VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PODPORA PĚSTOVÁNÍ VE MĚSTECH POMOCÍ WEBOVÉ APLIKACE

SUPPORT FOR PLANTING IN CITIES USING A WEB APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ALEXANDR ČELAKOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. DAVID BAŽOUT,**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

156254

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Čelakovský Alexandr**
Program:	Informační technologie
Název:	**Podpora pěstování ve městech pomocí webové aplikace**
Kategorie:	Webové aplikace
Akademický rok:	2023/24

Zadání:

1. Prozkoumejte problematiku městského pěstování a analyzujte uživatelské požadavky pro jeho správu.
2. Navrhněte vhodné uživatelské rozhraní reflektující požadavky ze strany uživatelů s prvky gamifikace.
3. Prozkoumejte existující front-end frameworky, vyberte vhodnou technologii a proveďte implementaci aplikace.
4. Otestujte UI na reálných úlohách a proveďte optimalizace.
5. Výsledky práce zdokumentujte formou plakátu a krátkého videa.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bažout David, Ing.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	9.11.2023

Abstrakt

Cílem této bakalářské práce je návrh a implementace webové aplikace pro chytrý skleník s prvky gamifikace. Mezi hlavní části systému patří pronájem záhonků v jednotlivých sklenících a chytré tržiště, které je plně automatizované. Backend je implementován ve frameworku Django a frontend je napsán v knihovně React. Byl zde kladen důraz na co největší automatizaci a jednoduchost uživatelského prostředí. Výsledkem práce je webová aplikace, která splňuje požadavky. Testy ukázaly, že výsledné řešení je funkční a uživatelsky přívětivé.

Abstract

The goal of this bachelor thesis is design and implementation of a web application for a smart greenhouse with elements of gamification. The main parts of the system include rental of flowerbeds in individual greenhouses and a smart marketplace that is fully automated. The backend is implemented in the Django framework and the frontend is written in React. Emphasis was placed on the greatest possible automation and simplicity of the user interface. The result of the work is a web application that meets the requirements. Tests have shown that the final solution is functional and user-friendly.

Klíčová slova

webová aplikace, skleník, tržiště, gamifikace, Django, Python, PostgreSQL, React, Typescript

Keywords

web app, greenhouse, marketplace, gamification, Django, Python, PostgreSQL, React, Typescript

Citace

ČELAKOVSKÝ, Alexandr. *Podpora pěstování ve městech pomocí webové aplikace*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. David Bažout,

Podpora pěstování ve městech pomocí webové aplikace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bažouta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Alexandr Čelakovský
9. května 2024

Poděkování

Tímto bych chtěl poděkovat Ing. Davidu Bažoutovi za skvělé vedení práce a za jeho cenné rady a připomínky. Dále bych chtěl poděkovat své rodině a přátelům za testování aplikace a za poskytnutí zpětné vazby.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je návrh a implementace webové aplikace pro chytrý skleník s prvky gamifikace.

Hlavní cílovou skupinou pro tuto aplikaci jsou lidé, kteří nemají vhodné prostory pro pěstování vlastních plodin nebo nemají čas na údržbu záhonků. Dalším kritériem může být, že nechtějí konzumovat potraviny pouze z obchodu, ale touží si vypěstovat plodiny vlastní, bez chemické úpravy. Tato aplikace může oslovit jak laiky a začátečníky, tak i zkušené pěstitele a může jim přinést i nečekané úspěchy, které je dozajista potěší. Nesporným přínosem je ušetření nákladů oproti nákupu z obchodu a také je tento přístup šetrnější k životnímu prostředí.

Aplikace bude sloužit nejen k pronájmu záhonků zákazníkům aplikace, ale také nabízí tržiště, ve kterém si zákazníci mohou nakoupit semínka, samostatné rostliny, případně nástroje a pomůcky pro obdělávání. Snahou bylo vložit prvky gamifikace, které zákazníka motivují a posouvají ho blíže k jeho osobnímu cíli. Dalším uživatelem této aplikace je správce skleníku, jeho rolí je starat se o úkony, které nelze automatizovat. Provozovatel skleníku získává monitoringem dokonalý přehled o stavu svého konkrétního skleníku. Důležité bylo všechny úkony co nejvíce optimalizovat, aby došlo k ušetření nákladů a služba nevyžadovala zaměstnance na plný úvazek.

Kapitola 2 se věnuje analýze požadavků. Nejprve je představen dosavadní systém, poté jsou upřesněny jednotlivé požadavky.

Následující kapitola 3, vysvětlí gamifikaci a její prvky. Jsou zde představeny prvky, které jsou v aplikaci využity.

V kapitole 4 je detailně popsán postup návrhu aplikace. Popisuji jednotlivé části aplikace jako platební systém, chytré tržiště a další. Kapitola je zakončena návrhem databáze.

Seznámení s technologiemi, které byly využity při implementaci aplikace, nalezneme v kapitole 5. Proběhne zde výběr technologií a zdůvodnění jejich výběru.

Kapitola 6 se věnuje implementaci aplikace. Popisuje architekturu aplikace a jednotlivé části aplikace.

Výsledky uživatelského testování se nachází v kapitole 7. Jsou zde popsány jednotlivá kola testování a zpětná vazba.

V poslední kapitole 8 shrnuji výsledky práce a zhodnocuji, zda byly splněny všechny požadavky a cíle práce.

Kapitola 2

Analýza požadavků

Před samotným návrhem a implementací aplikace je nutné si ujasnit všechny požadavky, které musí aplikace splňovat. Nejprve je představen dosavadní systém, poté jsou upřesněny jednotlivé požadavky a jednotlivé uživatelské role. Z jednotlivých rolí je vytvořen diagram případů užití.

2.1 Dosavadní systém

Předchozí řešení nebylo úplné, nicméně plnilo pár požadavků. Byla to tabulka, se seznamem nájemců. Specifikovala období pronájmu, jeho cenu a název záhonku. Pomocí Javascriptu bylo možné ručně odeslat mail s upomínkou o další platbě a poté bylo potřeba kontrolovat jednotlivé platby. Toto řešení bylo dostatečné v malém měřítku, ale při rostoucím počtu zákazníků a skleníků by mohly nastat komplikace a správa by byla velmi časově náročná. Mnoho potencionálních zakazníků také mohlo odradit napsaní e-mailu s žádostí o pronájem záhonku. Tato aplikace umožní všechnu zmíněnou funkcionalitu plně automatizovaně a obohatí ji o nové prvky.

2.2 Skleník

Jedná se o nejdůležitější část aplikace. Bez skleníku by nebylo možné pronajímat záhonky ani nabízet produkty na tržišti. Jedná se o skutečný fyzický objekt, který je umístěn na určitém místě. Příklad skleníku můžete vidět na obrázku 2.1. Pro správnou a jednoznačnou identifikaci skleníku je potřeba znát jeho název a adresu. Dále počet jednotlivých záhonků a zda jsou volné či obsazeny. Každý skleník má své tržiště. Správce či provozovatel musí být schopen skleník spravovat a mít přehled o jeho stavu.

2.3 Záhonek

Váže se k němu skleník, ve kterém se nachází. Má svůj název podle kterého lze jednoduše identifikovat. Skleník je buď obsazený nebo volný. Pokud je volný tak si ho zákazník může na určitou dobu pronajmout a sadit v něm své plodiny. Specifikuje své rozměry, aby mohl potencionální zájemce počítat s tím, jak velká plodina lze nasadit. Provozovatel může uvést, které plodiny jsou pro tento záhonek ideální a jaké nástroje jsou dostupné uvnitř skleníku. Obrázek 2.2 zobrazuje záhonek ve skleníku.

Obrázek 2.1: Chytrý skleník, převzato z [16].

Deník slouží k zápisu svých poznámek, které se týkají daného záhonku. Může se jednat o zápisy sazení, hnojení a stavu plodin. Taktéž zde může zákazník zaznamenávat své poznatky o pěstování a jeho další plány pro záhonek.

Historie sklizně umožní zákazníkovi sledovat svou úrodu na jednom místě. Specifikuje zde plodinu, datum sklizně a množství sklizených plodin.

2.4 Pronájem záhonku

Pronájem záhonku je spojen s určitým záhonkem a zákazníkem. Jeden záhonek může být pronajat pouze jednomu zákazníkovi v jeden čas. Zároveň je zde specifikováno, kdy začíná a končí pronájem. Před vypršením pronájmu ho může zákazník prodloužit o určité časové období, jinak pronájem končí a záhonek může být pronajat zákazníkovi jinému.

Každý záhonek má svou cenu za určitou jednotku času, která se mezi záhonky v jednom skleníku může lišit, např. podle velikosti záhonku. Zákazník si může záhonek pronajmout na určitou dobu. Cena pronájmu se vypočítá podle ceny za jednotku času a délky pronájmu.

2.5 Tržiště

Je to místo, kde provozovatel nabízí své produkty. Může se jednat o semínka, nástroje pro obdělávání záhonků a případně i o samotné plodiny. Zákazník si může vybrat z nabídky a přidat si je do košíku. Poté je možné je zakoupit. Celkový postup musí být plně automatizován.

Obrázek 2.2: Záhonek ve skleníku.

Nejprve je potřeba vytvořit jednotlivé produkty a nastavit jim svou cenu. Poté nastavit jejich kvantitu, která je k dispozici. Pokud je nějaký produkt vyprodán, tak se zobrazí jako nedostupný.

2.6 Objednávky

Zde může uživatel zkontrolovat své objednávky jak produktů z tržiště ale i pronájmů. Každá objednávka musí mít svůj stav. Váže se na zákazníka a obsahuje všechny zakoupené produkty a finální cenu celé objednávky. V případě nezaplacení musí být zákazník o této informaci obeznámen a vyzvat ho případně i opakovaně k platbě.

2.7 Výkaz práce

Správce skleníku si zapíše svou práci, kterou vykonal. Může se jednat o doplnění zásob a aktualizi inventáře tržiště, kontrolu stavu skleníku a přípravu záhonku pro nového pronajímatele.

Při zápisu upřesní datum, čas a popis práce, kterou vykonal. Provozovatel poté výkaz schválí nebo zamítne.

2.8 Statistiky

Statistiky jsou důležitou součástí pro provozovatele. Může zde sledovat výdělky za určité období, počet objednávek, počty prodaných jednotlivých produktů, pronájmy a počet zákazníků.

2.9 Role uživatelů

Uživatelé jsou rozděleni do několika rolí, kde každá role umožňuje jiné operace. Aplikace obsahuje následující role:

- Zákazník
- Správce
- Provozovatel
- Administrátor

Zákazník

Každý nový uživatel získává tuto roli automaticky. Umožňuje nákup na tržišti, pronájem záhonku a spravovat vlastní záhonek. Může si zobrazit své objednávky a jejich aktuální stav a obsah objednávky.

Správce

Správce má na starosti správu skleníku. Může zde upravovat informace o skleníku, přidávat nové záhonky a upravovat jejich informace. Spravuje také tržiště, kde může přidávat nové produkty a k nim fotky, popis a jejich aktuální kvantitu. Zapisuje svou práci, aby mohl být případně podle ní ohodnocen.

Provozovatel

Je vlastníkem jednoho či více skleníků a má všechny pravomoce správce. Může si také zobrazit statistiky, které se týkají jeho skleníku. Volí si správce skleníku.

Administrátor

Má všechny pravomoce. Navíc vytváří nové skleníky a nastavuje jejich provozovatele. Má přehled o všech uživatelích aplikace a má možnost je upravit. Zobrazí si všechny objednávky a případně je upravuje.

2.10 Diagram případů užití

Z pohledu uživatele zobrazuje jak se aplikace chová. Slouží k popisu funkcionality systému. Specifikuje jednotlivé požadavky, ale neříká jak je systém implementuje. Často je prvním diagramem, který ve fázi návrhu vznikne. Obsahuje aktéry, případy užití a vztahy mezi nimi. [10]

Při tvorbě diagramu bylo nutné uvažovat ještě o jedné roli. Uživatel, který není přihlášen nebo nemá žádný učet musí mít možnost do aplikace vstoupit. Proto diagram obsahuje dalšího aktéra, kterým je návštěvník. Diagram případů užití naleznete v příloze A.

Kapitola 3

Gamifikace

Tato kapitola se nejpve věnuje vysvětlení pojmu gamifikace a její historii. Na příkladech vysvětluje, jak je tento pojem využit v různých oblastech a jaké jsou její výhody. Dále bude objasněno, jaké prvky gamifikace budou využity v této aplikaci.

3.1 Definice a historie gamifikace

Existuje mnoho definic gamifikace.

Gamifikací se rozumí využití herního myšlení a herních prvků k řešení problémů, motivaci a podpoře učení. [13]

Gamifikace je obohacení neherního prostředí o herní prvky. [2]

Z těchto definic je patrné, že gamifikace může změnit průběh a výsledek určité činnosti. Může zvýšit motivaci, zábavu a zapojení uživatelů.

Historie výrazu gamifikace nesahá do dávné minulosti.

„Začátkem roku 2000 začali někteří herní designéři přemýšlet o způsobech, jak by mohlo být vzrušení a radost z hraní her přeneseno do reálného světa.(McGonigal, 2011). Termín „gamifikace“ byl přijat většinou výzkumníků a praktiků, původně jej vytvořil Nick Pelling v roce 2002 (Pelling, 2011)“ [18].

3.2 Prvky gamifikace

Mezi typické prvky gamifikace patří [19]:

- Body – Jsou udělovány v rámci aplikace.
- Tabulka výsledků – Uživatelé se mohou porovnávat s ostatními.
- Úrovně – Určují aktivitu uživatele.
- Systém odměn – Slouží k rozšíření cílů uživatele.

3.3 Gamifikace kolem nás

S gamifikací se můžeme setkat každý den. Je využívána v mnoha oblastech, jako jsou marketing, vzdělávání, zdravotnictví a mnoho dalších.

Marketing

Zde je gamifikace využita k získání nových zákazníků a udržení těch stávajících. Může se jednat o soutěže, kvízy a odměny za nákupy. Prvním příkladem mohou být eshopy, které nabízí slevy za první nákup. Každým dalším nákupem mu přibývají body, které může nakonec vyměnit za slevu na další nákup. Obchodní řetězce často využívají soutěže o ceny, které zákazníky motivují k nákupu. Zákazníci opět sbírají body za nákupy, více bodů znamená větší šanci na výhru. V případě nevýhry hlavní ceny mohou zkusit své štěstí u cen menších nebo body vyměnit za slevu.

Vzdělávání

Gamifikace neslouží pouze ke zvýšení výdělku, ale ve vzdělávání může zvýšit motivaci žáků k učení.

S herními prvky se žáci setkávají již zcela běžně ve škole. Mohou se s nima setkat například ve virtuálních prostředích, které jim pomáhají objevovat nové informace a upoutat jejich pozornost. [7]

Platforma Moodle[1] například umožňuje vytvářet kvízy. Je zde možnost zobrazit tabulku s nejlepšími výsledky, což může žáky motivovat k lepším výsledkům. Populární rozšíření do Moodle je "Level Up XP - Gamification"[2], které díky bodům, úrovním a tabulce výsledků může motivovat žáky k lepším výsledkům.

Aplikace Duolingo[3] je dalším příkladem gamifikace ve vzdělání. Slouží převážně k nauce jazyků. Za každou správnou odpověď získává uživatel body a postupuje na vyšší úrovně. Mezi další prvky patří počítadlo série dnů, které uživatel nepřetržitě cvičí. Je zde také možnost soutěžit s přáteli, což může být další motivací k učení.

Zdravotnictví a fitness

Jedním z příkladů gamifikace jsou aplikace na sledování své fyzické aktivity za pomoci chytrých hodinek nebo chytrých náramků. Mezi nejznámější patří Strava[4]. Uživatelé zde mohou zaznamenávat své aktivity, jako cyklistiku, běh, posilování a celou řadu dalších aktivit. Tyto aktivity jsou zobrazeny ve grafech, které mohou porovnávat s ostatními uživateli. Mezi další prvky patří výzvy, které mohou uživatele motivovat k další aktivitě.

Aplikace pro sledování půstů, jako je například Zero[5], umožňuji uživatelům zapisovat své půsty a sledovat jak dlouho půst drží. Příkladem gamifikace je zde zobrazení celkového počtu lidí, kteří aktuálně půst drží.

[1] https://moodle.org/
[2] https://moodle.org/plugins/block_xp
[3] https://www.duolingo.com/
[4] https://www.strava.com/
[5] https://zerolongevity.com/

3.4 Gamifikace v aplikaci

Tato sekce pojednává o využitých prvcích gamifikace v této aplikaci.

Odznaky

Systém odznaků odměňuje uživatele za splnění určitých úkolů. Každá typ odznaků obsahuje několik úrovní. Jsou navrženy tak, aby je nebylo možné splnit úplně všechny v krátkém časovém období, ale aby uživatelé měli motivaci se vracet a plnit další úkoly. Každý odznak má svou úroveň, název, popis a počet bodů zkušeností.

Typy odznaků jsou:

- Záhonky
- Nákupy
- Ušetřená uhlíková stopa
- Ušetřené peníze
- Speciální odznaky

Záhonky

První úroveň by měla být snadno dosažitelná, aby uživatelé měli motivaci pokračovat. Postupně úroveň stoupá až k nejtěžší úrovni. Poslední úrovně dosáhnou pouze ti nejnáročnější uživatelé, kterým pár záhonků nestačí. Odměny jsou uděleny po uhrazení částky za pronájem záhonku. Seznam odznaků za záhonky naleznete v tabulce 3.1.

Úroveň	Název	Popis	Body zkušeností
1	Začátečník	První záhonek	10
2	Zkušený zahrádkář	Dva záhonky	20
3	Profesionál	Tři záhonky	30
4	Expert	Čtyři záhonky	50
5	Maestro	Pět záhonků	100

Tabulka 3.1: Odznaky za záhonky.

Nákupy

Tenhle typ odznaku uživatel získá po nákupu na tržišti. Odznaky nejsou na základě počtu objednávek, ale za určitou částku, kterou uživatel na tržišti utratil. Takto je motivován k dalším nákupům. Tabulka 3.2 ukazuje jednotlivé úrovně odznaků za nákupy.

Ušetřená uhlíková stopa

Odznak je založen na odhadované ušetřené uhlíkové stopě. Nejsou za něj udělovány body zkušeností, jelikož skutečná sklizeň není nijak kontrolována. Jednotlivé úrovně odznaků za ušetřenou uhlíkovou stopu jsou zobrazeny v tabulce 3.3.

Úroveň	Název	Popis	Body zkušeností
1	Malé prasátko	100 Kč	10
2	Peněženka	500 Kč	20
3	Velká peněženka	1000 Kč	30
4	Zlato	5000 Kč	50
5	Diamant	10000 Kč	100

Tabulka 3.2: Odznaky za nákupy.

Úroveň	Název	Popis
1	Nováček ekolog	1 kg CO2
2	Zelený aktivista	5 kg CO2
3	Ekolog	10 kg CO2
4	Ekologický mistr	50 kg CO2
5	Eko-hrdina	100 kg CO2

Tabulka 3.3: Odznaky za ušetřenou uhlíkovou stopu.

Ušetřené peníze

Podobně jako u ušetřené uhlíkové stopy, ani zde nejsou udělovány body zkušeností. Jednotlivé úrovně odznaků za ušetřené peníze jsou zobrazeny v tabulce 3.4.

Úroveň	Název	Popis
1	Začátečník spořitel	100 Kč
2	Úsporný spořitel	500 Kč
3	Mistr spoření	1000 Kč
4	Velmistr spoření	2500 Kč
5	Šampion spoření	5000 Kč

Tabulka 3.4: Odznaky za ušetřené peníze

Speciální odznaky

Speciální odznaky jsou udělovány za speciální úkoly. Například za odběr novinek ohledně skleníku, za sdílení svého příspěvku na komunitní zdi a každoročně za výročí registrace.

Rarita odznaků

Každý odznak má svou raritu. Je vytvořena podle vzorce, který bere v potaz, kolik uživatelů má daný odznak. Čím méně uživatelů má daný odznak, tím je vzácnější. Rarita je vyjádřena v procentech, tedy kolik procent uživatelů má daný odznak. Nižší hodnota znamená vzácnější odznak. Pokud uživatelé uvidí, že jejich odznak má nízkou raritu, může to být další motivace k plnění dalších úkolů.

Ušetřená uhlíková stopa a peníze u sklizně

Na základě sklizně záhonku lze odhadnout, jaké množství uhlíkové stopy a peněz bylo ušetřeno.

Státní zemědělský intervenční fond několikrát do roka[6] zveřejňuje průměrné ceny zeleniny a ovoce. Tyto ceny jsou základem pro výpočet ušetřených peněz. Aby uživatel nemusel jednotlivé plodiny vážit, počítáme s průměrnou váhou na jeden kus plodiny.

V případě ušetřené uhlíkové stopy je odhadováno, kolik CO_2 bylo ušetřeno díky tomu, že zákazník si nekoupil zeleninu a ovoce v obchodě, ale vypěstoval si ji v jednom ze skleníků. Ušetřenou uhlíkovou stopu nastaví správce skleníku pro každou plodinu a v průběhu roku je třeba tyto hodnoty aktualizovat.

Na obrázku 3.1 je návrh, jak by mohla vypadat ušetřená uhlíková stopa a peníze.

Obrázek 3.1: Návrh ušetřené uhlíkové stopy a peněz, hodnoty jsou pouze ilustrační.

Podle [6] vyprodukuje auto z roku 2023 za 1 km jízdy 137 gramů CO_2. Tuto informaci můžeme použít pro přirovnání ušetřené uhlíkové stopy k cestě autem.

Newsletter

Administrátor aplikace může zasílat novinky ohledně skleníku formou e-mailu. Uživatelé, kteří se přihlásí k odběru novinek, mohou včetně novinek ohledně aplikace, skleníků a tržiště získat také slevový kód, který mohou uplatnit na tržišti. Tímto způsobem může být zákazník motivován k odběru novinek.

Komunitní zeď

V dnešní době, kdy jsou sociální sítě velmi populární, je důležité této skutečnosti využít i v této aplikaci. Uživatelé vytvoří příspěvek na jejich profilu, který bude zmiňovat určitý skleník nebo tržiště. Poté v aplikaci požádají o schválení příspěvku na komunitní zdi. Komunitní zeď je prostor na hlavní stránce, ve kterém je zobrazeno 5 náhodných schválených příspěvků. Tímto způsobem bude probíhat marketing dvěma směry. Uživatelé budou sdílet své příspěvky na svých sociálních sítích a takhle se dostane informace o skleníku k dalším lidem. Zároveň se zvýší motivace uživatelů k vytváření nových příspěvků, jelikož se mo-

[6] https://www.szif.cz/cs/zpravy-o-trhu?cdr=09&year=2024&ino=0

hou dostat na komunitní zeď. Taktéž návštěvníci hlavní stránky mohou získat nová spojení s lidmi, kteří mají stejné zájmy.

Systém úrovní

Systém úrovní je založen na získaných bodech zkušeností. Každá úroveň má svůj název a počet bodů zkušeností, kterých je potřeba dosáhnout k získání této úrovně. Každá úroveň je náročnější než ta předchozí. Uživatelé mohou získat body zkušeností za získání odznaků.

Za dosažení nové úrovně získá uživatel odměnu. Může se jednat o slevový kód na tržišti nebo odznak. Slevový kód obdrží formou e-mailu a odznak se zobrazí na stránce s odznaky.

Obrázek 3.2: Návrh systému úrovní a odznaků.

Kapitola 4

Návrh řešení

Po zjištění požadavků na aplikaci není ideální začít ihned s vývojem. Proto se tato kapitola věnuje návrhu webové aplikace. Návrh je důležitý pro vývojáře i pro zákazníka. Díky návrhu může zákazník vidět, jak aplikace bude vypadat a jak bude fungovat. V této fázi je důležitá komunikace se zákazníkem, protože změny v návrhu jsou mnohem jednodušší než v implementaci. Správný návrh může ušetřit mnoho času a peněz. Prototyp je skvělým nástrojem pro zjištění, zda se návrh ubírá správným směrem. Nemusí se jednat o funkční prototyp, ale i statický prototyp je dostačující. Často je prototyp pouze obrázek, který obsahuje pouze rozložení prvků na stránce.

Nejprve proběhne návrh uživatelského rozhraní v sekci 4.1. Další sekce 4.2 pojednává o návrhu jednotlivých částí, které budou klíčové pro implementaci aplikace. Návrh je zakončen návrhem jednotlivých tabulek databáze a vztahy mezi nimi. Schéma databáze je v sekci 4.3.

4.1 Návrh uživatelského rozhraní

Uživatelské rozhraní je částí aplikace se kterou uživatelé přímo komunikují. Rozhraní musí být pro uživatele co nejvíce přívětivé a každá jeho část zcela jasná. Jeho ovládání musí být intuitivní a poskytovat okamžitou zpětnou vazbu po každé interakci s ním. Uživatel by neměl být schopen zadat neplatné údaje nebo se dostat do situace, kdy neví, jak dále pokračovat. Tomu může zabránit například zobrazení chybové hlášky, která uživatele navede jak dosáhnout správného stavu. Nejlepším řešením je vytvoření takového uživatelského rozhraní, které takovou situaci nedovolí.

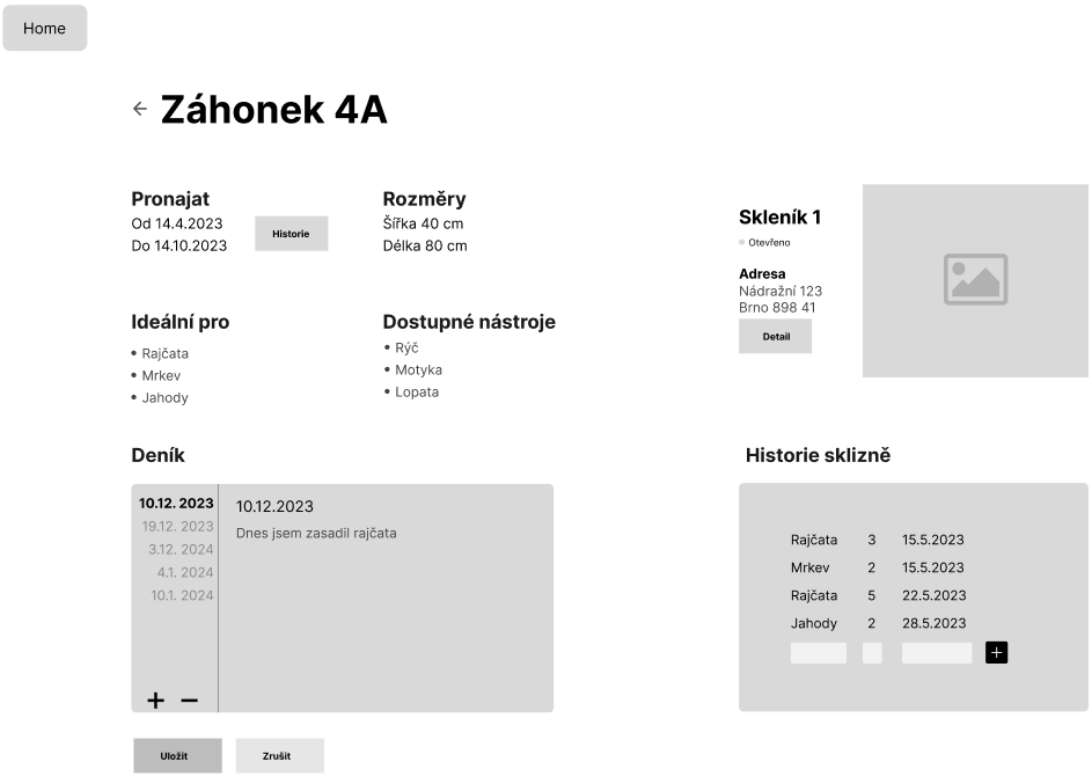
Návrh probíhal pomocí aplikace Figma[1]. Jedná se o aplikaci, která slouží nejčastější k tvorbě designu uživatelského rozhraní jak pro webové aplikace, tak pro mobilní zařízení a jiné platformy. Pracuje na bázi vektorové grafiky, což umožňuje vytvářet design, který je možné zvětšit i zmenšit bez ztráty kvality. V případě této aplikace jsem kromě nástrojů pro tvorbu návrhu využil i nástroj tvorby prototypu. Ten mi umožnil vytvořit interaktivní prototyp, který byl užitečný jak při ověřování správnosti návrhu, ale taky u testování. Prototyp sloužil k získání zpětné vazby a vedl k následné úpravě návrhu. Výsledný návrh vznikl jako wireframe, který obsahoval konkrétnější strukturu.

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

Wireframe

Jedná se o vizuální reprezentaci bez grafiky, barev nebo obsahu. Slouží k zobrazení struktury, rozložení a funkčnosti stránky při návrhu uživatelského rozhraní a tím zjistit, zda je uživatelský zážitek intuitivní. [17]

Příklad návrhu pro záhonek naleznete na obrázku 4.1. Zde je zobrazena stránka, kde uživatel může zobrazit detaily o záhonku. Obsahuje důležité informace jako název, popis skleníku ve kterém se nachází a období, ve kterém je záhonek pronajat. Deník pro zápis poznámek a historii sklizně pro sledování sklizených plodin.



Obrázek 4.1: Návrh záhonku vytvořený v nástroji Figma.

Struktura stránek

Z vytvořeného návrhu vznikla struktura stránek, která je zobrazena v tabulce 4.1. Pro přístup k odkazům, které začínají `/app/` je nutné být přihlášený a některé vyžadují určitá oprávnění. Ostatní odkazy jsou dostupné pro všechny uživatele.

Stránka	Url
Úvodní stránka	`/`
Přihlášení	`/signin`
Registrace	`/signup`
Reset hesla	`/reset-password`
Profil	`/app/profile`
Výpis skleníků	`/app/greenhouses`
Mé skleníky	`/app/my-greenhouses`
Detail skleníku	`/app/greenhouses/{id}`
Mé záhonky	`/app/my-beds`
Detail záhonku	`/app/beds/{id}`
Pronájem záhonku	`/app/beds/{id}/rent`
Prodloužení pronájmu záhonku	`/app/beds/{id}/extend-rent`
Výpis objednávek	`/app/orders`
Detail objednávky	`/app/orders/{id}`
Tržiště	`/app/marketplace`
Detail produktu	`/app/marketplace/products/{id}`
Košík	`/app/marketplace/cart`
Seznam uživatelů	`/app/users`
Detail uživatele	`/app/users/{id}`
Výpis odznaků	`/app/badges`
Výkazy práce	`/app/timesheets`
Detail výkazu práce	`/app/timesheets/{id}`
Tvorba newsletter	`/app/newsletter`
Správa komunitní zdi	`/app/socialposts`
Detail příspěvku na komunitní zdi	`/app/socialposts/{id}`

Tabulka 4.1: Struktura stránek s odkazy.

4.2 Návrh jednotlivých částí

Následující podsekce se věnují jednotlivým částem aplikace, které je nutné správně navrhnout, aby při implementaci nedošlo k omylu. Mezi tyto části patří například návrh způsobu platby, ověření platby, schvalovací proces pracovního výkazu a další.

Platby

Platby jsou jednou z nejdůležitější součástí tohoto systému. Proto je nutné vybrat vhodný druh platby. Nejprve představím typické možnosti se kterými se lidé setkávají v e-shopech a zmíním jejich výhody a nevýhody. Z nich poté vyberu ten nejvhodnější pro tuto aplikaci.

Druhy platby

Podle Heureka [14] v roce 2018 byla nejčastěji použita platba dobírkou. Nedaleko za ní byla platba při převzetí. Následovala online platba kartou. Znatelně menší podíl měla platba bankovním převodem a mezi poslední patřila platba přes PayPal, splátky a jiné formy platby.

Dobírka a platba při převzetí

Dobírka je druh platby, kdy zákazník zaplatí za zboží až při jeho převzetí. Tato forma platby je vhodná pro zákazníky, kteří nechtějí platit předem. Jednou z nevýhod pro prodejce je nutnost zaměstnat osobu, která bude vybírat platbu. V případě skleníku by se jednalo o zaměstnance na poloviční úvazek, který by ve skleníku vybíral platby za pronájem záhonků a produktů z tržiště. Takového zaměstnance by ideálně potřeboval každý skleník.

Online platba kartou

Tento způsob platby je vhodný pro zákazníky, kterým nevadí zboží zaplatit předem. Často je výhodou rychlejší doručení a předání zboží. Vrácení peněz je jednodušší než u dobírky, protože prodejce již zná kartu zákazníka, kterou platil. Jednou z nevýhod pro prodejce jsou poplatky za platební bránu. V případě skleníku by bylo nutné tuto bránu integrovat do systému.

Bankovní převod

Způsob platby, kdy se zákazník nemusí bát o bezpečnost údajů své bankovní karty. Stačí znát číslo účtu a variabilní symbol. Výhodou je, že prodejce nemusí hradit poplatky jako u předchozího druhu platby. Velká nevýhoda je délka platby. Dnes ne každá banka podporuje okamžitý bankovní převod a platba by mohla trvat i několik dní. V našem případě by byla nutnost automaticky kontrolovat všechny platby na určitém účtu.

PayPal

PayPal[2] se ze zmíněných způsobů platby nejblíže podobá online platbě kartou. S rozdílem, že zákazník nezadává údajé své karty, nýbrž se přihlásí do svého PayPal účtu, kde má buď uloženou kartu nebo peníze. Platba proběhne ihned, ale opět jsou zde poplatky za platbu a nutnost integrovat PayPal do systému. Nejedná se ani o nejčastější způsob platby v České republice.

Výběr druhu platby

Mezi požadavky patří plná automatizace aplikace. Proto je nutné zvolit způsob platby, který bude nejvíce automatizovaný. Dobírka a platba při převzetí odpadají, jelikož by bylo nutné zaměstnat dalšího člověka. Zbývají tedy online platba kartou, bankovní převod a PayPal. Nutnost platit poplatky za platební bránu a PayPal není ideální z důvodu snížení nákladů. Zbyl tedy bankovní převod. Tento způsob platby je lze nejjednoduše implementovat a nevyžaduje žádné poplatky a nutnost zaměstnávat dalšího člověka. Nicméně bylo nutné vyřešit problém s délkou platby. Zákazník by takto mohl být nespokojený, jelikož by musel čekat několik dní na potvrzení platby a následně na pronájem záhonku nebo vyzvednutí zboží. Byl zde zvolen kompromis. Zákazník bude moci záhonek využívat ihned po vytvoření objednávky. Podobně u tržiště bude schopen vyzvednout zboží okamžitě. Pokud platba nebude potvrzena další den, tak bude upomenut opakovaně e-mailem. K identifikaci platby bude sloužit variabilní symbol, který bude zákazníkovi zobrazen při vytvoření objednávky.

[2] https://www.paypal.com/

Ověření platby

Předchozí sekce se zabývala výběrem druhu platby. Nyní musíme vyřešit, jak platbu ověřit. Jelikož platba nemusí přijít okamžitě, musíme periodicky kontrolovat účet, na který byla platba zaslána. Proces kontroly plateb je nutné automatizovat. Při přijetí platby bude objednávka označena jako zaplacená. Nejprve rozeberu možnosti jakým způsobem můžeme jednotlivé platby kontrolovat a nakonec vyberu nejvhodnější způsob.

Manuální kontrola

Jedná se o nejjednodušší způsob ověření platby. Zaměstnanec musí pravidelně kontrolovat každou platbu a ručně ji v aplikaci označit jako zaplacenou. Může zde dojít k velkému počtu chyb a je velmi pracný. Velké množství objednávek by pravděpodobně vyžadovalo dalšího zaměstnance. Pokud by byla platba úspěšně provedena jako okamžitá, tak by ji uživatel nemusel okamžitě vidět v aplikaci jako zaplacenou.

Dativery

Tento systém[3] poskytuje službu, která umožní propojení téměř každého bankovního účtu. Vývojářům poskytuje unifikované api pro získání výpisů z bankovních účtů bez nutnosti žádat o speciální licenci. Nejedná se o službu zdarma.

API Komerční banky

Komerční banka poskytuje otevřené bankovnictví[4], pomocí kterého bychom mohli získat výpis plateb. Bohužel je zde nutno vygenerovat certifikát pro organizaci a následně požádat o přístup k API. Ceník služby se mi nepodařilo nalézt. Vyžaduje účet v Komerční bance.

API Fiobanky

Fiobanka nabízí API[5], které umožňuje výpis plateb. Tato služba je zcela zdarma a není nutné generovat certifikát ani žádat o přístup. Opět by ale došlo k nutnosti se vázat na jednu banku.

Nejvhodnější způsob ověření platby

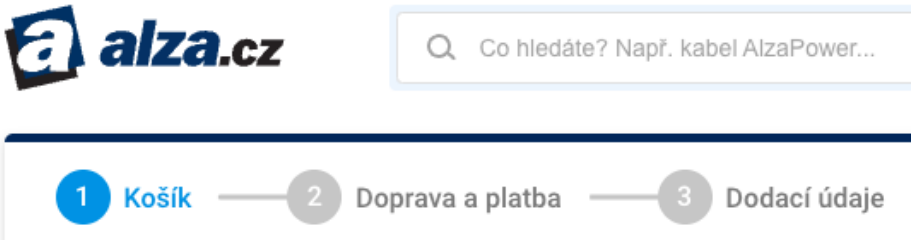
Manuální kontrola by připadala v úvahu pouze při prvotním nasazení systému, kdy by nemuselo být mnoho objednávek a mohl by je kontrolovat administrátor. V případě rychlého růstu by nakonec musela být zvolena jedna z automatizovaných služeb. Z možných variant vyhrála Fiobanka. Je zcela zdarma a není nutné žádat o přístup, který by mohl trvat neznámou dobu. Bylo nutné se vzdát velkého výběru bank, ale nejednalo se o velkou překážku.

Způsob vytvoření objednávky

Zde je možné vytvořit dva typy objednávek. Buď zákazník objednává určité produkty z tržiště nebo si pronajímá záhonek. Jelikož se jedná o stejnou operaci, je nutné stvořit jednotný

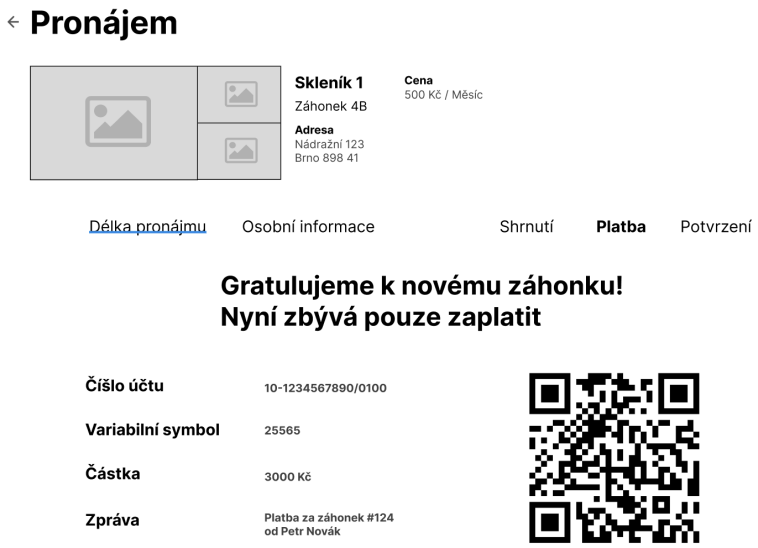
[3] https://bankovni-vypisy.cz/
[4] https://www.kb.cz/cs/kbapi/o-otevrenem-bankovnictvi
[5] https://www.fio.cz/bankovni-sluzby/api-bankovnictvi

formulář pro oba typy objednávek, aby uživatel nebyl překvapen z kompletně rozdílných postupů, které vedou ke vytvoření objednávky. Dnes se často setkáváme s formuláři, které obsahují více kroků. Tento způsob je vhodný pro delší formuláře, kde by zákazník mohl chybovat na více místech a musel by se vracet zpět. Formulář s více kroky využívá například Alza[6] na obrázku 4.2. Zde je formulář rozdělen na 3 kroky. Nejpve zákazník vloží zboží do košíku, poté vybere způsob dopravy a platby a nakonec potvrdí své dodací údaje a vytvoří objednávku. Podobný formulář nalezneme i na webu Mall[7].



Obrázek 4.2: Formulář s více kroky na webu Alza.

Jedním z posledních kroků formuláře bude v obou případech zaplacení objednávky. Jelikož byl zvolen bankovní převod jako jediná platební metoda. Musíme zobrazit zákazníkovi informace, které bude potřebovat k provedení platby. Musíme zobrazit číslo účtu, na který má provést platbu, variabilní symbol a částku, kterou má zaplatit. Pro usnadnění práce zákazníkovi by bylo ideální zobrazit QR kód platebního příkazu. Ten si buď naskenuje pomocí kamery nebo načte z uložené fotky ve své bankovní aplikaci. Tím se zjednoduší zadání údajů a zároveň se sníží možnost chyby. Návrh zobrazení platby naleznete na obrázku 4.3.



Obrázek 4.3: Návrh zobrazení platby.

[6] https://www.alza.cz/
[7] https://www.mall.cz/

Pronájem záhonku

Jedná se o klíčovou funkci tohoto této aplikace a je nutné ji správně navrhnout. Uživatel nejprve musí navštívit záhonek o který má zájem. Záhonek musí být aktivní a nesmí být pronajatý. V případě, že je volný, tak uživatel může stisknout tlačítko „Pronajmout“. Zobrazí se mu formulář, který byl zmíněn v předchozí podsekci. Obsahuje celkem 3 kroky. V prvním kroce uživatel zvolí období, na které chce záhonek pronajmout. V druhém kroku uživatel potvrdí své údaje a taky potvrdí období pronájmu spolu s cenou. Posledním krokem je zaplacení objednávky.

Chytré tržiště

Chytré tržiště je další klíčovou funkcí této aplikace. Každý skleník má tržiště vlastní. Tržiště je nutné nejprve naplnit produkty. Jelikož jeden produkt může být nabízen napříč více skleníků, tak administrátor může produkty vytvořit předem a tak ušetřit práci provozovatelům a správcům skleníků. Při vytváření sdíleného produktu je nutno vyplnit jeho název, popis a přiložit fotku. Jakmile je sdílený produkt vytvořen, tak jednotlivé skleníky mohou tento produkt přidat do svého tržiště. Takto se vyhnou nutnosti vyplnit informace o produktu a pouze nastaví cenu za kus a kvantitu zboží na skladě. Tento způsob také sníží duplicitní produkty na tržišti a sloučí všechny nabídky do jednoho produktu. Zákazník si tak může na základě ceny vybrat nejlevnější nabídku. Nezáleží však pouze na ceně. Pro zákazníka je klíčové vybrat takovou nabídku, která je nejblíže k jeho bydlišti. Proto vznikla možnost si zvolit preferovaný skleník. Svou preferenci může zvolit jak v košíku, tak i na stránce tržiště.

V košíku může upravovat kvantitu zboží a jednotlivé produkty z košíku odebrat. U každého produktu vidí cenu za kus a celkovou cenu za danou kvantitu. Pokud je zákazník spokojený s obsahem košíku, tak může pokračovat k dalšímu kroku. Tento krok jsem pojmenoval „Možnosti vyzvednutí“. Zde si zákazník může vybrat ze dvou možností. První možností je „Ideální vyzvednutí“. Systém se nejprve pokusí vybrat takové nabídky, které pochází z preferovaného skleníku. Pokud takové nabídky neexistují pro každý produkt, tak se zvolí nabídky od skleníku, který je nejblíže tomu preferovanému. Tento typ vyzvednutí urychlí zákazníkovi výběr, protože nemusí ručně hledat, který skleník je nejblíže jeho preferovanému. Druhou možností je „Nejlevnější vyzvednutí“. Systém nalezne nejlevnější nabídky pro každý produkt. Zákazník sice ušetří peníze, ale nemusí ušetřit čas, protože každá z nabídek může patřit k jinému skleníku, který se například nachází na jiné straně města. V tomto kroce uživatel potvrdí svou objednávku. Poté je mu zobrazena platba, kterou jsem již popsal v předchozí podsekci 4.2.

Zákazník si nemusí pamatovat, kde každý z produktů má vyzvednout, proto posledním krokem je zobrazení detailu objednávky. Zde vidí také seznam produktů, které si objednal. Produkty jsou seřazeny podle skleníku, ze kterého pocházejí. Včetně názvu skleníku vidí i jeho adresu a taky mapu s jeho polohou. Zákazník tak může snadno zjistit, kde má své zboží vyzvednout.

Výkazy práce

Provozovatel skleníku nemusí mít vždy čas pro správu jeho skleníku. V tomto případě může najmout správce, který se o úkony spojené se skleníkem může postarat. Aby nemusel provozovatel využívat jiného systému pro výkazy práce a měl vše na jednom místě bylo nutné tuto funkci vytvořit. Správce může vytvářet výkazy práce, ve kterých specifikuje ve kterém skleníku pracoval, jaké práce vykonal, časové období kdy pracoval a jakou výši

odměny požaduje. Po vyplnění výkazu ho správce odešle provozovateli, který může výkaz schválit nebo zamítnout. V případě schválení je výkaz práce uzamknut a považuje se za splněn. V případě zamítnutí je výkaz vrácen správci s poznámkou, co je potřeba opravit. Správce má taky na výběr ze dvou možností, buď výkaz zruší nebo ho upraví a znovu odešle. Celkový postup je graficky znázorněn na obrázku 4.4. Mezi odmítnutím a znovu odesláním výkazu může uplynout určitá doba a provozovatel si nemusí přesně pamatovat, proč byl výkaz zamítnut. Zde jsem se nechal inspirovat systémem Jira[8]. Ten slouží k správě projektů a úkolů formou tiketů. U každého tiketu je možné zobrazit historii. Této funkcionality jsem využil i zde. U každého výkazu lze zobrazit historii, kde je zobrazeno kdo a kdy výkaz vytvořil a jaké údaje vyplnil. V případě úpravy údajů může provozovatel jednoduše porovnat nové údaje s původními. Taktéž zobrazuje kdy byl výkaz schválen nebo zamítnut a z jakého důvodu. Tuto historii vidí jak správce, tak i provozovatel. Historie pomůže předejít nedorozuměním a zjednoduší komunikaci mezi správcem a provozovatelem.

Obrázek 4.4: Postup schválení pracovního výkazu.

Komunitní zeď

Jak jsem již zmiňoval v sekci 3.4, zde budou zobrazeny příspěvky uživatelů. Zvolil jsem Instagram[9] jako sociální médium, protože jeho hlavním zaměřením jsou příspěvky, které obsahují buď fotografie nebo video. Uživatelům stačí vyplnit odkaz na jejich příspěvek. Poté se jim zobrazí seznam svých příspěvků. U nich vidí jejich aktuální stav. Příspěvek buď čeká na schválení nebo je schválen. Uživatel má také možnost příspěvek smazat. Administrátor navíc vidí seznam všech příspěvků. Může je schvalovat, upravovat a mazat. Každý příspěvek je možné rozkliknout a zobrazit jej přímo v aplikaci, bez nutnosti návštěvy Instagramu.

[8] https://www.atlassian.com/software/jira
[9] https://www.instagram.com/

Newsletter

Newsletter je další funkcí, která byla zmíněna v kapitole Gamifikace 3.4. Uživatelé se mohou přihlásit k odběru newsletteru. Administrátor může vytvářet nové newslettery a odesílat je všem uživatelům, kteří se k odběru přihlásili. Tvorbu newsletteru musí zvládnout i administrátor, který není programátor. Proto musí být využit grafický editor, který umožní vytvořit newsletter bez nutnosti znalosti HTML. Newsletter může obsahovat text, obrázky, odkazy a tlačítka. Aby administrátor nemusel obrázky nahrávat do jiné aplikace a později vkládal do Newsletteru pouze přímý odkaz na obrázek tak je nutné tuto funkcionalitu také implementovat. Administrátor nahraje své obrázky buď zvlášť do galerie nebo přímo do newsletteru. Tak nebo tak se obrázek přidá do galerie a administrátor ho může použít i v budoucnu. Před odesláním newsletteru musí být zadán předmět. Poté dojde k odeslání newsletteru všem uživatelům, kteří ho odebírají. Konec každého newsletteru musí obsahovat možnost odhlášení odběru formou odkazu. Odběratel se tak nemusí přihlašovat do aplikace jen proto, aby se odhlásil z odběru newsletteru.

4.3 Schéma databáze

Nyní známe vše potřebné pro návrh databáze. Databáze obsahuje velké množství tabulek, které postupně popíšu. Schéma databáze naleznete v příloze B.

User

Tato tabulka popisuje uživatele aplikace. Uživatelem je jak zákazník, správce, provozovatel ale i administrátor. Každý uživatel má své id, uživatelské jméno, email, křestní jméno, příjmení a heslo. Je zde uložena i volba preferovaného skleníku a zda uživatel odebírá newsletter. Sloupec *activated* určuje zda je uživatel aktivní. Po registraci je uživatel neaktivní a musí potvrdit svůj e-mail. Jelikož potvrzovací emaily nemají dobu vypršení, tak je nutné detekovat, zda se jedná o první aktivaci. Pokud byl uživatel z nějakého důvodu deaktivován administrátorem tak by mohl svůj účet opět aktivovat pomocí potvrzovacího e-mailu. Tomu zamezí sloupec *activated_once*, který má výchozí stav `false`. Při aktivaci pomocí e-mailu se přepne na `true` a tak lze rozpoznat zablokovaného uživatele. Zda je uživatel administrátorem určuje sloupec *admin*. Ostatní role zde nejsou uvedeny, jelikož každý provozovatel nebo správce nemá přístup ke každému skleníku. Tato informace je uložena v tabulce *Greenhouse*.

Greenhouse

Údaje o skleníku jsou uloženy v tabulce *Greenhouse*. Každý skleník má své id, název, popis. Vytvoření skleníku, tvorba jeho záhonků a přidání produktů do tržiště se může protáhnout, proto sloupec *published* slouží k nastavení viditelnosti skleníku pro zákazníky. K rychlému rozeznání skleníků napomáhá další sloupec *image*, ten obsahuje odkaz na obrázek skleníku. Každý skleník může spravovat vždý jeden provozovatel a jeden správce. Sloupce *owner_id* a *caretaker_id* ukládají identifikátory těchto rolí. Posledním atributem je *greenhouse_address_id*, který odkazuje na tabulku *Greenhouse address*.

Greenhouse address

Předchozí tabulka již obsahuje velký počet atributů, proto jsem se rozhodl zvlášť definovat adresu skleníku. U adresy lze nastavit kód země, kraj, město, část města, ulici s číslem

popisním a PSČ. Obsahuje taktéž sloupce pro zeměpisnou šířku a délku. Tyto sloupce jsou nutné pro zobrazení skleníku na mapě.

Greenhouse business hour a Greenhouse business hour period

Každý skleník má své otevírací doby. Tyto údaje jsou uloženy v tabulce *Greenhouse business hour*. Sloupec *day* určuje den v týdnu. Jelikož jeden den může mít více otevíracích dob, tak v tabulce *Greenhouse business hour period* nalezneme sloupce *open* a *close*, které určují čas otevření a zavření skleníku.

Flowerbed

Tabulka *Flowerbed* obsahuje informace o záhonku. Každý záhonek má své id, název, šířku a délku. K určení ceny pronájmu záhonku slouží sloupec *price_per_day*, ten definuje cenu pronájmu záhonku na jeden den. Záhonek lze deaktivovat a aktivovat pomocí sloupce *disabled*. Sloupce *tools* a *idealPlants* definují jaké nářadí je k dispozici a jaké rostliny jsou ideální pro tento záhonek. Poslední sloupec *greenhouse_id* odkazuje na skleník, ve kterém se záhonek nachází.

User flowerbed

Aby uživatel nepřišel o své údaje ohledně záhonku, při prodloužení pronájmu záhonku nebo jeho opětovném pronájmu tak tato tabulka propojí uživatele a záhonek. Její *id* využijí tabulky *Flowerbed harvest* a *Flowerbed note* pomocí sloupce *user_flowerbed_id*.

Flowerbed harvest a Flowerbed notes

Sklizeň záhonku je uložena v tabulce *Flowerbed harvest*. Obsahuje sloupce název, kvantitu a datum sklizně. K zápisu poznámek o záhonku slouží tabulka *Flowerbed note*. Sloupec *note* ukládá poznámku a sloupec *date* datum vytvoření poznámky.

Rent

Tato tablka obsahuje sloupce, které určují pronájem záhonku. Sloupce *rented_from* a *rented_to* určují období pronájmu. Další sloupce *flowerbed_id* a *user_id* odkazují na záhonek a uživatele.

Product

Zde jsou uloženy informace o produktech. Každý produkt má své id, název popis a obrázek. Jednotlivé nabídky vždy odkazují na jeden produkt, takto dojde ke sloučení nabídek do jednoho produktu.

Marketplace product

Jedná se o nabídku produktu v tržišti, proto je spojena s tabulkou *Product* přes cizí klíč *product_id* a taky s tabulkou *Greenhouse* přes cizí klíč *greenhouse_id*. Sloupce *price* a *quantity* určují cenu za kus a množství zboží na skladě.

Discount

Slevy nalezneme v tabulce *Discount*. Mezi sloupce patří kód slevy a výše slevy. K rozeznání, že kód slevy byl použit slouží sloupec *used*.

Order, Flowerbed order a Product order

Všechny objednávky vzniknou v tabulce *Order*. Sloupec *status* určuje stav objednávky, stavy máme celkem 3:

- *created* - objednávka byla vytvořena
- *paid* - objednávka byla zaplacena
- *cancelled* - objednávka byla zrušena

Dále obsahuje sloupec *user_id*, který odkazuje na uživatele, který objednávku vytvořil a sloupec *created_at* určuje datum vytvoření objednávky. Cizí klíč *discount_id* odkazuje na tabulku *Discount*.

Systém má dva typy objednávek:

- *Flowerbed order* - objednávka záhonku
- *Product order* - objednávka produktu

Obě tyto tabulky odkazují na tabulku *Order* pomocí cizího klíče *order_id*. Objednávka záhonku se váže k pronájmu záhonku.

Product order item

Sortiment nabízených produktů na tržišti se může měnit a proto je nutné uchovat kompletní informace o objednaných produktech. Jednotlivé produkty jsou uloženy v tabulce *Product order item*. Obsahuje tyto sloupce:

- *id* - identifikátor tohoto záznamu
- *quantity* - množství objednaného produktu
- *price* - cena za kus
- *greenhouseName* - název skleníku ze kterého pochází produkt
- *greenhouseId* - odkazuje na skleník, ve kterém je produkt nabízen
- *productName* - název produktu
- *productImage* - odkaz na obrázek produktu
- *productId* - odkazuje na produkt, který již nemusí existovat
- *productOrder_id* - odkazuje na objednávku produktu

Timesheet

Výkazy práce jsou uloženy v tabulce *Timesheet*. Obsahuje sloupce jako název, popis, výši odměny, datum vytvoření, datum poslední úpravy a autora výkazu. Sloupec *status* určuje stav výkazu. Jednotlivé stavy si lze odvodit z popisu v podsekci 4.2, ale zde je upřesnění:

- *submitted* - výkaz byl vytvořen
- *approved* - výkaz byl schválen
- *rejected* - výkaz byl zamítnut
- *cancelled* - výkaz byl zrušen

Timesheet working hour

Výkaz práce může obsahovat více pracovních hodin. Tyto hodiny jsou uloženy v tabulce *Timesheet working hour*. Obsahuje sloupce *start* a *end*, které určují časové období, kdy byla práce vykonána. Sloupec *timesheet_id* je cizím klíčem odkazujícím na tabulku *Timesheet*.

Timesheet item

V této tabulce jsou uloženy jednotlivé práce, které byly vykonány. Sloupce *title* a *description* určují název a popis práce. Opět obsahuje cizí klíč *timesheet_id* odkazující na tabulku *Timesheet*.

Timesheet update

Jakákoliv změna výkazu práce je uložena v tabulce *Timesheet update*. Obsahuje podobné sloupce jako tabulka *Timesheet*, ale vyplnění jejího řádku závisí na typu změny. Při zamítnutí nebo schválení výkazu je vyplněn pouze autor změny, nový status a případně zpráva. Při úpravě výkazu jsou vyplněny taky sloupce platu a popisu. Nově vyplněné pracovní hodiny i práce odkazují na tuto tabulku.

Social post

V této tabulce nalezneme jednotlivé příspěvky na komunitní zdi. Obsahuje tyto sloupce:

- *id* - identifikátor příspěvku
- *url* - odkaz na příspěvek
- *approved* - schválení příspěvku
- *author_id* - odkazuje na uživatele, který příspěvek vytvořil
- *created_at* - datum vytvoření příspěvku
- *updated_at* - datum poslední úpravy příspěvku

Kapitola 5

Použité technologie

Tato kapitola pojednává o výběru technologií, které budou použity pro implementaci webové aplikace.

5.1 Docker

Je to nástroj, pomocí kterého lze izolovat aplikace a jejich závislosti do kontejnerů. Ty umožní aplikace spustit nezávisle na hostitelském prostředí. Docker se také stará o správu kontejnerů a celý jejich životní cyklus.[9]

Docker mi tedy umožní vytvořit kontejner pro každou službu, kterou budu potřebovat. Tím se zjednoduší vývojové prostředí i prostředí pro nasazení aplikace. Nemusím se starat o instalaci jednotlivých aplikací a všechny můžu ovládat pomocí *Docker CLI*.

Docker CLI

Umožňuje ovládat Docker z příkazové řádky. Při vývoji jsem využil tyto příkazy:

- `docker ps` - zobrazí běžící kontejnery
- `docker run` - spustí kontejner ze zvoleného image
- `docker build` - sestaví image
- `docker exec` - spustí příkaz uvnitř kontejneru

Docker image

Z příkazů v Dockerfile je vytvořen samostatný a komprimovaný software, který obsahuje vše potřebné pro vytvoření a spuštení kontejneru. Vzniklý software se označuje jako *Docker image*.[9]

Mezi příklady patří operační systémy, databáze, webové servery a další.

Dockerfile

V tomto souboru nejprve definujeme nadřazený image. Poté všechny příkazy a nastavení pro instalaci aplikace. Nakonec definujeme příkaz, kterým se aplikace spustí.

Docker Compose

Pokud máme více kontejnerů, které spolu navíc komunikují, tak správa každého kontejneru zvlášť může být složitá. Práci s kontejnery nám ulehčí *Docker Compose*. Pomocí souboru *docker-compose.yml* definujeme jednotlivé služby a propojení mezi nimi. Po spuštění příkazu `docker-compose up` vznikne z každé služby nový kontejner.

5.2 Webový server

Stará se o poskytování webový stránek webu. Aktuálně nejpopulárnější open source webové servery jsou Apache a Nginx [21].

Webový server nám umožní přístup ke všem službám z jediné domény a jednoho portu. Pro tuto aplikaci jsem zvolil Nginx, který obsahuje funkci *reverse proxy*, pomocí které můžeme směrovat požadavky na další služby.

5.3 Frontend

Část aplikace nebo webu, kterou uživatelé vidí a s níž interagují. Zahrnuje tlačítka, formuláře, rozvržení stránky a její design. Můžeme se setkat i s pojmenováním "klient". Vznikne kombinací programovacích jazyků HTML, CSS a JavaScriptu. [3]

Typescript

V roce 2012 firma Microsoft vytvořila tuto nadstavbu jazyka JavaScript. Přidává statické typování, třídy a další. [12] Upozorní nás na možné chyby v kódu jak při vývoji, tak i při kompilaci kódu do JavaScriptu. Pokud zvolíme správný textový editor, tak získáme nápovědu s vlastnostmi a metodami objektu.

Frontend framework

Slouží k tvorbe uživatelských rozhraní. Umožní vývojáři psát efektivní kód, který je snadno udržovatelný a rozšiřitelný. Možnost použít stejné komponenty napříč projekty pomáhá udržovat kód konzistentní. [20]

Mezi známé frontend frameworky patří:

- React
- Angular
- Vue

React

Knihovna pro tvorbu uživatelských rozhraní. Vyvinuta společností Facebook v roce 2013. Popularity dosáhla až koncem roku 2015. [15]

Sama o sobě neobsahuje velké množství stavebních bloků jako Angular. Typicky se používá ve spojení s dalšími komunitními moduly. Mezi ty patří například React Router[1], který se stará o routování aplikace.

[1] https://reactrouter.com/

Angular

Jedná se o framework, který umožňuje vytvářet uživatelská rozhraní. Z poskytnutých stavebních bloků lze rychle vytvořit udržovatelnou a rozšiřitelnou aplikaci.[1]

Vue

Vue je další frontend framework pro tvorbu uživatelských rozhraní. Evan You hledal alternativu k Angularu a Backbone.js, která by byla jednodušší a sloužila primárně k tvorbě prototypů. Alternativu nenašel a tak vytvořil Vue [8].

Výběr

Zvolil jsem React, hlavně kvůli jeho velké komunitě a množství modulů. Není tedy potřeba vytvářet každou komponentu od základu. Mé zkušenosti s touto knihovnou umožní urychlit proces vývoje.

UI knihovna

Jedná se o předem napsané komponenty, které usnadňují vývoj uživatelského rozhraní. Příkladem může být tlačítko, formuláře, modální okna a tabulka. Komponenty mají konzistentní design a taky API. Důvodem pro užití UI knihovny je urychlení vývoje a přístupnosti pro lidi s různými postiženími.

Mezi známé UI knihovny pro React patří:

MUI[2]

Tato UI knihovna využívá Material design[3] od Google. Nejedná se však o knihovnu od Googlu. Obsahuje velké množství komponent, jejich úprava je však složitější.

Chakra UI[4]

Modulární UI knihovna, která používá Emotion[5] jako CSS knihovnu. Méně komponent oproti MUI. Je možné jednoduše změnit design podle sebe.

NextUI[6]

Méně známá UI knihovna oproti předchozím, která používá TailwindCSS[7] jako CSS knihovnu. Má méně komponent oproti MUI. Úpravy designu jsou jednoduché.

Výběr

Vybral jsem `NextUI`, protože TailwindCSS mi osobně nejvíce vyhovuje, jelikož nemusím vytvářet objekty pro každou změnu designu, ale stačí nastavit komponentě třídu. Tato knihovna obsahuje většinu komponent, které budu potřebovat.

[2] https://mui.com/
[3] https://material.io/design
[4] https://chakra-ui.com/
[5] https://emotion.sh/
[6] https://nextui.org/
[7] https://tailwindcss.com/

5.4 Backend

Zatímco frontend se zabývá tím, co uživatelé vídí a s čím interagují, tak backend, často označovaný jako "serverová část", pohání funkčnost webové stránky na pozadí. Spravuje databázi, ukládá data, zpracovává požadavky a zajišťuje, aby všechno fungovalo hladce. [3]

Django

Django[8] je webový framework, který napsán v programovacím jazyce Python. Jedná se o kompletní řešení pro vytváření webových aplikací. Obsahuje mnoho nástrojů, které usnadní a urychlí vývoj. Pokud bychom použiti Flask[9], tak bychom museli hledat komunitní moduly pro určité nástroje. Django obsahuje nástroje pro komunikaci s databází, správu uživatelů, jejich práv, šablonovací systém a mnoho dalších. Díky jeho velké komunitě lze najít mnoho modulů, které framework rozšíří o nové funkce. Tato aplikace využije primárně jeho nástroje pro vytvoření databáze a správu uživatelů. Jelikož využíváme React, tak Django bude využit pouze jako REST API.

Django REST framework

Je rozšíření Djanga, které pomáhá vytvářet REST API. Výsledný kód je kratší a přehlednější. Mezi jeho funkce patří například autentizace uživatelů, serializace dat a routování příchozích požadavků.

REST

Tato podsekce je převzata z [11].

V dnešní době je nejpopulárnější volbou pro tvorbu API právě REST (representational state transfer). Využívají jej firmy jako Google, Stripe, Twitter a GitHub. REST API vystavuje data jako zdroje. Entita, která lze identifikovat, pojmenovat nebo adresovat na webu je zdrojem. HTTP metody provádí operace nad zdroji. Mezi tyto metody patří `GET`, `POST`, `PUT` a `DELETE`. Zavolání různých metod nad stejnou URL vykoná různé operace:

- Create - `POST` metoda vytvoří nový zdroj.
- Read - Díky metodě texttttGET získáme data z existujícího zdroje. Neprovádí žádné změny nad zdrojem.
- Update - Zde se využívají dvě metody `PUT` a `PATCH`. První nahradí zdroj novým, druhá změní pouze část zdroje.
- Delete - `DELETE` metoda zdroj odstraní.

Server vrací HTTP odpovědi, které obsahují statusový kód, ten označuje zda byl požadavek úspěšný nebo neúspěšný. Úspěch je označen kódem 2XX, přesun zdroje 3XX, při nesprávném požadavku ze strany klienta 4XX a chyba na straně serveru 5XX. Formátem odpovědi je nejčastěji JSON nebo XML, kde JSON je standardem pro moderní REST API.

Django REST framework umožňuje využít i formát XML, ale pro tohle využití bude použit JSON.

[8] https://www.djangoproject.com/
[9] https://flask.palletsprojects.com/

5.5 Databáze

K ukládaní dat využijeme relační databázi. Relační databáze ukládá data do tabulek, které jsou propojeny klíči. Django oficiálně podporuje několik databází:[10]

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

V úvahu přichází pouze databáze, které jsou zdarma. Proto Oracle odpadá. Každá ze zbylých databází by splňovala požadavky této aplikace. Zvolil jsem PostgreSQL, kvůli mé zkušenosti s touto databází.

5.6 Správa úloh

Aplikace musí být schopna periodicky provádět určité úkony jako kontrolu výpisu účtu. Django tuto funkci neobsahuje, proto je nutné využít nějaký existující nástroj pro plánování úkolů. Určité úkony jako odesílání emailů není ideální provádět přímo v Djangu, aby nedošlo ke zpomalení aplikace. Tyto úkony je vhodné uskutečnit na pozadí.

Celery

Celery[11] je nástroj pro správu úloh napsaný v jazyce Python. Umožňuje naplánovat úkony a také je spustit na pozadí.

Celery beat

Celery beat je modul pro Celery, který umožňuje plánovat úkony. Úkony lze plánovat jednorázově nebo periodicky. Díky modulu `django-celery-beat`, lze plánování provádět přímo v administraci Djanga.

Celery worker

Tento modul slouží k provádění úkolů. Nativní podpora Djanga umožňuje využít stejné API pro práci s databází jako v Backendu. Je spuštěn jako samostatný proces, takže nemá vliv na rychlost aplikace.

Redis

Jednotlivé Celery moduly musí komunikovat mezi sebou. Jejich prostředníkem bude Redis[12]. Umožní ukládat data v mezipaměti a také odesílat zprávy mezi jednotlivými moduly.

[10] https://docs.djangoproject.com/en/5.0/ref/databases/
[11] https://docs.celeryq.dev/en/stable/
[12] https://redis.io/

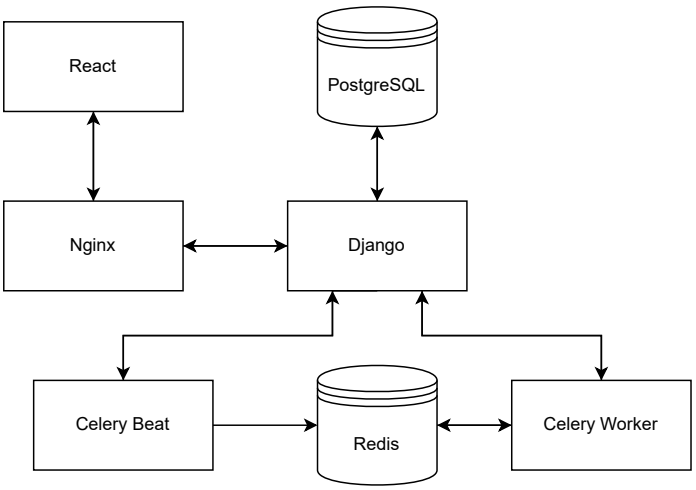
Kapitola 6

Implementace

Tato kapitola popisuje implementační detaily této aplikace. V první části se zaměřím na architekturu. Následně vysvětlím jak bylo vytvořeno vývojové prostředí a způsob jak jej spustit. Poté vysvětlím implementační detaily backendu a frontendu. Kapitola je zakončena způsobem detekce role uživatele, platebním systémem a implemetací newsletteru.

6.1 Architektura

Jednotlivé technologie byly popsány v kapitole 5. V této kapitole se zaměřím na jejich propojení. Diagram architektury nalezneme na obrázku 6.1. Každý požadavek nejprve prochází webovým serverem (Nginx), který podle URL adresy rozhodne, zda požadavek zpracuje Frontend (React) nebo Backend (Backend). Pokud se v URL za doménou prvního řádu nachází `/api`, `/admin` nebo `/media`, tak požadavek obslouží Backend, jinak ho obslouží Frontend. To znamená, že Frontend není přímo propojen s Backendem. Frontend získává veškerá data přes Backend. K databázi (PostgreSQL) má přístup pouze Backend. Plánovač úkolů (Celery Beat) přijímá úkoly z Backendu a příkazy k jejich provedení posílá přes Redis. Celery worker přijímá úkoly přes zprávy z Redisu a provádí je. Když potřebuje komunikovat s databází, tak se připojí přes Backend.



Obrázek 6.1: Grafické znázornění architektury.

6.2 Vývojové prostředí

Při vývoji byl využit Docker, ale ne v plném rozsahu. Některé služby byly spuštěny přímo na mém počítači kvůli častým změnám a snadnějšímu ladění. Ostatní služby byly spuštěny v kontejnerech. Jak jsem zmiňoval v sekci 5.1, tak pomocí Docker Compose spustím všechny služby najednou. Nastavení služeb pro vývoj nalezneme v souboru `docker-compose-dev.yml`.

docker-compose-dev.yml

```
services:
  reverse_proxy:
    build:
      context: ./nginx
      args:
        - ENV=dev
    volumes:
      - ./backend/media:/backend/media
      - ./backend/static:/backend/static
    ports:
      - 80:8080
  redis:
    image: redis:alpine
    ports:
      - 6379:6379
  db:
    image: postgres:16.0-alpine
    restart: always
    environment:
      - POSTGRES_USER=******
      - POSTGRES_PASSWORD=******
    ports:
      - 5432:5432
    volumes:
      - db:/var/lib/postgresql/data
volumes:
  db:
```

Výpis 6.1: Soubor `docker-compose-dev.yml`, který vytváří kontejnery vývojového prostředí.

Nejprve definuji jednotlivé služby pod klíčem `services`. Jednotlivé služby jsou `redis`, `reverse_proxy` a `db`. U každé služby nastavím image nebo cestu k Dockerfile. Pomocí nastavení `environment` mohu nastavit určité proměnné jako například uživatelské jméno a heslo pro databázi, tak lze jednoduše rozdělit konfiguraci pro vývoj a produkci. Nastavení `ports`, umožní připojení k portům kontejnerů z hostitelského počítače. Například u databáze mapuji port mého počítače 5432 na port 5432 kontejneru. Port 5432 je standardní port pro PostgreSQL. Tímto způsobem mohu přistupovat k databázi z grafických databázo-

vých klientů jako DBeaver[1] nebo DataGrip[2] a nemusím procházet databázi přes příkazovou řádku kontejneru.

[1] https://dbeaver.io/
[2] https://www.jetbrains.com/datagrip/

Spuštění vývojového prostředí

Tento postup je platný pro Linuxové distribuce a macOS. Pro spuštění vývojového prostředí je nutné mít nainstalovaný Docker a Docker Compose. Pro Frontend nainstalovaný Node.js, pnpm a balíčky z `package.json`. Backend vyžaduje Python a balíčky z `requirements.txt`. Spustit vývojové prostředí lze následovně:

1. Vytvoříme soubor `.env` ve složce `backend` a vyplníme ho podle souboru `.env.example`.
2. Pomocí příkazu `docker compose -f docker-compose-dev.yml up -d` spustíme kontejnery webového serveru a databází.
3. Ve složce `frontend` spustíme příkaz `pnpm dev -host`.
4. Aplikujeme migrace do databáze pomocí příkazu `python manage.py migrate` ve složce `backend`.
5. Nyní můžeme spustit backend příkazem `python manage.py runserver` ve stejné složce.
6. Celery beat spustíme příkazem `./celerybeat-start.sh`.
7. Celery worker spustíme příkazem `celery -A myproject worker -loglevel=INFO`.
8. Účet administrátora vytvoříme pomocí příkazu `python manage.py createsuperuser`.
9. Pokud vše proběhlo správně, tak na adrese `http://localhost` nalezneme spuštěnou aplikaci.

6.3 Backend

V této sekci je nejprve popsána adresářová struktura backendu. Následně je představen pojem „aplikace“ a její součásti.

Adresářová struktura

Každý Django backend obsahuje projekt a aplikace. Projekt obsahuje nastavení celého backendu a routování požadavků. Aplikace se stará o určitou funkcionalitu systému.

Mezi klíčové soubory kořenového adresáře patří:

- `.env` – Obsahuje nastavení proměnných např. pro připojení k emailovému serveru.
- `manage.py` – Nástroj pro správu projektu. Umožňuje spouštět server, generovat a aplikovat migrace, vytvářet administrátora a další.
- `media` – Zde se ukládají veškeré nahrané soubory uživatelů.
- `myproject` – Adresář projektu obsahující nastavení celé aplikace a routování požadavků.
- `requirements.txt` – Obsahuje všechny potřebné balíčky pro běh aplikace.

Aplikace

V tomto kontextu znamená "aplikace" samostatný modul, který se stará o určitou část systému. Mělo by tedy být možné modul přesunout do jiného projektu a tam bez velkých změn použit. Každá aplikace obsahuje svůj vlastní adresář, který může obsahovat modely, migrace a v případě této aplikace taky API endpointy a serializéry. Aplikace typicky obsahují tyto soubory:

- `models.py` – Zde jsou definovány modely, které se mapují na databázi.
- `serializers.py` – Serializéry, které převádí objekty na JSON.
- `views.py` – Obsahuje API endpointy.
- `urls.py` – Nastavení routování požadavků.
- `migrations/` – V této složce jsou uloženy migrace.

Aplikace `users`, se stará o uživatele. Obsahuje kompletní logiku pro vytváření, editaci a aktivaci uživatelů. Další aplikací je `orders`, která spravuje objednávky. Oddělení objednávek od jejich obsahu umožnilo snadnější integraci jak pro produkty tržiště, tak pro pronájem záhonků.

Modely

Model je definován jako třída, která obsahuje pole. Pomocí modelu definujeme databázovou tabulku a její sloupce. Jednotlivé moduly mohou mít mezi sebou vztahy. Vztahy jsou definovány pomocí cizích klíčů.

Jako příklad jsem zvolil část modelu `Flowerbed`, který reprezentuje záhonek:

```
class Flowerbed(models.Model):
    name = models.CharField(blank=True, null=True)
    greenhouse = models.ForeignKey(Greenhouse, models.DO_NOTHING)
    disabled = models.BooleanField(default=False, blank=True, null=True)
    pricePerDay = models.DecimalField(
        max_digits=10, decimal_places=5, default=50, blank=False, null=False
    )
    created_at = models.DateTimeField(default=timezone.now)
```

Výpis 6.2: Ukázka části modelu `Flowerbed`

Django obsahuje velké množství typů polí, které pokryjí většinu potřeb. Různá pole obsahují různá dodatečná nastavení. Například u pole `pricePerDay` lze nastavit maximálni počet desetinných míst a nejvyšší počet číslic. Nastavení `default` určuje výchozí hodnotu sloupce, která může být konstantní nebo výsledek funkce. Výchozí hodnoty nejsou nastaveny v databázi, ale Django je v případě nevyplnění nastaví sám. Takže při manuálním vytvoření nového řádku v databázi je třeba výchozí hodnotu nastavit ručně.

Migrace

Pokud vytvoříme nebo změníme model, tak musíme vytvořit migraci a poté ji aplikovat na databázi. Migrace jsou soubory, které obsahují seznam změn, které je potřeba provést, aby došlo k synchronizaci databáze s modelem. Django poskytuje nástroj `makemigrations`,

který jednotlivé migrace vytvoří. Nástrojem `migrate` aplikujeme migrace na databázi. První migrace vytvoří tabulky a jejich sloupce, další migrace obsahuje změny jako přidání, přejmenování nebo odstranění sloupců.

Serializéry

Jsou to třídy, které převádí objekty na JSON a naopak. Příchozí i odchozí data jsou serializována. Využívají se v API endpointech. Validují příchozí data, aby nedošlo k chybám při zpracování. Chyby jsou vráceny v odpovědi serveru, lze je využít pro zobrazení chybové hlášky uživateli. Serializéry je možné vytvářet na základě modelu a tak lze předejít duplikaci kódu. Mohou obsahovat vlastní logiku pro správu dat.

Pohledy

Jsou třídy nebo funkce, které přijímají požadavky a vrací odpovědi. Jeden pohled může mít jeden nebo více API endpointů. V těle pohledu můžeme zpracovat příchozí data, pracovat s databází a vrátit odpověď. Díky knihovně Django REST framework je možné vytvářet pohledy na základě modelu, což usnadňuje vytváření API.

Příklad pohledu pro výpis seznamu skleníků:

```
class GreenhouseViewSet(viewsets.ModelViewSet):
    queryset = Greenhouse.objects.all()
    serializer_class = GreenhouseSerializer
    permission_classes = [IsAuthenticated]

    def list(self, request, *args, **kwargs):
        if request.user.is_superuser or request.user.is_staff:
            queryset = Greenhouse.objects.all()
        else:
            queryset = Greenhouse.objects.filter(
                Q(owner=request.user.profile)
                | Q(caretaker=request.user.profile)
                | Q(published=True)
            )
        serializer = GreenhouseSerializer(queryset, many=True)
        return Response(serializer.data)
```

Výpis 6.3: Příklad pohledu pro výpis seznamu skleníků

Tento pohled je dostupný pouze pro přihlášené uživatele. Na základě role získá uživatel seznam skleníku. Pokud je uživatel administrátor, tak má přístup ke všem skleníkům. Pokud je uživatel běžný uživatel, tak má přístup pouze ke skleníkům, které vlastní, spravuje nebo jsou zveřejněné.

6.4 Frontend

Začátek této sekce popisuje adresářovou strukturu frontendu. Poté jakým způsobem je zajištěna komunikace s backendem. Nakonec vysvětluji jak funguje routování stránek.

Adresářová struktura

Kořenový adresář aplikace obsahuje několik složek a souborů. Klíčové z nich se nachází v následujícím seznamu.

- `package.json`
- `node_modules/`
- `src/`
  - `components/`
  - `features/`
  - `utils/`
  - `App.tsx`
  - `main.tsx`

V souboru `package.json` jsou definovány všechny balíčky a jejich verze potřebné pro běh aplikace. Po instalaci aplikace pomocí `pnpm install` se vytvoří složka `node_modules`, která obsahuje zdrojový kód těchto balíčků. Zdrojový kód aplikace nalezneme ve složce `src`, její obsah je popsán níže.

Components

Složka obsahuje komponenty, které jsou použity v různých částech aplikace. Takto stačí kód změnit na jednom místě. Příkladem je Loading.tsx, který obsahuje načítací ikonu.

Features

Každá část aplikace má vlastní složku. Například `auth` obsahuje všechen kód, který se týká autentizace.

Obsahuje také další podsložky jako:

- `components` – Zde nalezneme komponenty, které jsou použity pouze v této části aplikace. Například modální okno pro vytvoření nového skleníku.
- `hooks` – Obsahuje logiku komponent a volání API .
- `routes` – Zde jsou definovány stránky této části aplikace.

Utils

Obsahuje pomocné funkce, které jsou použity v různých částech aplikace. Příkladem jsou funkce pro formátování data a času.

App.tsx

V tomto souboru je definována hlavní struktura aplikace. Zde se nachází routování a hlavní layout aplikace.

main.tsx

Je vstupním souborem aplikace. Probíhá zde inicializace knihovny React a její vykreslení do prohlížeče.

Komunikace s API

Pro komunikaci s API jsem využil knihovnu Axios[3]. Axios je knihovna pro vytváření HTTP požadavků. V souboru `api.ts` je definována instance této knihovny. V instanci jsou nastaveny hlavičky, které API požaduje. Všechny požadavky jsou odeslány přes tuto instanci. Knihovna TanStack Query[4] se stará o správu stavu aplikace. Ukládá data z API do mezipaměti a zajišťuje jejich aktuálnost. Její výhody lze představit na příkladu, kdy uživatel chce upravit svůj profil. V modálním okně se zobrazí formulář, ve kterém může upravit své údaje. Po úspěšném odeslání dat z formuláře je potřeba data aktualizovat napříč aplikací. Tuto funkcionalitu zajišťuje TanStack Query.

Generování typů pro TypeScript

Není třeba psát vlastní typy pro každý endpoint backendu. DRF nám umožní vygenerovat OpenAPI[5] schému. Následně stačí použít nástroj `openapi-typescript`[6], který vygeneruje typescriptové typy pro většinu endpointů. Tímto způsobem lze předejít chybám při práci s daty, protože známe strukturu dat. Generování nových typů jsem využíval při každé změně v API, proto spustitelný soubor `generate-types.sh` generuje jak nové OpenAPI schéma, tak i nové typy, abych nemusel spouštět příkazy zvlášť.

Routování

Jednotlivé stránky aplikace jsou definovány v souboru `App.tsx`. O zobrazení a přesměrování se stará React Router[7]. Aplikace obsahuje velké množství stránek. Jsou rozděleny do tří skupin, které popíši níže.

Veřejné stránky

Jsou dostupné pro všechny uživatele. Mezi ně patří hlavní stránka.

Stránky pro nepřihlášené uživatele

Každá tato stránka je obalena komponentou `RedirectRoute`. Pokud je uživatel přihlášen, tak je automaticky přesměrován do aplikace. Tento způsob ušetří uživateli čas a nemusí se přihlašovat znovu. Patří sem stránky jako přihlášení, registrace a obnovení hesla.

[3] https://axios-http.com/
[4] https://tanstack.com/query/
[5] https://swagger.io/specification/
[6] https://github.com/drwpow/openapi-typescript
[7] https://reactrouter.com/

Stránky pro přihlášené uživatele

Tyto stránky může zobrazit pouze přihlášený uživatel. O detekci přihlášení se stará komponenta `ProtectedRoute`. Mezi tyto stránky většina stránek aplikace jako zobrazení skleníků, objednávek a tržiště.

Detekce přihlášení

Na komponentě `ProtectedRoute` vysvětlím, jak probíhá detekce přihlášení uživatele.

```
export const ProtectedRoute = ({ children }: { children?: ReactNode }) => {
    const navigate = useNavigate()
    const query = useProfile()
    useEffect(() => {
        if (!query.isFetching && query.error) {
            toast.error("You must be signed in to access this page")
            navigate("/signin", { replace: true })
        }
    }, [navigate, query.error, query.isFetching])
    return (
        <><Layout>
                {query.isLoading ? (
                    <Loading />
                ) : children ? (children) : (<Outlet />)}
        </Layout></>
    )
}
```

Výpis 6.4: Komponenta `ProtectedRoute`.

Nejprve je vyslán požadavek na backend pomocí funkce `useProfile`, která určí, zda je uživatel přihlášen. K tomu je využit endpoint profilu zákazníka. Dokud backend nevrátí odpověď, tak uživatel vidí načítací ikonu. Na základě odpovědi serveru zobrazí další stránku. Pokud je uživatel přihlášen, tak je zobrazena požadovaná stránka. V opačném případě je uživatel informován, že se musí přihlásit a funkce `navigate` jej přesměruje na stránku příhlášení.

Komponenta `RedirectRoute` funguje přesně naopak. Pokud je uživatel přihlášen, tak je přesměrován do aplikace jinak je zobrazena požadovaná stránka.

6.5 Detekce rolí uživatele

Tato sekce popisuje způsob ověření rolí uživatele. Klíčovou roli hraje převážně backend, protože je zdrojem dat. Pokud by ověření rolí probíhalo pouze tam, tak frontend by mohl sice načíst rozložení stránky, ale nemohl by načítat data nebo provádět akce, ke kterým nemá oprávnění. Uživatel by byl informován že došlo k chybě. To není ideální, proto detekce probíhá i na straně frontendu. Uživatel vidí pouze části aplikace, ke kterým má oprávnění.

Detekce rolí na backendu

Django poskytuje nástroje pro kontrolu rolí uživatelů. Každý příchozí HTTP požadavek obsahuje objekt uživatele. Pokud je objekt definován, tak je uživatel přihlášen. Vlastnost

`is_superuser` vrací `True` pokud je uživatel administrátor, jinak vrací `False`. Pro kontrolu, zda je uživatel provozovatelem nebo správcem skleníku je třeba zkontrolovat pole `owner` a `caretaker` v modelu `Greenhouse`. Pokud je uživatel v jednom z těchto polí, tak má přístup ke skleníku. Příklad detekce rolí naleznete v této ukázce kódu 6.3. V případě, že uživatel přistoupí k API endpointu, ke kterému nemá oprávnění, tak je vrácena statusová chyba 403 se zprávou, že nemá dostatečná oprávnění.

Detekce rolí na frontendu

Na frontendu je detekce rolí uživatele složitější. Uživatel může mít různé role v závislosti na specifický skleník. Proto endpoint `/api/auth/profile` vrací informace o uživateli. Kromě osobních údajů obsahuje taky informace o rolích uživatele. Ukázku odpovědi serveru naleznete v tomto výpisu 6.5.

```
{
    "first_name": "Petr",
    "last_name": "Novak",
    "username": "pnovak",
    "email": "pnovak@email.cz",
    "owned_greenhouses": [],
    "caretaker_greenhouses": [
        {
            "id": 1,
        }
    ],
    "superuser": true
}
```

Výpis 6.5: Ukázka odpovědi endpointu `/api/auth/profile`.

Prázdné pole `owned_greenhouses` značí, že uživatel není vlastníkem žádného skleníku. Pole `caretaker_greenhouses` obsahuje seznam skleníků, které uživatel spravuje. Pokud je uživatel administrátor, tak je vlastnost `superuser` nastavena na `true`. Na základě těchto informací je možné zobrazit nebo skrýt určité části aplikace.

6.6 Platební systém

Tato sekce nejprve popíše způsob, jakým probíhá získání výpisu účtu a následně se zaměří na periodické stažení výpisu účtu. V závěru sekce je popsáno, jak jsou vytvořeny platební údaje a generování QR kódu.

Získání výpisu účtu

Jelikož využíváme bankovního převodu, tak je nutné získat výpis účtu. K tomu využijeme API bankovnictví od Fio banky[8]. V Internetovém bankovnictví vygenerujeme token, který nám umožní využívat API.

Manuál API Fio banky [5] zmiňuje několik možností, jak získat výpis účtu. Buď můžeme získat výpis za určité období, výpis určité transakce podle roku a jejího ID nebo výpis pohybů na účtu od posledního stažení.

[8] https://www.fio.cz/bankovni-sluzby/api-bankovnictvi

Druhá možnost je nejméně vhodná, protože by bylo nutné volat API pro každou transakci zvlášť. Třetí možnost jsem nejprve považoval za nejvhodnější, ale po bližším zkoumání jsem se rozhodl tuto možnost zavrhnout. Manuál popisuje [5], že tento endpoint funguje na principu zarážky, která se posouvá s každým stažením. Zarážku lze posunout na určité datum nebo na určitou transakci. Mým důvodem zavrhnutí této možnosti je, že by mohlo dojít k situaci, kdy server požádá o výpis, zarážka se posune a následně dojde k výpadku serveru nebo jiné chybě. Po obnovení serveru by byla zarážka již posunuta a nedošlo by k zpracování všech transakcí. Proto jsem se rozhodl pro první možnost, kdy získám výpis za určité období. Období jsem zvolil tak, že začíná vždy předchozím dnem a končí aktuálním dnem. Takto předejdu situaci, kdy je přijata transakce v čase 23:59:59 a výpis je stažen v 00:00:00 následujícího dne. Aplikace tak bude pracovat s větším množstvím dat, než u možnosti se zarážkou, ale zaručí, že všechny transakce budou zpracovány.

Příklad volání API dne 4. května 2024:

```
response = requests.get(
    f"https://www.fio.cz/ib_api/rest/periods/{BANK_TOKEN}
    /2024-05-03/2024-05-04/transactions.json"
).json()
```

Proměnnou `BANK_TOKEN` nahradíme tokenem, který jsme získali v Internetovém bankovnictví. API nám umožňuje zvolit formát dat ve kterém chceme výpis. Z nabízených možností jsem zvolil JSON, kvůli snadné manipulaci s daty.

Periodické stažení výpisu účtu

Zákazníci jsou zvyklí na rychlé zpracování plateb u plateb kartou. Abychom se co nejvíce přiblížili rychlosti plateb kartou, je nutné získat výpis účtu co nejčastěji. Na obrázku 4.3 lze vidět, že systém čeká na platbu od zákazníka. Pokud zákazníkova banka podporuje okamžité platby, tak by bylo ideální na stejné stránce zobrazit, že platba byla úspěšně přijata. Periodu stažení výpisů jsem zvolil na 10 vteřin. Toto číslo je dostatečně malé, aby zákazník nečekal dlouho na potvrzení platby, ale zároveň dostatečně velké, aby nedošlo k přetížení serveru a případné kolizi, pokud by systém nestihl zpracovat všechny transakce.

Pro uskutečnění periodického stažení jsem využil Celery. Nejprve jsem vytvořil úlohu, která získá veškeré nezaplacené objednávky. Následně dojde k získání výpisu účtu a porovnání transakcí s objednávkami. Pokud se shoduje částka i variabilní symbol, tak je objednávka označena jako zaplacená. V administraci je nutné nastavit periodu, ve které se má úloha spouštět. Perioda lze měnit i za běhu aplikace.

Endpoint `/api/orders/{id}/` je volán každých 5 vteřin ze strany frontendu, aby získal stav objednávky. Po změně stavu objednávky na zaplacenou je zobrazena informace o úspěšné platbě.

Vytvoření platebních údajů a QR platby

O platební údaje objednávky se stará endpoint `/orders/{id}/get_payment`. Vrací variabilní symbol, částku a číslo účtu pro platbu. Tyto údaje jsou zobrazeny zákazníkovi. Variabilní symbol se shoduje s identifikátorem objednávky v databázi.

Česká bankovní asociace[9] vytvořila standard pro vytváření QR plateb. Po přečtení tohoto standardu [4] jsem zvolil následující strukturu QR kódu:

[9] https://www.cbaonline.cz/

- `SPD` – Označuje, že se jedná o příkaz k úhradě.
- `ACC` – Číslo účtu ve formátu IBAN.
- `AM` – Výše platby.
- `CC` – Měna, ve které je částka uvedena.
- `PT` – Typ platby, v tomto případě je to okamžitá platba a je použitá hodnota `IP`.
- `X-VS` – Variabilní symbol.

Příkladem QR platby je následující řetězec:

```
SPD:1.0*ACC:CZ6508000000192000145399*AM:100*CC:CZK*PT:IP*X-VS:1337*
```

Z tohoto řetězce je pomocí knihovny `react-qr-code`[10] vytvořen QR kód, který je zobrazen zákazníkovi. Tento QR kód zákazník naskenuje ve své bankovní aplikaci a provede platbu.

6.7 Newsletter

Nejprve popíši na základě čeho jsem zvolil editor pro tvorbu newsletteru. Následně rozeberu jakým způsobem funguje nahrávání obrázků do editoru. Poté vysvětlím, jak probíhá odeslání newsletteru a funkce odhlášení newsletteru pomocí odkazu. V závěru je představena funkcionalita historie odeslaných newsletterů.

Volba editoru

Nutnost grafického editoru pro tvorbu newsletteru zkomplikovala implementaci. Nejprve jsem zvažoval vytvoření vlastního editoru, ale nakonec jsem se rozhodl využít některé z existujících řešení. Předpokládal jsem, že najít vhodný editor bude snadné, ale opak byl pravdou. Nejprve jsem vyzkoušel knihovnu `react-email-editor`[11]. Podle počtu hvězd na GitHubu jsem očekával, že bude jasnou volbou. Knihovnu jsem vyzkoušel a byla by vhodným kandidátem pro tvorbu newsletteru. Později jsem zjistil, že editor není součástí knihovny. Editor byl načten z externího serveru. Toto řešení jsem zavrhl, protože by mohlo dojít k ukončení provozu serveru a funkce newsletteru by byla nefunkční. Konečnou volbou byl editor `email-builder-js`[12]. Převzal jsem kód z balíčku `editor-sample`[13], který obsahuje kompletní editor.

Rozšíření editoru o možnost nahrání obrázků

Zvolený editor neumožňuje nahrávání obrázků, ale pouze vkládání obrázků pomocí URL adresy. Jelikož jsem měl přístup k kompletnímu zdrojovému kódu, tak jsem mohl editor upravit. Nejprve jsem vytvořil nové endpointy na backendu, které umožňují nahrání obrázku a získání seznamu nahraných obrázků. Následně jsem do editoru přidal novou sekci galerie, ve které lze zobrazit všechny nahrané obrázky a taky nahrát nový obrázek. Poté jsem upravil blok obrázku, aby uživatel nemusel kopírovat odkazy na obrázky. Administrátor může obrázek nahrát přímo v editoru nebo zvolit z galerie obrázků.

[10] https://www.npmjs.com/package/react-qr-code
[11] https://github.com/unlayer/react-email-editor
[12] https://github.com/usewaypoint/email-builder-js
[13] https://github.com/usewaypoint/email-builder-js/tree/main/packages/editor-sample

Odeslání newsletteru

Odeslání probíhá v nové sekci editoru. Uživatel zvolí předmět a níže vidí náhled newsletteru. Po odeslání newsletteru dojde k získání HTML výstupu editoru. Tento HTML kód je spolu s předmětem odeslán na backend. Požadavek zpracuje pohled

```
class SendNewsletterView(APIView):
    def post(self, request):
        # Admin only
        if not request.user.is_staff and not request.user.is_superuser:
            return Response({"message": "Unauthorized"}, status=403)

        # Get the email from the request
        title = request.data.get('title')
        html = request.data.get('html')

        # Call celery task
        send_newsletter.apply_async((title,html,))

        return Response({"message": "Email sent successfully"}, status=200)
```

Výpis 6.6: Pohled pro odeslání newsletteru

Jedná se o pohled, který zpracuje pouze POST požadavek a neváže se na žádný model a proto jsem zvolil třídu `APIView`. Nejprve je zkontrolováno, zda je uživatel administrátorem. Poté jsou získány data z požadavku. Pokud by byl newsletter odeslán přímo v pohledu, tak by mohl zpomalit server a proto jsem vytvořil Celery úlohu `send_newsletter`. Tato úloha je spuštěna s parametry `title` a `html`. Úloha nejprve získá seznam všech uživatelů, kteří odebírají newsletter. Poté přidá na konec emailu odkaz na odhlášení newsletteru. Poté je email odeslán přes další Celery úlohu `send_single_email`. Tato úloha vyžaduje cílovou emailovou adresu, předmět a HTML obsah emailu. Je využita i v jiných částech aplikace, kde je potřeba odeslat email jako například při registraci nebo obnovení hesla.

Odhlášení newsletteru pomocí odkazu

Při rušení newsletteru u jiných služeb jsem vždy preferoval odkaz, který automaticky odhlásí uživatele z newsletteru. Často musí uživatel ručně vybírat z možností proč se odhlašuje a následně ještě odhlášení potvrdit. Pro jednoduchost použití stačí kliknout na odkaz a uživatel je automaticky odhlášen. Endpoint `/api/unsubscribe` musí zpracovat `GET` požadavek, jelikož prohlížeče používají tuto metodu pro načtení stránky. Odkaz vypadá následovně:

`https://domena-aplikace.cz/api/unsubscribe?email=email&token=token`

V případě metody `GET` lze parametry definovat přímo v URL adrese. Jsou vyžadovány dva parametry `email` a `token`.

Token slouží k ověření uživatele. Je generován následovně:

```
mixed = EMAIL_UNSUBSCRIBE_TOKEN + email
token = sha256(mixed.encode()).hexdigest()
```

`EMAIL_UNSUBSCRIBE_TOKEN` je proměnná, která obsahuje tajný řetězec nastavený v souboru `.env`. Poté dojde ke spojení tajného řetězce s emailem. Pomocí hashovací funkce `sha256`, je vytvořen token. Ověření tokenu probíhá tak, že je získán email a token z URL

adresy. Následně je vytvořen token stejným způsobem jako při odeslání emailu. Pokud se oba tokeny shodují, tak je uživatel odhlášen z newsletteru.

Historie newsletteru

Je další sekcí editoru. Zde jsou uloženy všechny odeslané newslettery. Ze seznamu lze zvolit konkrétní newsletter a poté je načten v editoru. Tato funkcionalita je užitečná, pokud administrátor udělal chybu a chce newsletter opravit a poté odeslat znova. Dalším vhodným využitím je použití předchozího newsletteru jako šablony pro nový.

Při každém odeslání newsletteru je navíc odeslán výstup editoru ve formátu JSON. Výstup editoru obsahuje všechny bloky, pomocí kterých newsletter vznikl. Backend ho poté společně s předmětem a datem odeslání uloží do databáze.

Kapitola 7

Uživatelské testování

7.1 Testování na návrhu

První testování probíhalo v rámci návrhu aplikace. Jak jsem již zmínil v kapitole 4.1, tak testování probíhalo za použití interaktivní prezentace v nástroji Figma. Cílem testování bylo zjistit, zda je návrh srozumitelný pro uživatele.

Testovací skupina byla složena ze tří testerů, kteří se lišili věkem a zkušenostmi s prácí s počítačem. Testování probíhalo osobně, kdy jsem testerům představil návrh a následně dostali úkoly, které měli splnit. Testování probíhalo odděleně, aby se zamezilo ovlivnění výsledků.

Testování probíhalo následovně:

1. Zadání úkolu
2. Sledování, jak tester postupuje
3. Závěrečná diskuze

Úkolem testerů bylo:

- Nakoupit produkty na tržišti
- Pronajat si záhonek

Testování mělo ověřit, zda je postup dosažení cíle logický a zda je uživatel schopen úkol splnit. Výsledky testování byly pozitivní. Každý z testerů zvládl úkol splnit. Nicméně dva z nich byli zmateni z procesu vytvoření objednávky. Nebylo jim jasné, kolik kroků ještě zbývá k dokončení objednávky. Tento problém jsem vyřešil přidáním formuláře s více kroky jak je popsáno v sekci 4.2.

7.2 Testování při vývoji

Druhé testování probíhalo během vývoje aplikace. Po dokončení určitých části aplikace jsem požádal testery o otestování nových funkcí přímo v aplikaci. Testování probíhalo opět osobně. Testovací skupina byla složena z předchozích tří testerů a tří nových testerů.

Od předchozího testování se lišilo tím, že testeři pouze neklikali na interaktivní prezentaci, ale testovali reálnou aplikaci. Museli vyplnit jednotlivé formuláře a provést platbu. Netestovala se pouze role zákazníka, ale i role správce skleníku.

Testování role zákazníka zahrnovalo splnění následujících úkolů:

- Pronajmout si záhonek
- Koupit určitý produkt na tržišti
- Přihlásit se k newsletteru
- Přidat příspěvek na komunitní zeď

Při testování role správce skleníku bylo nutné splnit následující úkoly:

- Úprava skleníku
- Přidání nového produktu na tržiště
- Vytvoření nového záhonku
- Vytvoření výkazu práce

Důraz byl kladen především na roli zákazníka, jelikož je to nejčastější role. Proto jsem proces pronájmu záhonku a nákupu produktu měřil stopkami. Placení probíhalo přes poskytnutý mobilní telefon s bankovní aplikací.

7.2.1 Pronájem záhonku

Systém obsahoval několik skleníků s různým počtem záhonků. Některé záhonky byly již pronajaty, ostatní byly volné. Tester měl za úkol si pronajmout záhonek v určeném skleníku. Výsledky testování jsou v tabulce 7.1. Čas je uveden v minutách a sekundách. Od času byl odečtena doba strávená platbou. Testeři jsou rozděleni na dvě skupiny. V první jsou testeři z prvního testování a ve druhé testeři noví.

Tester	Čas	Počet dotazů	Průměrný čas skupiny
1	0:29	0	0:42
2	0:57	0	
3	0:39	0	
4 (Nový)	1:01	1	0:52
5 (Nový)	0:40	0	
6 (Nový)	0:56	0	

Tabulka 7.1: Výsledky testování pronájmu záhonku.

Výsledky ukazují, že průměrný čas, byl u první skupiny o 10 vteřin rychlejší. Důvodem může být, že tito testeři již měli zkušenosti s aplikací. Novým testerům trvalo déle se zorientovat v aplikaci. Většina testerů byla schopna úkol splnit bez dotazů. Tester 4 měl problém s vyplněním období pronájmu. Špatně vyplnil datum ukončení pronájmu.

7.2.2 Nákup produktu

Dalším úkolem bylo koupit 2 balení semen rajčat z určitého skleníku. Pro stížení úkolu jsem přidal větší množství produktů na tržiště. Sledoval jsem, zda využili funkci volby preferovaného skleníku. Výsledný čas opět neobsahuje dobu zaplacení. V tabulce 7.2 jsou výsledky testování.

Tester	Čas	Počet dotazů	Využita volba	Průměrný čas skupiny
1	0:21	0	Ano	0:32
2	0:45	0	Ano	
3	0:30	0	Ano	
4 (Nový)	0:49	0	Ano	0:37
5 (Nový)	0:28	0	Ne	
6 (Nový)	0:34	0	Ano	

Tabulka 7.2: Výsledky testování nákupu produktu.

Rozdíl mezi první a druhou skupinou je o 5 vteřin menší, než u pronájmu záhonku. Testeři se shodli, že všichni často nakupují na internetu a byli zvyklí na tento typ úkolu. Téměř všichni využili funkci volby preferovaného skleníku. Tester 5 preferoval zobrazit detail produktu a zvolit skleník tam. Pokud by ale objednával produktů více, tak by využil funkci volby preferovaného skleníku.

7.2.3 Zpětná vazba testerů

Většina testerů byla spokojena s aplikací jak z role zákazníka, tak z role správce skleníku. Poskytli několik návrhů na zlepšení, které jsem následně implementoval. Níže jsem uvedl několik návrhů, které byly nejčastěji zmiňovány.

U výkazu práce byl návrh na zlepšení zápisu práce. Původně bylo nutné ručně vyplnit každou práci ručně. Nyní lze vybrat z předem definovaných prací.

V nastavení skleníku bylo nutné nastavit nadmořskou šířku a délku. Tyto údaje nebylo možné získat z adresy skleníku bez poplatku. Tester navrhl využít mapu a díky ní vybrat polohu skleníku. K tomu jsem využil knihovnu react-leaflet[1].

U přidání příspěvku na komunitní zeď nebylo některým testerům jasné, odkud získat odkaz na jejich příspěvek. Instagram totiž přesunul tlačítko pro sdílení odkazu na jiné místo. Do aplikace jsem přidal návod jak odkaz získat.

Zadávání období pronájmu bylo pro některé testery matoucí. Pro usnadnění tohoto kroku jsem se inspiroval od služby Airbnb[2], kde je možné vybrat období pronájmu pomocí dvou kalendářů. Knihovna react-day-picker[3] umožnila zobrazení kalendáře.

Dalším návrhem bylo zobrazení volných záhonků u výpisu skleníků. Zákazník tak může ušetřit čas a navštívit pouze skleníky s volnými záhonky. Tato funkcionalita byla implementována a jako další vylepšení jsem barevně odlišil skleníky s volnými záhonky.

7.3 Testování celé aplikace

Poslední testování probíhalo po dokončení všech částí aplikace. Byly otestovány všechny role a všechny funkce. Testování probíhalo osobně a aplikace byla testována jak na počítači, tak na mobilním telefonu. Testovací skupina byla stejná jako u předchozího testování.

Testována byla nasazená aplikace na produkčním serveru. Testerům jsem vysvětlil pravomoce a vlastnosti zbylých rolí a taky funkcionalitu gamifikace. Nejprve proběhlo měřené testování nákupu produktu a pronájmu záhonku. Následně byly otestovány zbylé části aplikace.

[1] https://react-leaflet.js.org/
[2] https://www.airbnb.com/
[3] https://react-day-picker.js.org/

7.3.1 Pronájem záhonku a nákup produktu

Testování probíhalo podobně jako u předchozího testování. Testeři měli za úkol si pronajat záhonek a koupit produkt na tržišti. Výsledky obou testů jsou v tabulce 7.3.

Tester	Doba pronájmu záhonku	Doba nákupu produktu
1	0:22	0:23
2	0:50	0:44
3	0:35	0:28
4	0:58	0:41
5	0:44	0:25
6	0:57	0:36

Tabulka 7.3: Výsledky druhého testování pronájmu záhonku a nákupu produktu.

Porovnáním výsledků s předchozím testováním jsem dospěl k závěru, že výsledky se zlepšily u většiny testerů. Konzultace výsledků s testery ukázala, že implementace jejich návrhů vedla k lepšímu uživatelskému zážitku a rychlejšímu dokončení úkolů.

7.3.2 Testování zbylých částí aplikace

I zde testeři ocenili implementaci jejich návrhů. Testeři měli jediný úkol, kterým bylo projít všechny části aplikace a zjistit, zda vše funguje správně. Při testování byly objeveno pár chyb, které jsem následně opravil. Převážně se jednalo o chyby spojené s úpravou produktů. Při úpravě produktu nebyl správně aktualizován obrázek produktu. Nebyly objeveny žádné další závažné chyby a aplikaci testeři označili za funkční.

Prvky gamifikace ocenili především testeři ve věku 20-24 let. Ti označili tuto funkcionalitu za zábavnou a měli zájem o získání všech odznaků. Ostatní testeři nepovažovali gamifikaci jako klíčovou, ale byli rádi za její přítomnost, protože díky ní mohli získat slevu.

7.3.3 Zpětná vazba testerů

Celková zpětná vazba byla pozitivní. Testeři ocenili rychlost a jednoduchost aplikace. Opět byly zaznamenány návrhy na zlepšení.

U newsletteru byl návrh na zobrazení aktuálního počtu odběratelů. Implementovat tuto funkci nebyl problém a nyní v sekci odeslání newsletteru je zobrazen aktuální počet odběratelů.

Dalším návrhem bylo vybrat preferovaný skleník na základě aktuální polohy. Tuto funkcionalitu jsem se nejprve pokusil implementovat v prázdném projektu, ale nepodařilo se mi konzistentně získat polohu zařízení. Proto jsem se rozhodl tuto funkcionalitu nepřidat.

Testeři, kteří ocenili gamifikaci navrhli přidat tabulku s pořadím hráčů, aby mohli soutěžit s ostatními. Dalším návrhem bylo přidat více odznaků. Tyto návrhy jsem ocenil a jejich implementace je možná v budoucní verzi aplikace.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo vytvořit webovou aplikaci, která umožní lidem bez vlastních prostor pěstovat rostliny a zároveň jim poskytnout místo, kde mohou nakoupit různé produkty pro pěstování.

Nejprve proběhla analýza požadavků, která rozčlenila aplikaci na několik důležitých částí a rozdělila funkcionalitu mezi několik rolí. Z analýzy požadavků vznikl diagram případů užití. Následně proběhl výzkum gamifikace a její využití v aplikacích. Na základě výzkumu byly navrženy prvky gamifikace pro aplikaci. Analýza sloužila jako základ pro vytvoření návrhu. Návrh popisoval uživatelské rozhraní, jednotlivé části aplikace a návrh databáze. Výsledkem návrhu byl prototyp vytvořen v nástroji Figma a taky schéma databáze.

Poté byly zvoleny vhodné technologie pro implementaci aplikace. Jako front-end framework byl vybrán React a backendem byl Django. V implementaci byla popsána architektura aplikace a poté provedena implementace. Výsledkem implementace byla webová aplikace, která byla nasazena na produkční server.

Proběhly celkem tři kola testování. Nejprve byl testován návrh aplikace, poté byla testována aplikace během vývoje a nakonec proběhlo testování celého aplikace. Závěrem každého testování byla zpětná vazba, která byla převážně pozitivní a testeři zde navrhli několik vylepšení. Většina návrhů byla implementována a to vedlo k optimalizaci uživatelského rozhraní.

Završením práce byl plakát a krátké video. Plakát slouží k prezentaci výsledků práce a naleznete ho v příloze C. Video prezentuje aplikaci pro potenciální uživatele a naleznete ho na přiloženém médiu.

V této práci jsem využil své znalosti ze studia a praxe, ale také jsem musel řešit problémy, se kterými jsem se dosud nesetkal. Díky tomu jsem si rozšířil znalosti v oblasti vývoje webových aplikací, gamifikace a platebních systémů. Zadání považuji za splněné a mám zájem o další rozvoj aplikace. Myslím, že aplikace má potenciál a mohla by být v budoucnu využita v reálném prostředí.

Webová aplikace je dostupná na webové adrese `https://baka.aston.dev/`.

Literatura

[1] *Angular* [online]. [cit. 2024-1-15]. Dostupné z: https://developers.google.com/learn/topics/angular.

[2] *What is gamification?* [online]. [cit. 2024-4-22]. Dostupné z: https://www.biworldwide.com/what-is-gamification.

[3] *How do Frontend and Backend Technologies Work Together?* 2023 [cit. 2024-1-15]. Dostupné z: https://mdevelopers.com/blog/frontend-backend-tech-integration.

[4] ASOCIACE Česká bankovní. *Standardy bankovních aktivit - Formát pro sdílení platebních údaju v rámci tuzemského platebního styku v CZKprostrednictvím QR kódu* [online]. červen 2021. Dostupné z: https://cbaonline.cz/upload/1645-standard-qr-v1-2-cerven-2021.pdf.

[5] BANKA, F. *FIO API BANKOVNICTVÍ* [online]. červenec 2023. Dostupné z: https://www.fio.cz/docs/cz/API_Bankovnictvi.pdf.

[6] DOPRAVA Čistá. *Průměrné emise CO2 nových automobilů v roce 2023 v Česku nepatrně klesly, premiantem Citroën, Škoda sedmá* [online]. Čistá doprava, leden 2023 [cit. 2024-4-27]. Dostupné z: https://www.cistadoprava.cz/tiskove-zpravy/prumerne-emise-co2-novych-automobilu-v-roce-2023-v-cesku-nepatrne-klesly-premiantem-citroen-skoda-sedma/.

[7] FIALA, J. Gamifikace ve výuce. Leden 2019. ISSN 1802-4785. Dostupné z: https://spomocnik.rvp.cz/clanek/21961/GAMIFIKACE-VE-VYUCE.html.

[8] FILIPOVA, O. *Learning Vue.js 2.* 1. vyd. Packt, prosinec 2016. ISBN 978-1786469946.

[9] GRYGAŘÍKOVÁ, M. *Docker, Kubernetes a kontejnery. Jak fungují a proč je chtít* [online]. Srpen 2019 [cit. 2024-4-28]. Dostupné z: https://www.master.cz/blog/docker-kubernetes-kontejnery-jak-funguji-proc-je-chtit/.

[10] HARTINGER, D. *Lekce 2 - UML - Use Case Diagram* [online]. ITnetwork [cit. 2024-4-28]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram.

[11] JIN, B., SAHNI, S. a SHEVAT, A. *Designing Web APIs: Building APIs That Developers Love.* 1. vyd. O'Reilly Media, 2018. ISBN 978-1492026921.

[12] KVAPIL, J. *Lekce 1 - Úvod do TypeScriptu* [online]. [cit. 2024-1-15]. Dostupné z: https://www.itnetwork.cz/javascript/typescript/uvod-do-typescriptu.

[13] M., K. K. *The gamification of learning and instruction: game-based methods and strategies for training and education.* 1. vyd. Pfeiffer, 2012. ISBN 978-1118096345.

[14] MAV. *Podíl plateb pomocí online karty se v e-shopech zvyšuje* [online]. MediaGuru, červenec 2018 [cit. 2024-4-27]. Dostupné z: https://www.mediaguru.cz/clanky/2018/07/podil-plateb-pomoci-online-karty-se-v-e-shopech-zvysuje/.

[15] PALFREEMAN, M. *Why React? A Deeper Look into Facebook's Hit Framework* [online]. Srpen 2017 [cit. 2024-1-15]. Dostupné z: https://www.codefellows.org/blog/why-react-a-deeper-look-into-facebooks-hit-framework/.

[16] SENSORIE. *Sensorie – Skleník* [online]. Listopad 2023 [cit. 2024-1-14]. Dostupné z: https://www.instagram.com/sensorie.cz/p/CzWc_qoL3sw/?img_index=1.

[17] SOEGAARD, M. *How to Create Wireframes: An Expert's Guide* [online]. Interaction Design Foundation - IxDF, říjen 2023 [cit. 2024-4-27]. Dostupné z: https://www.interaction-design.org/literature/article/create-wireframes.

[18] SPANELLIS, A., DÖRFLER, V. a MACBRYDE, J. Gamification and innovation: a mutually beneficial union. In: *BAM 2016: 30th Annual Conference of the British Academy of Management* [online]. Září 2016, s. 1 [cit. 2024-4-22]. Dostupné z: https://www.researchgate.net/publication/306364960_Gamification_and_innovation_a_mutually_beneficial_union.

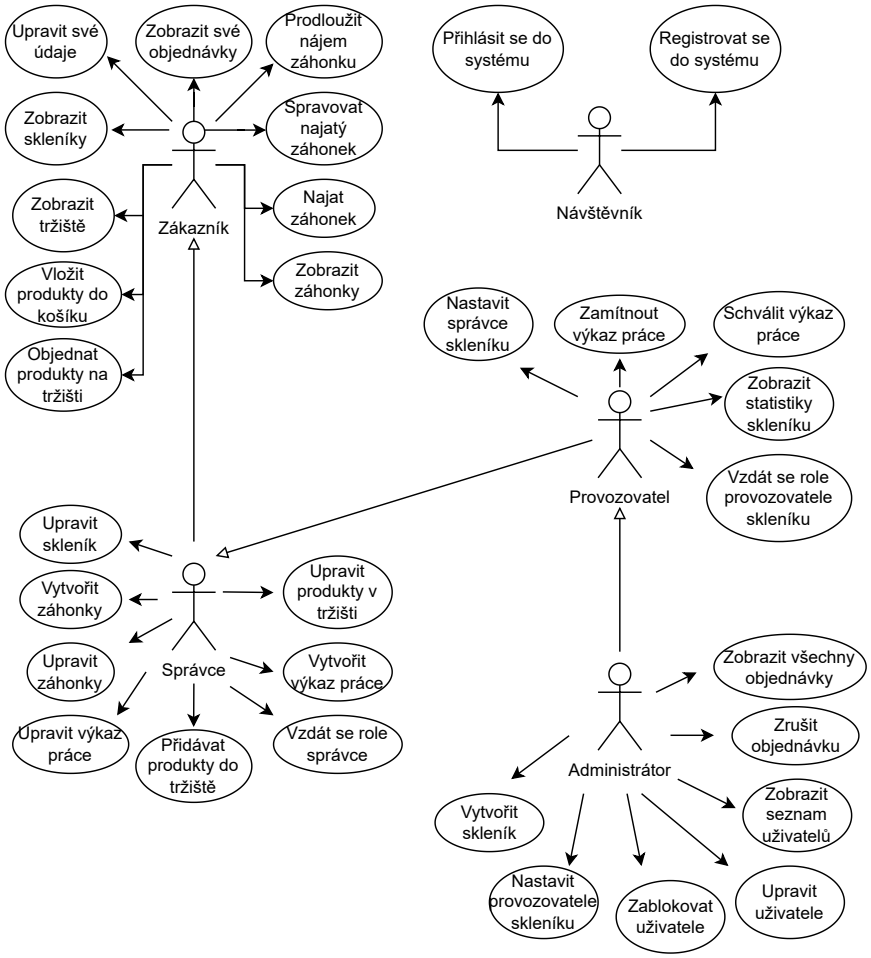
[19] STIEGLITZ, S., LATTEMANN, C., ROBRA BISSANTZ, S., ZARNEKOW, R. a BROCKMANN, T., ed. *Gamification*. 1. vyd. Basel, Switzerland: Springer International Publishing, říjen 2016. Progress in IS. ISBN 978-3319455570.

[20] TIMBÓ, R. *Front End Frameworks: What They Are, and Best Options* [online]. Leden 2023 [cit. 2024-1-15]. Dostupné z: https://www.revelo.com/blog/front-end-frameworks.

[21] TRAN, T. *Introduction to Web Servers* [online]. Červen 2022 [cit. 2024-3-23]. Dostupné z: https://www.digitalocean.com/community/conceptual-articles/introduction-to-web-servers.

Příloha A

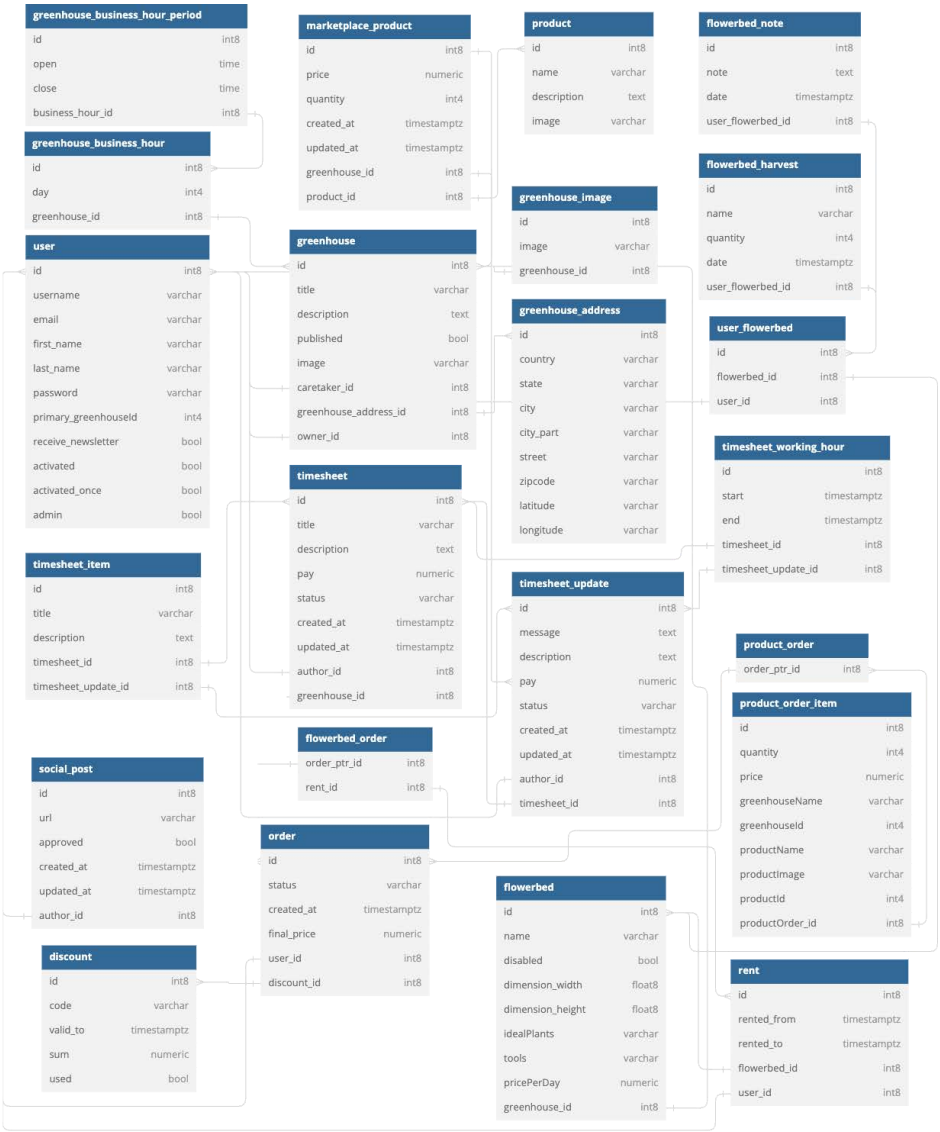
Diagram případu užití



Obrázek A.1: Diagram případu užití tohoto systému

Příloha B

Diagram databáze



Obrázek B.1: Schéma databáze

Příloha C

Plakát

Obrázek C.1: Plakát