Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny nástroj na viacrozmerné plánovanie agilných projektov

Bakalárska práca

2023
Pavol Repiský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny nástroj na viacrozmerné plánovanie agilných projektov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Aplikovaná Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. František Gyárfáš, PhD.

Bratislava, 2023
Pavol Repiský

37862688

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Pavol Repiský
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Interaktívny nástroj na viacrozmerné plánovanie agilných projektov
Interactive tool for multidimensional planning in agile projects

Anotácia: Cieľom práce bude vytvoriť webový softvérový nástroj, ktorý bude umožňovať hodnotiť náročnosť úloh pri agilných metodológiách (viacrozmerný Scrum poker online). Umožní vytvárať online sedenia pre skupiny riešiteľov s viacrozmernou stupnicou hodnotenia úloh na základe rôznych parametrov (zložitosť, časová náročnosť, užitočnosť, rizikovosť, atď). Účastníci budú môcť zadať svoje hodnotenia tak, aby neboli viditeľné pre ostatných. Odhaliť hodnotenia bude môcť správca sedenia. (Zadanie vzniklo z vnútornej potreby tímov poprednej rakúskej telekomunikačnej spoločnosti. Súčasťou práce bude aj možnosť konzultovať zadanie s pracovníkmi zodpovednými za implementáciu agilných metodológií v rámci danej spoločnosti.) Aplikácia bude vyvíjaná ako webová aplikácia s použitím nástrojov: REST API, React, NodeJS, TypeScript, JavaScript, TDD, Websocket, Git.

Vedúci: Ing. František Gyarfaš, CSc.
Konzultant: Bc. Marek Drnzík
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 30.06.2022

Dátum schválenia: 17.10.2022 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Rád by som sa poďakoval môjmu školiteľovi Ing. Františkovi Gyárfášovi, PhD. za jeho ochotu a priateľský prístup. Obrovské ďakujem patrí aj môjmu konzultantovi Bc. Marekovi Drnzikovi za jeho trpezlivosť, cenné rady a množstvo času, ktoré mi venoval počas tvorby mojej bakalárskej práce. Nakoniec by som sa chcel ešte poďakovať účastníkom testovania použiteľnosti.

Abstrakt

Cieľom bakalárskej práce bolo navrhnúť, implementovať a nasadiť interaktívny webový nástroj na plánovanie agilných projektov, založený na technike nazývanej Plánovacia hra. Tento nástroj bude umožňovať agilným tímom vytvárať online sedenia, na ktorých budú môcť účastníci hodnotiť náročnosť úloh na základe rôznych parametrov. Na rozdiel od ostatných nástrojov bude tento nástroj poskytovať dvojrozmerné stupnice hodnotenia, a teda bude umožňovať viacrozmerné plánovanie agilných projektov. Počas vývoja bol kladený dôraz na splnenie viacerých noriem profesionálneho moderného softvérového inžinierstva. Výsledkom práce je voľne dostupný interaktívny nástroj, ktorý bol nasadený do prevádzky s cieľom pomôcť agilným tímom efektívnejšie plánovať a riadiť projekty.

Kľúčové slová: agile, Plánovacia hra, testami riadený vývoj, webová aplikácia

Abstract

The goal of the bachelor thesis was to design, implement and deploy an interactive web tool for planning agile projects, based on a technique called Planning poker. This tool will allow agile teams to create online sessions where participants can rate the difficulty of tasks based on various parameters. Unlike other tools, this tool will use two-dimensional evaluation matrices, thus enabling multidimensional planning of agile projects. During development, emphasis was placed on meeting several standards of professional modern software engineering. The result of the work is a freely available interactive tool that was deployed, intending to help agile teams plan and manage their projects more effectively.

Keywords: agile, Planning poker, test-driven development, web application

Obsah

Úvod **1**

1 Východiská práce **3**
1.1 Agile . . . 3
1.1.1 Refaktorizácia . . . 5
1.1.2 Testami riadený vývoj . . . 5
1.2 Plánovacia hra . . . 6
1.3 Existujúce riešenia . . . 8
1.3.1 PlanningPoker.com . . . 8
1.3.2 PlanningPokerOnline.com . . . 9
1.3.3 FirePoker.io . . . 9
1.4 Použité technológie . . . 10
1.4.1 Back-End . . . 10
1.4.2 Front-End . . . 11
1.4.3 Back-End a Front-End . . . 11
1.4.4 Testovanie . . . 12

2 Špecifikácia a návrh **13**
2.1 Užívateľské role . . . 13
2.2 Požiadavky . . . 13
2.2.1 Funkčné požiadavky . . . 13
2.2.2 Iné požiadavky . . . 15
2.3 Štruktúra databázy . . . 16
2.3.1 Dátový model . . . 16
2.4 Serverové API . . . 17
2.4.1 Autentifikácia . . . 17
2.4.2 Správa užívateľov . . . 18
2.4.3 Správa hodnotiacich matíc . . . 18
2.4.4 Správa hlasovacích relácií . . . 19
2.5 Aplikácia na strane klienta . . . 19
2.5.1 Užívateľské rozhranie . . . 19

3 Implementácia **25**
3.1 Agile . . . 25
3.2 Testovanie . . . 25
3.3 Správa verzií . . . 26
3.4 Kódové štandardy . . . 27
3.5 Refaktorizácia . . . 28
3.6 Štruktúra klienta . . . 28
3.6.1 Spravovanie formulárov . . . 28
3.6.2 Komunikácia so serverom . . . 28
3.7 Štruktúra servera . . . 29
3.7.1 Smerovanie (Routing) . . . 30
3.7.2 Kontroléri (Controllers) . . . 30
3.7.3 Validácia . . . 31
3.7.4 Modely (Models) . . . 32
3.7.5 Služby (Services) . . . 33
3.8 Autentifikácia . . . 33
3.9 Real-time komunikácia . . . 34
3.10 Lokalizácia a internacionalizácia . . . 34
3.11 Dockerizácia . . . 34

4 Nasadenie **37**

5 Testovanie použiteľnosti **39**

Záver **41**

Úvod

Plánovanie je dôležitou súčasťou projektového riadenia a vývoja. Existuje množstvo rôznych metodík projektového riadenia a vývoja, avšak v posledných rokoch sa do popredia dostávajú predovšetkým agilné metodiky. Na rozdiel od tradičných metodík sú iteratívne, ľahko prispôsobiteľné zmenám a zamerané na intenzívnu spoluprácu so zákazníkom. Jednou z kľúčových výziev v agilných projektoch je odhadnutie potrebného úsilie na dokončenie jednotlivých úloh. Bolo preto vytvorených viacero rôznych techník, za účelom uľahčenia tohto problému, pričom jednou z najpoužívanejších z nich je takzvaná Plánovacia hra (z angl. Planning poker).

Plánovacia hra je široko používaná kolaboratívna technika na odhadnutie náročnosti úloh v agilných projektoch. Členovia vývojového tímu spolu diskutujú a pomocou hracích kariet samostatne hlasujú o zložitosti jednotlivých úloh s cieľom dosiahnutia vzájomnej zhody. Pomocou tejto techniky vieme zabezpečiť zvýšenú presnosť odhadov, väčšie zapojenie tímu do plánovania, zlepšenie tímovej komunikácie a presnejšiu identifikáciu potenciálnych rizík a výziev. Tradične si však vyžaduje osobné stretnutia tímu a vytvorenie fyzických kariet, čo ale nie je vždy možné napríklad v prípade, že členovia tímu pracujú z domu. Za účelom prekonanie týchto obmedzení bolo vyvinutých viacero online nástrojov, ktoré umožňujú tímom viesť takéto stretnutia.

Napriek pomerne vysokej popularite plánovacej hry, dostupnosť online nástrojov je obmedzená, pričom si často vyžadujú zakúpenie predplatného pre ich plnú funkcionalitu. Zároveň tieto nástroje poskytujú len jednorozmerné stupnice hodnotenia s obmedzenými možnosťami hodnôt. Výsledkom čoho môže byť sťažený a menej presný odhad.

Existujú však aj inovatívnejšie, experimentálne prístupy k hodnoteniu úloh. Napríklad použitie dvojrozmernej hodnotiacej stupnice pre viacrozmerné plánovanie agilných projektov na základe viacerých parametrov. Výsledkom použitia týchto matíc by mohli byť presnejšie a lepšie odhady. Hodnotiacou stupnicou by bola matica, pričom jedna os by predstavovala napríklad zložitosť úlohy a druhá zas úroveň jej rizika. Navyše hodnotami tejto stupnice by mohli byť akékoľvek reťazce podľa potrieb tímov.

Cieľom tejto bakalárskej práce je teda navrhnúť, implementovať a nasadiť interaktívny webový nástroj, ktorý bude umožňovať vytvárať online stretnutie na hranie plánovacej hry, vytvárať vlastné dvojrozmerné stupnice hodnotenia a bude voľne do-

stupný. Pričom počas vývoja bude kladený dôraz na splnenie viacerých noriem profesionálneho moderného softvérového inžinierstva, s cieľom zaručenia vyššej kvality výslednej aplikácie.

Kapitola 1

Východiská práce

Táto kapitola je venovaná popisu teórie nevyhnutnej pre realizáciu bakalárskej práce. Taktiež obsahuje prehľad niekoľkých existujúcich riešení a ich zhodnotenie. V neposlednom rade približuje zoznam použitých technológií pri vývoji aplikácie.

1.1 Agile

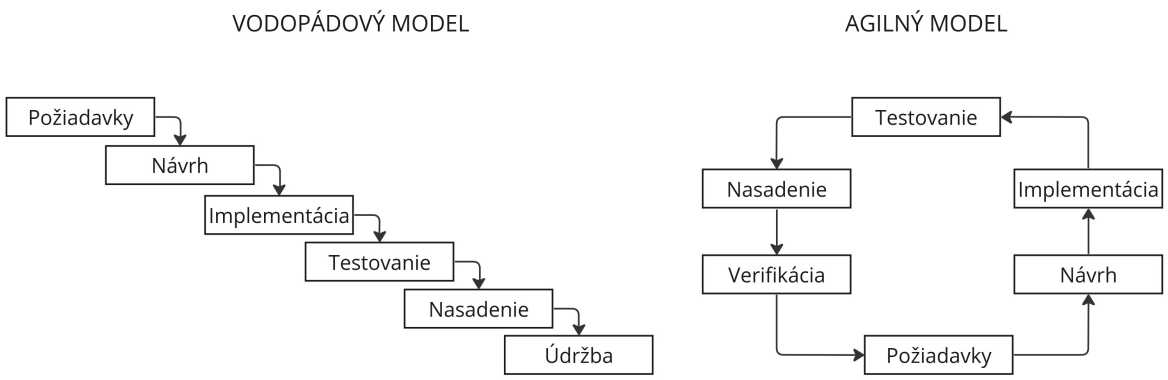
Agile je skupina iteratívnych prístupov k riadeniu a vývoju predovšetkým softvérových projektov. Kladie dôraz na priebežné dodávanie produktov alebo služieb zákazníkom. Zákazník počas celej doby vývoju aktívne spolupracuje s vývojovým tímom, čo dopomáha k rýchlemu odhaleniu a uplatneniu potrebných zmien v rámci projektu, a taktiež k vyššej spokojnosti zákazníka. Je opakom tradičných prístupov k riadeniu a vývoju projektov, akým je napríklad vodopádový prístup (pozri obrázok 1.1). Základom agilných prístupov je Manifest agilného vývoja softvéru, ktorý je založený na nasledovných hodnotách[17].

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

Manifest, taktiež definuje 12 princípov agilného vývoja softvéru, ktorými by sa mali projekty riadiť, aby sa mohli považovať za agilné. Medzi najznámejšie agilné metodiky patrí Scrum, Kanban a Extrémne programovanie (XP)

Predovšetkým v agilnej metodike Scrum sú členovia tímov delení to nasledujúcich rolí.

Product Owner



Obr. 1.1: Diagram porovnania životného cyklu vodopádového a agilného modelu

Product owner je člen tímu, ktorý komunikuje so zákazníkom a zároveň úzko spolupracuje s vývojovým tímom. Má na starosti spravovanie product backlogu, stanovenie priorít jednotlivých úloh a nesie hlavnú zodpovednosť za prácu vývojového tímu a aj za samotný projekt.

Lead/Scrum Master
Jedná sa o člena tímu, ktorého hlavnou úlohou je viesť vývojový tím. Jeho ďalšou úlohou je zabezpečiť, aby všetci členovia tímu mali prístup k potrebným informáciám a presne vedeli, čo majú robiť. Taktiež má na starosti vybavovanie administratívnych úloh a organizovanie stretnutí tímu.

Vývojový tím
Vývojový tím je skupina členov tímu, ktorí sú zodpovední za vytvorenie výsledného produktu. Členovia tohto tímu postupne implementujú požiadavky do projektu. Medzi členmi tohto tímu zvyčajne nájdeme napríklad programátorov, testerov, UX špecialistov alebo produktových dizajnérov.

Pred začatím vývoja projektu je nutné špecifikovať kompletný zoznam úloh, ktoré musia byť pred jeho dokončením splnené. Tento zoznam sa nazýva „**produktový backlog**“ (z angl. „product backlog“). Počas vývoja sa môže dynamicky meniť, môžu byť doň pridané nové úlohy alebo naopak múžu byť z neho nejaké úlohy odobraté, v závislosti od požiadaviek zákazníka alebo nových zistení. Každá úloha v tomto zozname má stanovenú prioritu, pričom platí, čím vyššia je priorita, tým skôr bude daná úloha splnená. Pre jednoduchšiu komunikáciu so zákazníkom môžu byť jednotlivé úlohy vyjadrené pomocou takzvaných „**používateľských príbehov**“ (z angl. „user stories“). Používateľský príbeh je neformálne jednoduché vysvetlenie novej funkcionality, ktorú je potrebné do softvéru pridať, napísané z pohľadu koncového používateľa. Zvyčajne

má nasledujúci formát.

„Ako [používateľ] chcem [funkciu], aby [výsledok].“

Agilné projekty sú rozdelené na krátke, opakovateľné fázy, zvyčajne dlhé jeden až štyri týždne, nazývané „**iterácie**“ alebo „**šprinty**“ (z angl. „sprints“). Počas tohto obdobia pracuje vývojový tím na pevne určenom zozname úloh z produktového backlogu nazývanom „**šprint backlog**“. Tento zoznam je vytvorený na „**plánovacom stretnutí**“ (z angl. „planning meeting“), ktoré sa koná pred každým šprintom. Počas tohto stretnutia sa okrem tvorby tohto zoznamu, určuje cieľ šprintu a odhaduje sa náročnosť práce. Výsledkom každého šprintu by mala byť plne funkčná verzia softvéru s novými funkciami, ktorá môže byť potenciálne odovzdaná zákazníkovi. Táto nová verzia softvéru je predstavená zákazníkovi na stretnutí nazývanom „**šprint review**“, ktoré sa koná po skončení šprintu (Spracované podľa [21] a [22]).

1.1.1 Refaktorizácia

Refaktorizáciou sa nazýva proces zmeny vnútornej štruktúry existujúceho kódu, bez dopadu na jeho vonkajšie správanie. Mala by sa vykonávať formou opakovaných malých úprav v kóde, pričom prípadné testy pôvodného kódu poskytujú istotu, že výsledný kód zostáva správny. Jej výsledkom by malo byť zlepšenie a zjednodušenie vnútornej štruktúry kódu, kód by mal byť čitateľnejší, zrozumiteľnejší ľahšie udržateľný a rozšírovateľný [20]. Kód by mal byť zbavený problémových častí, akými sú napríklad dlhé metódy a funkcie, duplicity alebo metódy a funkcie s priveľa parametrami.

1.1.2 Testami riadený vývoj

Testami riadený vývoj (z angl. test driven development skrátene tdd) je jedná z agilných metodík, ktorá sa bežne využíva v praxi pri vývoji softvéru. Jej základom je napísanie testu pre novú funkcionalitu, pred tým, než sa začne implementovať. Tento test musí prvotne zlyhať. Týmto postupom vieme zabezpečiť, že výsledný kód a aj všetky jeho ďalšie verzie budú neustále kontrolované a každá chyba pokrytá testami bude po spustení testov okamžite odhalená. Samotný cyklus testami riadeného vývoja je nasledovný.

1. Začne sa pridaním nového testu pre požadovanú funkcionalitu. Tento test musí určite zlyhať, keďže implementácia tejto funkcionality zatiaľ chýba.

2. Nasleduje spustenie všetkých testov a skontrolovanie, že nový test naozaj zlyhá.

3. Teraz nasleduje samotného implementácia, pričom sa kladie dôraz na napísanie len toľko kódu, aby test prešiel.

4. Opäť sa spustia všetky testy a skontroluje sa, že tentokrát všetky testy úspešne prešli.

5. Nakoniec ešte môže nasledovať refaktorizácia kódu, ak je to potrebné.

6. Cyklus sa potom opakuje s novým testom.

Existuje viacero typov automatizovaných testov, najčastejšie sa však stretneme s týmito tromi typmi.

1. Jednotkové testy - Testujú funkčnosť určitej jednotky (z angl. unit) programu. Takouto jednotkou môže byť napríklad funkcia, trieda alebo komponent, záleží od konkrétneho prípadu. Sú vykonávané v izolácii od zvyšku softvéru a prípadné závislosti sú simulované.

2. Integračné testy - Testujú správne spoločné fungovanie skupiny jednotiek softvéru. Sú komplikovanejšie, ako jednotkové testy a ich priebeh trvá dlhšie.

3. E2E (end-to-end) - Testujú funkcionalitu celej aplikácie. Používajú skutočné používateľské rozhranie, skutočné databázy a iné služby. Trvajú najdlhšie a pokrývajú najviac kódu.

(Spracované podľa [18])

1.2 Plánovacia hra

Každému šprintu v agilných projektoch predchádza jeho plánovanie. Dôležitou súčasťou tohto plánovania je odhadovanie náročnosti a potrebného úsilia na dokončenie jednotlivých úloh v šprint backlogu. Tieto odhady môžu byť veľmi užitočné a nápomocné pri delení projektu na krátkodobé, zvládnuteľné úlohy. Avšak správne odhadnúť náročnosť úlohy nie je jednoduché a nesprávny odhad môže viesť k viacerým závažným problémom a komplikáciám. Bolo preto vytvorených viacero techník, ktoré sa snažia proces odhadovania uľahčiť. Medzi najznámejšie z nich patria techniky, ako: „**Burndown chart**“, „**Velocity**“ alebo „**Plánovacia hra**“. Moja práca sa zameriava práve na poslednú z menovaných techník.

Plánovacia hra je agilná technika plánovania a odhadovania založená na konsenze účastníkov. Používa sa na hodnotenie úloh v nadchádzajúcom šprint backlogu, odhaduje, koľko času a úsilia je potrebné na dokončenie každej z nich. Vyžaduje si použitie určitej hodnotiacej stupnice, tradične je táto stupnica reprezentovaná pomocou fyzických kariet, odtiaľ pochádza slovo poker z anglického názvu. Pomocou týchto kariet priraďujú hráči úlohám hodnoty, nazývané „**príbehové body**“ (z angl. „story points“). Tieto body určujú zložitosť a potrebné úsilie na dokončenie danej úlohy. Hodnotiaca

stupnica tradične predstavuje nejakú postupnosť. Medzi najpoužívanejšie postupnosti patria: Fibonacciho postupnosť, postupnosť veľkostí tričiek (xs, s, m, l, xl, ...), mocniny dvojky alebo lineárne postupnosti. Ale samotný výber stupnice hodnotenia závisí od viacerých faktorov, ako napríklad preferencie tímu, vlastností projektu a typy úloh. Pričom ich hlavným účelom by malo byť vyhnutie sa nadmernému analyzovaniu odhadu. Samotný priebeh hry je nasledovný.

1. Tím si naplánuje stretnutie a zvolí sa moderátor, ktorý bude hru viesť.

2. Hra sa začína prečítaním popisu prvej úlohy z šprint backlogu.

3. Následne účastníci môžu klásť otázky k danej úlohe s cieľom, aby všetci správne pochopili podstate danej úlohy a vyhli sa tak prípadným nedorozumeniam.

4. V ďalšom kroku si každý člen tímu vyberie kartu, ktorá predstavuje jeho odhad náročnosti danej úlohy a položí ju lícom nadol na stôl bez toho, aby odhalil svoj odhad.

5. Nasleduje otočenie všetkých vybraných kariet.

6. V prípade, že sa všetci členovia zhodli na rovnakej karte, odhad sa zapíše a pokračuje sa na ďalšiu úlohu.

7. V opačnom prípade prebehne diskusia, počas ktorej hráči vysvetlia, ako sa k danému odhadu dostali a proces sa opakuje.

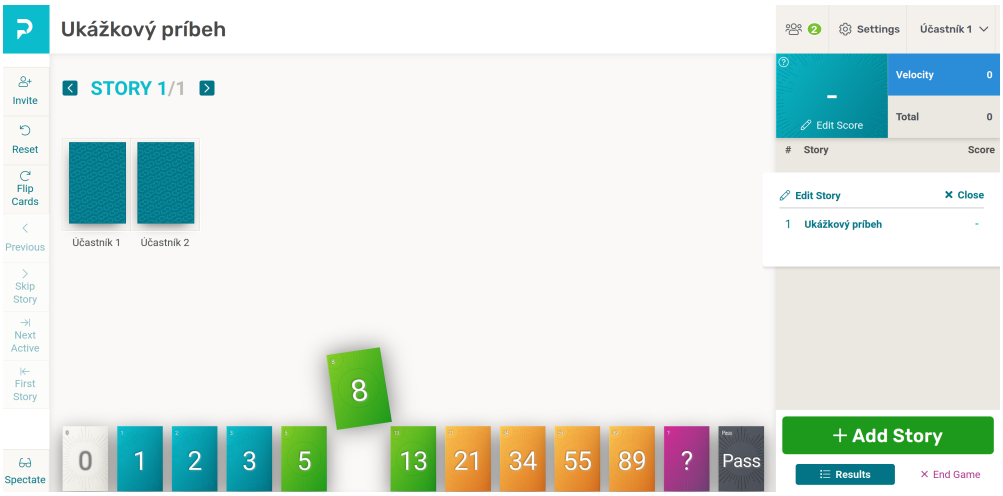
V plánovacej hre sa bežne používajú jednorozmerné hodnotiace stupnice, ale v súčasnosti sa začína v niektorých agilných tímoch presadzovať inovatívna, viacdimenzionálna verzia tejto hry. Táto pokročilejšia verzia umožňuje hodnotiť úlohy na základe viacerých parametrov, pomocou viacrozmerných stupníc hodnotenia. Napríklad úlohy sa hodnotia nielen podľa náročnosti, ale aj podľa riskantnosti a prínosu. Navyše sa môžu využiť aj obrazné hodnoty, ako napríklad zvieratá, aby sa úlohy hodnotili a porovnali vizuálne. Napríklad úloha môže byť ohodnotená, ako úloha s nízkym rizikom, ale náročnou implementáciou, čo by sa dalo znázorniť pomocou symbolu slona. Slon je veľký, ale nie je príliš nebezpečný. Naopak, úloha s vysokým rizikom a jednoduchou implementáciou by mohla byť symbolizovaná škorpiónom, ktorý je malý, ale nebezpečný. V rámci rakúskej telekomunikačnej spoločnosti A1 sa niektoré agilné tímy začali zaoberať experimentovaním s touto verziou plánovacej hry. V tejto spoločnosti vzniklo aj samotné zadanie pre bakalársku prácu, ktorej hlavným cieľom je vytvorenie nástroja, ktorý by ponúkal takúto funkcionalitu. Tieto informácie boli poskytnuté mojím konzultantom, ktorý v týchto tímoch pracoval, ako programátor.

1.3 Existujúce riešenia

Zvyšujúca sa popularita plánovacej hry a zvyšujúci sa dopyt po zamestnaniach s možnosťou práce z domu, podnietili vytvorenie viacerých webových nástrojov na jej realizáciu. Vybrané tri z nich sú predstavené v tejto sekcii.

1.3.1 PlanningPoker.com

Jedným z najpoužívanejších nástrojov pre realizáciu online plánovacej hry je nástroj Planningpoker.com[7]. Veria mu známe spoločnosti, akými sú napríklad: Amazon, Tesla, Adobe, IBM alebo Coca-Cola. Poskytuje mnoho rôznych nastavení, aby vyhovel potrebám rôznych tímov. Príkladmi sú možnosť nastaviť, či moderátor hry taktiež hlasuje, automatické otáčanie kariet alebo časovač pre hlasovanie. Taktiež poskytuje extra



Obr. 1.2: Ukážka priebehu plánovacej hry v nástroji PlaningPoker.com

funkcionality, akými sú napríklad možnosť importovania úloh zo CSV súborov, JIRA integrácia alebo automatický výpočet priemerného skóre. Užívateľské rozhranie je intuitívne a prehľadné, avšak v porovnaní s inými nástrojmi je pomerne nemoderné. Umožňuje vytvárať vlastné jednorozmerné stupnice hodnotenia, pričom hodnoty jednotlivých kariet môžu obsahovať iba číselné hodnoty, ku ktorým môže byť priradený názov maximálnej dĺžky 3 znaky.

Jeho najväčšou nevýhodou je nutnosť jeho zakúpenia pre plnú funkcionalitu po uplynutí skúšobnej doby a limitovaný počet účastníkov v hre v závislosti od zakúpeného plánu. Moderátor hry sa musí zaregistrovať, ostatní účastníci musia pri pripojení zadať svoje meno a e-mailovú adresu.

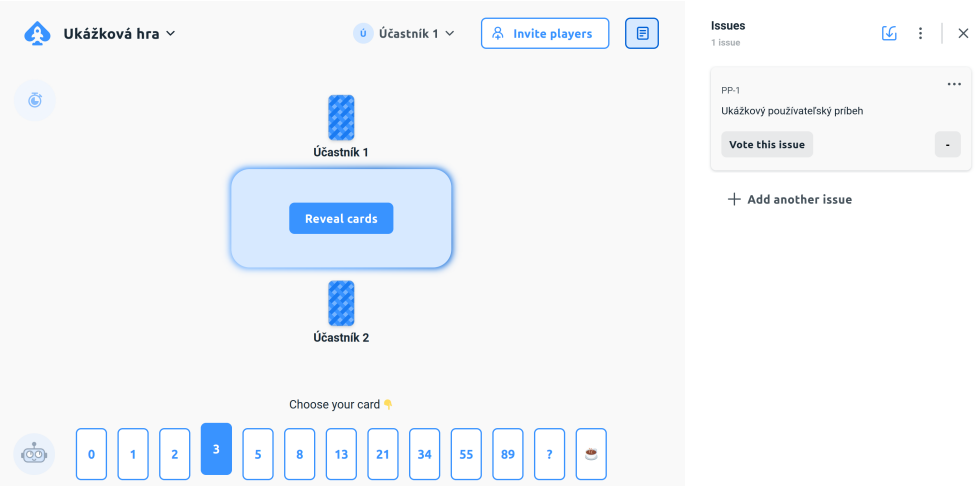
Planningpoker.com je ideálnym nástrojom pre agilné tímy, ktoré sú ochotné si za nástroj priplatiť, hľadajú spoľahlivé a overené riešenie s vysokou mierou prispôsobenia potrebám tímu.

1.3.2 PlanningPokerOnline.com

Ďalším populárnym nástrojom je PlanningPokerOnline.com. Vybrali si ho spoločnosti, ako napríklad Google, Microsoft a Nokia[8]. Jeho najväčšou prednosťou je jeho design. Užívateľské prostredie je veľmi jednoduché a intuitívne, pripomínajúce rozhranie skutočných kartových hier. Nástroj si nevyžaduje registráciu na jeho používanie. Rovnako, ako predchádzajúci nástroj, poskytuje JIRA integráciu a možnosť importovania úloh zo CSV súborov. Umožňuje vytvárať jednorozmerné hodnotiace stupnice s kartami, ktorých hodnotami môžu byť iba reťazce maximálnej dĺžky 3 znaky.

Jeho nevýhodou je nutnosť zakúpenia predplatného pre kompletnú funkcionalitu. Bezplatná verzia poskytuje obmedzený počet účastníkov a úloh v jednej hre. Oproti predchádzajúcemu nástroju poskytuje nižšiu úroveň prispôsobenia.

PlanningPokerOnline.com je správnou voľbou pre agilné tímy, ktoré hľadajú nástroj s hravým designom, vysokou spoľahlivosťou a nevyžadujú si extra funkcionalitu.



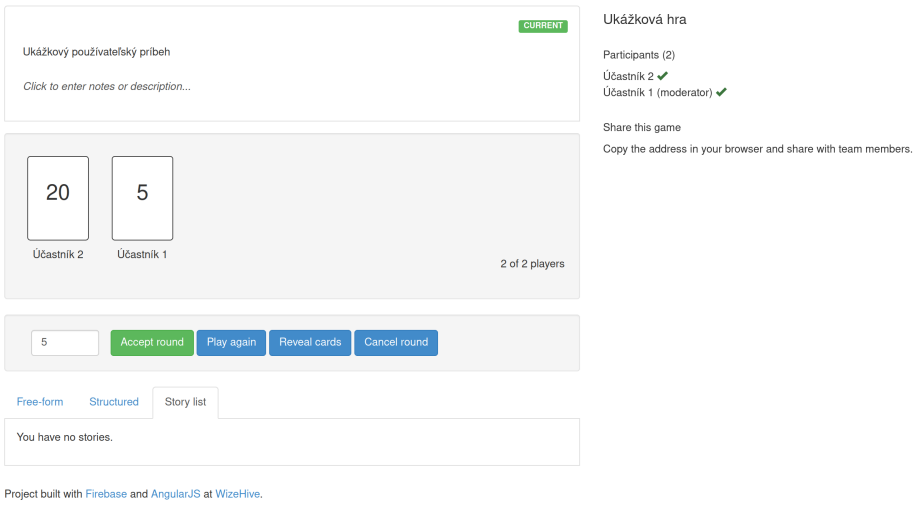
Obr. 1.3: Ukážka priebehu plánovacej hry v nástroji PlaningPokerOnline.com

1.3.3 FirePoker.io

Posledným nástrojom je FirePoker.io. Od väčšiny nástrojov sa odlišuje tým, že je kompletne bezplatný a open-source[9], to sa však odzrkadľuje na jeho kvalite. Jeho užívateľské rozhranie je veľmi jednoduché, ale zároveň nemoderné. Celý priebeh hry od jej vytvorenia až po je koniec si vyžaduje iba pár kliknutí a je veľmi jednoduchý aj pre používateľa, ktorý tento nástroj používa prvý krát. Nástroj si nevyžaduje ani neposkytuje možnosť registrácie. Neumožňuje vytvárať vlastné stupnice hodnotenia, používateľ má na výber z piatich preddefinovaných štandardných stupníc hodnotenia. Poskytuje iba nevyhnutnú funkcionalitu ani nič viac.

Tento nástroj bude vyhovovať tímom, ktoré nechcú strácať čas ani peniaze. Chcú sa vyhnúť dlhému študovaniu, ako daný nástroj používať a rovno sa pustiť do hry.

Preferujú jednoduchý design a štandardnú funkcionalitu bez zbytočných rozptýlení.



Obr. 1.4: Ukážka priebehu plánovacej hry v nástroji FirePoker.io

1.4 Použité technológie

V tejto sekcii sú opísané najdôležitejšie technológie použité pri vývoji aplikácie.

1.4.1 Back-End

Node.js

Node.js je prostredie pre spúšťanie JavaScript kódu mimo internetového prehliadača. Používa udalosťami riadenú, I/O neblokujúceho architektúru, vďaka tomu dokáže spracovať veľké množstvo požiadaviek. Jeho primárnym účelom je tvorba webových serverov[1].

Express.js

Express je rozšírenie Node.js, ktoré poskytuje veľké množstvo funkcií na uľahčenie a urýchlenie procesu budovania webových a mobilných aplikácií[2].

Prisma

Prisma je open-source databázový nástroj pre Node.js a TypeScript. Skladá sa z nasledujúcich troch častí[15].

- Prisma Client - automaticky vygenerovaný systém na tvorbu databázových dopytov s typovou kontrolou
- Prisma Migrate - migračný systém

- Prisma Studio - grafický nástroj, určený na prezeranie a úpravu údajov v databáze

1.4.2 Front-End

React.js

React.js je JavaScriptová knižnica určená na vytváranie používateľských rozhraní. Je základom sú komponenty, ktoré reprezentujú logické opakovane použiteľné časti užívateľského rozhrania. Možno si ich zjednodušene predstaviť, ako nezávislé Lego bloky. Spojením týchto komponentov do jedného celku dostávame celé používateľské rozhranie aplikácie. React využíva deklaratívny prístup k budovaniu používateľských rozhraní: to znamená, že na základe napísaného kódu a dát sa postará o všetky potrebné manipulácie DOM-u. Často si však vyžaduje použitie dodatočných knižníc, ako napríklad React Router na smerovanie požiadaviek alebo Formik na jednoduchšie vytváranie a spracovávanie formulárov[10].

Material UI

Material UI je React knižnica, poskytujúca kolekciu vopred zostavených komponentov, ktoré sú pripravené na použitie a ľahko prispôsobiteľné potrebám vývojárov[6].

JWT

JSON Web Token (JWT) je štandard pre vytváranie tokenov s údajmi vo formáte JSON, ktoré sa používajú na autentifikáciu a autorizáciu v aplikáciách. V Node.js sa pre prácu s JWT používa knižnica jsonwebtoken[4].

1.4.3 Back-End a Front-End

TypeScript

TypeScript je nadstavba JavaScriptu, ktorá obsahuje všetku funkcionalitu JavaScriptu a pridáva navyše svoju vlastnú, predovšetkým statické typovanie. Statické typovanie zabezpečuje, že typ premennej nemožno zmeniť v žiadnom bode programu. Vďaka tomu vieme zabrániť množstvu nechcených chýb. Všetky JavaScriptové programy sú teda spustiteľné aj v TypeScripte a všetok TypeScriptový kód sa vo výsledku kompiluje naspäť do JavaScriptu[12].

Socket.io

Socket.io je JavaScriptová knižnica, ktorá umožňuje obojsmernú komunikáciu založenú na udalostiach medzi klientom a serverom pomocou web-socketov. Využíva sa predovšetkým v aplikáciách, ktoré si vyžadujú real-time komunikáciu[3].

Docker

Docker je virtualizačný nástroj, ktorý umožňuje vytvárať, nasadzovať a spúšťať aplikácie v kontajneroch[16]. Kontajner je softvérový balíček, ktorý obsahuje iba kód aplikácie a nevyhnutné závislosti na jej spustenie. Tieto kontajnery umožňujú spúšťať aplikáciu, kdekoľvek bez ohľadu na dané prostredie a izolovať ju od systému a iných aplikácii[13].

Rest API

REST je softvérová architektúra, ktorá určuje, ako má API fungovať. Obsahuje koncové body, ktoré sú dostupné na určitých URL adresách. Pomocou týchto koncových bodov môže klientska aplikácia s daným API komunikovať cez HTPP protokol. Telo požiadavky je zvyčajne písané komunikačným formátom JSON[14].

1.4.4 Testovanie

Jest

Jest je JavaScriptová knižnica na vytváranie, spúšťanie a štruktúrovanie testov. Jest využíva na písanie testov takzvané matchers, ktoré porovnávajú či sa výstup kódu zhoduje s očakávaným výstupom[5].

Kapitola 2

Špecifikácia a návrh

Táto kapitola sa venuje špecifikácii a návrhu aplikácie. Konkrétne opisuje užívateľské role, špecifikáciu požiadaviek zadávateľa projektu, návrh užívateľského rozhrania, dátový model a návrh funkcionality.

2.1 Užívateľské role

Nástroj rozlišuje nasledujúce dva typy používateľov.

Neprihlásený používateľ (hosť)
Tento typ používateľa ma prístupnú iba obmedzenú časť aplikácie. Aplikácia mu umožňuje registrovať si účet, prihlásiť sa do existujúceho účtu a resetovať si heslo. Okrem toho má ešte možnosť pripojiť sa do už existujúcej hlasovacej relácie, v ktorej môže hlasovať.

Prihlásený používateľ
Prihlásený používateľ ma prístupnú kompletnú funkcionalitu aplikácie, okrem registrácie, prihlásenia sa a resetovania hesla. Môže si upraviť svoje osobné údaje, zmeniť heslo, manažovať a vytvárať stupnice hodnotenia, vytvárať hlasovacie relácie, vytvárať hlasovania a hlasovať.

2.2 Požiadavky

Nasledujúce požiadavky na funkcionalitu aplikácie boli určené zadávateľom projektu.

2.2.1 Funkčné požiadavky

Požiadavky pre správu užívateľa

1. Registrácia - Používateľ má možnosť zaregistrovať si nový používateľský účet. Po otvorení registračnej stránky a vyplnení formuláru bude používateľovi odoslaný e-mail s potvrdzovacím odkazom. Používateľ potvrdí vytvorenie svojho účtu kliknutím na tento odkaz. Po potvrdení sa účet aktivuje a používateľ sa môže prihlásiť.

2. Prihlásenie - Používateľ má možnosť prihlásiť sa do svojho účtu. Po otvorení prihlasovacej stránky a vyplnení formuláru môžu nastať dve situácie. V prípade, že zadané údaje sú správne a účet používateľa je aktivovaný, bude používateľ autentifikovaný a presmerovaný na jeho domovskú stránku. V opačnom prípade bude upozornený o nesprávnych údajoch alebo neaktívnom účte.

3. Zabudnutie hesla - Ak Používateľ zabudne heslo, má možnosť kliknúť na odkaz pre zabudnuté heslo na prihlasovacej stránke. Po otvorení stránky a vyplnení správneho e-mailu bude používateľovi odoslaný e-mail s odkazom na obnovenie hesla. Po kliknutí na tento odkaz bude používateľ presmerovaný na stránku pre obnovenie hesla. Po správnom vyplnení nového a potvrdzovacieho hesla bude heslo obnovené.

4. Odhlásenie - Po kliknutí na tlačidlo odhlásenia bude prihlásený používateľ odhlásený z aplikácie a presmerovaný na domovskú stránku.

5. Zmena hesla - Po otvorení stránky pre zmenu hesla a správnom vyplnení aktuálneho a nového hesla, bude heslo prihláseného používateľa zmenené. V opačnom prípade bude používateľ upozornený na chybu.

Požiadavky pre správu hodnotiacich matíc

6. Zobrazenie matíc - Na domovskej stránke hodnotiacich matíc sa používateľovi zobrazuje zoznam všetkých ním vytvorených matíc. Každá položka v tomto zozname bude obsahovať názov matice, jej veľkosť a dátum vytvorenia. Používateľ vie zoradiť matice abecedne podľa názvu alebo chronologicky podľa dátumu vytvorenia. Každý záznam bude navyše obsahovať tlačidlo na zobrazenie detailu danej matice a na jej vymazanie.

7. Vytvorenie matice - Používateľ môže vytvoriť novú maticu hodnotenia vyplnením formuláru, kde zadefinuje jej názov, veľkosť a samotné hodnoty.

8. Zobrazenie matice - Používateľ si môže detailne prezrieť jednu hodnotiacu maticu, pričom sa mu zobrazí jej grafická reprezentácia.

9. Upravenie matice - Používateľ môže upraviť existujúcu hodnotiacu maticu, vie aktualizovať jej názov, veľkosť a hodnoty.

10. Vymazanie matice - Používateľ môže jednotlivo odstrániť hodnotiace matice.

Požiadavky pre správu hlasovacích relácií

11. Vytvorenie relácie - Používateľ môže vytvoriť novú hlasovaciu reláciu vybraním si hodnotiacej matice, ktorá sa bude v relácii používať a zadefinovaním jej názvu. Tvorca relácia bude zároveň jej administrátorom.

12. Pripojenie sa do relácie - Každý používateľ sa môže pripojiť k relácii, buď ako hosť alebo prihlásený používateľ. Na pripojenie potrebuje používateľ poznať jej jedinečnú URL adresu.

13. Vytvorenie hlasovania - Každá hlasovacia relácia pozostáva zo série nezávislých hlasovaní. Vytvorením nového hlasovania je staré hlasovanie automaticky zatvorené. Používateľ môže vytvoriť nové hlasovanie kedykoľvek, bez ohľadu na stav aktuálneho hlasovania. Keď je hlasovanie vytvorené, účastníkom sa zobrazia všetky hlasovacie možnosti, hlasy ostatných účastníkov sú skryté.

14. Hlasovanie v relácii - Každý účastník môže hlasovať v aktuálnom hlasovaní kliknutím na jednu z hlasovacích možností. Svoj hlas môže meniť, kým administrátor nezobrazí výsledky hlasovania.

15. Zobrazenie výsledkov hlasovania - Administrátor relácie môže kedykoľvek zobraziť výsledky hlasovania, bez ohľadu na to, či všetci účastníci už zahlasovali. Systém vráti sumár počtu hlasov pre každú zahlasovanú možnosť zoradený podľa najvyššieho počtu hlasov. Hlasy všetkých účastníkov budú zobrazené.

2.2.2 Iné požiadavky

16. Aplikácia bude vyvíjaná, ako webová aplikácia s využitím REST API.

17. Back-End bude vyvíjaný využitím prostredia Node.js a jazyku TypeScript.

18. Front-End bude vyvíjaný pomocou knižnice React a jazyku TypeScript.

19. Celá aplikácia bude dockerizovaná, teda Back-End aj Front-End budú bežať v samostatných docker kontajneroch.

20. Aplikácia bude vyvíjaná pomocou testami riadeného vývoja a Git-u.

21. Na real-time komunikáciu bude využívať Websockety.

22. Bude umožňovať vytvárať dvojrozmerné hodnotiace matice, teda matice, ktoré sa môžu skladať z viacerých riadkov hodnôt.

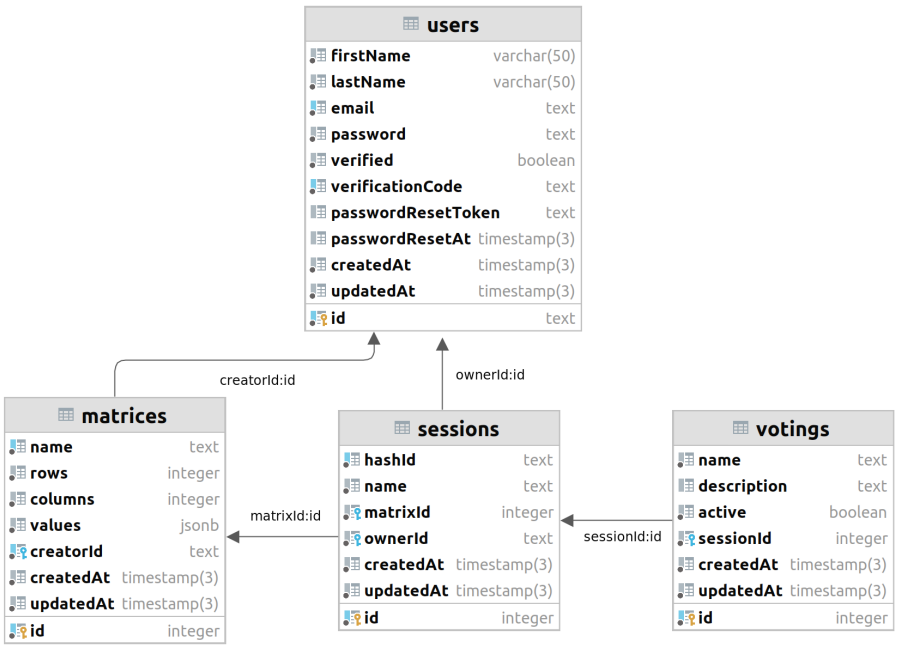
2.3 Štruktúra databázy

Aplikácia využíva na ukladanie dát PostgreSQL databázu a používajú sa v nej nasledujúce štyri entity:

- Používateľ - Reprezentuje registrovaného používateľa.
- Matica - Predstavuje hodnotiacu stupnicu.
- Relácia - Reprezentuje hlasovaciu reláciu.
- Hlasovanie - Predstavuje hlasovanie v rámci nejakej hlasovacej relácie.

2.3.1 Dátový model

Diagram 2.1 zobrazuje návrh reprezentácie spomínaných entít, ako databázových tabuliek a vzťahov medzi nimi. Všetky tieto tabuľky obsahujú nasledujúce automaticky vy-



Obr. 2.1: Diagram návrhu dátového modelu

generované atribúty. Identifikátor (id) je primárnym kľúčom a jednoznačne identifikuje záznam v danej tabuľke. Dátum vytvorenie záznamu a dátum posledného upravenia záznamu.

Users

Tabuľka Users reprezentuje entity používateľa. Primárne bude uchovávať jeho osobné

údaje: krstné meno a priezvisko, prihlasovacie údaje: email a heslo. Navyše bude obsahovať ďalšie pomocné atribúty, ktoré sa budú využívať napríklad pri resetovaní hesla alebo overení e-mailu. Je najdôležitejšou tabuľkou celej aplikácie, od ktorej sú závislé všetky ostatné tabuľky.

Matrices
Tabuľka Matrices reprezentuje entity matice. Bude slúžiť na ukladanie údajov o jednotlivých hodnotiacich matíc, akými sú: názov, veľkosť formou počtu riadkov a stĺpcov a samotné hodnoty. Hodnoty budú implementované, ako dvojrozmerný zoznam, pričom do databázy budú ukladané skonvertované na JSON reťazec. Ďalším dôležitým atribútom je identifikátor používateľa, ktorý označuje tvorcu danej matice.

Sessions
Tabuľka Sessions reprezentuje entity relácií. Jej úlohou je zaznamenávanie vytvorených hlasovacích relácii, bude teda uchovávať informácie, ako: názov relácie, jedinečný reťazec, ktorý bude použitý pri zostrojení unikátnej URL adresy na pripojenie sa do danej relácie. Ďalej obsahuje identifikátor, teda cudzí kľúč používateľa, ktorý ju vytvoril a identifikátor matice, ktorá bude použitá v hre.

Votings
Posledná tabuľka Votings predstavuje entity hlasovania. Slúži na ukladanie informácii, akými sú: názov hlasovania, popis hlasovania, informáciu o tom, či je hlasovanie aktívne a identifikátor, teda cudzí kľúč hlasovacej relácie, do ktorej patrí.

2.4 Serverové API

Aplikácia je postavená na architektúre server/klient a REST API. Server poskytuje klientovi rôzne zdroje prostredníctvom REST-ovského API. Koncové body servera (endpoints) definujú funkcionality, ktoré môže klient využívať pomocou HTTP požiadaviek. Stateless autentifikácia užívateľa na strane servera zabezpečuje bezpečnú komunikáciu medzi serverom a klientom. Na komunikáciu medzi serverom a klientom sa využíva komunikačný formát JSON. Zoznam návrhu koncových bodov, ktoré server sprístupňuje pre klienta, spolu s ich korešpondujúcimi HTTP metódami je uvedený nižšie.

2.4.1 Autentifikácia

Server poskytuje nasledujúce koncové body zamerané na autentifikáciu.

- POST /register - Umožňuje klientovi registrovať nové používateľské účty. V tele požiadavky sa musia nachádzať nasledujúce atribúty: krstné meno, priezvisko,

email, heslo a potvrdzovacie heslo.

- POST /login - Umožňuje klientovi autentifikovať sa na serveri. V prípade úspešnej autentifikácii server vygeneruje a vráti autentifikačný token a nastaví potrebné cookies. V tele požiadavky sa musí nachádzať email a heslo.
- GET /refresh - V prípade správnej hodnoty obnovovacieho tokenu v HTTPOnly cookie, obnoví a vráti autentifikačný token tak, aby nebol používateľ odhlásený z aplikácie.
- GET /lougout - Vyžaduje si autentifikáciu. Slúži na odhlásenie používateľa zaslaním cookies so skončenou platnosťou, tým pádom budú okamžite odstránené z prehliadača.
- GET /verify-email/:verificationCode - V prípade správneho verifikačného kódu v parametroch požiadavky aktivuje účet prislúchajúceho používateľa.
- POST /forgot-password - Odošle na zadaný e-mail v tele požiadavky odkaz na resetovanie hesla.
- PATCH /reset-password/:resetToken - V prípade správneho a resetovacieho tokenu v parametroch požiadavky a korektného nového a potvrdzovacieho hesla v tele požiadavky, nastaví heslo používateľa na novú hodnotu.

2.4.2 Správa užívateľov

Server poskytuje nasledujúce koncové body pre správu užívateľov.

- GET /users - Vyžaduje si autentifikáciu, na základe ktorej identifikuje používateľa a vráti jeho údaje.
- PATCH /users/name - Umožňuje autentifikovanému klientovi zmeniť si svoje osobné údaje. V tele požiadavky sa musí nachádzať krstné meno a priezvisko.
- PATCH /users/password - Umožňuje autentifikovanému klientovi zmeniť si svoje heslo. V tele požiadavky sa musí nachádzať súčasné heslo, nové heslo a potvrdzovacie heslo.

2.4.3 Správa hodnotiacich matíc

Server poskytuje nasledujúce koncové body pre správu hodnotiacich matíc.

- POST /matrices - Umožňuje autentifikovanému klientovi vytvoriť hodnotiacu maticu. Telo požiadavky tvoria atribúty: názov matice, počet riadkov, počet stĺpcov a samotné hodnoty vo forme dvojrozmerného poľa.

- PATCH /matrices/:id - Umožňuje autentifikovanému klientovi upraviť hodnotiacu maticu. V tele požiadavky sa musí nachádzať názov matice, počet riadkov, počet stĺpcov a samotné hodnoty vo forme dvojrozmerného poľa. V parametroch požiadavky sa musí nachádzať id upravovanej hodnotiacej matice.

- GET /matrices - Vráti autentifikovanému klientovi všetky jeho vytvorené hodnotiace matice.

- GET /matrices/:id - Vráti autentifikovanému klientovi ním vytvorenú hodnotiacu maticu na základe zadaného identifikátora v parametroch požiadavky.

- DELETE /matrices/:id - Umožňuje autentifikovanému klientovi vymazať ním vytvorenú hodnotiacu maticu na základe zadaného id v parametroch požiadavky.

2.4.4 Správa hlasovacích relácií

Server poskytuje nasledujúce koncové body pre správu hodnotiacich matíc.

- POST /sessions - Umožňuje autentifikovanému klientovi vytvoriť hlasovaciu reláciu. V tele požiadavky sa musí nachádzať názov relácie a identifikátor hodnotiacej matice.

- GET /sessions/:hashId - Umožňuje akémukoľvek klientovi sa pripojiť do hlasovacej relácie na základe zadaného unikátneho identifikátora relácie v parametroch požiadavky.

- POST /sessions/:hashId/voting - Umožňuje autentifikovanému klientovi, ktorý je zároveň tvorcom danej hlasovacej relácie, vytvoriť nové hlasovanie v tejto relácii. Telo požiadavky musí obsahovať názov hlasovania a prípadne môže obsahovať jeho podrobnejší popis. V parametroch požiadavky sa musí nachádzať unikátny identifikátor danej relácie.

2.5 Aplikácia na strane klienta

Aplikácia na strane klienta bude komunikovať so serverom pomocou HTTP protokolu, využitím serverového API definovaného v sekcii 2.4.

2.5.1 Užívateľské rozhranie

Užívateľské rozhranie je plne responzívne pre všetky zariadenia, vrátane mobilných zariadení. Je intuitívne a jednoduché na použitie aj pre bežného používateľa. Skladá zo stránok uvedených nižšie. Pričom každá z uvedených stránok navyše obsahuje panel

nástrojov v hornej časti stránky. Stránky určené pre prihlásených používateľov, taktiež obsahujú navigáciu v ľavej časti stránky. Stránky pre neprihlásených používateľov obsahujú v paneli nástrojov tlačidlo na otvorenie nastavení. V prípade stránok pre prihlásených používateľov sa toto tlačidlo nachádza priamo v navigácii.

Úvodná stránka
Jedná sa o prvú stránku, s ktorou príde používateľ do kontaktu pri prvotnom navštívení aplikácie. Táto stránka slúži, ako vstupná brána do aplikácie, preto obsahuje základné informácie o tom, na čo aplikácia slúži, odkazy na stránku pre registráciu, prihlásenie a pripojenie sa do hlasovacej relácie. Okrem toho obsahuje odkazy na GitHub repozitáre aplikácie.

Stránka pre registráciu
Jedna sa o štandardnú registračnú stránku. Obsahuje formulár s poľami pre zadanie krstného mena, priezviska, e-mailu, hesla a potvrdzovacie hesla. Pod formulárom sa nachádza tlačilo na jeho odoslanie, odkaz na stránku prihlásenia a tlačidlo na vrátenie sa späť na úvodnú stránku.

Stránka pre prihlásenie
Táto stránka obsahuje formulár s poľami pre zadanie e-mailu a hesla. Pod formulárom sa nachádza tlačidlo na jeho odoslanie, odkazy na stránku pre registráciu, resetovanie hesla a nakoniec tlačidlo na vrátenie sa na úvodnú stránku.

Stránka pre resetovanie hesla
Stránka obsahuje formulár s polom pre zadanie e-mailu používateľa, tlačidlo na jeho odoslanie a odkaz pre vrátenie sa na stránku prihlásenia.

Domovská stránka pre prihlásených používateľov
Obsahuje stručné informácie o tom, ako vytvoriť hodnotiace matice, hlasovacie relácie a samotné hlasovanie v rámci relácie.

Stránka konta prihláseného používateľa
Táto stránka sa skladá z dvoch polovíc. Ľavá polovica obsahuje ikonu avatara, pod ktorou sa nachádza meno a email prihláseného používateľa. Pravá polovica sa skladá z dvoch podstránok, pričom používateľ sa môže preklikávať medzi nimi. Prvá podstránka slúži na zmenu osobných údajov, obsahuje formulár s poľami pre zadanie krstného mena a priezviska. Pod týmto formulárom sa nachádza tlačidlo na jeho odoslanie. Druhá podstránka slúži na zmenu hesla, obsahuje formulár s poľami pre zadanie súčasného hesla, nového hesla a potvrdzovacieho hesla. Pod formulárom sa opäť nachádza tlačidlo na

jeho odoslanie. Medzi týmito dvoma podstránkami sa vieme preklikávať pomocou odkazov. V paneli nástroj sa navyše nachádza tlačidlo pre odhlásenie sa z aplikácie.

Domovská stránka hodnotiacich matíc

Táto stránka obsahuje tabuľku všetkých doteraz vytvorených hodnotiacich matíc prihláseného používateľa. Stĺpcami tabuľky sú názov matice, počet jej riadkov, počet jej stĺpcov a dátum, kedy bola vytvorená. Kliknutím na názov jedného z týchto stĺpcov bude tabuľka podľa neho zoradená vzostupne alebo zostupne. Každý záznam v tejto tabuľke navyše obsahuje menu s tlačidlami pre upravenie, zobrazenie detailu a odstránenie matice. Taktiež sa na stránke nachádza tlačidlo na pridanie novej matice. Po kliknutí na tlačidlo pre upravenie alebo vytvorenie matice sa používateľovi zobrazí nové okno s formulárom. Formulár obsahuje pole pre zadanie názvu a mriežku s poľami pre zadanie jednotlivých hodnôt. Táto mriežka navyše obsahuje tlačidla na pridanie a odstránenie riadku alebo stĺpca polí. Pod formulárom sa nachádza tlačidlo na jeho odoslanie a tlačidlo na zatvorenie okna.

Stránka detailu hodnotiacej matice

Táto slúži na zobrazenie reprezentácie hodnotiacej matice formou hlasovacích kariet. Ďalej obsahuje jej názov a tlačidlá na jej upravenie a vymazanie.

Domovská stránka hlasovacích relácii

Stránka obsahuje iba dve tlačidla. Tlačidlo pre vytvorenie novej relácie, po jeho kliknutí sa používateľovi zobrazí okno s formulárom s polom pre zadanie názvu relácie a pole s možnosťami na výber hodnotiacej matice, ktorá sa v hre použije. Pod formulárom sa nachádza tlačidlo na jeho odoslanie a tlačidlo na zatvorenie okna. Druhým tlačidlom na stránke je tlačidlo na pripojenie sa do existujúcej hlasovacej relácie. Po jeho kliknutí sa zobrazí okno s formulárom s polom pre zadanie unikátneho reťazca relácie. Pod formulárom sa nachádza tlačidlo na jeho odoslanie a tlačidlo na zatvorenie okna.

Stránka prebiehajúcej hlasovacej relácie

Táto stránka sa bude skladať z dvoch typov rozhrania, rozhranie pre administrátora relácie a rozhranie pre bežných účastníkov. Oba typy rozhraní budú obsahovať zoznam všetkých momentálnych účastníkov spolu s identifikátorom, či daný používateľ už hlasoval alebo hodnotou jeho hlasu, ak boli hlasy účastníkov odkryté. Taktiež budú obsahovať panel pre aktuálne hlasovanie, kde sa bude nachádzať názov a prípadný popis hlasovania. Administrátorovi sa v tomto paneli navyše zobrazujú tlačidlo pre vytvorenie nového hlasovania. Po kliknutí naň sa mu zobrazí okno s formulárom s poľami pre zdanie názvu a popisu nového hlasovania. Pod formulárom sa nachádzajú tlačidlá na jeho odoslanie a zatvorenie okna. Taktiež sa mu navyše v tomto paneli zobrazuje

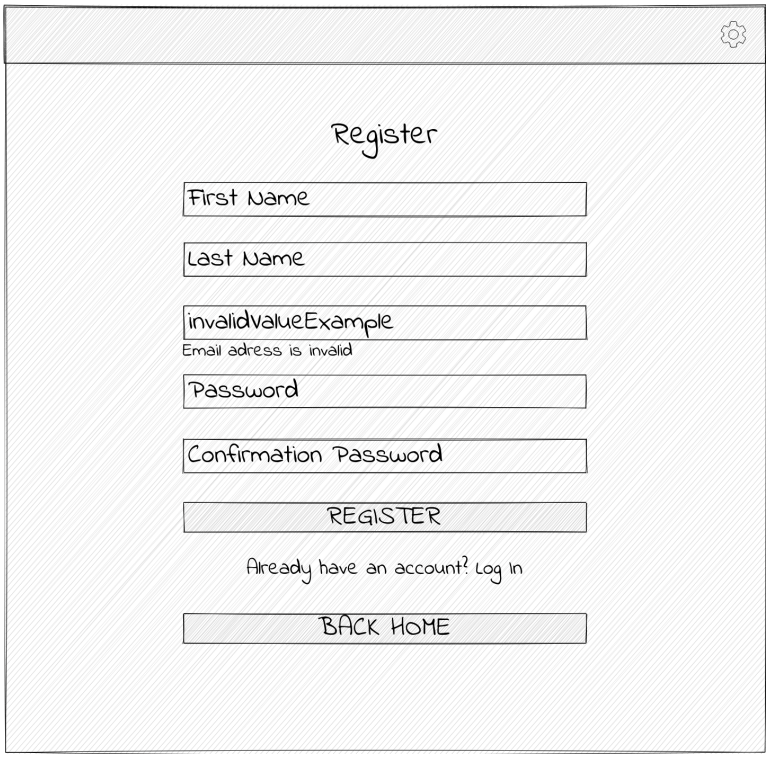
tlačidlo na zobrazenie hlasov účastníkov. V oboch verziách sa počas prebiehajúceho hlasovania budú zobrazovať hlasovacie karty, kliknutím na jednu z nich používateľ zahlasuje a daná karta bude označená, ako vybraná. Ak hlasovanie neprebieha, karty budú skryté.

Navigácia

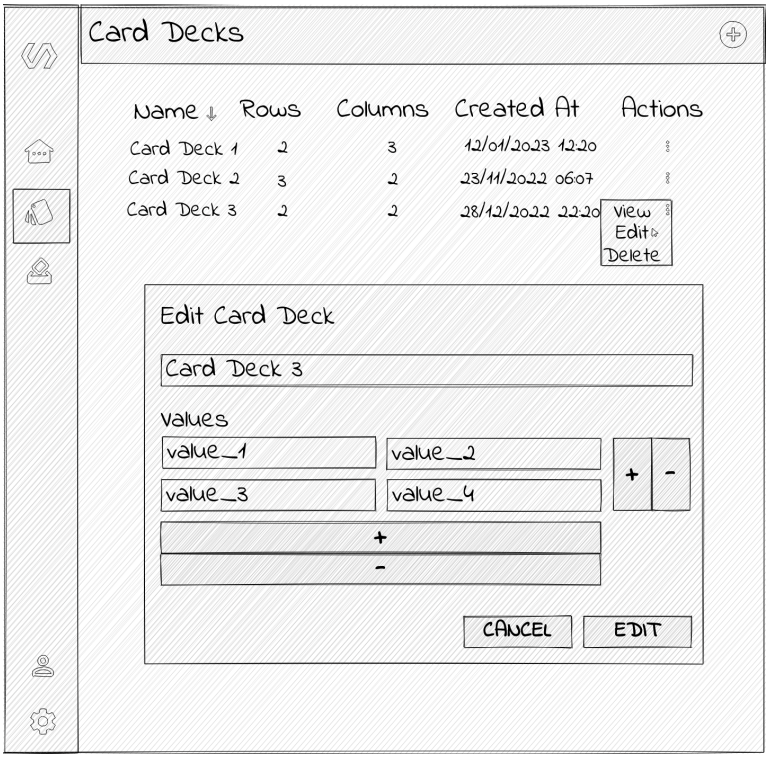
Navigáciu je možné zobrazovať v dvoch módoch, buď ako rozbalenú alebo zabalenú. Ak je navigácia zabalená, obsahuje iba tlačidlá s ikonami, ktoré reprezentujú jednotlivé stránky aplikácie. Ak je navigácia rozbalená, obsahuje navyše aj názvy jednotlivých stránok vedľa korešpondujúcich ikon.

Panel s nastaveniami

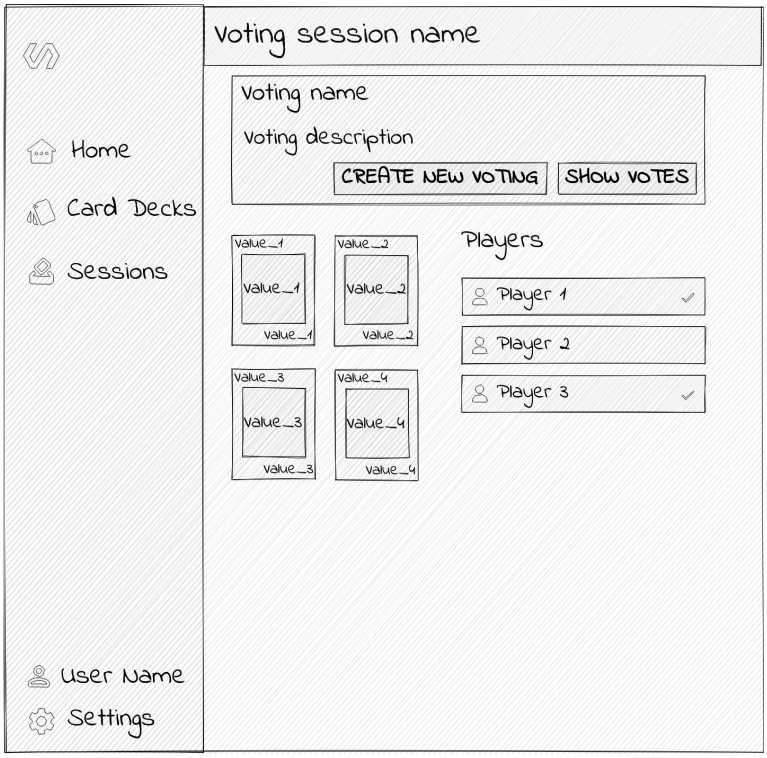
Rozbalí sa nad aktuálne otvorenou stránkou kliknutím na tlačidlo nastavení. Obsahuje tlačidlá na zmenu jazyka, zmenu témy na svetlú alebo tmavú a zmenu smeru textu. V prípade, že je používateľ prihlásený obsahuje navyše tlačidlo na zmenu módu navigácie.



Obr. 2.2: Návrh užívateľského rozhrania stránky pre registráciu s chybovou hláškou pre pole e-mailovej adresy



Obr. 2.3: Návrh užívateľského rozhrania domovskej stránky matíc s otvoreným oknom pre editáciu matice a zabalenou navigáciou



Obr. 2.4: Návrh užívateľského rozhrania prebiehajúcej hlasovacej relácie s aktívnym hlasovaním a rozbalenou navigáciou

Kapitola 3

Implementácia

Počas vývoja aplikácie bol kladený dôraz na splnenie viacerých noriem a postupov profesionálneho moderného softvérového inžinierstva, vďaka čomu bola zaručená vyššia kvalita výslednej aplikácie. Niekoľko z nich je opísaných v tejto kapitole, taktiež je tu opísaná štruktúra aplikácie a spôsoby implementácie niektorých jej častí.

3.1 Agile

Pri vývoji a riadení projektu bol uplatnený agilný prístup, s ohľadom na možnosti, ktoré bakalárska práca poskytuje. Na začiatku som spolu s mojim konzultantom zozbieral a spísal požiadavky a obmedzenia pre aplikáciu. Na základe týchto požiadaviek sme zostrojili produktový backlog v Kanban-e, ktorý sa dynamicky menil a aktualizoval počas celého vývoja. Vývoj bol založený na iteratívnom a inkrementálnom prístupe, realizovaný formou dvojtýždňových šprintov. Na začiatku každého šprintu som si vytvoril zoznam úloh, ktoré bolo treba do jeho konca dokončiť. Každý šprint bol zakončený stretnutím s konzultantom, na ktorom som predviedol aktuálnu verziu aplikácie a následne sme zhodnotili jej smerovanie a prípadné nedostatky. Vďaka úzkej spolupráci a komunikácii s mojím konzultantom bolo zabezpečené, že výsledná aplikácia bude zodpovedať požiadavkám a očakávaniam zadávateľa.

3.2 Testovanie

Aplikácia bola vyvíjaná s pomocou testami riadeného vývoja, čo znamená, že pred implementáciou väčšiny funkcionality boli najprv napísané testy, ktoré slúžili, ako formálna špecifikácia danej funkcionality. Vďaka tomu bola zabezpečená vyššia kvalita kódu, skoršie odhalenie chýb a lepšie porozumenie problematike. Zároveň napísané testy slúžili, ako kontrola po refaktorizácii a rozšírení kódu. Na testovanie boli využité predovšetkým integračné testy.

V prípade Back-Endu boli testované celé endpointy, čo znamená, že boli zároveň testované časti serveru, ako správne prijímanie a validovanie požiadaviek, ich spracovanie a odpovedanie na ne. Na tento účel som použil testovacie nástroje supertest a jest. Dôvodom testovania endpointov a nie samostatných middleware funkcií pomocou jednotkových testov bolo, že celý server sa skladá hlavne z middleware funkcií, ktoré prijímajú, ako argument objekt požiadavky a odpovede, ktoré sa ťažko simulujú. Taktiež sú tieto funkcie závislé jedna od druhej a unit testy nezaručujú, že budú spoločne fungovať správne. Dôležité je tiež spomenúť, že aplikácia využíva dve databázy, jednu dlhodobú pre ukladanie všetkých potrebných údajov a druhú krátkodobú pre ukladanie dočasných údajov počas testovania. Dlhodobá databáza predstavuje niečo, ako domáce zvieratko, jej štruktúra a obsah bola dlhodobo budovaná a je jedinečná v danom momente. Naopak, testovacia databáza je pri každom teste rovnaká a na konci testovania sa celý jej obsah vymaže. Ide o analógiu, ktorá sa vyskytuje v DevOps a nazýva sa aj domáce zvieratko verzus hromadný dobytok (z angl. Pet vs Cattle).

Pri testovaní Front-Endu som sa riadil filozofiou knižnice react testing library, ktorá hovorí „Čím viac sa testy podobajú spôsobu používania vášho softvéru, tým väčšiu istotu vám môžu poskytnúť“[19]. Zameriaval som sa na testovanie primárnych komponentov, ktoré využívajú iné menšie komponenty, pričom niektoré závislosti boli simulovali pomocou takzvaného „mockovania“. Na testovanie som využil nástroje react testing library a jest.

Niekedy však nebolo možné uplatniť TDD, napríklad z dôvodu chýbajúcich predošlých skúseností s danou technológiou, zložitosti danej časti aplikácie alebo časového obmedzenia. Niektoré testy boli teda dopísané dodatočne a niektoré časti aplikácie nie sú pokryté testami.

3.3 Správa verzií

Aplikácia využíva nástroje Git a GitHub na správu jej verzií. Všetky zmeny v kóde sú teda evidované a je možné kedykoľvek sa vrátiť k jej predchádzajúcim verziám. Celá aplikácia je zálohovaná a voľne dostupná pre kohokoľvek. Skladá z dvoch separátnych repozitárov pre Front-End a Back-End, pričom oba repozitáre využívajú nasledovný model vetvenia. Hlavnými vetvami sú vetvy main a develop, tieto vetvy majú nekonečnu životnosť a nesmú sa nikdy vymazať. Vetva main funguje štandardným spôsobom, obsahuje plne funkčný produkčný kód. Vetva develop obsahuje kód s najnovšími funkciami pre ďalšie vydanie. Keď zdrojový kód vo vetve develop dosiahne stabilný bod a je pripravený na vydanie, všetky zmeny sa zlúčia do vetvy main. Okrem týchto dvoch vetiev repozitáre využívajú pomocné krátkodobé vetvy, ktorých životnosť je dočasná. Jedná sa konkrétne o tieto typy vetiev.

- Feature vetvy
- Release vetvy
- Hotfix vetvy

Feature vetvy sa používajú na vývoj nových funkcií pre nadchádzajúcu verzie aplikácie. Existujú iba dovtedy, kým vývoj danej funkcie nie je dokončený. Potom sú zlúčené späť do develop, odkiaľ aj vznikli a následne sú vymazané. Release vetvy slúžia na pripravenia kódu do produkcie. Umožňujú opravy menších chýb a iných malých úprav. Keď je vetva pripravená na skutočné vydanie, je zlúčená s vetvou main a develop. Hotfix vetvy sú podobné release vetvám, sú tiež určené na prípravu na nové produkčné vydanie, avšak sú neplánované. Keď musí byť kritická chyba v produkčnej verzii okamžite vyriešená, vytvorí sa nová hotfix vetva od main vetvy. Po jej dokončení musí byť hotfix vetva zlúčená späť do main a develop vetvy a následne vymazaná.

3.4 Kódové štandardy

Počas vývoja aplikácie bol kladený dôraz na dodržiavanie určitých kódových štandardov, ktoré pomohli zabezpečiť konzistentnosť a prehľadnosť kódu. Jedným z týchto štandardov bolo konzistentné formátovanie kódu, ktoré zahŕňa napríklad použitie rovnakej syntaxe, odsadenia a medzier. Na zabezpečenie tohto formátovania bol použitý nástroj Prettier, ktorý automaticky formátuje kód na základe určenej konfigurácie. Okrem formátovania kódu bol tiež použitý konzistentný štýl názvoslovia pre premenné, funkcie, metódy a triedy v anglickom jazyku. V aplikácii sa používa štýl nazývaný camelCase, ktorý stanovuje, že každé slovo v názve začína veľkým písmenom s výnimkou prvého slova. Pri písaní kódu bolo tiež dbané na použitie moderných syntaktických prvkov TypeScriptu, ako sú napríklad await-async alebo rest operátory. Tieto prvky zjednodušujú syntax kódu a zlepšujú jeho výkon a čitateľnosť. Na druhej strane, zastarané prvky, ako sú var alebo then-catch, neboli používané, aby sa zabezpečilo, že kód je moderný a efektívny. Pre odhalenie a odstránenie niektorých chýb v kóde bol taktiež použitý nástroj ESLint. Tento nástroj pomáha odhaliť a opraviť problémy s kódom a zabezpečuje dodržiavanie určitých kódových štandardov. Kód aplikácie bol rozdelený do priečinkov podľa jeho funkcie. Napríklad priečinok „controllers“ obsahuje funkcie, ktoré spracovávajú a odpovedávajú na požiadavky, priečinok „services“ obsahuje služby, ktoré komunikujú s vonkajšími zdrojmi dát a priečinok „utils“ obsahuje pomocné funkcie a utility. Dodržiavanie kódových štandardov a použitie vhodných nástrojov a techník pomáha zabezpečiť kvalitný a prehľadný kód, čo zvyšuje efektivitu a rýchlosť vývoja.

3.5 Refaktorizácia

Refaktorizácia je dôležitou súčasťou testami riadeného vývoja a bola teda pravidelne uplatňovaná. Aplikácia taktiež prešla niekoľkými kompletnými refaktorizáciami, pričom sa používal prístup postupného vykonávania malých zmien v kóde a ich kontrolovanie spustením testov. Tento postup umožnil prehodnotiť a zlepšiť kód aplikácie, aby bol efektívnejší a ľahšie udržateľný. Taktiež umožnil identifikovať a opraviť problémy v kóde predtým, ako sa stanú závažnými. Výsledný kód bol vďaka tomu lepšie organizovaný a čitateľnejší.

3.6 Štruktúra klienta

Front-End aplikácie je implementovaný pomocou knižnice React a programovacieho jazyka TypeScript, navyše využíva knižnicu Material UI. Jeho základná verzia bola vytvorená pomocou nástroja React Create App. Tento nástroj umožňuje rýchle a jednoduché vytvorenie a nakonfigurovanie novej React aplikácie, vďaka čomu som sa mohol rovno pustiť do písania kódu. Na naprogramovanie základnej funkcionality aplikácie a definovanie vlastných štýlov pre Material UI komponenty bola využitá open-source šablóna administračnej aplikácie[11]. Táto šablóna poskytuje napríklad upravené komponenty z knižnice Material UI, funkcionalitu zobrazovania upozornení, funkcionalitu nastavení a omnoho viac. Celý Front-End aplikácia je delení do malých logických komponentov.

3.6.1 Spravovanie formulárov

Väčšina stránok aplikácie obsahuje formulár v nejakej podobe, bolo teda dôležité vybrať správne nástroje na uľahčenie procesu ich vytvárania, odosielania a validácie. React poskytuje všetku potrebnú funkcionalitu pre vykonanie spomínaných procesov. Avšak tento spôsob implementácie je časovo náročnejší, komplikovanejší a výsledný kód je horšie čitateľný. Preto som radšej siahol po dvoch populárnych nástrojoch. Prvým z nich je Formik, tento nástroj zjednodušuje proces vytvárania formulárov poskytnutím sady komponentov a react hook-ov. Druhým nástrojom je Yup, jedná sa o validačnú knižnica, ktorú je možné použiť spolu s Formik-om na validáciu údajov z formulárov pred ich odoslaním. Spojením týchto dvoch nástrojov dostávame robustné riešenie, ktoré zabezpečí kompletné spravovanie formulárov.

3.6.2 Komunikácia so serverom

Klient musí pravidelne komunikovať so serverom, či už sa jedná o poslanie dát na server, načítanie dát zo serveru alebo autentifikáciu používateľa. Javascript preto poskytuje

Fetch API, ktoré umožňuje takúto komunikáciu, avšak funkcionalita tohto API je obmedzená. Rozhodol som sa preto pre populárnu alternatívou nazývanú Axios. Axios funguje na podobnom princípe ako Fetch API, avšak poskytuje extra funkcionalitu, ako napríklad: zrušenie požiadavky, časový limit pre odpoveď serveru a zachytávanie požiadaviek. Druhým nástrojom, ktorý zjednoduší spravovanie komunikácie, je nástroj React Query. React Query je predkonfigurovaná knižnica na správu údajov v React aplikáciách. Poskytuje funkcionalitu, ako napríklad: caching, aktualizácia neaktuálnych údajov na pozadí alebo spájanie viacerých žiadostí o rovnaké údaje do jednej. Spojením týchto dvoch nástroj a ich implementáciou„ ako react hook dostávame jednoduché riešenie pre zabezpečenie komunikácie so serverom.

Algoritmus 3.1: Príklad implementácie funkcionality pre odosielanie HTTP požiadavky o registráciu používateľa pomocou nástrojov Axios a React Query

```
const register = async (userData: {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
  confirmationPassword: string;
}): Promise<string> => {
  const { data } = await axios.post('/register', userData);
  return data.data.userId;
};

export const useRegister = () => {
  axios = useAxios();
  const { isLoading, mutateAsync } = useMutation(register);
  return { isRegistering: isLoading, register: mutateAsync };
};
```

3.7 Štruktúra servera

Server využíva prostredie Node.js a programovací jazyk TypeScript. Dáta sú ukladané do PostgreSQL databázy, s ktorou server interaguje pomocou ORM knižnice Prisma. Na zjednodušenie a urýchlenie vývoja sa na serveri využíva Node.js rozšírenie nazývané Express.js. Server sa skladá sa nasledujúcich primárnych časti.

3.7.1 Smerovanie (Routing)

Každá prijatá HTPP požiadavka od klienta musí byť patrične spracovaná. Proces mapovania HTPP požiadaviek na funkcie, ktoré ich spracujú a odošlú odpoveď späť klientovi, sa nazýva smerovanie. Na tomto serveri je smerovanie zabezpečené pomocou triedy express.Router, ktorá umožňuje definovanie modulárnych, pripojiteľných tras pre požiadavky. Takáto trasa sa skladá z nasledujúcich troch časti. Route method symbolizuje HTTP metódu danej trasy. Route path označuje URL cestu danej trasy. A route handler označuje funkciu na spracovanie danej požiadavky. Na serveri sú tieto trasy definované v súboroch v priečinku routes, pričom využívajú funkcie z kontrolérov. Do definície trasy je taktiež možné pridať middleware funkcie napríklad pre zabezpečenie validácie.

Algoritmus 3.2: Príklad trasy pre požiadavku na registráciu používateľa

```
export const authRouter = express.Router();

authRouter.post(
  '/register',
  validate(regsiterSchema),
  registerUserHandler
);
```

3.7.2 Kontroléri (Controllers)

Pojem kontrolér označuje skupiny funkcií, ktoré slúžia na spracovanie HTTP požiadaviek na určitých endpointoch. Typicky spracujú dáta prijate od klienta v požiadavke a odošlú patričnú odpoveď na danú požiadavku klientovi. Vzhľadom na to, že server využíva rozšírenie Express.js, takéto funkcie dostanú tri argumenty: objekt požiadavky Request, objekt odpovede Response a funkciu Next. Objekt požiadavky obsahuje rôzne informácie o danej požiadavke a odoslané dáta od klienta. Objekt odpovede reprezentuje odpoveď na HTTP požiadavku, ktorú server odosielal späť klientovi a obsahuje rôzne pomocné metódy a vlastnosti na jej modifikovanie. Funkcia Next slúži na presunutie spracovania požiadavky na nasledovnú middleware funkciu.

Algoritmus 3.3: Príklad funkcie z kontroléra na vrátenie informácii o prihlásenom používateľovi

```
export const getUserHandler = async (
  req: Request,
  res: Response,
  next: NextFunction
```

```
): Promise<void> => {
  try {
    const user = res.locals.user;
    const { id, firstName, lastName, email } = user;

    res.status(HttpCode.OK).json({
      message: req.t('user.get.success'),
      data: {
        user: {
          id,
          firstName,
          lastName,
          email,
        } as UserData,
      },
    });
  } catch (err: any) {
    next(err);
  }
};
```

3.7.3 Validácia

Klientske požiadavky, ktoré obsahujú určité dáta, si vyžadujú skontrolovanie splnenia zadefinovaných kritérií na ich formát. Validáciu požiadaviek na serveri zabezpečuje knižnica Yup. Jedná o validačnú knižnicu, ktorá poskytuje kolekciu funkcií na validáciu a sanitizáciu dát od používateľa. Pravidlá, ktoré musia dáta z požiadavky spĺňať, môžeme vďaka tejto knižnice definovať pomocou takzvaných validačných schém. Ukážku takejto schémy nájdete, ako Algoritmus 3.4. V prípade, že validácia zlyhá, odošle sa klientovi odpoveď na požiadavku so status kódom 400 (Bad Request) a zoznamom s informáciami o jednotlivých validačných chybách tak, aby mohli by na strane klienta spracované a zobrazené. Všetky validačné schémy sa nachádzajú v priečinku schemas.

Algoritmus 3.4: Príklad schémy pre validáciu požiadavky na resetovanie hesla

```
export const resetPasswordSchema = yup.object({
  body: yup.object({
    password: yup
      .string()
      .required('common.validations.required')
```

```
      .matches(
        passwordRegex,
      'common.validations.password.weak'
      ),
    confirmationPassword: yup
      .string()
      .required('common.validations.required')
      .oneOf(
        [yup.ref('password')],
        'common.validations.password.match'
      ),
  }),
});
```

3.7.4 Modely (Models)

Štandardne sa modely v Node.js vytvárajú manuálne pomocou knižníc ORM (objektovo-relačné mapovanie). Modely vytvorené pomocou týchto knižníc umožňujú definovať štruktúru dát a vzťahov medzi nimi využitím JavaScript kódu, pričom tieto modely následne môžu byť použité na interakciu s databázou. Na serveri sa využíva ORM knižnica Prisma, ktorá pristupuje k tvorbe modelov odlišným spôsobom. Prisma vytvára modely na základe štruktúry databázy, ktorú vývojár zadefinoval v špeciálnom jazyku zvanom Prisma Schema Definition Language (SDL) v súbore schema.prisma. Po definovaní schémy sa Prisma automaticky dokáže pripojiť k databáze, vytvoriť tabuľky a taktiež vygeneruje Prisma Client-a, na základe danej schémy, ten bude umožňovať interakciu s databázou pomocou jednoduchého API. Všetky modely sú teda na serveri definované v súbore schema.prisma.

Algoritmus 3.5: Príklad modelu používateľa definovaný v prisma.schema

```
model User {
  id           String  @id @default(uuid())
  firstName    String  @db.VarChar(50)
  lastName     String  @db.VarChar(50)
  email        String  @unique
  password     String
  verified           Boolean  @default(false)
  verificationCode   String   @unique @default(uuid())
  passwordResetToken String?
  passwordResetAt    DateTime?
```

```
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  matrices Matrix[]
  sessions Session[]
  @@map("users")
}
```

3.7.5 Služby (Services)

Služby slúžia na oddelenie logiky aplikácie od jej ostatných častí. Obsahujú funkcie na zjednodušenie spracovania požiadaviek. Na tomto serveri sa používajú predovšetkým na komunikáciu s databázou využitím Prisma Client API. Poskytujú teda funkcionalitu na vytváranie, aktualizovanie a vymazávanie záznamov a vytváranie dopytov. Kontroléri využívajú tieto funkcie pri spracovávaní požiadaviek. Nachádzajú sa v priečinku services.

3.8 Autentifikácia

Aplikácia využíva Stateless autentifikáciu implementovanú pomocou JWT (JSON Web Token) access a refresh tokenu. Server teda neuchováva žiaden stav o prihlásenom klientovi. JWT je formát tokenu, ktorý obsahuje v sebe informácie o prihlásenom používateľovi a je zakódovaný pomocou tajného kľúča na strane servera. Konkrétna implementácia autentifikácie v aplikácii je nasledovná. Klient odošle na server požiadavku o prihlásenie. V prípade korektných údajov server vytvorí refresh a access token, pričom oba tieto tokeny majú nastavenú určitú dobu platnosti, v prípade access tokenu je táto doba výrazne kratšia. Následne vráti klientovi odpoveď, v ktorej tele sa bude nachádzať access token. Zároveň odošle aj takzvané HTTPOnly cookies, ktoré zabezpečujú, že sa k ním nedá pomocou Javscriptu dostať a budú uchovávať access a refresh token. Klient následne musí pri každej požiadavke na chránenú cestu špecifikovať access token v hlavičke „Authorization“ alebo v HTTPOnly cookie. Akonáhle zlyhá požiadavka na takúto cestu z dôvodu vypršania platnosti daného tokena, klient automaticky odošle požiadavku na jeho obnovenie, pričom sa využije refresh token z HTTPOnly cookie. Ak je refresh token platný, server vytvorí nový access token a vráti ho klientovi. V opačnom prípade, ak je refresh token neplatný, bude používateľ odhlásený z aplikácie. Dôvodov, prečo som si zvolil stateless a nie statefull autentifikáciu, je niekoľko: je jednoduchšia na implementáciu, nie je potrebné načítať stav z databázy, čo vedie k rýchlejšiemu spracovaniu požiadaviek a vo všeobecnosti je bezpečnejšia.

3.9 Real-time komunikácia

Počas hlasovania je kľúčové, aby všetci účastníci boli informovaní v reálnom čase o vykonaných akciách ostatných účastníkov. Na tento účel som použil nástroj Socket.io, ktorý umožňuje obojstrannú komunikáciu medzi serverom a klientom. Zakaždým, keď klient vykoná nejakú akciu, o ktorej by mali byť ostatní účastníci informovaní, odošle o tom na server informáciu. Ten ju interne spracuje, uchová potrebné informácie v pamäti a upravenú ju prepošle všetkým účastníkom hlasovacej relácie. Keďže každé pripojenie k relácii vytvára nové socket spojenie, musel som vyriešiť problém s detekciou rovnakého používateľa v prípade otvorenia novej karty alebo okna prehliadača. Mojím riešením bolo vygenerovanie unikátneho reťazca a jeho uloženie do lokálnej pamäte. Pri každom pripojení sa potom použil tento reťazec na identifikáciu používateľa. Ak bol používateľ prihlásený, použil som jeho id, ako unikátny identifikátor vďaka tomu bude detegovaný, ako rovnaký používateľ aj v rámci viacerých prehliadačov. Týmto spôsobom som zabezpečil, že všetci účastníci hlasovania boli spoľahlivo informovaní o všetkých potrebných udalostiach.

3.10 Lokalizácia a internacionalizácia

Lokalizácia je dôležitým aspektom aplikácií, umožňuje prispôsobiť aplikáciu jazyku a regiónu používateľa. V mojej aplikácii sa tým zaoberá nástroj i18next, ktorý funguje na strane klienta aj servera. Jeho použitie je pomerne jednoduché. Každý podporovaný jazyk má definovaný svoj vlastný JSON súbor, ktorý obsahuje preklady všetkých fráz a slov. Okrem toho je potrebné vytvoriť konfiguračný súbor, kde sa definuje, ktoré jazyky aplikácia podporuje, kde sa nachádzajú prekladové súbory a iné pomocné nastavenia pre nástroj i18next. Táto aplikácia podporuje Anglický a Slovenský jazyk. V React aplikácii sa používa hook useTranslation, ktorý umožňuje jednoduché použitie nástroja i18next. V Node.js sa na to používa funkcia t() z objektu požiadavky. Server deteguje jazyk klienta na základe hlavičky „Accept-Language“ a potom odpovedá v tomto jazyku. Správne nastavenie jazyka je kritické najmä v prípade validačných chýb, ktoré nie je možné detegovať na strane klienta, ako napríklad duplicitné použitie e-mailu alebo nesprávne prihlasovacie údaje. V takomto prípade budú na strane klienta zobrazené priamo chybové hlášky z odpovedi serveru.

3.11 Dockerizácia

Dockerizácia aplikácie spočíva v tom, že sa vytvoria takzvané docker kontajneri, v ktorých bude aplikácia bežať. Front-End, Back-End a obe databázy bežia v samostatných kontajneroch. Na vytvorenie kontajnera sa používa súbor Dockerfile, ktorý definuje,

ako bude kontajner zostavený a súbor docker-compose.yml zabezpečujúci vzájomnú konfiguráciu viacerých kontajnerov. Po spustení príkazu docker-compose up sa tieto kontajnery naštartujú a spustí sa celá samotná aplikácia. Dockerizácia aplikácie mala viacero výhody. Izolovala aplikáciu od systému, aby sa zabránilo konfliktom s inými aplikáciami. Nasadenie do prevádzky bolo jednoduchšie, keďže nebolo nutné priamo inštalovať a konfigurovať závislosti v danom systéme. Taktiež bolo možné aplikáciu spustiť v akomkoľvek prostredí, kde bol nainštalovaný Docker. Ďalšou výhodou je možnosť jej jednoduchšieho škálovania v budúcnosti.

Kapitola 4

Nasadenie

Výsledná aplikácia bola nasadená do prevádzky pomocou platform Amazon Web Services (AWS) a Docker Hub. AWS je cloudová platforma, ktorá poskytuje rôzne služby a APIs pre jednotlivcov aj podniky. Zahŕňa úložiská, databázy a virtuálne počítače, čo umožňuje jej používateľom vyhnúť sa budovaniu vlastnej infraštruktúry. Docker Hub je služba od spoločnosti Docker, ktorá umožňuje vyhľadávať a zdieľať obrazy docker kontajnerov.

Pred nasadením aplikácie bolo nutné pripraviť projekt na produkčnú verziu. Skontroloval som jeho správne fungovanie, nakonfiguroval som ho na prevádzkovú verziu a vytvorili docker obrazy, ktoré ho prevedú do produkčného stavu. Potom som si zaregistroval účet na oboch spomínaných platformách. Docker Hub ponúka bezplatné zverejňovanie a využívanie verejných obrázkov, zatiaľ čo AWS ponúka 12 mesačné obdobie bezplatného využívania niektorých ich služieb po prvom registrovaní. Jednou z týchto služieb je EC2 (Elastic Compute Cloud), ktorá poskytuje virtuálne počítače v cloude. Následne som vytvoril a nakonfiguroval EC2 inštanciu, tak aby som sa vedel na ňu pripojiť pomocou SSH.

Nasledovalo zverejnenie vytvorených produkčných docker obrazov na Docker Hub. Následne som ich stiahol do EC2 inštancie a vytvoril z nich kontajnery, ktoré už predstavujú samotný Back-End a Front-End aplikácie. Posledným krokom bolo nakonfigurovanie portov inštancie tak, aby bola aplikácia dostupná zvonka pre každého. Od dokončenia tohto procesu je aplikácia voľne dostupná pre každého, v čase písania bakalárskej práce na pridelenej adrese: http://13.48.124.74.

Kapitola 5

Testovanie použiteľnosti

Po nasadení aplikácie do produkcie bolo realizované testovanie použiteľnosti s cieľom posúdiť, ako dobre používatelia dokážu pracovať s aplikáciou. Jeho hlavným zámerom bolo odhaliť potenciálne nedostatky a chyby, ktoré by mohli negatívne ovplyvniť použiteľnosť a spokojnosť používateľov s aplikáciou. Testovanie bolo vykonané dvomi osobami, vo vekoch 19 a 21 rokov bez predošlých znalostí o danej problematike.

Ich úlohou bolo vykonať niekoľko jednoduchých úloh a okomentovať ich myšlienkové pochody. Konkrétne sa jednalo o úlohy typu: zaregistrovať sa, vytvoriť nové hlasovanie alebo resetovať heslo. Počas plnenia týchto úloh som monitoroval ich interakcie s produktom a zaznamenával ich spätnú väzbu a reakcie. Po ukončení testovania boli získané výsledky analyzované a vyhodnotené.

Bolo objavených niekoľko nedostatkov a zlých designových rozhodnutí. Medzi identifikované nedostatky patrilo napríklad nesprávne zobrazenie dlhých hodnôt na hlasovacích kartách, chýbajúce tlačidlo pre skopírovanie unikátneho identifikátora pre pripojenie sa do hlasovacej relácie alebo chýbajúce tlačidlo pre zobrazenie znakov hesla vo formulároch. Zlými designovými rozhodnutiami bolo napríklad: málo informatívne tlačidlo pre pridanie hodnotiacej matice alebo chýbajúce varovanie o neexistujúcich hodnotiacich maticiach pri vytváraní hlasovacej relácie.

Na základe týchto a ďalších zistení som v aplikácii opravil objavené chyby a implementoval nové zlepšenia. Následne som vydal a nasadil do prevádzky novú verziu aplikácie.

Záver

Cieľom bakalárskej práce bolo vytvoriť interaktívny nástroj na viacrozmerné plánovanie agilných projektov, pomocou techniky Plánovacia hra. Pričom mal poskytovať možnosť vytvárania dvojrozmerných hodnotiacich matíc. Počas vývoja aplikácie mal byť kladený dôraz na dodržiavanie moderných noriem softvérového inžinierstva. Nástroj by mal byť taktiež nasadený do prevádzky.

Výsledná aplikácia spĺňa všetky spomínané požiadavky a je voľne dostupná na použitie. Je implementovaná pomocou žiadaných technológie a pri vývoji boli použité viaceré postupy a normy profesionálneho moderného softvérového inžinierstva. Aplikácia je otestovaná a ľahko rozšíriteľná o novú funkcionalitu.

Aplikácia by ďalej mohla by rozšírená vo veľa rôznych smeroch. Určite by bolo vhodné pridať lepšie manažovanie hlasovacích relácii, napríklad ich zaheslovanie, a možnosť administrátora vyhodiť účastníka alebo zamedziť jeho možnosť hlasovať. Ďalej by mohla byť pridaná možnosť chatovania počas hlasovania, vzhľadom na to, že aplikácia už poskytuje funkcionalitu real-time komunikácie, implementácia by bola pomerne jednoduchá. Taktiež by bolo vhodné pridať možnosť nahrania avatara používateľa, na čo už je používateľské rozhranie pripravené.

Prínosom bakalárskej práce pre mňa bolo naučenia sa a spoznanie veľkého množstva nových technológií, získanie nových skúseností v oblasti FullStack vývoja webových aplikácií, oboznámenie sa a osvojenie si niektorých noriem moderné a profesionálnych prístup k vývoju a riadeniu projektov. Zároveň agilné tímy, napríklad z rakúskej spoločnosti A1, majú teraz k dispozícii online nástroj poskytujúci inovatívny prístup k hodnoteniu úloh v agilných projektoch.

Literatúra

[1] About node.js. [Naposledy navštívené 21/05/2023] Dostupné z https://nodejs.org/en/about.

[2] Express.js. [Naposledy navštívené 21/05/2023] Dostupné z https://expressjs.com.

[3] Introduction. [Naposledy navštívené 21/05/2023] Dostupné z https://socket.io/docs/v4.

[4] Introduction to json web tokens. [Naposledy navštívené 21/05/2023] Dostupné z https://jwt.io/introduction.

[5] Jest. [Naposledy navštívené 21/05/2023] Dostupné z https://jestjs.io.

[6] Material ui. [Naposledy navštívené 21/05/2023] Dostupné z https://mui.com.

[7] Planningpoker.com. [Naposledy navštívené 21/05/2023] Dostupné z https://planningpoker.com.

[8] Planningpokeronline.com. [Naposledy navštívené 21/05/2023] Dostupné z https://planningpokeronline.com.

[9] Planningpokeronline.com. [Naposledy navštívené 21/05/2023] Dostupné z http://firepoker.io.

[10] React. [Naposledy navštívené 21/05/2023] Dostupné z https://react.dev.

[11] React material admin. [Naposledy navštívené 21/05/2023] Dostupné z https://github.com/m6v3l9/react-material-admin.

[12] Typescript. [Naposledy navštívené 21/05/2023] Dostupné z https://www.typescriptlang.org.

[13] What is a container? [Naposledy navštívené 21/05/2023] Dostupné z https://www.docker.com/resources/what-container.

[14] What is a rest api? [Naposledy navštívené 21/05/2023] Dostupné z `https://www.ibm.com/topics/rest-apis`.

[15] What is prisma? [Naposledy navštívené 21/05/2023] Dostupné z `https://www.prisma.io/docs/concepts/overview/what-is-prisma`.

[16] Why docker? [Naposledy navštívené 21/05/2023] Dostupné z `https://www.docker.com/why-docker`.

[17] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland and Dave Thomas. Manifesto for agile software development, 2001. [Naposledy zobrazené 21/05/2023] Dostupné z `https://agilemanifesto.org`.

[18] Kent Beck. *Test Driven Development: By Example*. O'Reilly Media, 2002.

[19] Kent C. Dodds. *Guiding Principles*. 2018. [Naposledy navštívené 21/05/2023] Dostupné z `https://testing-library.com/docs/guiding-principles`.

[20] Martin Fowler and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.

[21] J. Myslín. *Scrum: Průvodce agilním vývojem softwaru*. Computer Press, 2016.

[22] Z. Šochová and E. Kunce. *Agilní metody řízení projektů*. Computer Press, 2014.