FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Organizátor akcí
Student:	Michal Štefaňák
Vedoucí:	Ing. David Šenkýř
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této bakalářské práce je návrh a implementace systému pro organizování akcí – primárně zaměřeného na evidenci potvrzení (ne)účasti pozvaných hostů.

Postupujte v následujících krocích.

1. Stručně analyzujte existující systémy podporující organizaci akcí.
2. Navrhněte požadavky na funkcionalitu nového systému, který by měl umožnit:
   a. vytvoření akce podle šablony,
   b. rozeslání pozvánek v podobě QR kódu nebo textového odkazu,
   c. potvrzení (ne)účasti pozvaných hostů,
   d. sdílení odkazů spojených s akcí – například odkaz na fotografie.
3. Zvolte vhodnou implementační platformu.
4. Návrh implementujte a otestujte.
5. Shrňte dosažené výsledky.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 11. ledna 2022 v Praze.

Bakalárska práca

ORGANIZÁTOR AKCIÍ

Michal Štefaňák

Fakulta informačních technologií
Katedra softwarového inžinierstva
Vedúci: Ing. David Šenkýř
11. mája 2022

Obsah

Zoznam obrázkov

Zoznam výpisov kódu

Chcel by som poďakovať môjmu vedúcemu Ing. Davidovi Šenkýrovi za jeho pomoc pri tvorbe bakalárskej práce a za jeho neoceniteľnú podporu. Taktiež ďakujem mojim rodičom, priateľke a kamarátom za podporu počas celého štúdia.

Vyhlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č.121/2000 Sb., autorského zákona, v znení neskorších predpisov, najmä skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o užití tejto práce ako školského diela podľa 60 § odst. 1 citovaného zákona.

V Prahe dne 11. mája 2022 .....

Abstrakt

Táto bakalárska práca sa zaoberá vývojom informačného systému pre podporu a pomoc s organizovaním spoločenských akcií rôzneho typu. Systém má užívateľovi pomôcť vytvoriť zoznam hostí, odoslať pozvánky a zbierať od hostí odpovede ohľadne účasti na akcii a odpovede na organizátorom vytvorené otázky. Práca sa v úvode zaoberá prieskumom existujúcich konkurenčných riešení a následne špecifikáciou funkčných a nefunkčných požiadavkov. Na základe vykonanej analýzy sa vyberie vhodná platforma a technológia, v ktorej bude systém vyvinutý. Výsledkom je funkčný systém, ktorý umožní organizovať akcie. Na záver práce sa opíše užívateľské a automatické testovanie vytvoreného systému.

Kľúčové slová organizácia spoločenskej akcie, návrh webovej aplikácie, REST, Single Page App, Django, React

Abstract

This bachelor thesis focuses on the development of an information system for support and assistance with organizing social events of various types. The system is designed to help a user create a guest list, send invitations and collect answers from guests regarding participation in the event and answers to questions created by the organizers. In the beginning, the work deals with the survey of existing competitive solutions and, subsequently the specification of functional and non-functional requirements. Based on the performed analysis, a suitable platform and technology in which the system will be developed will be selected. The result is a functional system that allows users to organize events. At the end of the work I will describe the user testing and automatic testing of the created system.

Keywords event organization, web application design, REST, Single Page App, Django, React

Zoznam skratiek

API	Application Programming Interface
REST	Representational State Transfer
DOM	Document Object Model
JSON	Javascript Object Notation
CRUD	Create, Retrieve, Update, Delete
ORM	Object-Relational Mapper
UUID	Universal Unique Identifier
JWT	JSON Web Token

Úvod

Každý, kto kedy organizoval spoločenskú akciu, či už menšiu pre pár ľudí, alebo väčšiu v pre stovky známych, vie, že udržať prehľad o hosťoch nie je vôbec jednoduché. Najmä, ak k tomu nemá vhodný systém a prostriedky. Vždy je potrebné zostaviť zoznam hostí, každému doručiť pozvanie, získať odpoveď, napríklad, či príde alebo nepríde a odpoveď v najlepšom prípade nezabudnúť a nestratiť. Samozrejme je možné si všetky detaily pracne zapisovať a zbierať, no bolo by určite príjemnejšie a jednoduchšie riešenie tohoto problému automatizovať. No a práve dnešná doba má schopnosť takéto prostriedky poskytnúť oveľa jednoduchšie ako kedykoľvek predtým.

Užívateľsky prívetivý systém, ktorý by s organizovaním pomohol, by určite ocenila väčšina ľudí, ktorý sú radi, keď spomínané situácie s organizovaním akcií majú pod kontrolou. A v takejto situácii sa ocitol skoro každý.

Ja som taktiež neraz organizoval takúto akciu a bol som pomerne nespokojný s pomocou, ktorú som našiel v dostupných systémoch. Niektoré plnili účel, ale dostať sa k výsledku bolo veľmi zložité, či už pre mňa, alebo hostí. Iné boli veľmi formálne, ďalšie boli formálne málo. Vo väčšine prípadov mi chýbala možnosť pozvanie kreatívne vyjadriť sa. Keď tam táto možnosť bola, tak chýbal spomínaný zber odpovedí. Preto som sa rozhodol, že daný systém vo svete chýba a ja mám možnosť ho navrhnúť.

V tejto práci sa budem takýmto návrhom systému na organizovanie spoločenských akcií zaoberať. Ako prvé je potrebné zhodnotiť aktuálne možnosti organizovania. Tomu sa budem venovať v prvej kapitole práce – *Analýza konkurencie*. V tejto časti sa zameriam na nedostatky, ale aj užitočné aspekty pozorovaných systémov. Z týchto zistení v ďalšej kapitole zostavím požiadavky na mnou vyvíjaný systém. V tretej kapitole práce systém navrhnem. V kapitole *Implementácia* systém implementujem a v poslednej kapitole ho patrične otestujem.

Cieľ práce

Práca sa zameriava na návrh a implementáciu systému pre podporu organizovania akcií. Systém zjednoduší vytváranie pozvánok pre spoločenskú akciu, rozosielanie pozvánok a následne sprehľadní zber odpovedí od jednotlivých hostí. K dosiahnutiu požadovaného výsledku je potrebné vykonať niekoľko čiastočných krokov.

Analýza existujúcich riešení

Na začiatku je potrebné analyzovať aktuálne možnosti, ktoré už danú problematiku riešia. Zo získaných informácií sa následne zistí, čo aktuálnym riešeniam chýba a čo by sa dalo vylepšiť.

Návrh požiadavkov

Na základe toho sa vykoná analýza funkčných a nefunkčných požiadavkov. Ďalej je potrebné vytvoriť návrh architektúry, tak aby vyhovovala funkčným a nefunkčným požiadavkom.

Výber implementačnej platformy

Ďalšou dôležitou časťou je výber platformy, pre ktorú bude systém vyvinutý. Pre danú platformu sa vyberie vhodná technológia. Technológia bude posudzovaná podľa výhod a nevýhod, ktoré pri vývoji a pre výsledný produkt prináša.

Implementácia

Na záver je potrebné návrh implementovať a patrične ho otestovať.

Kapitola 1

Analýza konkurencie

Pri návrhu akéhokoľvek systému pre zadaný problém je potrebné zo všetkého najskôr zistiť, či už neexistuje rovnaké, alebo podobné riešenie problému. V tejto kapitole preskúmam služby, ktoré sa venujú organizovaniu akcií v miere, v akej predpokladám, že by mal fungovať aj mnou navrhnutý systém. Zaoberať sa teda budem službami, ktoré majú možnosť vytvoriť spoločenskú akciu a pozvať na ňu hostí. Zameriam sa na to, čo prinášajú navyše, ako tvorbu akcie uľahčujú a čo by sa dalo zmeniť k lepšiemu. Analýzu budem vykonávať nad aplikáciami, ktoré sú všeobecne známe a neskôr nad aplikáciami dostupnými z počítača, keď budem vyhľadávať službu na internete podľa popisu. Budem sa zameriavať na možnosti pozývania hostí, ako napríklad, či ide pozývať cez link, alebo QR kód. Ďalej sa budem zameriavať na možnosti úpravy pozvánky, na možnosti získavania informácií od hostí a poprípade ďalšie výhody konkurenčných služieb.

1.1 Google Calendar

Keď sa povie systém na organizovanie akcie, tak väčšinu ľudí prvé napadne pravdepodobne Google Calendar, ktorý sa stal súčasťou života väčšiny ľudí s Google účtom, alebo mobilným zariadením so systémom android. Užívateľ môže v aplikácii vytvoriť udalosť a pozvať na ňu ľudí cez e-mail. Pozvaný hosť následne svoju účasť potvrdí, alebo zamietne.

Nevýhody:

- Nie je možné upravovať možnosti na odpoveď, alebo pridávať ďalšie otázky.
- Nedá sa upraviť vzhľad pozvánky, ktorú hosť obdrží.
- Akciu nie je možné jednoducho zdieľať cez QR kód. Užívateľ by ho musel vytvoriť pomocou inej služby.

1.2 Akcia na Facebooku

Ďalšou populárnou možnosťou je vytvoriť akciu v službe Facebook. Organizátor môže vytvoriť akciu verejnú, alebo súkromnú, kde pridá a pozve hostí. K akcii môže pridať detailný popis a obrázky. Organizátor ďalej môže zbierať informácie o pozvaných hosťoch, ktoré sú dostupné na Facebooku a o tom či účasť potvrdili, alebo nepotvrdili.

Nevýhody:

- Hostia musia mať účet na Facebooku.
- Pozvánka, alebo v tomto prípade stránka akcie je upraviteľná, len do určitej miery. Môže sa pridať titulný obrázok akcie a zmeniť názov a popis, no detaily ako farba a veľkosť písma zostávajú z pôvodných nastavení.
- Akciu nie je možné jednoducho zdieľať cez QR kód. Užívateľ by ho musel vytvoriť pomocou inej služby.

Výhody:

- Hostia môžu následne zdieľať média na stránke eventu.

1.3 EventCreate

Prvá možnosť, na ktorú som narazil, keď som sa vo vyhľadávači snažil nájsť niečo, čo by spĺňalo požadované kritéria, bola stránka *EventCreate*[1]. Služba ponúka vytvorenie akcií rôznych druhov. Pre akciu môže užívateľ vytvoriť šablónu, ktorú hostia uvidia. Užívateľ sa môže hostí opýtať na rôzne otázky, ktoré vytvorí a následne od nich získavať data. Veľmi zaujímavou funkciou je vyberanie vstupného na akciu, alebo predávanie produktov spojených s akciou. Služba ponúka rôzne programy pre užívateľov, od základného, ktorý je zadarmo, po drahšie, mesačne platené balíky. S narastajúcou cenou sa pri tom odomknú ďalšie funkcie. Hostia ale, po potvrdení účasti na akcii, nemôžu pridávať žiadne príspevky k akcii. Ďalšou nevýhodou je, že hostia po obdržaní pozvania sa na akciu registrujú sami, zadaním svojho e-mailu a mena. Poslednou nevýhodou je, že služba neponúka možnosť zdieľať odkaz na akciu v podobe QR kódu.

Nevýhody:

- Hostia musia sami vkladať svoje údaje, pri registrovaní na akciu, na ktorú sú pozvaní.
- Hostia nemôžu zdieľať vlastné príspevky v akcii.
- Služba neponúka zdieľanie odkazu pomocou QR kódu.

Výhody:

- Platené programy. Je to skôr výhoda pre službu ako pre užívateľa. Užívateľ si vyberá rôzne programy, podľa ktorých sa mu odomknú nové možnosti (napríklad viac možností na úpravu pozvánky, viac štatistík).
- Vyberanie vstupného a predaj predmetov spojených s akciou.

[1] https://www.eventcreate.com/

- Rozsiahly zber dát a prehľadné štatistiky o užívateľoch.

1.4 RSVPify

RSVPify[2] je webová služba, rovnako zameraná na organizáciu akcií. Služba je zameraná viac na komerčné podniky ako menšie akcie. Užívateľ môže vytvoriť akciu a z množstva šablón pre ňu vybrať vzhľad. Môže pozvať hostí pomocou e-mailu, odkazu, alebo QR kódu. Hostia sa po otvorení odkazu takisto musia na akciu registrovať sami. Užívateľ môže pre hostí vytvoriť formulár, z ktorého o nich následne získava informácie.

Nevýhody:

- Hostia musia sami vkladať svoje údaje pri registrovaní na akciu, na ktorú sú pozvaní.
- Hostia nemôžu zdieľať vlastné príspevky k akcii.

Výhody:

- Platené programy. Je to skôr výhoda pre službu ako pre užívateľa. Užívateľ si vyberá rôzne programy, podľa ktorých sa mu odomknú nové možnosti (napríklad viac možností na úpravu pozvánky, viac štatistík).
- Vyberanie vstupného na akciu.
- Rozsiahly zber dát a prehľadné štatistiky o užívateľoch.
- Systém na usadenie hostí.
- Možnosť zdieľať odkaz na akciu pomocou QR kódu.

[2] https://rsvpify.com/

Kapitola 2

Špecifikácia požiadavkov

Táto kapitola sa venuje tvorbe funkčných a nefunkčných požiadavkov môjho systému. Požiadavky vytváram podľa cieľov bakalárskej práce a takisto na základe analýzy konkurencie z predchádzajúcej kapitoly.

2.1 Funkčné požiadavky

F1: Vytvorenie akcie

Užívateľ bude v systéme schopný vytvoriť novú akciu, ku ktorej priradí názov a dátum uskutočnenia.

F2: Vytvorenie a pridanie hostí

Užívateľ bude môcť vytvoriť hostí a priradiť hostí k akcii.

F3: Vytvorenie šablóny pre pozvánku

Užívateľ bude môcť zadefinovať základné údaje o šablóne pre pozvánku na akciu. Medzi tieto údaje patrí farba pozadia a textu, font textu zo sady predvolených fontov, a rozloženie obrazovky podľa vopred vytvorených rozložení. Nebude schopný vytvárať vlastné rozloženie komponent pozvánky, ani nebude schopný pridávať ľubovolný počet komponent.

F4: Vytvorenie otázky pre hostí

Užívateľ bude schopný vytvoriť otázku pre hostí. Otázka bude môcť byť s otvorenou odpoveďou, alebo s predvolenými možnosťami pre odpoveď, kde bude buď možné zvoliť len jednu odpoveď, alebo viacero odpovedí naraz. Užívateľ nebude schopný vytvoriť iný typ otázky.

F5: Odoslanie pozvánok

Užívateľ bude mať možnosť odoslať všetky pozvánky cez e-mail všetkým hosťom, alebo bude mať možnosť získať odkaz na pozvánku pre každého hosťa zvlášť, ak nebude chcieť využiť odoslanie cez e-mail. Užívateľ nebude môcť vybrať podskupinu hostí, ktorým chce pozvánku odoslať, ani nebude môcť odoslať pozvánku iným spôsobom ako cez e-mail v rámci systému.

F6: QR kód s pozvánkou

Užívateľ bude môcť vygenerovať QR kódy s pozvánkami pre všetkých hostí naraz, alebo bude môcť generovať QR kódy jednotlivo. QR kód bude môcť upraviť tým spôsobom, že mu zadá farbu pozadia a farbu popredia, môže zmeniť spôsob vykresľovania jednotlivých štvorcov na predom definované možnosti. Nebude môcť vložiť vlastný obrázok do QR kódu a ani nebude môcť zadať obsah kódu sám.

F7: Autentifikácia hosťa pomocou linku

Hosť sa bude môcť prihlásiť k svojej pozvánke pomocou linku ktorý obdrží od organizátora akcie formou QR kódu, alebo linkom. Hosť nebude mať možnosť prihlásiť sa iným spôsobom.

F8: Zdieľanie médií na akcii v pozvánke

Hosť bude mať možnosť v svojej pozvánke zdieľať média vo forme odkazov a správ s ostatnými hosťami a organizátorom. Hosť bude schopný nahrávať len obsah v podobe textu.

F9: Odpovedanie na otázky

Hosť bude schopný cez svoju pozvánku odpovedať na otázky vytvorené organizátorom.

F10: Registrácia užívateľa

Užívateľ, ktorý bude chcieť vytvárať akcie sa bude môcť na začiatku registrovať.

F11: Prihlásenie užívateľa

Po registrácii sa užívateľ bude môcť prihlásiť k svojmu účtu a v ňom bude mať prístup k jemu vytvoreným akciám.

2.2 Nefunkčné požiadavky

N1: Dostupnosť cez internet

Aplikácia bude dostupná po pripojení na internet z rôznych zariadení. Užívateľ bude mať zo všetkých zariadení prístup k svojim údajom.

N2: Dostupnosť na rôznych zariadeniach

Aplikácia bude responzívna, aby umožňovala používanie aj na mobilných telefónoch a na tabletoch.

N3: Šifrované ukladanie citlivých údajov

Citlivé údaje (napr. heslá) sa budú pred uložením šifrovať.

Kapitola 3
Návrh

V tejto kapitole sa budem zaoberať návrhom systému na základe funkčných a nefunkčných požiadavkov. Kapitola sa delí na dve časti. V prvej časti navrhujem funkčnú časť aplikácie a v druhej časti sa venujem návrhu technickej stránky projektu.

3.1 Funkčný návrh

V tejto sekcii najprv popíšem doménu systému a jej jednotlivé prvky. Z nich odvodím konceptuálny model domény. Ďalej píšem o typoch užívateľov systému. Súčasťou sekcie je aj aktivity diagram a jeho popis, analýza použitia systému a na záver papierové modely pre užívateľské rozhranie systému.

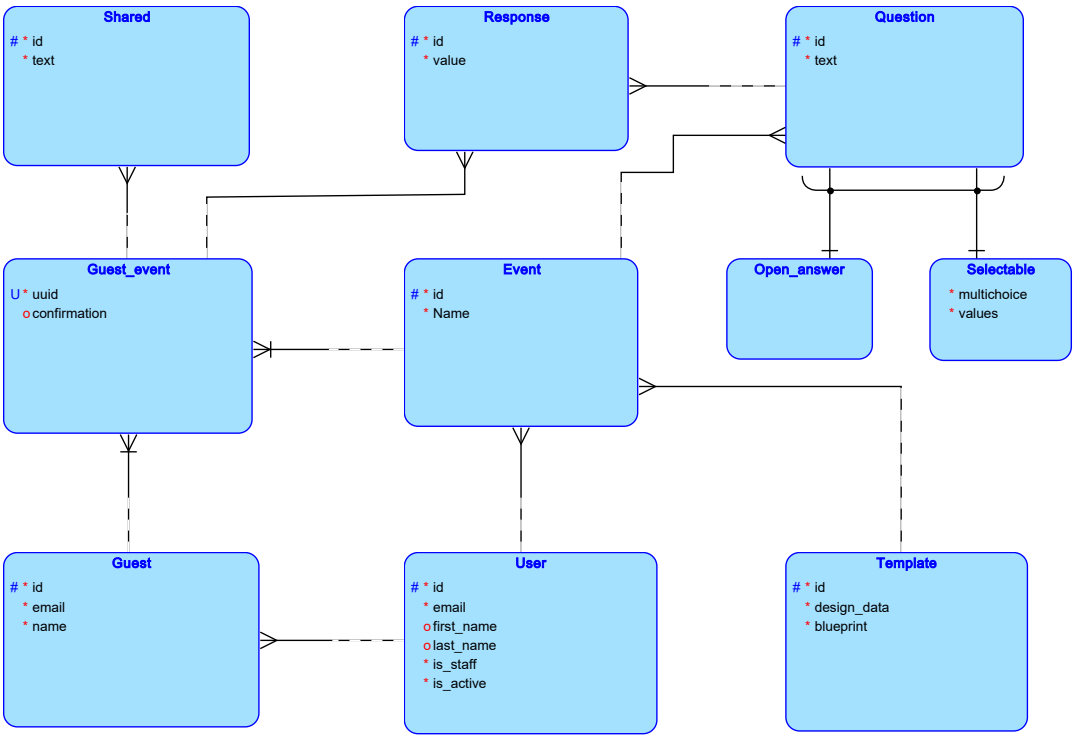
3.1.1 Popis domény

V systéme vystupuje viacero entít, ktoré som zadefinoval následovne:

- *Organizátor* (`User`) je základným užívateľom systému.
- *Hosť* (`Guest`) je užívateľ vytvorený organizátorom.
- *Pozvánka* na spoločenskú akciu odoslaná organizátorom hosťovi. Predstavuje spojenie medzi hosťom a akciou, keďže tieto dve entity môžu existovať nezávisle na sebe.
- *Akcia* (`Event`) predstavuje spoločenskú akciu vytvorenú organizátorom.
- *Vzor pozvánky* (`Template`) obsahuje vzhľad pozvánky, ktorý organizátor upraví a ktorý sa zobrazí hosťovi po otvorení pozvánky.
- *Otázka* (`Question`), ktorú vytvára organizátor a hostia na ňu neskôr odpovedajú.
- *Odpoveď* (`Response`) hosťa na otázku vytvorenú organizátorom.
- *Zdieľaný obsah* (`Shared`) je obsah, ktorý hostia zdieľajú v rámci akcie. Môže to byť link na rôzne média, alebo len obyčajný text.

3.1.2 Konceptuálny model

Z rozboru domény v predchádzajúcej sekcii som vytvoril konceptuálny model, ktorý môžeme vidieť na nasledujúcom obrázku:



Obr. 3.1 Konceptuálny model

Entita `User` predstavuje základného užívateľa aplikácie, teda organizátora. Organizátor môže vytvárať Akcie (`Event`) a pozývať hostí (`Guest`). Každá akcia má priradenú šablónu pozvánky (`Template`), ktorá má voľnejšiu podobu. Má len atribút `blueprint`, ktorý označuje šablónu ako základ pre iné šablóny a atribút `design_data`, ktorý bude obsahovať informácie o šablóne, podľa toho ako si ich klient databázy nadefinuje.

Za bližšiu pozornosť v modeli stojí dekompozícia vzťahu medzi akciou (`Event`) a hosťom (`Guest`), z čoho vznikla entita `Guest_event`. Táto entita nesie informáciu o tom či hosť prijal, alebo odmietol pozvánku na akciu, na ktorú bol pozvaný a taktiež je vhodná na spojenie s entitou `Response`, kde hosť odpovedá na otázky vytvorené hostiteľom a entitou `Media`, ktorá predstavuje zdieľaný obsah v rámci akcie. Obsahuje aj atribút `uuid`, ktorý predstavuje kľúč, pomocou ktorého sa hosť prihlasuje k svojej pozvánke. Viacej sa mu venujem v časti 4.2.4.

Entita `Question` predstavuje otázku, ktorú organizátor akcie vytvorí a bude v pozvánke zobrazená hosťom, aby na ňu odpovedal. Delí sa pomocou relácie XOR na otázku s voľnou odpoveďou (`Open_answer`) a na otázku s odpoveďami preddefinovanými organizátorom akcie (`Selectable`). Druhá možnosť otázky má okrem preddefinovaných odpovedí (`values`) aj možnosť označiť otázku (`multichoice`), či sa na ňu dá odpovedať viacerými možnosťami, alebo iba jednou.

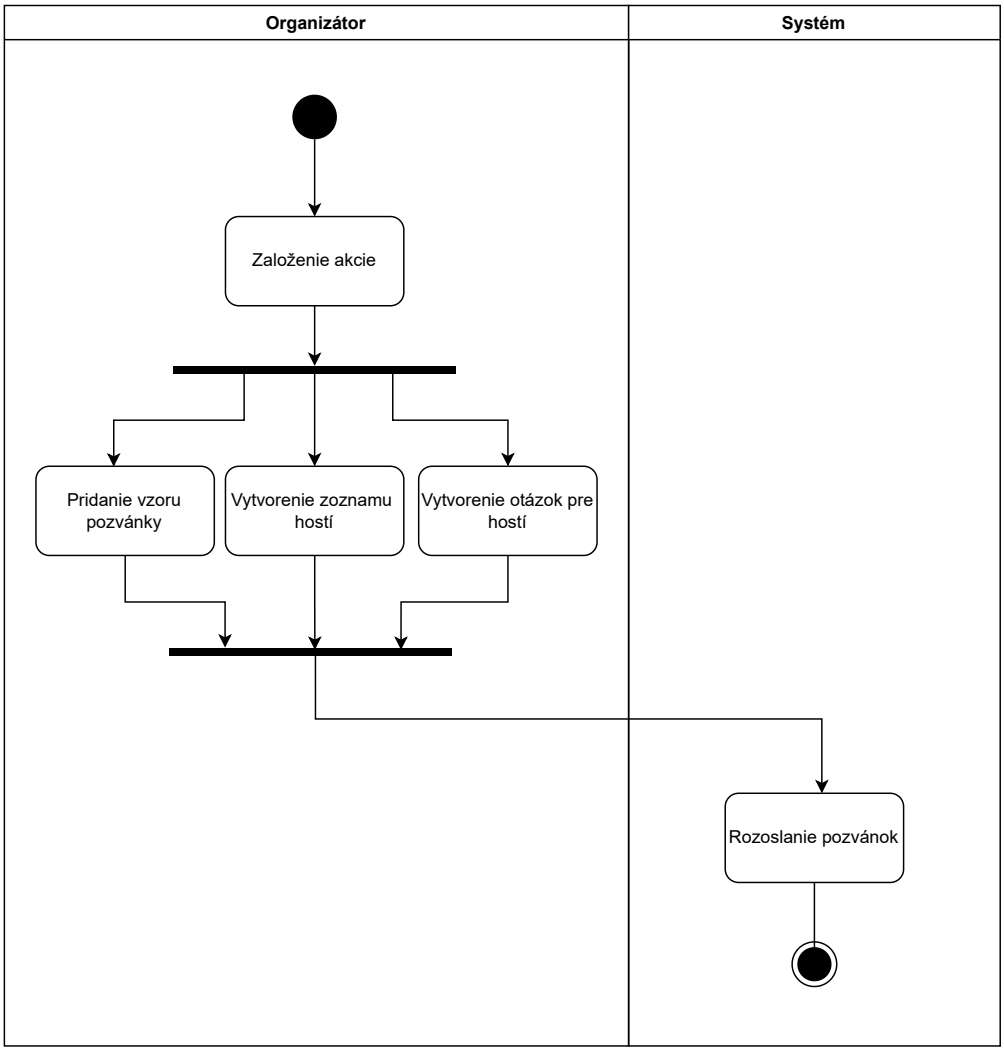
3.1.3 Typy užívateľov

Systém bude mať dva druhy užívateľov – *organizátora akcie* a *hosťa*. *Organizátor akcie* je základný užívateľ, ktorý sa do systému registruje sám a má možnosť vytvárať akcie, na ktoré

môže pozývať hostí. *Hosťa* však najprv vytvára *organizátor*, čiže hosť sa do systému neregistruje sám. *Hosť* má možnosť potvrdiť, respektíve zamietnuť svoju účasť na akcii, odpovedať na otázky vytvorené organizátorom a zdieľať obsah v rámci akcie.

3.1.4 Proces vytvárania akcie

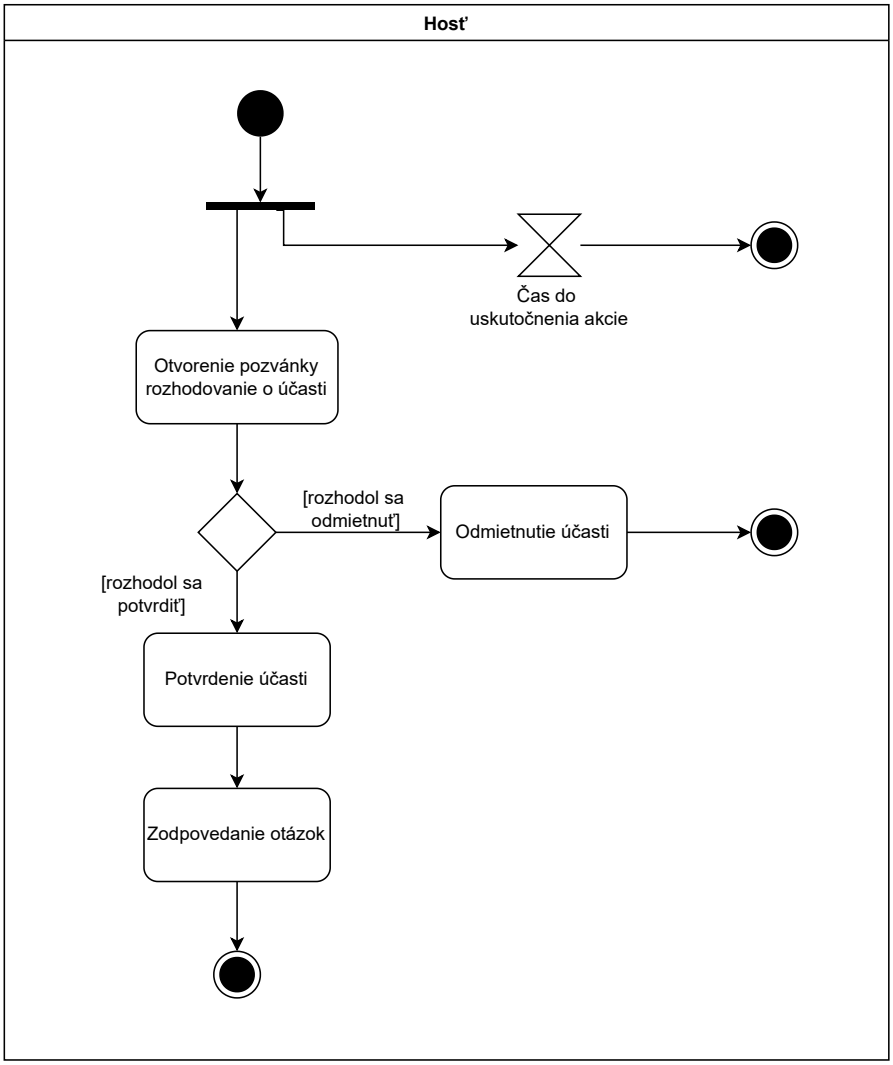
Spomínaný proces organizátora pri vytváraní akcie a pozývaní hostí zachytáva diagram aktivít na obrázku 3.2. Tento proces som vybral a znázornil preto, lebo zo všetkých procesov je najpodstatnejší a hlavný pre fungovanie aplikácie.



Obr. 3.2 Aktivity diagram procesu vytvárania akcie

3.1.5 Proces spracovania pozvánky hosťom

Druhý dôležitý proces vykonáva hosť akcie, ku ktorej dostal pozvanie. Je znázornený na obrázku 3.3.



Obr. 3.3 Aktivity diagram procesu spracovania pozvánky hosťom

3.1.6 Analýza použitia

V rámci analýzy použitia som vytvoril scenáre, ktoré ukazujú predpokladané použitie systému.

3.1.6.1 Registrácia užívateľa

- Aktéri
  - Neregistrovaný užívateľ – organizátor
- Podmienky pre dokončenie

- Užívateľ je úspešne zaregistrovaný
- Scenár
  - Užívateľ zadá nové autentizačné údaje
  - Systém skontroluje údaje (tvar emailu, bezpečnosť hesla)
  - Užívateľ je zaregistrovaný

3.1.6.2 Prihlásenie užívateľa

- Aktéri
  - Užívateľ – organizátor
- Podmienky pre spustenie
  - Užívateľ je registrovaný
- Podmienky pre dokončenie
  - Užívateľ je prihlásený a pustený do systému
- Scenár
  - Užívateľ zadá prihlasovacie údaje
  - Systém skontroluje správnosť údajov
  - Užívateľ je vpustený do systému

3.1.6.3 Vytvorenie akcie

- Aktéri
  - Užívateľ – organizátor
- Podmienky pre spustenie
  - Užívateľ je prihlásený
- Podmienky pre dokončenie
  - Nová akcia je vytvorená
- Scenár
  - Užívateľ zadá meno akcie a dátum uskutočnenia akcie
  - Systém akciu vytvorí a priradí ju užívateľovi

3.1.6.4 Pridanie hosťa k akcii

- Aktéri
  - Užívateľ – organizátor
- Podmienky pre spustenie
  - Užívateľ je prihlásený a má vytvorenú akciu, ku ktorej chce hosťa pridať
- Podmienky pre dokončenie

- Hosť je priradený k akcii

- Scenár (hosť neexistuje)
  - Užívateľ vyberie akciu, ku ktorej chce nového hosťa priradiť
  - Užívateľ zadá e-mail a meno hosťa
  - Systém hosťa vytvorí
  - Systém hosťa priradí ku akcii

- Alternatívny scenár (hosť existuje)
  - Užívateľ vyberie akciu, ku ktorej chce nového hosťa priradiť
  - Užívateľ priradí hosťa zo zoznamu existujúcich hostí
  - Systém hosťa priradí ku akcii

3.1.6.5 Vytvorenie pozvánky

- Aktéri
  - Užívateľ – organizátor

- Podmienky pre spustenie
  - Užívateľ je prihlásený a má vytvorenú akciu, ku ktorej chce pridať pozvánku

- Podmienky pre dokončenie
  - Pozvánka je vytvorená

- Scenár
  - Užívateľ zvolí akciu, ku ktorej chce pozvánku vytvoriť
  - Užívateľ pozvánku vytvorí a uloží

3.1.6.6 Vytvorenie otázky pre hostí

- Aktéri
  - Užívateľ – organizátor

- Podmienky pre spustenie
  - Užívateľ je prihlásený a má vytvorenú akciu, pre ktorú chce otázku vytvoriť

- Podmienky pre dokončenie
  - Otázka je vytvorená a priradená k akcii

- Scenár
  - Užívateľ vyberie akciu, ku ktorej chce otázku priradiť
  - Užívateľ zadá text otázky
  - Užívateľ zvolí, či sa jedná o otázku s voľnou odpoveďou, alebo o otázku s odpoveďou predvolenou
  - Ak zvolil otázku s predvolenou odpoveďou, tak vyberie, či môže mať viac odpovedí alebo jednu, inak pokračuje
  - Ak zvolil otázku s predvolenou odpoveďou, tak zadá možné odpovede, inak pokračuje
  - Systém otázku priradí k akcii

3.1.6.7 Odoslanie pozvánok

- Aktéri
  - Užívateľ – organizátor
- Podmienky pre spustenie
  - Užívateľ je prihlásený, má vytvorenú akciu, vytvorenú pozvánku pre a priradených hostí k akcii
- Podmienky pre dokončenie
  - Hostia obdržia pozvanie
- Scenár
  - Užívateľ zvolí akciu, pre ktorú chce odoslať pozvánky
  - Užívateľ zvolí odoslanie pozvánok
  - Systém pozvánky odošle

3.1.6.8 Vygenerovanie QR s odkazom na pozvánku pre všetkých hostí

- Aktéri
  - Užívateľ – organizátor
- Podmienky pre spustenie
  - Užívateľ je prihlásený, má vytvorenú akciu, vytvorenú pozvánku pre a priradených hostí k akcii
- Podmienky pre dokončenie
  - Užívateľ má QR kódy s odkazmi na pozvánky hostí
- Scenár
  - Užívateľ zvolí akciu, pre ktorú chce vygenerovať QR kódy
  - Užívateľ zvolí vygenerovanie QR kódov
  - Systém vygeneruje QR kódy
  - Užívateľ si kódy stiahne

3.1.6.9 Potvrdenie/odmietnutie pozvania

- Aktéri
  - Užívateľ – Hosť
- Podmienky pre spustenie
  - Hosť obdržal pozvanie k akcii, ku ktorej je priradený, buď QR alebo link
- Podmienky pre dokončenie
  - Hosť potvrdí alebo odmietne pozvanie
- Scenár (potvrdenie účasti)

- Hosť otvorí obdržaný odkaz
- Hosť potvrdí svoju účasť na akcii
- Systém odpoveď uloží

- Alternatívny scenár (zamietnutie účasti)
  - Hosť otvorí obdržaný odkaz
  - Hosť zamietne svoju účasť na akcii
  - Systém odpoveď uloží

3.1.6.10 Odpovedanie na otázky od organizátora akcie

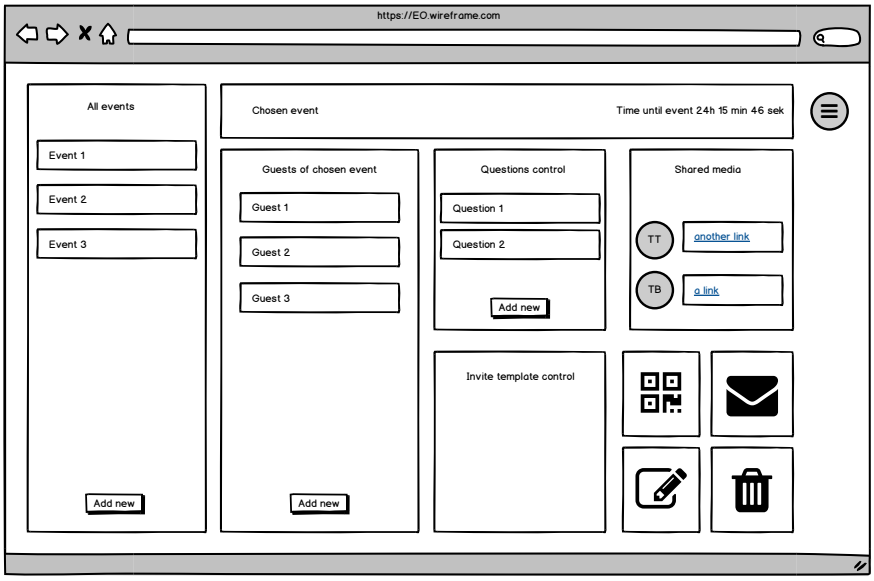
- Aktéri
  - Užívateľ – Hosť
- Podmienky pre spustenie
  - Hosť obdržal pozvanie k akcii, ku ktorej je priradený, buď QR alebo link a potvrdil svoju účasť
- Podmienky pre dokončenie
  - Hosť odpovie na otázky
- Scenár
  - Hosť otvorí obdržaný odkaz
  - Hosť vyplní odpovede na otázky
  - Hosť odošle odpovede
  - Systém odpovede uloží

3.1.6.11 Zdieľanie obsahu o akcii

- Aktéri
  - Užívateľ – Hosť
- Podmienky pre spustenie
  - Hosť obdržal pozvanie k akcii, ku ktorej je priradený, a potvrdil pozvanie
- Podmienky pre dokončenie
  - Zdieľaný obsah sa uloží a priradí ku akcii
- Scenár
  - Hosť otvorí obdržaný odkaz
  - Hosť vyplní pole, kde môže zadať zdieľaný obsah
  - Hosť odošle zdieľaný obsah
  - Systém uloží zdieľaný obsah

3.1.7 Papierové modely

Papierové modely užívateľského rozhrania (alebo *wireframes*) som vytváral v softwari *Balsamic*[1]. Modelov vzniklo viac, tu ukážem len hlavné dva pohľady dvoch užívateľov systému, teda pohľad organizátora a pohľad hosťa.

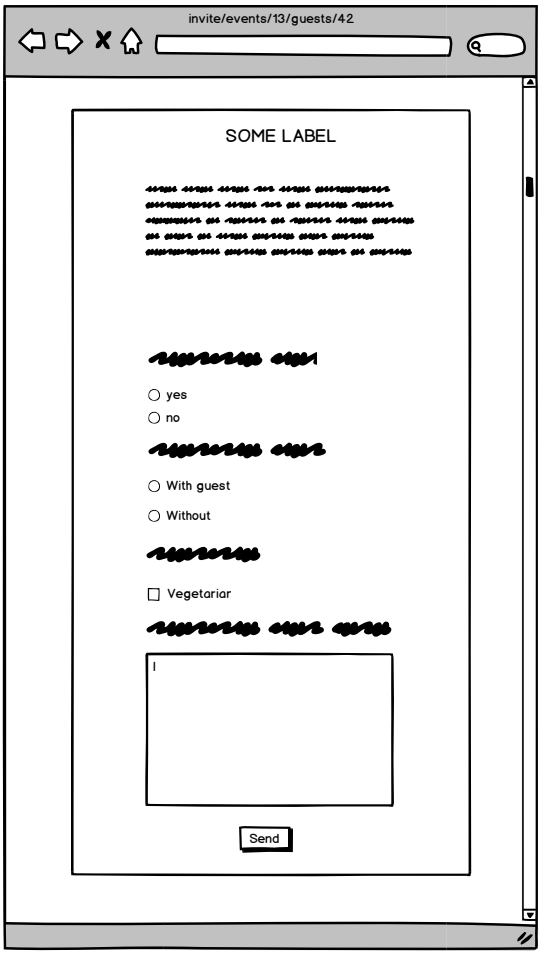


Obr. 3.4 Pohľad organizátora akcie na stránku s akciami

V pohľade organizátora na obrázku 3.4 je zobrazená stránka so štýlom *dashboard*, ktorá predstavuje hlavný kontrolný panel, z ktorého organizátor riadi svoje akcie. Na ľavej strane je zoznam vytvorených akcií. Postupne ako sa akcie budú pridávať, tak sa daná bunka bude dať posúvať pomocou vlastného posuvníku. Bunka, ktorá je umiestnená najvyššie, slúži na zobrazenie informácií o aktuálne upravovanej akcii. Prvá bunka z ľavej strany pod ňou obsahuje zoznam pozvaných hostí. Zoznam sa bude taktiež rolovať pomocou posuvníku, ak sa naplní. Na jednotlivých hostí by sa malo dať kliknúť a tým si zobraziť podrobnejšie informácie o hosťovi. Vrchná bunka na pravo od bunky s hosťami je bunka na vytváranie a spravovanie otázok pre hostí. Na pravo od nej je bunka s médiami, ktoré hostia v rámci akcie zdieľajú. Pod bunkou s otázkami sa nachádza bunka zodpovedná za editovanie pozvánky a na pravo od nej je skupina buniek s funkciami generovanie QR kódov, odosielanie pozvánok, úpravu akcie a odstránenie akcie. Na pravej strane je tlačidlo, ktoré otvorí menšie menu, v ktorom užívateľ spravuje svoj účet a odhlasuje sa z aplikácie.

[1] https://balsamiq.com/

Na obrázku 3.5 je pohľad hosťa na pozvánku, ktorú otvoril pomocou odkazu, ktorý s ním organizátor zdieľal. Na vrchnej časti sú informácie o akcii zostavené organizátorom danej akcie. Pod touto častou sú otázky organizátora a na spodnej časti stránky je sekcia, v ktorej budú môcť hostia zdieľať obsah.



Obr. 3.5 Pohľad hosťa na pozvánku

3.2 Technický návrh

V tejto sekcii sa zaoberám technickým návrhom systému. Najprv zvolím vhodnú platformu, na ktorej bude aplikácia fungovať a následne vzhľadom na zvolenú platformu zvolím moderné technológie pre implementáciu.

3.2.1 Forma systému

Pri výbere formy systému je potrebné zohľadniť viaceré kritéria. Systém by sa mal dostať k čo najširšiemu množstvu užívateľov a mal by byť ľahko dostupný. Z toho vyplýva, že systém by mal byť dostupný z viacerých platforiem. Na docielenie tohoto je k dispozícii viacero foriem, v akých by mohol systém vzniknúť. Tieto spôsoby dopodrobna rozoberiem.

3.2.1.1 Mobilná aplikácia

Mobilná aplikácia sa inštaluje do smartphonu. Typicky je dostupná cez aplikačné obchody ako napríklad *App Store*, alebo *Google Play*. Často využíva funkcie zabudované v smarthphonoch ako napríklad GPS, odosielanie správ, nahrávanie videa a zvuku. Takáto aplikácia môže byť vyvinutá rôznymi spôsobmi. Hlavné rozdelenie definuje aplikáciu ako natívnu a hybridnú. Natívna aplikácia poskytuje lepší prístup k funkciám smartphonu ako hybridná aplikácia. Je písaná v natívnom jazyku zariadenia. To však znamená, že rovnaká aplikácia pre dva operačné systémy, napríklad Android a iOS, musí byť prakticky implementovaná dvakrát, vždy v jazyku daného operačného systému. Na druhú stranu, hybridná aplikácia je písaná v alternatívnom jazyku alebo frameworku a pre viacero operačných systémov stačí jeden zdrojový kód. Mobilné aplikácie typicky nefungujú na počítačoch a tak sú často sprevádzané buď desktopovou aplikáciou, alebo webovou aplikáciou.

3.2.1.2 Desktopová aplikácia

Desktopová aplikácia sa inštaluje do počítača užívateľa. To je v prípade môjho systému nevýhoda, pretože inštalácia môže veľa užívateľov odradiť. Desktopová aplikácia sa často využíva, keď aplikácia nepotrebuje prístup na internet vôbec, alebo dokáže niektoré funkcie vykonávať bez internetu. To však môj systém nemusí splňovať. Desktopové aplikácie majú rovnako ako mobilné aplikácie rôzne nároky na vývoj podľa operačného systému, pre ktorý je aplikácia určená. To by mohlo komplikovať vývoj.

3.2.1.3 Webová aplikácia

Webová aplikácia je dostupná cez internetový prehliadač a typicky nie je inštalovaná do zariadenia užívateľa. Je teda dostupná skoro z akéhokoľvek operačného systému. Ďalšou výhodou webovej aplikácie je, že užívateľ ju nemusí aktualizovať. Nevýhodou webovej aplikácie je, že nie je dostupná bez internetového pripojenia. [1]

3.2.1.4 Zvolená forma systému

Ako najvhodnejšiu formu systému som zvolil webovú aplikáciu z viacerých dôvodov. Webová aplikácia bude dostupná z akéhokoľvek systému s internetovým prehliadačom, čím sa splní nefunkčný požiadavok N1. Ďalší dôvod je, že väčšina ľudí spoločenské akcie neorganizuje pravidelne a keby aplikáciu museli inštalovať, tak by ich to mohlo odradiť, alebo by ju po použití nemali nutnosť ďalej uchovávať v svojom zariadení a teda by ju odinštalovali. Aplikácia nebude vyžadovať žiadne funkcie konkrétneho zariadenia a pre jej chod je internetové pripojenie potrebné.

3.2.2 Technológie

V nasledujúcej sekcii sa budem venovať výberu vhodnej technológie na implementáciu aplikácie.

Typická webová aplikácia pozostáva z frontendovej časti, tiež nazývanej klientskej časti, s ktorou interaguje užívateľ a backendovej alebo serverovej časti, s ktorou komunikuje frontendová časť a bežný užívateľ ju nevidí. Serverová časť sa stará o ukladanie a spracovanie dát, ktoré predáva klientskej časti aplikácie. Tieto dve časti môžu byť úzko spojené, alebo sa môžu vyvíjať nezávisle od seba. Obidva prístupy majú svoje výhody a nevýhody.

Prvý spôsob sa nazýva *Multi-Page aplikácia*. Vždy keď užívateľ vykoná zmenu, tak pošle na server požiadavok. Serverová časť požiadavok spracuje a ako odpoveď posiela klientovi HTML kód s konkrétnymi datami. Druhý spôsob – *Single-Page aplikácia* oddeľuje klientskú a serverovú časť tak, že klientská časť sa renderuje priamo v browseri klienta. Serverová časť posiela klientovi len dáta, o ktoré si klient zažiada a potom ich sama spracúva a vyzualizuje. Tým sa klienstská časť nemusí aktualizovať celá pri každom požiadavku a namiesto toho prekreslí len časť, v ktorej sa

zmenili dáta. V dnešnej dobe, vo väčších tímoch a na väčších projektoch sa podľa môjho názoru viac využíva druhý spôsob. Výhody tohoto postupu sú, že aplikácia je ľahšie modulovateľná a rozšíriteľná a na oboch častiach sa dá pracovať paralelne, takže vývoj a nasadenie je rýchlejšie. Ďalšou výhodou je, že technológie oboch častí sa rýchlo vyvíjajú a komplikujú, tým pádom je ťažké byť expertom v oboch častiach a obidve časti využiť naplno a rozdelenie aplikácie na dve časti s dvoma tímami expertov tento problém rieši.

3.2.2.1 Komunikácia mezdi serverom a klientom – REST

Pri tomto postupe klientská časť komunikuje so serverom za pomocou API. API je sada definícií a protokolov, ktoré slúžia ako kontrakt medzi poskytovateľom informácií a ich používateľom. Na implementáciu API použijem architektonický štýl, ktorý sa nazýva REST API, alebo RESTful API.

Ako sa uvádza v [2] a [3], aby bolo API považované za RESTful musí spĺňovať nasledujúce kritéria:

- Klient-server architektúra, ktorá pozostáva z klienta, serveru, zdrojov, a požiadavkov spracovanými cez HTTP.
- Bezstavová komunikácia medzi klientom a serverom. Žiadna informácia o klientovi nie je uchovávaná serverom a jednotlivé požiadavky sú oddelené a nezávislé. Každý požiadavok musí obsahovať všetky informácie potrebné k jeho vykonaniu.
- Zdroje v komunikácii musia byť označené ako cacheovateľné, alebo necacheovateľné.
- Zdroje sú prenášané za pomoci reprezentácie, ktorá jé známa a spoločná pre klientskú a serverovú časť. Reprezentácie môžu mať rôznu podobu (JSON, HTML, XML, a.t.ď), z ktorých je najpoužívanejší JSON (*Javascript Object Notation*), ktorý je aj napriek jeho názvu jazykovo agnostický a čitateľný pre ľudí aj počítače. Klient nepracuje priamo zo zdrojmi, ale s ich reprezentáciou.
- Vrstevnatosť systému, ktorá sa prejavuje tým, že klient za normálnych okolností nevie, či komunikuje priamo s koncovým serverom, alebo sprostredkovateľom medzi nimi. Ak sa medzi klienta a server vloží napríklad proxy, neovplyvní to ich komunikáciu a ani jedna strana nebude potrebovať zmeny v implementácii.
- Voliteľný požiadavok Code-On-Demand – funkcionalita klienta môže byť rozšírená o kód, ktorý server zašle klientovi.

Klentská časť aplikácie bude posielať požiadavky na server, ktorý späť posiela odpoveď. Dáta, alebo zdroje budú prenášané vo formáte JSON. Budú sa prenášať pomocou HTTP protokolu, ktorý obsahuje štyri metódy: GET, POST, PUT a DELETE. Tieto metódy sa používajú na definovanie CRUD operácií a vysvetľujú ako má server so zdrojmi pracovať. [2]

3.2.2.2 Backend

Na vytvorenie backendovej aplikácie, ktorá bude poskytovať REST API existuje veľa frameworkov. Ja budem používať Django, najmä kvôli mojim predošlým skúsenostiam s ním. Django je open source webový framework pre jazyk Python. Vydaný v roku 2005 [4], v tej dobe používaný na tvorbu multipage webových aplikácií, sa dnes vďaka takzvaného prístupu *Api-first*, ktorý rozdelil webové aplikácie na serverovú a klientskú časť, stal využívaným nástrojom na spravovanie databázy a tvorbu API. V knihe [5] autor píše: *„Vo veľkých spoločnostiach sa dnes Django používa častejšie len ako back-end API a nie ako úplné monolitické webové riešenie.“*. Konkrétne na vytvorenie REST API budem používať Django REST framework. Django REST framework je nástroj postavený na Django web frameworku, ktorý uľahčuje písanie REST rozhrania. *„Je vyspelý, plný funkcií, prispôsobiteľný, testovateľný a mimoriadne dobre zdokumentovaný.“* [5]

3.2.2.3 Frontend

Frontend aplikácie bude napísaný v jazyku Typescript. Typescript je nadstavba nad jazykom JavaScript, ktorá rozširuje jazyk o statické typovanie a ďalšie výhody známe z iných objektovo orientovaných jazykov. Kód písaný v Typescripte sa pred spustením kompiluje do JavaScriptu. To umožňuje odhaliť chyby v kóde, ktoré by sa bez Typescriptu objavili až za behu programu. Ďalšou výhodou Typescriptu je užšia integrácia s editorom. To ponúka napríklad lepšie možnosti našeptávania v editore a zvýrazňovanie chýb. Vďaka tomu je písanie kódu plynulejšie.

Single-Page aplikácia sa dá v JavaScripte respektíve Typescripte vytvoriť za pomoci mnohých knihovien a frameworkov: Angular, Vue, React, Next a mnoho ďalších. Všetky ponúkajú nástroje na vytvorenie Single-Page aplikácie a prívetivého užívateľského rozhrania. Ja som sa rozhodol použiť React, kvôli mojim predošlým skúsenostiam s ním. React je open-source knižnica zameraná na budovanie užívateľského rozhrania. Bol vyvinutý spoločnosťou *Facebook* (dnes *Meta*) a vydaný v roku 2013. Má veľkú komunitu a dobre spracovanú dokumentáciu. Na budovanie užívateľského rozhrania sa používajú komponenty. Komponenty reprezentujú logické, znovupoužiteľné časti užívateľského rozhrania. Interaktívna webová aplikácia musí aktualizovať DOM vždy keď nastane zmena. Tento proces môže byť pomalý a náročný. React tento problém rieši za použitia virtuálnej DOM, ktorá umožňuje aktualizovať len komponenty, ktorých sa zmena týka. To zvyšuje rýchlosť a priehľadnosť aplikácie. [6]

Ďalšou možnosťou bolo použiť framework Next (alebo NextJS), ktorý je postavený na Reacte. Obohacuje React o mnohé veci, ako napríklad Server side rendering, alebo zabudovaný routing. Aj napriek všetkým výhodám, ktoré Next ponúka, som sa rozhodol použiť React. Moja aplikácia bude pracovať s rôznymi užívateľskými vstupmi a datami, ktoré sa budú často meniť. Podľa článku [7] sa práve na takúto aplikáciu hodí viac React, pretože by aplikácia dostatočne nevyužila výhody ktoré Next prináša.

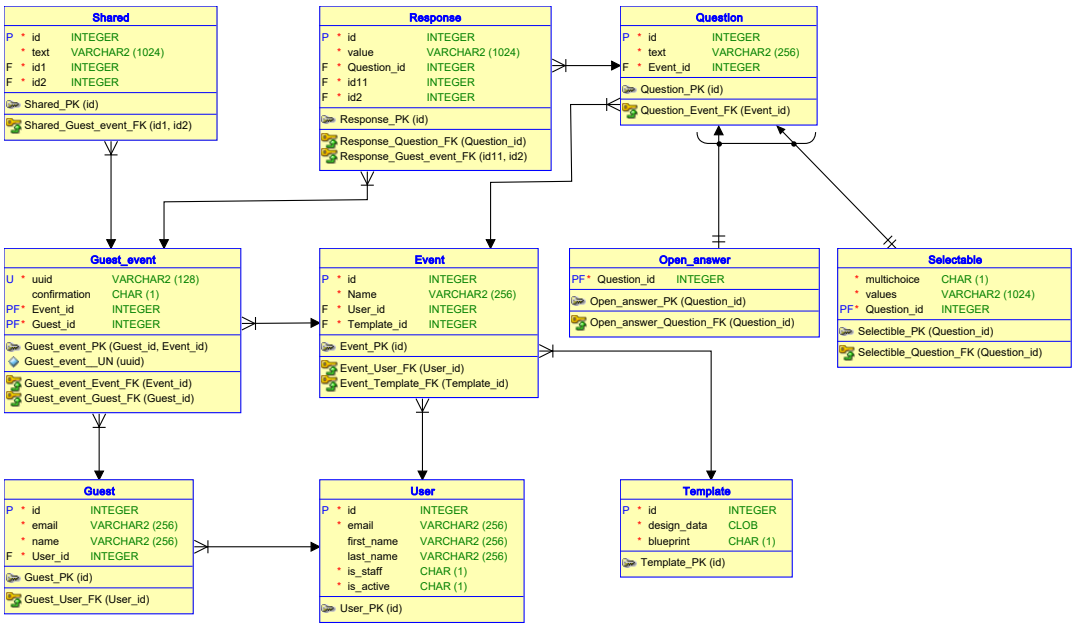
Kapitola 4

Implementácia

Táto kapitola sa zameriava na implementovanie aplikácie, ktorú som v minulej kapitole navrhol s použitím zvolených technológií. V kapitole spomeniem návrhové a architektonické vzory, ktoré som počas vývoja využil.

4.1 Návrh databázy

Model databázy aplikácie vychádza z konceptuálneho modelu (obrázok 3.1) a počas vývoja prešiel viacerými zmenami. Výsledný model vyzerá nasledovne:



Obr. 4.1 Relačný model

4.2 Implementácia serverovej časti aplikácie

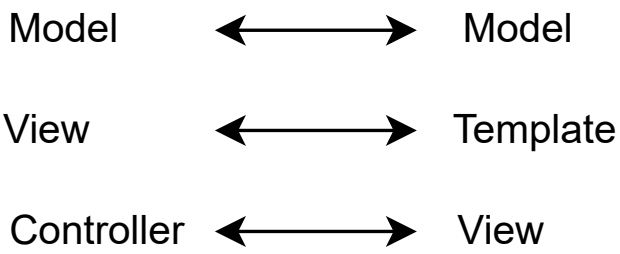
Táto sekcia sa bude venovať technológiám a postupom, ktoré som použil pri písaní serverovej časti aplikácie. Výsledkom tejto časti práce je funkčná serverová časť aplikácie, v podobe REST API, s 56 koncovými volaniami.

4.2.1 Virtuálne prostredie v Pythone

Virtuálne prostredie je nástroj, ktorý nám pomáha spravovať závislosti projektu. Závislosti sa vďaka nemu neinštalujú globálne, ale pre každý projekt zvlášť. Medzi výhody, kvôli ktorým som virtuálne prostredie v mojej aplikácii používal, patrí napríklad to, že projekt aplikácie sa vďaka tomu stáva samostatný, bez závislostí mimo projektu a ľahko sa dá reprodukovať vďaka súboru, ktorý obsahuje zoznam závislostí. Existuje viacero nástrojov na správu virtuálneho prostredia pre jazyk Python. Ja som vyskúšal dva a to už v Pythone zabudovaný modul *vevn* a externý nástroj *Conda*. Z týchto dvoch nástrojov som nakoniec používal Condu, kvôli lepšej a jednoduhšej integrácií s textovým editorom *Visual Studio Code*, ktorý som používal.

4.2.2 Django MVT

Django narozdiel od iných webových frameworkov nepoužíva bežnú MVC architektúru, ale nahrádza ju MVT architektúrov. Takisto sa jedná o trojvrstvú architektúru, zloženú z dátovej, biznis, a prezenčnej vrstvy. Jednotlivé časti oboch architektúr, by sa k sebe dali prirovnať ako na obrázku 4.2. Najväčší rozdiel je v tom, že o veľkú časť funkcionality, o ktorú by sa obvykle staral Controller sa stará priamo Django. [8] [9]



Obr. 4.2 Porovnanie Django MVT architektúry a MVC architektúry

Model je datovou vrstvou architektúry. Je zodpovedná za štrukturovanie dát a manipuláciu s dátami. Používa na to vlastný ORM, ktorý umožňuje interagovať s databázou. Vo výpise kódu 4.1 je ukážka toho, ako definovanie modelu v kóde vyzerá.

Výpis kódu 4.1 Django Model

```
from django.db import models

class Event(models.Model):
    name = models.CharField(max_length=255)
    date = models.DateField()
    owner = models.ForeignKey(Account, on_delete=models.CASCADE)
    template = models.ForeignKey(
        Template, on_delete=models.SET_DEFAULT, null=False, default=1)
```

View je biznis vrstvou. Spracúva užívateľské požiadavky a vracia odpovede na ne. Slúži ako most medzi modelom a templatom (serializérom). Jeho použitie je znázornené vo výpise kódu 4.2.

Výpis kódu 4.2 Django View

```
from rest_framework import generics, permissions, mixins

class EventList(mixins.ListModelMixin,
                mixins.CreateModelMixin,
                generics.GenericAPIView):
        queryset = Event.objects.all()
        serializer_class = EventSerializer
        permission_classes = [permissions.IsAuthenticated]

        def get(self, request, *args, **kwargs):
            """ Get all the events """
            return self.list(request, *args, **kwargs)

        def post(self, request, *args, **kwargs):
            """ Create new event """
            return self.create(request, *args, **kwargs)

class EventDetail(mixins.RetrieveModelMixin,
                  mixins.UpdateModelMixin,
                  mixins.DestroyModelMixin,
                  generics.GenericAPIView):
        queryset = Event.objects.all()
        serializer_class = EventSerializer
        permission_classes = [
            permissions.IsAuthenticated & IsEventOwner]

        def get(self, request, *args, **kwargs):
            """ Get event with {id} """
            return self.retrieve(request, *args, **kwargs)

        def put(self, request, *args, **kwargs):
            """ Update event with {id} """
            return self.update(request, *args, **kwargs)

        def delete(self, request, *args, **kwargs):
            """ Delete event with {id} """
            return self.destroy(request, *args, **kwargs)
```

Template vrstva by sa dala prirovnať k View vrstve klasickej MVC architektúry. V obyčajnom Djangu používa vlastné Template API na vytvorenie HTML stránok, ktoré sa odosielajú v odpovedi užívateľovi. V Django REST Frameworku táto časť odpadá, pretože odpoveďou na požiadavku klienta nie je HTML stánka, ale JSON s dátami z modelu. Z časti túto úlohu plnia serializéry. Sú to objekty, ktoré príjmu model a spracujú ho do JSON podoby, alebo opačne z JSON do modelu.

Výpis kódu 4.3 Django Serializer

```
from rest_framework import serializers
from ..models import Event

class EventSerializer(serializers.ModelSerializer):
    owner = serializers.PrimaryKeyRelatedField(
        many=False, read_only=True)
    template = serializers.PrimaryKeyRelatedField(
        many=False, read_only=True)

    class Meta:
        model = Event
        fields = [
            "id",
            "name",
            "date",
            "owner",
            "template",
        ]
```

4.2.3 Autentifikácia a autorizácia užívateľa pomocou JWT

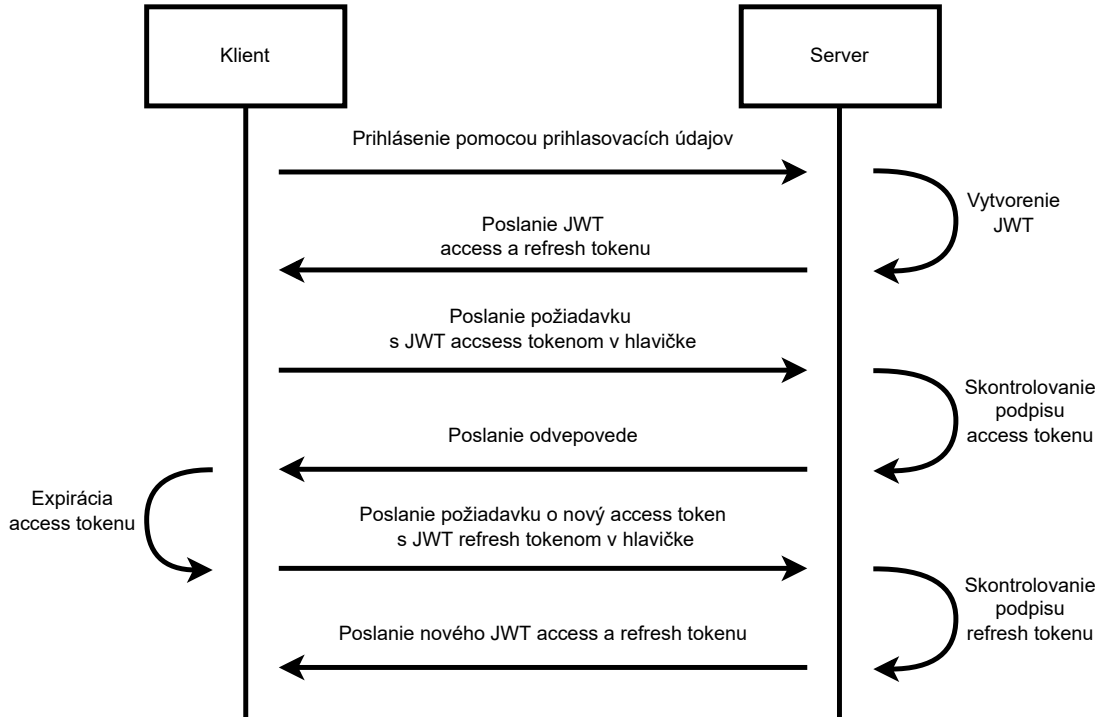
Aplikácia bude mať dva druhy užívateľov. Prvý z nich je organizátor akcií. Užívateľa pre prístup k jeho informáciám treba autentifikovať a keďže REST API je bezstavové, tak následne každý jeho požiadavok autorizovať. Django REST Framework ponúka v základe 4 druhy autentifikácie pre užívateľov:

- *Basic Authentification* využíva základnú autentifikáciu HTTP autentifikáciu a mala by sa využívať len vo fázy vývoja a testovania.
- *Token Authentification* používa jednoduchú autentizačnú schému založenú na generovaní Tokenu pre každého užívateľa. Tento token sa ukladá do databázy. Užívateľ po zadaní svojich prihlasovacích údajov obdrží token, ktorý posiela s každým požiadavkom v hlavičke požiadavku. Táto autentifikácia je vhodná pre klient-server aplikáciu.
- *Session Authentification* používa základnú autentifikáciu pôvodného Djanga. Je vhodná najmä pre AJAX klienta. Server vyžiada užívateľa, aby zadal svoje prihlasovacie údaje. Tie následne skontroluje a ak sú správne, server si uloží session id v pamäti a pošle ho späť klientovi. Od tej chvíle klient používa obdržanú session id, aby ho server spoznal.
- *Remote User Authentification* deleguje autentifikáciu na webový server, v ktorom aplikácia beží.

Prvé tri vyššie spomínané spôsoby autentifikácie užívateľa využívajú ukladanie údajov do databáze, čo môže zvýšiť priestorové nároky serveru. [10]

Existuje ďalšie veľmi využívané riešenie, ktoré som nakoniec použil v mojej aplikácii a nazýva sa *JSON Web Token*. Umožňuje presunúť ukladanie autentifikačných údajov na stranu klienta.

Autentizačné údaje sa predávajú serveru ako JSON objekt, ktorý sa skladá z hlavičky (header), dát (payload) a podpisu (signature). Vďaka podpisu môže byť informácia v posielanom JSON objekte overená a dôveryhodná, nie však šifrovaná. Podpis pozostáva z hlavičky a dát, ktoré sa šifrujú pomocou algoritmu uvedenom v hlavičke. Potom sa každá časť zašifruje pomocou `Base64URL` a na záver sa časti spoja. Proces autentifikácie začína rovnako, zaslaním prihlasovacích údajov užívateľa. Server potom vygeneruje JWT pre užívateľa za pomoci súkromného kľúča. Token sa pošle naspäť klientovi, kde sa väčšinou uchováva v lokálnom úložisku prehliadača. Pri ďalších požiadavkách sa token vkladá do hlavičky požiadavku. Server potom už len validuje podpis v tokene namiesto toho, aby ho hľadal v databázi [11]. Podľa štandardu *OAuth 2.0* server posiela klientovi dva tokeny. Ten, ktorý sa vkladá do hlavičiek požiadavkov sa nazýva *access token*. Má krátku životnosť a preto existuje druhý token, nazývaný *refresh token*, ktorý má dlhú životnosť a používa sa k obnoveniu access tokenu. Celý proces je znázornený na obrázku 4.3.



Obr. 4.3 Proces autentifikácie a autorizácie pomocou JWT

Na implementáciu JWT som použíl plugin pre Django REST Framework – *Simple JWT*.

4.2.4 Prihlasovanie hosťa pomocou UUID

Jedna z hlavných zmien, ktoré moja aplikácia ponúka oproti konkurenčným riešením je, že hostia sa na akciu nemusia registrovať sami po otvorení pozvánky, ale organizátor akcie ich zaregistruje sám a každému hosťovi sa vygeneruje unikátna pozvánka. Každému hosťovi sa pri vytvorení vygeneruje UUID.

UUID je 128-bitový reťazec hexadecimálnych číslic používaný na jedinečnú identifikáciu. V závislosti na použitom mechanizme je buď zaručená jeho jedinečnosť, alebo je zaručená veľmi veľká pravdepodobnosť, že bude rozdielny od všetkých vygenerovaných UUID do roku 3400. Python ponúka modul v štandardnej knižnici *uuid*, ktorý vie UUID generovať. Aby sa zaistila čo najväčšia jedinečnosť vygenerovaného reťazca, používa sa UUID verzie 1 alebo 4. Verzia 1 je mimo iné generovaná zo sieťovej adresy počítača, na ktorom je generovaná. To môže odhaliť

nechcené informácie. Verzia 4 generuje UUID náhodne, čiže šance na kolíziu sú väčšie, ale stále zanedbateľné. Ja používam verziu 4.

Vygenerované UUID bude použité na autentifikáciu a autorizáciu hosťa. Hosť obdrží pozvánku od organizátora v podobe linku, ktorý bude mať parameter obsahujúci UUID reťazec. Ten sa pri každom požiadavku na server vloží do hlavičky požiadavku. [12]

4.2.5 Generovanie QR kódu

Ďalší funkčný požiadavok bolo generovanie QR kódu, ktorý by obsahoval link na pozvánku hosťa. Na to som použil Python knižnicu `qrcode`. Táto knižnica dokáže generovať QR kódy a pritom ich aj upravovať zmenou farby, alebo tvarov blokov kódu. Proces vytvárania kódu bol nepriehľadný a obsahoval veľa parametrov, ktoré sa menili podľa vstupu užívateľa. To som vyriešil použitím návrhového vzoru Builder (Staviteľ) vytvorením classy `QRCodeBuilder`.

4.2.5.1 Návrhový vzor Builder

Builder (Staviteľ) je návrhový vzor, ktorý sa radí medzi vytváracie vzory. Využíva sa na vytvorenie komplexných objektov krok po kroku. Vzor umožňuje produkovať rôzne typy a reprezentácie výsledného objektu za použitia jedného konštrukčného rozhrania. Proces konštrukcie požadovaného objektu sa presunie do nového objektu - staviteľa. Proces sa v ňom rozdelí na viacero krokov, ktoré zadefinujú jednotlivé časti výsledného objektu. Dôležitou časťou je, že nie je potrebné vždy vykonať všetky kroky. Staviteľ by mal obsahovať metódu na vyžiadanie výsledného produktu, ktorý bol zostavený.

4.2.6 Dokumentácia API

Dôležitou časťou vývoja API je jeho dokumentácia. *Open API Specification* definuje jazykovo agnostické rozhranie, používané na dokumentáciu REST API, ktoré umožňuje ľuďom aj počítačom ľahko pochopiť spôsob používania API bez znalosti zdrojového kódu. Vďaka tomu je ďalej možné generovať kód pre klientskú alebo serverovú časť aplikácie. Nástroj ako Swagger vie ďalej túto dokumentáciu zobraziť v užívateľsky prívetivej forme. Na generovanie tejto dokumentácie som použil nástroj *drf-spectacular*. Je to nástroj špeciálne určený pre Django REST Framework. Značne mi to uľahčilo vývoj a možnosti testovania API.

Vygenerovanú dokumentáciu som neskôr použil aj na generovanie klientských volaní na server. Skúsil som pri tom dva rôzne generátory. Prvým bol webový generátor *Swagger Editor*. Celý generátor funguje online a preto bolo jeho použitie jednoduché. Kód som generoval v TypeScripte tak, že volania využívali `Fetch Api`. Vygenerovaný kód bol veľmi dlhý, v jednom súbore o zhruba 6000 riadkoch a navyše obsahoval chyby, ktoré by sa museli ručne odstrániť. To by bol problém, pretože pri úprave API a znovu vygenerovaní kódu by sa kód musel upravovať tiež znovu.

Ďalší nástroj ktorý som použil, *swagger-typescript-api*, mal lepšie výsledky. Je dostupný zo správcu balíčkov *npm*. Generuje klientský kód priamo pre TypeScript v dvoch možnostiach: buď s využitím `Fetch Api`, alebo `Axios`. V základe tiež generuje kód do jedného súboru, no dá sa to zmeniť pomocou prepínača `--modular`. Celý príkaz na generovanie je zobrazený vo výpise kódu 4.4. `schema.yaml` je súbor s dokumentáciou API a parameter za prepínačom `-o` je názov priečinku, kde sa má výsledok vygenerovať. Výsledný kód bol bez chýb a rozdelený na súbory podľa jednotlivých entít. Okrem všetkých volaní pre REST API tento nástroj vygeneroval aj datové typy v jazyku TypeScript, ktoré tieto volania prijímajú, alebo vracajú.

Výpis kódu 4.4 Generovanie klienta so swagger-typescript-api

```
npx swagger-typescript-api -p ./schema.yaml -o ./modular-src --modular
```

4.3 Implementácia klientskej časti aplikácie

Po implementácii serverovej časti som prešiel na klientskú časť. To, že som najprv implementoval serverovú časť mi pomohlo v tom, že som nemusel vytvárať imitáciu chovania serveru pomocou mockou. Táto sekcia sa bude venovať zaujímavým technológiám a postupom, ktoré som pri implementácii použil. Okrem toho obsahuje aj obrázky niektorých stránok z výslednej aplikácie. Zvyšné obrázky stránok je možné vidieť v dodatku B.

4.3.1 Knižnica MUI

Väčšinu štýlov komponent mojej aplikácie som vytváral sám za pomoci `scss`. No vytvárať niektoré často objavujúce sa komponenty a ich funkcie by bolo zbytočne zdĺhavé. Preto som použil knižnicu komponent *MUI*. Knižnicu som využil napríklad na vytvorenie modálnych okien. Spomínané modálne okno prišlo s plnou funkcionalitou, s otváraním, zatváraním všetkými spôsobmi (kliknutie mimo onka, stlačenie klávesy `ESC`) a animáciou zobrazenia. Bez *MUI* by som musel všetko toto písať manuálne sám. Modálne okno s otázkou pre hostí a ich odpoveďami je zobrazené na obrázku 4.6.

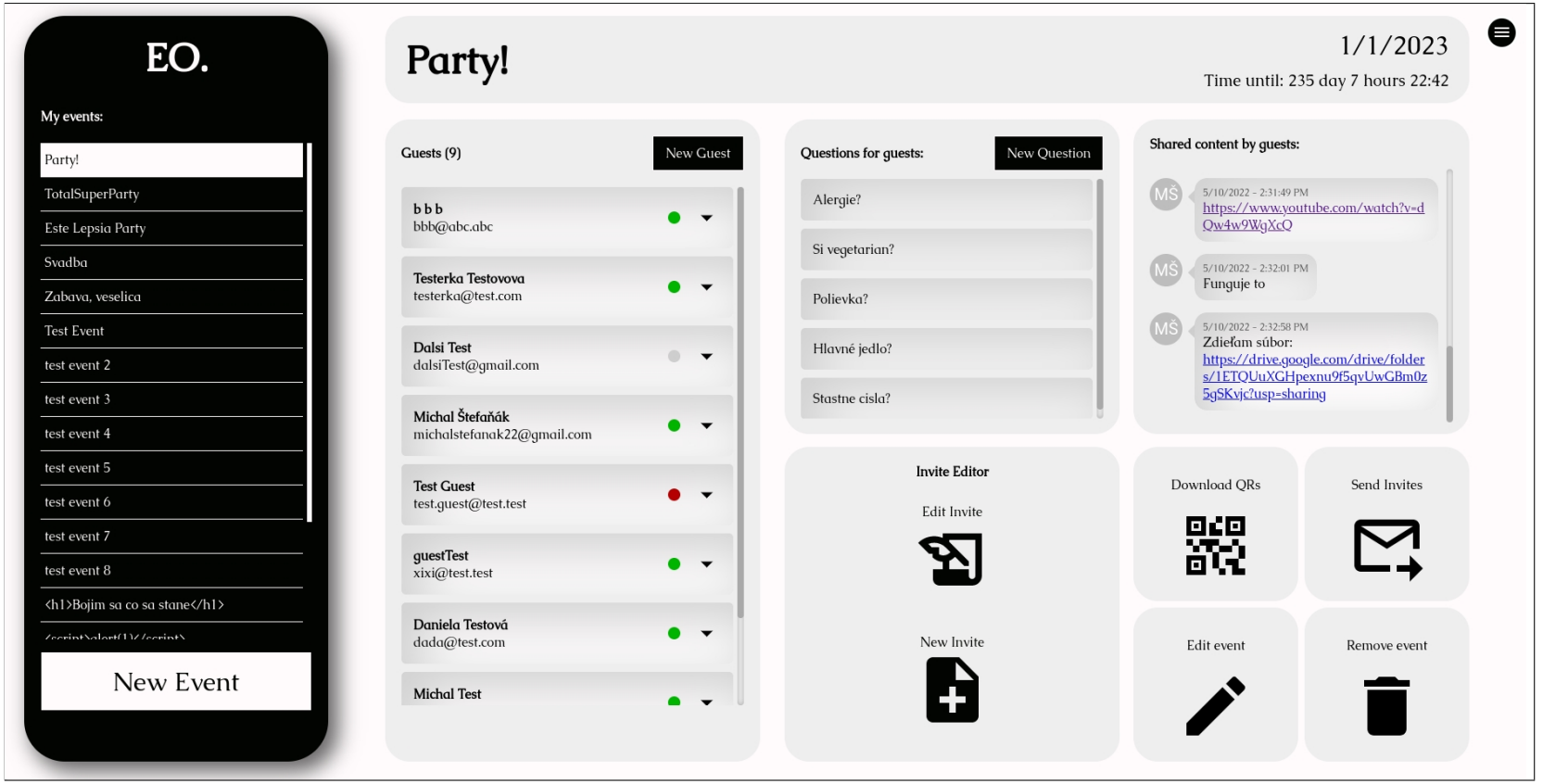
Ďalšou užitočnou vecou, ktorú *MUI* ponúka je rozsiahla knižnica ikón. Táto knižnica sa inštaluje samostatne po nainštalovaní základnej knižnice *MUI*. Použité ikony je možné vidieť na obrázku 4.4.

4.3.2 React useForm

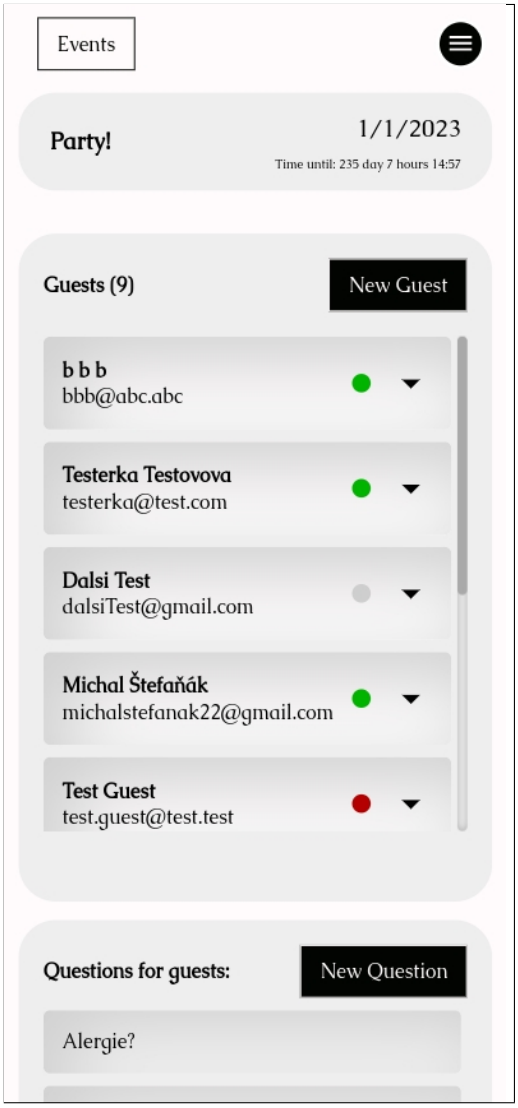
Aplikácia obsahuje veľký počet formulárov. Napríklad pri vytváraní novej akcie, pridávaní hosťa, alebo vytváraní otázok pre hostí. Písanie týchto formulárov mi uľahčil React hook s názvom *useForm*. Tento hook poskytuje knižnica *react-hook-form*, ktorá sa inštaluje zo správcu balíčkov pre JavaScript – *npm*. Hook *useForm* spolupracuje s jazykom TypeScript a vďaka tomu pri odosielaní formulára daný formulár poskytne dátový typ, ktorý som určil a ja s ním môžem ľahko ďalej pracovať. To sa dopĺňa s dátovými typmi vygenerovanými z dokumentácie. Generovanie týchto typov som opisoval v sekcii 4.2.6. Hook rieši aj validáciu jednotlivých polí a ľahkú manipuláciu s nimi.

4.3.3 Responzívny dizajn

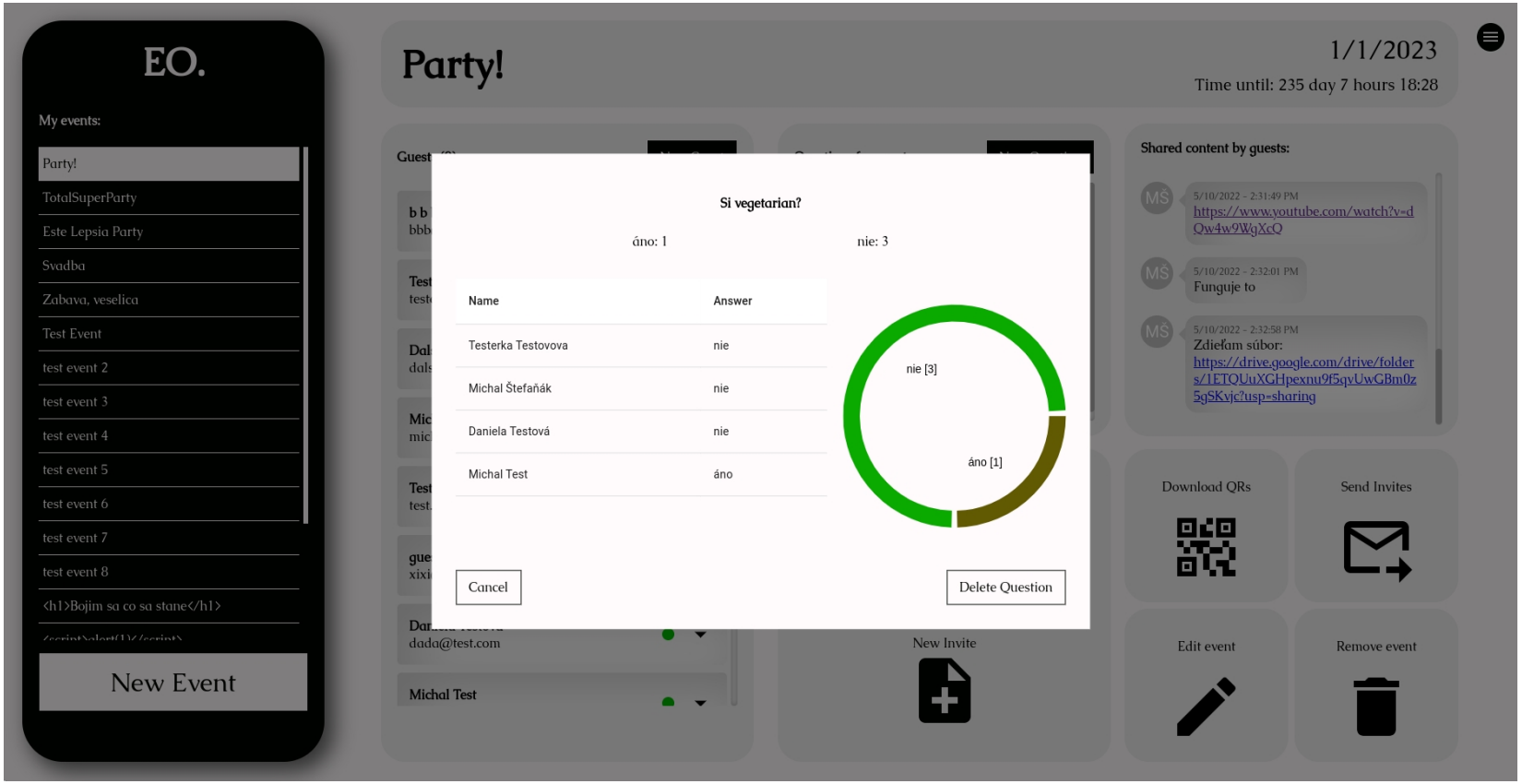
Z nefunkčných požiadavkov vychádzalo, že aplikácia má byť dostupná na rôznych zariadeniach. To zabezpečí responzívny dizajn aplikácie. Vytvoril som súbor `breakpoints.scss`, kde som zadefinoval veľkosti jednotlivých najpoužívanejších zariadení. Aby som tento súbor nemusel importovať v každej komponente zvlášť, tak som použil balík *CRACO*. *CRACO* poskytuje konfiguračnú vrstvu pre REACT aplikácie vytvorené pomocou *create-react-app*. Vďaka nemu som súbory spojené so štýlmi komponent importoval do komponent priamo pri zostavovaní aplikácie. Obrázky 4.4 a 4.5 znázorňujú použitie responzívneho dizajnu, konkrétne pohľad z obrazovky počítača a pohľad z mobilného telefónu, v tomto poradí.



Obr. 4.4 Pohľad organizátora na ovládací panel akcií – veľká obrazovka



Obr. 4.5 Pohľad organizátora na ovládací panel akcií – malá obrazovka



Obr. 4.6 Pohľad organizátora na ovládací panel akcií – veľká obrazovka

Kapitola 5
Testovanie

V tejto časti budem testovať aplikáciu, ako bolo v uvedené v zadaní práce. Kapitola sa rozdeľuje na dve časti. V prvej sa venujem automatickému testovaniu aplikácie a v druhej užívateľskému testovaniu so skutočnými testermi.

5.1 Automatické testovanie

„*Píš testy. Nie príliš veľa. Najmä integračné.*“ [13] – uviedol riaditeľ a zakladateľ spoločnosti *Vercel* – Guillermo Rauch v príspevku na sociálnej sieti *Twitter*. V mojej práci som písal najmä integračné testy. Integračné testy testujú viac jednotlivých častí a to ako spolupracujú. Takéto testy pokryjú veľkú časť scenárov z časti 3.1.5 a tým teda prípady toho ako užívateľ s aplikáciou interaguje.

Konkrétne som spomínaným typom testov testoval API. Django REST Framework poskytuje viac spôsobov ako kód testovať. Ja som použil dedenie od `ApiTestCase` triedy. Vďaka tomu bolo písanie integračných testov jednoduché. Príklad testu ktorý testuje vytvorenie nového hosťa s prihláseným organizátorom je vo výpise kódu 5.1. Na druhom riadku je Python dekorátor, čo je funkcia, ktorá vezme inú funkciu a rozšíri jej správanie bez toho, aby pôvodnú funkciu modifikovala. Tento dekorátor vytvorí testovacieho užívateľa a zároveň ho prihlási a to pred vykonaním funkcie, nad ktorou bol dekorátor použitý. Test ďalej prebiehal za použitia princípu AAA, teda Arrange (priprav), Act (konaj) a Assert (over platnosť).

Výpis kódu 5.1 Testovanie vytvorenia nového hosťa Model

```
class GuestViewTests(APITestCase):
    @create_and_login_user()
    def test_create_guest(self):
        # arrange
        initial_guest_count = Guest.objects.all().count()
        guest = {"name": "Bob", "email": "bob@test.com"}
        # act
        response = self.client.post(reverse("guests"), guest)
        # assert
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(initial_guest_count + 1,
                         Guest.objects.all().count())
```

5.2 Užívateľské testovanie

Po implementácii aplikácie, keď bola v použiteľnom a takmer dokončenom stave, som usporiadal užívateľské testovanie. Na testovanie som, bohužiaľ, nemohol pozývať ľudí pomocou mojej aplikácie, pretože ešte nebola nasadená a verejne dostupná. Pozvaní testeri mali rôzne technické schopnosti. Každý tester pracoval s rovnakou verziou aplikácie a aplikácia mu bola najprv predstavená. Aplikácia bola predstavená len slovne bez ukážky funkčnosti, aby som simuloval náhodného užívateľa, ktorý aplikáciu použije na internete. Každý testovací subjekt dostal rovnaký testovací scenár, ktorý je dostupný ako príloha práce, podľa ktorého aplikáciu obsluhoval. Počas testovania som si robil poznámky, keď subjekt nesplnil časť scenáru správne, alebo ju splnil alternatívnym spôsobom. Scenár sa skladal zo štyroch častí. V prvej časti bol testovaný subjekt v úlohe organizátora akcie, ktorý chce založiť akciu a pozvať na ňu hostí. V druhej časti sa ocitá v roli hosťa, ktorého na svoju akciu pozval a tento hosť pozvanie prijíma a ďalej pracuje s pozvánkou. V tretej časti je hosťom, podobne ako v druhej časti, no tentokrát pozvanie odmieta. V poslednej časti testovania sa tester stáva znovu organizátorom, kde má za úlohu pozrieť sa na to, ako hostia reagovali na pozvánku. Po každej časti som s testermi urobil rozhovor, týkajúci sa danej časti.

5.2.1 Subjekt A

Prvý testovací subjekt nemá technické zázemie a schopnosti užívania počítača sú na administratívnej úrovni (MS Word, MS Excel). Rozhovori v tejto sekcii sa vzťahujú k [14].

5.2.1.1 Testovanie – prvá časť

Subjekt mal zo začiatku problém pri registrácii, pretože nevedel nájsť tlačidlo na registráciu v úlohe 1). Subjekt uviedol, že zo spôsobu, ktorým bolo zamýšľané sa registrovať nie je poznať, že sa o registráciu jedná. Taktiež uviedol, že by pomohla možnosť zmeny jazyka aplikácie. Ďalšie časti testovania prebehli v poriadku, až do bodu 5), kde subjekt do otázky s preddefinovanými odpoveďami nezadal odpoveď. Systém to zároveň povolil a na túto skutočnosť neupozornil. Táto chyba by mala byť opravená s vysokou prioritou. Ďalšie časti prvej časti scenára prebehli bez chýb.

V rozhovore po prvej časti testovania uviedol, že aplikácia bola intuitívna a mala pekný minimalistický dizajn. Subjekt ďalej uviedol, že mu chýbali oznámenia pri vykonaní zmien, ktoré by značili úspešnosť, alebo neúspešnosť prevedenej operácie. Napríklad v úprave pozvánky pri uložení, alebo pri pridaní hosťa aplikácia nepotvrdila úspech operácie.

5.2.1.2 Testovanie – druhá a tretia časť

Druhá časť prebehla bez problémov, alebo poznámok od testera. V rozhovore uviedol, že oceňuje jednoduchosť procesu potvrdenia pozvania a odpovedania na organizátorové otázky.

5.2.1.3 Testovanie – štvrtá časť

V poslednej časti prvej úlohy subjekt nenašiel spôsob, akým sa pozrieť na odpovede len jedného hosťa. Namiesto toho tieto údaje získaval v časti s otázkami, postupným prezeraním jednotlivých otázok. Ďalšie časti prebehli bez problémov.

5.2.2 Subjekt B

Druhý testovací subjekt bol študentom informatiky. Rozhovory v tejto sekcii sa vzťahujú k [15].

5.2.2.1 Testovanie – prvá časť

Subjekt mal v prvej úlohe rovnaký problém nájsť miesto registrácie ako subjekt A. V tretej úlohe subjekt prekvapilo, že je možné vytvoriť akciu v minulosti. To ale aplikácia umožňuje a nepovažuje za chybu. V siedmej úlohe skúšal zadať vysoké hodnoty veľkosti textu, ktoré neboli obmedzené a obmedziť by sa mali.

V rozhovore po prvej testovacej časti uviedol, že aplikácia sa jednoducho a intuitívne používa, no tiež mu chýbali oznamy o úspechu, respektíve neúspechu prevedenej operácie.

5.2.2.2 Testovanie – druhá a tretia časť

V rozhovore po druhej časti subjekt uviedol, že okno so správami je príliš veľké, čo môže spôsobiť to, že si ho niekto bez posunu stránky dole nevšimne. Taktiež chýbali oznamy po prevedených operáciách.

5.2.2.3 Testovanie – štvrtá časť

V poslednej časti subjekt splnil všetky úlohy. Uviedol, že mu chýbalo potvrdzovacie okno pri mazaní hostí. To by zabránilo nechcenému zmazaniu.

5.2.3 Subjekt C

Tretí testovací subjekt bol tiež študentom informatiky. Rozhovory v tejto sekcii sa vzťahujú k [16].

5.2.3.1 Testovanie – prvá časť

Prvé tri úlohy zvládol subjekt bez problému. V šiestej úlohe zostali možnosti, ktoré subjekt zadal v predchádzajúcej úlohe. Môže za to to, že stav klientskej časti sa nezmenil po odoslaní formulára. To sa dá jednoducho opraviť, no aj tak má táto chyba vysokú prioritu. Podobne iné formuláre zachovávali obsah aj po odoslaní, čo predstavuje skôr nevýhodu ako výhodu v tomto prípade.

Subjekt uviedol v rozhovore po prvej časti, že sa mu páči dizajn aplikácie a je jednoduchý pre nových užívateľov. Chýbala mu možnosť znovu odoslať pozvanie hosťom, ktorý na pozvanie doposiaľ neodpovedali. Touto možnosťou aplikácia zatiaľ nedisponuje.

5.2.3.2 Testovanie – druhá a tretia časť

Druhá časť bola prevedená bez problémov. V rozhovore subjekt uviedol, že by bolo dobré pridať možnosť na zmenu odpovede, teda keď človek príjme, respektíve odmietne pozvanie, tak by mal mať možnosť si svoju voľbu neskôr rozmyslieť.

5.2.3.3 Testovanie – štvrtá časť

V poslednej časti subjekt nemal problém so žiadnou úlohou. Podotkol, že by sa mu páčila možnosť filtrovať hostí podľa toho, či pozvanie prijali, odmietli, alebo ešte neodpovedali. Ďalej uviedol, že prehľad odpovedí hostí s vygenerovanými grafmi je pre organizátora užitočný.

5.2.4 Záver užívateľského testovania

Väčšina problémov, ktoré testeri s aplikáciou mali, boli spojené s klientskou, alebo vizuálnou stránkou aplikácie. Napríklad, že formuláre, ktoré vyplnili a odoslali následne nevymazali odpovede, alebo že chýbali oznamy o vykonanej operácii užívateľa. Našli sa aj významnejšie chyby v

aplikácii, ktoré majú vysokú prioritu, s ktorou ich je potrebné opraviť. Ďalej testovanie prinieslo viacero nápadov na zlepšenie aplikácie. Medzi ne patrí možnosť filtrovať hostí podľa stavu ich pozvánky, pridanie voľby jazyka aplikácie, znovu odoslanie pozvania hosťom, ktorý pozvanie zatiaľ nepotvrdili, alebo potvrdzovacie okno pre vymazanie hosťa. Ďalšou pripomienkou bolo, že hosť nemá v momentálnej verzii možnosť zmeniť svoje rozhodnutie pri prijatí/odmietnutí pozvania. To by mohlo mať zaujímavé riešenie, a to také, že by organizátor sám určil, či rozhodnutie hosťa je záväzné, alebo ho môže zmeniť.

Záver

V tejto práci som navrhol a vytvoril systém, ktorý užívateľom pomôže vytvárať spoločenské akcie, pozývať na akcie hostí a zbierať od nich informácie. Na začiatku som preskúmal už existujúce riešenia. Z ich silných a slabých stránok som zostavil funkčné a nefunkčné požiadavky pre môj systém. Tým som splnil prvý a druhý cieľ mojej práce, ktoré som zadefinoval v úvodnej kapitole.

Aby som splnil tretí cieľ práce, bolo potrebné aplikáciu navrhnúť. Návrh mal dve časti: navrhnúť ako bude aplikácia používaná a v druhej časti navrhnúť technológie vhodné pre implementáciu. Zvolil som formu webovej aplikácie, rozdelenú na serverovú a klientskú časť.

Posledným cieľom práce bolo aplikáciu implementovať a otestovať. Serverovú časť som písal v jazyku *Python* za pomoci frameworku *Django*. Klientská časť je napísaná v jazyku *TypeScript* s knižnicou *React*. Tieto dve časti spolu komunikujú pomocou *REST API*.

V poslednej časti práce som sa venoval testovaniu mojej aplikácie. Najprv som použil automatické testy, ktoré mi pomáhali už počas vývoja, napríklad na zachovanie cielenej funkcionality po zásahoch do kódu a po dokončení implementácie potvrdili, že všetko funguje tak ako má. Okrem automatických testov prebehli aj užívateľské testy, ktoré mi pomohli odhaliť nedostatky a vďaka ktorým som dostal tipy na zlepšenie aplikácie, ktoré som opisoval v kapitole 5.

Spolu so spomínanými vylepšeniami, ktoré boli navrhnuté pri testovaní, by som chcel navrhnúť aj ďalšie vylepšenia aplikácie:

- Pridávanie obrázkov k šablónam pozvánok.
- Tvorba zasadacieho poriadku pre hostí.
- Zdieľanie vytvorených šablón pozvánok medzi organizátormi akcií.
- Lokalizácia aplikácie do iných jazykov.
- Podrobnejšie nastavenia a možnosti pre organizátora ohľadne akcie (napríklad povoliť, alebo zakázať zmenu odpovede na pozvanie, alebo možnosť znovu odoslať pozvánky hosťom, ktorý ešte na pozvanie neodpovedali).

Výsledok mojej práce ocenia ľudia, ktorí často organizujú rôzne spoločenské akcie, o ktorých chcú mať prehľad. Vďaka otázkam ktoré položia hosťom v rámci pozvania budú mať všetky potrebné informácie o hosťoch na jednom mieste. Okrem organizátorov aplikáciu ocenia aj hostia, ktorí pozvanie obdržia, pretože odpovedať na pozvanie je jednoduché a rýchle.

Dodatok A

Scenár užívateľského testovania

Scenár organizátora akcie – Vytváranie akcie

1. Zaregistruj sa do aplikácie.
2. Vytvor novú spoločenskú akciu.
3. Pozvi na akciu troch hostí (e-mail nemusí existovať).
4. Vytvor pre hostí otázku, ktorá bude vyžadovať otvorenú odpoveď hosťa. (Príklad: Aká je tvoja obľúbená farba?)
5. Vytvor pre hostí otázku, ktorá bude mať dopredu definované možné odpovede a hosť si môže vybrať jednu odpoveď. (Príklad: Si vegetarián? Odpovede: áno/nie)
6. Vytvor pre hostí otázku, ktorá bude mať dopredu definované možné odpovede a hosť môže vybrať viacero odpovedí. (Príklad: Ktoré miesta chceš v rámci akcie navštíviť? Odpovede: Hrad/Obchod/Park)
7. Uprav vzhľad pozvánky.
   - Zmeň farbu pozadia.
   - Zmeň farbu hlavného textu.
   - Zmeň farbu sekundárneho textu.
   - Zmeň obsah hlavného textu.
   - Zmeň obsah sekundárneho textu.
   - Zmeň veľkosť písma hlavného textu.
   - Zmeň veľkosť písma sekundárneho textu.
8. Stiahni QR kódy ku pozvánkam.
9. Odošli pozvánky.

Scenár hosťa – Odpoveď na pozvánku – potvrdenie účasti

1. Potvrď účasť na akcii.
2. Vyplň odpovede.

3. Napíš správu na akciu.
4. Napíš správu na akciu obsahujúcu ľubovoľný link.

Scenár hosťa – Odpoveď na pozvánku – zamietnutie účasti

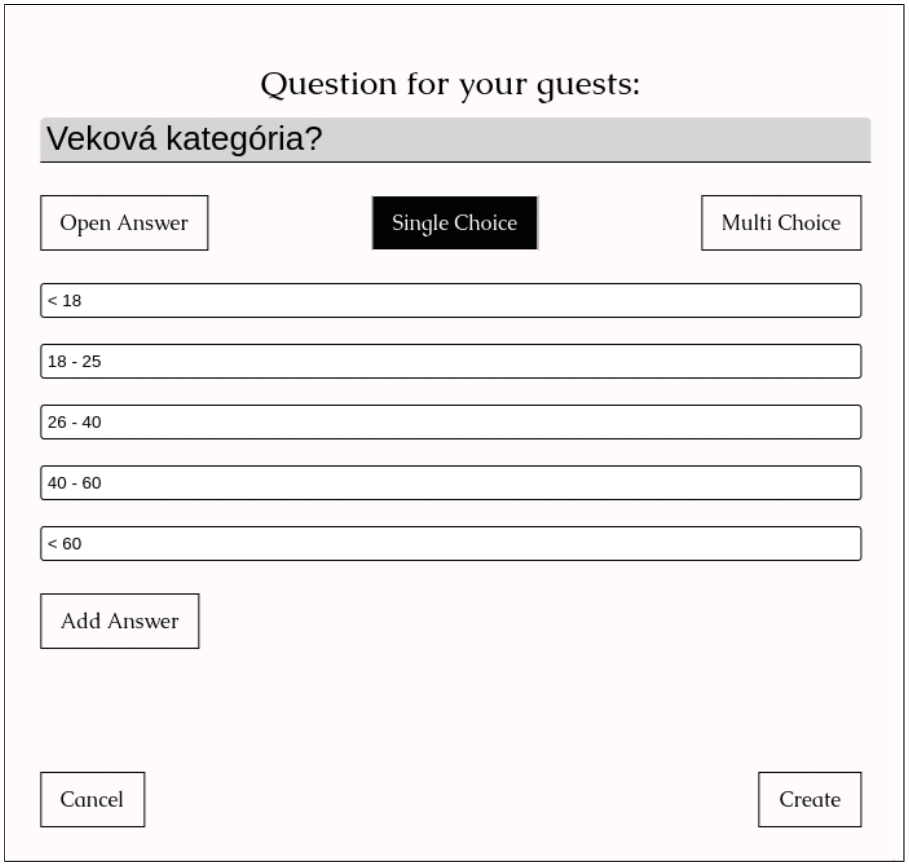
1. Zamietni účasť na akcii.

Scenár organizátora akcie – zhodnotenie dát od hostí

1. Pozri sa ako na vytvorené otázky odpovedal hosť, ktorý účasť potvrdil
2. Pozri sa ako všetci hostia odpovedali na otázku s viacerými možnosťami na odpoveď.
3. Odstráň hosťa, ktorý účasť zamietol.
4. Odstráň otázku s voľnou odpoveďou.

Dodatok B

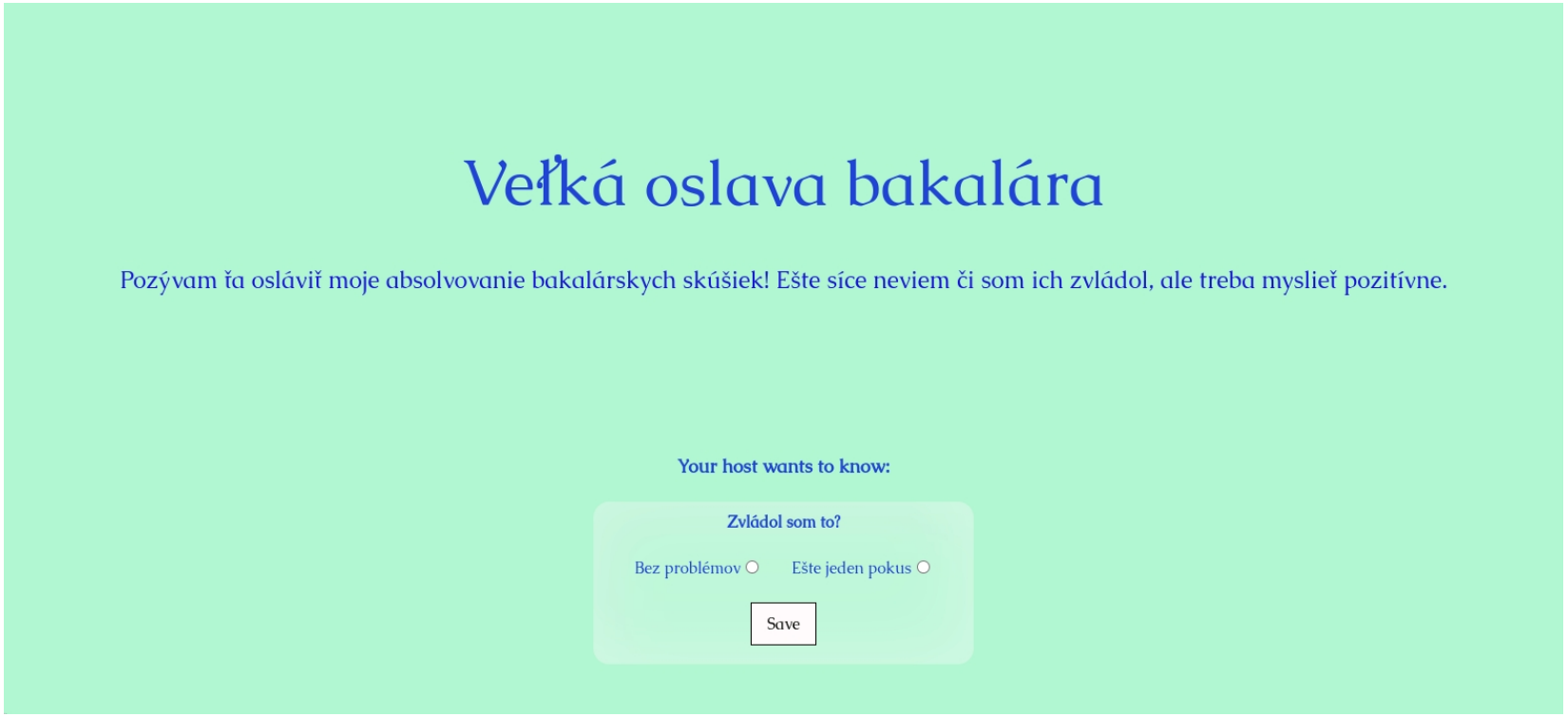
Ukážka aplikácie



Obr. B.1 Pohľad organizátora na vytváranie otázky pre hostí

Obr. B.2 Pohľad organizátora na upravovanie pozvánky

Obr. B.3 Pohľad hosťa na zatiaľ nezodpovedanú pozvánku



Obr. B.4 Pohľad hosťa na potvrdenú pozvánku

Bibliografia

1. BOTERHOVEN, Daniel. *Web, Mobile or Desktop App – Which Is Right for Your Project?* [Online]. Denim, Máj 2020 [cit. 2022-04-27]. Dostupné z : `https://denimdev.com.au/blog/web-mobile-or-desktop/`.
2. REDHAT. *What is a REST API?* [Online]. 8 Máj 2020 [cit. 2022-04-27]. Dostupné z : `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.
3. GUPTA, Lokesh. *What is REST* [online]. 7 Apríl 2022 [cit. 2022-05-02]. Dostupné z : `https://restfulapi.net/`.
4. MANSFIELD, Timothy. *The Best Guide to Know What Is React* [online]. Máj 2019 [cit. 2022-04-28]. Dostupné z : `https://interaction.net.au/articles/what-django-and-why-it-great-creating-websites/`.
5. VINCENT, William S. *Django for APIs: Build web APIs with Python and Django*. WelcomeToCode, 2020. ISBN 978-1735467221. (preklad vlastný).
6. DESHPANDE, Chinmayee. *The Best Guide to Know What Is React* [online]. December 2021 [cit. 2022-04-28]. Dostupné z : `https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs`.
7. GARDÓN, Diego S. *Next.js vs. React: The Difference & Best Frontend Framework* [online]. 8 Apríl 2022 [cit. 2022-04-27]. Dostupné z : `https://snipcart.com/blog/next-js-vs-react`.
8. OYOM, Ada N. *Understanding the MVC pattern in Django* [online]. 19 Júl 2017 [cit. 2022-05-02]. Dostupné z : `https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f`.
9. *Django MVT Architecture* [online]. August 2020 [cit. 2022-04-28]. Dostupné z : `https://www.askpython.com/django/django-mvt-architecture`.
10. *Django Rest Framework* [online]. [B.r.] [cit. 2022-04-28]. Dostupné z : `https://www.django-rest-framework.org/`.
11. *JSON Web Tokens* [online]. [B.r.] [cit. 2022-05-10]. Dostupné z : `https://auth0.com/docs/secure/tokens/json-web-tokens`.
12. GILLIS, Alexander S. *UUID (Universal Unique Identifier)* [online]. August 2021 [cit. 2022-04-27]. Dostupné z : `https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier`.
13. RAUCH, Guillermo [online]. 10 December 2016. Dostupné tiež z: `https://twitter.com/rauchg/status/807626710350839808`. (preklad vlastný).
14. KOHÚTOVÁ, Daniela. *Rozhovor pri užívateľskom testovaní* [osobný rozhovor]. 2022.

15. MELKO, Michal. *Rozhovor pri užívateľskom testovaní* [osobný rozhovor]. 2022.
16. GLÁZR, Tomáš. *Rozhovor pri užívateľskom testovaní* [osobný rozhovor]. 2022.

Obsah priloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `server` ..... serverová časť
    - `client` ..... klientská časť
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF