Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Darovací platforma pro vzdělávací instituci

Diplomová práce

Autor: Bc. Jiří Kumprecht
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, PhD.

Hradec Králové duben 2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 25.4.2023 Bc. Jiří Kumprecht

Poděkování:

Děkuji vedoucí diplomové práce Mgr. Daniele Ponce, PhD. za metodické vedení práce, pomoc a cenné připomínky.

Anotace

Diplomová práce se zaměřila na vývoj darovací platformy na míru pro konkrétní školu Základní školu speciální a praktickou školu Diakonie Českobratrské církve evangelické Vrchlabí. Škola se na autora práce obrátila z nutnosti aktuální situace, kterou zapříčinila končící smlouva pronájmu prostor školy a požádala jej o vývoj aplikace pro další způsob, jak sehnat finanční prostředky na výstavbu nové školy ve Dvoře Králové nad Labem. Pro vytvoření a vývoj aplikace byly podkladem funkční a nefunkční požadavky ze strany zadavatele, tedy ředitelky školy. Na základě těchto požadavků došlo k vypracování návrhu, podle kterého byl systém v konečné fázi implementován. Hlavním cílem bylo analyzovat, navrhnout a implementovat webovou aplikaci ve formě darovací platformy na míru, aby se vypomohlo s realizaci cílů školy.

Annotation

Title: Donation platform for an educational institution

The diploma thesis focused on the development of a tailor-made donation platform for a specific school, the Primary Special and Practical School of the Evangelical Church of Czech Brethren in Vrchlabí. The school approached the author of the thesis out of necessity of the current situation caused by the expiring lease contract of the school premises and asked him to develop an application for another way to raise funds for the construction of a new school in Dvůr Králové nad Labem. The basis for the creation and development of the application was the functional and non-functional requirements from the client, i.e. the school principal. Based on these requirements, a proposal was developed, according to which the system was finally implemented. The main objective was to analyze, design and implement a customized web application in the form of a donation platform to help realize the school's goals.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V této diplomové práce se autor zaměřil na vývoj darovací platformy pro Základní školu speciální Diakonie ČCE Vrchlabí. Vedení školy požádalo autora práce o pomoc při získávání peněz na pozemek pro výstavbu nové školy ve Dvoře Králové nad Labem.

Hlavním cílem byla analýza, návrh a implementace darovací platformy na míru podle představ ředitelky školy.

Pro vytvoření a vývoj aplikace bylo potřeba nejprve analyzovat problematiku sběrem funkčních a nefunkčních požadavků, které sloužily jako podklad pro další fázi vývoje.

Následovalo seznámení s architekturou aplikace, návrhového vzoru MVC a způsobem komunikace mezi dílčími částmi systému.

Další fází vývoje byl návrh, ve kterém spočívalo si ujasnit entity, které budou v datové části použity a jejich potřebné atributy. Následné v této fázi byly také navrhnuty způsoby procesu přispívání na jedinečný projekt „Kup si svoji část pozemku“.

Po fázi návrhu následuje rozsáhla kapitola implementace, ve které jsou popsány jak technologie, které byly při vývoji webové aplikace použity, tak i autorem práce vymyšlená konkrétní řešení a implementace dílčích úloh aplikace.

Diplomová práce se zabývá také bezpečnostní částí, je popsán princip autentizace a autorizace a jejich implementace ve vyvíjené webové aplikaci.

Byla popsána implementace modelů, dostupné REST API backendové části aplikace a všechny stránky frontendové části systému. V závěru je popsáno samotné nasazení aplikace online.

2 Cíl práce

Hlavním cílem diplomové práce je analyzovat, navrhnout a implementovat darovací platformu pro vzdělávací instituci.
Škola získá vlastní darovací platformu vyvinutou na míru podle představ ředitelky. Pozitivně přispěje k realizaci cílů školy.

3 Metodika zpracování

Základní fáze vývoje softwarového programu jsou úvodní studie, analýza, návrh a implementace. Všechny výše zmíněné fáze vývoje byly řádně provedeny, a to v uvedeném pořadí. Pro vývoj webové aplikace byl zvolen přírůstkový model v kombinaci s agilním vývojem. Tento přístup umožňuje rozčlenit projekt na dílčí segmenty, zohledňuje rozšiřitelnost a poskytuje možnost zavádění změn v průběhu procesu vývoje aplikace. Použitím přístupu tohoto modelu bylo možné aplikaci vyvíjet po částech prostřednictvím malých sérií vodopádů. Pro uvedení příkladu si lze jednotlivé vodopády představit jako průběh vývoje jednotlivých částí systému. Pokaždé se nejdříve začalo analýzou dané funkcionality, která byla hlavním bodem schůzky s ředitelkou školy, ve které se specifikovala funkčnost daného prvku. Po provedení analýzy se přešlo k následující fázi, kterou byl návrh. Na základě požadavků z analýzy byl sestaven návrh, pomocí kterého bylo možné implementovat požadovanou funkcionalitu. Implementace je finálním krokem, který vede k dokončení vyvíjené části. Tento krok představuje vytvoření kódu, který realizuje navržené řešení. Tímto způsobem se postupovalo při vývoji všech částí a funkcionalit darovací platformy na míru.

4 Analýza

V této první hlavní kapitole je popsána analýza, která je nezbytná pro další fáze vývoje aplikace.

4.1 Výchozí stav, potřeby a cíle školy

Základní škola speciální a praktická škola Diakonie ČCE Vrchlabí [1] se musí vypořádat s nepříjemností, a to velmi zásadní, pro provozování její činnosti. Je nezbytné zajistit nové prostory, ve kterých by mohl probíhat výchovně-vzdělávací proces. Konkrétně se to týká odloučeného pracoviště dané vzdělávací instituce ve Dvoře Králové nad Labem. Nyní výuka probíhá v pronajatých prostorech s končící smlouvou k srpnu 2025. Do té doby si musí škola zajistit nové prostory a jedno z dostupných a možných řešení je stavba nové školy v tomtéž městě. Ředitelka školy si tuto variantu stanovila za svůj rozvojový cíl, kterého chce dosáhnout. Pro jeho dosažení je třeba zrealizovat kroky s časovým harmonogramem, zajistit mnoho úkolů, které jsou v kompetenci ředitelky. Jedním z těchto úkolů bylo obstarat finance na pozemek pro výstavbu nové školy. S tímto dílčím úkolem měl právě možnost vypomoci autor této práce prostřednictvím tvorby darovací platformy. Než k tomu ale došlo, škola se pokoušela k finančním prostředkům dostat jinými alternativami. Snažila se oslovovat malé dárce, nadace a velké firmy o podporu ve formě finančního daru, většinou osobní schůzkou. Dále se pokoušela oslovovat veřejnost vytisknutím a vylepením letáků s informacemi o stavbě nové školy a s číslem sbírkového účtu pro finanční dary. Na letáku se také nachází QR kód pro pohodlnější způsob platby. Následující variantou bylo umístění přibližně 15-ti kasiček do různých obchodů a firem ve Dvoře Králové nad Labem. Těmito způsoby se však darovalo pouze menší množství finančního obnosu, a tak se muselo přijít s další variantou.

Hlavním problémem je tedy získání finančních prostředků pro výstavbu nové školy. Jak již bylo zmíněno, bylo použito mnoho způsobu k získání finančních zdrojů. Dalším dílčím problémem je, jak motivovat přispěvatele, aby byli ochotni přispět a pomoci tak k realizaci cílů školy.

4.2 Stávající řešení třetích stran

Zvažovalo se využít online darovacích platforem, konkrétně služby Darujme.cz [2]. Škola již tuto platformu využívala dříve na jiné své projekty, avšak nikdy nedosáhla na této platformě většího úspěchu. Vždy se vybraly pouze jednotky procent z požadované celkové částky. Na této darovací platformě se dá přispívat jak jednorázově, tak i měsíčně. Darování mají uzpůsobeno tak, že si dárce vybere jednu z předpřipravených částek, které zde zadal zadavatel projektu, na který se vybírá. Nebo je zde možnost přispět jinou, vlastní částku, kterou si dárce zadá dle svého uvážení. Na této online platformě chybí interaktivní a motivační prvky. Není zde ani veřejně uvedeno, kdo přispěl, kolik přispěl a není zde ani ta možnost zpřístupněna. Někteří dárci chtějí být vidět, a to je jedním z důvodů, proč vytvořit vlastní darovací platformu.

Další alternativou darovací služby Darujme.cz je platforma Daruj správně [3]. Jedná se o velmi podobnou darovací platformu s několika rozdíly. Jedním z hlavních rozdílů je možnost si u projektu zobrazit přispěvatele. Každý přispěvatel se může rozhodnout, zda chce darovací částku zobrazit nebo dar anonymizovat. Dalším rozdílem je, že zde nejsou předpřipravené částky ve formě tlačítek, ale každý přispěvatel si sám napíše částku, kterou je ochotný darovat.

Následující variantou je možnost využití služby Donio [4]. Tato darovací platforma umožňuje zobrazit seznam dárců u jednotlivých projektů a také aktuality a komentáře. Dárce má možnost přispět vlastní či předpřipravenou částku.

Všechny zmíněné platformy fungují na velmi podobném principu. Jednoduše přispěvatel vybere jednu z přednastavených částek nebo si zvolí vlastní částku. Využití zahraničních darovacích platforem pro projekt školy nemá žádný smysl. Pro přispěvatele ze zahraničí by škola přišla jako malá, neznámá organizace, pro ně nic neříkající. Je potřeba zacílit na české přispěvatele, nejlépe v regionu Dvora Králové nad Labem.

Pokud se všechny tyto platformy shrnou a abstrahují se z nich základní principy fungování a používání, tak by shrnutí všech těchto platforem vypadalo následovně. Uživatel/zadavatel projektu na těchto platformách zaregistruje svoji organizaci a vypíše všechny potřebné základní údaje o této organizaci. Následně je

schopna zadávat jednotlivé tzv. projekty, na které chce zadavatel vybírat peníze. K těmto projektům lze zpravidla napsat nějaký popis, v textové formě a lze připojit i obrázek. Dále je možné zadat požadovanou celkovou částku, kterou organizace na daný projekt cílí vybrat. Většinou se udává i časové rozmezí, dokdy je potřeba daný obnos peněz vybrat. Bývá zde možnost napsat ideální částku příspěvku nebo alespoň 4 pevné varianty. Na všech platformách je možnost darovat vlastní napsanou částku dárcem. Darovací platformy vždy zobrazují stav vybírání na projekt, tzn. kolik se již z požadované celkové částky prozatím darovalo peněz. Ať už ve formě procent nebo nějaké zabarvování postupného načítacího baru. Některé platformy zobrazují své dárce. Buď zobrazují všechny dárce, kde si jednotliví dárci mohou zvolit, zda chtějí darovanou částku zveřejnit či nikoliv. Nebo jsou zobrazovány pouze některé nejvyšší dary nebo určité poslední množství příspěvků. Žádná z těchto dříve zmíněných platforem nemá poutavý či interaktivní způsob přispívání. Všechno je generické, což dává smysl, aby zde mohla být využita znovu použitelnost. Jenže genericita neupoutá pozornost dárců a žádným stylem nevyčnívá, aby fascinovala přispěvatele. Pro větší šanci na úspěch by škola ráda vyzkoušela nějakého jedinečného prvku. Něco, v čem bude vybírání na projekt odlišný, poutavější a více motivační. Více o novém řešení a nové myšlence, která vnese nový nádech projektu darování na pozemek pro výstavbu nové školy je napsáno v další kapitole. Samozřejmě, že čím odlišnější a výjimečnější přístup k vybírání na projekt bude zvolen, tím nižší znovu použitelnost takového typu projektu bude. Ale přeci jen se jedná o darovací platformu na míru a z tohoto důvodu lze i takový přístup, tvorby jedinečného projektu, zrealizovat.

4.3 Vlastní řešení na míru

Vlastní řešení na míru znamenalo vytvořit novou darovací platformu podle požadavků školy. Hlavním důvodem, proč se tak zadavatel rozhodl učinit, byla představa v novém přístupu k online darování. Paní ředitelka školy měla představu o průběhu darování na pozemek následujícím způsobem. Jejím požadavkem bylo vyobrazit interaktivní mapu, na které by byl vykreslený pozemek. Dárce by mohl interaktivně s mapou manipulovat a interagovat. Mapa by byla segmentovaná na několik tisíc částí o různých cenách, které by šly nakupovat. Dárce by tedy měl

možnost přispět přímo na jeho vybranou část pozemku. Po nakoupení by u této části na mapě byly vidět informace o dárci, pokud by to bylo jeho přáním. Pokud jde o ostatní projekty, na které lze přispívat, ty budou již dosazeny do šablony. Tyto projekty nemají tak vysokou prioritu, a tak jedinečnost pro ně v tuto chvíli není žádoucí, ale časem se jejich existence nevylučuje.

4.3.1 Požadavky

V této kapitole jsou popsány funkční a nefunkční požadavky navrhované ředitelkou školy. Požadavky byly autorem práce korigovány a pozměněny takovým způsobem, aby se předešlo nechtěným chybám a nepříjemnostem při vývoji aplikace a zároveň, aby použitelnost aplikace pro běžné uživatele byla co nejpřirozenější.

Funkční požadavky

Funkce a funkcionality, které by aplikace měla umět a být schopná zrealizovat, byly konzultovány s ředitelkou školy. Autor práce si domluvil několik osobních schůzek, ze kterých pro vyvíjenou aplikaci nakonec vyplynuly tyto požadavky. V aplikaci bude možnost nahlédnout do galerie fotek, které graficky znázorňují představu o podobě budovy nově postavené školy. V aplikaci bude zobrazen seznam projektů. Po rozkliknutí daného projektu bude zobrazen jeho detail. Na projekty jako takové nepůjde přispívat, ale ke každému projektu bude možné přidat tzv. „darovatelné boxy“, na které lze teprve darovat. Každý takový box má zobrazovat cílovou částku, kolik již bylo vybráno z této částky a animaci a grafickou reprezentaci této hodnoty. U každého „darovatelného boxu“ bude možnost zobrazit si přispěvatele a u projektů přidávat komentáře. Dále zde budou umístěny i aktuality, které reprezentují aktuální informace o stavu projektu. Postup přispívání by měl fungovat následovně. Dárce si nejprve zvolí projekt, který ho upoutal, následně si v něm vybere „darovatelný box“, tj. konkrétní věc, na kterou chce přispět. Zvolí jednu z předpřipravených částek nebo si zaklikne možnost „Vlastní částka“. V tu chvíli se mu ukáže možnost zadat vlastní hodnotu. Následně si dárce musí zvolit viditelnost jména svého daru, to znamená, zda chce mít v seznamu přispěvatelů zveřejněno své jméno. Po zvolení viditelnosti si bude možné napsat vlastní krátkou poznámku. Nakonec již zbývá stisknout tlačítko s nápisem Přispět. V tu chvíli by se klient ocitl

na stránce s košíkem, odkud se uživatel může proklikat na vyplnění údajů k objednávce, kde je potřeba vyplnit kontaktní údaje. Pokud je příspěvek pořizován na firmu, tak jsou vyžadovány firemní údaje. Jestli dárce vyžaduje certifikát o darování, např. z důvodu, že si jej chce odečíst z daní, musí ještě vyplnit osobní údaje. Aplikace má umožnit přihlášení dvou typů uživatelů:

- Administrátor (správce)
- Klasický uživatel (dárce)

Uživatelé by měli mít možnost se zaregistrovat přímo na darovací platformě, tzn. vytvořit lokální účet, nebo přes svůj Google účet. Registrovaní uživatelé mohou spravovat své objednávky.
Správce je vytvořen jeden a smí v aplikaci provádět následující operace:

- Založit nový projekt
- Editovat či smazat projekt
- Vytvořit, editovat či smazat „darovatelný box“
- Přidávat aktuality k jednotlivým projektům
- Přidávat komentáře ke všem projektům

Nefunkční požadavky

Při použití webové aplikace jsou nejvyšší prioritou požadavky na bezpečnost. Pro zajištění této bezpečnosti je nutné zajistit šifrovanou komunikaci mezi klientem a serverem, například pomocí SSL certifikátu.
Dostupnost je dalším důležitým faktorem, a proto by mělo být možné přistupovat k aplikaci odkudkoli z libovolného zařízení s použitím internetového prohlížeče. Je také důležité zajistit, aby aplikace byla vždy dostupná, pokud možno 24 hodin denně, 7 dní v týdnu, aby uživatelé mohli kdykoliv provádět operace v aplikaci. Proto je nutné zvolit vhodné řešení pro hosting aplikace, které zajistí její dostupnost a stabilitu.
Pro správnou funkčnost uživatelských účtů je nutné zajistit správnou autentizaci a autorizaci, aby uživatelské operace v aplikaci byly správně posouzeny jako oprávněné.

Důraz byl také kladen na udržitelnost, a proto byl systém rozdělen do komponent, aby bylo možné opravovat pouze ty části, ve kterých se vyskytnou problémy, a nemusel být upravován celý rozsah aplikace.
Rozšiřitelnost je další požadavek, který umožňuje přidat novou funkčnost nebo upravovat již existující funkce bez ovlivnění existujícího systému.
A konečně, pro plynulý chod aplikace pro více uživatelů současně, by se požadavky uživatelů neměly vzájemně blokovat.

4.3.2 Definice pojmů v oblasti darování

V této kapitole jsou uvedeny a definovány jednotlivé termíny, které se s problematikou, řešenou v této práci, váží.
Dar: Finanční prostředky věnované dárcem na konkrétní charitativní sbírku.
Cílová částka: Částka, kterou zadavatel projektu či sbírky uvede jako hraniční pro úspěšnou realizaci vytyčeného cíle.
Dárce: Osoba, která poskytuje dar na konkrétní sbírku, může být buď fyzickou nebo právnickou osobou.
Darovací platforma: Online prostředí, nejčastěji webová aplikace, ve které mohou dárci přispívat finanční částky na sbírky.
Sbírka: V této práci je reprezentována „darovatelným boxem“. Definuje konkrétní věc či službu, která se za vybrané dary pořídí. Právě na ni dárci přispívají svými dary. Vždy je součástí některého z projektů.
Projekt: Udává rámcové informace o sbírkách s obdobnou charakteristikou.
Uživatel: Návštěvník nebo registrovaný uživatel darovací platformy využívající jejich služeb.

5 Návrh

V této kapitole jsou popsány základní části aplikace, použitý návrhový vzor a jakým způsobem probíhá komunikace mezi těmito částmi. Následně je rozebrána datová část aplikace, tedy objekty, které se v databázi budou uchovávat.

Autor práce se rozhodl na základě nefunkčních požadavků vytvořit webovou aplikaci s responzivním designem tak, aby byla dostupná na všech zařízeních, s různými rozměry obrazovek a odlišnými operačními systémy.

5.1 MVC architektura aplikace

Návrhový model MVC rozděluje aplikaci na 3 hlavní části. První částí je **model**, ten reprezentuje datový model aplikace a umožňuje manipulovat s daty. Ať už se jedná o ukládání, aktualizaci nebo získávání dat z databáze či jiných datových zdrojů. Model je navržen tak, aby byla zachována škálovatelnost aplikace, tzn., že je oddělen od prezentační vrstvy. Model je úzce propojen s **řadičem** (controller), který má díky tomu možnost přistupovat k datům a umožňuje prezentační vrstvě přístupu k datům. Prezentační část neboli **view**, má za úkol prezentovat data uživateli v grafické podobě, zobrazuje je ve formě textu, tabulek, grafů nebo jinými způsoby. Dále umožňuje uživateli interagovat s aplikací, a tak manipulovat s daty prostřednictvím grafického rozhraní např. tlačítky či dalšími prvky uživatelského rozhraní. Pokud dojde ke změně dat v modelu, změna se promítne automaticky v grafickém rozhraní. Pohled bývá oddělen od modelu a řadiče, je tedy možnost vytvářet více uživatelských rozhraní nad daty aplikace.

5.1.1 Klient-server

Architektura spočívá v rozdělení aplikace na části, které zastupují roli klienta nebo serveru. Pokud aplikace právě zasílá požadavek např. pomocí síťového protokolu HTTP, pak zastává roli klienta. Aplikace, která je připravena naslouchat těmto požadavkům, patřičně je zpracovávat a poskytovat odpovědi či služby, zastává pak roli serveru.

5.1.2 Frontend

Tento termín označuje část webové aplikace, která je zobrazena uživateli. Jedná se o grafické uživatelské rozhraní, které umožňuje uživateli zobrazovat data a přes patřičné nástroje s nimi manipulovat. Např. pomocí formulářů, tlačítek atd. Frontend tedy zastává prezentační vrstvu a v kombinaci s návrhovým vzorem MVC může být vytvořeno více prezentačních částí nad stejným modelem. Může se jednat o pohled pro prohlížeče, dále např. pohled mobilní aplikace.

5.1.3 Backend

Jan Štráfelda [5] k pojmu backend uvádí: *„Jako backend se označuje část webové aplikace, která slouží k administraci webu a ke zpracování dat.“* Tento termín označuje zadní část aplikace, tedy tu, která uživateli není přímo zviditelněna a umožňuje např. pomocí REST API zpracovávat data a manipulovat s nimi.

5.1.4 REST API

Pojmem REST API se rozumí komunikace mezi částmi frontend a backend. Webový server obsluhuje http požadavky klienta a vrací např. data ve formátu JSON. Požadavky z frontendu směřují na specifickou URL adresu a obsahují http metodu např. GET, POST, PATCH, DELETE. Na základě URL cesty a metody se požadavek dostane na konkrétní API serveru, které požadavek obslouží.

5.2 Návrh struktury zdrojů

V níže popsaných podkapitolách jsou definovány jednotlivé entity, ze kterých poté bude následně vycházet implementace těchto objektů. U každé dílčí entity je uveden seznam vlastností, které o sobě uchovávají a které budou později ukládány do databáze.

5.2.1 Uživatel

Jednou z nejdůležitějších entit této aplikace je uživatel. U uživatele je vhodné si uchovávat tyto informace:

- **name, surname**: Křestní jméno a příjmení uživatele. Tyto údaje budou využity zejména pro uvedení jména dárce v sekci přispěvatelé, pro vystavení

faktury či certifikátů po zaplacení objednávky a zejména u vložených komentářů k projektům.

- **email**: Kontaktní emailová adresa, pomocí které se může uživatel přihlásit do aplikace.
- **password**: Tato položka reprezentuje uživatelské heslo v šifrované podobě. Heslo je vyžadováno pro přihlášení.
- **role**: Uchovává informaci, zda je uživatel klasický dárce nebo správce.
- **isLocallyCreated**: Uchovává pravdivostní hodnotu, zda byl uživatel s daným emailem řádně zaregistrován přes formulář v aplikaci.
- **isGoogleAssociated**: Také uchovává pravdivostní hodnotu. Kde nabývá hodnoty **true**, pokud byl účet asocióván s lokálně vytvořeným účtem nebo přihlášen s použitím Google API.
- **receiveNews**: Obsahem je pravdivostní hodnota, která určuje, zda je daný uživatel přihlášen k odběru novinek a budou mu tedy zasílány emaily.

5.2.2 Projekt

Projekt poskytuje rámcové informace o sbírkách v něm vnořených. Návrh jednotlivých vlastností této entity vypadá následovně:

- **title**: Tato vlastnost uchovává název projektu ve formě textového řetězce
- **urlTitle**: Uchovává část URL adresy na daný projekt. Je využita proto, aby adresa na detail projektu byla vizuálně přívětivá a v adresním řádku prohlížeče nebylo uvedeno ID projektu.
- **desc**: Obsahuje popis projektu.
- **type**: Uchovává informaci, o jaký typ projektu se jedná. Může být základní nebo speciální, jako např. projekt „Kup si svoji část pozemku“.
- **photo**: Zde je uložena URL adresa na obrázek projektu.
- **deleted**: Obsahuje pravdivostní hodnotu, zda je projekt smazán.

5.2.3 Sbírka

Sbírka neboli „darovatelný box“ je entitou, u které jsou uchovány tyto informace:

- **title**: Pro uchování názvu sbírky.

- **desc**: Vlastnost, ve které je uchován popis sbírky.
- **photo**: Pro uložení URL adresy obrázku.
- **earnedMoney**: Slouží pro uchování číselné hodnoty, která reprezentuje dosud vybrané finance na sbírku.
- **demandedMoney**: Číselná hodnota, která uchovává celkovou vyžadovanou částku pro splnění a realizaci cílů sbírky.
- **preparedPrices**: Jde o pole číselných hodnot, které reprezentují předpřipravené částky, které jsou u sbírky k výběru pro darování.
- **projectId**: Uchovává referenci na projekt, ke kterému je vázán.
- **deleted**: Pravdivostní hodnota určující, zda je sbírka smazána.

5.2.4 Dar

Dar je entitou, u které jsou drženy tyto informace:

- **price**: Nejdůležitější informací je u daru cena. Reprezentuje výši daru.
- **createdAt**: Uchovává datum a čas, kdy byl dar vytvořen.
- **purchasedAt**: Tato hodnota reprezentuje datum a čas úspěšného zaplacení daru.
- **isPurchased**: Pravdivostní hodnota nabývající hodnotu true, pokud byl dar úspěšně zaplacen. Od té doby se započítává do vybrané částky sbírky a je zobrazen ve výpisu přispěvatelů.
- **isAnonymous**: Vlastnost uchovávající pravdivostní hodnotu, zda je dar anonymní, tzn., že nemá být zveřejněno jméno dárce ve výpisu přispěvatelů.
- **name**: Jméno dárce či název firmy, která dar poskytla. Tato vlastnost je uvedena u daru, protože jsou podporovány i nákupy bez registrace.
- **note**: Vlastní poznámka dárce k daru.
- **donatableId**: Uchovává referenci na sbírku, ke které je dar vázaný.

5.2.5 Objednávka

Objednávka je nejrozsáhlejší entitou tohoto projektu. Uchovává v sobě velké množství informací. V první řadě u sebe uchovává kontaktní informace dárce:

- **name, surname**: Jméno a příjmení dárce
- **email**: Kontaktní email dárce, na který se zašle email o vytvoření objednávky a následně o jejím úspěšném zaplacení.
- **mobile**: Telefonní kontakt na dárce.

Následně je u této entity vlastnost **buyingAsCompany**. Reprezentuje informaci o tom, zda dárce poskytuje dar jako firma. Pokud tato vlastnost nabývá pravdivé hodnoty, poté jsou vyžadovány další informace:

- **title**: Název společnosti.
- **ico**: Slouží pro uchování identifikačního čísla fyzické nebo právnické společnosti.
- **dič**: Daňové identifikační číslo.

U této entity je další vlastnost uchovávající pravdivostní hodnotu a tou je **wantsCertificate**. Pokud nabývá hodnoty true, jsou vyžadovány tyto informace:

- **street_num**: Textový řetězec, který představuje název ulice a číslo popisné.
- **city**: Obsahuje název města/obce.
- **zipCode**: Poštovní směrovací číslo.

Zbylé informace u této entity jsou

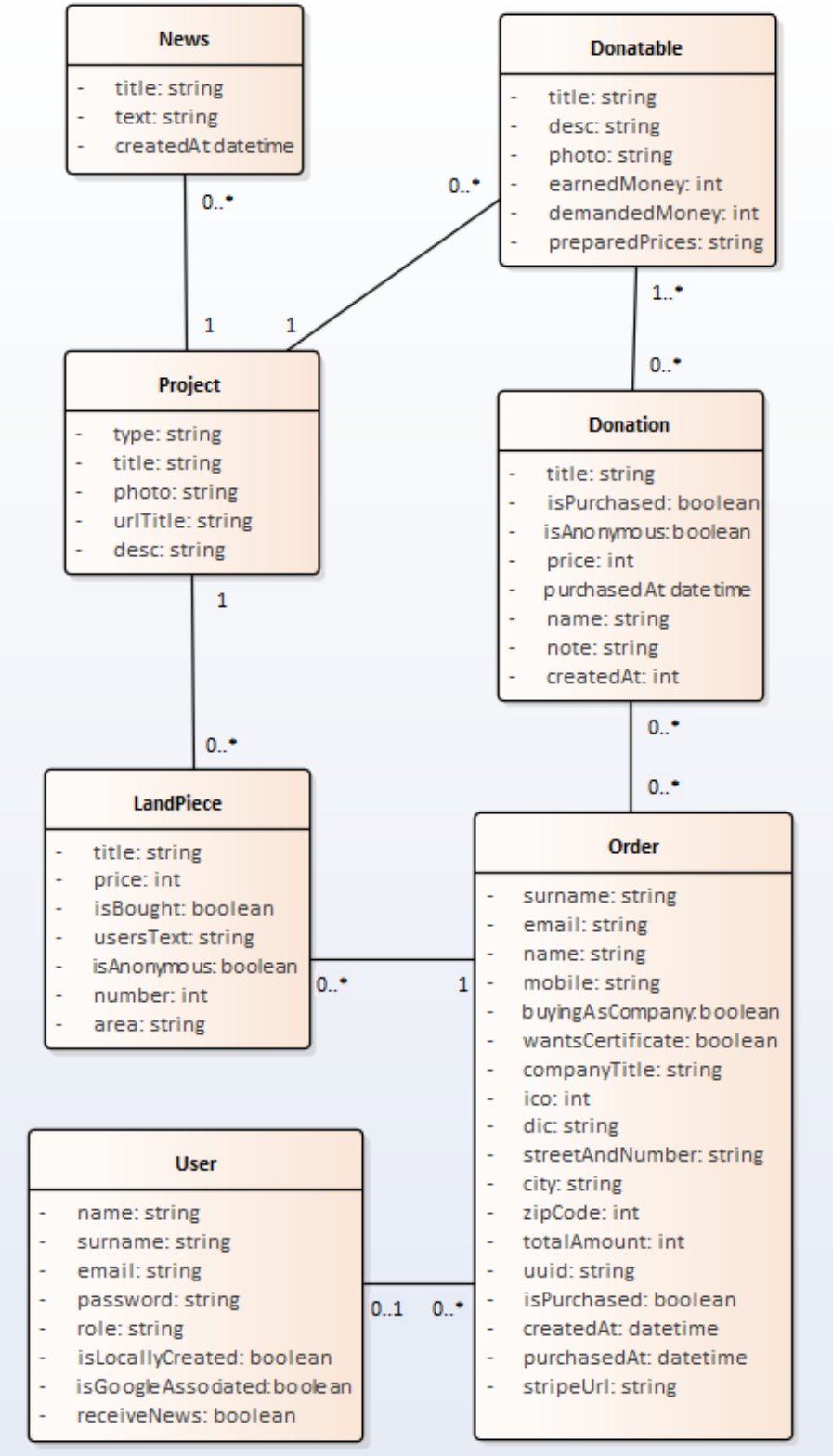
- **paymentMethod**: Určuje typ platební metody (kartou/ bankovním převodem).
- **donations**: Představuje pole ID darů, aby byla zajištěna vazba mezi objednávkou a darem.
- **landPieces**: Jde o pole ID částí pozemku.
- **totalAmount**: Číselná hodnota definující celkovou částku v dané objednávce.
- **uuid**: Náhodně vygenerovaný kód v šestnáctkové soustavě, který je potřeba při zobrazení detailu objednávky přes odkaz bez nutnosti registrace.
- **isPurchased**: Vlastnost, která uvádí, zda je již objednávka zaplacena.
- **purchasedAt**: Datum a čas zaplacení objednávky.
- **createdAt**: Datum a čas vytvoření objednávky.

- **stripeUrl**: Odkaz na platební bránu, který je pro objednávku vygenerovaný. Slouží pro opětovné přesměrování na platební bránu po jejím neúspěšném dokončení.

5.2.6 Segment pozemku

Část pozemku je entitou, která byla použita pro první verzi interaktivní mapy. V další verzi se již tato entita nevyskytuje. Obsahuje vlastnosti:

- **title**: Nadpis segmentu.
- **number**: Číslo označující segment. Jedinečné pro každý kus.
- **price**: Cena segmentu.
- **usersText**: Vlastní poznámka k segmentu, kterou může dárce při darování sdělit.
- **isAnonymous**: Určuje, zdali je dar na segment anonymní.
- **isBought**: Zdali je segment již nakoupený nějakým dárcem.
- **area**: Označuje oblast segmentů.



Obr. 1 Model tříd

5.3 Návrh průběhu darování na projekt o pozemku

5.3.1 Myšlenka prvního řešení

První myšlenka řešení nakonec byla nahrazena myšlenkou druhou, přesto si prošla fázemi zpracování projektu diplomové práce, a tudíž je popsána i v této práci. Obrázky v této kapitole jsou pořízeny až po implementaci. I když tato kapitola je o návrhu, pro doplnění představy k holému textu jsou tyto obrázky vyhovující.

Zmiňovanou jedinečností a odlišností je následující myšlenka. Zobrazit pozemek graficky v prohlížeči, rozčlenit ho na části, tyto části nacenit a umožnit jejich jednotlivý nákup. To je ten odlišný a jedinečný přístup oproti ostatním projektům, kde dárce buď zvolí jednu z předpřipravených částek nebo přispěje vlastní. Po přispění na danou část pozemku bude mít dárce možnost připsat vlastní poznámku k tomuto dílku, pokud bude chtít zůstat v anonymitě, i tato možnost zde bude zohledněna. Pokud jde tedy o realizaci této nové myšlenky, je potřeba se vypořádat vykreslením pozemku a jeho částí na obrazovkách uživatelů. Cílem školy je získat celkem 4 000 000 Kč pro nákup pozemku. Bylo tedy potřeba rozdělit tuto částku na nějaké dílčí částky. Toto rozdělení celkové ceny přímo souvisí s počtem jednotlivých dílků, které vzniknou po segmentaci pozemku. Následně je zobrazena tabulka, jakým stylem se pozemek rozčlenil, na kolik části a o jakých cenách. Byla tvořena společně s ředitelkou školy na základě jejich představ.

typ	počet částí	cena části	hodnota částí
1	2000	300	600000
2	7000	100	700000
3	3000	200	600000
4	2000	500	1000000
5	1000	1000	1000000
6	10	10000	100000
	15010		4000000

Tabulka 1 Informace o sekcích pozemku, 1. část

Z tabulky zobrazené výše je patrné, že pozemek se rozčlenil celkově na 15 010 segmentů. Jedná se o 6 různých druhů částí, které budou dostupné k přispění. Součtem cen všech částí je potřebných 4 000 000 Kč ke koupi pozemku. Dále

všechny části musí být nějak graficky interpretovány jako nějaký 2D objekt. V následující tabulce je znázorněno rozvržení jednotlivých sekcí, jejich pojmenování a grafická reprezentace na pozemku.

typ	barva	tvar	popisek
1	červený	kruh	Pro dobrý skutek
2	oranžový	trojúhelník	Pro děti z kasičky
3	zelený	čtverec	Pro babičky a dědy z důchodu
4	žlutý	kruh	Pro pracující s empatickým cítěním
5	modrý	obdélník	Pro rodiče a přátelé školy
6	zlatý	hvězda	Pro firmy

Tabulka 2 Informace o sekcích pozemku, 2. část

Pokud se tedy shrnou obě tabulky dohromady a interpretuje se například druhý typ částí, interpretace zní následovně. Počet částí druhého typu je 7000, každá část z nich má cenu 100,-, celkem za všechny části je možné vybrat 700000,-. Každá část bude graficky vyobrazena jako 2D trojúhelník oranžové barvy. Popisek shrnující celou sekci nese název „Pro děti z kasičky“. Nyní je objasněno, jakým stylem je pozemek rozčleněn na části, kolik by měly jednotlivé části stát a jak mají vypadat. Představa je taková, zobrazit pozemek a před ním jednotlivé objekty/části pozemku, které by šly zaklikávat. Než se však vykreslí jednotlivé objekty, nejdříve by se vykreslovaly sekce s popiskem a cenou, kde by si uživatel vybral, do které sekce se chce dostat na základě toho, kolik je ochotný darovat. Na obrázku je názorně předveden druhý typ částí.

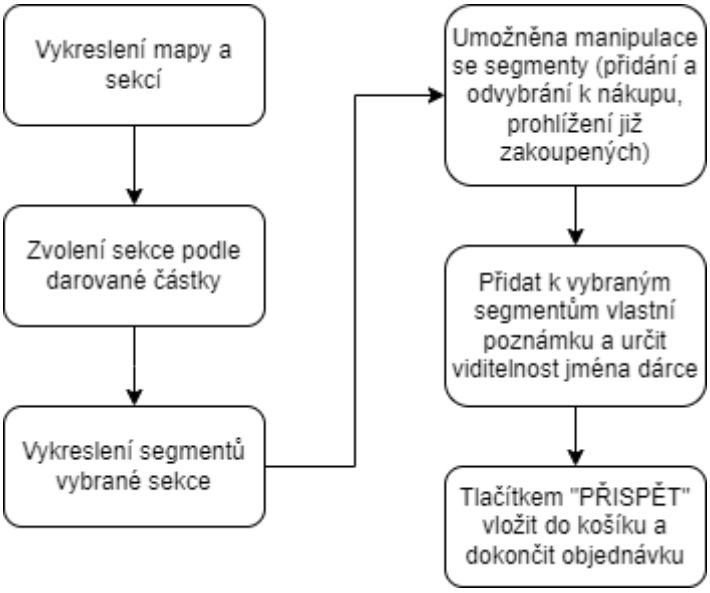
Obr. 2 Vizualizace sekce při najetí kurzorem myši

Pokud se kurzorem myši přejíždí přes pozemek, zviditelňují se jednotlivé sekce. Po jejich rozkliknutí by bylo vhodné přejet kamerou blíže k dané části pozemku, aby se uživatel nemusel sám kolečkem myši přibližovat k dané sekci. Celý proces byl více uživatelsky přívětivý. Další úlohou po rozkliknutí by bylo načtení informací o všech dílčích částech sekce, aby bylo jasné, které jsou již zakoupené, které jsou dostupné, a jaká data u již zakoupených částí zobrazit. Informace jako jméno, příjmení, pokud si je tedy dárce přeje zveřejnit. Dále vlastní poznámka přispěvatele.

Na následujícím obrázku je znázorněn pohled na vykreslené segmenty po rozkliknutí sekce – Pro děti z kasičky.

Obr. 3 Segmenty sekce

Výše na obrázku jsou znázorněny již zakoupené části oranžovou barvou. Uživatel se s použitím myši přiblíží a nasměruje na část, podle jeho volby. Je zde zobrazeno 7000 trojúhelníků.



Obr. 4 Flowchart: Proces přispívání na pozemek – první verze

Nedostatky prvního řešení

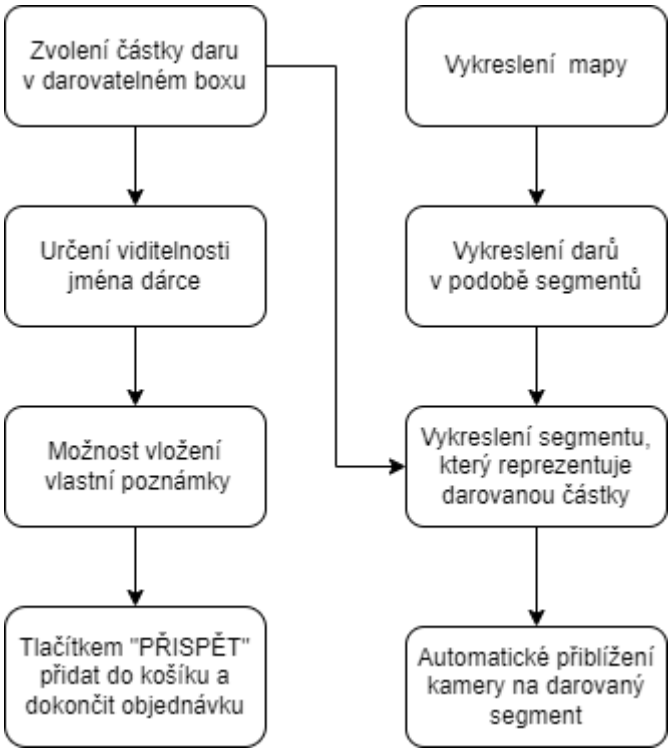
Po částečné implementaci, kdy bylo možné alespoň z nějaké části nechat tento přístup otestovat, se přišlo na některá omezení. Tím hlavním omezením bylo, že množina segmentů v každé sekci byla konečná. Tudíž jakmile by se vyprodaly všechny části sekce, nebylo by již možné částku sekce přispět. To je nejhlavnější důvod, proč se muselo přejít k jinému přístupu. V této verzi není možné přispět vlastní libovolnou částku. Dále pro některé uživatele byla obtížnější interakce s mapou, zejména na mobilních zařízeních.

5.3.2 Druhý návrh procesu přispívání

V druhém a konečném návrhu chtěl zadavatel zachovat v určité formě interaktivní mapu. Mapa v tomto způsobu řešení zůstává, ale s určitými změnami. K přispění na pozemek již není potřeba s mapou vůbec interagovat, stále ale zůstává přítomna jako vizualizace již poskytnutých darů a ukázka toho, jak moc ovlivní dárcem chtěná částka na celkové ploše pozemku. Průběh darování by vypadal následovně. U projektu s vybíráním na pozemek by se nacházel „darovatelný box“ jako u ostatních projektů. Dárce si zvolí částku, kterou chce darovat. Ihned poté se tato částka vykreslí ve formě čtverce/obdélníku (podle velikosti částky) na mapě, a pro větší uživatelskou přívětivost se kamera ve scéně nasměruje na tuto vykreslenou část. Uživatele by tento způsob grafické vizualizace mohl motivovat k darování vyšší

částky. Po dokončení darování by zakoupená část byla zobrazena všem uživatelům. V této variantě návrhu již není potřebná entita Segment pozemku. Části jsou vykresleny na základě entity Dar. Pro vykreslení části pozemku postačuje informace z této entity, a tou je cena, jméno dárce a vlastní poznámka k daru.

Není potřeba si hlídat informace, který segment je zakoupený a v jaké sekci se nachází. Oproti předchozímu návrhu jsou viditelné pouze ty části, které jsou již zakoupené. Při najetí kurzorem myši nad část pozemku se vykreslí informace o daru. Tento způsob řešení odstraňuje nedostatky z předchozího návrhu. Dárce již není omezen částkou, kterou může darovat. Tato alternativa je otevřena všem možnostem. Když se uvedou extrémní případy jako 1 dar o plné částce pozemku či všechny dary budou např. o velikosti 100 Kč, obě možnosti jsou reálně možné a tento návrh jejich provedení neomezuje. Problém s nutností interakce s mapou, aby bylo možné darovat, je také odstraněn. Darování je realizováno pomocí rozhraní ve formě tlačítek v „darovatelném boxu“.



Obr. 5 Flowchart: Proces přispívání na pozemek - druhá verze

6 Implementace a nasazení

6.1 Použité technologie

Následující tři kapitoly obsahují popis technologií, které autor práce pro vývoj webové aplikace používá.

6.1.1 HTML, CSS a SASS

Základem tvorby webových stránek je znalost značkovacího jazyka HTML, který určuje obsah stránky. Tvoří jej párové a nepárové tagy, ke kterým je možno uvést různé atributy. Elementům webové stránky je potřeba dodat vzhled na míru. K popisu vzhledu slouží technologie CSS. Elementům lze pomocí selektorů nastavit např. barvu, rozměry, velikost písma, animaci atd. Pro efektivnější práci při vytváření vzhledu je využita nadstavba SASS, která poskytuje funkcionality jako jsou např. zavádění proměnných, vnořování selektorů a jiné.

6.1.2 Koncept MERN

Jedná se o koncept čtyř technologií, jejichž prostřednictvím se vyvíjejí osvědčené a moderní webové aplikace [6]. Programátorovi pro použití těchto technologií stačí znalost jazyka JavaScript [7].

MongoDB

Představuje NoSQL databázi, která je snadno škálovatelná a vysoce flexibilní. Umožňuje ukládat data ve formátu BSON, který je velmi blízký formátu JSON. [8] Metody pracující s databází jsou většinou asynchronní. Je tedy potřeba využít jednu z funkcionalit jazyka JavaScript pro volání asynchronních metod. Dá se přistupovat dvěma způsoby. Buď s použitím příslibů nebo syntaxe async-await. V tomto projektu se využívají oba způsoby řešení.

Express

Rychlý a nenáročný webový framework pro Node.js. [9] Poskytuje mnoho užitečných funkcí, jako je vytváření REST API, routování a podporu pro integraci šablonovacích enginů.

React

React je knihovna pro JavaScript, pomocí které lze vytvářet uživatelská rozhraní. [10] Umožňuje vytvářet prezentační vrstvu webu složenou z individuálních dílků nazývané komponenty. React je velmi rychlý díky své vlastnosti používat virtuální DOM (Document Object Model). Elementy stránky se definují pomocí jazyka JSX, který umožňuje vytvářet HTML elementy přímo v kódu JavaScriptu.

Node.js

JavaScriptový runtime pro běh serverových aplikací. [11] Umožňuje spouštět kód aplikace přímo na zařízení mimo webový prohlížeč. [12] Právě v této serverové aplikaci je využit již výše zmíněný framework Express.

6.1.3 Three.js a React Three Fiber

V této kapitole jsou popsány technologie pro vykreslení interaktivní mapy pro projekt „Kup si svoji část pozemku“. V první verzi řešení je použito **Three.js**. Jedná se o knihovnu využívající WebGL pro zobrazení grafiky. Umožňuje vytvářet a manipulovat s geometrickými tvary, materiály, texturami a dalšími prvky, které jsou pro tvorbu 3D grafiky nezbytné. Knihovna také poskytuje velké množství funkcí, jako jsou například kamera, pohyb myší, přibližování a jiné.

React Three Fiber je nadstavbou nad Three.js. Jedná se o knihovnu, která umožňuje vytvářet interaktivní grafiku v Reactu. Tato knihovna využívá síly Three.js a vychytávek Reactu, a společně tvoří velmi populární, zjednodušený a moderní přístup k tvorbě grafiky ve webových aplikacích.

6.2 Struktura projektu

Vytvoření projektu frontendové části se provádí příkazem „**create-react-app** název-aplikace“. Tento skript vytvoří jednoduchou SPA (Single Page Application) a poskytuje základní stavební kámen, na kterém lze dále stavět. Pro inicializaci backendové části aplikace postačilo spuštění příkazu „**npm init“** v kořenové složce projektu. Pomocné funkce, moduly, knihovny či frameworky je možné, jak na fronetend, tak na backend, doinstalovat použitím příkazu „**npm install** název-knihovny“.

6.3 Zabezpečení

Zabezpečení aplikace, validace dat na obou částech aplikace, autentizace a autorizace jsou témata, která jsou v této kapitole popsána. Pokud jde o zabezpečení komunikace mezi serverem, klientem a jednotlivými částmi systému, je použit protokol SSL, který tuto komunikaci šifruje. Dále pro vyšší zabezpečení aplikace a nastavení hlaviček HTTP je využit balíček **helmet**. Pro zajištění přístupnosti k API pouze z domény nasazené webové aplikace, jsou pomocí balíčku **cors** správně nakonfigurovány přístupy. Nyní z testovacích účelů je mezi přístupy povolena i lokální doména localhost.

6.3.1 Validace vstupních dat

V této kapitole je popsané řešení validace vstupních dat jak na frontendové části, tak na backendu. Je potřeba validovat vstupní data na klientovi např. z důvodu, aby uživatel věděl, zda má správně vyplněný formulář a bez správného vyplnění nepůjde běžným způsobem odeslat. Avšak není to nemožné. Proto validace na frontendu nestačí a je spíše dekorativním či pohodlným prvkem pro uživatele, než aby sloužila pro zajištění bezpečnosti chodu aplikace. Z toho důvodu je validace na backendové části mnohem důležitější, protože tam se již bezpečnosti dá docílit.

Na straně klienta

Validace hodnot vstupů formulářových prvků na frontendu je řešena pomocí vlastních validačních funkcí, které jsou definovány v souboru utils/**validators.js**. Jsou zde přítomny validační funkce pro email, maximální či minimální délku řetězce, maximální či minimální číselnou hodnotu, nebo zda vstup vůbec nabývá nějaké hodnoty. Použití těchto funkcí je realizováno ve složitějším procesu celkového řízení formulářů, a tak podrobnější informace o jejich použití je zmíněno v kapitole – 6.4.4 Řešení formulářů.

Na straně serveru

K validaci vstupních dat na backendu je použit modul **express-validator**. Z tohoto modulu je využita funkce **check**, které je jako parametr předáno ID vstupu. Následně

se dají řetězově navázat jednotlivé požadavky na hodnotu vstupu, které má splňovat.

Zde je uveden příklad validace při pokusu přihlášení uživatele, který se nachází v souboru validations/**authVlds.js**:

```
exports.postLogin = [
  check("email").notEmpty().isEmail().withMessage("Zadejte email"),
  check('password').notEmpty().withMessage("Zadejte heslo")
];
```

Obr. 6 Kód: validace na serveru

Z obrázku lze vyčíst, že se validuje vstup s ID hodnotou „email“, tato hodnota nesmí být prázdná a zároveň musí splňovat parametry emailu. Pokud tomu tak není, je frontendu zaslaná chybová hláška „Zadejte email“. Dále se kontroluje vstup „password“. Pokud ten nenabývá žádné hodnoty, je zaslána chyba s textem „Zadejte heslo“.

Tímto způsobem jsou ošetřeny všechny endpointy backendu, na které je očekáván přísun vstupních dat.

6.3.2 Autentizace a autorizace

Tato kapitola je jednou z nejdůležitějších, pokud jde o zabezpečení aplikace a jejich dat. Jedná se o procesy autentizace a autorizace. Autentizace verifikuje uživatele. Když se uživatel prokazuje, že je tím, za koho se vydává, např. vyplňuje přihlašovací informace do systému, následně systém provede proces autentizace uživatele a tím jej ověří. Průběh tohoto procesu je vysvětlený v kapitole 6.4.9. - Přihlášení

Autorizace je proces ověřování práv uživatele, při provádění určité operace. Například pokud bude chtít uživatel smazat v této aplikaci projekt, musí na to mít přidělená práva. Pokud tedy přijde požadavek na API, které tuto funkcionalitu poskytuje, je požadavek nejdříve poslán do middlewaru **readToken.js**. Zde je z hlavičky požadavku vytažen autorizační token. Jedná se o zabezpečený string, který v sobě uchovává zašifrované informace o uživateli. Tento token je zde dešifrován a jsou z něj vytažená data o uživateli. Práci s tokenem usnadňuje modul **jsonwebtoken**. Ověření tokenu a jeho dešifrovanou podobu lze získat zavoláním

metody **verify**, dostupnou z tohoto balíčku. Po úspěšném dešifrování je požadavek přesměrován do middlewaru **isAdmin.js**. Zde se ověří, zda práva uživatele, která jsou získána a dešifrována z tokenu se rovnají administrátorským právům. Pouze při kladném vyhodnocení této podmínky požadavek proputuje do dalšího middlewaru, v tomto případě už do konečného middlewaru, který zajišťuje smazání projektu. V opačném případě je zapsána odpověď požadavku s chybovou hláškou "Nemáte administrátorská práva".

6.4 Aplikační vrstva

V této kapitole jsou popsány všechny stránky aplikace, které byly vytvořeny. Nejprve je popsán grafický layout, který mají všechny stránky této SPA společné a následně jsou rozebrány jednotlivé stránky s jejich funkcionalitami a jejich řešením. Pokud se autor práce odkazuje na adresu na frontendu, používá označení F, pod kterým se nachází adresa serveru. To samé s backendovou částí, ta má označení B.
Příklady:

- **F**/kosik je ekvivaletní s **http://localhost:3000**/kosik
- **B**/api/auth/login reprezentuje **http://localhost:5000**/api/auth/login

6.4.1 Implementace modelů

V této kapitole autor práce popisuje, jakým způsobem řeší implementaci modelů v Node.js. Pro zvýšení efektivity práce je použit modul **mongoose** [13]. Tento balíček poskytuje řešení, založené na schématech, k modelování aplikačních dat a mnoho dalších funkcí, aby se v kódu snáze s daty pracovalo.
Nejprve je potřeba z balíčku mongoose importovat třídu **Schema**. Následně je zapotřebí inicializovat objekt této třídy a jako parametr do konstruktoru uvést objekt, který definuje schéma budoucího modelu. Po definici schématu je na jeho základě možno vytvořit model. Toho se docílí zavoláním metody **model,** která je dostupná z balíčku mongoose. Tato metoda vyžaduje dva parametry. Prvním je název modelu a druhým samotné schéma, které modelu udává strukturu. Kód naznačující přípravu schématu pro model Sbírka:

```
const donatableSchema = new Schema({
    title: {
        type: String,
        required: true,
    },
    desc: {
        type: String,
        required: true
    },
```

Obr. 7 Kód: náznak vytváření schématu v mongoose

Ukázka vytvoření modelu:

```
mongoose.model('Donatable', donatableSchema)
```

Obr. 8 Kód: vytvoření modelu v mongoose

Tímto způsobem je řešeno vytvoření všech potřebných modelů aplikace.

6.4.2 Dostupné REST API

V této kapitole jsou popsány veškeré endpointy backendové části aplikace. Jsou zde popsány API manipulující s dokumentem o uživateli. Následně ke kterým má právo pouze uživatel, poté administrátor a nakonec všechna ostatní otevřená API.

API pro správu uživatele

Endpointy tohoto druhu se nachází v souboru **authRoutes.js**, který obsahuje níže popsané endpointy. Všechny API mají prefix v relativní URL **/api/auth.**

Rozhraní s cestou **/login**, za použití metody **post**, obstarává přihlašovací proces.

Rozhraní s cestou **/login-google**, za použití metody **post**, obstarává přihlašovací proces uživatele přes službu Google.

Rozhraní s cestou **/register-user**, za použití metody **post**, obstarává registrační proces uživatele.

Rozhraní s cestou **/register-admin**, za použití metody **post**, obstarává registrační proces administrátora.

API pro běžného uživatele

I když je prozatím v této sekci pouze jeden endpoint, i tak pro případnou rozšiřitelnost aplikace je vytvořen separátní soubor **userRoutes.js**. Všechny API mají prefix v relativní URL **/api/user.**

Rozhraní s cestou **/orders-by-user-email**, za použití metody **get**, vrací všechny objednávky uživatele podle emailu z tokenu.

API pro administrátora

Endpointy pro administrátora jsou obsaženy v souboru **adminRoutes.js**. Veškeré API mají v relativní URL prefix **/api/admin**.

Rozhraní s cestou **/create-project**, za použití metody **post,** vytváří nový projekt.

Rozhraní s cestou **/edit-project/:projectId**, za použití metody **patch**, aktualizuje projekt podle ID projektu.

Rozhraní s cestou **/project/delete/:projectId**, za použití metody **patch**, aktualizuje atribut deleted na hodnotu true, u dokumentu projektu podle ID projektu.

Rozhraní s cestou **/create-news/:urlTitle**, za použití metody **post,** vytváří novou aktualitu pro projekt na základě parametru urlTitle.

Rozhraní s cestou **/edit-news/:newsId**, za použití metody **patch**, aktualizuje aktualitu podle ID aktuality.

Rozhraní s cestou **/news/delete/:newsId**, za použití metody **patch**, aktualizuje atribut deleted na hodnotu true, u dokumentu aktuality, podle ID aktuality.

Rozhraní s cestou **/create-donatable/:projectId**, za použití metody **post,** vytváří novou sbírku pro projekt na základě ID projektu.

Rozhraní s cestou **/edit-donatable/:donatableId**, za použití metody **patch**, aktualizuje sbírku podle ID sbírky.

Rozhraní s cestou **/donatable/delete/:donatableId**, za použití metody **patch**, aktualizuje atribut deleted na hodnotu true, u dokumentu sbírky, podle jejího ID.

Otevřené API

Endpointy v této části jsou veřejné a dostupné bez autentizace. Nachází se v souboru **sharedRoutes.js** a všechny endpointy mají v relativní URL prefix **/api**.

Rozhraní s cestou **/projects/:projectId**, za použití metody **get**, vrací projekt podle jeho ID.
Rozhraní s cestou **/projects-by-url-title/:urlTitle**, za použití metody **get**, vrací projekt podle parametru urlTitle.
Rozhraní s cestou **/projects**, za použití metody **get**, vrací všechny projekty.
Rozhraní s cestou **/news/:projectId**, za použití metody **get**, vrací aktuality podle ID projektu.
Rozhraní s cestou **/news-item/:newsId**, za použití metody **get**, vrací aktualitu podle jejího ID.
Rozhraní s cestou **/donatables-by-project-id/:projectId**, za použití metody **get**, vrací sbírky podle ID projektu.
Rozhraní s cestou **/donatables/:donatableId**, za použití metody **get**, vrací sbírku podle jejího ID.
Rozhraní s cestou **/donations/:donatableId**, za použití metody **get**, vrací dary na danou sbírku, podle ID sbírky.
Rozhraní s cestou **/create-order**, za použití metody **post**, provádí proces vytvoření objednávky.
Rozhraní s cestou **/order/:orderId**, za použití metody **get**, vrací dary na danou sbírku podle ID sbírky.

6.4.3 Vlastní stylování

Pro stylování prvků v této aplikaci nebylo využito žádných pomocných knihoven, jako je např. Bootstrap [14]. Responzivita i stylování vzhledu všech elementů stránky jsou vytvořeny ručně autorem práce.
Jelikož je vyvíjená aplikace zaštítěna organizací Diakonie, bylo potřeba dodržet jejich korporátní identitu a dodržovat určité požadavky, jako např. použití určitého fontu, přesně stanovených barev, správné zacházení s logem společnosti a jiné.
Stylování v této práci není nijak hlouběji rozebíráno, protože se jedná o základní problematiku tvorby webových stránek.

6.4.4 Řešení formulářů

Formuláře jsou nedílnou součástí téměř každé webové aplikace a ani tato není výjimkou. V této aplikaci je velké množství formulářů např. pro přihlášení, registraci, vytvoření projektu, vyplnění údajů k objednávce a tak dále. Bylo potřeba se jednotně vypořádat s jejich logikou a procesem zpracování vstupů. Autorem práce byl tedy vytvořen vlastní háček **g-form-hook.js**, který takové funkce řeší. Pro jeho inicializaci je potřeba vložit jako parametr objekt na základě této šablony.

```
const formDataLayout = {
  parts: {
    partId: {
      required: false,
      partIsValid: false,
      inputs: {
        inputId: {
          value: "",
          isValid: "",
          isTouched: false,
        },
      },
    },
  },
  formIsValid: false,
};
```

Obr. 9 Kód: šablona pro formulářová data

Vysvětlení jednotlivých položek šablony

parts: Je zde proto, aby toto vlastní řešení šlo použít i pro případ, kdy je potřeba mít formulář rozčleněn do sekcí. Například formulář pro vyplnění údajů pro dokončení objednávky obsahuje 4 sekce: kontaktní údaje, firemní údaje, údaje pro certifikát a výběr platební metody. Tento objekt může mít libovolné množství klíčů s různými partId, které reprezentují dané sekce.

partId: Reprezentuje danou sekci formuláře. Uchovává si v sobě pravdivostní hodnotu **required**, která uvádí, zda je vyplnění údajů v dané sekci povinné a zda se budou validovat vstupy v této sekci, když se bude validovat celý formulář. Hodnota **partIsValid** udává, jestli jsou všechny vstupy sekce validní. Dále pod klíčem **inputs** může být libovolné množství objektů s odlišným klíčem **inputId**, které v sobě uchovává informace o vstupních prvcích sekce. Obsahují **value**, tedy hodnotu prvku.

Dále pravdivostní hodnotu **isValid**, která určuje, zda je hodnota prvku validní a proměnná **isTouched**, která určuje, zda se uživatel již pokusil zadat hodnotu prvku. Dá se využít pro zobrazení chybové hlášky v době, kdy se uživatel pokusil vyplnit vstupní prvek, ale neprošel validací a zadávání ukončil.

Dostupné funkce vlastního háčku g-form-hook.js

V této části jsou popsány funkce, které tento háček vrací komponentě, která ho zavolala

- **inputChange**: Jedná se o funkci, která se volá při změně hodnoty formulářového prvku, aby se hodnota promítla do dat v háčku. Je potřeba uvést jako parametry funkce partId, inputId, value a validators v tomto pořadí. Háček na základě parametrů partId a inputId zjistí, kam má novou hodnotu prvku uložit. Provede validaci nové hodnoty se všemi validačními funkcemi v poli validators. Na základě výsledku stanoví, zdali je nová hodnota validní, a nakonec aktualizuje validitu dané sekce i celého formuláře.
  Příklad**:**

```
onChange={(e) => {
  inputChange('contactInfoPart', 'email', e.currentTarget.value, [VALIDATOR_EMAIL()])
}}
```

Obr. 10 Kód: změna vstupu formulářového prvku

- **touchHandler**: Funkce, která se zavolá při opuštění pozornosti prvku, tzn. při události onBlur. Přijímá jako parametry partId a inputId. Nastavuje konkrétnímu vstupu hodnotu isTouched na true.
- **setRequired**: Tato funkce přijímá jako parametry partId a pravdivostní hodnotu true/false. Na základě parametrů nastavuje hodnotu **required** dané sekci.

Háček spolu s těmito funkcemi ještě vrací objekt **formState** s aktuálními daty o formuláři.

Pro znovu použitelnost formulářů bylo autorem práce vytvořeno několik dílčích komponent.

BForm.js

Tato komponenta vrací element form se všemi vnořenými potomky a ohraničuje celý formulář.

BFormPart.js

Komponenta vrací element div s nadpisem části díky props a všechny vnořené vstupní prvky, které této sekci náleží.

BInput.js / BTextarea.js

Toto je komponenta, která reprezentuje vstupní formulářový prvek input/textarea, jeho nadpis a oboje sdružené v jednom div elementu. Parametry props, které přijímá, jsou uvedeny v obrázku na konci této kapitoly.

BSubmit.js

Komponenta vracející element button pro odeslání formuláře. Umožňuje odeslání pouze, pokud je celý formulář validní. Validita formuláře i funkce, která se má po kliknutí vykonat, jsou potřeba této komponentě předat pomocí props.

```
<BForm classNames='column'>
    <BFormPart title="Přihlašovací údaje">
        <BInput
            title="Email"
            input={formState.parts.loginPart.inputs.email}
            partId="loginPart"
            inputId="email"
            validators={[VALIDATOR_REQUIRE(), VALIDATOR_EMAIL()]}
            error="Prosím, zadejte email"
            inputChange={inputChange}
            touchHandler={touchHandler}
        />
        <BInput
            title="Heslo"
            input={formState.parts.loginPart.inputs.password}
            partId="loginPart"
            inputId="password"
            validators={[VALIDATOR_REQUIRE()]}
            error="Prosím, zadejte heslo"
            inputChange={inputChange}
            touchHandler={touchHandler}
            secret={true}
        />
    </BFormPart>
    <BSubmit isValid={formState.formIsValid} onClick={postLoginHandler}>Přihlásit</BSubmit>
</BForm>
```

Obr. 11 Kód: příklad vytvoření formuláře v JSX

6.4.5 Zasílání požadavků

V tomto projektu probíhá komunikace mezi frontendovou a backendovou částí pomocí http požadavků. Velké množství komponent posílá tyto požadavky za různými účely. Pro jejich snadnější zasílání a pro možnost zobrazování stavu požadavku si autor práce vytvořil vlastní háček, tzn. custom hook. Tento háček dokáže vyhodnotit, zda nastala nějaká chyba a umí se s chybou vypořádat. Samotný háček se nachází v souboru **http-hook.js** ve složce hooks. Po jeho inicializaci vrací komponentě, která si ho vyžádala, tyto prvky:

- **sendRequest**: Funkce, která přijímá jako parametry URL adresu, případně metodu http požadavku (get, post, patch, delete), tělo požadavku a hlavičky. Tato funkce implementuje rozhraní FetchAPI, kterým moderní prohlížeče disponují.
- **error**: Jedná se o proměnnou, která se po neúspěšném vyřízení požadavku naplní chybovou hláškou ze serveru. Tu lze poté zobrazit klientovi v prohlížeči například pomocí vyskakovacího okna.
- **clearError**: Funkce, která vynuluje proměnnou error. Je velmi užitečnou pro zavření vyskakovacího okna s chybou, pokud se vyskakovací okno zobrazuje pouze v případě, když nabývá proměnná **error** nějaké neprázdné hodnoty.
- **isLoading**: Pravdivostní proměnná udávající stav zasílání požadavku. Nabývá hodnoty true pouze od doby zaslání požadavku do přijetí jeho odpovědi. V tomto čase je vhodné informovat klienta o čekání na odpověď například pomocí drobné načítací animace.

6.4.6 Rozložení obsahu grafického rozhraní

Na každé stránce se vykresluje záhlaví, obsah dané stránky a zápatí. V této kapitole je rozebráno pouze záhlaví a zápatí. Obsahy stránek mají své samostatné kapitoly.

Záhlaví

Definice záhlaví se nachází v souboru **Header.js**. Tato komponenta se stará o vykreslení loga společnosti a jednotlivých položek navigace.

Obr. 12 Záhlaví stránky

Za použití vestavěné funkce **useState**, kterou poskytuje sám React od verze 16.8, si uchovává informace o tom, zda má zobrazit boční menu, které je připravené pro zařízení s menší šířkou obrazovky (mobily, tablety). Toto menu se zobrazí stisknutím tlačítka MENU a opětovně skryje kliknutím libovolně mimo vyskakovací menu, tedy na zašedlou část obrazovky. Tlačítko MENU se zobrazí až při dostatečně malé šířce obrazovky, přičemž se zároveň zakryjí položky pro širší obrazovku.

Obr. 13 Záhlaví stránky na mobilní obrazovce

Obr. 14 Mobilní navigace

Přesměrování mezi stránkami je zajištěné komponentami **Link** či **NavLink**. Tyto komponenty jsou součástí modulu **react-router-dom**, který je potřeba doinstalovat. Těmto komponentám je nutné přiřadit cestu, na kterou mají odkazovat, a to jako hodnotu k atributu **to**. Níže je ukázka v kódu, alespoň jedné z položek z menu.

```
<NavLink className="btn--secondary btn-small" to="/kosik">
  Košík
</NavLink>
```

Obr. 15 Kód: odkaz na stránku pomocí komponenty NavLink

Po klinutí na logo společnosti se uživatel dostane na domovskou stránku, tedy na F:/.

Zápatí

Implementace zápatí je v souboru **Footer.js**. V zápatí je zobrazen pouze element footer, uvnitř kterého je text „2023 Diplomová práce | Vytvořil: Jiří Kumprecht“

Směrovač stránek

Směrovač stránek rozhoduje, které komponenty se mají vykreslit na základě URL adresy. V případě této aplikace se o to stará modul react-router-dom, který díky komponentám Router, Routes a Route toto umožňuje. Tento seznam možností neboli rozcestník, je definován v souboru **App.js**. Zde je uveden příklad definice jedné položky rozcestníku.

```
<Route path="/kosik" element={<CartPage />} />
```

Obr. 16 Kód: položka v rozcestníku stránek

Je tedy zapotřebí definovat atribut **path**, tedy část URL adresy a atribut **element**, který přijímá komponentu, kterou má případně vyrenderovat.

6.4.7 Hlavní stránka

Obsah hlavní stránky se vykreslí, pokud se uživatel nachází na adrese F:/. Přesněji se vykreslí komponenta **Homepage.js**, která má za úkol vykreslit dvě následující komponenty.

AboutSection.js

V této sekci je uveden uvítací text této stránky motivující dárce k přispění daru. Dále obsahuje tlačítko s názvem „Chci pomoci“, které odkazuje na projekt „Kup si svoji část pozemku“, a to proto, že se jedná o projekt s nejvyšší prioritou.

Obr. 17 Obsah domovské stránky

ProjectsSection.js

V této komponentě jde především o zobrazení seznamu projektů, na které lze přispívat. Po jejím vykreslení zašle pomocí funkce sendRequest požadavek na **B:/api/projects** pro získání dat o projektech. Vykonání kódu po vykreslení komponenty obstarává vestavěná funkce Reactu **useEffect**. Pokud je přihlášen správce, je vykreslený ještě odkaz na založení nového projektu.

Po získání dat o projektech jsou data předána pomocí props do komponenty **Projects.js**. Pomocí props lze v Reactu předávat data vnořeným komponentám. Data zde mají formu pole objektů, kde pro každý z nich je vytvořena komponenta **ProjectItem.js**. Tato komponenta představuje jednotlivý projekt na uvítací stránce. Obsahuje název, obrázek projektu a je celá ohraničená elementem **Link**, aby po kliknutí na komponentu byl klient přesměrován na detail projektu. Potřebná data jako název, URL adresu obrázku a URL na detail projektu získává z props. Pro představu, jak ve skutečnosti vypadá hlavní stránka aplikace, slouží následující obrázek.

6.4.8 Registrace

V této kapitole je popsán průběh registrace jak klasického uživatele, tak administrátora.

Uživatel

Komponenta pro vykreslení obsahu registrační stránky se nazývá **SignupUserPage.js** a vykresluje se na adrese F:/registrace. Komponenta vykresluje formulář s titulkem „Uživatelské údaje“. Vstupní prvky formuláře nesou názvy Jméno, Příjmení, Email, Heslo a Heslo znovu. Každý z těchto prvků má jednoduchou validaci, aby byla zadaná alespoň nějaká hodnota. U prvku s názvem Email je vyžadováno, aby hodnota nabývala tvaru emailu. U prvku Heslo je vyžadováno, aby splňovalo bezpečnostní vzor a obsahovalo alespoň 8 znaků včetně velkého písmene, číslice a speciálního znaku. Po splnění těchto validačních požadavků je povoleno stisknutí tlačítka „Registrovat“. Toto tlačítko po stisknutí obsluhuje metoda **postRegisterHandler**, která obstarává vytvoření a zaslání požadavku na B/api/auth/register-user. Do těla požadavku vkládá všechny údaje z formuláře. Po přijetí požadavku na straně backendu jsou hodnoty v těle požadavku zkontrolovány validační funkcí postRegisterUser. Následně je požadavek poslán do middlewaru **postRegisterUser**. V první řadě je zapotřebí zjistit, zda uvedený email ze vstupních dat požadavku je evidovaný v databázi u některého z uživatelů. Toho se dá docílit zavoláním metody **findOne** na modelu User. Jako parametr se zde uvádí objekt, který reprezentuje filtr, podle kterého se vyhledávají data. Filtru je tedy předán email z těla požadavku. Pokud uživatel již existuje, je frontendu vrácena chyba s textem „Uživatel s tímto emailem již existuje“ a proces registrace se přeruší. Jestli je email dostupný, tak se pomocí funkce **hash** z balíčku **bcrypt** zašifruje heslo do zabezpečené podoby. Následně se ověří, zda byl tento email už někdy k vyvíjené aplikaci přihlášený přes službu Google. Pokud ano, aktualizují se pouze některé údaje o tomto uživateli. Pokud ne, je v databázi vytvořen úplně nový dokument o uživateli. V případě úspěšného dokončení procesu registrace na straně serveru je klientovi zaslána informace o tomto skutku. Frontend přesměruje uživatele na přihlašovací obrazovku pomocí funkce **navigate**, která je součástí balíčku **react-router-dom**. Nyní je proces registrace dokončen i na straně klienta.

Administrátor

Pro registraci administrátora je dostupné API B/auth/register-admin. Postup registrace na straně backendu je stejný jako pro registraci běžného uživatele kromě dvou rozdílů. Zaprvé u dokumentu administrátora je nastavena vlastnost s názvem role na hodnotu „admin“. Zadruhé na začátku middlewaru je v databázi ověřeno, zda již admin existuje. Pokud již existuje, vrátí API odpověď ve formě chyby s hláškou „Admin už existuje, není povoleno vytvářet více administrátorů“. Tímto ověřením je zabezpečeno, že administrátor bude pouze jeden. Pro jeho vytvoření není zřízený formulář, protože by se použil pouze jednou a vícekrát ne. Vytvořit účet administrátora lze například posláním požadavku na API z nástroje Postman [15].

6.4.9 Přihlášení

Stránka pro přihlášení se nachází na adrese F/prihlaseni. Komponenta **SigninPage.js** vykresluje formulář s titulkem „Přihlašovací údaje“. Vstupnímu prvku „Email“ je přiřazena validace na ověření, zda jeho hodnota reprezentuje email. U vstupního prvku „Heslo“ je vyžadován alespoň jeden znak. Za splnění těchto dvou podmínek je validní celý formulář a v té chvíli je zpřístupněno i tlačítko „Přihlásit“. Po stisknutí tohoto tlačítka se zavolá funkce **postLoginHandler**, která zašle data z formuláře metodou POST na adresu B/api/auth/login. Toto API je na serveru definováno v souboru authRoutes.js. Logika backendu nejdřív pošle data z formuláře k validaci a až po jejím úspěšném dokončení se spustí logika samotného přihlášení, která je implementována v metodě **postLogin** v souboru **authCtrl.js**.

Zaprvé je nutné zjistit, zda je uvedený email z těla požadavku evidovaný v databázi u některého dokumentu z uživatelů. K tomu se použije metoda **findOne** na modelu User. Pokud se v databázi mezi uživateli nenajde žádný dokument, je zpět frontendu zaslána chyba s hláškou „Email není registrovaný“. Pokud se dokument najde, uživatel s tímto emailem tedy existuje a může se pokračovat v ověřování. Nejprve se zjistí, zda byl uživatel registrován lokálně nebo pomocí služby třetích stran (Google). Pokud lokálně ještě vytvořen nebyl, je zaslána jako odpověď požadavku chyba s hláškou „Uživatel není registrovaný lokálně. Přihlaste se přes služby třetích stran (Google) nebo zaregistrujte tento email tlačítkem registrovat níže.“ Pokud je ale email registrovaný lokálně, pokračuje se další fází přihlášení, kterou je porovnání

hesel. Pro porovnání zašifrovaného hesla uživatele z databáze a čitelného hesla, zaslaného v těle požadavku, slouží funkce **compare** z balíčku **bcrypt**. Pokud se neshodují, zašle se frontendu chyba s textem „Nesprávné heslo“. Jestliže se shoda hesel úspěšně ověří, je vygenerován token zavoláním metody **sign** z modulu **jsonwebtoken**. Do tokenu jsou zašifrovány všechny informace o uživateli kromě hesla. Tokenu je z bezpečnostních důvodů nastavená expirační doba. Token spolu s dokumentem uživatele je poslán jako odpověď klientovi.

Klient odpověď zpracuje a zavolá metodu **login**, které jako parametry předá data o uživateli. Tato funkce je z vlastního háčku **auth-hook.js**. Tento háček poskytuje funkce pro přihlášení, odhlášení a s použitím funkce **useContext**, která je v Reactu dostupná, je možné data o uživateli předávat všem vybraným komponentám. Funkce login ukládá také data o uživateli do lokálního úložiště. Po úspěšném přihlášení je klient přesměrován na domovskou stránku F/.

Komponenta **SigninPage.js** ještě vykresluje element pro přihlášení přes službu Google a odkaz na registrační formulář, tedy na adresu F/registrace.

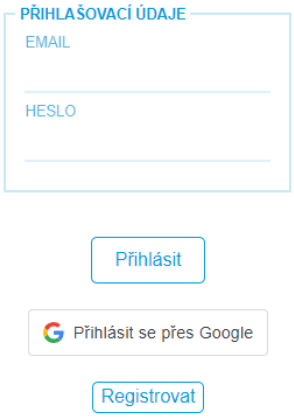
Přihlášení přes Google

Vyvíjená aplikace integruje přihlášení přes autentizační služby třetí strany, konkrétně služby Google. Pro tuto autentizaci se používá protokol OAuth 2.0. Jeho použití zvyšuje bezpečnost a umožňuje integraci mezi různými webovými službami či aplikacemi. Pro implementaci je potřeba mít účet od služby Google. S ním se přihlásit do administrátorského rozhraní Google Cloud, ve kterém je potřeba vytvořit nový projekt a správně službu nakonfigurovat. Uvést adresu aplikace, ze které je možné se přihlásit a také uživatelské účty, které se v rámci testování mohou přihlásit. Z testovacích účelů je prozatím uveden pouze http://localhost:3000/. Služba vygeneruje „clientID“, díky kterému je možno využívat API Googlu přímo z vyvíjené aplikace. Na přihlašovací stránce F:/prihlaseni je nejprve nutno inicializovat použití tohoto způsobu autentizace. Uvést „clientID“ id a funkci, která bude obsluhovat odpověď po pokusu autentizace. Je nutné také vykreslit tlačítko pro možnost přihlášení přes Google. Toho se docílí zavoláním metody renderButton, kde se jako parametry nastaví element DOMu a základní vlastnosti tlačítka.

```
google.accounts.id.initialize({
    client_id: process.env.REACT_APP_GOOGLE_CLIENT_ID,
    callback: handleCallbackResponse
})
google.accounts.id.renderButton(
    googleDivRef.current,
    { theme: 'outline', size: "large" }
)
```

Obr. 18 Kód: konfigurace pro přihlášení přes Google

Funkce **handleCallbackResponse**, která se spustí při získání odpovědi od Googlu po pokusu autentizace, v parametru získá zašifrovaný token. Ten je potřeba dešifrovat. Jeden ze způsobů je použít dešifrovací funkci **jwt_decode** ze stejnojmenného balíčku, který je potřeba doinstalovat. Po dešifrování jsou k dispozici již čitelná data o uživateli získaná z tokenu. Pro účely aplikace dostačují 3 informace: email, jméno, příjmení. Ty jsou předány jako parametry funkci **postLoginGoogle**, která tato data posílá požadavkem na B/api/auth/login-google. Server nejdříve data zvaliduje a následně je požadavek propuštěn do **postLoginUser_Google**. Zde se ověří, zda již uživatel s tímto emailem existuje v databázi. Pokud ne, je vytvořen a uložen nový dokument uživatele na základě dostupných informací. Následně proběhne klasický postup přihlašování, jako je generování tokenu a jeho zaslání na frontend. Jestli ale uživatel s daným emailem z požadavku již existuje, je pouze v jeho účtu nastavena hodnota isGoogleAssociated na true. Dále proběhne generování tokenu a jeho zaslání klientovi stejně jako v předchozím případě. Frontend pomocí funkce login přihlásí uživatele v SPA a uloží si token získaný z backendu. V posledním kroku přesměruje uživatele na hlavní stránku.



Obr. 19 Stránka s přihlášením

Ukázka vykresleného obsahu komponenty **SigninPage.js**.

6.4.10 Vytvoření projektu

Stránka pro vytváření projektu, která se nachází na adrese F/novy-projekt, je dostupná pouze pro administrátora. To je zajištěno následujícím kódem.

```
{auth.role == Roles.ADMIN && <Route path="/novy-projekt" element={<CreateProjectPage />} />}
```

Obr. 20 Kód: vložení položky do rozcestníku na základě práv uživatele

Tento řádek kódu porovnává roli přihlášeného uživatele, jestli je rovna administrátorským právům. Pouze v případě úspěšného splnění podmínky je tato stránka zařazena do seznamu v rozcestníku. Samozřejmě šikovný uživatel dokáže na frontendu zobrazit tuto stránku, ale zastavila by ho nakonec neúspěšná autorizace ze strany backendu.

Komponenta **CreateProjectPage.js** vykresluje titulek stránky a dále komponentu **CreateProjectForm.js**, ve které se nachází formulář pro vytvoření nového projektu. Formulář nese název „Informace o projektu“ a obsahuje následující vstupní prvky s názvy: Název, Popis, URL název, Obrázek. Na straně frontendu je pro každé pole použita alespoň základní validace vyžadující jakoukoliv neprázdnou hodnotu. Název, URL název a Obrázek jsou klasickým vstupem typu text. Popis je typu textarea, protože se počítá s delším množstvím znaků. Maximálně však 2048. Obrázek lze vložit pomocí jeho URL adresy, jestli je již někde na internetu dostupný.

Ve formuláři je však možnost nahrát i vlastní obrázek ze zařízení. Tento proces je popsán ke konci této kapitoly. V obou případech je nakonec serveru zaslán pouze URL odkaz obrázku. Po vyplnění všech údajů formuláře se jeho odesílací tlačítko stane dostupným. Toto tlačítko nese název „Vytvořit projekt“ a po kliknutí jej obsluhuje metoda s názvem **postCreateProject**. V této metodě se zasílá požadavek na B/api/admin/create-project metodou POST. V těle požadavku jsou zaslána výše uvedená data z formuláře. Do hlavičky byl přiřazen také autorizační token, aby bylo možné na straně serveru ověřit, zda požadavek odeslal autorizovaný uživatel. Na straně serveru se nejprve přečte token z hlavičky, ověří se autorizace a zvalidují se data pomocí **createOrEditProject** v souboru adminVlds.js. Následně je požadavek puštěn do middlewaru **postCreateProject** v adminCtrl.js. Zde se vytvoří nový dokument podle schématu Project, který se pomocí metody **save** uloží do databáze. Pokud během ukládání dojde k jakékoliv chybě, nebude úspěšně dokončeno, odešle se jako odpověď chyba s hláškou „Nepodařilo se vytvořit projekt“. V opačném případě se zašle klientovi pozitivní odpověď. Klient uživatele přesměruje na detail projektu pomocí funkce navigate.

Nahrávání obrázků ze zařízení

Pro nahrávání obrázků ze zařízení bylo zvolené cloudové úložiště S3 [16], které Amazon poskytuje zdarma. Pro jeho získání a použití je nutno si vytvořit registraci na Amazon Web Services [17]. V sekci úložiště S3 je potřeba vytvořit nový „backet“. Jedná se o kontejner pro data. Tento backet je potřeba správně nakonfigurovat. Konfigurace zde popsaná není, jedná se o složitější problematiku používání cloudových služeb Amazonu. Popsána je zde pouze implementace z pohledu vyvíjené aplikace.

Proces nahrávání obrázku funguje tak, že je do úložiště S3 ukládán přímo z frontednové aplikace. Tento přístup ulehčí backendové části od zpracování obrázků. Implementace připojení k backetu a funkce pro uložení obrázku jsou v souboru utils/**s3.js**. Pro napojení k úložišti je nutné zavolat metodu S3, která je dostupná z modulu **aws-sdk**. Jako parametr uvést objekt s atributy region, accessKeyId, secretAccessKey a signatureVersion. Klíče jsou vygenerovány při vytváření a konfiguraci backetu. Metoda, která obstarává nahrání obrázku do

backetu nese v souboru název **generateUploadURL**. Tato metoda vytvoří obrázku unikátní název s použitím balíčku **uuid**, aby nebylo možné přepsat obrázek se stejným názvem. Obrázek nahraje do úložiště, a nakonec vrátí URL odkaz na nahraný obrázek.

6.4.11 Detail Projektu

Podrobné informace o projektu se naleznou na adrese F/projekt/:urlTitle. Při přesměrování na tuto adresu se vykreslí **ProjectDetailPage.js**, která ze všeho nejdříve zašle požadavek na B/api/projects-by-url-title/:urlTitle. Toto API na backendu vyhledá v databázi projekt na základě atributu urlTitle, který u sebe projekt uchovává a zašle dokument projektu frontendu. Jakmile komponenta ProjectDetailPage.js obdrží jako odpověď projekt, rozhodne se ve switchi na základě jeho atributu **type**, kterou komponentu dále vykreslí a předá jí data o projektu v props. Pokud nabývá hodnoty „donate“, jedná se o klasický projekt a je vykreslena stránka ProjectDetailDonatePage.js. V opačném případě je nutno další možnosti definovat a uvést odpovídající komponentu. V této aplikaci je jeden speciální projekt a tím je „Kup si svoji část pozemku“. Ten má v atributu type hodnotu „donate-land“ a v tomto případě se vykreslí komponenta KupSiSvojiCastPozemkuPage.js.

Klasický projekt

V této aplikaci jdou administrátorem vkládat pouze projekty typu „donate“. K jeho vytvoření není potřeba znalosti programování a lze jej snadno vytvořit pomocí formuláře na adrese F/novy-projekt.

Obsah klasického projektu se nachází v komponentě **ProjectDetailDonatePage.js.** Ta na základě dat z props o projektu jako první vykresluje název projektu, dále zobrazuje obrázek. Poté vykresluje komponentu **News.js**, která reprezentuje sekci o aktualitách, a komponentu **Donatables.js**, která má za úkol vykreslovat sbírky, tzv. „darovatelné boxy“ projektu. Pokud je přihlášen administrátor, jsou zde pro manipulaci s projektem dostupná i tlačítka s názvy Upravit a Odstranit.

Úprava projektu

Na stránku pro úpravu projektu se administrátor dostane kliknutím na tlačítko „Upravit“ v detailu projektu. Tlačítko přesměruje prohlížeč na stránku

F/upravit/projekt/:projectId a vykreslí se element **EditProjectPage.js**. Tato komponenta z URL adresy vytáhne pomocí metody useParams, z balíčku react-router-dom, hodnotu **projectId**. Poté zašle požadavek na B/api/projects/:projectId pro získání informací o projektu. Na straně serveru se tohoto úkolu ujme middleware **getProject**, který pomocí metody findOne v databází najde dokument projektu a vrátí jej klientovi. Klient si data pomocí funkce setProject uloží do proměnné project. Následně je vykreslen titulek stránky „Úprava projektu“ a komponenta **EditProjectForm.js**, které jsou předána data projektu. Formulář na úpravu projektu není nijak odlišný od formuláře pro vytvoření projektu, kromě dvou záležitostí. První změnou je, že má formulář již přednastavené hodnoty ve svých vstupních prvcích a je celý ve výchozím stavu validní. Druhou změnou je, že po kliknutí ho obsluhuje jiná metoda, a to **updateProjectHandler**. Metoda updateProjectHandler zasílá v těle požadavku všechny hodnoty vstupních prvků formuláře na B/api/admin/edit-project/:projectId. Backend zvaliduje vstupy stejně jako při vytváření nového projektu, proběhne autentizace a autorizace. Dále se požadavku ujme middleware **patchEditProject**, který pomocí metody findByIdAndUpdate, zavolané na modelu Project, vyhledá konkrétní dokument projektu a aktualizuje u něho údaje pomocí dat z těla požadavku. Po úspěšném provedení operace je klientovi zaslána pozitivní zpráva. Frontend poté přesměruje správce zpět na stránku projektu příkazem navigate(-1).

Smazání projektu

Pro smazání projektu, resp. jeho pouhé označení jako smazaný, je nutné stisknout tlačítko „Odstranit“ v detailu projektu. Následné se objeví tlačítko „Doopravdy odstranit“ a až po jeho stisknutí se vykoná metoda **deleteProjectHandler**. Uvnitř této metody je pomocí metody sendRquest implementováno zaslání požadavku na B/api/admin/project/delete/:projectId.

6.4.12 Projekt „Kup si svoji část pozemku“

V této kapitole je popsán, jak dosavadní stav implementace první verze, která však byla v průběhu vývoje uzavřena ze známých nedostatků popsaných výše v práci, tak i verze druhá, která je konečná a plně implementovaná. První verze byla

implementována čistě s použitím technologie Three.js. Druhá verze využívá knihovny **react-three-fiber**.

První verze

Dostupná je na adrese F/kup-si-svoji-část-pozemku-prvni-verze. Komponenta **PozemekWebGlSection.js** renderuje element **canvas** s třídou „pozemek-webgl“. Po vykreslení elementu je spuštěna funkce s názvem **pozemekThreeStart**, která je v souboru **PozemekThreeJsScript.js.** V tomto souboru je uložena implementace čistě v Three.js. Byla implementována jedna sekce se svými dílčími segmenty. Poté se přišlo na nedostatky této metody a došlo tedy k rozhodnutí změnit přístup, který byl realizován v druhé verzi. Pro použití technologie Three.js bylo potřeba nainstalovat modul **three**.

Zaprvé je nutné vytvořit scénu.

```
const scene = new THREE.Scene()
```

Obr. 21 Kód: vytvoření scény

Dále je potřeba nastavit kameru, resp. pozici pozorovatele. Byla využita perspektivní kamera, která umožňuje realistický vhled do prostoru. Tzn, že objekty se se zvětšující se vzdáleností zmenšují. Kameru vhodně umístit a pak pomocí metody **add** přidat do scény.

```
const camera = new THREE.PerspectiveCamera(35, sizes.width / sizes.height, 0.1, 5000)
camera.position.set(-76, -8.09, 1713)
scene.add(camera)
```

Obr. 22 Kód: definice kamery

Následně je zapotřebí vytvořit renderer a napojit jej na canvas element na stránce. Referenci na canvas lze získat například metodou querySelector.

```
const renderer = new THREE.WebGLRenderer({
    canvas: canvas
})
```

Obr. 23 Kód: vytvoření rendereru

Pro zobrazení mapy je potřeba nejdříve načíst textury. K tomu je využit **TextureLoader**, která je součástí balíčku three. Pro načtení textury stačí na tomto objektu zavolat metodu **load** a jako parametr uvést cestu k obrázku. Textura je namapována na dvoudimenzionální plochu. Nejprve je nutno vytvořit geometrii plochy zavoláním THREE.PlaneGeometry(„výška“, „šířka“). Následně se musí vytvořit materiál, kterému se definuje textura. Jakmile je vytvořená jak geometrie objektu, tak materiál, který se na něj má aplikovat, lze zkompletovat takzvaný mesh.

```
const mapGrey = new THREE.Mesh(mapGeo, MapGreyMat)
```

Obr. 24 Kód: vytvoření meshe (grafického objektu)

Tímto stylem se vytvářejí veškeré grafické objekty v Three.js. Zavoláním metody **add** na objektu **scene** se do scény objekty vloží, pokud se předají jako parametr. Po vykreslení mapy je potřeba vykreslit sekce pozemku. Je vykreslena pouze první sekce „Pro děti z kasičky“. Pro vykreslení je použit stejný postup jako při vykreslení mapy. Odlišnost je pouze v textuře, ve velikosti a umístění plochy. Plocha, která sekci vykresluje, se má zobrazovat pouze při přejetí myši nad její oblastí. Proto je potřeba definovat raycaster a při každém ticku zaslat paprsek ze směru kamery k pozici kurzoru myši. Musí se porovnávat protnutí paprsku se všemi objekty ve scéně reprezentující sekci, ty jsou uložené v poli areasToTest. Všechny zasažené sekce paprskem se vyskytnou v poli intersectAreas. Pokud toto celé má probíhat, závisí na proměnné isAreaChoosingMode. Tato proměnná nabývá hodnoty true, dokud se nevybere sekce, ve které si chce dárce zakoupit segment.

```
if (isAreaChoosingMode) {
  raycaster.setFromCamera(mouse, camera)
  const intersectsAreas = raycaster.intersectObjects(areasToTest);
  for (const intersect of intersectsAreas) {
    intersect.object.visible = true;
  }
  for (const object of areasToTest) {
    if (!intersectsAreas.find((intersect) => intersect.object === object)) {
      object.visible = false;
    }
  }
}
```

Obr. 25 Kód: zjištění průsečíků se sekcemi pomocí raycastu

Nakonec se zobrazí pouze ta sekce, přes kterou je umístěn kurzor myši. Všechny ostatní sekce, které nebyly zasaženy paprskem, jsou zneviditelněny. Informace, kde se kurzor myši ve scéně nachází, je získána následujícím způsobem.

```
const mouse = new THREE.Vector2();
window.addEventListener("mousemove", (event) => {
  mouse.x = (event.clientX / sizes.width) * 2 - 1;
  mouse.y = -(event.clientY / sizes.height) * 2 + 1;
});
```

Obr. 26 Kód: definice myši a transformace souřadnic pozice kurzoru

Objekt **mouse** je obyčejný dvoudimenzionální vektor, který si uchovává X a Y souřadnice kurzoru myši. K přepočtu dojde ve chvíli, kdy uživatel změní pozici kurzoru. To je docíleno použitím této obsluhy události mousemove. Přepočet pro obě hodnoty souřadnic je na rozmezí -1 až 1.

Po kliknutí na sekci se nastaví režim isAreaChoosingMode na false. To zamezí vykreslování sekcí při pohybu myši nad jejími oblastmi. Dále se nasměruje kamera blíže k sekci pomocí zavolání metody camera.position.**set**(x,y,z). Nakonec jsou vykresleny segmenty dané sekce. V tomto případě tedy 7 000 trojúhelníků. Jsou vykresleny v mřížce pomocí dvou vnořených cyklů. Pokud by se pokračovalo v implementaci první verze dále, každý dílek by představoval svůj dokument v databázi na základě návrhu. Vývoj skončil u vykreslení těchto segmentů bez datového propojení s databází. Je tedy alespoň znázorněná grafická interpretace. Každý vykreslený trojúhelník uchovává informace: souřadnice **i** a **j** v rámci mřížky, číselné označení (**number**), zda je dostupný (**available**) a zda je vybraný k nákupu (**selected**). Po kliknutí na daný segment, potom, co se pomocí raycastu zjistí, na který se kliklo, se vykoná následující algoritmus.

```
if (intersect.object.userData.avaible) {
  // segment lze zakoupit
  if (intersect.object.userData.selected) {
    // segment již vybrán, odebrat ze seznamu vybraných částí
    intersect.object.material.color.set(parameters.availableC)
    intersect.object.userData.selected = false
    removePiece(intersect.object.userData.number)
  } else {
    // vybrání segmentu
    intersect.object.material.color.set(parameters.O3_hoverC)
    intersect.object.userData.selected = true
    addToBuy({ ...intersect.object.userData })
  }
} else {
  // segment již zakoupen (zobrazení jména a poznámky kupujícího)
}
```

Obr. 27 Kód: rozhodovací větev pro manipulaci se segmentem

Mohou teoreticky nastat 2 případy. Buď je segment **available**, tzn. že ještě nebyl zakoupen a je možné jej libovolně vkládat do košíku či jej z něj vyndávat. Nebo byl již zakoupen, dostupným není a po kliknutí je zde prostor pro implementaci zobrazení informace o kupujícím.

Pomyslná vzdálenost celkové implementace této první verze pro splnění funkčnosti prvního způsobu přispívání od dosavadní implementace není daleká, ale rozhodnutí ze strany zadavatele nakonec zaúkolovalo autora práce změnit přístup a začít s implementací druhého způsobu.

Druhá verze

Druhá verze zobrazení mapy je funkční a plně implementovaná. Je obsažena na adrese F/projekt/kup-si-svoji-cast-pozemku. Na této stránce se vykresluje komponenta **KupSiSvojiCastPozemkuPage.js**. Jako v každém projektu této platformy je obsažen název, obrázek, popis projektu a aktuality, ani tento není výjimkou. Pouze zde nejsou dynamické „darovatelné boxy“, se kterými by mohl správce manipulovat. Je zde jeden napevno připravený „darovatelný box“ a prostor pro zobrazení a vizualizaci mapy.

Modul react-three-fiber

Jak už bylo zmíněno dříve, v této druhé verzi byl použit modul react-three-fiber. Pro použití technologie Three.js v kombinaci s Reactem je to nejvhodnější volba. Ulehčuje například práci s raycastem. Již není potřeba zasílat paprsek pro zjištění, jestli se kliklo na nějaký objekt, zavádět si vektor se souřadnicemi myši, převádět tato data do normalizovaných souřadnic, zjišťovat průsečíky se zasaženými objekty v poli objektů. Objekty jsou s tímto novým přístupem reprezentovány jako komponenty Reactu a je možné jim přiřadit atribut onClick a jako hodnotu vložit metodu, která bude po kliknutí na objekt provedena. Již není potřeba zabývat se tím, jak zjistit, jestli se kliklo na daný objekt, ale co dělat, když se na něm kliknutí provede. Tento přístup umožňuje se zabývat přímo aplikační logikou a programování základních principů zde není potřeba.

Implementace vykreslení

Pro vizualizace mapy a darů na pozemek je potřeba data nejprve získat ze serveru. Toho se docílí zasláním požadavku na B/api/donations/:donatableId. Server vrátí jako odpověď pole darů na pozemek. Klient si je uloží do proměnné donations a data pomocí props pošle dál komponentě ThreeJS_Canvas_Land.js. Ta vykresluje element **Canvas** z modulu react-three-fiber. Kamera se do scény nastavuje jako atribut k elementu Canvas. Uvnitř Canvasu se renderuje komponenta **Experience_Land.js**. Tato komponenta uchovává element **OrbitControls**, který umožňuje uživateli manipulovat s kamerou použitím myši. Dále kromě komponent pro použití osvětlení obsahuje také 4 další komponenty – Map, LandFrame, LandPieces, LandPieceToDonate

Komponenta Map.js

Tato část je odpovědná za vytvoření a zobrazení mapy na scénu. Následující kód definuje geometrii, materiál a vytvoří z nich mesh.

```
<mesh>
  <planeGeometry args={[1772, 785]} />
  <meshStandardMaterial map={land_map} />
</mesh>
```

Obr. 28 Kód: vytvoření meshe v RTF

U elementu planeGeometry v atributu args je uvedeno pole o dvou číselných hodnotách, ty reprezentují výšku a šířku plochy mapy. V elementu meshStandardMaterial v atributu map je vložena reference na texturu. Ta je načtena pomocí TextureLoaderu stejně jako v předchozí verzi.

Komponenta LandFrame.js

Tato komponenta je zodpovědná za zobrazení ohraničení pozemku, na který se přispívá. Ohraničení má velikost 870x330, má světle modrou barvu plochy, která je z 20% průhledná. Hrany této plochy průhledné nejsou.

Komponenta LandPieces.js

Zodpovídá za vykreslení všech darů na projekt „Kup si svoji část pozemku“. Vykresluje libovolně dlouhé obdélníky dle jejich výše jejich ceny. Cílová částka k vybrání je 4 900 000 Kč. Nejnižší velikost daru na tento projekt je 100 Kč, ten má podobu čtverce a všechny vyšší částky jsou už obdélníkového tvaru. Je potřeba zjistit obsah pozemku daru za 100 Kč. Obsah plochy pozemku je 870 * 330, což se rovná 287 000. Tento obsah je třeba vydělit 49 000, aby se zjistila plocha daru za cenu 100 Kč. Výsledkem je 5,859. Odmocninou tohoto výsledku se získá velikost hrany čtverce daru za 100 Kč, a to je 2,42. V kódu je tato hodnota uložena v konstantě A_LENGTH. Nyní je známa výška každého segmentu, šířka je však závislá na výši darované částky.

Tato komponenta přijímá v props pole darů a jsou v ní definovány následující 2 důležité metody.

createLandPieces

```
const createLandPieces = () => {
  for (let i = 0; i < props.donations.length; i++) {
    let donationIsWholeDrawn = false
    let donationWidth = (props.donations[i].price / 100) * A_LENGTH
    while (!donationIsWholeDrawn) {
      if (availableRowWidth < donationWidth) {
        // Dar bude třeba segmentovat, protože se nevejde do konce řádku
        const widthToDrawNext = donationWidth - availableRowWidth
        createLandPiece(availableRowWidth, props.donations[i])
        donationWidth = widthToDrawNext
        startNewRow()
      }
      if (donationWidth <= availableRowWidth) {
        // Dar není třeba rozdělit na více částí
        createLandPiece(donationWidth, props.donations[i])
        donationIsWholeDrawn = true
      }
    }
    if (i + 1 === props.donations.length) {
      setLastOffsets(landPieceOffset_X, landPieceOffset_Y)
    }
  }
}
```

Obr. 29 Kód: tvorba segmentů na základě daru

V této metodě se nachází for cyklus, který se provede přesně tolikrát, kolik prvků je v poli donations. Pro každý dar je potřeba vytvořit grafickou reprezentaci na mapě. Na začátku cyklu je deklarována pravdivostní proměnná **donationIsWholeDrawn** a je jí přiřazena hodnota false. Tato proměnná určuje, zda je již dar celý vykreslený. Proměnná **donationWidth** obsahuje šířku daru. Ta je spočítána jako (výše_daru/100)*(2,42). Např. v případě daru výše 200 Kč, bude plocha široká (200/100)*(2,42)=4,84. Dále je v algoritmu uveden cyklus while, který se opakuje do té doby, dokud se nepodaří vykreslit celou plochu daru. Pokud se stane, že spočítaná šířka daru je větší než dostupné místo na řádku, je potřeba dar segmentovat. Tzn. vykreslit část daru, která se na zbytek řádku vejde, spočítat zbylou šířku daru a pokusit se vykreslit zbytek daru na další řádek. Pokud se zbytek daru opět nevejde do zbylého místa na řádku, je dar segmentován, dokud se celý nevykreslí. Jestli je šířka daru menší než dostupné místo na řádku, vytvoří se jeho

reprezentace a proměnná **donationIsWholeDrawn** se nastaví na true a cyklus while tedy skončí své provádění. Pokud se jednalo o poslední dar v poli donations, jsou zachovány souřadnice posledního vykresleného segmentu, a to proto, aby bylo možné uživateli při výběru darované částky zobrazit jeho potencionální dar na mapě na pozici za posledním darem.

createLandPiece

```
const createLandPiece = (donationWidth, donation) => {
  // Umístit do poloviny velikosti daru pro zobrazení
  landPieceOffset_X += donationWidth / 2
  createdLandPieces.push(
    <LandPiece
      key={Math.random()}
      width={donationWidth}
      offsetX={landPieceOffset_X}
      offsetY={landPieceOffset_Y}
      donation={donation}
    />
  )
  // Přičíst druhou polovinu daru pro získání pozice na konci daru
  landPieceOffset_X += donationWidth / 2
  // Aktualizace dostupné šířky pro další dar
  availableRowWidth -= donationWidth
}
```

Obr. 30 Kód: pozicování segmentu daru

Metoda createLandPiece se zabývá vytvářením jednotlivých ploch daru. Jako parametr přijímá již spočítanou šířku daru a data o daru. Před vykreslením daru je nutné změnit pozici vykreslení do středu místa daru a nezačínat hned na konci předešlého daru. Plocha daru se totiž při vykreslení rozpíná do levé i pravé strany, a pokud by nedošlo k přemístění, nový dar by částečně překrýval plochy darů předchozích. Následně je vytvořena komponenta LandPiece. Nakonec je aktualizovaná x-ová souřadnice na pozici za vykreslený dar a je přepočítán zbytek prostoru na řádku.

Komponenta LandPiece.js

Tato komponenta se stará již o finální vykreslení plochy. Jsou jí pomocí props předány klíčové informace, jako je šířka plochy, pozice pro vykreslení a data o daru.

```
<planeGeometry args={[props.width, A_LENGTH]} />
<meshBasicMaterial
  color={getColorByPrice(props.donation.price)}
  transparent
  opacity={0.4}
/>
{showHTML && (
  <Html className="landPiece-donation-info--wrapper">
    <p>Dárce: {props.donation.name}</p>
    <p>Částka: {props.donation.price}</p>
    <p>Chci sdělit: {props.donation.note}</p>
  </Html>
)}
```

Obr. 31 Kód: vykreslení darovaného segmentu

LandPiece.js tedy vykresluje plochu. Barvu plochy určuje funkce **getColorByPrice**, která na základě ceny vyhodnotí, která barva se má použít. Materiál je ve výchozím stavu ze 60% průhledný. Při najetí myší na plochu se materiál dané plochy zneprůhlední nastavením atributu opacity na 1 a zároveň se vykreslí i element **Html** s informacemi o daru, protože se pravdivostní proměnná **showHTML** nastaví na true. Html element je dostupný z balíčku **@react-three/drei**. Při opuštění prostoru plochy kurzorem se showHTML nastaví zpět na false, a tím se skryjí informace o daru. Následně se materiál plochy znovu částečně zprůhlední.

Komponenta LandPieceToDonate.js

Zobrazuje se pouze v případě, je-li vybraná nějaká částka k darování. Graficky uživateli vizualizuje částku, v podobě plochy, kterou chce přispět. Informaci o ceně získává z props. Komponenta je velmi podobná LandPiece.js, ale je obalená ještě elementy **Bound** a **SelectToZoom**. Tyto elementy jsou dostupné z balíčku @react-three/drei a způsobují automatické přiblížení kamery k vykreslené ploše daru po vybrání ceny.

Komponenta LandPiecesDonationOptions.js

Jedná se o element, který zobrazuje 4 tlačítka s cenami 100, 200, 500 a 1000 Kč. Následně je vykresleno tlačítko „Vlastní částka“ po jehož kliknutí se zobrazí vstup pro zadání vlastní částky. Následně si uživatel vybere mezi tlačítky „Zobrazit jméno u daru“ a „Skrýt jméno“. Tato tlačítka určují viditelnost daru. Dále je zde dostupné

vstupní pole „Chci sdělit“, kam dárce může napsat vlastní poznámku k daru. Pro uchování dat formuláře se využívá háčku autora práce - g-form-hook.js. Jako poslední se v tomto elementu vykresluje tlačítko „Přispět“, které po kliknutí obsluhuje metoda cart.**addDonations**, která vkládá dary do košíku.

6.4.13 Sbírky – „darovatelné boxy“

Sbírky jsou dostupné na stránce s detailem klasického projektu F/projekt/:urlTitle. O jejich vykreslení se stará element **Donatables.js.** Tato komponenta nejdříve zašle požadavek na B/api/donatables-by-project-id/:projectId. Server v databázi podle uvedeného ID projektu nalezne a vrátí klientovi všechny dostupné sbírky pro daný projekt. Klient toto pole dokumentů o sbírkách funkcí map celé proputuje a pro každý prvek vrátí element **Donatable.js**, který reprezentuje jednotlivou sbírku. Ke každé sbírce jsou vykresleny základní informace, jako název, obrázek, popisek, cílová částka a množství již vybraných peněz. Vedle základních informací je zde zobrazené tlačítko „Přispěvatelé“, které po kliknutí povolí vykreslení komponenty **Donators.js.** Tento element vykresluje všechny dary v podobě informací: Jméno dárce, vlastní poznámka, darovaná částka. Data komponenta získá zasláním požadavku na B/api/donations/donatableId. Backend tedy podle ID sbírky nalezne a zašle zpět frontendu všechny dary, které jsou s ní ve vztahu. Pod elementem Donators.js se vykresluje komponenta **ProgressBar.js**, která v podobě načítací lišty vizualizuje procentuální množství vybrané částky z cílové částky. Poslední element, který se v Donatable.js renderuje je **DonationOptions.js**. Tato komponenta, na základě hodnot v poli preparedPrices u objektu sbírky, vykresluje předpřipravené částky ve formě tlačítek. Dále vykresluje tlačítko „Vlastní částka“, po jehož kliknutí se zobrazí vstup pro zadání vlastní částky uživatelem. Po vybrání částky se vykreslí tlačítka, která určují, zda se bude jednat o anonymní dar či veřejný. Po vybrání viditelnosti daru se zobrazí textový vstupní prvek, který reprezentuje vlastní poznámku k daru. Tato poznámka je nepovinná. Následně je dostupné tlačítko „Přispět“, po jehož kliknutí se spustí obslužná metoda a ta provede zavolání metody cart.**addDonations**, která jako parametr přijímá pole darů, které vkládá do košíku. Poté proběhne automatické přesměrování na košík pomocí příkazu navigate(„/kosik“).

Vytvoření sbírky

Administrátorovi je v detailu projektu v elementu Donatables.js vykresleno tlačítko „Přidat darovatelný box“. Po jehož kliknutí je přesměrován na adresu F/novy-darovatelny-box/:projectId. Na této stránce se renderuje komponenta **CreateDonatablePage.js**, která obsahuje titulek stránky „Vytvořit darovatelný box“. Následně je vykreslený formulář **CreateDonatableForm.js**, který obsahuje vstupní prvky: Název, Popis, Cílová částka, Předpřipravené částky, Obrázek. Všechny prvky obsahují základní validaci na neprázdnou hodnotu. Předpřipravené částky se píšou za sebou do jednoho textového řetězce a jednotlivé položky jsou odděleny čárkou. Obrázek se nahrává stejným způsobem jako při vytváření projektu. Po vyplnění všech vstupů je povoleno stisknout tlačítko „Vytvořit sbírku“, které obsluhuje metoda **postCreateDonatable**. Ta zašle data vstupních prvků formuláře spolu s požadavek na B/api/admin/create-donatable/: projectId. Server zvaliduje hodnoty zaslané v těle požadavku, provede autorizaci uživatele a následně požadavek spolu s daty pustí do middlewaru **postCreateDonatable**. Zde je do databáze uložen nový dokument sbírky. Frontendu je zasláno potvrzení o dokončení operace. Uživatel je přesměrován zpět na detail projektu příkazem navigate(-1).

Úprava sbírky

Pro upravení dat sbírky je nutné na „darovacím boxu“ stisknout tlačítko „Upravit“. Toto tlačítko je viditelné pouze administrátorovi. Po kliknutí je přesměrován na adresu F/upravit/sbirka/:donatableId, kde se vykresluje element **EditDonatablePage.js**. Na této stránce je zobrazen titulek „Úprava sbírky“. Pod titulkem je zobrazen formulář **EditDonatableForm.js**. Formulář obsahuje stejné vstupy jako při vytváření nové sbírky. V tomto formuláři jsou předvyplněné vstupy daty o konkrétní sbírce, získané z databáze zasláním požadavku na B/api/donatables/:donatableId. Formulář se po vyplnění daty nastaví jako validní. Pokud má formulář validní všechny vstupy, stane se tlačítko „Aktualizovat sbírku“ dostupným. Obsluhuje jej metoda **updateDonatableHandler**, která ve svém těle zasílá požadavek s daty formuláře na B/api/admin/edit-donatable/:donatableId. Server zaprvé provede autorizaci uživatele, dále provede validaci hodnot vstupních prvků formuláře a nakonec požadavek proputuje až do middlewaru

patchEditDonatable, kde se pomocí metody **findByIdAndUpdate** na modelu sbírky nalezne a aktualizuje dokument o konkrétní sbírce. Poté je frontendu poslána odpověď a klient je přesměrován zpět na detail projektu.

Smazání sbírky

Administrátor má u každé sbírky dostupné tlačítko „Odstranit“. Po jeho kliknutí se spustí metoda **deleteDonatableHandler**, která jako parametr přijímá ID sbírky, na základě, které je sbírka jednoznačně identifikována. Metoda posílá požadavek na B/api/admin/donatable/delete/:donatableId. Backend provede autorizaci uživatele a poté požadavek pustí dále do middlewaru **patchSetDonatableDeleted**, který metodou findOneAndUpdate aktualizuje v dokumentu sbírky vlastnost deleted na hodnotu true. Po úspěšném dokončení operace je i na frontendu sbírka smazána z lokálních dat, aby se dále nezobrazovala.

6.4.14 Aktuality

Aktuality jsou zobrazeny v detailu projektu. **News.js** po svém prvním vykreslení zasílá požadavek na B/api/news/:projectId, aby získala všechna data o aktualitách, které k danému projektu patří. To na backendu zařídí middleware **getNewsByProjectId**, který z databáze načte aktuality, které patří k uvedenému projektu pomocí projectId a v poli deleted mají hodnotu false, aby se nenačetly již smazané aktuality. Následně je zašle klientovi. Klient si aktuality uloží do **loadedNews**, a protože jsou v datové struktuře pole, tak je nad ním zavolaná metoda **map**, která projde všechny prvky pole, resp. všechny aktuality, a pro každý objekt vytvoří a vykreslí komponentu **NewsItem.js**, které předá data o aktualitě v props. NewsItem.js zobrazuje data aktuality. Vykresluje její název, čas vytvoření a samotný text. Obsahuje také tlačítka pro editaci a smazání aktuality.

Vytvoření aktuality

Administrátor má v News.js dostupné tlačítko „Přidat aktualitu“. Jeho stisknutím se prohlížeč přesměruje na adresu F/nova-aktualita/:urlTitle. Zde je vykreslena komponenta **CreateNewsPage.js**, která zobrazuje titulek stránky „Vytvořit novou aktualitu“ a element **CreateNewsForm.js**. Tento element obsahuje formulář se dvěma vstupními prvky s názvy Nadpis a Text. Používá základní validaci na oba

vstupy, aby měly neprázdné hodnoty. Jakmile jsou obě pole vyplněna, formulář se stane validním a v tu chvíli je dostupné tlačítko „Vytvořit aktualitu“. Jeho událost po kliknutí obsluhuje metoda **postCreateNews**, která zašle http požadavek na B/api/admin/create-news/:urlTitle a do těla požadavku umístí hodnoty ze vstupních prvků. Na serveru nejprve proběhne autorizace správce, validace vstupů a poté je požadavek poslán dále do **postCreateNews.** Nejprve se v databází pomocí parametru urlTitle nalezne projekt, ke kterému tato hodnota patří. Z nalezeného objektu se získá ID Projektu. Následně je již vytvořen objekt aktuality na základě jeho schématu, jsou mu přiřazeny hodnoty z formuláře a nalezené ID projektu. Poté je pro uložení objektu do databáze zavolána metoda save. Po úspěšném vytvoření je klientovi zaslána kladná informace o dokončení. Klient je přesměrován zpět na detail projektu metodou navigate, kde již bude zobrazena vytvořená aktualita.

Úprava aktuality

Pro administrátora je u každé položky aktuality dostupné tlačítko „Upravit“, které po kliknuté přesměruje klienta na stránku F/upravit/aktualita/:newsId. Tato stránka je velmi podobná stránce pro vytváření aktuality. Renderuje se komponenta **EditNewsForm.js**. Titulek stránky nese název „Úprava aktuality“ a formulář je umístěn v komponentě **EditNewsForm.js**. Formulář obsahuje stejné vstupní prvky jako při vytvoření aktuality, ale s již předvyplněnými hodnotami, které jsou získány zasláním požadavku na B/api/news-item/:newsId. Server podle ID aktuality vrátí daný dokument frontendu a ten data vloží do formuláře. Po kliknutí na tlačítko „Aktualizovat“ se provede metoda **updateNewsHandler**, která pošle http požadavek na B/api/admin/edit-news/:newsId. Server provede autorizaci, zvaliduje vstupy a následně metodou findByIdAndUpdate nalezne a aktualizuje údaje aktuality. Backend pošle frontendu pozitivní zprávu o dokončení aktualizace a klient je následně přesměrován zpět na detail projektu.

Smazání aktuality

V komponentě News.js je definovaná funkce **deleteNewsHandler**, která slouží pro odstranění aktuality. Jako parametr přijímá newsId, s pomocí kterého vytvoří URL pro zaslání požadavku na adresu B/api/admin/news/delete/:newsId. Backend nejprve provede autorizaci administrátora, poté požadavek skončí v middlewaru

patchSetNewsDeleted, který pomocí metody **findOneAndUpdate** vyhledá aktualitu podle ID a nastaví položku deleted na hodnotu true. Po úspěšném dokončení tohoto úkonu, je poslaná pozitivní odpověď frontendu. Ten si aktualitu smaže i ze svého pole **loadedNews** pomocí metody **setLoadedNews**, do které je jako parametr uvedeno nové pole, ale bez již smazaného prvku.

```
setLoadedNews(loadedNews.filter(news => news._id !== newsId))
```

Obr. 32 Kód: smazání položky z pole na základě ID

6.4.15 Nákupní košík

Stránka s košíkem se nachází na adrese F/kosik a jako obsah stránky je zde zobrazen element **CartPage.js**. Na stránce je vyrenderován titulek „Košík“, pod ním je vykreslen seznam položek košíku. Každá položka je reprezentována elementem **CartItem_Donation.js**. Tento element vykresluje název sbírky, obrázek, popis, cenu a vlastní poznámku k daru. Dále vykresluje ikonu koše, po jejímž stisknutí se spustí metoda **removeDonation**, která podle ID položky odstraní dar z lokálního úložiště. Tato komponenta má přístup k datům košíku, které jsou uložené v lokálním úložišti. O jejich ukládání a načítání se stará háček **cart-hook.js.** Metodou **addDonations**, kterou háček cart-hook.js exportuje, se vkládají veškeré dary do košíku. Metodou **clearCart** se po dokončení objednávky odstraní všechny položky z košíku. Na této stránce je ještě vykresleno tlačítko „Nakoupit“, které po stisknutí přesměruje uživatele na stránku pro dokončení objednávky - F/nakup.

6.4.16 Nákup

Pro dokončení objednávky je nutné vyplnit údaje k objednávce. Formulář pro jejich vyplnění se nachází na adrese F/nakup. Na této adrese se zobrazuje stránka **OrderFillingInfoPage.js**. Zde se vykresluje formulář, který je rozdělen do tří částí. První část nese titulek „Kontaktní údaje“ a vyžaduje následující informace: jméno, příjmení, email a tel. číslo. Pro každou z nich je vytvořen textový vstup. Na stránce je vykresleno tlačítko „Nakupuji na firmu“, po jehož stisknutí se objeví druhá část formuláře s názvem „Firemní údaje“, kde jsou vyžadovány údaje: název společnosti, IČO, DIČ. Poslední část formuláře s název „Údaje pro certifikát“ se vykreslí po

stisknutí tlačítka „Chci certifikát“. Tato část požaduje informace: ulice a číslo popisné, město a PSČ. Pro všechny vstupní prvky formuláře je přinejmenším zajištěna validace pro neprázdnou hodnotu vstupu. U vstupu pro emailovou adresu je vyžadován správný formát. Pod formulářem je zobrazeno tlačítko „Dokončit objednávku“, který obsluhuje metoda **postCreateOrder**. Tato metoda zašle požadavek na B/api/create-order spolu se všemi údaji z formuláře. Na backendu proběhne validační proces a poté je požadavek puštěn do middlewaru **postCreateOrder**. Zde se nejprve vypočítá celková částka všech položek z košíku. Následně pro každý prvek košíku, respektive pro každý dar, je vytvořen dokument v databázi. U každého takového dokumentu je v atributu isPurchased přiřazena hodnota false. Ostatní atributy jsou vyplněny daty z položek košíku. Poté je vytvořen dokument objednávky. Do atributů tohoto dokumentu jsou přiřazeny hodnoty zaslané v těle požadavku. Hodnota atributu isPurchased je nastavena na false. Je vygenerováno unikátní ID pomocí metody uuidv4, která je dostupná z balíčku **uuid**. Využití této hodnoty je popsáno v kapitole 6.4.17 - Detail objednávky. Dokument objednávky je uložen do databáze metodou save. Následně je potřeba využít funkcionality platební brány.

V této aplikaci je integrovaná platební brána Stripe [18], která je otevřená vývojářům a poskytuje testovací prostředí bez nutnosti uzavírání smluv, na rozdíl např. od společnosti Comgate [19].

Pro využívání platební brány Stripe je nutné provést registraci na jejich webových stránkách a nechat si vygenerovat klíč, pomocí kterého systém platební brány ví, se kterým uživatelem komunikuje. Pro implementaci v Node.js je nutné nainstalovat modul **stripe**. Pro vytvoření relace platby je nutné zavolat metodu **create**, která je z tohoto balíčku dostupná. Jako parametr je žádoucí zadat nakupované položky a informace o nich, jako je např. cena, měna, název položky. Dalšími důležitými parametry jsou **success_url** a **cancel_url**. Jedná se o adresy, na které systém platební brány přesměruje uživatele v případech, že se platba buď úspěšně provedla, nebo platba neproběhla v pořádku a nebyla dokončena. Obě adresy směřují na detail objednávky, ale s odlišnými URL parametry. V pozitivním případě je do URL přidán parametr **success** s hodnotou true, v opačném případě atribut **canceled** s hodnotou true. Po vytvoření relace platby je získán odkaz pro

přesměrování na platební bránu. Tento odkaz je uložen do dokumentu objednávky pro případný opětovný pokus o zaplacení v případě, že se platbu na první pokus nepodaří úspěšně provést.

Dále je vytvořena faktura s pomocí balíčku **pdf-creator-node**. Tento modul umožňuje tvorbu PDF souborů na základě šablony vytvořené v HTML. Do této šablony lze dynamicky vkládat data, jako je např. jméno dárce, částka a jiné. Faktura je uložena na serveru a prozatím není uživateli zaslána. Šablona i implementace vytvoření PDF se nachází na backendu ve složce utils. Jedná se o soubory **pdf_template.html** a **pdf_service.js**. Pro vytvoření faktury je zavolána metoda **createBillPDF**.

V dalším dílčím kroku je uživateli zaslán email, který jej informuje o vytvoření objednávky. K zasílání emailu byla vytvořena emailová schránka na platformě Google ve tvaru postavskolu@gmail.com. Pro zasílání z aplikace Node.js byl použit modul **nodemailer**, který práci s odesíláním emailů řeší. Konfigurace modulu a implementace odesílání emailů je obsažena v souboru **mail_service.js**. Pro zaslání informace o vytvoření objednávky je zavolána metoda **sendEmail_OrderCreated**.

Posledním úkonem middlewaru postCreateOrder je zaslání odpovědi zpět klientovi, ve které je obsažena URL adresa relace platební brány. Klient odpověď zpracuje a přesměruje uživatele na platební bránu.

Po úspěšném dokončení platby systém Stripe zašle vyvíjené aplikaci informace o tomto skutku. Pro přijímaní této informace je vytvořen endpoint, nacházející se v souboru app.js začínající řádkem 48. Server zde poslouchá události zasílané službou Stripe. Pokud je událost typu „checkout.session.completed“, signalizuje to úspěšné zaplacení objednávky. Z objektu session je možné získat success_url, ve které je obsaženo ID objednávky. Na základě této hodnoty je v databázi aktualizován dokument objednávky, konkrétně jeho vlastnost **isPurchased** na true. Dále jsou aktualizovány všechny dokumenty o darech dané objednávky. U všech darů je vlastnost **isPurchased** nastavena na true a od této chvíle jsou dary v aplikaci promítnuty a jsou viditelné. Posledním krokem celého procesu nákupu je zavolání metody **sendEmail_OrderPurchasedAndBill**, která uživateli zašle email o úspěšném provedení platby a v příloze zasílá i dříve vytvořenou fakturu.

6.4.17 Detail objednávky

Detail objednávky je dostupný na adrese F/objednavka/:orderId a vykresluje se zde stránka **OrderDetailPage.js**. Na tuto stránku se uživatel dostane buď z přehledu objednávek, z odkazu na detail objednávky zaslaným mailem nebo ihned po pokusu provedení platby z platební brány.

Stránka zobrazuje číslo objednávky, údaje dárce a seznam zakoupených položek. Dále je zde zobrazen stav, zda je objednávka zaplacena. Pokud ne, je zde vykresleno tlačítko „Zaplatit“, které přesměruje na platební bránu. Data stránka získá zasláním požadavku na B/api/order/orderId?uuid=orderUUID. Na backendu se požadavek dostane do middlewaru **getOrderByIdAndUUID**. V databázi se nalezne dokument o objednávce, kde se shodují jak ID objednávky, tak i unikátní ID, které je zasláno jako parametr URL adresy. Pouze v tomto případě vrátí server klientovi data o objednávce. Samotná znalost ID objednávky nestačí. Unikátní druhé ID je použito právě kvůli bezpečnosti, aby se na detail objednávky mohl podívat pouze zakladatel objednávky.

Pokud se na stránku detailu objednávky uživatel dostane z platební brány, tak v URL adrese se buď nachází parametr success nebo cancel. Podle toho, který z nich je zaslán, je zobrazeno okno s informací o úspěšném nebo neúspěšném dokončení platby.

Přehled objednávek

Přehled objednávek je dostupný pouze pro přihlášené uživatele, nachází se na adrese F/moje-objednavky a reprezentuje jej element **MyOrdersPage.js**. Tato komponenta nejprve načte data o všech objednávkách přihlášeného uživatele zasláním požadavku na B/api/user/orders-by-user-email spolu s tokenem v hlavičce požadavku. Server nejprve ověří, zda je token validní, a pak na základě dešifrovaného emailu nalezne všechny k němu přidružené objednávky, které vrátí zpět klientovi. Vyhledávání probíhá v middlewaru **getOrdersByUserEmail**.

6.5 Nasazení

Pro nasazení webové aplikace online je využita služba Heroku. Poskytuje pouze placený plán, a pro co nejlevnější způsob nasazení se frontend i backend sloučí do jednoho serveru. Jedná se o testovací nasazení a nese s sebou i omezení, např. neúplná funkčnost platební brány, proto pro vyzkoušení aplikace jsou všechny objednávky ihned po vytvoření nastaveny jako zaplacené.

Prvním krokem je vytvořit build frontendové části, toho se docílí spuštěním příkazu **npm run build**. Po dokončení operace jsou výsledné sestavené soubory aplikace ve složce **build**. Je potřeba tyto soubory zkopírovat do složky **public** do backendové části aplikace.

Vložením následujícího middlewaru bude backend vracet frontendovou aplikaci.

```
app.use((req, res) => {
  res.sendFile(path.resolve(__dirname, 'public', 'index.html'))
})
```

Obr. 33 Kód: servírování aplikace React ze strany serveru

Je nutné tento middleware umístit až pod definici REST API, aby endpointy zůstaly dostupné. Nyní jsou sloučené obě části aplikace.

Nasazení aplikace bylo provedeno připojením Git repositáře. K vyzkoušení je dostupná na adrese https://postav-skolu.herokuapp.com/.

Pro vyzkoušení platby lze využít testovací kreditní kartu s číslem **4242 4242 4242 4242, libovolným expiračním datem i CVC.**

Pro vyzkoušení funkcionalit administrátora lze využít dočasného účtu s údaji:

email: **admin@test.cz**

heslo: **Admin123!**

7 Shrnutí výsledků

Hlavním výsledkem práce je vyvinutá webová aplikace s ohledem na návrhový vzor MVC, která slouží jako darovací platforma pro Základní školu speciální a praktickou školu Diakonie ČCE Vrchlabí. Na úplném začátku byla provedena analýza požadavků. Požadavky byly sbírány metodou volného rozhovoru. Ředitelka školy spolu s autorem práce tímto způsobem došli k funkčním i nefunkčním požadavky. Na základě sdělených požadavků byl následně vytvořen návrh. Byly navrhnuty všechny funkcionality vyplývající z funkčních požadavků. Návrh byl nezbytný a sloužil jako podklad pro implementaci webové aplikace. Darovacím prvkem systému jsou projekty se sbírkami, na které lze přispívat. Přispívat lze buď jednu z předpřipravených částek nebo částku vlastní, zadanou dárcem. V rozhovorech s ředitelkou školy byly rozebrány způsoby používání darovací platformy a různé formy procesu přispívání. Ve všech projektech kromě projektu „Kup si svoji část pozemku“, byl způsob přispívání sjednocen přes „darovatelný box“. V případě projektu darování na pozemek byly navrhnuty 2 způsoby přispívání. V prvním se však v průběhu implementace začalo přicházet na nedostatky této varianty, které jsou v práci blíže popsané. Bylo tedy navrhnuto nové řešení, které předešlé problémy odstranilo. Webová aplikace zadané požadavky ze strany školy v celém rozsahu splňuje. Došlo k implementaci všech požadavků ředitele. Do systému je možno vstoupit v roli uživatele, správce nebo jako neregistrovaný uživatel. Každá z těchto rolí má své funkcionality. Pro správce byly implementovány funkce: vytvářet, spravovat a odstraňovat projekty, sbírky a aktuality. Všem rolím je umožněno přispívat na sbírky dle jejich volby. Webová aplikace byla vyvinuta s použitím hlavních technologií React a Node.js. React byl použit pro vytvoření grafického uživatelského rozhraní. Pro vytvoření webového serveru byla použita technologie Node.js. Výsledkem analýzy, návrhu a implementace všech požadavků je systém umožňující darovat finanční prostředky, tedy darovací platforma. Hlavní cíl, analyzovat, navrhnout a implementovat darovací platformu na míru, byl splněn.

8 Závěry a doporučení

Předmětem diplomové práce byl vývoj darovací platformy na míru pro Základní školu speciální a praktickou školu Diakonie ČCE Vrchlabí. Hlavním cílem práce bylo analyzovat, navrhnout a implementovat darovací platformu, která pozitivně přispěje k realizaci cílů školy. Na základě ředitelkou školy stanovených požadavků došlo k vypracování návrhu, podle kterého byl systém v poslední fázi implementován v podobě webové aplikace. V diplomové práci jsou uvedeny cíle a potřeby školy, které sloužily jako podklad pro vypracování diplomové práce. Došlo k představení architektury aplikace, návrhového vzoru MVC a způsobu komunikace mezi jednotlivými částmi aplikace. Dále byly představeny způsoby a myšlenky různých procesů přispívání. Následně byly uvedeny entity, u kterých se detailně popsaly informace, které se u nich uchovávají. Poslední hlavní kapitolou byla shrnuta implementace a nasazení aplikace, kde byly charakterizovány použité technologie a koncept MERN, jejímž prostřednictvím byla webová aplikace vyvinuta. V poslední řadě byla popsána struktura projektu s jejími dílčími částmi, implementace jednotlivých funkcionalit a nasazení aplikace na platformu Heroku. Bylo by vhodné provést rozsáhlejší uživatelské testování, aby se zjistily nedostatky systému a přibyly návrhy a nápady na vylepšení aplikace. Prozatím testování proběhlo pouze s ředitelkou školy a několika jejími zaměstnanci.

Doporučením pro budoucí vývoj aplikace by mohlo být přidání funkce pro ověření emailu při registraci, aby se zabránilo neoprávněnému přiřazení adresy k dokumentu uživatele. Na stránce s projekty by bylo vhodné administrátorovi umožnit změnit pořadí jednotlivých projektů. Po přidání aktuality by mohl správce zaškrtnout políčko, zda chce zaslat text aktuality i všem přispěvatelům na daný projekt.

Následujícím vylepšením aplikace by mohla být možnost zaslání dárku dárci po darování daru. Mohly by se dárcům zasílat drobné produkty, které jsou vytvořeny dětmi ve výtvarných hodinách, jako např. náramky z korálků. Zasílání dárků by sloužilo jako drobné gesto za laskavé srdce dárců.

V další verzi aplikace je potřeba se zaměřit více na strukturu faktury a vzhled všech generovaných PDF souborů. Dále by bylo vhodné zajistit přísnější validaci dat jak na

straně klienta, tak na straně serveru. V nynější verzi jsou použity základní validační ověření.

Dalším vylepšením by mohlo být automatické vyplnění kontaktních údajů při následující objednávce uživatele. Pokud je uživatel zaregistrován, mohlo by mu být nabídnuto vyplnění údajů na základě předchozí objednávky, kterou na platformě provedl.

Jako poslední vylepšení je uveden příklad, že si dárce při zaškrtnutí anonymního daru může zvolit anonymní označení, které se mu líbí. Mohl by vybírat například z možností, jako jsou „Dobrý anděl“, „Já“, „Dobrá duše“.

9 Seznam použité literatury

[1] Z. speciální a praktická škola D. Vrchlabí, „Speciální škola Vrchlabí", *Základní speciální a praktická škola DČCE Vrchlabí*. https://specialniskolavrchlabi.diakonie.cz/ (viděno 6. leden 2023).

[2] Darujme.cz, „Darujme.cz", *Darujme.cz*. https://www.darujme.cz/ (viděno 5. leden 2023).

[3] D. S. a.s, „Daruj správně - individuální dárcovství", *Daruj správně*. https://www.darujspravne.cz/ (viděno 9. leden 2023).

[4] Donio, „Chcete založit dobročinnou sbírku nebo si splnit vlastní sen?", *Donio*. https://www.donio.cz/navrh (viděno 13. březen 2023).

[5] J. Štráfelda, „Co je backend", *Co je backend*. https://www.strafelda.cz/backend (viděno 12. duben 2023).

[6] V. Subramanian, *Pro MERN stack: full stack web app development with Mongo, Express, React, and Node*. in For professionals by professionals. Berkeley, California: Apress, 2017.

[7] D. Flanagan, *JavaScript: the definitive guide*, Fifth edition. Beijing ; Sepastopol, CA: O'Reilly, 2006.

[8] MongoDB, „JSON And BSON", *MongoDB*. https://www.mongodb.com/json-and-bson (viděno 14. duben 2023).

[9] Expressjs, „Express - Node.js web application framework", *Express - Node.js web application framework*. https://expressjs.com/ (viděno 14. duben 2023).

[10] React, „React", *React*. https://react.dev/ (viděno 14. duben 2023).

[11] Nodejs, „Node.js", *Node.js*. https://nodejs.org/en (viděno 14. duben 2023).

[12] A. Young *et al.*, *Node.js in action*, Second edition. Shelter Island: Manning, 2017.

[13] Mongoosejs, „Mongoose ODM v7.0.2", *Mongoose ODM*. https://mongoosejs.com/ (viděno 22. březen 2023).

[14] M. O. contributors Jacob Thornton, and Bootstrap, „Bootstrap", *Bootstrap*. https://getbootstrap.com/ (viděno 6. duben 2023).

[15] Postman, „Postman | The Collaboration Platform for API Development", *Postman*. https://www.postman.com/ (viděno 28. březen 2023).

[16] Amazon Web Services, Inc., „Cloud Object Storage – Amazon S3 – Amazon Web Services", *Amazon Web Services, Inc.* https://aws.amazon.com/s3/ (viděno 27. březen 2023).

[17] Amazon Web Services, Inc., „Cloud Computing Services - Amazon Web Services (AWS)", *Amazon Web Services, Inc.* https://aws.amazon.com/ (viděno 28. březen 2023).

[18] Stripe, „Stripe | Payment Processing Platform for the Internet", *Stripe | Payment Processing Platform for the Internet*. https://stripe.com/en-cz (viděno 5. duben 2023).

[19] ComGate, „Platební brána a terminály | Comgate", *ComGate Payments*. https://www.comgate.cz (viděno 5. duben 2023).

10 Přílohy

1) Zdrojový kód aplikace

Součástí diplomové práce je i zdrojový kód webové aplikace, který je odevzdán v podobě souboru. Nejaktuálnější verze zdrojových kódů jsou dostupné na adresách:

https://github.com/kumprji1/postav-skolu-fe

https://github.com/kumprji1/postav-skolu-be

Aplikace je pro vyzkoušení zveřejněna na adrese:

https://postav-skolu.herokuapp.com

Pro vyzkoušení funkcí administrátora lze využít dočasného účtu s údaji:

email: **admin@test.cz**

heslo: **Admin123!**

Zadání diplomové práce

Autor:	**Bc. Jiří Kumprecht**
Studium:	I2100069
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Darovací platforma pro vzdělávací instituci**
Název diplomové práce AJ:	Donation platform for an educational institution

Cíl, metody, literatura, předpoklady:

Cíl: Analyzovat, navrhnout a implementovat darovací platformu pro vzdělávací instituci.

Metodika zpracování: Přírůstkový model

Přínos: Škola získá vlastní darovací platformu vyvinutou na míru podle představ ředitele. Pozitivně přispěje k realizaci cílů školy.

Osnova:

1. Úvod
2. Cíl práce
3. Metodika zpracování
4. Analýza
5. Návrh
6. Implementace
7. Shrnutí
8. Závěr a doporučení

1) FLANAGAN, David. JavaScript: the definitive guide. Fifth edition. Sebastopol: O´Reilly, 2006. ISBN 978-059-6101-992.

2) SUBRAMANIAN, Vasan. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Berkeley, CA: Apress, 2017. ISBN 978-1484226520.

3) YOUNG, Alex R., Bradley MECK a Mike CANTELON. Node.js in action. Second edition. Shelter Island: Manning, [2017]. ISBN 16-172-9257-5.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D.
Datum zadání závěrečné práce:	15.10.2021