĺpca. Tento princíp tabule s papierikmi je stále preferovaný aj v dnešnej dobe. Avšak má svoje nevýhody, ktoré viedli k druhému typu vizualizácie tabule úloh: softvéru.



Obr. 2.5: Papierová verzia tabule úloh

Softvérová verzia

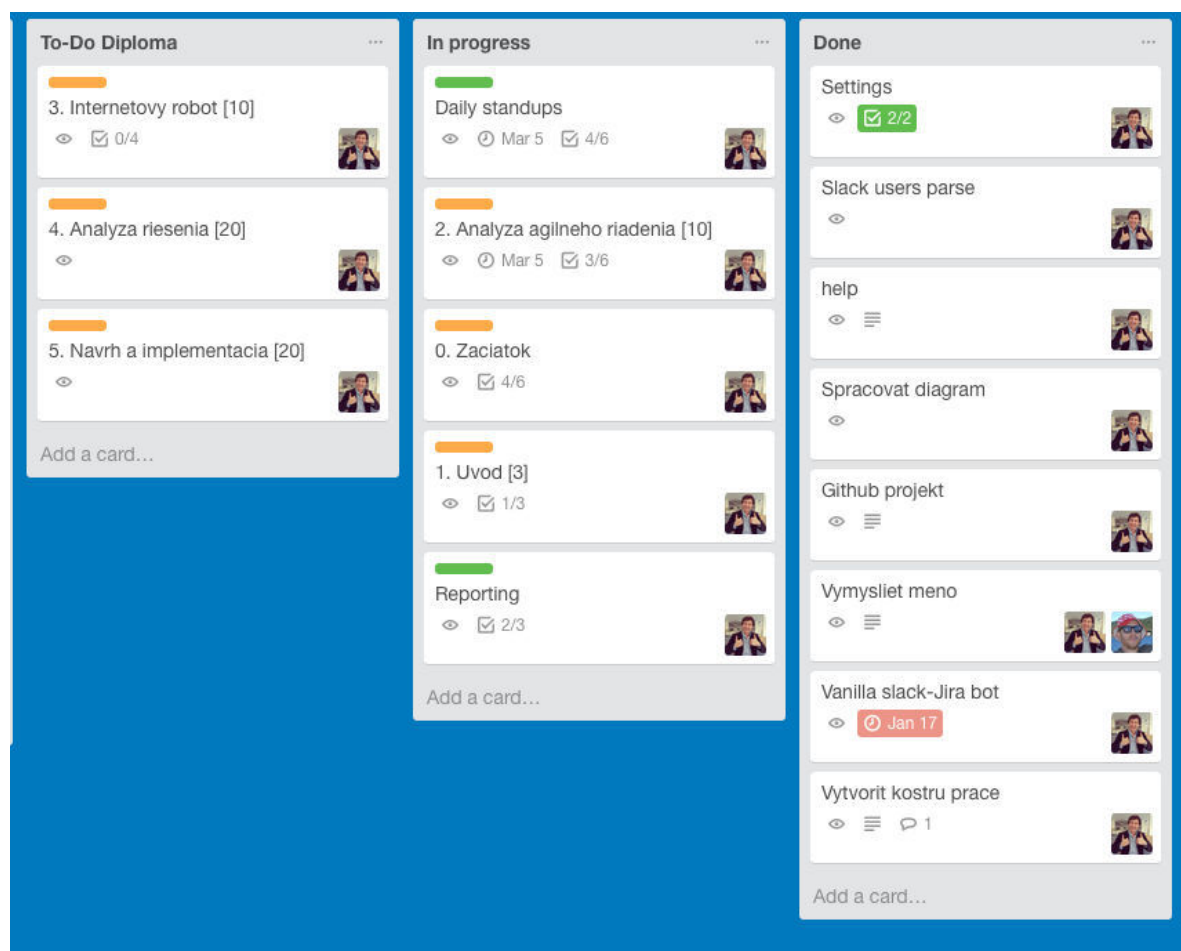
Digitalizácia papierového variantu riešila jeden zo zásadných problémov tabule úloh: distribuované tímy. Pokiaľ nemali všetci členovia tímu prístup k aktuálnej tabule úloh a nepracovali všetci v jednej miestnosti, mohlo to priniesť mnoho problémov do procesu vývoja. Softvér tak ponúkol zdieľaný priestor, ku ktorému každý z členov mohol pristupovať nezávisle od jeho pracovného pôsobiska. Na obrázku 2.6 sa nachádza tabula úloh v nástroji Trello.

Väčšina týchto nástrojov je v dnešnej dobe spustiteľná v internetovom prehliadači. Samozrejme, tieto nástroje poskytujú funkcie nad rámec zobrazovania tabule úloh. Mnohé softvérové riešenia ponúkajú jednoduché vytváranie a správu používateľských príbehov, možnosť komentárov, kategorizovanie, pridávanie obrázkov, zobrazujú stav

aktuálneho burndown grafu a podobne.

Porovnanie

Obe z riešení majú svoje výhody a nevýhody. V tejto časti si ich priblížime a porovnáme. Papierový variant je z pohľadu motivácie lepšou voľbou, keďže sa tabula nachádza v miestnosti spolu s členmi tímu a púta pozornosť ak niekto pri tabuli presunie úlohu. Toto riešenie je taktiež lacnejšie ako softvér, nielen z pohľadu licencie ale aj realizácie. Jednoducho sa aktualizuje a nie je potrebné ho konfigurovať ani špeciálne nastavovať. Nevýhodou však ostáva skutočnosť, že papieriky z tabule môžu padať a priestor na ktorej sú rozmiestnené je obmedzený. Taktiež priestor na papieriku reprezentujúci úlohu je obmedzený a väčší popis v podobe poznámok alebo komentárov je nerealizovateľný.

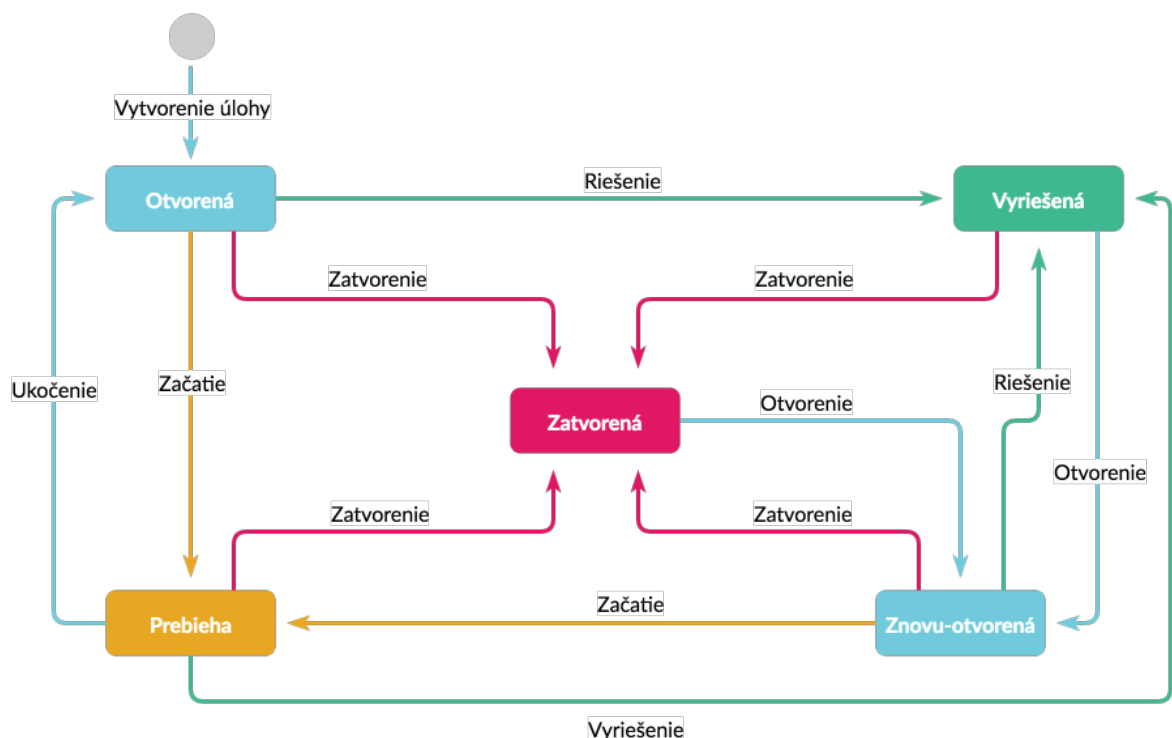


Obr. 2.6: Softvérová verzia tabule úloh v nástroji Trello

Softvér je ideálnym riešením pre distribuované tímy a členov, ktorí pracujú na rozdielnych pracoviskách. Každý z členov tak má prístup k aktuálnej tabuli úloh. Ak je nakonfigurovaný správne, je jednoduchý na použitie a má oveľa viac možností v popise úloh, ako stena s papierikmi. Prístup k histórii, archivovaným úlohám a detailnejšiemu popisu je taktiež výhodou. Nevýhodou ostáva cena softvéru, obmedzenie pri internetovom pripojení alebo registrácia všetkých členov k prístupu.

2.3.2 Atlassian Jira

Jira je softvér na sledovanie úloh vyvíjaný spoločnosťou Atlassian, ktorá stojí za mnohými ďalšími produktami na podporu agilného riadenia ako napríklad Confluence, Bitbucket alebo HipChat. 'Jira Agile', jeden z modulov Jiry, je špeciálne vyvinutý pre agilné riadenie projektov s integrovanými Kanban a Scrum tabulami úloh, pre ich jednoduché sledovanie a správu (*JIRA User's Guide, 2017*), [8].



Obr. 2.7: Štandardný pracovný postup úlohy v Jire (*JIRA User's Guide, 2017*) [8]

Hlavné pojmy vyskytujúce sa vo webovej aplikácii Jira sú projekt, úloha a pracovný postup. **Projekt** je súhrn úloh pre konkrétny produkt. Typickým príkladom projektu je softvérová aplikácia. **Úlohy**, ktoré patria k projektu možno rozdeliť a zgrupovať. Úloha sa môže skladať z podúloh a viacero úloh tvorí používateľský príbeh. Súčasťou

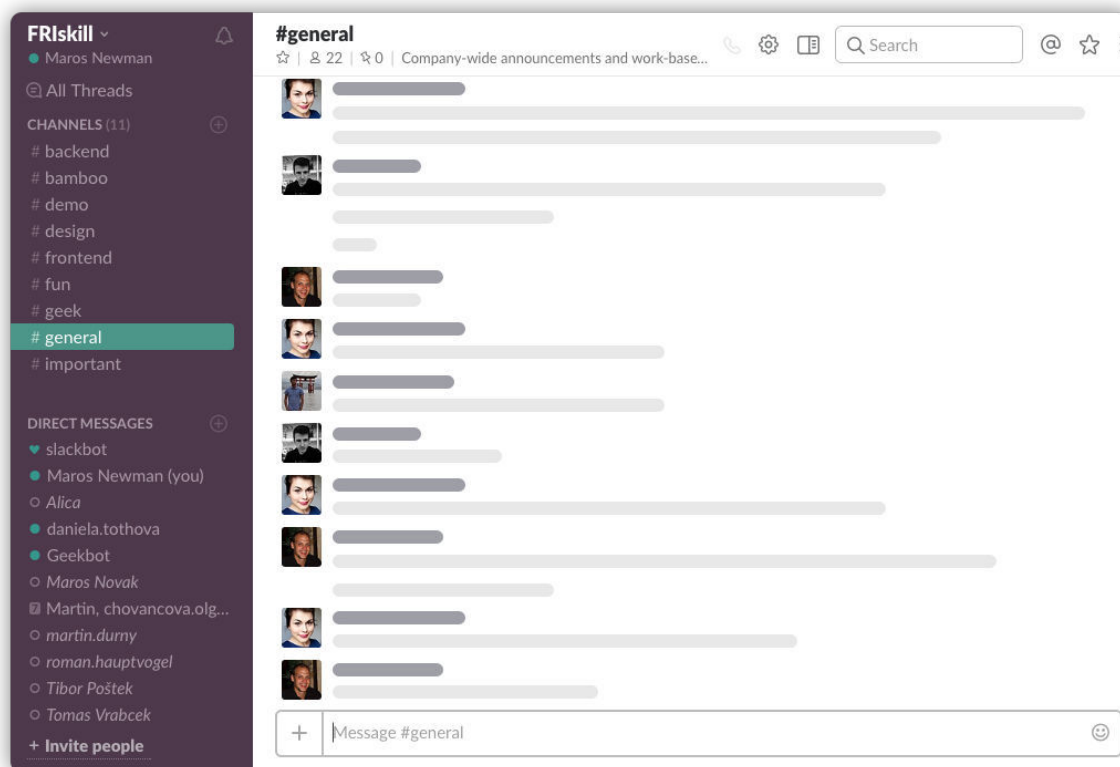
projektu je aj koncept verzionovania. Úlohy môžu byť prepojené k určitej verzii. To umožňuje lepšiu prehľadnosť a sledovanie, ktorá úloha bola vykonaná v ktorej verzii. Jira umožňuje spravovať viacero projektov a vďaka správe používateľov nastaviť ich prístup (*JIRA User's Guide, 2017*) [8].

Typická úloha je reprezentovaná časťou práce, ktorú vykonáva vývojár alebo iný člen tímu. Používatelia môžu pripojiť súbory, poznámky, komentáre, časový odhad, priradiť úlohu sebe alebo niekomu inému a pod. Existuje niekoľko typov úloh s rôznymi dátami a pracovnými postupmi. Pod pojmom **pracovný postup** rozumieme, že každá úloha má vždy určitý stav v ktorom sa nachádza. Tieto stavy môžu byť napríklad 'otvorená' alebo 'In Progress' a tento stav by mal vyjadrovať, čo sa s danou úlohou deje. Stavy majú zvyčajne pravidlá o tom, ako môžu používatelia prejsť z jedného stavu do druhého. Napríklad z 'uzavretého' môže používateľ zmeniť status iba na 'znovu-otvorený' tak, ako ukazuje diagram na obrázku 2.7. Tieto pracovné postupy možno prispôbiť individuálnym potrebám projektu. Jira poskytuje množstvo zásuvných modulov pre vylepšenie používateľského zážitku a taktiež možnosť pre vývojárov naprogramovať vlastný modul (*JIRA User's Guide, 2017*) [8].

2.3.3 Slack

Rovnako ako tabuľa úloh v akejkoľvek podobe je pri agilnom vývoji dôležitý aj komunikačný nástroj. Členovia distribuovaných tímov potrebujú často komunikovať a spolupracovať. V projekte Fri-Skill, ktorému sa venujeme v tejto práci, sa okrem Jiry osvedčil aj komunikačný nástroj Slack (obr. 2.8).

Slack je multiplatformová chat aplikácia pre tímy a spoločnosti vybavená mnohými funkciami. Správy je možné posilať vo verejných a súkromných kanáloch, alebo priamo konkrétnej osobe. Umožňuje nahrávať a zdieľať súbory, video-hovory medzi používateľmi, vytvárať vlastné emotikony či vyhľadávať v histórii správ. Slack umožňuje interagovať s inými aplikáciami a službami. Poskytuje verejné rozhranie pre programovanie aplikácií, vďaka ktorému môžu vývojári navrhovať vlastné integrácie ako napríklad chatbot.



Obr. 2.8: Multiplatformová chat aplikácia Slack

2.4 Identifikácia problémových oblastí agilného vývoja

Aj napriek adoptovaní služieb na zamedzenie časti problémov spájajúcich sa s agilným riadením v distribuovaných tímoch, dokážeme identifikovať viaceré kritické oblasti. V tejto časti si popíšeme jednotlivé problémy, s ktorými sme sa stretli v projekte Fri-Skill. Pri analýze sme využili výstupy **sprint retrospective** stretnutí, ktoré sa nachádzajú v prílohe A.

2.4.1 Komunikácia

Často krát sa v pripomienkach sprint retrospective vyskytovali problémy s komunikáciou. Členovia tímu poukazovali na skutočnosť, že pri riešení úloh nevedia, kto je za určité softvérové časti zodpovedný a s kým majú komunikovať. Obzvlášť kľúčovú úlohu zohráva komunikácia pri distribuovaných tímoch. Ako bolo spomenuté v úvode práce, študenti nepracovali na projekte v rovnaký čas ani pravidelne každý deň. Problém teda nastal, akonáhle viacerí vývojári pracovali spoločne na určitej funkcionalite.

2.4.2 Daily standup

Z analýzy Fri-Skill retrospektív vyplýva, že tím identifikoval denné standupy ako problémové. Členovia navrhli vlastný Slack kanál na dennú aktualizáciu stavu. Denné stretnutia sú prínosom pre tím, pokiaľ sa konajú každé ráno a všetci sú prítomní v jednej miestnosti. Avšak, ak sa členovia tímu nemôžu stretávať na dennej báze a nie všetci sú dostupní v rovnakom čase, môže to znamenať problém pri sledovaní progresu. Ďalším problémom, ktorý sme boli schopní identifikovať, bola absencia kontroly odpovedí jednotlivých vývojárov. Pri krátkom rannom stretnutí nie je čas na overenie progresu každého člena na tabuli úloh. Komplikované sa môžu zdať aj spätné kontroly odpovedí. Bez poznámok Scrum mastra tím nemá prehľad o tom, ktorý člen na čom pracoval.

2.4.3 Estimácia

Plánovanie šprintu má v sebe taktiež viacero kľúčových momentov, ktoré môžu skomplikovať agilné riadenie projektov. Pri odhadovaní náročnosti práce jednotlivých používateľských príbehov pomocou priradovania bodov sa využíva tzv. planning poker. Členovia tímu hodnotia príbehy na základe popisu vlastníka produktu, analyzujú možné komplikácie a porovnávaním s referenčným používateľským príbehom hodnotia náročnosť. Podobný problém ako pri denných stretnutiach môže nastať aj pri tejto aktivite. Distribuovaný tím musí využívať softvérové nástroje na hodnotenie úloh, video-konferencie, tabuľkové editory alebo poznámky. Pri hodnotení často krát dochádza k dvom javom: nadhodnotenie, alebo podhodnotenie používateľského príbehu. Dôsledkom môže byť aj zle zadefinovaný referenčný príbeh alebo nejasnosti a 'tmavé miesta' pri popise.

2.4.4 Monitorovanie

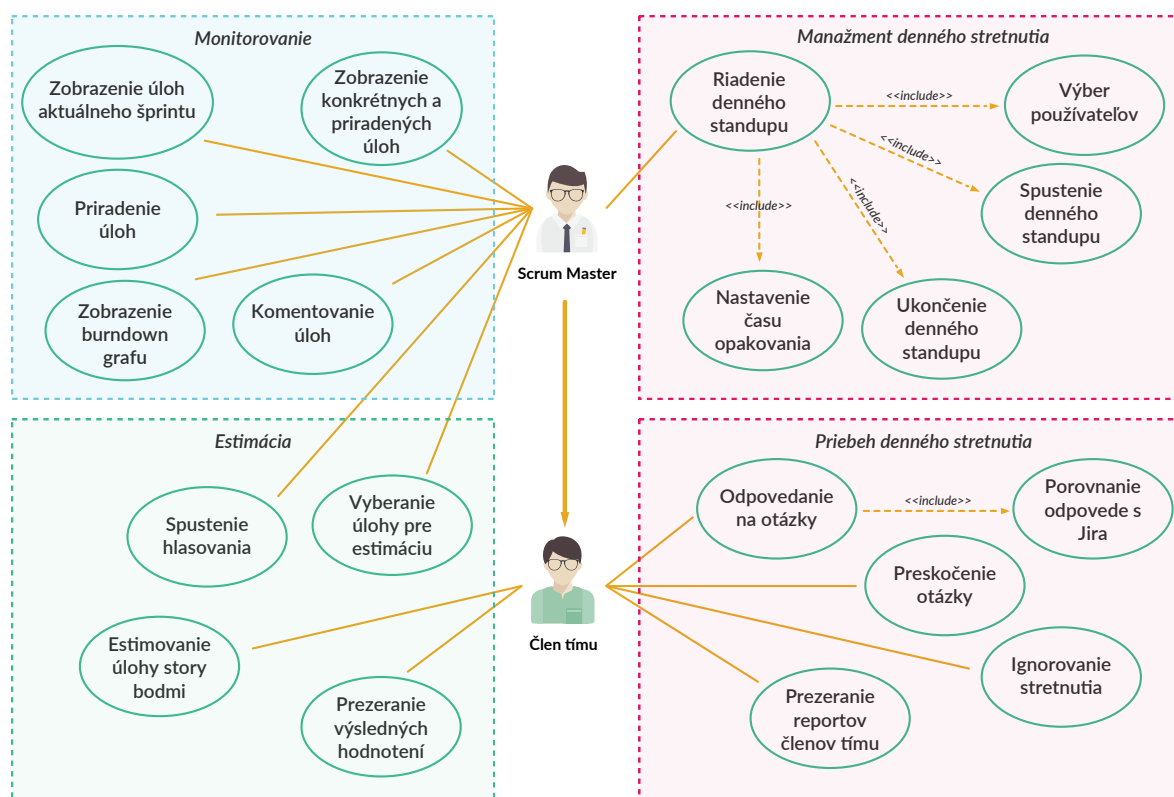
Počas aktívneho šprintu, keď vývojársky tím pracuje na dokončení používateľských príbehov, je jednou z úloh Scrum mastra aj kontrola progresu. Niekedy je ale ťažšie zistiť, na čom skutočne pracujú členovia tímu, obzvlášť, keď si zabudnú priradiť úlohy. Ďalšou kompetenciou Scrum mastra je aj kontrola burndown grafu, či je progres dostatočne rýchly a všetky úlohy budú dokončené načas. Identifikácia kritických momentov sa preto javí ako jeden z problémov.

3. Analýza riešenia

V tejto časti práce sa budeme venovať hodnoteniu existujúcich riešení a identifikácii komplexnej vízie, ktorá by bola schopná riešiť väčšiu časť komplikácií vyskytujúcich sa pri metodike Scrum, ktoré boli definované v kapitole 2.4. Súčasťou kapitoly je aj oboznámenie s teoretickými znalosťami použitých technológií a služieb, využívaných pri riešení problému.

3.1 Definovanie hodnotiacich kritérií riešení

Pre analýzu existujúcich riešení sme zvolili hodnotiace kritéria, ktoré by mali spĺňať. V kapitole 2.4 sa nám podarilo identifikovať problémové oblasti agilného vývoja a na základe tejto analýzy a podkladov zo stretnutí Fri-Skill sme definovali kľúčové funkcie pomocou diagramu **prípadov použitia** (obr. 3.1). Hľadané funkcie sme rozdelili na tri časti: monitorovanie, estimáciu a denné stretnutia (daily standup).



Obr. 3.1: Konceptuálny diagram prípadov použitia

Monitorovanie by malo Scrum mastrovi zjednodušiť prehľad nad úlohami členov tímu. Hľadaná platforma by mala ponúkať možnosť zobrazenia aktuálnych úloh šprintu, spolu s priradenými osobami, stavom v akom sa nachádzajú a podobne. Mala by poskytovať možnosti priradenia a komentovania používateľských príbehov. Spomínanou kritickou oblasťou bolo aj sledovanie burndown grafu, aby vedel Scrum master zareagovať na prípadné komplikácie a spomalenia vo vývoji.

Druhou oblasťou je **estimácia**. Proces plánovania je popísaný v kapitole 2.2.3. Softvér by v prípade distribuovaných tímov mal ponúkať riešenie, ako efektívne hodnotiť používateľské príbehy. Scrum master by v tomto prípade vybral používateľský príbeh, ktorý sa bude hodnotiť a členovia mu priradia body.

Kritérium	Atribúty / Funkcie	Spôsob hodnotenia
Monitorovanie	Zobrazenie úloh	max 4 body
	Priradenie úlohy	
	Komentovanie úlohy	
	Burndown graf	
Estimácia	Výber úlohy na etimáciu	max 4 body
	Informovanie členov o estimácii	
	Estimácia úlohy členmi tímu	
	Prezeranie výsledkov hlasovania	
Denné stretnutia	Výber členov tímu	max 6 bodov
	Spustenie standupu	
	Ukončenie standupu	
	Nastavenie času opakovania	
	Porovnanie odpovede s Jira	
	Prezeranie reportov členov tímu	
Integrácie	Podpora integrácie s nástrojom Slack	Áno/Nie
	Podpora integrácie s nástrojom Jira	Áno/Nie
	Iné nástroje pre integráciu	Počet
Použitelnosť	Prehľadné rozhranie	max 4 body
	Možnosti modifikácie	
	Moderný dizajn	
	Multiplatformová aplikácia	
Cena	Výdavky na plné fungovanie	€/mesiac/osoba

Tabuľka 3.1: Hodnotiace kritéria existujúcich riešení

Denné stretnutia sú v diagrame prípadov použitia (obr. 3.1) rozdelené na dve časti: manažment a samotný priebeh. Scrum master by mal byť schopný pomocou hľadanej platformy osloviť členov tímu, pokiaľ nie je možné uskutočniť stretnutie v

jednej miestnosti. Distribuovaný tím by mal obdržať otázky o ich výsledkoch a plánoch, ideálne automatizovaným spôsobom. Softvér by mal na základe vybraných používateľov a v určitý čas notifikovať tím. Dôležitou požiadavkou na systém je overenie odpovedí s nástrojom pre podporu agilného riadenia - **Jira**. Po komunikácii s lídrami projektu Fri-Skill by táto funkcionality mohla priniesť lepší prehľad o stave úloh a motiváciu pre členov, spravovať priradené úlohy.

Okrem vyššie spomínaných funkcionalít očakávame od existujúcich riešení, aby podporovali **integrácie**, ideálne so zaužívanými platformami, akými sú Jira a Slack. Ďalším kritériom je **použitelnosť**, pod ktorou môžeme rozumieť prehľadné rozhranie aplikácie, moderný dizajn, možnosti modifikovania danej služby podľa vlastných preferencií a multiplatformovú podporu rozšírenú aj o mobilné zariadenia. Posledným hodnotiacim kritériom je **cena**, ktorá by mala byť čo najnižšia, v ideálnom prípade hľadáme riešenie zadarmo. Všetky kritéria aj so spôsobom hodnotenia sú znázornené v tabuľke 3.1.

3.2 Existujúce riešenia

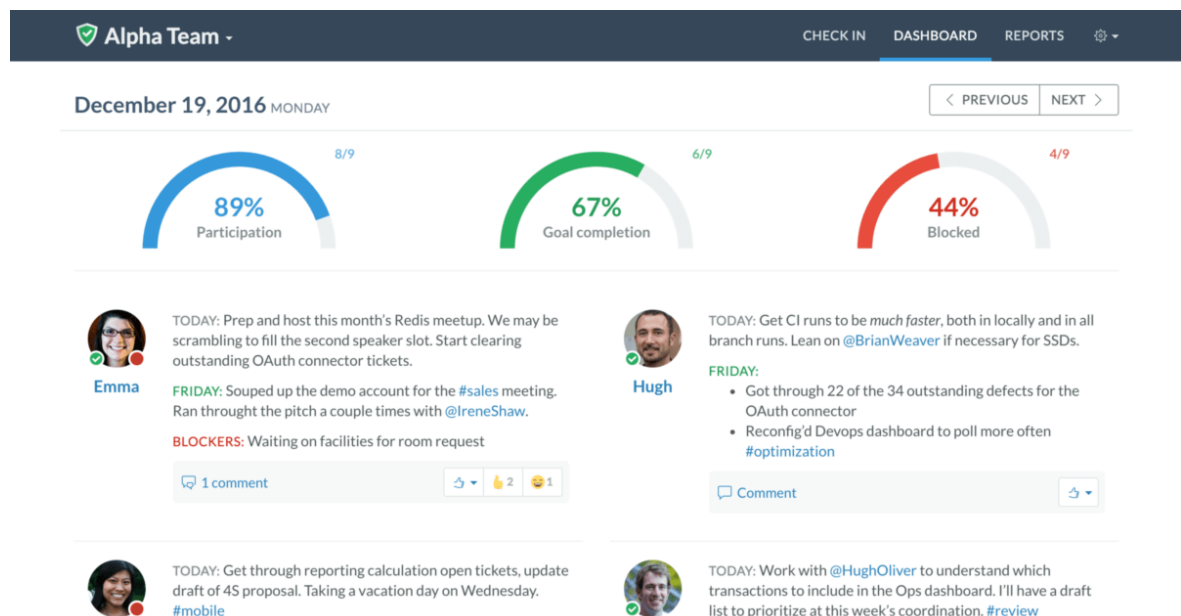
Nasledujúce riešenia sú rozdelené do dvoch podkapitol, na samostatne fungujúce softvérové služby a na doplnkové moduly pre komunikačný prostriedok Slack alebo nástroj Jira.

3.2.1 Samostatné softvérové riešenia

Samostatné softvérové riešenia predstavujú webové aplikácie, ktoré sú vyvinuté ako nezávislé platformy. Vývojový tím sa pre ich používanie musí zaregistrovať a používať novú službu. Väčšinou sa jedná o platený softvér.

Status Hero

Prvou analyzovanou aplikáciou je Status Hero (obr. 3.2). Softvér prevažne pre distribuované tímy v rámci metodiky Scrum poskytuje pekné webové grafické rozhranie a jednoduchú funkcionálnosť. Zameriava sa na **denné standup stretnutia**, pomáha zlepšiť transparentnosť v tíme a prehľadnosť úloh, zodpovedností a plánov. Aplikácia kontroluje denný progres, blokujúce úlohy, uchováva dáta pre neskoršiu kontrolu, kategorizuje odpovede členov tímu. Poskytuje súhrnné správy do emailu, alebo pomocou integrácie so Slackom do určeného kanálu.

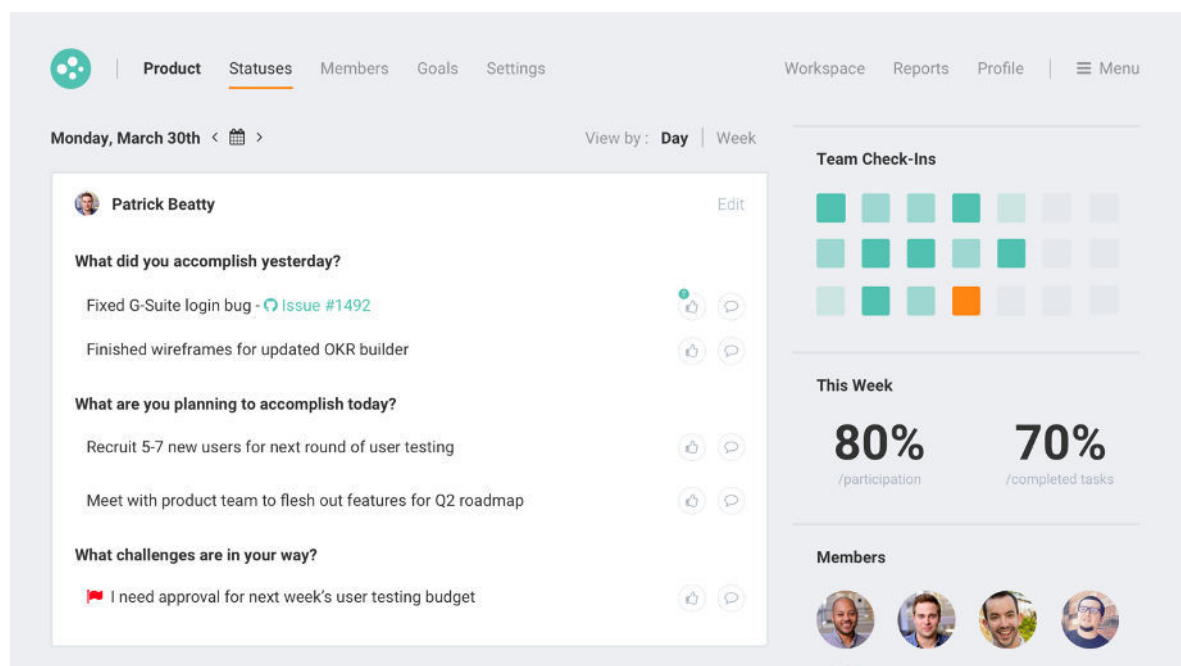


Obr. 3.2: Webová aplikácia Status Hero (*statushero*, 2017) [10]

Hodnotenie: Aplikácia ponúka riešenie pre riadenie agilných projektov v oblasti kontroly úloh a denných stretnutí. Riešenie však nie je prepojené so žiadnym softvérom na správu používateľských príbehov, alebo tabule úloh. Všetky dáta musia byť kontrolované ručne. Integrácia so Slackom neumožňuje pridávanie odpovedí na otázky denného standup-u, slúži iba na výpis reportu. Cena základnej služby bez možnosti integrácií alebo archivácie je 3\$/používateľ/mesiac. Cena za plnohodnotnú verziu končí na 7\$/používateľ/mesiac.

Jell

Ďalšou aplikáciou je Jell.com (obr. 3.3). Softvér ponúka možnosť denných online stretnutí, podobne ako Status Hero, pod názvom **SMART standup**. Odpovede na otázky slúžia aj ako vstupy pre denný 'To-Do List'. Aplikácia umožňuje definovanie cieľov, ako krátkodobých (denných), tak aj dlhodobých, týkajúcich sa produktu. Poskytuje vytváranie reportov, ktoré sú podobne ako pri Status Hero prepojené so Slackom. Služba tak prepája vývojársky tím v kontexte ich plánov a denných aktualizácií. Aplikácia disponuje kvalitným UX, ako aj dizajnom a je dostupná pre mobilné zariadenia.

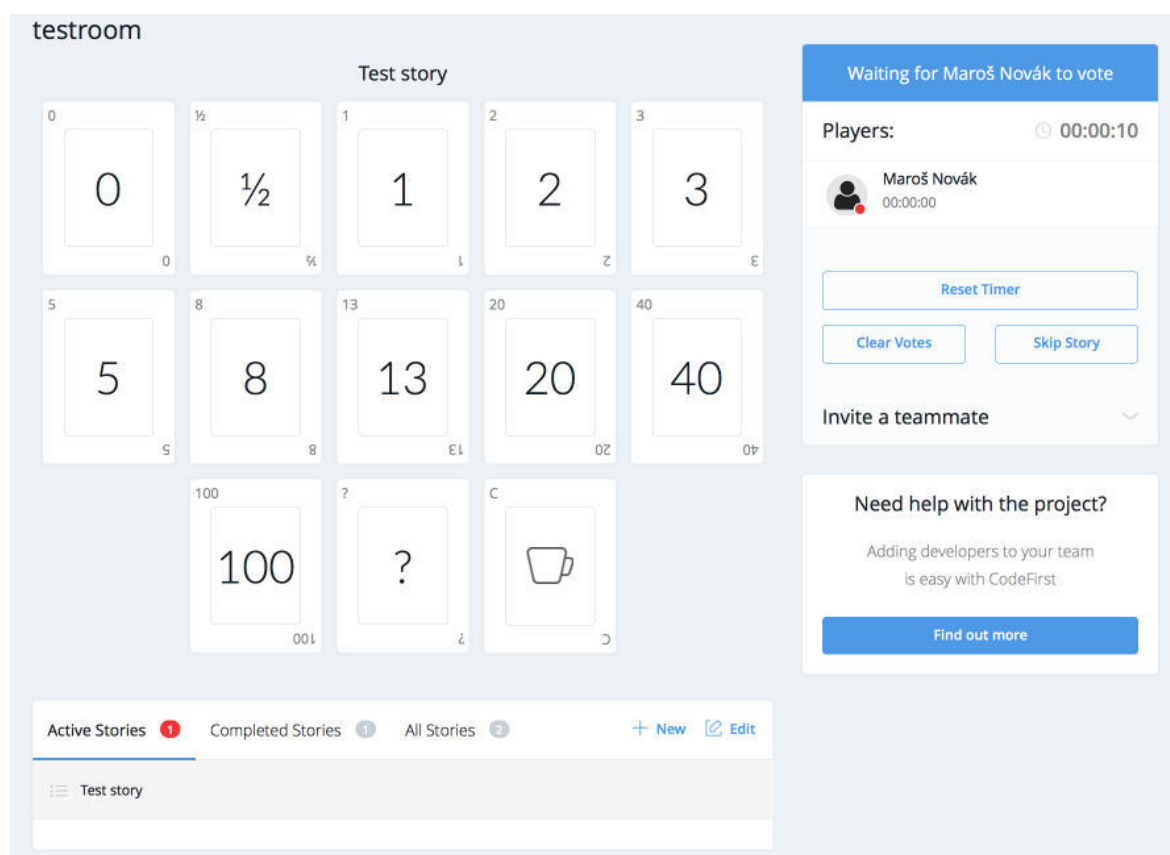


Obr. 3.3: Webová aplikácia Jell (jell, 2017) [11]

Hodnotenie: Jell je komplexnejšia služba oproti Status Hero, avšak ponúka riešenie len zlomku zadaných problémov. Neponúka integráciu s Jirou, alebo podobným softvérom pre podporu riadenia úloh a integrácia so Slackom umožňuje len zobrazovanie reportov. Softvér je dostupný za 10\$/používateľ/mesiac.

PlanITpoker

Táto webová aplikácia sa zameriava na inú špecifickú časť metodiky Scrum a tou je **estimácia**. Planning poker je realizovaný vo virtuálnej miestnosti, do ktorej sú pozvaní ostatní členovia tímu. V tejto miestnosti je možné pridať používateľské príbehy, ktoré sú následne predmetom hodnotenia. Keď Scrum master spustí hlasovanie, každému členovi sa zobrazia kartičky s hodnotami story bodov (obr. 3.4). Po odhlasovaní všetkých členov sa zobrazia výsledky a estimáciu je možné buď opakovať, alebo prejsť k ďalšiemu príbehu.

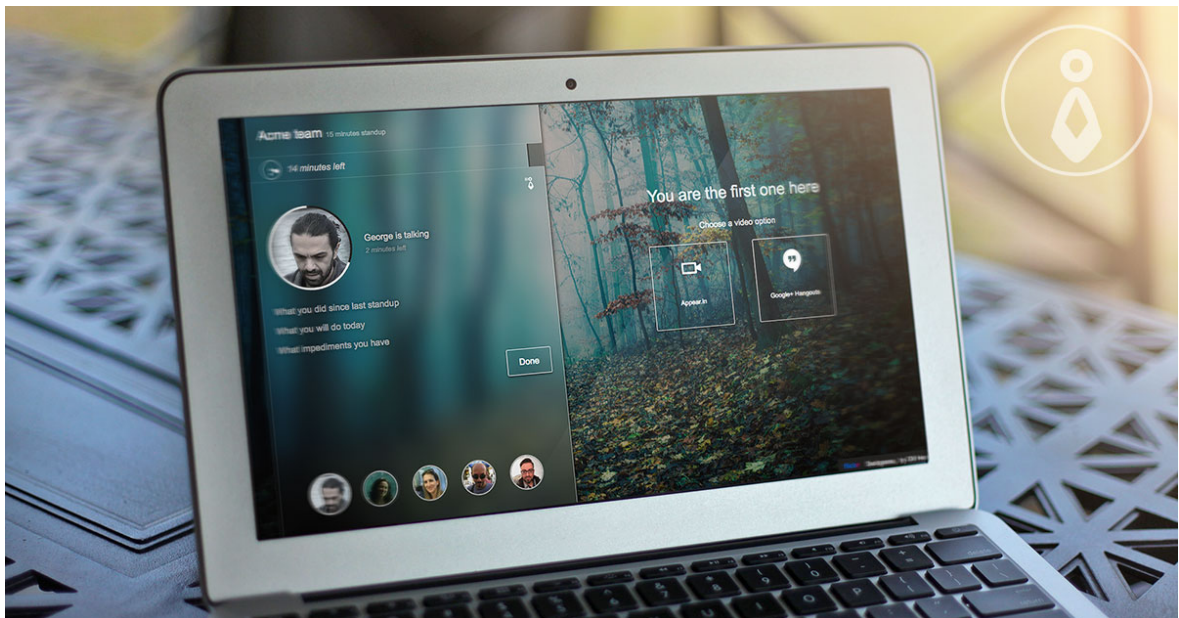


Obr. 3.4: Webová aplikácia PlanITpoker (planitpoker, 2017) [12]

Hodnotenie: Webová služba je účelná a prehľadná, absentuje však prepojenie s nástrojom na správu úloh. Výhodou je responzívny dizajn, ktorý umožňuje hlasovanie aj na mobilných zariadeniach. PlanITpoker je zadarmo a je využívaný v spoločnostiach ako IBM či Dell.

Team O'clock

Poslednou analyzovanou aplikáciou v kategórii samostatných softvérových riešení je Team O'clock (obr. 3.5). Zameriava sa na **denné stretnutia** v rámci distribuovaných tímov v peknom, webovom rozhraní. Súčasťou je okno na 'chatovanie', aj priestor pre poznámky. Ak je člen tímu na rade, je vyzvaný asistentom (softvérovým) a na odpovedanie na otázky má 3 minúty. Vďaka spolupráci s Google Hangouts a appear.in je možné využiť video-hovor. Táto platforma kladie dôraz na čas pri dennom standupe, keďže má byť stručný a krátky. Výhodou je integrácia so Slackom, pomocou ktorej je tím informovaný o začatí stretnutia. Team O'clock poskytuje priamo v Slacku riešenie oblasti estimácie. Pomocou integrácie je možné odhadovať používateľské príbehy priamo v komunikačnom nástroji.



Obr. 3.5: Webová aplikácia Team O'clock (*teamoclock*, 2017) [13]

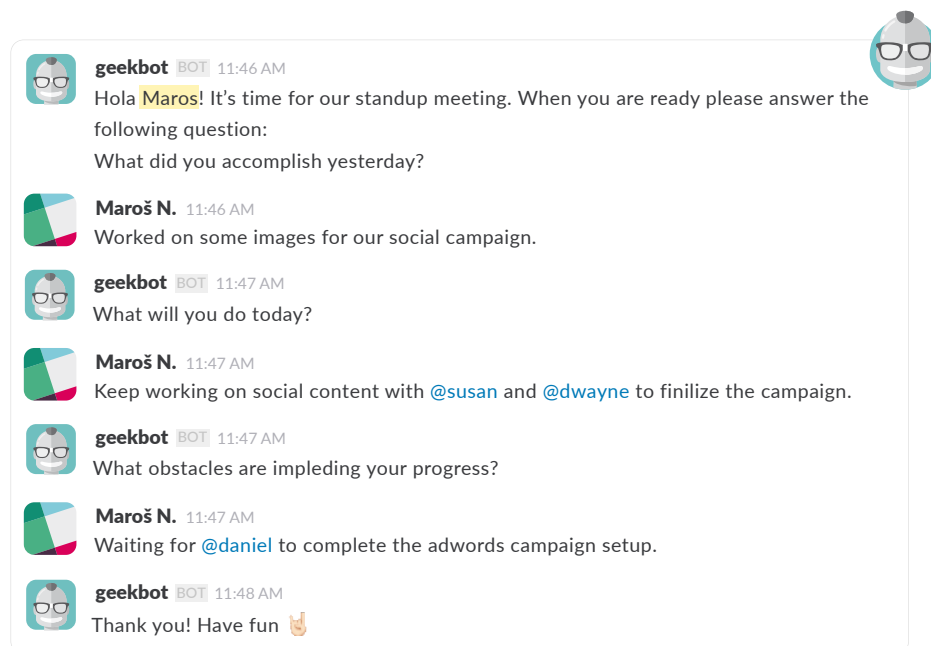
Hodnotenie: Z pohľadu komplexnosti je táto aplikácia najlepšou voľbou, keďže pokrýva čiastočne dve z troch problémových oblastí. Jej funkcie sú ale málo prepracované a chýba aj prepojenie s riadením úloh. Softvér je pekne dizajnovane spracovaný a vývojári ho ponúkajú zdarma.

3.2.2 Doplnkové moduly

Doplnkové moduly si môžeme predstaviť ako integrácie pre Slack, napríklad v podobe botov. Podobne ako samostatné softvérové riešenia eliminujú časť problémov agilného riadenia, avšak bez potreby registrovania, alebo adoptovania nového softvéru. Doplnkové moduly fungujú na platforme Slack, prevažne ako noví používatelia.

Geekbot

Geekbot je integrácia pre Slack určená primárne pre **denné stretnutia**. Bot každý deň v určitý čas rozpošle tímu otázky týkajúce sa ich práce na projekte (obr. 3.6). Odpovede následne obalí do prehľadného reportu a odošle do skupiny. Integrácia tak uľahčuje prácu Scrum mastrovi a zjednodušuje tento proces. GeekBot poskytuje aj webové rozhranie pod názvom **dashboard**, kde si admin môže nastaviť čas, kedy sa odošlú otázky, členov standup tímu a skupinu, kde sa zobrazia reporty. GeekBot reaguje na ukončenie alebo pozastavenie denných standup-ov priamo v chate pomocou príkazov.

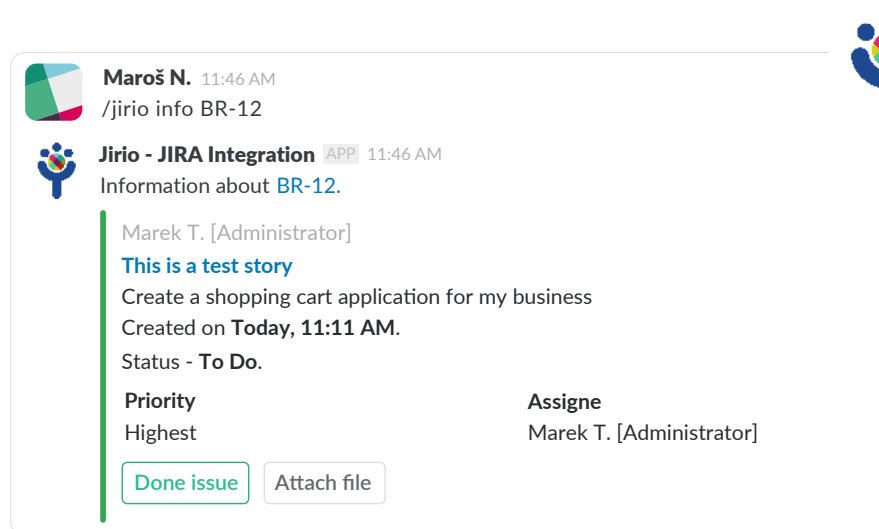


Obr. 3.6: Slack integrácia GeekBot

Hodnotenie: Oproti softvérovým riešeniam predstavuje GeekBot oklieštenú funkcionality, no jeho výhodou je jednoduchá integrácia, keďže nevyžaduje žiadnu registráciu ani prihlasovanie. Absentuje prepojenie so správou úloh, alebo logika nad odpoveďami členov tímu. Integrácia GeekBot je zdarma.

Jirio

Jirio predstavuje tzv. **slash command** integráciu pre Slack (obr. 3.7). Poskytuje riadenie, vytváranie a prezeranie Jira úloh priamo v konverzácii Slacku. Každá správa začína príkazom `/jirio`, za ktorým nasleduje akcia, napríklad `info číslo-úlohy`. Pomocou príkazu `/jirio auth` sa služba autorizuje voči Jire. Jirio ponúka aj prémium verziu svojich služieb, ktorá ponúka funkcie ako sú pridávanie komentárov, rýchle vyhľadávanie, hlasovanie, pridávanie súborov priamo do Jira úloh, zadávanie odpracovaného času a priradenie úlohy sebe. K dispozícii je aj zložitejšie vyhľadávanie pomocou JQL (JIRA Query Language). Pridanie a konfigurácia Jiria je jednoduchá vďaka používateľskému rozhraniu.

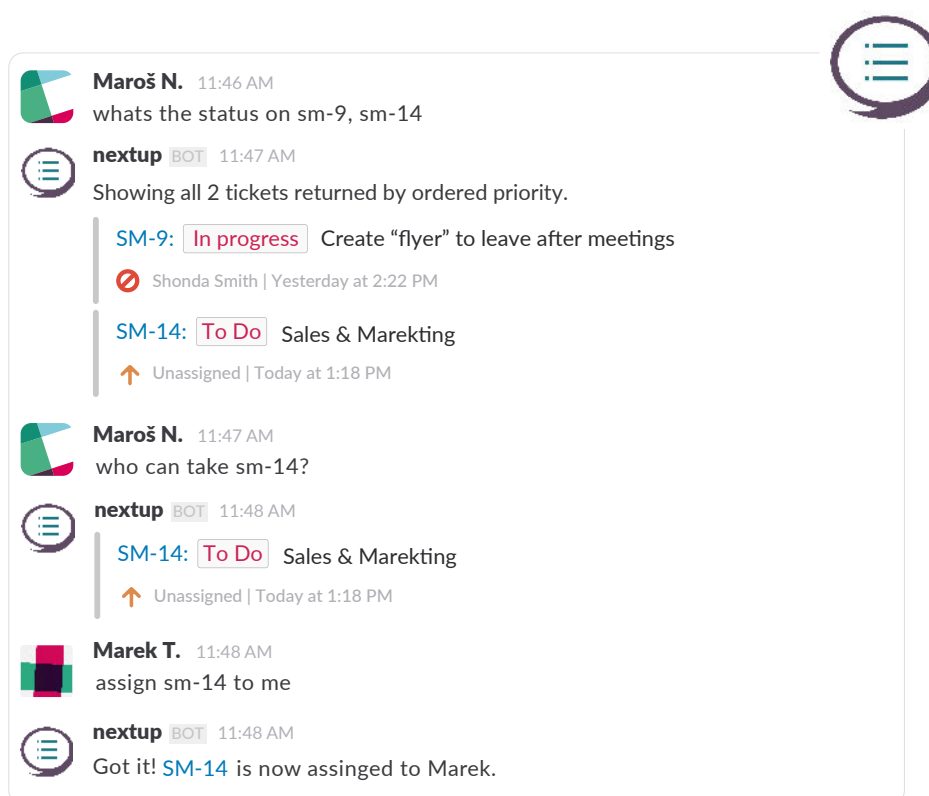


Obr. 3.7: Slack integrácia Jirio

Hodnotenie: Integrácia rieši odlišné aspekty ako doteraz spomínané služby. Zjednodušuje prístup a správu nad úlohami v rámci metodiky Scrum. Neposkytuje však komplexnejšie prepojenie medzi používateľmi v Slacku a Jire. Verzia zadarmo poskytuje iba niekoľko funkcionálít. Mesačné predplatné pre 20 členov je 4.97\$, nad 21 členov 19.97\$.

Nextup

Nextup je poslednou analyzovanou službou. Jedná sa o **Slack bota**, ktorý je prepojený s Jirou podobne ako v prípade integrácie Jirio, avšak poskytuje prepracovanejšiu komunikáciu a funkcionality (obr. 3.8). Bot sa správa ako skutočný používateľ, ktorý spracováva požiadavky. Tie nepredstavujú len príkazy ako pri slash command integrácii, ale môžu to byť aj celé vety. Bot ponúka prehľad, pridelovanie, editovanie úloh, tlačidlá na sledovanie, hlasovanie alebo zmenu statusu.



Obr. 3.8: Slack integrácia nextup

Hodnotenie: Bot pokrýva značnú časť manažmentu úloh a dokáže spravovať aj zložitejšie príkazy. Bot však nedokáže analyzovať zmeny a kritické okamihy pri vývoji, iba reaguje na podnety. Prepojenie tejto integrácie s Geekbotom a kontrolou burndown grafu by bolo skvelým riešením vyššie spomínaných problémov agilného riadenia. Bot je zadarmo pre menej ako 10 používateľov. Cena pre 10 a viac je 0.50\$/používateľ/mesiac.

3.2.3 Zhrnutie analýzy existujúcich riešení

Softvérové riešenia aj napriek ich príjemnému dizajnu a spoľahlivej funkcionalite pokrývajú iba čiastočné problémy. Absentuje hlavne prepojenie s iným softvérom pre správu používateľských príbehov a úloh. Reporty obsahujú len slovné popisy členov tímu a ich progresu. Denné online standupy pre distribuované tímy predstavujú prepracované riešenia a výhodou je aj archivácia odpovedí.

Doplňkové moduly taktiež spĺňajú riešenia čiastočných problémov. Chýbajú im ale komplexné funkcie, pokrývajúce viacero aspektov agilného vývoja. Najväčšou výhodou integrácií je, že sú postavené nad používaným softvérom. Oproti softvérovým riešeniam prepájajú služby na správu úloh, no chýba im interakcia medzi používateľskými vstupmi a pridelenými úlohami, taktiež monitorovanie a analyzovanie kritických udalostí.

	Status Hero	Jell	PlanTpoker	Team O'clock	Geekbot	Jirio	Nextup
Monitorovanie	1	1	0	0	0	3	3
Estimácia	0	0	4	4	0	0	0
Denné stretnutia	4	5	0	5	5	0	0
Podpora integrácie s nástrojom Slack	Áno	Áno	Nie	Áno	Áno	Áno	Áno
Podpora integrácie s nástrojom Jira	Nie	Nie	Nie	Nie	Nie	Áno	Áno
Iné nástroje pre integráciu	6	1	0	2	0	0	0
Použitelnosť	4	4	3	2	4	3	4
Cena	6.6	7.6	Zdarma	Zdarma	2.8	3.8/18	Zdarma

Tabuľka 3.2: Hodnotenie existujúcich riešení

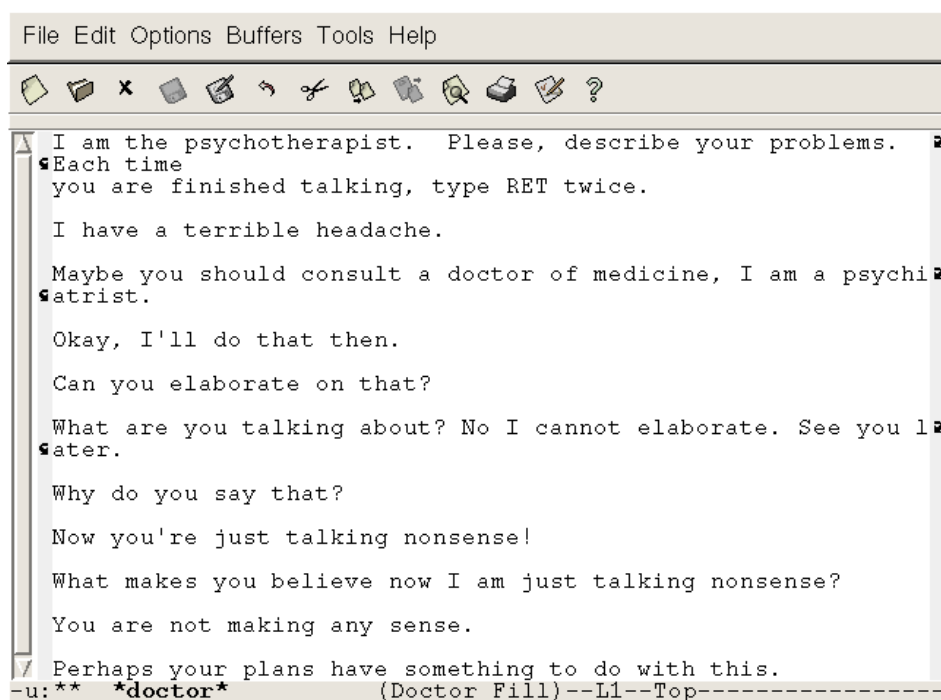
Z tabuľky 3.2 vyplýva, že sme **nedokázali nájsť existujúce riešenie** pokrývajúce všetky nami identifikované problémové oblasti. Definované prípady použitia sú riešené len čiastočne. Preto sme sa rozhodli navrhnúť vlastné riešenie s využitím komunikačnej platformy Slack a nástroja pre podporu riadenia agilných tímov. Keďže sa snažíme zjednodušiť procesy a čo najefektívnejšie odstrániť problémové oblasti, nie je v záujme tohto projektu vyvíjať nové webové rozhranie. Ideálnym riešením môže byť práve spojenie internetového bota prepájajúceho úlohy Jiry v nástroji Slack.

3.3 Technológie a služby

V tejto časti sa budeme venovať teoretickému pozadiu použitých technológií a služieb, ktoré je potrebné vedieť pri neskoršej implementácii. Analýza technológií neporovnáva viaceré riešenia ale popisuje a zdôvodňuje konkrétny výber. V rámci agilného vývoja sa kladie dôraz na väčšiu komunikáciu a keďže je v distribuovaných tímoch priama komunikácia veľmi obmedzená, zamerali sme sa na systémy konverzačných robotov, takzvaných chatbotov. Chatbot môže byť tou správnou softvérovou podporou pri riadení tímu, v rámci komunikačného kanála.

3.3.1 Chatbot

Chatbot alebo konverzačný robot, môže byť definovaný ako počítačový program, ktorý interaguje s používateľmi s využitím prirodzeného jazyka, niekedy v určitej doméne, alebo na určitú tému, prípadne môže obsahovať modul na rozpoznávanie reči (*Mauldin M., 1994*) [5]. Je známy aj pod názvom konverzačný agent alebo dialóg systém, ktorého úlohou je simulovať správanie a vyjadrovanie človeka najlepším možným spôsobom. Konverzačný robot však nie je pojem posledných rokov.



Obr. 3.9: Príklad komunikácie s prvým botom Eliza

História

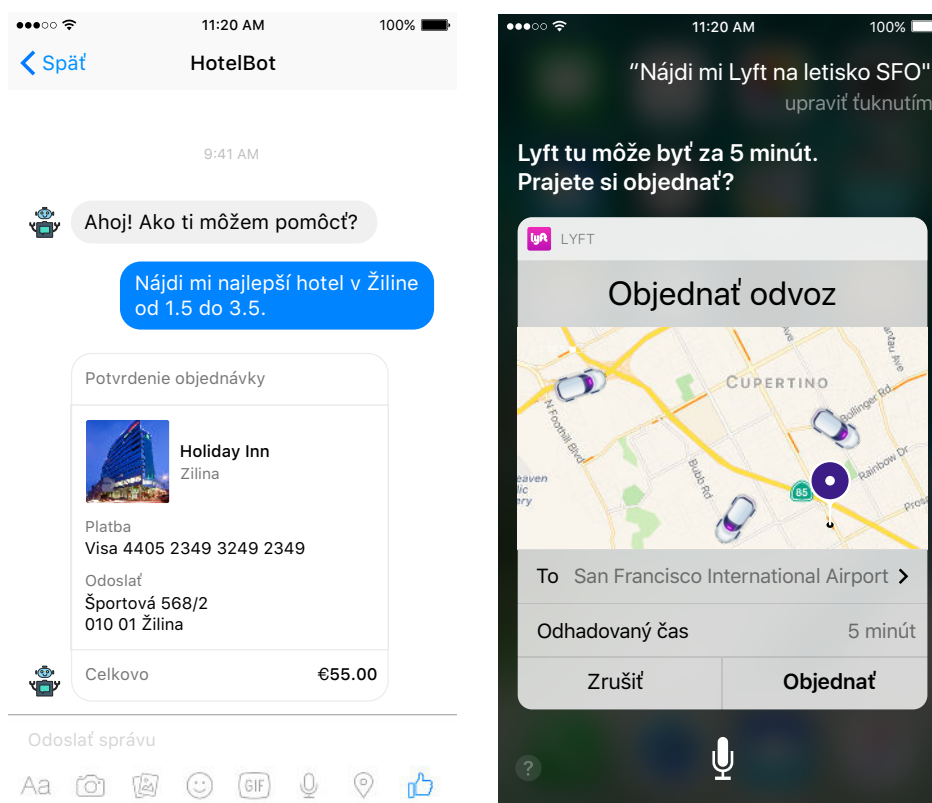
Úplne prvý chatbot, **Eliza** (obr 3.9), bol vytvorený v roku 1966. Jeho tvorca Joseph Weizenbaum z Massachusetts Institute of Technology (MIT) vyvinul tento systém na IBM 7094. Komunikácia s človekom bola zabezpečená cez klávesnicu a monitor, vstup do počítača bol napísaný v prirodzenom jazyku s bežnou interpunkciou a vetnými štruktúrami. Eliza bola pôvodne určená pre klientov jedného psychiatrického oddelenia. Weizenbaum bol šokovaný, keď videl ľudí presvedčených o tom, že Eliza je skutočný človek a nadobudli toľko dôvery, že s ňou boli schopní diskutovať o osobných problémoch. Bola prvým chatbotom, ktorý prešiel **Turingovým testom** (*Bhatnagar S., 2017*) [18].

Eliza bola základom pre neskorší vývoj komunikačného robota **A.L.I.C.E** (Artificial Linguistic Internet Computer Entity), ktorý môžeme nazvať aj určitým rozšírením bota ELIZA. Alice je ocenený open source chat robot, využívajúci umelú inteligencia, ktorý používa AIML (Artificial Intelligence Markup Language) pre vytvorenie odpovedí na otázky (*Wallace R. S., 2009*) [7].

Súčasný stav

Veľký míľnik v tejto oblasti zaznamenala Apple Siri (obr. 3.10) v roku 2011. Bot na báze rozpoznávania hlasu, ktorý využíva umelú inteligencia na vyhodnocovanie požiadaviek a poskytuje inteligentné odpovede. Možno povedať, že digitálni asistenti ako **Siri**, **Alexa** od Amazonu, **Cortana** od Microsoftu alebo **Google Now** sú chatboti, ktorí rozpoznávajú a analyzujú hlasové príkazy a prekladajú ich na text. Aj keď schopnosti hlasom riadených digitálnych asistentov sú pôsobivé, v posledných dvoch rokoch sa spoločnosti stále viac venujú skúmaniu možností chatbotov (obr. 3.10). Ak sa firma pravidelne zaoberá zákazníkmi, budú mať pravdepodobne konzistentné problémy alebo otázky, a budovanie komunikačného bota je jednoduchý spôsob, ako poskytnúť informácie efektívnejšie, než zákaznícky servis (*Martin E., 2016*) [17].

Preto sa v súčasnosti čoraz viac stretávame s chatbotmi aj vďaka rozšíreným sociálnym sieťam. Spektrum konverzačných robotov, z ktorých si dnes môžeme vybrať a jednoducho nainštalovať na platformách Facebook, Twitter či Skype je naozaj rozmanité. Niektorí námedia pomôcť s výberom filmu podľa našich preferencií, poradia pri výbere oblečenia alebo dokonca zabezpečia rezerváciu hotela.

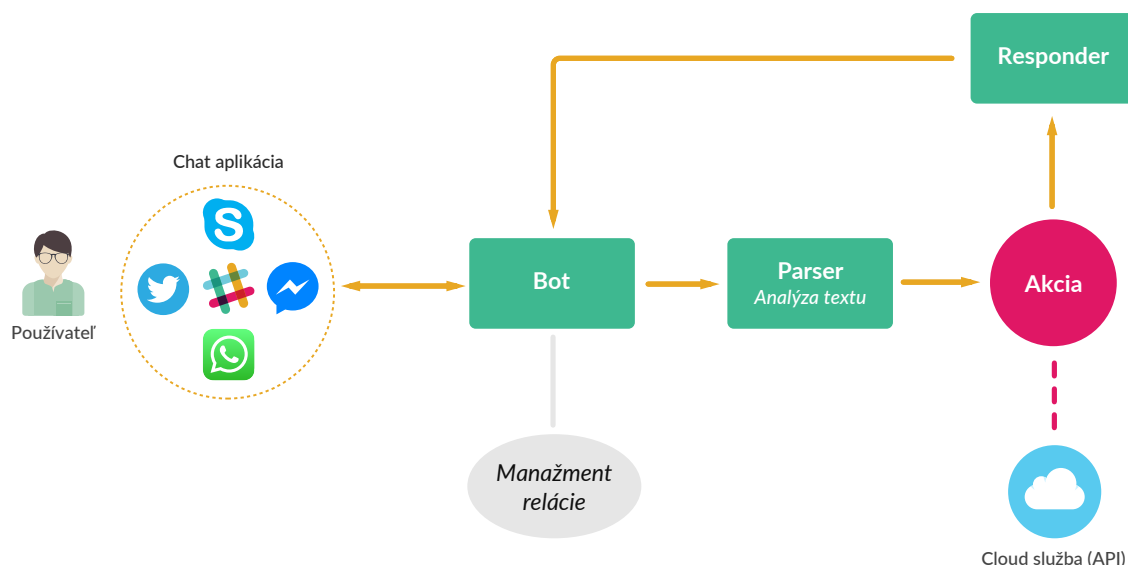


Obr. 3.10: Klasický textový chatbot a digitálny asistent Apple Siri

Všeobecná architektúra

Tradičné bot systémy fungujú tým spôsobom, že porovnávajú používateľský vstup s pred-pripravenými odpoveďami pomocou regulárnych výrazov. Pre vstupy sú uložené preddefinované odpovede. Úspech týchto systémov závisí na tom, či majú dostatočne veľký súbor dvojíc vzorka/reakcia, keďže používatelia môžu hovoriť o akejkoľvek téme (*T. Bickmore, 1999*) [6]. Dnešné systémy pracujú na komplexnejšej báze vďaka algoritmom spracovania prirodzeného jazyka a schopnosti samoučenia sa.

Najdôležitejšou komponentou architektúry každého bota je prirodzene **používateľ**. Predstavuje zdroj vstupov a príkazov, vďaka ktorým ovplyvňuje správanie aplikácie. Používateľ využíva kanál, pomocou ktorého môže komunikovať s botom. Sem patria bežné **chat aplikácie** ako napríklad Facebook Messenger, Skype, Twitter, What's up, alebo Slack. Správy, ktoré odošle používateľ sú ďalej prijaté komponentom **Bot**. Bot je prvok, ktorý očakáva správu a spúšťa konverzáciu. **Relácia** je spustená, pokiaľ používateľ nedokončí svoju sadu požiadaviek. Tieto správy sú následne odovzdané časti s názvom **Parser**. Jeho primárnou funkciou je analyzovať správu prijatú od používateľa v



Obr. 3.11: Všeobecná architektúra chatbota

bežnom jazyku a skonvertovať ju na štruktúru dát spracovateľnú programom. Výsledkom je správa rozdelená do entít, ktoré určujú akú akciu má bot vykonať. Napríklad ak sa chce používateľ dozvedieť, aké je počasie, Parser rozpozná kľúčové slová a **Akcia** vykoná požiadavku na zistenie aktuálneho počasia, volaním **cloud služby**. Akonáhle sú dáta z API (Application programming interface) obdržané, presunú sa poslednému logickému prvku pod názvom **Responder**. Jeho úlohou je zostavenie odpovede pre používateľa prostredníctvom prevedenia súboru informácii na prirodzený text. V príklade počasia to môže byť výstup: *"Dnes bude slnečno, maximálne 22 stupňov"*. Táto správa sa odošle späť chatbot komponentu, ktorý ju odovzdá používateľovi (Bhatnagar S., 2017) [18]. Popísaný postup predstavuje životný cyklus (obr. 3.11) jedného vstupu od používateľa.

Dostupné prostriedky pre vývojárov

V roku 2006 sa zúčastnil IBM Watson, počítačový systém vyvinutý za účelom odpovedania na otázky, populárneho kvízu pod názvom Jeopardy. Watson použil v hre algoritmy spracovania prirodzeného jazyka, ktoré sú už dnes dostupné pre vývojárov (Bhatnagar S., 2017) [18]. API, ukážky kódu, príklady riešení a návody pre začiatočníkov sú dostupné na webovej stránke github.com (IBM-Watson, 2017) [14].

V roku 2016 oznámil Facebook, že sprístupní svoju platformu Messenger pre chatbotov tretích strán. Facebook Messenger má v súčasnej dobe viac ako 900 miliónov používateľov mesačne a týmto krokom rozšíril bežné konverzácie o priestor pre biznis

a manažment (*Martin E. 2016*) [17]. Viac ako 65 tisíc vývojárov (*wit.ai, 2017*) [16] používa **wit.ai**, nástroj od Facebooku, ktorý poskytuje platformu na vytváranie inteligentných chatbotov pre Messenger. Facebook však nie je jediná spoločnosť, ktorá podporuje vývojárov komunikačných botov. Microsoft priniesol vlastný **Bot Framework**. Na rozdiel od wit.ai, Bot Framework nemá prepojenie s konkrétnou platformou a je ho možné nasadiť v rôznych prostrediach. Amazon ponúka svoje **Alexa Voice Service API**, podobne ako **Slack** poskytuje pre vývojárov niekoľko integrácií do ich chat aplikácie.

V súčasnosti sa čoraz viac startup-ov venuje vývoju stabilných knižníc a služieb, pre jednoduchý návrh a implementáciu komunikačných botov. Napríklad **api.ai** získala investíciu viac ako 8.6 milióna dolárov a ponúka riešenie pre analýzu prirodzeného jazyka. Medzi ďalšie patria **motion.ai**, **chatfuel** a za zmienku stojí aj **Botkit**. Open-source je dostupný na GitHube (*Botkit, 2017*) [15] a poskytuje jadro pre integrácie s aplikáciami Slack, Messenger a iných chat platforiem. Tieto služby sa zameriavajú hlavne na spracovanie jazyka. Keďže je analýza textu (Parser) jedna z najkompletnejších častí architektúry, budeme sa jej venovať detailnejšie v nasledujúcej kapitole.

3.3.2 Parser - Analýza textu

Ako už bolo spomenuté v predchádzajúcej podkapitole 3.3.1, Parser má za úlohu analyzovať textový vstup a preniesť ho na štruktúru dát (obr. 3.12). To je možné doceliť niekoľkými spôsobmi. Najjednoduchšia technika je **vyhľadávanie vzorov** (v originály Pattern Matching). Táto technika sa pokúša nájsť určitý vzor vo vetách na základe pravidiel, ktoré sú zadané v systéme. K tomu je možné využiť **regulárne výrazy** (RegEx). RegEx je dostupný v takmer všetkých programovacích jazykoch, avšak je to veľmi obmedzená metóda na analyzovanie textu. Jeho nevýhodou je, že textový vstup bude kontrolovať jedine v prípade, ak bude zodpovedať zadanému vzoru.



Obr. 3.12: Príklad spracovania textu z prirodzeného jazyka na štruktúru dát

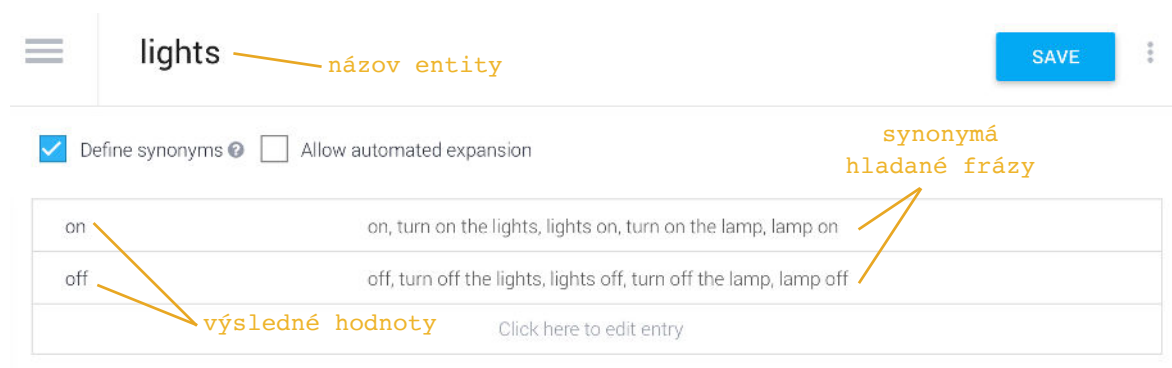
Omnoho sofistikovanejším prístupom je využitie metód umelej inteligencie, ako sú **strojové učenie** (machine learning - ML) a **spracovanie prirodzeného jazyka** (natural language processing - NLP). Strojové učenie je proces, ktorý umožňuje softvéru 'učiť sa', pomocou pozorovaní (vzorky dát), alebo na základe skúseností. Patrí do oblasti umelej inteligencie, ktorá predstavuje súbor štatistických metód pre riešenie problémov. NLP sa zaoberá problematikou softvéru 'porozumieť' štruktúre a významu prirodzeného textu. Prekladá text do strojového formátu, spracováva jeho význam (sumarizácia, syntaktický rozbor, atď.) a generuje zrozumiteľnú odpoveď späť používateľovi. Aby bol softvér schopný pochopiť význam prirodzeného jazyka, musí sa naučiť, ako to urobiť. Preto sa používa ML v rámci NLP (*Bhatnagar S., 2017*) [18].

API.AI

API.AI je platforma pre vytváranie konverzačného rozhrania pre botov, aplikácie a zariadenia. V architektúre chat bota môže pokrývať funkcie komponentov Parser a Responder. Platforma využíva spomínané algoritmy strojového učenia a spracovania prirodzeného jazyka. Úlohou tejto služby je prijatý vstupný text zanalyzovať, nájsť v ňom určitý zámer alebo parametre a previesť ho na štruktúru dát.

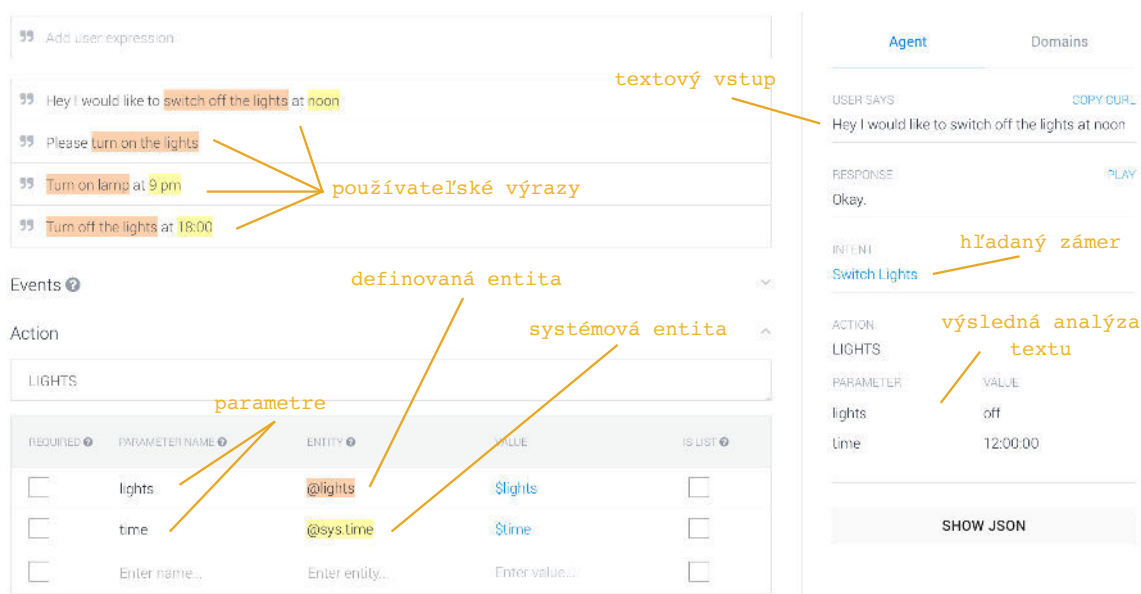
Platforma definuje tri kľúčové komponenty: agentov, zámery a entity. **Agent** predstavuje jednu inštanciu modulov pre rozpoznávanie jazyka. V agentovi je možné nastaviť jazyk, v akom prebieha komunikácia s chatbotom, jeho názov, popis a podobne. Po vytvorení agenta sa vygenerujú API kľúče pre komunikáciu a autentifikáciu.

Entity slúžia na identifikovanie a extrahovanie parametrov z textu. Existujú dva typy entít: systémové (definované API.AI) a vývojárske (definované v rámci agenta). Prvé spomínané je možné použiť na identifikáciu dátumu, polohy, hodnoty, kontaktov alebo jednotiek. Napríklad, ak využijeme systémovú entitu `@sys.address`, API.AI nájde v texte časť, kde je spomenutá adresa v rozpoznateľnom formáte a vyberie ju medzi parametre. Ak chceme zdefinovať vlastnú entitu, je potrebné vytvoriť jej názov a zadať hodnoty ktoré budú hľadané v kontexte. Napríklad, ak chceme zistiť, či sa v texte spomína niečo o zapnutí alebo vypnutí žiarovky, vytvoríme entitu `@lights` s hodnotami `on` a `off`. Týmto hodnotám ďalej priradíme synonymá, ako napríklad *turn on the light*, *lights on*, *lamp on* pre hodnotu `on` a podobne pre vypnutie (obr. 3.13). Agent následne rozpozná v texte výskyt entity a prenesie na štruktúra `parameter:hodnota`.



Obr. 3.13: Príklad entity pre hodnoty zapnutia/vypnutia svetla v platforme api.ai

Tretím komponentom je **zámer** (v originály intent). Ten sa skladá z viacerých entít a určuje, aký význam má daný textový vstup. Obsahuje vety alebo výrazy, ktoré určujú rovnaký zámer, ale sú formulované rôzne (obr. 3.14). Na základe týchto príkladov sa systém učí a zlepšuje. Ak agent obdrží správu, vyberie zámer, ktorý sa najviac zhoduje so vstupom a spracuje ho na dáta vo formáte JSON. Výhodou api.ai je, že umožňuje testovanie priamo na platforme a taktiež poskytuje knižnice pre komunikáciu so službami tretích strán.



Obr. 3.14: Príklad zámeru s vlastnou a systémovou entitou v platforme api.ai

Existuje viacero podobných služieb (pozri 3.3.1) poskytujúcich analýzu textu, alebo platformu pre vytváranie chatbotov, ale api.ai sa zdá byť najkvalitnejšia voľba. Vďaka akvizícii so spoločnosťou Google sa z nej stala stabilná platforma a ponúka svoje služby zadarmo. Má jednoduché a prehľadné rozhranie a mnoho možností a funkcionalít.

3.3.3 Slack API

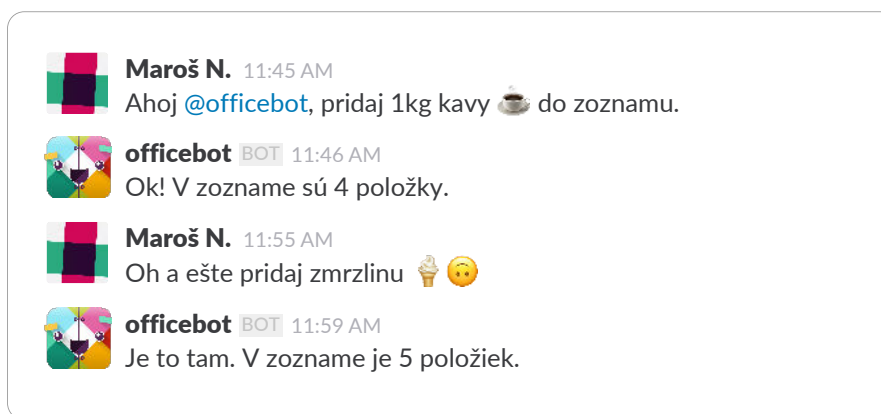
Ako už bolo spomenuté v podsekcii 2.3.3, Slack ponúka širokú škálu možností pre vývojárov vďaka ich prehľadne zdokumentovanému API. Vývoj nad platformou Slack umožňuje prispôbenie funkčnosti aplikácie pre vlastný tím, alebo vytvorenie bota zdieľateľného s celým svetom. Možnosti aplikácií: (*api.slack.com*, 2017) [9]

- odosielať bohatých a načasovaných upozornení vďaka **incoming webhooks**
- pomáhať používateľom pri vyvolaní pracovných postupov podľa ich potreby pomocou **slash commands**
- vytvárať **konverzačných botov** pre viac priateľské a prirodzené odpovede
- inštalovať integrácie vo svojom tíme bez potreby **OAuth**
- integrovať **tlačidlá v správach** pre lepšiu interakciu
- čítať a zapisovať dáta s **WEB, RTM a Events** rozhraniami
- starostlivo vyberať a obmedzovať prístup k vlastným aplikáciám iba **oprávneným osobám**
- distribuovať vlastnú aplikáciu v **App Directory** pre ostatné tímy

Pre každý Slack tím sú dostupné dve možnosti služieb: aplikácie a vlastné integrácie. **Aplikácie** sú komplexné riešenia, ktoré môžu obsahovať bota, incoming webhook, interaktívne správy, alebo slash command. **Vlastné integrácie** sú iba samostatne stojace programy, fungujúce v rámci jedného tímu (*api.slack.com*, 2017) [9]. Keďže v našej práci budeme využívať rozšírené funkcie, ktoré sú dostupné iba aplikáciám, zameriame sa na popis botov vytvorených v aplikácii.

Bot

Komponent Bot v architektúre na obrázku 3.11 reprezentuje v našom konkrétnom riešení práve Slack Bot. Slack boti (obr. 3.15) majú mnoho rovnakých možností ako bežní používatelia: majú profilové fotky, mená, popisy, ktoré existujú v tímovom adresári, môžu prijímať priame správy alebo byť spomenutí v konverzácii, môžu odosielať správy, nahrávať súbory a byť pridaní alebo odstránení z kanálov a skupín.



Obr. 3.15: Príklad komunikácie so Slack botom

Najväčší rozdiel medzi botmi a bežnými používateľmi je, že namiesto interakcie s tímom cez jednu z mobilných alebo desktopových aplikácií, Slack boti sú riadení programovo prostredníctvom **tokenu**, ktorý pristupuje k jednému alebo viacerým Slack rozhraniám. V rámci komunikačného kanálu, môžu boti robiť takmer čokoľvek, čo je možné naprogramovať aby robili (*api.slack.com*, 2017) [9].

Primárny spôsob, ako môžu boti komunikovať s ľuďmi v danom tíme je pripojenie k **API Real Time Messaging** (skrátene RTM API) a otvoriť tak WebSocket spojenie so Slackom. WebSocket bude odosielať všetky správy a aktivity, ktoré sa dejú vo verejných a súkromných kanáloch, v ktorých sa bot nachádza, rovnako ako v priamych správach. Alternatívny spôsob je využitie **Events API**, kde sa bot namiesto pripojenia cez WebSocket, prihlási k odberu konkrétnych udalostí alebo správ a Slack ich odosiela na server.

Vytváranie a programovanie bota

Vytvorenie bota je jednoduché (pozri Prílohu B). Stačí pridať novú aplikáciu pod vybraným tímom, v ktorom sa má bot zobrazovať a napísať jeho meno. Po pridaní sa vygeneruje prístupový token, ktorý bude používaný pri pripájaní k Slack API. Prístupový token predstavuje všetko potrebné pre nadviazanie komunikácie.

Možností vývoja z pohľadu programovacích jazykov existuje mnoho. Pre C#, PHP, Go, JavaScript, Ruby či Python existujú open-source knižnice pre komunikáciu so Slackom. Najväčšiu podporu spomedzi spomínaných programovacích jazykov má **Node.js a Python**. Pre dané jazyky existuje SDK (Software development kit) priamo od vývojárov

platformy Slack.

Najjednoduchší spôsob, ako vybudovať kvalitného bota v jazyku Node.js, je **Botkit**. Botkit je rámec, ktorý sa stará o väčšinu komunikácie so Slack API a tak sa vývojár môže sústrediť na správanie a logiku aplikácie (*api.slack.com*, 2017) [9].

Real Time Messaging

RTM umožňuje prijímať udalosti zo Slacku v reálnom čase a odosielať správy rovnakým spôsobom, ako používatelia. Je základom pre všetkých Slack klientov. RTM relácia sa spustí volaním metódy `rtm.start`. Tým sa inicializujú metadáta tímu a URL komunikácia medzi WebSocket serverom. Po úspešnom pripojení server poskytuje prúd správ, ako aj informácie o zmene stavu v tíme. Akákoľvek činnosť vykonaná v Slacku automaticky odošle udalosť pripojenému klientovi. Udalosť je reprezentovaná nasledovne ako JSON správa:

```
{
  "type": "message",
  "ts": "1358878749.000002",
  "user": "U023BECGF",
  "text": "Ahoj"
}
```

Každá udalosť obsahuje atribút **type**, ktorý popisuje o aký typ udalosti sa jedná. Ďalej sa v správe nachádza používateľ **user**, ktorý správu poslal, jej **text** a časovú pečiatku **ts**.

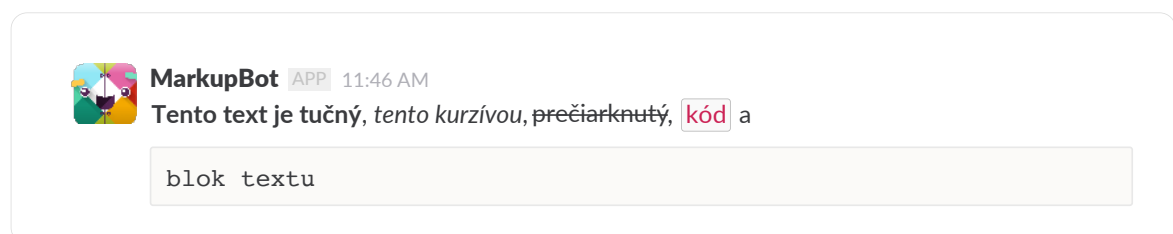
Pri odosielaní by každá správa mala mať jedinečné ID. Všetky odpovede na konkrétnu správu použijú toto číslo umožňujúce klientovi koreláciu odpovedí s odoslanými správami. Taktiež každá správa obsahuje **type** a **text**. Poslednou súčasťou je **channel**, čiže kanál do ktorého sa má správa odoslať a tie rozlišujeme na súkromné skupiny alebo priame správy používateľovi.

Druhou možnosťou odosielania správ je odpovedanie na správu, kedy nie je potrebné vedieť ID kanálu alebo používateľa. V tomto prípade využívame ID správy, ktorá je spomínaná vyššie. JSON vtedy obsahuje atribút **reply_to**. Keďže klient môže v jednom čase prijať väčšie množstvo správ od rôznych používateľov, vďaka ID udržiava konverzácie separátne.

Formátovanie správ

RTM API podporuje odosielanie správ naformátovaných podľa definovanej štruktúry. Práve formátovanie správ sa využíva v komponente Responder z obrázka 3.11. Správy môžu obsahovať ľubovoľnú postupnosť Unicode znakov (všetky správy musia používať kódovanie UTF-8). Môžu zahŕňať odkazy do iných skupín začínajúce znakom #, spomenutie používateľa pomocou znaku @, url odkazy, formátované dátumy, emotikony a pod. Základné formátovania textu je možné docieľiť nasledujúcimi značkami: “pre”, ‘code’, *_italic_*, ***bold*** a ~~~strike~~~. Využitie základného formátovania je prezentované v nasledujúcej ukážke. Pre danú JSON správu zodpovedá výstup na obrázku 3.16.

```
{
  "mrkdwn": true,
  "username": "MarkupBot",
  "text": "*Tento text je tučný*, _tento kurzívou_, ~prečiarknutý~,`kód` a ```blok textu.```"
}
```

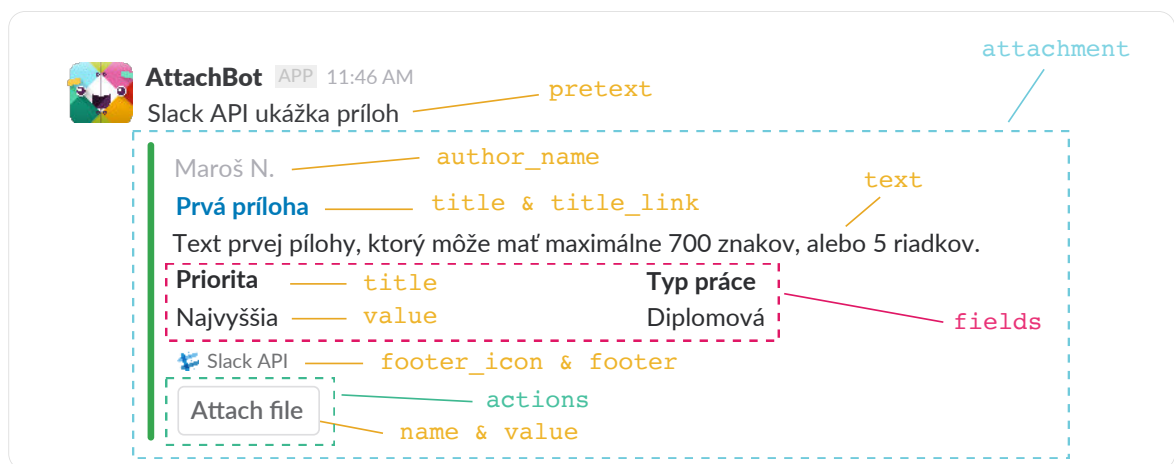


Obr. 3.16: Zobrazenie formátovania jednoduchých správ v aplikácii Slack

Štruktúra JSON udalosti definuje aj oveľa komplexnejšie správy využívajúce tzv. prílohy, ktoré umožňujú pridať ďalší kontext. Parametre prílohy umožňujú prispôbovať vzhľad správy, tak ako je to prezentované na obrázku 3.17. Niektoré z nich sú:

- `fallback` obsahuje zhrnutie, čo je obsahom správy
- `color` predstavuje farbu danej prílohy. Vizualne tak oddeľuje kontext správy
- `author_name` a `author_icon` znázorňujú koho sa príloha týka
- `title` a `text` reprezentujú názov a text jednotlivých príloh

- **fields** je definované ako pole a objekty v ňom budú zobrazené v tabuľke vo vnútri prílohy
 1. **title** je nadpis zobrazený nad hodnotou
 2. **value** je jednoduchá hodnota pod nadpisom
 3. **short** indikuje či budú objekty vedľa seba alebo pod sebou
- **image_url** importuje náhľad obrázku
- **footer** poskytuje možnosť zobrazit' text v pätičke správy aj s ikonou
- **actions** predstavuje pole interaktívnych tlačidiel



Obr. 3.17: Zobrazenie formátovania správy v aplikácii Slack s prílohou aj s prislúchajúcimi atribútmi

Slack Web API

Toto Slack rozhranie umožňuje vytváranie komplexnejších a zložitejších aplikácií. Web API je postavené na HTTP-RPC frameworku (remote procedure call - vzdialené volanie procedúr), ktorý predstavuje odľahčenú open-source knižnicu nad službou REST. Skladá sa z HTTP metód vo forme `https://slack.com/api/METÓDA`.

Všetky metódy musia byť volané pomocou zabezpečeného protokolu HTTPS. Argumenty môžu byť odovzdané ako GET alebo POST parametre, prípadne ich kombinácie. Odpoveď obsahuje objekt JSON, v ktorom sa bude vždy nachádzať logický boolean

atribút najvyššej úrovne **ok**, označujúci úspech alebo neúspech volania. Pri neúspešnom výsledku je súčasťou odpovede atribút **error**, kde sa nachádza stručný popis problému.

Na autorizáciu voči Slack API je postačujúci token vygenerovaný pri vytváraní bota, ktorý sa odosiela pri každej požiadavke.

3.3.4 Jira Cloud API

Na obrázku 3.11 môžeme vidieť Cloud službu (API) tretej strany, ktorú využíva chatbot. V našej práci tento komponent reprezentuje práve Jira API. Spoločnosť Atlassian poskytuje pre svoju službu Jira, podobne ako Slack, dobre zdokumentované rozhranie pre vývojárov. Pre prístup k rozhraniu existujú dve možnosti. Prvá možnosť je využiť rámec **Atlassian Connect**, ktorý sa používa pre integráciu s Atlassian Cloud aplikáciami a na vývoj zásuvných modulov. Alternatívny spôsob je pristupovať k Jira Cloud API priamo, pomocou **RESTful Web služieb**.

Jira REST API

Kedže sa v práci neplánujeme zaoberať vývojom zásuvných modulov, pre pripojenie k Jira Cloud API využijeme práve druhú spomínanú možnosť: REST služby. Autorizáciu v tomto prípade zabezpečuje jedna z dvoch metód: OAuth, alebo Basic HTTP.

OAuth je overovací protokol, ktorý umožňuje používateľovi (majiteľovi zdrojov) udeliť prístup k aplikácii tretej strany k svojim informáciám. JIRA používa **3-legged OAuth (3LO)**, čo znamená, že používateľ sa podieľa na procese overovania tým, že povoľuje prístup k svojim údajom.

Basic HTTP znamená, že Jira REST API je chránené rovnakými obmedzeniami, ako v prípade prihlásenia do štandardného webového rozhrania aplikácie. Službe tretej strany sú na základe mena a hesla sprístupnené údaje, ku ktorým má prístup používateľ po prihlásení do Jiry. Táto metóda autorizácie je odporúčaná pri vývoji skriptov alebo botov, pre jej jednoduchú implementáciu.

Štruktúra URI

Jira REST API poskytuje prístup k dátam prostredníctvom URI ciest. Aplikácia tretej strany musí vykonať HTTP požiadavku, po ktorej obdrží odpoveď. JIRA REST API používa JSON ako svoj komunikačný formát a štandardné metódy HTTP (**GET**,

PUT, POST a DELETE). Štruktúra URI k API zdroju vyzerá nasledovne: `http://host:port/context/rest/názov-api/verzia-api/názov-zdroja`. Napríklad pre prístup k úlohe s názvom JRA-9 je potrebné použiť štruktúru: `https://jira.atlassian.com/rest/api/latest/issue/JRA-9`.

3.3.5 Node.js

Manažment relácie (obr. 3.11), čiže prepojenie jednotlivých komponentov zabezpečuje programovací jazyk. V našom prípade sa jedná o platformu Node.js. Node.js je multiplatformové open source **JavaScript** prostredie, určené primárne pre vývoj serverových aplikácií. Je postavený nad Google V8 JavaScript jadre pre Google Chrome. JavaScript je odľahčený programovací jazyk, často používaný vo webových prehliadačoch. Vďaka udalosťami riadenej architektúre a neblokujúcemu I/O API, ktoré optimalizuje výkon požiadaviek, Node.js vyniká pokiaľ ide o komunikáciu v reálnom čase (*Mardan A., 2014*) [19].

Správca balíčkov **NPM** (Node package manager) umožňuje jednoduchú inštaláciu a publikovanie modulov tretích strán z veľkého a rastúceho úložiska. Každý modul má súbor s názvom manifest, ktorý popisuje jeho názov, verziu, detaily a závislosti. Výhody Node.js (*Mardan A., 2014*) [19], (*nodejs.org, 2017*) [20]:

- je odľahčené a rýchle
- jednoducho nastaviteľné a prispôsobiteľné
- dostupný manažér balíčkov obsahuje viac ako 140 tisíc knižníc tretích strán
- odstraňuje problémy medzi frontend a backend časťami, ktoré môžu byť zlúčené do jedného celku
- prístup k databázovým objektom vo formáte JSON
- je schopné zvládnuť veľký počet súbežných pripojení. Preto vyniká v implementáciách rýchlych a škálovateľných sieťových aplikáciach
- umožňuje znovu použiteľnosť kódu ako na klientskej časti, tak aj na strane servera
- BotKit framework pre vývoj chatbotov je naprogramovaný pre Node.js
- podpora komunikácie so Slack a Jira API

Total.js

Total.js je plne vybavený framework, ktorý účinne zlepšuje najdôležitejšie aspekty projektu počas jeho životnosti. (*total.js*, 2017) [21] Je **nezávislý** od iných knižníc a poskytuje dobre zdokumentovanú funkcionalitu. Je kompatibilný so všetkými verziami Node.js. Zjednodušuje vývoj a architektúru aplikácií, poskytuje šablónu pre štruktúru projektu. Motiváciou pre výber Total.js pre túto prácu bolo stretnutie s jeho tvorcom, Petrom Širkom. Na stretnutí objasnil technickú špecifikáciu technológie a výhody, ktoré ponúka pri vývoji:

- vstavaná NoSQL databáza
- RESTful smerovanie koncových bodov API
- žiadne závislosti na ďalších knižniciach
- kompletná asynchrónnosť
- izomorfné programovanie

3.3.6 Heroku

Heroku je navrhnuté tak, aby pomáhal vývojárom vytvárať webové aplikácie a publikovať online bez toho, aby sa museli starať o správu serverov, škálovanie operácií, alebo proces nasadenia. Po nasadení riadi všetky záležitosti spojené s web hostingom a bezproblémovým behom aplikácie. Predstavuje riešenie 'platforma ako služba' (Platform as a Service) pre malé a stredné podniky. Aplikácie je možné vyvíjať lokálne a pomocou Git príkazov jednoducho nasadiť. Medzi hlavné výhody patria:

- ideálna pre malé projekty ako napríklad chatbot
- možnosti nasadenia zdarma
- prehľadná dokumentácia a podpora pri celom procese nasadenia

4. Návrh a implementácia

V tejto kapitole sa budeme zaoberať návrhom systému z rôznych architektonických pohľadov, požiadavkám, spôsobu implementácie internetového bota od popisu funkcií až po nasadenie na server.

4.1 Definovanie cieľov a podmienok

Pri hľadaní vhodných riešení, ktoré by pokrývali problematiku agilného vývoja, sme definovali v kapitole 3.1 určité kritéria na hľadaný systém. Okrem požadovaných funkcií v podobe diagramu prípadov použitia bolo súčasťou kritérií aj prehľadný dizajn, jednoduchý používateľský prístup alebo čo najnižšia cena. Pre naše riešenie sme tieto požiadavky rozšírili o technické podmienky. Definované sú nasledovne:

- vytvoriť komunikačného bota nad používanými platformami v projekte Fri-Skill: Slack a Jira
- implementovať funkcie problematických oblastí: monitorovanie, estimácia a denné stretnutia
- pri vývoji využiť prostriedky, služby a knižnice dostupné zdarma
- využiť algoritmy spracovania prirodzeného jazyka a strojového učenia
- nasadiť server do cloudovej služby, aby bol chatbot dostupný nonstop
- dodržiavať dizajn manuál pri tvorbe správ chatbota v aplikácii Slack
- server navrhnuť modulárne a dynamicky, pre potreby ďalšieho rozšírenia funkcií

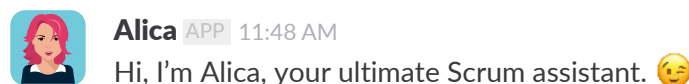
4.2 Návrh funkcií a príkazov

Príkaz môžeme chápať ako používateľský textový vstup pre bota, ktorý na základe analýzy vykoná určitú akciu. V zadaní práce predstavujú udalosti práve spomínané príkazy a akcie môžeme rozumieť aj ako funkcie bota. Tie vyplynuli z definovaných prípadov použitia v kapitole 3.1. Sú preto rozdelené do troch modulov: monitorovanie, estimácia a denné stretnutia. Predtým, ako popíšeme jednotlivé moduly, definujeme komunikačný jazyk a identitu bota.

4.2.1 Jazyk a identita bota

Navrhnutý bot komunikuje výhradne v **anglickom jazyku**. Prvým dôvodom je zložitosť a komplexnosť slovenskej gramatiky pri analyzovaní prirodzeného jazyka. Taktiež sme nenašli žiadnu službu podobnú `api.ai`, ktorá by dokázala rozpoznávať slovenský text. Algoritmy strojového učenia sú najpokročilejšie práve pre anglický jazyk. Druhým dôvodom je možné publikovanie bota, ktorý by bol dostupný pre všetky slack tímy. Takáto aplikácia by mala byť orientovaná na zahraničný trh a komunikovať v anglickom jazyku. Projekt Fri-Skill, v ktorom bude bot nasadený, obsahuje všetky dokumenty, popisy úloh, komentáre, prezentácie práve v angličtine. Nie je preto problém osvojiť si chatbota v cudzom jazyku.

Keďže boti v komunikačných nástrojoch majú mnoho vlastností a možností ako bežní používatelia, naše navrhované riešenie potrebuje určitú identitu. Ako meno bota sme zvolili **Alica**, inšpirované úplne prvým chatbotom Eliza. Profilový obrázok sme vybrali na webovej stránke `iconfinder.com` [22]. Obrázok je dostupný pod základnou licenciou [23], ktorá umožňuje jeho používanie na súkromné účely, v mobilných a webových aplikáciách, dokumentoch a prezentáciách. Alica s používateľmi komunikuje v bežnom neformálnom jazyku, napríklad využívaním emotikonov (obr. 4.1).

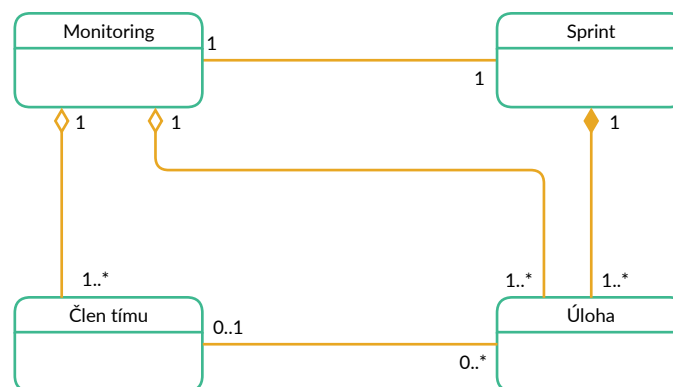


Obr. 4.1: Identita komunikačného bota Alica v aplikácii Slack a príklad jej správy

V nasledujúcich kapitolách sa budú v príkladoch komunikácie objavovať okrem Alice aj **Scrum master** a **Maroš N.**, ako člen tímu.

4.2.2 Monitorovanie

Prvou oblasťou, ktorej sa budeme venovať je monitorovanie. Funkcie spojené s monitorovaním majú sprehľadniť priradené úlohy členov tímu, uľahčiť ich priradenie, zmeniť status alebo pridať komentár priamo v komunikačnom kanáli Slack. V rámci šprintu máme vybranú množinu úloh. Každá úloha môže (ale nemusí) mať priradeného práve jedného člena tímu. Úlohy a im priradení členovia sú kľúčovými entitami funkcií monitorovania (obr. 4.2).



Obr. 4.2: Doménový model monitorovania úloh a členov tímu

Na scenári (obr. 4.3) si ukážeme jednotlivé príkazy a funkcie bota Alica. Pri každej správe je číslo, na ktoré sa budeme odkazovať. Konverzácia prebieha v spoločnom kanáli, kde sa nachádza Scrum master aj členovia tímu. Scrum master otvorí debatu a opýta sa Alice na aktuálny status úloh (1). Je možné získať aj vlastné úlohy použitím `my issues` v kontexte správy, alebo úlohy priradené konkrétnemu členovi: `@používateľ issues`. Alica musí byť označená v správe, aby bola schopná text spracovať, pokiaľ sa nachádza v kanáli. Zo správy zistí jej zámer a zobrazí úlohy, ktoré majú stav 'to do' alebo 'in progress' (2). Úlohami vo všeobecnosti označujeme používateľské príbehy, chyby, tasky a podobne. Vďaka bohatým správam (pozri obr. 3.17) dokáže bot zobraziť zoznam, kde má každá úloha farbu podľa priority, informáciu o type úlohy, aj ktorý člen ju má priradenú. V scenári Scrum master zisťuje, že jedna z úloh nie je priradená a opýta sa, kto má možnosť si ju priradiť (3). Jeden z členov sa chce dozvedieť o úlohe viac a tak požiada Alicu o detail úlohy (4). Každá úloha obsahuje kľúč, ktorý sa skladá z prefixu a čísla. Bot dokáže rozoznať kľúč a na základe synonymu slova `detail` aj zámer správy. Alica následne zobrazí detailnejšie informácie o úlohe (5). Vďaka interaktívnym tlačidlám (pozri kapitolu 4.5.6) je možné priamo v správe zmeniť status úlohy.

(1) **Scrum Master** 11:47 AM
Good morning **#team**. **@alica** please show me summary status.

(2) **Alica** APP 11:48 AM
📦 Here are 3 issues returned from Jira.

- To Do** **FR-684** Design guidelines
💡 Story | Priority: Medium | Assignee: Not assigned
- In Progress** **FR-480** Diploma thesis
💡 Story | Priority: Highest | Assignee: Maros Novak
- To Do** **FR-555** Fix login response
🐛 Bug | Priority: Low | Assignee: Peter Reguli

(3) **Scrum Master** 11:47 AM
Who can take issue FR-684?

(4) **Maroš N.** 11:48 AM
@alica what is the detail of fr-684?

(5) **Alica** APP 11:48 AM
FR-684 Design guidelines

💡 Story
As developer I want to have design guidelines for smoother development.

Status	Priority
To Do	Medium

Assignee	Last updated
Not assigned	April 8th at 21:11

Do you want to change the status?

(6) **Maroš N.** 11:48 AM
@alica assign to me issue fr-684 please.

(7) **Alica** APP 11:48 AM
👏 Done. Issue **FR-684** assigned to **@maros.novak**.

(8) **Scrum Master** 11:47 AM
@alica add comment to fr-684 'Needs to be done until Friday'.

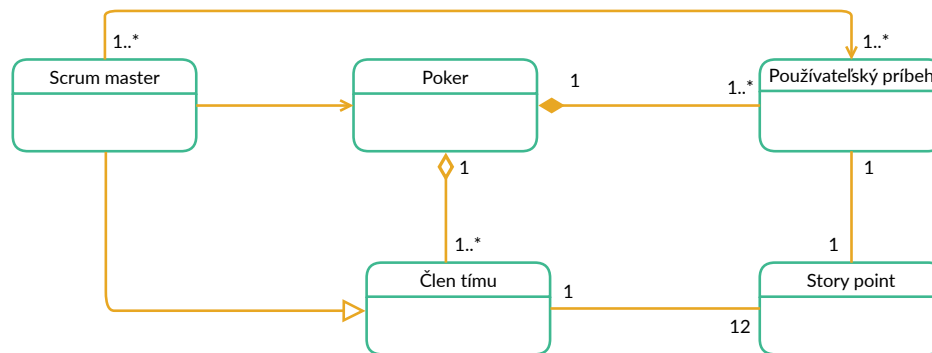
(9) **Alica** APP 11:48 AM
💬 Comment added to **FR-687**:

Needs to be done until Friday

Obr. 4.3: Monitorovanie - konverzácia v kanáli

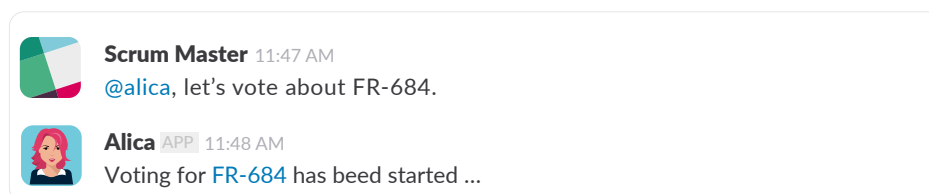
V ďalšej správe člen tímu vyzve Alicu aby mu priradila spomínanú úlohu (6). Podľa kľúčového slovného spojenia **assign to me** dokáže Alica rozpoznať, čo má s úlohou vykonať. Taktiež je možné priradiť úlohu niekomu inému, pomocou **assign to @používateľ**. Alica po úspešnom priradení správy v Jire informuje členov tímu (7). Scrum master následne priradí komentár (**add comment**) cez Slack priamo do Jiry (8). Alica extrahuje zo správy text v úvodzovkách a zobrazí úspešný výsledok (9). Spôsob extrahovania informácií z príkazov bude popísané v kapitole 4.5.3.

4.2.3 Estimácia



Obr. 4.4: Doménový model estimácie používateľských príbehov

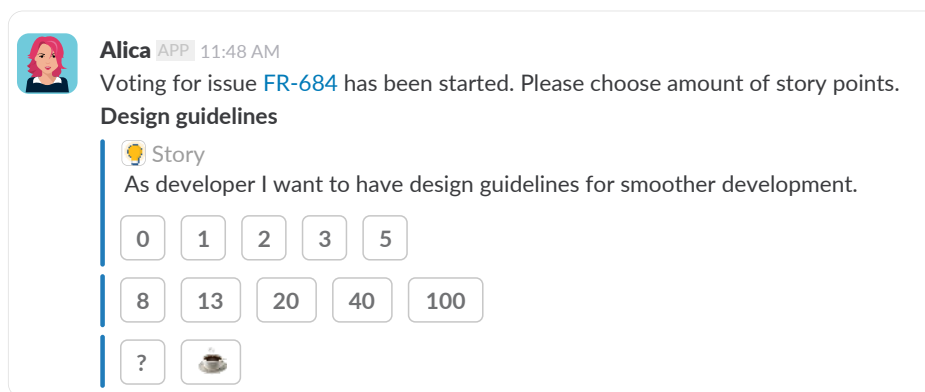
Estimácia je ďalším modulom chatbota Alica. Táto časť bola popísaná v kapitole 2.2.3 a identifikovaná ako jedna z problémových oblastí. Scrum master spúšťa planning poker prvým používateľským príbehom. V rámci jedného plánovania je možné hodnotiť postupne viacero príbehov. Každý člen tímu je vyzvaný k hodnoteniu tzv. story pointami, ktorých je 10 typov s rôznymi hodnotami. Na konci stretnutia by mal mať každý používateľský príbeh priradený jeden story point, ako to vyplýva z doménového modelu na obrázku 4.4.



Obr. 4.5: Spustenie hlasovania o používateľskom príbehu - konverzácia v kanáli

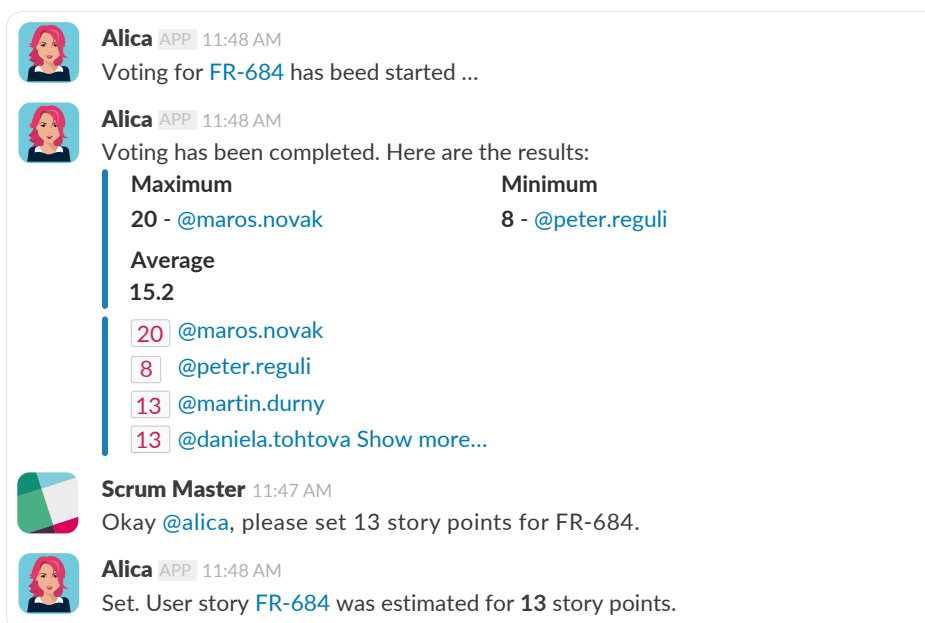
Prvým príkazom, ktorý je možné použiť v rámci slack kanálu, je začatie hlasovania (obr. 4.5) o niektorom z používateľských príbehov, ktoré existujú v nástroji Jira. Je potrebné, aby bol tento príkaz spustený v určitom kanáli, pretože Alica vyberie všetkých členov tímu, ktorí sú priradení v kanáli a informuje ich o hlasovaní. Preto je vhodné vytvoriť v Slacku jeden kanál, napríklad `#planning` alebo `#poker`, v ktorom budú všetci členovia pripravení na hlasovanie. Spustenie príkazu je umožnené iba Scrum mastrovi a musí obsahovať meno bota, používateľský príbeh a kľúčové slovné spojenie ako napríklad `let's vote`, `start voting`, alebo iné synonymá.

Po tom, ako Scrum master spustí hlasovanie, obdrží každý člen tímu, ktorý sa



Obr. 4.6: Hlasovanie - súkromná konverzácia

nachádza v danom kanáli, správu od Alice (obr. 4.6). Tá obsahuje popis úlohy a hodnoty story pointov, na ktoré je možné kliknúť. Takéto interaktívne správy s tlačidlami sú popísané v kapitole 4.5.6. Po odhlasovaní každého vyzvaného člena tímu Alica spracuje výsledky a uverejní ich v rovnakom kanáli (obr. 4.7).

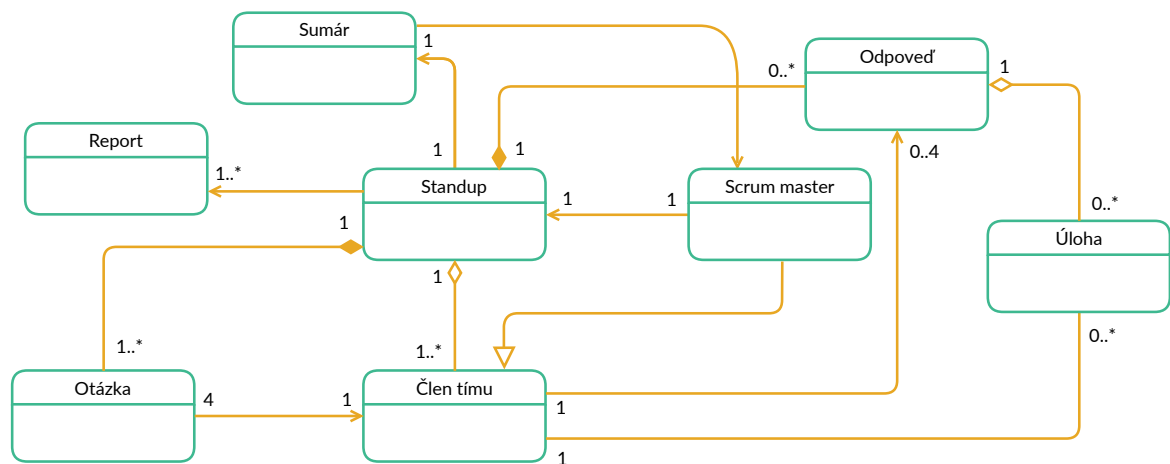


Obr. 4.7: Výsledky hlasovania a nastavenie bodov - konverzácia v kanáli

Táto správa obsahuje informácie o hlasovaní, najvyššej a najnižšej hodnote, ako aj priemer všetkých bodov. Na základe výsledkov môžu členovia diskutovať, alebo schváliť zvolené story pointy a pridať ich používateľskému príbehu. Scrum master môže používateľskému príbehu následne nastaviť výsledné body pomocou príkazu `set story points`. Táto funkcia je nad rámec definovaných prípadov použitia, ale je vhodným vylepšením procesu estimácie.

4.2.4 Denné stretnutia - standup

Posledným modulom riešenia je standup. Ten môžeme považovať aj ako hlavnú entitu doménového modelu (obr. 4.8). Princíp tohto stretnutia bol popísaný v kapitole 2.2.3. V našom riešení sa skladá z členov tímu, štyroch otázok, na ktoré odpovedajú a množinou ich odpovedí. Scrum master má ako jediný možnosť spustiť standup, alebo nastaviť jeho parametre. Každý člen tímu môže mať priradené úlohy, ktoré spomenie vo svojej odpovedi. Výstupom denného standupu je report, od každého člena tímu zvlášť a sumár, ktorý obdrží iba Scrum master. Jednotlivým častiam sa budeme venovať nižšie.

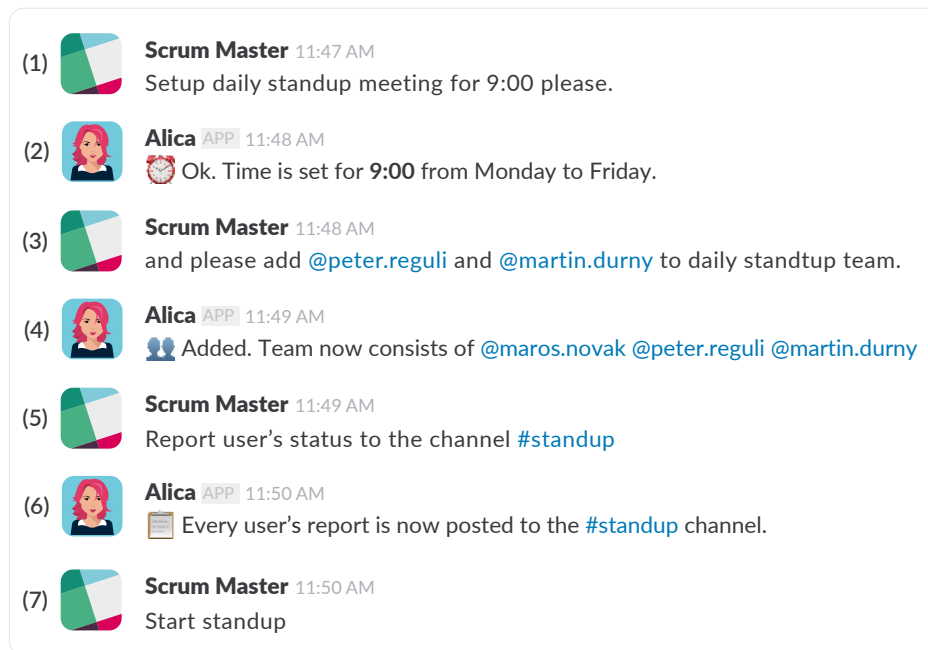


Obr. 4.8: Doménový model denného standupu

Riadenie standupu

Riadenie standupu umožňuje Scrum mastrovi nastavovať parametre denného stretnutia. Prvým parametrom je **nastavenie času**, o koľkej sa má denný standup začať každý pracovný deň. Na obrázku 4.9 je to prvá požiadavka Scrum mastra. Vďaka algoritmom spracovania prirodzeného jazyka dokáže bot extrahovať čas z vety a aj kontext, či sa jedná o nastavenie času. Alica vždy prepíše predchádzajúce nastavenie novým a informuje používateľa. Ďalej je možné **pridať**, respektíve **odobrať členov** tímu denného standupu (3). Ak vo vete použijeme kľúčové slovo **add**, alebo jej synonymá, môžeme ďalej vymenovať zoznam osôb, ktoré majú byť pridané. Podobne je to aj s odstránením osôb. Pre denné stretnutia je dôležitý aj **kanál**, do ktorého Alica pridáva reporty členov tímu po zodpovedaní všetkých otázok (5). Reporty členov budú podrobnejšie rozobraté v ďalšej podkapitole. Scrum master má možnosť **začať denné stretnutie** (7) príkazom

`start standup`, nezávisle od nastaveného času, alebo ho ukončiť príkazom `cancel`.



Obr. 4.9: Riadenie denného standupu - súkromná konverzácia

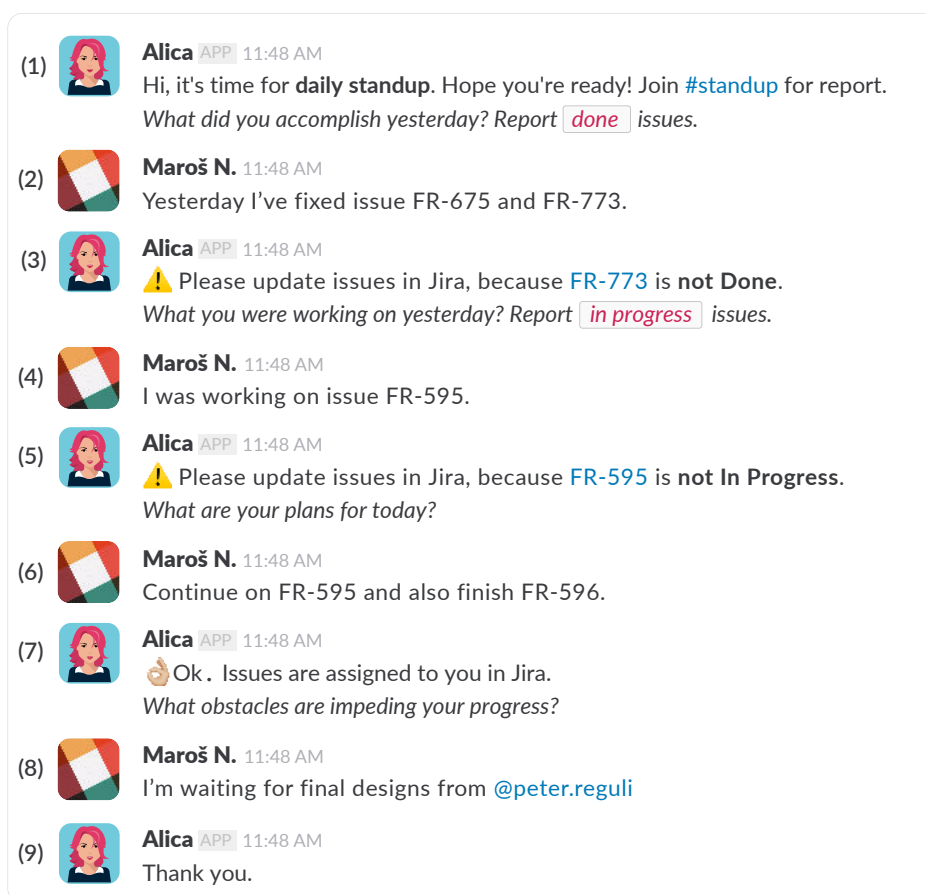
Priebeh standupu

Po spustení standupu (manuálne Scrum mastrom, alebo automaticky), každý člen tímu obdrží súkromnú správu od Alice (obr. 4.10). Táto konverzácia sa skladá zo štyroch otázok:

1. *Ktoré úlohy si včera dokončil?* Pre prehľadnosť sme rozdelili otázku včerajšieho progresu na dve samostatné otázky. V prvej Alica vyzýva člena tímu o popis úloh, ktoré dokončil (1). Bot spracuje textovú odpoveď (2) a porovná spomenuté úlohy s Jirou, či sú naozaj dokončené a aktualizované. V prípade, že dokončené nie sú, informuje člena tímu aby ich aktualizoval (3). Táto **pridaná hodnota umelej inteligencie** zjednodušuje prácu Scrum mastra a pomáha udržiavať stavy úloh aktualizované.
2. *Na ktorých úlohách si včera pracoval, ale ešte nie sú hotové?* V druhej otázke bot požaduje úlohy, ktoré sú 'in progress' (3). Podobne ako pri prvej ich skontroluje a informuje člena tímu, aby ich prípadne označil, že na nich pracuje (5).
3. *Aké sú tvoje dnešné plány?* V tretej otázke Alica kontroluje, či plánované úlohy

sú priradené práve jemu (7). Samozrejme, používateľ nemusí spomenúť žiadne úlohy. V tomto prípade členovi tímu pripomenie všetky, ktoré má priradené v Jire so statusom 'in progress'. Bot tak nielen **kontroluje progres**, ale aj pripomína a motivuje k dokončeniu ďalších úloh.

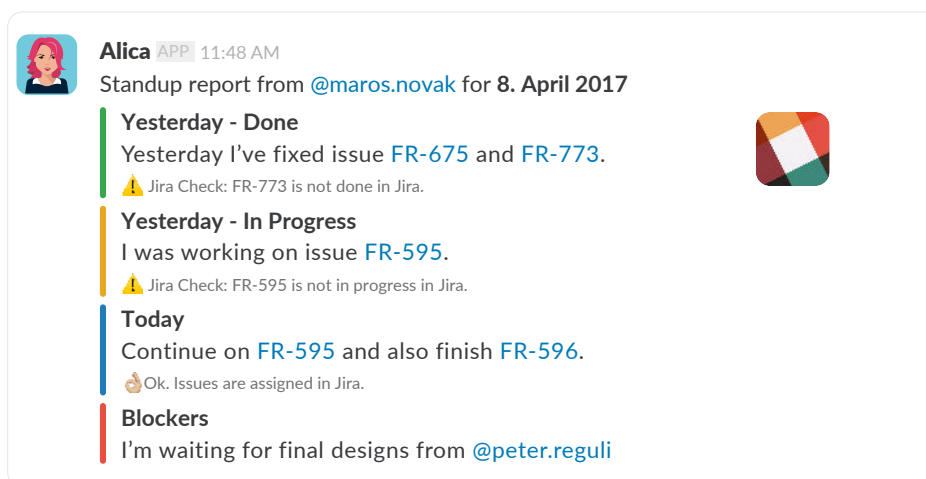
4. *Aké prekážky bránia tvojmu progresu?* V poslednej otázke Alica zisťuje, či má člen tímu nejaké **blokujúce prekážky**, ktoré by mu mohli brániť vo vývoji (8). Scrum master po skončení denného standupu dokáže identifikovať problémy a následne ich riešiť.



Obr. 4.10: Priebeh denného standupu - súkromná konverzácia

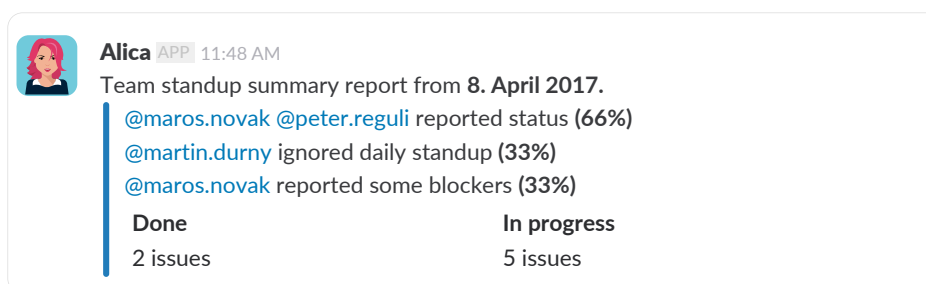
Každý člen môže otázku preskočiť príkazmi **skip**, **no**, **nope**, **nothing**, **none** alebo ignorovať celé stretnutie príkazom **ignore**. Po ukončení konverzácie s členom tímu, Alica spracuje všetky odpovede a vytvorí **prehľadný report**, ktorý vypíše do zvoleného kanálu (obr. 4.11).

V reporte sa nachádzajú odpovede člena aj s tzv. **Jira Check** riadkom. V ňom je zhrnutá kontrola úloh, ktorú Alica vykonáva po odpovedaní na otázky. Tento report



Obr. 4.11: Report člena tímu - konverzácia v kanáli

je súčasťou spoločného kanálu a každý ho môže vidieť. Pokiaľ je niektorý používateľ označený v cudzom reporte, napríklad že blokuje niekoho iného, je o tom notifikovaný.

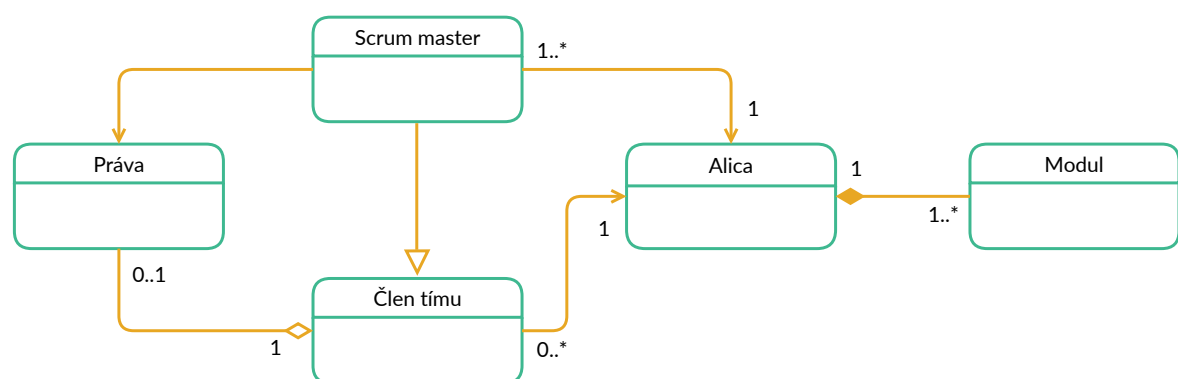


Obr. 4.12: Sumár denného standupu - súkromná konverzácia

Poslednou funkciou v rámci modulu denný standup je odoslanie **sumáru** Scrum mastrovi po skončení stretnutia (obr. 4.12). Alica v ňom spomenie používateľov, ktorí sa zúčastnili a odpovedali na otázky aj tých, ktorí ignorovali. Ďalej oznámi Scrum mastrovi kto nahlásil blokovanie jeho progresu a koľko úloh je dokončených od posledného stretnutia a na koľkých sa pracuje.

4.2.5 Nastavenia systému

Po návrhu modulov sme zistili, že potrebujeme niekoľko príkazov na ich nastavenia a udelenie práv používateľom. Scrum master v navrhovanom systéme môže predstavovať aj úlohu administrátora. Aj keď sa bežne v agilnom tíme nachádza iba jeden Scrum master, v navrhnutom riešení dokáže priradiť rovnaké práva viacerým členom, čo môže byť vhodné pri zastupovaní Scrum mastra (napríklad v prípade dovolenky, choroby, ...). Alica predstavuje aplikáciu, ktorá sa skladá z modulov a je obsluhovaná Scrum mastrom a členmi tímu. Každý z členov tímu môže mať pridelené práva od Scrum mastra. (obr. 4.13).



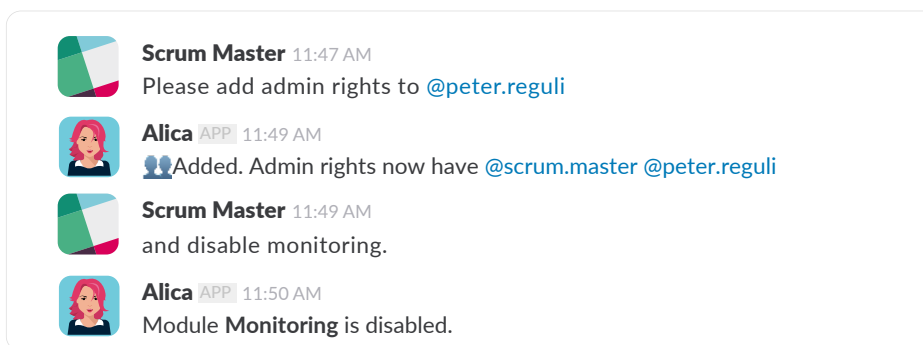
Obr. 4.13: Doménový model bota Alica

Prvou funkciou je **nastavenie administrátorských práv** členovi tímu (obr. 4.14). Napríklad, ak nie je Scrum master k dispozícii pri plánovaní, alebo je potrebné spustiť denný standup, môže ho nahradiť iný člen. Kľúčovým slovným spojením je `add admin rights` alebo jeho varianty. Alica rozpozná v texte označených členov a pridelí im práva. Samozrejme je možné aj členov odobrať pomocou príkazu `remove admin rights`. Alicinou odpoveďou je aktualizovaný zoznam.

Každý z modulov je taktiež možné **zapnúť** alebo **vypnúť**. Alica tak zabráni používaniu príkazov daného modulu členom tímu.

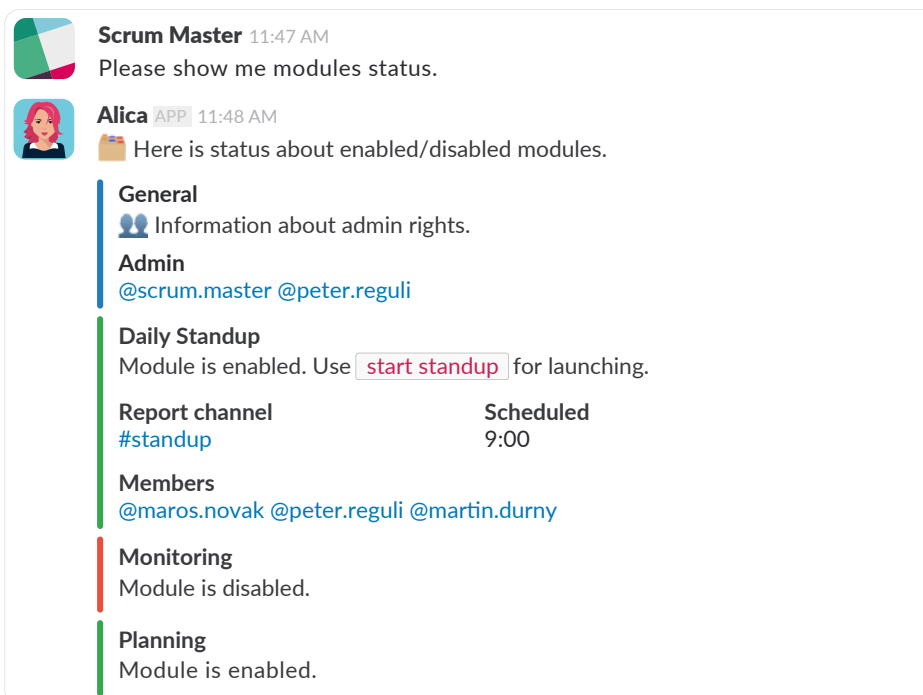
Ďalšou funkciou nastavenia systému je **prehľad modulov** (obr. 4.15). Pomocou kontextu `modules status` môže Scrum master požiadať o zoznam aktívnych a neaktívnych modulov. Správa od Alice obsahuje všeobecné informácie (General) o pridelených právach. Modul denný standup zahŕňa členov tímu, čas opakovania a výstupný kanála pre reporty. Ostatné dva moduly informujú používateľa, či sú aktívne alebo neaktívne.

Alica počúva aj na špeciálny príkaz `help`, po ktorom používateľovi zobrazí príklady



Obr. 4.14: Priradenie práv a vypnutie modulu - súkromná konverzácia

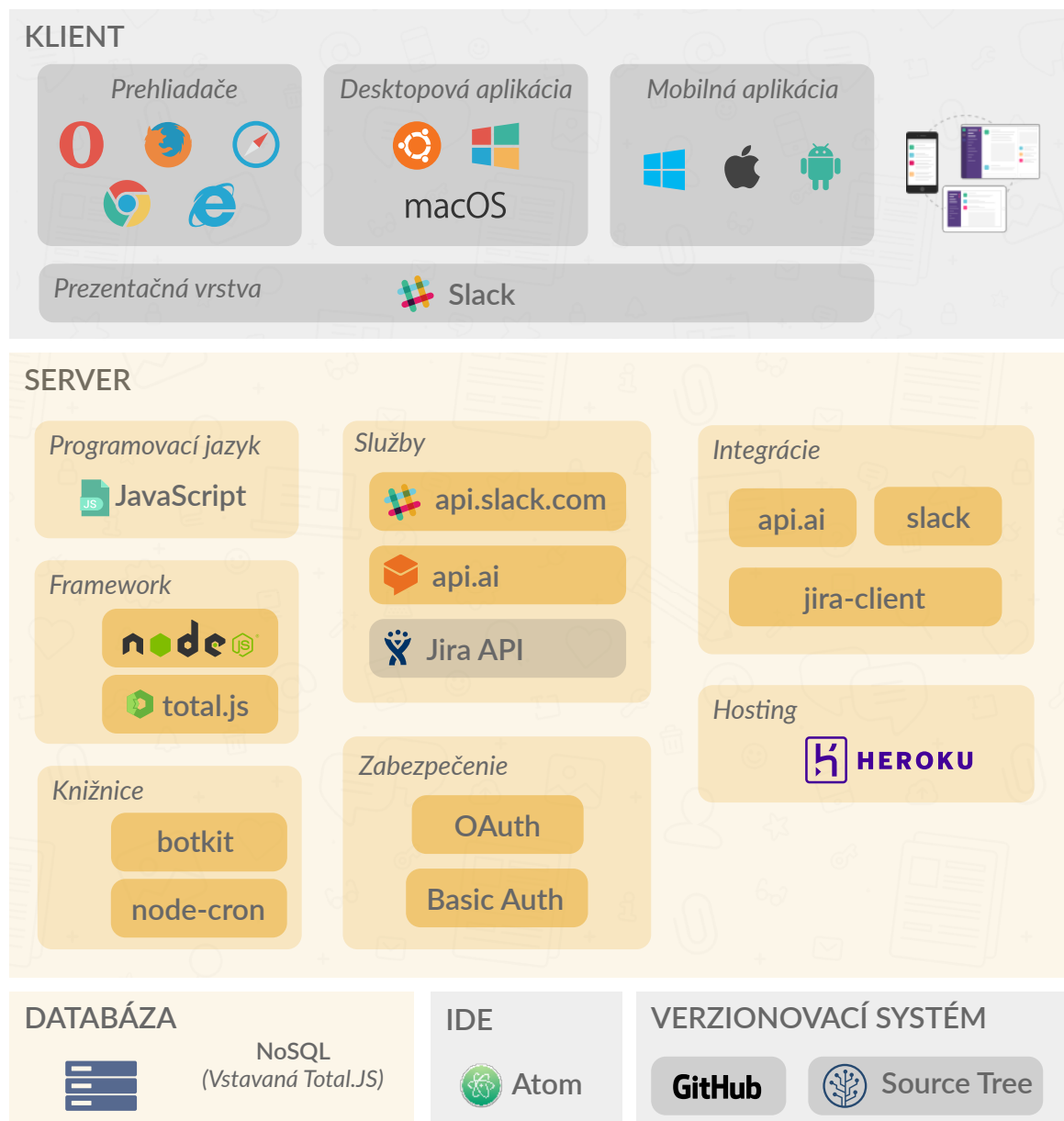
správ, podobne ako boli uvedené v tejto kapitole. Okrem toho je možné s Alicou rozprávať o bežných veciach, zaujímať sa o jej pocity, pýtať sa na jej vek alebo pôvod. Vďaka pred-programovanému modulu **small talk** dokáže viesť neformálnu konverzáciu.



Obr. 4.15: Prehľad modulov - súkromná konverzácia

4.3 Popis technológií

Niektoré technológie už boli popísané v kapitole 3.3. V tejto časti ich zjednotíme pomocou prehľadnej štruktúry nazývanej technologický 'stack' (obr. 4.16). Ten predstavuje kombináciu programovacích jazykov, aplikácií, služieb, knižníc a pod., potrebných pri vývoji. Diagram je farebne odlíšený na **šedú a oranžovú časť**. Šedá časť predstavuje oblasti, ktoré nebolo potrebné nijakým spôsobom nastavovať, modifikovať alebo programovať, jednoducho boli využívané pri tvorbe bota. Oranžová časť je tvorená serverom a databázou, teda súčasťami, ktoré sme na začiatku práce nemali k dispozícii.



Obr. 4.16: Popis technológií

4.3.1 Klientská časť

Výhodou chatbota vo všeobecnosti je, že má v každom komunikačnom nástroji zabezpečenú prezentačnú vrstvu. V prípade nástroja Slack sa jedná o graficky prepracované chat okno, vďaka ktorému sa nemusíme zaoberať vytváraním rozhrania, napríklad pomocou značkovacích jazykov a kaskádových štýlov. Druhou obrovskou výhodou je, že náš bot je dostupný na **všetkých platformách** na ktorých aj Slack. Funkcie bota sú teda len vďaka serverovej časti dostupné vo webovej (Opera, Mozilla, Chrome, Safari, Internet Explorer), desktopovej (Linux, Windows, macOS) alebo mobilnej verzii (Windows Phone, iOS, Android) komunikačného nástroja.

4.3.2 Serverová časť

Serverová časť predstavuje našu chatbot aplikáciu, ktorá bude obsluhovať požiadavky členov tímu a Scrum mastra.

Programovací jazyk a frameworky

Pre vývoj sme vybrali programovací jazyk **JavaScript**, na ktorom je postavený framework **node.js** a total.js. Internetový roboti pre komunikačné nástroje majú najväčšiu podporu pre vývojárov práve pre jazyk JavaScript a platformu node.js. **Total.js** je náhradou za robustný framework express, ktorý bol potrebný k plnohodnotnému vývoju aplikácií a obsahoval mnoho závislostí na ďalších knižniciach.

Knižnice

Medzi potrebné knižnice však patrí botkit a node-cron. **Botkit** sme už spomínali v kapitole 3.3.1 ako dostupný prostriedok pre vývojárov chatbotov. Narozdiel od iných prostriedkov, botkit je plnohodnotná knižnica postavená na platforme node.js, ktorá zapudzuje komunikáciu s chat aplikáciou a poskytuje rozhranie pre jednoduchú výmenu správ. Knižnica **node-cron** je využívaná pre plánovanie a opakovanie akcií autonómne, napríklad pri začatí denného stretnutia každý pracovný deň o rovnakom čase.

Služby a integrácie

Službu v serverovej časti môžeme rozumieť aj ako externé prostredie, ktoré bot využíva vďaka integráciám. V prípade Slacku bolo potrebné bota vytvoriť a pridať do projektového tímu. Vďaka slack integrácií má bot prístup k **api.slack.com**, odkiaľ môže získať údaje o členoch tímu alebo komunikačných kanáloch. **Api.ai** platforma bola popísaná v samostatnej podkapitole 3.3.2. Naša aplikácia môže využívať toto prostredie naplno až po vytvorení agenta a zámerov. Tieto dve služby bolo potrebné nakonfigurovať tak, aby správne fungovali pre naše účely. Poslednou využívanou službou je **Jira**, ktorá obsahuje používateľov, úlohy, šprinty a pod., ku ktorým sa bot vďaka **jira-client** integrácii pripojí. Na obrázku 3.3 je zobrazená šedou farbou, pretože ju nebolo potrebné nijakým spôsobom upravovať.

Zabezpečenie a hosting

Komunikácia medzi chatbotom a používateľmi Fri-Skill v nástroji Slack je zabezpečená pomocou overovacieho protokolu **OAuth**. Nie je preto možné získať tímovú komunikáciu bez členstva a autorizácie. V prvom kroku je server spojený pomocou **Client ID** a **Client secret**. Po spojení je bot inicializovaný vďaka **Bot User OAuth Access Token**. Jira v rámci projektu Fri-Skill využíva školský hosting, ktorý nie je zabezpečený SSL certifikátom. Má pridelenú katedrovú doménu a port. Prístup do systému je možný pomocou používateľského mena a hesla. Preto sa v aplikácii využíva **Basic Auth** k získaniu dát.

Hosting je zabezpečený na platforme Heroku. Publikovanie a aktualizovanie zdrojového kódu je jednoduché pomocou gitu. Aplikácia tak beží v zabezpečenom cloude a k dispozícii je aj HTTPS doména.

4.3.3 Databáza

Navrhovaný bot nevyžaduje komplexné databázové riešenia. Na uchovanie používateľov a informáciach o moduloch stačí aj jednoduchá NoSQL vstavaná databáza, ktorá je súčasťou frameworku total.js.

4.3.4 Vývojárske prostredie

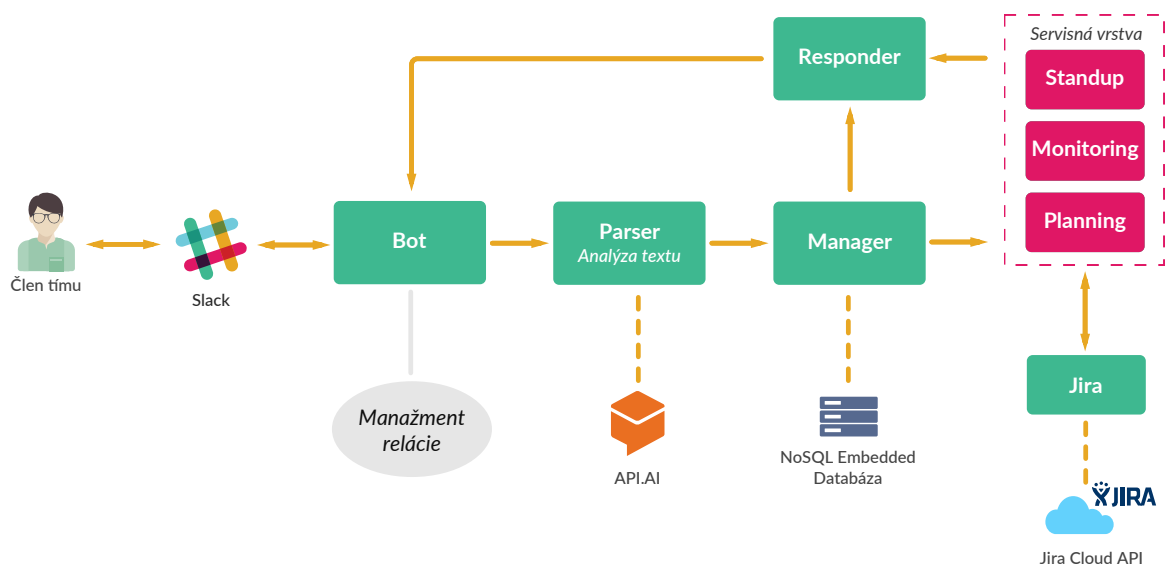
Na vývoj sme využili aplikáciu Atom. Poskytuje prehľadné rozhranie, možnosti pridania rôznych balíčkov a aj podporu verzionovacieho systému.

4.3.5 Verzionovací systém

Na uchovanie, aktualizovanie a správu zdrojového kódu sme využili systém **git**. Aplikácia sa nachádza v službe github, ktorá poskytuje úložisko zadarmo, pokiaľ je aplikácia dostupná na prezeranie. Jednotlivé git príkazy (**commit**, **push**, **pull** a pod.) boli realizované v produkte Source Tree od spoločnosti Atlassian. Ten ponúka prehľadné rozhranie a intuitívnu správu.

4.4 Konceptuálny návrh architektúry

V kapitole 3.3.1 sme opisovali všeobecnú architektúru bota (obr. 3.11) aj so životným cyklom prijatej správy od používateľa. Túto architektúru je však potrebné pre potreby nášho riešenia rozšíriť. Obrázok 4.17 predstavuje navrhovaný model architektúry riešenia. Pvvým rozšírením je pridanie **manažéra** (manager). Tento komponent má na starosti komunikáciu so vstavanou databázou, v ktorej sú uložené informácie o moduloch a používateľoch. Na základe dát obdržaných od prvku Parser rozhodne, akú požiadavku používateľa má obslúžiť. Manažér môže ďalej poslať dáta do Respondera, alebo **servisnej vrstvy**. Tá predstavuje druhé rozšírenie. Servisná vrstva bude obsahovať moduly (sadu funkcií) a komunikovať s nástrojom **Jira**, pomocou rovnomeného komponentu. Ten je spojený s Jira Cloud API a parsuje dané požiadavky a poskytuje interface spracovaných dát pre moduly.



Obr. 4.17: Konceptuálny návrh architektúry riešenia

Responder následne prijíma správy od Manažéra alebo modulov, kde sa sformuluje odpoveď. Posledným pridaným prvkom je služba **api.ai**. Parser vďaka tejto službe extrahuje dáta zo správy v prirodzenom jazyku a odovzdá ich manažérovi. Pomocou tejto architektúry udržíme zapuzdrené informácie v komponentoch, ktorých sa týkajú.

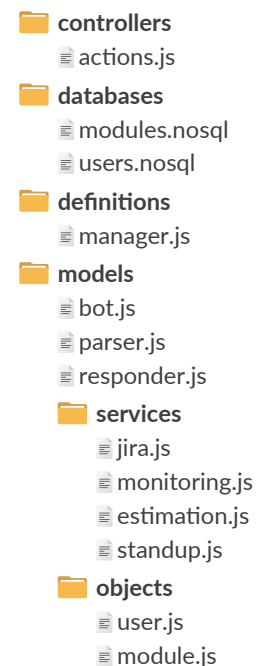
4.5 Implementácia komponentov

V tejto kapitole sa budeme zaoberať jednotlivými komponentami a ich implementácií. Komponent (prvok) by sme v tejto časti mohli označovať aj slovom trieda, avšak JavaScript nie je objektovo-orientovaný jazyk, takže si pod komponentom môžeme predstaviť logickú časť programu, ktorá obsahuje funkcie a atribúty. Namiesto tried budeme používať tzv. **schémy**, ktoré definuje framework total.js.

4.5.1 Štruktúra projektu

Predtým, ako sa dostaneme k jednotlivým komponentom, zdefinujeme štruktúru projektu, ktorú určuje framework total.js (obr. 4.18). Všetky priečinky sú voliteľné a po spustení aplikácie sa inicializujú v nasledovnom poradí: **modules** > **models** > **definitions** > **controllers**.

Ako prvé sú to **moduly (modules)**. Tie nie sú v štruktúre napravo zahrnuté, pretože predstavujú integrácie a knižnice, pridané do projektu pomocou balíčkového manažéra **npm** a príkazu `npm install {balíček} --save`. Po tomto príkaze je knižnica pridaná do zoznamu závislostí v súbore `package.json`. Tento súbor okrem externých balíčkov obsahuje aj informácie o autorovi, názov aplikácie, umiestnenie git repozitára a podobne. Ďalšie sú **modely (models)**. Túto časť sme rozdelili na služby (**services**), ktoré obsahujú súbory servisnej vrstvy a objekty (**objects**), kde sa nachádzajú schémy reprezentujúce databázu. Modely obsahujú hlavné komponenty aplikácie: **bot**, **parser** a **responder**. **Definície (definitions)** obsahujú jediný súbor - **manager**. Tento súbor je inicializovaný automaticky a využíva už existujúce modely. Definície predpisujú správanie celej aplikácie. Ako posledné sú inicializované **radiče (controllers)**. Sú zodpovedné za vybavovanie požiadaviek z webového prehliadača, mobilnej aplikácie alebo akéhokoľvek iného systému, ktorý je schopný komunikovať so serverom. V našom prípade je to aplikácia Slack.

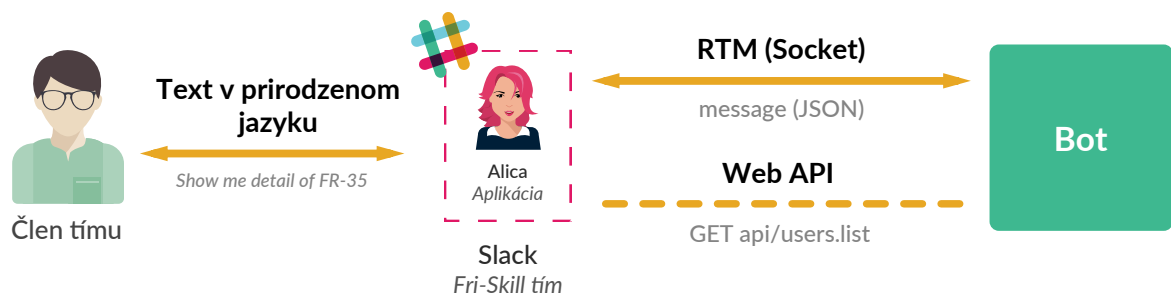


Obr. 4.18: Štruktúra projektu

Posledným priečinkom v štruktúre sú **databázové súbory (databases)** s príponou `.nosql`. Tieto súbory nie je potrebné inicializovať, obsahujú dáta vo formáte JSON a sú reprezentované objektami `module` a `user`. Okrem všetkých spomínaných priečinkov obsahuje aplikácia aj dva hlavné súbory `debug.js` a `release.js`. Vďaka nim je možné aplikáciu spustiť v ladiacom alebo nasadenom prostredí.

4.5.2 Bot

Prvým krokom implementácie bolo vytvorenie aplikácie na platforme `api.slack.com` (pozri prílohu B). Po vytvorení takejto aplikácie pre nás slack vygeneroval všetky kľúče a tokeny, potrebné pre nadviazanie komunikácie. Vygenerovanú aplikáciu sme pridali do tímu `friskill.slack.com`. Alica sa následne stala súčasťou tímu a bolo ju možné nájsť v zozname používateľov, avšak zatiaľ v režime offline. V nasledujúcich kapitolách budeme rozlišovať medzi aplikáciou (integráciou) Alica, vytvorenou v rámci tímu a naším serverom.



Obr. 4.19: Komunikácia medzi aplikáciou Alica a serverovým komponentom Bot.

Druhým krokom bolo nadviazanie komunikácie medzi naším serverom a Slackom. Framework `total.js` definuje v rámci schém okrem funkcií aj tzv. **workflows** a **operations**, ale pre jednoduchosť budeme hovoriť všeobecne o funkciách. V schéme Bot, ktorý zastrešuje komunikáciu, sme vytvorili funkciu `connect`, určenú na inicializáciu dvoch premenných. Jedna je inštanciou knižnice `botkit`, ktorá vďaka `Client_ID` a `Client_Secret` sprostredkuje bezpečné spojenie. Druhá je samotný bot, ktorý ako parameter využije `OAuth Access Token`. Po týchto nastaveniach a spustení servera sa Alica v komunikačnom kanáli Slack prepne z režimu offline do online a náš server je schopný prijímať správy.

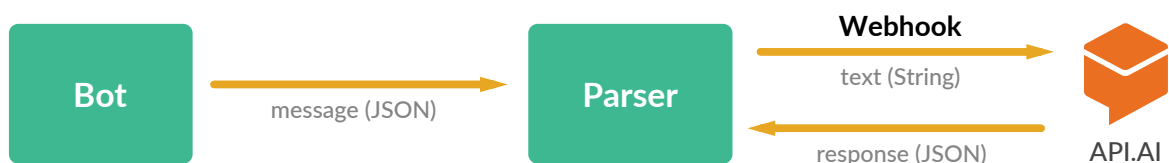
Na obrázku 4.19 je znázornená komunikácia medzi koncovým používateľom, integráciou Alica a naším vytvoreným komponentom Bot. Člen tímu si vymieňa s Alicou textové správy, ktoré sú ďalej zabalené do štruktúry dát vo formáte JSON a odoslané pomocou otvorenej socketovej komunikácie RTM (Real Time Messaging). Tieto dáta obsahujú: originál správy, id odosielateľa, id kanálu, id tímu, časovú pečiatku a typ správy. Existujú 4 typy správ: `direct_message`, `direct_mention`, `mention` a `ambient`. Pre naše účely postačuje, aby server počúval na priame správy (`direct_message`) a označenie v kanáli (`direct_mention`).

Vytvorená inštancia `bot` vie pristupovať aj k Web API pomocou REST služieb. Jednou z využívaných požiadaviek je zoznam používateľov v rámci tímu. Komponent Bot obsahuje ďalšiu inštanciu, ktorá je jediná v celej aplikácii s názvom `parser`. Po prijatí správy posieľa dáta priamo tomuto komponentu. Poslednou súčasťou schémy je sada funkcií obsluhujúcich komunikáciu:

- `reply` - odpovedanie používateľovi (napr. reakcia na správu)
- `postMessage` - odoslanie správy (napr. do spoločného kanálu)
- `startConversations` - spustenie súkromných konverzácií (napr. denný standup)
- `askQuestion` - polozenie otázky (napr. v rámci denného standupu)

4.5.3 Parser

Komponent Parser má jednoduchú úlohu. Po obdržaní objektu `message` nadviaže komunikáciu so službou `api.ai` a odošle prijatý text na analýzu (obr. 4.20). Po získaní odpovede vytvorí JSON objekt s názvom `intent`, ktorý sa odošle manažérovi.



Obr. 4.20: Komunikácia medzi komponentom Parser a službou `api.ai`

V službe `api.ai` sme sa najskôr bezplatne zaregistrovali. Následne sme vytvorili agenta, ktorý bude pokrývať všetky naše požiadavky. Po vytvorení agenta nám služba

vygenerovala `Client access token`, ktorým sa autorizujeme voči službe. Zoznam všetkých zámerov sa nachádza v prílohe C. Náš Parser obsahuje inštanciu knižnice `apiai`, pomocou ktorej môžeme komunikovať. Tento typ komunikácie sa nazýva **webhook**. Ak nastane nejaká udalosť, webhook odošle HTTP POST na náš server. Tou udalosťou je prijatie nášho textu agentom v externej službe. Po obdržaní odpovede, Parser vytvorí objekt `intent` (obr. 4.21), v ktorom využije zanalyzované dáta (`module`, `type`, `parameters`). Tieto tri atribúty spolu s originálom správy (`message`) sú povinné. Odpoveď na otázku počas standupu obsahuje aj objekt `conversation`, v ktorej sú všetky dáta súkromnej konverzácie medzi botom a členom tímu, a ďalšie Boolean hodnoty.

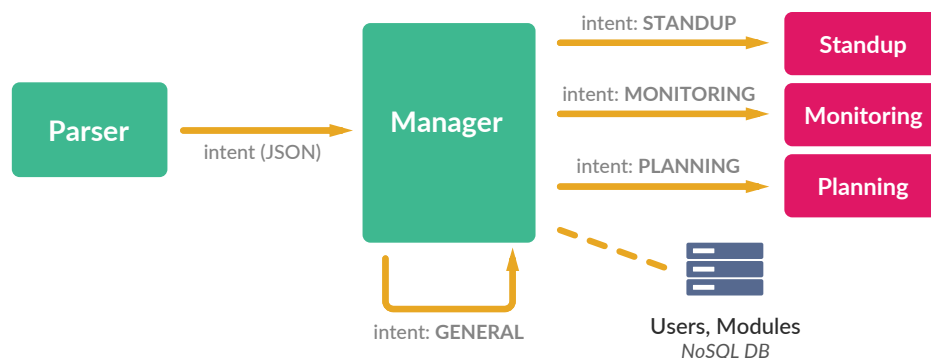
```
{
  module: 'STANDUP',
  type: 'STANDUP-ANSWER',
  canceled: false,
  skipped: false,
  ignored: false,
  message: message,
  conversation: conversation,
  parameters: {
    issues: ['FR-53']
  }
}
```

Obr. 4.21: Príklad objektu `intent` pre denný standup a monitorovanie úloh

Atribút `parameters` môže obsahovať rôzne údaje - zoznamy úloh, používateľov, čas, dátum, komunikačný kanál a podobne. Pri analýze textu a extrahovaní týchto údajov nemusí byť služba `api.ai` úplne presná. Pre tento prípad sme v Parseri využili aj **regulárne výrazy**, pre kontrolu kľúčov úloh. Ak sa majú nachádzať v texte úlohy, napr. *Show me detail of FR-35*, prejdeme pole v parametroch, každý objekt upravíme na malé písmená a skontrolujeme regulárnym výrazom.

4.5.4 Manager

Manager môžeme označovať aj ako 'mozog' celého servera. Po inicializácii spustí `connect` funkciu bota a počúva na prijatie `intentu`. `Intent` má definované 4 moduly: `general`, `standup`, `monitoring` a `estimation` (obr. 4.22). `General` je určený iba komponentu Manager za účelom nastavenia práv, správu modulov a podobne. Ostatné tri spúšťajú aktivity servisnej časti. Atribút `type` určuje konkrétnu funkciu modulu.

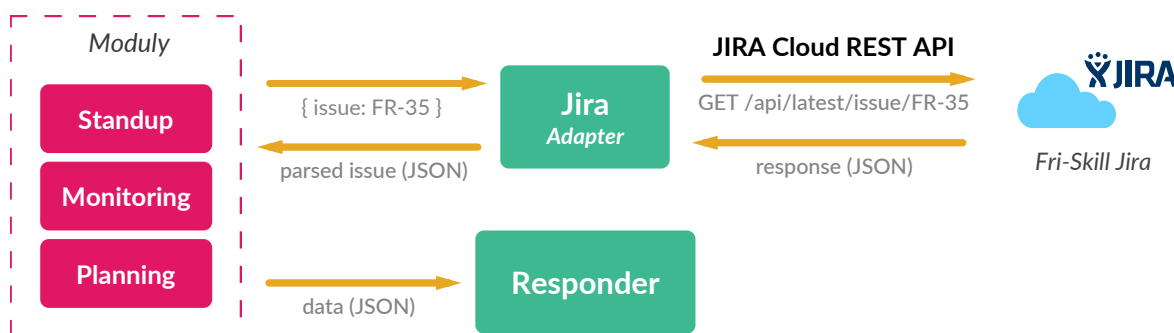


Obr. 4.22: Komunikácia komponentu Manager

Manažér má niekoľko úloh pred volaním funkcie servisnej vrstvy. Prvou je kontrola, či je daný modul aktívny. Po výbere z databázy môže byť objekt odoslaný inštancii modulu. Napríklad v prípade denné standupu sú v ňom uložené informácie o členoch, čase a pod. Druhou úlohou je overenie používateľa, ktorý príkaz zadal, či má oprávnenie. Taktiež ako v prípade modulu je objekt vybraný z databázy na základe identifikátora, ktorý je súčasťou `message`. Treťou úlohou je odchyťovanie chýb, ktoré môžu byť súčasťou `callback` funkcie niektorého z modulov. Manažér vtedy volá Responder na odoslanie informácie o chybe.

4.5.5 Servisná vrstva

Všetky funkcie, spracovania požiadaviek, analýzy denných stretnutí a podobne sú zapuzdrené v tejto časti. Servisná vrstva sa skladá z jednotlivých modulov, z ktorých každý obsahuje inštanciu adaptéra Jira a inštanciu Respondera. Na obrázku 4.23 je znázornená komunikácia a typy objektov, ktoré sa medzi prvkami posielajú. Jednotlivé komponenty si detailnejšie popíšeme v tejto kapitole.



Obr. 4.23: Komunikácia medzi komponentami servisnej vrstvy

Jira

Jira využíva knižnicu `jira-client`, ktorá zapuzdruje volania API. K inicializácii sú potrebné atribúty pre nadviazanie spojenia: `protocol`, `host`, `port`, `username` a `password`. Jira komponent je navrhnutý ako návrhový vzor **Adapter**. Princípom adaptéra je zmeniť rozhranie jednej triedy na iné, univerzálnejšie. Vďaka nemu môžeme vytvoriť univerzálny interface, ktorý môže využívať servisná vrstva aj po zmene nástroja na riadenie agilných tímov. Jira tak môže byť nahradená iným softvérom, ktorý poskytuje API rozhranie, ale interface adaptéra sa nezmení, iba spôsob parsovania dát. Výhodou adaptéra je, že odfiltruje nepotrebné parametre, ktoré obdržíme z Jira API a tak môžeme vytvoriť čisté a konzistentné rozhranie. Náš navrhnutý interface obsahuje nasledujúce funkcie:

- `getIssues{issues[]}` - vráti zoznam úloh (v prípade jednej, detail úlohy)
- `getUsersIssues{email}` - vráti priradené úlohy používateľa
- `getSprintIssues` - vráti úlohy aktuálneho šprintu
- `getContextIssues` - vráti úlohy, v ktorých boli zaznamenané zmeny za posledné dva dni
- `addComment{issueKey, text}` - pridá komentár k úlohe
- `assign{issueKey, email}` - priradí úlohu používateľovi
- `changeStatus{issueKey, status}` - zmení status úlohy
- `setStoryPoints{issueKey, points}` - nastaví body používateľského príbehu

Každá funkcia má zadanú štruktúru výstupu (**response**), aké informácie majú dáta obsahovať, aké atribúty majú úlohy a či volanie prebehlo bez problémov.

Monitoring

Jira je najviac využívaný komponent modulu Monitoring. Jej úlohou je volanie funkcií Jira rozhrania, kontrola vstupných atribútov a odpovedí. Po získaní odpovede volá funkciu Respondera, ktorý dáta spracuje do výslednej správy. Monitoring je hlavne zobrazovací modul, neobsahuje veľa logiky.

Standup

Komplexnejším modulom je standup. Pokrýva všetky funkcie popísané v kapitole 4.2.4. Manager po spustení denného standupu vytvorí práve jednu inštanciu tohto modulu, ktorá existuje až do odpovedania všetkých otázok všetkými členmi, alebo manuálneho ukončenia. Standup obsahuje funkciu `startStandup`, ktorá sa zavolá po inicializácii. V nej sa nastaví potrebné parametre, vytvoria súkromné konverzácie s každým členom tímu a stiahnu kontextové úlohy. Naš server musí obsluhovať paralelne viacero požiadaviek naraz, preto sme museli vytvoriť asynchrónne konverzácie s každým používateľom, ktorého sa denné stretnutia týkajú. Diagram aktivít konverzácie sa nachádza v prílohe D. Po spustení konverzácií členovia obdržia prvú otázku a server čaká na odpovede. Slack pri odpovedi posiela aj objekt `conversation`, v ktorom je uložená celá konverzácia a vďaka ktorej vieme odlišiť jednotlivých používateľov. Tieto konverzácie sú uložené v module. Konverzácia je odstránená z pola po tom, ako člen tímu odpovie na poslednú otázku, alebo pošle príkaz `ignore`. Po odstránení všetkých konverzácií sa standup ukončí a odošle sumárna správa Scrum mastrovi.

Modul má uložené aj tzv. **kontextové úlohy**. Keďže server obsluhuje paralelne viacero používateľov, ktorí mohli pracovať na spoločných úlohách, kontextovými úlohami zamedzíme duplicitným požiadavkám z Jira API. Modul si po spustení uloží úlohy, ktoré boli modifikované za posledné dva dni. Veľkú časť z nich totiž môžu členovia tímu spomínať v odpovediach. Úlohy majú iba potrebné atribúty na ich kontrolu - priradeného používateľa (`assignee`), status a kľúč úlohy (`issueKey`). Na nasledujúcom príklade si ukážeme, akým spôsobom je spracovaná jedna odpoveď člena tímu. Alica v tomto prípade bude reprezentovať komplet aplikáciu aj s našou serverovou časťou. V príklade je tretia otázka v poradí, ktorá zisťuje, čo má člen tímu v pláne dnes a kontroluje jeho priradené úlohy.

1. Alica odošle otázku (*Aké sú tvoje dnešné plány?*)
2. Člen tímu odpovie na otázku
3. Ak sa v odpovedi nachádzajú úlohy
 - 3.1. Ak sa niektoré nenachádzajú v kontextových úlohách
 - 3.1.1. Alica stiahne úlohu a pridá medzi kontextové
 - 3.2. Alica skontroluje odpoveď s kontextovými úlohami
 - 3.3. Ak úlohy nemá priradené
 - 3.3.1. Alica informuje člena tímu, aby si ich priradil
4. Ak sa v odpovedi nenachádzajú úlohy
 - 4.1. Alica stiahne jeho úlohy so statusom 'In Progress'
 - 4.2. Alica informuje člena tímu o jeho rozpracovaných úlohách
5. Alica uloží odpoveď pre report
6. Alica odošle ďalšiu otázku

Modul Standup volá funkcie Respondera v dvoch prípadoch. Prvým je vytvorenie **reportu**, ktorý je odoslaný do spoločného komunikačného kanála. Responder obdrží objekt, v ktorom sú informácie o používateľovi, všetky otázky a odpovede, korektné a nekorektné úlohy. Report je zobrazený na obrázku 4.11.

Druhé využitie prvku Responder je po ukončení standupu. Dáta o všetkých členoch tímu sú ukladané v premennej **reports**. Z nich Responder extrahuje potrebné informácie, aby vytvoril **sumár denného standupu** pre Scrum mastra (obr. 4.12).

Planning

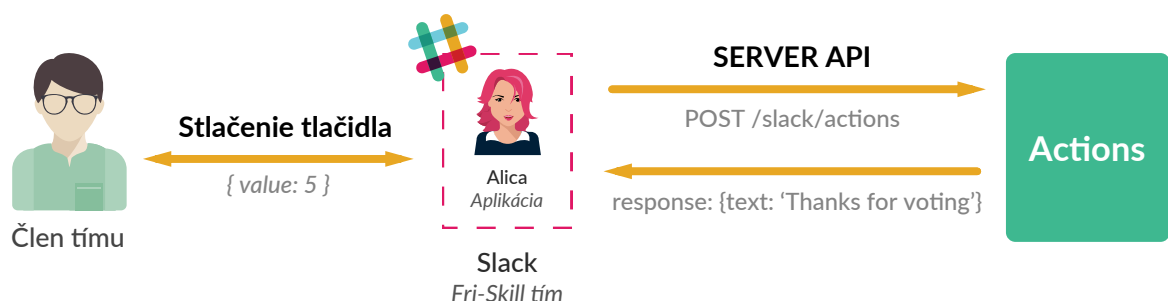
Posledným modulom v servisnej vrstve je Planning. Jednoduchá schéma, ktorá riadi hodnotenie používateľských príbehov. Obsluhuje príkazy, ktoré sú popísané v kapitole 4.2.3. Tak ako ostatné moduly, aj Planning je vytvorený v manažérovi, kde čaká na spustenie svojej funkcie. Plánovanie je možné spustiť iba vo verejnom kanáli, takže ak sa používateľ pokúsi spustiť príkaz v súkromnej správe, Alica ho upozorní aby požiadavku opakoval v niektorom z kanálov. Náš server potrebuje s každým členom,

ktorý je prítomný v komunikačnom kanáli, nadviazať súkromnú konverzáciu a odoslať detail úlohy, o ktorej sa má hlasovať. Súčasťou správy sú tlačidlá s hodnotami story pointov (obr. 4.6).

Po úspešnom spustení hodnotenia sa vykoná niekoľko kľúčových procesov. Najskôr Bot získa zoznam používateľov v komunikačnom kanáli. Tento zoznam môže obsahovať aj botov a iné integrácie, takže je potrebné odfiltrovať iba členov tímu. Výstup je odoslaný modulu Planning, kde môžu byť následne inicializované dve premenné. Prvou je **results**, v ktorej sa nachádza zoznam **users** s hodnotami **key** (id používateľa) a **value** (hodnota). Táto hodnota je pri inicializácii **null** a naplní sa po stlačení tlačidla. Druhý atribút je výstupný kanál (**channel**), do ktorého budú odoslané výsledky. Modul následne získa informácie o používateľskom príbehu pomocou Jira komponentu a odošle Responderu, ktorý zostaví správu. Táto správa je odoslaná všetkým v zozname. Po odhlasovaní každého člena sa spracuje výsledná správa, ktorú Alica vypíše do spoločného kanálu. Spôsob prijatia hodnoty z tlačidla je popísaný v ďalšej kapitole. Po hlasovaní je možné priradiť hodnotu story bodov priamo do úlohy v Jire cez spoločný kanál.

4.5.6 Actions

Komponent Actions je jediný zástupca radiča (controller) v aplikácii. Okrem RTM komunikácie a Slack Web API, dokáže náš server prijímať dáta aj pomocou tohto REST rozhrania (obr. 4.24). Využíva iba HTTP metódu POST a jeho úlohou je spracovať požiadavku, ktorú Slack odošle po stlačení tlačidla (akými sú tlačidlá v rámci planning poker).



Obr. 4.24: REST komunikácia medzi rozhraním Actions a Slack aplikáciou

Predtým, ako controller obdrží dáta o stlačení interaktívneho tlačidla, bolo potrebné vykonať niekoľko nastavení. Slack dokáže odoslať požiadavku iba na HTTPS url,

teda adresu zabezpečenú SSL certifikátom. Na testovacie účely sme využili službu **localtunnel**, ktorá dokázala simulovať takúto adresu pre určitý PORT, v našom prípade 8000. V nasadenom prostredí nám SSL zabezpečil Heroku, ktorý pre každú aplikáciu vytvorí url automaticky s HTTPS. Absolútnu adresu sme zadali vytvorenej integrácii Alica na `api.slack.com` (pozri Príloha B). Naš server bol po tomto kroku schopný prijímať požiadavky na ceste `/slack/actions`. Prijatá požiadavka obsahuje originál správy, hodnotu tlačidla, používateľa aj tzv. **verification token**. Ten je potrebné porovnať s tokenom vygenerovaným po vytvorení integrácie v službe `api.slack.com`. Iba používatelia Slack tímu, ktorý obsahuje bota Alica, môžu odosielať požiadavku na náš server. Po jej prijatí a úspešnej verifikácii je potrebné do troch sekúnd odoslať odpoveď (HTTP 200 OK). Súčasťou odpovede môže byť nová správa, ktorá nahradí pôvodnú s tlačidlami. Komponent Actions odošle prijaté dáta do modulu Planning, aby boli pridané medzi odpovede (**results**). Celý proces plánovania je popísaný sekvenčnými diagramami v prílohe E.

4.5.7 Responder

Poslednou časťou je Responder. Jediná funkcia tohto komponentu je **formátovanie správ**. To sme spomínali v kapitole 3.3.3. Responder poskytuje jednoduchý interface modulom a manažérovi s množstvom typov správ. Každá z funkcií má presne definované vstupné parametre, aby bola schopná poskladať JSON objekt s odpoveďou. Odpoveď je odoslaná v `callback` funkcii cez volaný komponent až do Bota.

4.6 Nasadenie

Nasadenie nášho servera do zvolenej cloudovej služby Heroku bol jednoduchý. Najskôr bolo potrebné sa zaregistrovať a vytvoriť aplikáciu vo webovom rozhraní Heroku. Aplikácia zastrešuje inštanciu nášho servera a poskytuje mnoho nastavení a prehľadov. Po vytvorení aplikácie pre nás Heroku vygeneroval url adresu: `https://alicabot.herokuapp.com/` a nakonfiguroval SSL certifikát.

Celý proces nasadenia sme realizovali cez terminál použitím **git** a **heroku príkazov**. Najskôr bolo potrebné vytvoriť vzdialený repozitár, teda priestor pre zdrojové kódy. Navigovali sme sa do priečinka s našim vytvoreným serverom a zadali príkaz **git add .** nasledovaný príkazom **commit**. Lokálny git repozitár sme mali vytvorený pred nasadením, keďže sme robili priebežnú zálohu zdrojových kódov. Po pridaní tzv. 'commitu' do lokálneho repozitára sme mohli odoslať zdrojový kód do cloudovej služby (**git push heroku master**). Súčasťou priečinka bol aj súbor **.gitignore**, ktorý filtruje nepotrebné súbory. Po nahratí priečinka (verzie), Heroku automaticky vykonal príkaz **node release.js**, teda spustenie súboru, ktorý je súčasťou aplikácie. Predtým však stiahol potrebné závislosti a knižnice. Alica je po týchto jednoduchých krokoch nasadená online. Podobný postup sme vykonávali opakovane kvôli aktualizácii funkcií, ktoré sme implementovali lokálne. Nevýhodou Heroku, na ktorú sme narazili pri neskoršom testovaní bolo, že sa inštancia servera uspí po 30 minútach bez interakcie. Zobudená je po prvej požiadavke, ktorú treba následne opakovať znova.

4.7 Overenie riešenia

Riešenie sme po nasadení otestovali v Slack tíme Fri-Skill. Na testovanie sme vytvorili vlastný kanál pre estimáciu a reporty denného standupu. Chatbot reagoval na príkazy, avšak zo začiatku bolo potrebné zadať viacero variácií textových vstupov, aby algoritmy strojového učenia vedeli spoľahlivejšie analyzovať požiadavky. Okrem predpokladaných odpovedí bol však schopný vypisovať chybové hlášky, ak nestal nejaký problém. Po internom testovaní môžeme vyhlásiť, že internetový chat robot Alica je pripravená na používanie v tíme Fri-Skill.

5. Záver

Cieľom diplomovej práce bolo navrhnuť a implementovať internetového robota, ktorý bude nasadený v komunikačnom prostriedku Slack a bude integrovať nástroj na riadenie agilných projektov Jira. Práca sa predovšetkým zameriava na riešenie problémov v agilných distribuovaných tímoch, akým je aj univerzitný projekt Fri-Skill.

Analytická časť práce je venovaná agilnému vývoju, popisu metodiky Scrum, vysvetleniu pojmov a identifikácii problémových oblastí, ktoré vyplynuli z retrospektív stretnutí projektu Fri-Skill. Na základe výstupov identifikovaných oblastí a nami zvolených hodnotiacich kritérií, boli zadané požiadavky na softvérové riešenie. Po vyhodnotení existujúcich nástrojov a integrácií sme dospeli k záveru navrhnuť vlastného komunikačného bota. V analytickej časti sa ďalej nachádzajú teoretické informácie o internetových robotoch, podpore ich vývoja na platforme Slack, spracovaniu prirodzeného jazyka, možnosti využitia nástroja Jira, použitých programovacích jazykoch a cloudových službách.

Praktická časť začína návrhom príkazov, ktoré by mal byť internetový robot schopný spracovať. Jednotlivé funkcie sme rozdelili na oblasti podľa ich riešenia problematiky na estimáciu, denné stretnutia a monitorovanie priradených úloh. V tejto časti sme sa ďalej zamerali na návrh architektúry a komunikácie internetového bota s externými službami. V závere sa nachádza popis všetkých navrhnutých komponentov z pohľadu implementácie až po nasadenie na platformu Heroku.

Vytvorený internetový robot dokáže komunikovať v prirodzenom jazyku a obsluhovať požiadavky používateľov v nástroji Slack. V rámci denného stretnutia dokáže zastúpiť úlohy Scrum mastra a klásť členom tímu otázky o ich prograse. Jednotlivé odpovede analyzuje, porovnáva s úlohami v nástroji Jira a spracuje prehľadný report pre ostatných členov tímu. Tým sme podstatne zjednodušili procesy v distribuovaných agilných tímoch, kde sa členovia nemôžu fyzicky stretnúť. Vďaka našej integrácii môžu mať používatelia prehľad o ich priradených úlohách alebo aktuálnom šprinte, môžu priradovať a komentovať úlohy priamo v komunikačnom kanáli. Pomocou príkazov estimácie je tím schopný pohodlne hodnotiť používateľské príbehy počas sprint planning stretnutia, taktiež priamo v prostriedku na chatovanie. Súčasťou bota je aj prehľad a nastavenia modulov. Internetový robot môže výrazne sprehľadniť komunikáciu medzi členmi tímu

a je schopný konkurovať iným webovým aplikáciám alebo bot integráciám.

5.1 Možnosti rozšírenia

Naše riešenie bolo potrebné nasadiť a otestovať v projekte Fri-Skill, avšak momentálne nemôže byť distribuované ako verejná integrácia. Publikovanie by teda mohlo byť prvým rozšírením riešenia. V rámci funkcionalít je naozaj veľa možností rozšírenia. Jedným z nesplnených požiadaviek na systém bolo sledovanie burndown grafu. Internetový robot by mohol informovať Scrum mastra o nepriaznivom vývoji vďaka analýzy grafu rýchlosti ubúdania používateľských príbehov. Taktiež Jira ponúka mnoho funkcií, pre ktoré je dostupné aj verejné API, napríklad vytváranie, pokročilé vyhľadávanie alebo úprava úloh.

Literatúra

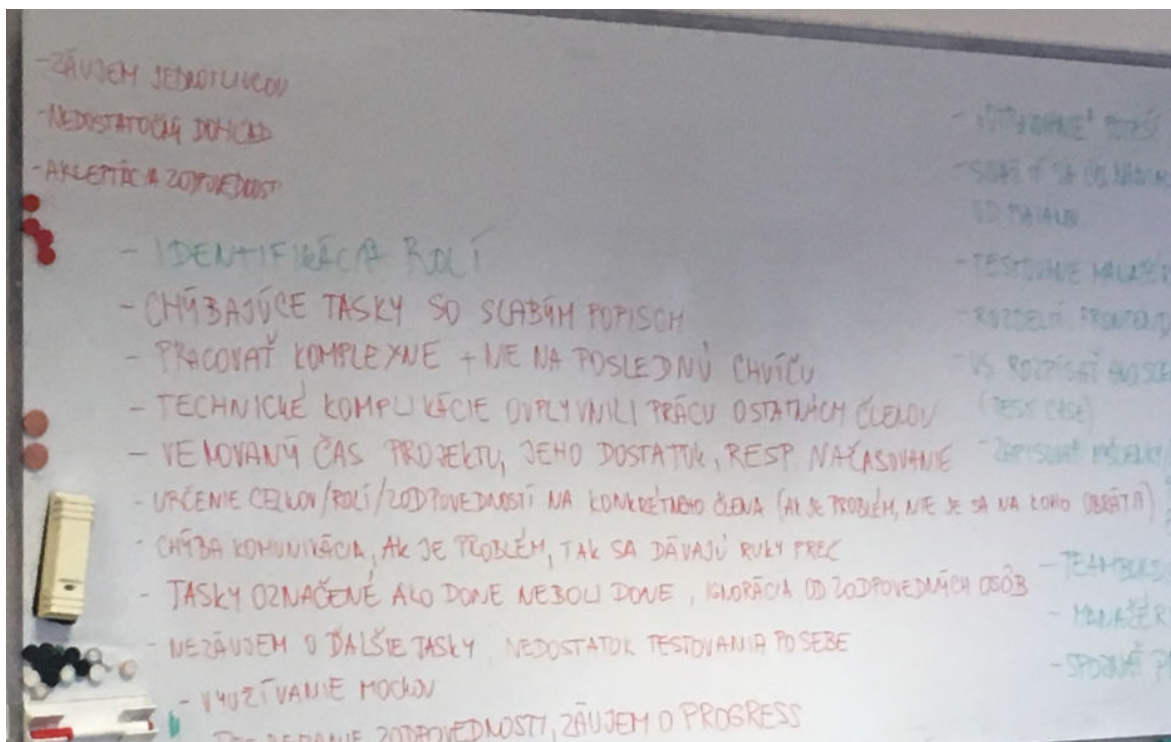
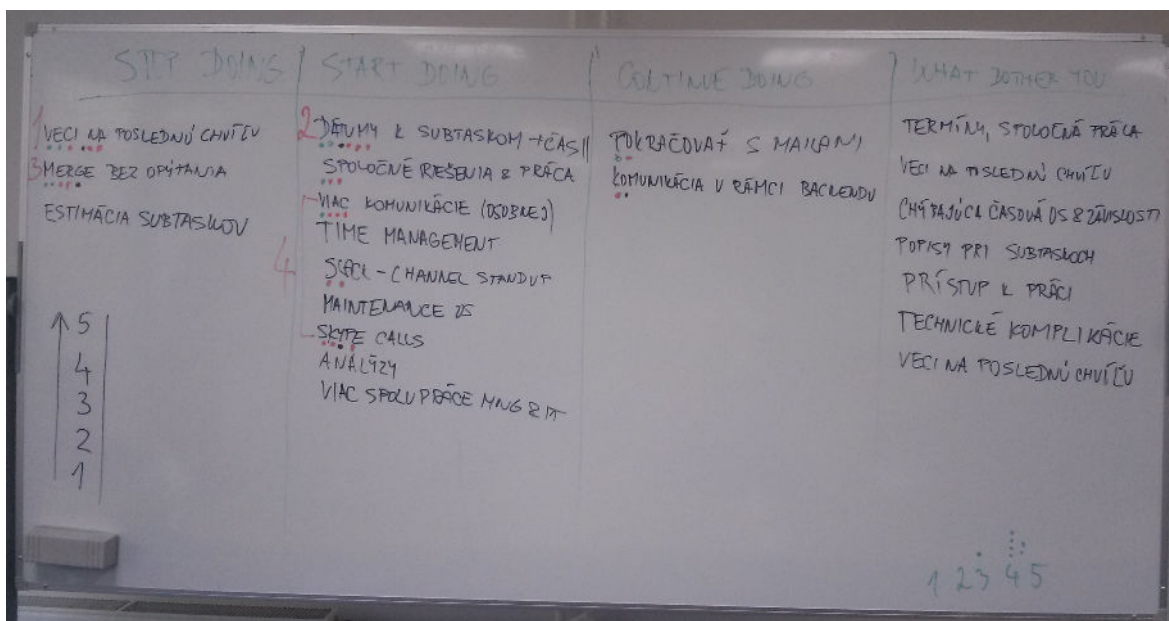
- [1] Stellman, A. Greene, J., *Learning Agile: Understanding Scrum, XP, Lean, and Kanban.*, O'Reilly Media, Inc., 2014, ISBN 978-14-4933-192-4.
- [2] Ashmore, S. Runyan, K., *Introduction to Agile Methods.*, Addison-Wesley, 2014, ISBN 978-03-2192-956-3.
- [3] VersionOne prieskum, Výsledky k 10 ročnému stavu agilných metód, 19.4.2016, https://www.infoq.com/news/2016/04/2015_state_of_agile_results
- [4] Wirdemann, R. *Scrum mit User Stories.*, Carl Hanser Verlag, 2011, ISBN 978-34-4642-660-3.
- [5] Mauldin M., *Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition.*, John Wiley & sons ltd, 1994.
- [6] Bickmore T., *A computational model of small talk.*, Massachusetts Institute of Technology, 1999.
- [7] Wallace, R. S., The Anatomy of A.L.I.C.E., 2009 <http://www.alicebot.org/anatomy.html>
- [8] JIRA User's Guide, Atlassian, 4.3.2017, <https://confluence.atlassian.com/jiracoreserver073/jira-core-overview-861255604.html>
- [9] Slack API dokumentácia, Slack, 12.3.2017, <https://api.slack.com>
- [10] StatusHero, 14.3.2017, <https://statushero.com>
- [11] Jell, 14.3.2017, <https://jell.com>
- [12] PlanITpoker, 1.4.2017, <https://planitpoker.com>
- [13] Team O'clock, 1.4.2017, <https://teamoclock.com>
- [14] Github repozitár IBM-Watson, 16.4.2017, <https://github.com/IBM-Watson>
- [15] Github repozitár Botkit, 16.4.2017, <https://github.com/howdyai/botkit>
- [16] Wit.ai, 14.3.2017, <https://wit.ai>

- [17] Martin E., Technický článok v thenextweb.com, 9.12.2016, <https://thenextweb.com/opinion/2016/12/09/3-reasons-2017-year-develop-company-chatbot/>
- [18] Bhatnagar S., Udey online kurz, Build Incredible Chatbots, 2017, <https://www.udemy.com/build-incredible-chatbots/>
- [19] Mardan A., *Practical Node.js: Building Real-World Scalable Web Apps.*, Apress, 2014, ISBN 1430265957
- [20] O node.js, 17.3.2017, <https://nodejs.org/about/>
- [21] Total.js, 17.3.2017, <https://totaljs.com/>
- [22] Ženský avatar komunikačného robota Alica, 9.4.2017, https://www.iconfinder.com/icons/414792/avatar_person_user_woman_icon
- [23] Zmluvné podmienky k používaniu obrázka z platformy iconfinder.com, 9.4.2017, <https://www.iconfinder.com/licenses/basic>

Prílohy

Príloha A

Fotodokumentácia tabúl zo stretnutí sprint retrospective v rámci univerzitného projektu Fri-Skill

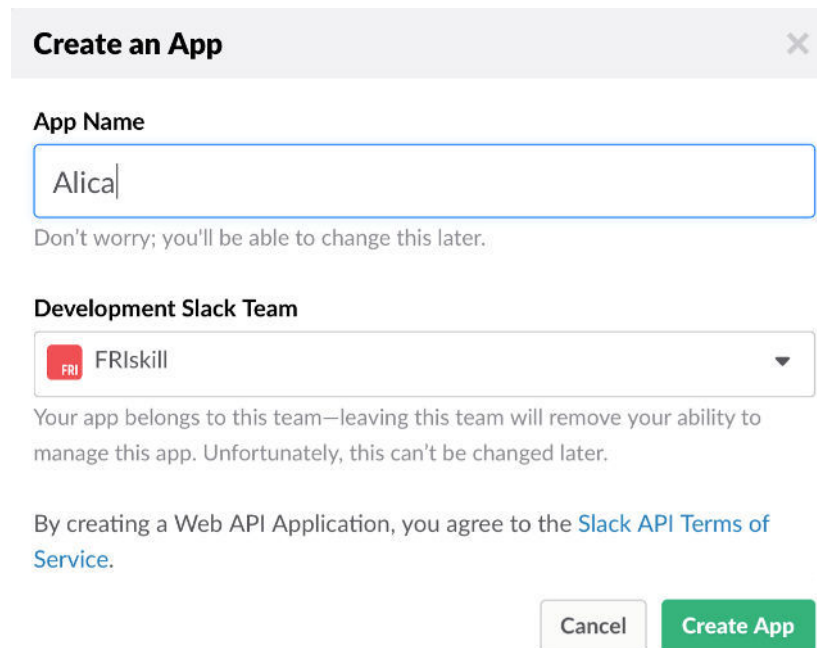


START DOING	CONTINUE DOING	STOP DOING	WHAT BOTHERS ME
MORE TIME FOR PREP. // EARLIER START OF WORK // BETTER TIME ORGANISATION // ACCEPT NIGHT HOURS & DAY TESTING // (AS A NEW TASK) CONSIDER PRIVACY OF SLACK MESSAGES (IF PROBLEM OCCURS)	+ACTIVITIES TEAMBUILDINGS // COMMUNICATION & INTEREST // TEAMWORK, MEETINGS FACE TO FACE // INVOLVING NEW MEMBERS TO THE PROJECT & COMM. DOCUMENTATION (CODE IS MORE HELPFUL THAN CONF. DOCUMENTATION) WORK THROUGH ALL LAYERS	MORE TIME FOR GIVING FAULT TO SOMEBODY THAN TO PROBLEM SOLVING	COMMUNICATION - MSG TASKS ARE NOT ADDED TO JIRA // (MORE SMALLER TASKS)
SOLUTION EARLIER START OF WORK (NOT MONDAY MORNING, BUT TUESDAY) TEAMBUILDINGS COMMUNICATION INTEREST FACE TO FACE MEETINGS SMALLER TASKS			
> MORE TIME ORGANISATION, LESS PROCRASTINATION < > MORE FACE 2 FACE MEETING, LESS BLAMING < > MORE PARTICULAR TASKS, LESS NIGHTWORK <			

Príloha B

Postup vytvorenia Slack aplikácie na platforme `api.slack.com`

1. Na webovej stránke `api.slack.com/apps` pridáme novú aplikáciu a zadáme jej meno a Slack tím.



Create an App ✕

App Name

Alica

Don't worry; you'll be able to change this later.

Development Slack Team

FRI FRIskill ▼

Your app belongs to this team—leaving this team will remove your ability to manage this app. Unfortunately, this can't be changed later.

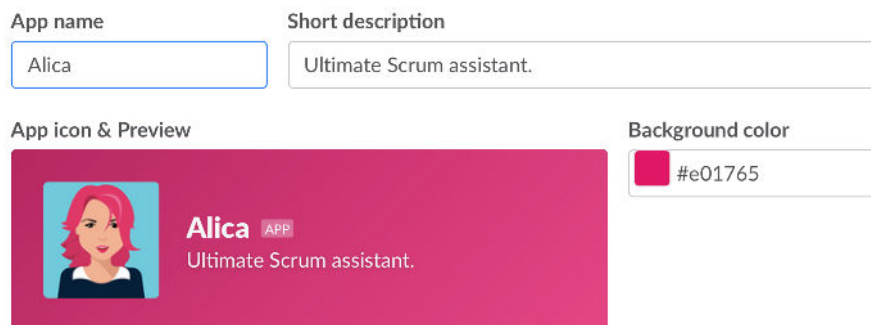
By creating a Web API Application, you agree to the [Slack API Terms of Service](#).

Cancel Create App

2. Po vytvorení doplníme informácie o aplikácii (avatar, farbu, popis).

Display Information

This information will be shown in the Slack App Directory and in the Slack App. For more information, view our [App Detail Guidelines](#).



App name

Alica

Short description

Ultimate Scrum assistant.

App icon & Preview

Background color

#e01765

3. Z funkcionalít aplikácie pridáme integráciu Bot, interaktívne správy a povolíme prijímanie správ cez Slack API.

Add features and functionality



Choose and configure the tools you'll need to create your app (or review all [our documentation](#)).

Incoming Webhooks

Post messages from external sources into Slack.

Interactive Messages

Add buttons to your app's messages, and create an interactive experience for users.

Slash Commands

Allow users to perform app actions by typing commands in Slack.

Event Subscriptions

Make it easy for your app to respond to activity in Slack.

Bots

Add a bot to allow users to exchange messages with your app.

Permissions

Configure permissions to allow your app to interact with the Slack API.

4. Pod interaktívnymi správami pridáme url, na ktorú bude Slack odosielať HTTP POST požiadavku po stlačení tlačidla.

Interactive Messages

You can add buttons to messages from your app, which will send interactions to a URL you specify. [Learn more](#).

Request URL

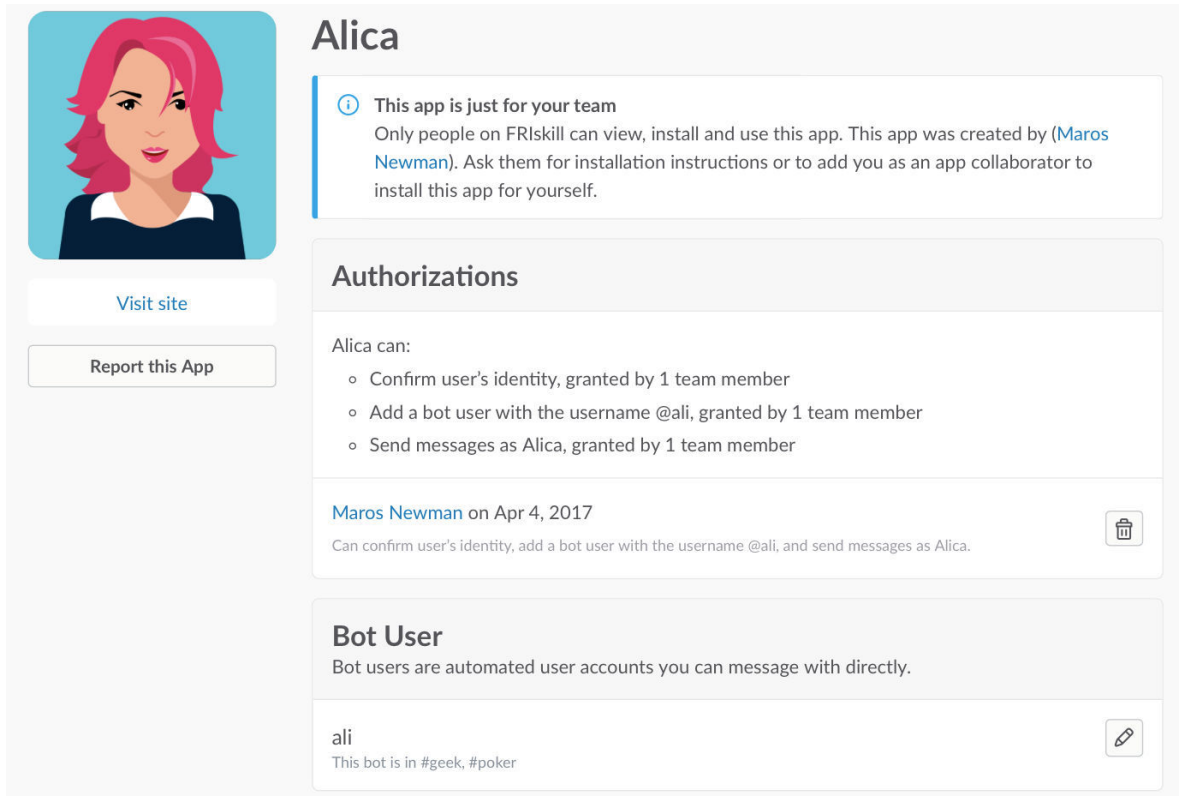
We'll send an HTTP POST request with information to this URL when users invoke message buttons.

Options Load URL

For message menu actions with `"data_source": "external"` we'll send an HTTP POST request with information to load options from this URL when users invoke message menus.

Save changesDisable Interactive Messages

5. Aplikácia je vytvorená a nastavená. Potom sa navigujeme na stránku **friskill.slack.com/apps** a pridáme Alicu ako novú integráciu pre tím. Po tomto kroku sa v tíme **friskill.slack** zobrazí nový používateľ Alica.



The screenshot shows the Slack interface for an app named 'Alica'. On the left is a profile card with a pink-haired avatar, a 'Visit site' button, and a 'Report this App' button. The main content area has a title 'Alica' and an information box stating the app is for the team and was created by Maros Newman. Below this is the 'Authorizations' section, which lists permissions granted by one team member: confirming identity, adding a bot user '@ali', and sending messages as Alica. A log entry shows Maros Newman on April 4, 2017, performing these actions. The 'Bot User' section at the bottom shows the bot's name 'ali' and its presence in the #geek and #poker channels.

Alica

This app is just for your team
Only people on FRIsSkill can view, install and use this app. This app was created by (Maros Newman). Ask them for installation instructions or to add you as an app collaborator to install this app for yourself.

[Visit site](#)

[Report this App](#)

Authorizations

Alica can:

- Confirm user's identity, granted by 1 team member
- Add a bot user with the username @ali, granted by 1 team member
- Send messages as Alica, granted by 1 team member

Maros Newman on Apr 4, 2017
Can confirm user's identity, add a bot user with the username @ali, and send messages as Alica.

Bot User
Bot users are automated user accounts you can message with directly.

ali
This bot is in #geek, #poker

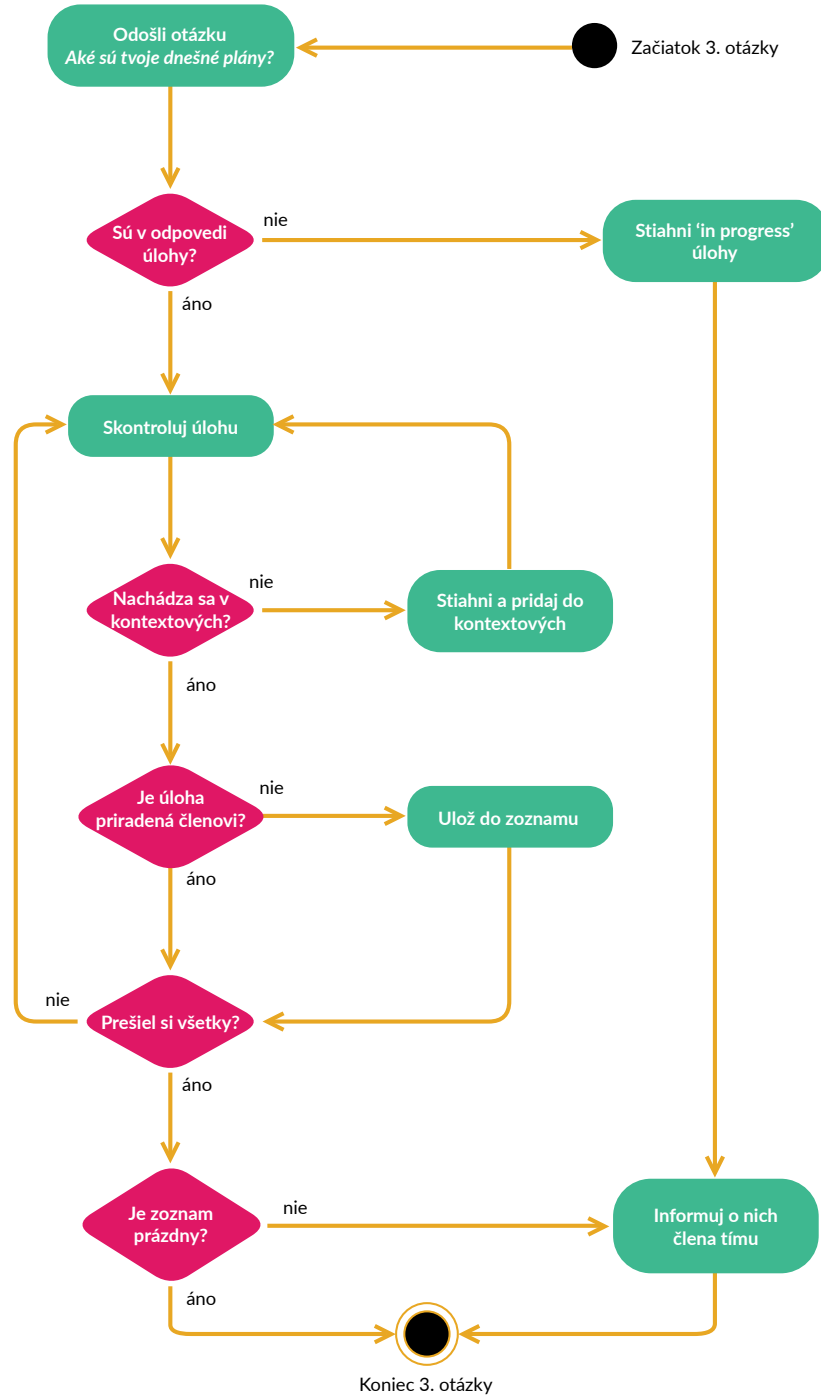
Príloha C

Zoznam zámerov (intent) v službe api.ai

Kategória	Názov zámeru (intent)	Príklad správy	Parametre
General	HELP	How are you working?	-
	MODULES-STATUS	Can you show me modules status?	-
	MODULES-ACTIVATION	Turn off reporting please. Can you enable standup?	module: @Module enabled: @Enabled
	MODULES-ADMIN-RIGHTS	Please add admin rights to <@ID>. Remove rights for <@ID>.	users: @UserID action: @Action
Monitoring	JIRA-ADD-COMMENT	add comment to issues FR-12 'comment'.	issue: @sys.any comment: @sys.any
	JIRA-ISSUES	Show me status of FR-12, FR-33.	issues: @sys.any
	JIRA-MY-ISSUES	Can you show me my tickets?	-
	JIRA-SUMMARY	Please show me all issues.	-
	JIRA-USER-ISSUES	Can you show me issues of <@ID>.	user: @UserID
	JIRA-ASSIGN	Assign issue FR-12 to <@ID>.	issue: @sys.any user: @UserID
Standup	STANDUP-ANSWER	I was working on FR-15 and FR-20.	issues: @sys.any
	STANDUP-CHANNEL	Setup report channel to <#ID>.	channel: @Channel
	STANDUP-TIME	Setup daily meeting for 9:00 please.	time: @sys.time
	STANDUP-USERS	Add <@ID> and <@ID> to standup. Please remove from standup <@ID>	users: @UserID action: @Action
Estimation	START-VOTING	Let's vote about FR-51.	issue: @sys.any
	JIRA-STORY-POINTS	Setup story points 5 to issue FR-25.	issue: @sys.any points: @sys.number

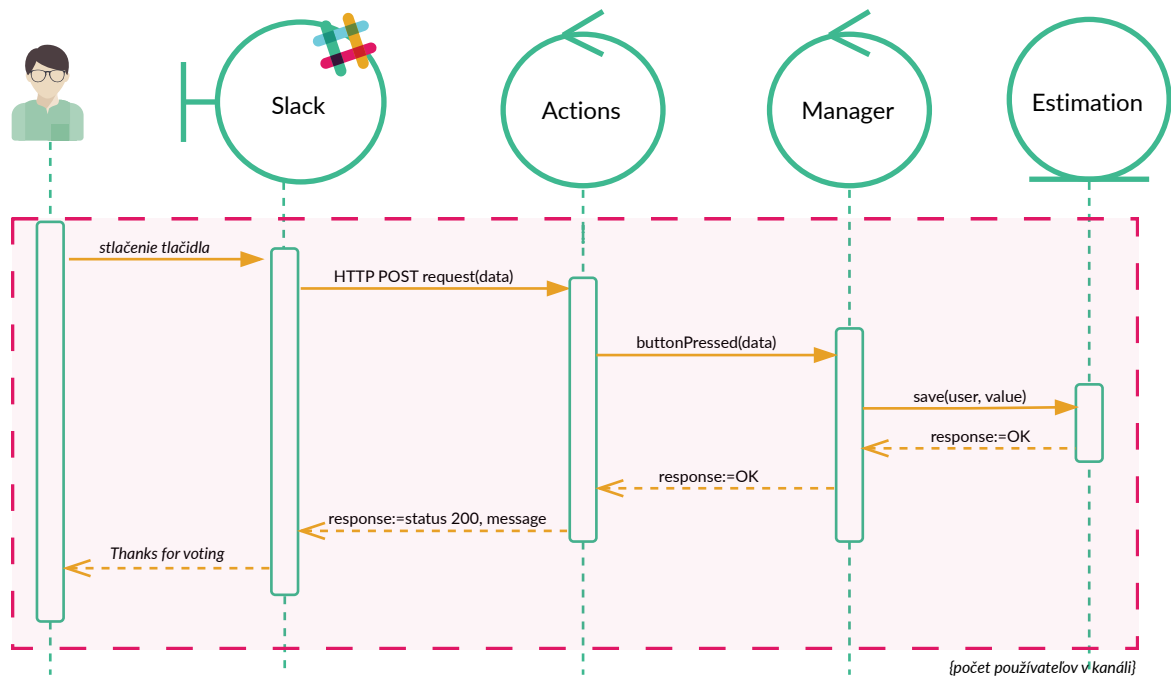
Príloha D

Diagram aktivít priebehu denného stretnutia (3. otázka - Aké sú tvoje dnešné plány?)



Príloha E

Sekvenčný diagram procesu stlačenia tlačidla v interaktívnej správe



Sekvenčný diagram procesu spustenia estimácie používateľského príbehu

