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

WEBOVÁ APLIKACE PRO SPRÁVU CHYTRÉHO TRŽIŠTĚ
WEB APPLICATION FOR SMART MARKETPLACE MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ROMAN MAREK**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154714

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Marek Roman**
Program:	Informační technologie
Název:	**Webová aplikace pro správu chytrého tržiště**
Kategorie:	Webové aplikace
Akademický rok:	2023/24

Zadání:

1. Prozkoumejte způsob fungování chytrého tržiště a analyzujte uživatelské požadavky pro jeho správu.
2. Navrhněte vhodné uživatelské rozhraní reflektující požadavky ze strany uživatelů.
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

Hlavní motivace pro vytvoření práce se odvíjí od již existujícího projektu Ing. Davida Bažouta, který se jmenuje chytrý skleník. Ten funguje na bázi webové aplikace, ve které si jednotliví uživatelé sami volí podmínky, ve kterých své plodiny chtějí pěstovat. V rámci jednoho komunitního skleníku je dostupných více ploch, které si zákazník může pronajmout a následně na nich pak vysadit své plodiny. Část plochy skleníku k pronájmu není, protože ji spravuje správce příslušného skleníku. Zde přichází do hry možnost využití aplikace pro chytré tržiště. Ta umožňuje prodej produktů, které by přišly nazmar a zároveň i automatizaci úkolů správce, vyměňování zpráv mezi uživateli a další funkcionalitu, která by celý proces ulehčila. K implementaci byl využit framework `Django`, který využívá databázi `SQLite` pro uložení potřebných informací. Pro vytvoření uživatelského rozhraní bylo využito knihovny `React` a jejich komponent.

Abstract

The main motivation for creating the thesis derives from the already existing project of Ing. David Bazout, which is called the Smart Greenhouse. This project works on the basis of a web application in which individual users choose the conditions in which they want to grow their crops. Within a single community greenhouse, multiple areas are available for customers to rent and then plant their crops. However, a part of the greenhouse area is not available for rent because it is managed by the respective greenhouse manager. This is where the smart marketplace application comes in play. This enables the sale of the products that would go to waste. At the same time, however, it also includes automation of administrator tasks, message exchange between users and other functionality that would make the whole process easier. For implementation it was applied the `Django` framework, which uses the `SQLite` database to store the necessary information. For creation of user interface, the `React` library and its components were used.

Klíčová slova

webová aplikace, Python, Django, React, React-Redux, on-line tržiště, PayPal

Keywords

web app, Python, Django, React, React-Redux, on-line market, PayPal

Citace

MAREK, Roman. *Webová aplikace pro správu chytrého tržiště*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. David Bažout

Webová aplikace pro správu chytrého tržiště

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.

.....
Roman Marek
9. května

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Davidu Bažoutovi, za objasnění zadání práce, jejího účelu a také za věnovaný čas ve formě konzultací, které byly pro vývoj projektu velkou pomocí. Rád bych také poděkoval své rodině, přítelkyni a přátelům, kteří mi byli během studia velkou oporou.

Obsah

Kapitola 1

Úvod

V posledních letech dochází k zhoršení zemědělské situace v rámci celého světa. Dostat se ke kvalitním a bezpečným potravinám je stále těžší. Neustále dochází k vytváření nových pesticidů k ničení různých škůdců. Také spotřeba průmyslových hnojiv je čím dál tím větší. Vezměme si například dusíkatá hnojiva, které jsou hlavním zdrojem dusičnanů v potravinách. Tyto dusičnany jsou spouštěčem mnoha chemických reakcí, u kterých vznikají dusitany. Je dokázáno, že vliv dusitanu má velmi negativní dopad na malé děti. Jak se tedy vyhnout těmto rizikům? Jednoduše, vypěstovat si zeleninu sám.

Je nutné si uvědomit, že ne všichni lidé disponují prostředky, aby si své plodiny mohli sami vypěstovat. Přeci jen každý nemá zahradu a také samotný čas na to, aby se o ni denně staral. Ing. David Bažout přišel se skvělým řešením, jež všechny tyto problémy řeší.

Chytrý skleník je projekt, který krom klasického pěstování v půdě umožňuje i pěstování zeleniny pomocí hydroponie a poskytuje uživateli jednoduché ovládání, které ho nestojí prakticky žádný čas. Samotné pěstování pomocí hydroponie je časově velmi náročné, člověk totiž musí fyzicky do skleníku dojít a živný roztok upravit. U chytrého skleníku si všechno uživatel řídí pomocí webové aplikace z pohodlí domova. O veškerý chod se pak postará systém sám. Jako dalším logickým krokem pro rozšíření celého projektu se nabízí vytvoření tzv. chytrého tržiště.

Cílem aplikace pro správu chytrého tržiště je prodej plodin, které se v rámci chytrého skleníku vypěstují. Tedy v podstatě rozšíření působnosti chytrého skleníku i zákazníkům, kteří sami pěstovat nechtějí, ale pouze by si rádi zakoupili potraviny, které byly šetrně vypěstovány. Aplikace tedy bude sloužit jako klasický on-line obchod, ve kterém bude možné si potraviny zakoupit. Celý proces zakoupení bude však automatizován, takže si zákazník po zaplacení bude moci produkty sám vyzvednout. Tato skutečnost znamená, že aplikace nevyžaduje zaměstnance, který objednávky bude vyřizovat. Tím dojde k razantnímu snížení osobních nákladů na zaměstnance, které by se jinak mohly pohybovat v řádu deseti tisíců.

Aplikace je implementována v jazyce `Python` za využití frameworku `Django`, který nám umožňuje aplikaci rozdělit do tzv. `MVT` modelu. `MVT` model je společně s `MVC` aktuálně nejvyužívanějším v rámci implementace webových aplikací. Pro vytvoření uživatelského rozhraní byla využita knihovna `React` a její komponenty. K částečné stylizaci bylo taky využito knihovny `Tailwind`. Pro uchování vnitřního stavu aplikace byl využit `React-Redux`.

Kapitola 2

Vývoj webových aplikací

Vývoj robustních webových aplikací je komplexní proces, pří kterém musíme použít znalosti z různých odvětví. Nutnosti je znalost programovacích jazyků jak pro frontendovou část (`HTML`, `CSS`, `JS`), tak i pro backendovou část, která obsahuje samotnou logiku aplikace (`Python`, `PHP`, `C#`). Nezbytností je také využití verzovacího systému (`GIT`, `Mercurial`) a samotné otestování aplikace.

2.1 Architektura

Při vytváření větších aplikací může být velmi složité udržet zdrojový kód čitelný a přehledný. Obzvláště u aplikací, které kombinují více programovacích jazyků. Za tímto účelem vznikly různé architektonické vzory, které popisují, do jakých sekcí by měl být zdrojový kód rozdělen. V dnešní době se využívá např. `Model-View-Controller Architecture`, `Client-Server Architecture`, `Microservices Architecture`.

2.1.1 Model-View-Controller Architektura

`MVC` je architektonický model, který se hojně využívá při vývoji webových aplikací. Zajišťuje nám oddělení mezi logikou aplikace a samotným výstupem, který vidí uživatel. Rozdělení nám pak umožňuje snadnější a přehlednější práci, která může vést k jednodušší údržbě v případě výpadků atd. Jak už je z názvu patrné, rozděluje nám celou aplikaci na tři různé komponenty.

Model

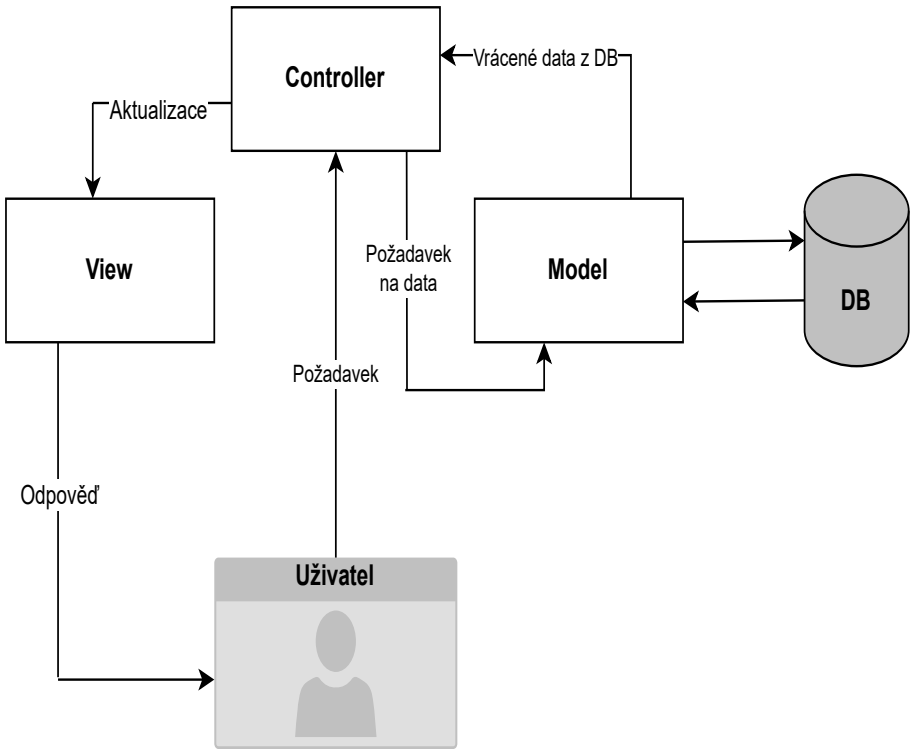
Model představuje data a aplikační logiku webové aplikace. Uchovává v sobě data či databázi a má na starosti veškeré výpočty nebo vyhodnocování pravdivosti. Neví o existenci dalších komponent – `controller`, `view` a slouží tak pouze k provedení jednotlivých procesů. Jelikož nemá možnost komunikovat s ostatními částmi, nemůže data nikam posílat. Proto si část `view` musí příslušná data vyzvednout sama.

View

`View` zobrazuje data zpracovaná modelem a uživatelské rozhraní webové aplikace. Určuje tedy, jakým způsobem mají být data prezentována v okně prohlížeče. Zpracované požadavky si sám vyzvedává z modelu.

Controller

`Controller` slouží k propojení výše zmíněných komponent. Obsahuje logiku, kterou aktualizuje `model` nebo `view` v závislosti na vstupu uživatele z aplikace.



Obrázek 2.1: Model komunikace v MVC

2.2 Backendové technologie

Pod pojmem backendový framework rozumíme kolekci nástrojů, knihoven a standardů, které umožňují vývojářům rychlejší a efektivnější vývoj webových aplikací. Umožňují zpracovávání požadavků ze strany klienta, jejich následné obsloužení a samotnou komunikaci s databází. Obsahuje také funkce, které pomáhají zajistit celkovou bezpečnost aplikace proti různým útokům jako je například `Cross Site Scripting`[1].

2.2.1 Python

Python je vysokoúrovňový interpretovaný skriptovací jazyk, který využívá dynamické typování [9]. Jedná se o objektově orientovaný jazyk, který podporuje všechny 4 hlavní pilíře objektově orientovaného programování:

- Abstrakce - koncept, který nezobrazuje komplexní implementační detaily, ale pouze esenciální informace pro uživatele
- Polymorfismus - odkazovaný objekt se chová podle toho, jaké třídy je instancí.

[1] https://owasp.org/www-community/attacks/xss/

- Zapouzdření - spojení dat a metod, které tyto data modifikují, do jednoho objektu. V `Pythonu` takové spojení nazýváme třídou. Ostatní třídy nemůžou přímo pracovat s proměnnými a metodami daného objektu. Díky tomu nemůže dojít k nekonzistentní změně dat
- Dědičnost - možnost dědit informace z ostatních tříd. Výhodou je znovupoužitelnost kódu bez nutnosti jeho kopírování

Mezi velkou výhodu také patří samotná struktura kódu. Na rozdíl od ostatních programovacích jazyků `Python` totiž využívá odsazení. Odsazení reprezentuje jednotlivé bloky kódu. Je také vysoce flexibilní, což vysvětluje jeho použití v různých programových odvětvích:

- Systémové programování
- Webové aplikace
- Analýza dat
- Umělá inteligence

V rámci vývoje webových aplikací jsou nejpopulárnější frameworky Django a Flask.

Flask

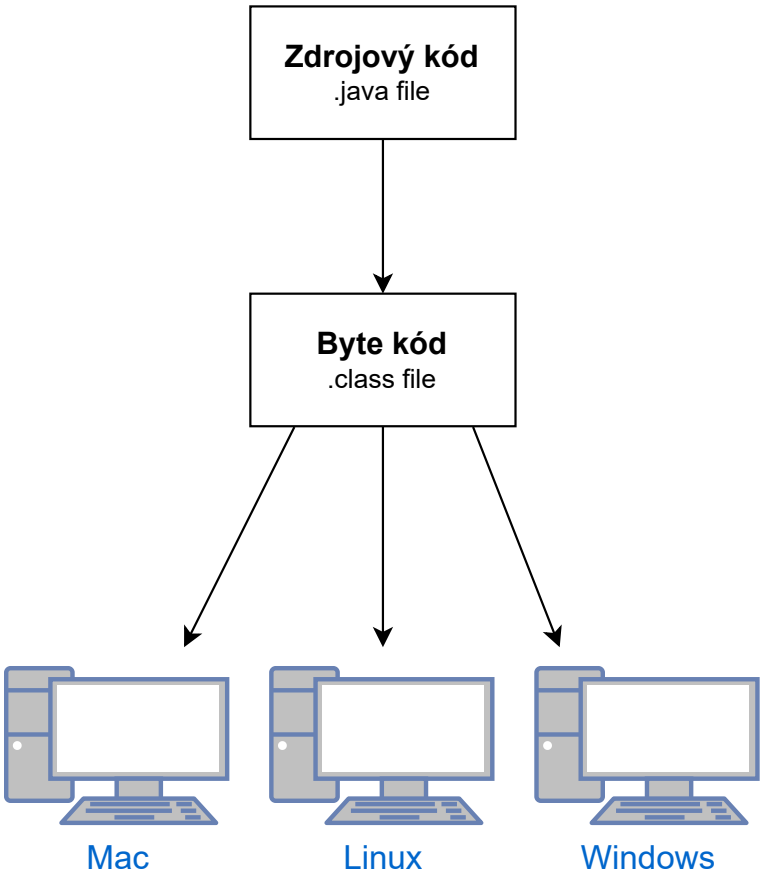
Flask je micro-framework, který nevyžaduje žádné externí knihovny k použití svých funkcí. Využívá se zejména k implementaci menších projektů. Umožňuje rychlý vývoj za pomocí jednoduchých funkcí. Nevýhodou `Flasku` je neschopnost vytvoření tzv. `multi-page` aplikací.

Django

Django je `full-stack` framework, který nabízí nástroje pro tvorbu jak klientské tak i serverové strany. Slouží převážně pro robustnější aplikace, kterým nabízí řadu vestavěných funkcí. Vývojář má tak možnost se spíše zaměřit na unikátní parametry jednotlivých aplikací, než dokola implementovat stejné standardní funkce [16]. Na rozdíl od Flasku obsahuje admin panel, který umožňuje zobrazení a vykonání CRUD operací nad modely v databázi. Dále také umožňuje různé typy autentizace uživatele, na rozdíl od Flasku, který podporuje autentizaci pouze pomocí cookies. Je tedy zřejmé, že Django je poměrně jasným vítězem, avšak velké množství funkcí, kterým Django disponuje, může být pro nové uživatele matoucí a může je od tohoto frameworku zcela odradit.

2.2.2 Java

`Java` je také objektově orientovaný jazyk, jehož syntaxe připomíná syntaxi jazyka `C`. Javu můžeme považovat jak za kompilovaný, tak i interpretovaný jazyk. Zdrojový kód `Java` souborů je prvně přeložen do binárního bytového kódu, který je následně vykonán interpretem, jenž nazýváme `Java Virtual Machine`[2]. Aplikace psané v `Javě` jsou spustitelné na libovolné platformě, kdežto `Python` vyžaduje interpret kompatibilní pro danou platformu.



Obrázek 2.2: Nezávislost Javy na architektuře

`Java` využívá statické typování. Je tedy nutné specifikovat typ proměnné. Hlavní výhodou je široká škála frameworků pro vytváření grafického uživatelského rozhraní. Mezi nejrozšířenější se řadí `JavaFX` a `Swing`. Pro vývoj backendové části, v rámci jazyku `Java`, se většina uživatelů uchyluje ke dvěma možnostem: `Spring` a nebo `Struts`.

Spring

`Spring` je `open-source` framework, který poskytuje komplexní programovací a konfigurační model pro moderní `Java` aplikace [14]. Pomáhá vývojářům vytvářet robustní aplikace, kterým stačí využívat tzv. `plain old Java object` [1]. Jedná se o vysoce flexibilní framework, který zvyšuje efektivitu kódování a redukuje celkový čas strávený nad vývojem aplikace. Nevýhodou může být právě jeho komplexnost. Obsahuje velké množství modulů a tříd, což prakticky znemožňuje jeho plnohodnotné využití novými vývojáři.

[2] https://www.javatpoint.com/jvm-java-virtual-machine

Struts

Struts je také open-source framework, který využívá MVC architektury. Umožňuje vývoj flexibilních, snadno udržovatelných webových aplikací v Javě. Výhodou je možnost centralizované konfigurace. Různé hodnoty mohou být reprezentovány jako XML, takže není nutné kódovat tyto informace přímo do Java souborů [4]. Zároveň všechny náležitosti související s webovou aplikací mohou být přivolány pomocí souboru Struts_config.xml.

2.2.3 PHP

PHP řadíme do skupiny skriptovacích jazyků, které se provádějí na straně serveru. PHP je na serveru závislé, protože na něm běží jeho interpret, který skripty provádí. Slouží ke generování HTML kódu stránky, jenž pak server odesílá do prohlížeče. PHP jazyk je nezávislý na platformě a dokáže pracovat s širokou škálou různých databází jako např.: MySQL, Oracle, PostgreSQL. Samotný PHP kód se zapisuje přímo do HTML stránky, což často může vést k neuspořádanému kódu. Velkou nevýhodou je však nedostatečná možnost samotného debuggování aplikace.

2.3 Frontendové technologie

Jak již bylo zmíněno v úvodu kapitoly 2, frontendová část se většinou skládá z 3 základních programovacích jazyků:

2.3.1 HTML

Jedná se o značkovací jazyk používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. Vytváří základní strukturu webové aplikace. Dalo by se říct, že HTML vytváří poznámky k textu pro počítač a to tak, aby věděl, jak s příslušným textem zacházet. K zajištění této skutečnosti jazyk HTML využívá množinu značek, které nazýváme tagy. Mezi jednotlivé značky se uzavírají části textu dokumentu.

```
1  <!DOCTYPE HTML>
2  <html>
3   <head>
4    <title> Titulek </title>
5   </head>
6   <body>
7    <p class="paragraph"> Hello world! </p>
8    <p> This is my new page. </p>
9    <hr id="line">
10  </body>
11 </html>
```

Výpis 2.1: Ukázka HTML kódu

Je zřejmé, že jednotlivé tagy zobrazené v ukázce 2.1 jsou ohraničeny úhlovými závorkami. Pokud se nachází mezi závorkami symbol lomítka indikujeme, že zde působnost daného tagu končí. Existují tzv. nepárové tagy, které neobsahují žádný obsah, u těchto značek není tato syntaxe nutná.

2.3.2 CSS

CSS je formátovací jazyk, který popisuje způsob zobrazení stránek napsaných v HTML. Hlavním úkolem CSS je oddělení obsahu a struktury stránky od samotného vzhledu. Syntaxe CSS funguje na bázi pravidel. Každé pravidlo obsahuje selektor a blok deklarací. Každá deklarace se skládá z identifikátoru vlastnosti a její hodnoty. Změnit vizuální podobu příslušného tagu je možné i pomocí atributu style, ve kterém následně můžeme deklarovat příslušné styly. V praxi se však tato možnost nevyužívá, protože odporuje základnímu pravidlu CSS o oddělení obsahu.

```
1   .paragraph {
2       color: red;
3       font-size: 28px;
4   }
5   #line {
6       border: 10px solid green;
7   }
8   p:last-child {
9       color: blue;
10  }
```

Výpis 2.2: Ukázka CSS kódu

Ukázka ve výpisu 2.2 zobrazuje různé typy selektorů, které změní vzhled elementů popsané v 2.1. Široká škála selektorů je esenciální, jelikož u robustnějších aplikací může neustále vytváření nových identifikátorů vést k celkově méně přehlednému kódu.

2.3.3 JavaScript

JavaScript je interpretovaný, dynamicky typovaný jazyk. Jedná se o multiplatformní, objektově orientovaný jazyk. Často se označuje jako tzv. jazyk webového prohlížeče [3]. Jeho využití je tedy primárně pro tvorbu webových stránek. Jelikož je v dnešní době takřka nutností reagovat na vstupy uživatele a dynamicky stránku v závislosti na nich měnit, je použití JS nezbytnou součástí tvorby webových aplikací. V posledních letech však dochází k integraci JavaScriptu i do serverové částí. Populárním prostředím, které dokáže JS kód spouštět mimo prohlížeč je například node.js.

Je nutné podotknout, že prostý JS bez dodatečných funkcí a knihoven se využívá primárně u projektů, které jsou jednoduché a nevyžadují týmovou spolupráci. V případě vývoje robustních, komplexních projektů, je doporučeno využití JS knihovny, případně frameworku.

React

React je open-source knihovna jazyka JavaScript, která slouží pro tvorbu uživatelského rozhraní webových aplikací. Tato knihovna byla vytvořena v roce 2011 vývojáři Facebooku. Důvodem bylo právě vytvoření komplexního, vysoce výkonného uživatelského rozhraní. React je velmi užitečný v případech, kde je potřeba častá interakce s uživatelem. React umožňuje rychlou odpověď, jelikož využívá virtuální DOM 2.3, který si uchovává veškeré změny provedené v rámci webové aplikace. Díky tomu pak aktualizuje pouze části, u kterých byla detekována změna [15]. Důležitou funkcí Reactu je také možnost vytvoření vlastních komponentů, které pak můžeme v rámci projektu znovu použít.

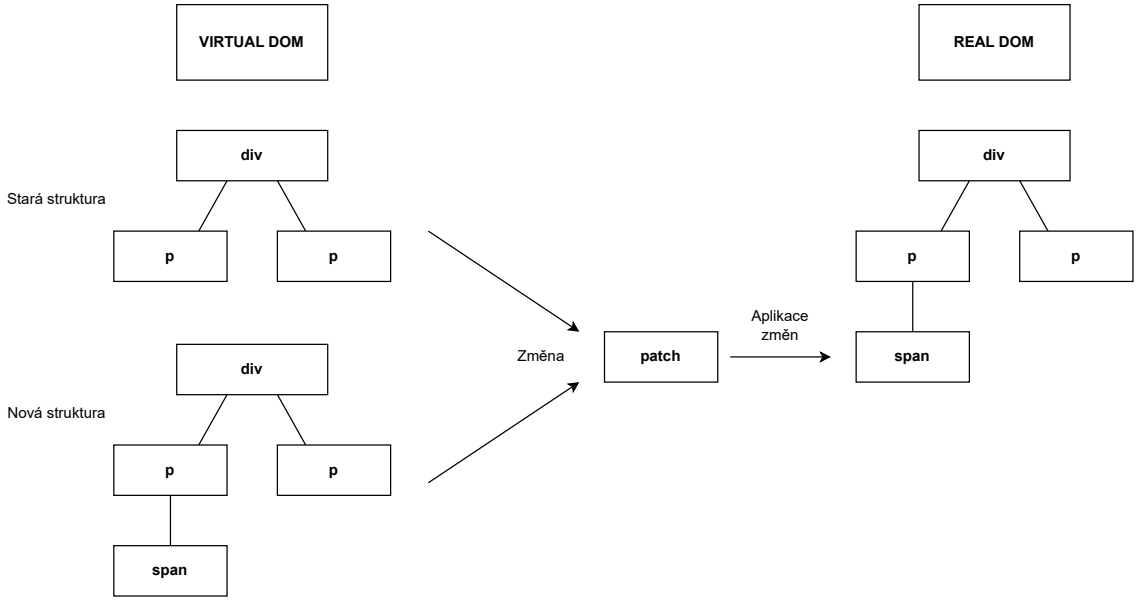
```
1   import React from 'react'
2   const header = () => {
3    return (
4     <>
5       <h1> This is my new header! </h1>
6       <p> Hello world! </p>
7     </>
8    )
9   }
10  export default Header
```

Výpis 2.3: Demonstrace, jak vytvořit jednoduchý komponent v Reactu

V příkladu 2.3 lze vidět, že příkaz `return` umožňuje vracet pouze jeden uzel. Je tedy nutné jednotlivé elementy zapouzdřit. K tomu můžeme využít `tag` jako je `div`. V praxi však nedává smysl vytvářet blokový element, který v rámci aplikace nemá význam. Proto `React` umožňuje využití prázdného `tagu`, který nazýváme fragment.

Vue.js

`Vue.js` je `open-source JS` framework, který slouží pro budování uživatelského rozhraní a webových aplikací. Slouží k zjednodušení procesu tvorby interaktivních a dynamických aplikací. Stejně jako `React` využívá virtuální `DOM` 2.3. Virtuální `DOM` je reprezentován jako stromová struktura. Každý element, který se v aplikaci nachází, je zobrazen ve stromové struktuře jako uzel. V případě, že dojde k jakékoliv změně u libovolného elementu, je vytvořen nový strom. Poté dochází k porovnání nového a starého stromu a následně jsou změny mezi nimi uloženy. `Vue` pak najde nejlepší možnost, jak tyto změny aplikovat do reálného `DOMu`.



Obrázek 2.3: Popis funkčnosti virtuálního DOMu

2.4 Uživatelské rozhraní

Uživatelské rozhraní je souhrn způsobů, jakými uživatelé ovlivňují chování systémů. Uživatelské rozhraní slouží k interakci člověka se strojem. Cílem této interakce je efektivně manipulovat a kontrolovat stroj z pozice člověka, mezitím, co stroj kontinuálně informuje člověka o svém stavu tak, aby mohl člověk provádět rozhodnutí. [13] Je to tedy pomyslný prostředník, který zprostředkovává komunikaci mezi člověkem a počítačem nebo informačním systémem. Dobré uživatelské rozhraní by mělo splňovat následující kritéria.

- Jednoznačnost
- Stručnost
- Podobnost
- Konzistence
- Estetičnost

`UI` design řeší pouze jak daný web vypadá, proto se spojuje ještě s `UX` designem. `UX` se zjednodušeně zabývá tím, jaký má `UI` na uživatele dopad a jakou z něj má zkušenost. Cílem `UX` tedy je vytvoření aplikace, která bude pro uživatele jednoduchá a intuitivní, což povede k příjemnému uživatelskému zážitku.

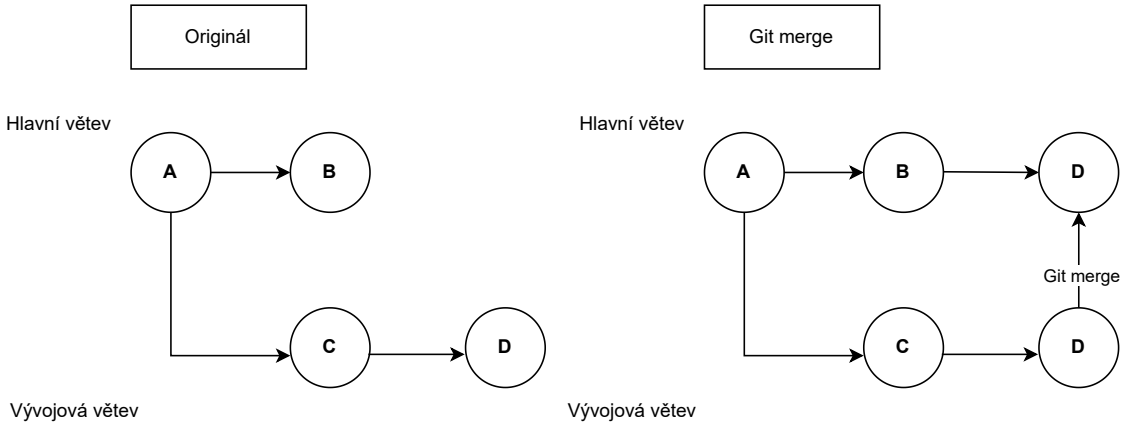
2.5 Verzovací systémy

Pod pojmem verzování rozumíme uchovávání historie veškerých změn provedených v kódu nebo datech. Nejčastěji se používá pro sledování změn v zdrojovém kódu během vývoje. Verzovací systém obsahuje repositář pro zdrojový kód softwaru a nabízí různorodé funkce pro manipulaci se zdrojovým kódem v repositářích. Umožňuje uživateli nahrávat a stahovat jednotlivé verze svého projektu.

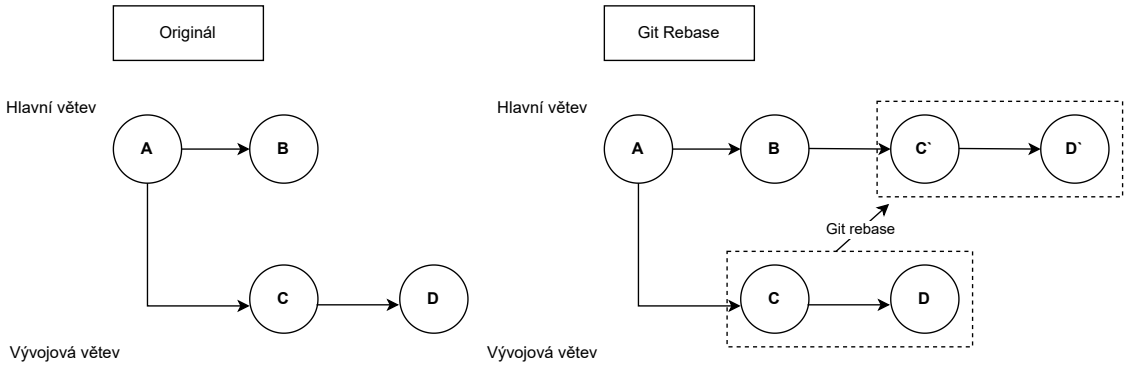
Změny kódu by se do repositáře měly přidávat v malých dávkách. Tyto změny nazýváme `commity`. Systém si u jednotlivých `commitů` ukládá jejich logy. To nám umožňuje například změnu vrátit nebo zrekonstruovat dávnou funkční podobu aplikace. Změny můžeme nahrávat pomocí příkazu `push`, stahovat (`pull`) a nebo kombinovat různé vývojové větve pomocí `merge` 2.4 nebo `rebase` 2.5.

Typy verzovacích systému mohou být centralizované nebo distribuované. Mezi centralizované se řadí například `Subversion`[3]. Nevýhodou těchto systému je, že nám neumožňují mít celou kopii projektu lokálně uloženou. Historie kopií se ukládá na jeden server. Pokud chceme s těmito změnami pracovat, případně je vrátit, je nutná komunikace se serverem. Tento problém řeší právě distribuované systémy. Mezi nejrozšířenější se řadí `Git`[4] či `Mercurial`[5].

[3] https://subversion.apache.org/
[4] https://git-scm.com/
[5] https://www.mercurial-scm.org/



Obrázek 2.4: Diagram změn v případě použití merge



Obrázek 2.5: Diagram změn v případě použití rebase

2.6 Testování

Testování je proces, při kterém se ujišťujeme že vyvíjený software odpovídá specifikaci, kterou jsme si ustanovili se zákazníkem. Web testování zajišťuje bezproblémový chod a funkčnost aplikace. Odhaluje veškeré chyby a `bugy` ještě před tím, než se mohou dostat k uživatelům. Dále můžeme testovat dobu načtení různých komponent stránky, responzivnost, bezpečnost apod. Mezi nejpoužívanější testy pro vývoj různorodého softwaru patří [12]:

- Jednotkové testy - obvykle testují univerzální knihovny, nepíšeme je pro kód specifický pro danou aplikaci [17]. Jednotkové testy testují třídy, přesněji jejich metody, jednu po druhé. Předávají jim různé vstupy a zkouší, zda jsou jejich výstupy korektní. Řadí se mezi tzv. `whitebox` testy, což znamená, že víme, jak testovaný kód uvnitř funguje.

- Akceptační testy - každý test obvykle testuje nějakou konkrétní funkčnost. Testujeme tedy specifickou logiku aplikace. Konkrétně její výstup, nikoliv její vnitřní kód. Jelikož neřeší to, jak je aplikace naprogramovaná, jedná se o tzv. `blackbox` testy.
- Integrační testy - u robustních aplikací se vývoj dělí na více služeb, integrační testy dohlíží, aby do sebe jednotlivé služby zapadaly.
- Systémové testy - slouží k otestování na produkčním zařízení. Využívá např. zátěžové testy, které mohou simulovat připojení tisíce uživatelů do aplikace.

Kapitola 3

Návrh aplikace

V první řadě bylo nutné samotné problematice porozumět. Jak již bylo zmíněno v úvodní kapitole 1, tato aplikace již navazuje na existující projekt `Sensorie`. Tento projekt je již funkční a v provozu. Jelikož je tato bakalářská práce jakýmsi rozšířením tohoto projektu, bylo nutné se seznámit s činností chytrého skleníku a zároveň s jeho možnostmi. Vzhledem k tomu, že vedoucí této práce Ing. David Bažout je autorem tohoto projektu, bylo mi umožněno si celý tento proces na živo prohlédnout. V rámci této exkurze mi bylo také ukázáno dosavadní řešení chytrého tržiště. Jednalo se o prostý plakát s možností `QR` platby.

Po prvotní analýze uživatelských potřeb byl zpracován drátěný model, který byl vytvořen ve webové aplikaci `Figma`. Tento model byl následně otestován na potenciálních uživatelích a na jeho bázi byl vytvořen `mockup`. `Mockup` je de facto maketa, která však už obsahuje samotný design stránky. Poté došlo k další iteraci testování právě za použití této makety.

V neposlední řadě bylo nutné vybrat technologie, ve kterých bude aplikace implementována. Kapitoly 2.2 a 2.3 nabízí poměrně důkladnou analýzu nejpopulárnějších technologií a jejich vzájemné porovnání.

V rámci backendové části bylo hlavní rozhodování mezi použitím frameworku `Django`, který má syntaxi jazyka `Python`, a frameworku `Spring` v jazyce `Java`. Jazyk `PHP` byl vyřazen kvůli limitovaným možnostem `debuggování`. `Flask` byl zajímavou možností, ale bohužel se hodí jen k vývoji jednoduchých, většinou jednostránkových aplikací, čili i ten byl vyloučen. Na konec jsem se rozhodl pro framework `Django`, jelikož má rozsáhlou dokumentaci, která vše důkladně popisuje a jevil se jednodušší.

V rámci frontendové části byla vybrána knihovna `React`. Jedná se o nejrozšířenější `JavaScriptovou` knihovnu, se kterou mám poměrně rozsáhlé zkušeností.

3.1 Funkce aplikace

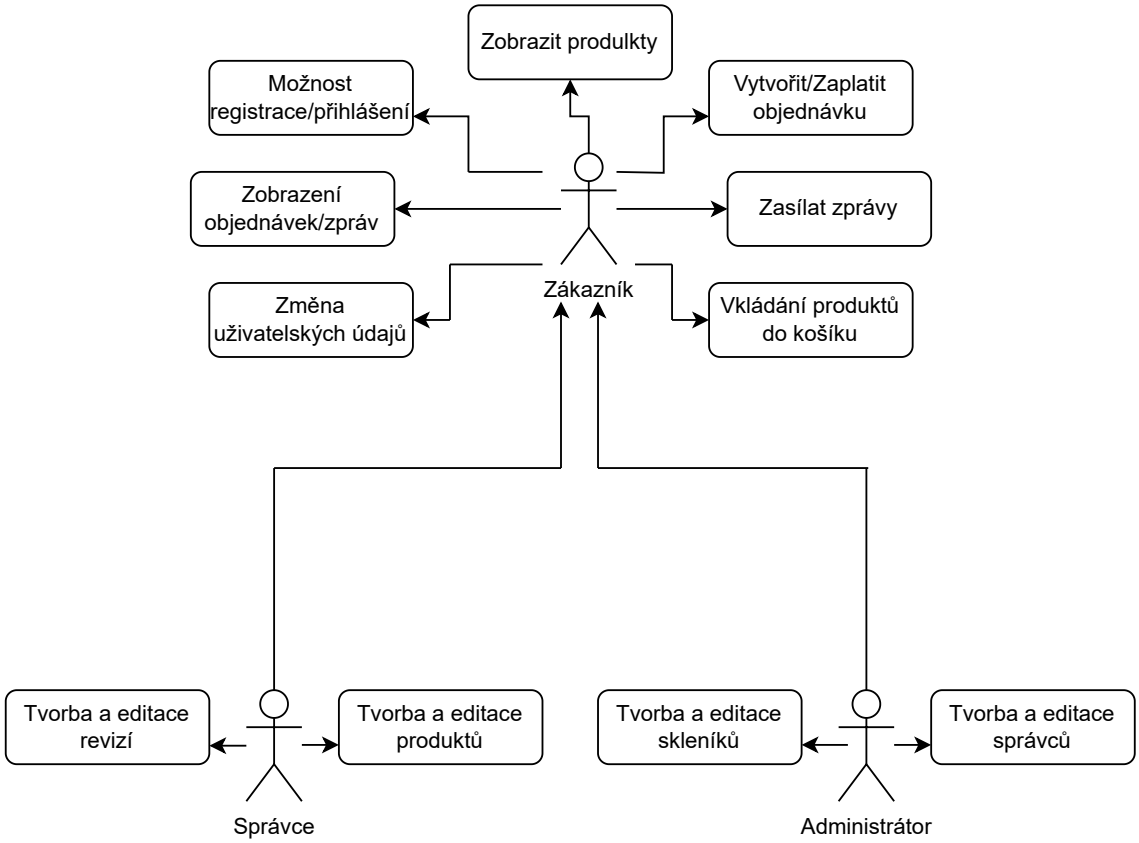
Aplikace podporuje tři typy uživatelů - zákazník, správce a administrátor. Jelikož musíme u uživatele rozlišit jeho typ, je nutné zavést autentizační systém, který vyhodnotí, jaké pravomoci bude v systému mít. Proto je vyžadována registrace a následné přihlášení uživatele. Zákazník však nemusí být registrován, aby si něco mohl zakoupit.

Každý uživatel, nehledě na roli, má možnost si zobrazit nabídku produktů. V případě, že má zájem o některý z produktů, má možnost jej umístit do svého košíku a následně vyřídit svou objednávku. V případě, že je uživatel registrován a zároveň přihlášen, je jeho role obohacena o další funkce. Má možnost si zobrazit své objednávky, změnit svoje údaje

a další. Důležitou možností registrovaných uživatelů je schopnost zasílat zprávy ostatním uživatelům. V případě, že uživatel při vyzvednutí potravin viděl ve skleníku něco chybného, může obeznámit správce i ostatní uživatelé. Stačí mu k tomu pouze e-mail daného uživatele.

Funkcionalita správce je obohacena o možnost vytváření tzv. revizí. Revizí myslíme úkony spojené s údržbou skleníku. Správce má možnost tyto revize vytvářet, editovat a přehledně si zobrazit jejich popis. Zároveň je správci umožněno také vytváření nových produktů a jejich editace. Správce si ve svém panelu jednoduše zobrazí jednotlivé produkty, u kterých může měnit příslušné atributy. Správce má možnost produkt kompletně odstranit např. v případě, že daný produkt už aktuálně není skladem.

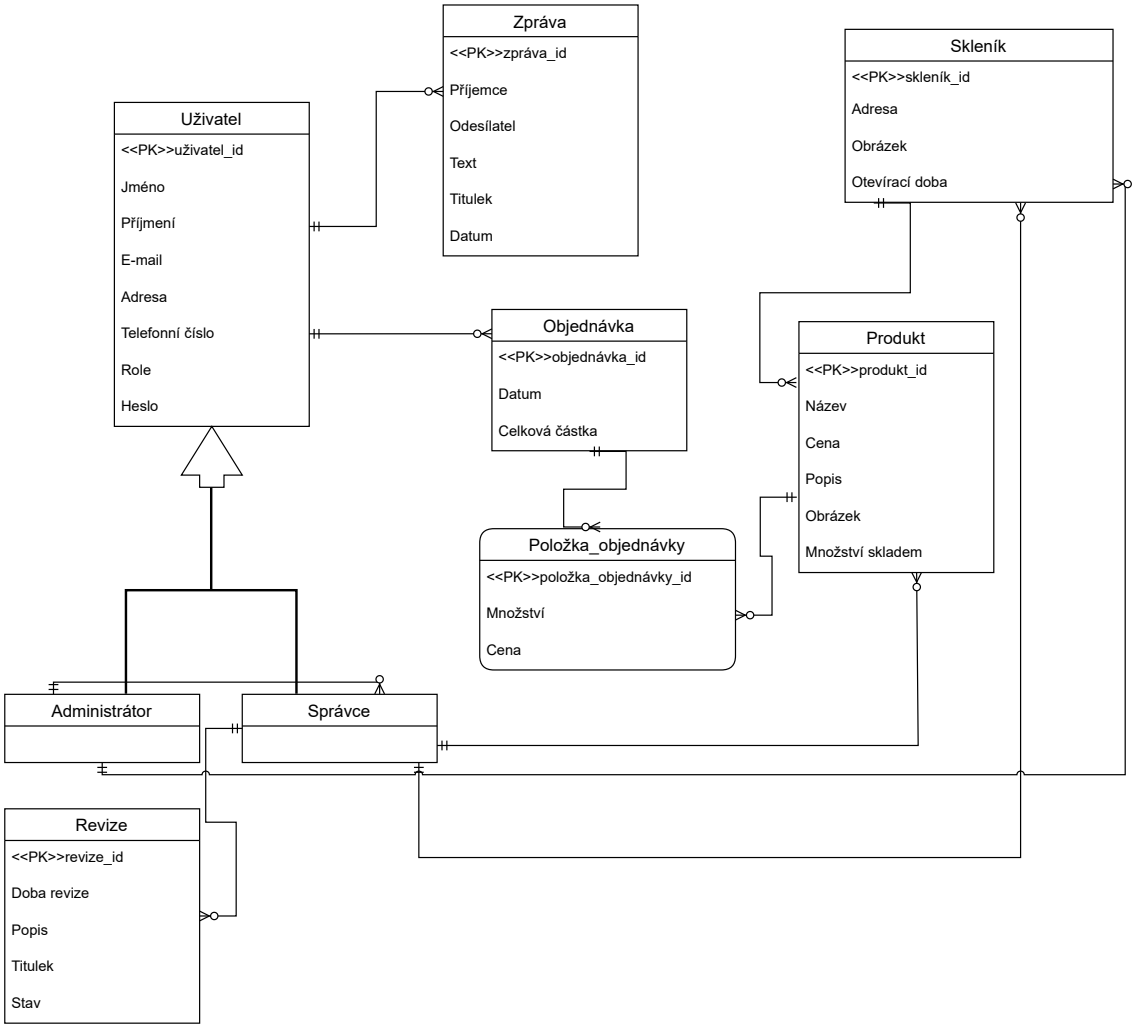
Administrátor má veškerou kontrolu nad aplikací. Disponuje všemi funkcemi regulérního uživatele. Na víc má možnost tvorby a editace jednotlivých skleníků. Administrátor může také jednotlivé skleníky odstranit např. v případě, že příslušný skleník už nebude funkční. Další důležitou funkcí je registrování správců. Administrátor vytváří účet pro jednotlivé správce. Může jim sám editovat údaje a případně status správce odebrat.



Obrázek 3.1: Use case diagram uživatelů aplikace

3.2 Databáze

Databáze aplikace se dá abstraktně znázornit pomocí `Entity-Relationship-Diagramu` zkráceně `ERD`. Využívá se právě před implementováním databáze, což nám usnadňuje snadné `debuggování` případných problémů. Součásti tohoto `ERD` je několik entit, každá entita je definována primárním klíčem, který musí být unikátní. Slouží nám k identifikací konkrétní instance. Součástí každé entity také můžou být atributy.



Obrázek 3.2: ER diagram databáze

Uživatel

Entita uživatele je základním kamenem celé aplikace. Obsahuje řadu atributů, které jsou vyžadovány při registraci.

- `uživatel_id` - jedná se o jedinečný identifikátor. Tento identifikátor nemusíme explicitně definovat, `Django` jej totiž vytváří samo. Stejně tak se `Django` stará i o jeho inkrementaci v případě vytvoření nových instancí.
- `Jméno` - křestní jméno uživatele, tento údaj je vyžadovaný při registraci.
- `Příjmení` - Stejně jako u jména je uživatel povinen tuto informaci poskytnout.
- `E-mail` - funguje jako přihlašovací jméno. V rámci databáze musí být tento e-mail unikátní.
- `Adresa` - v budoucím vývoji může tento údaj být užitečný při tvorbě statistik. Proto je vyžadován.
- `Telefonní číslo` - stejně jako u atributu `Adresa`, může sloužit např. k notifikacím uživatelů v budoucím vývoji aplikace.
- `Role` - tento atribut symbolizuje oprávnění, které příslušný uživatel má. Jelikož stránky, které jsou určené pro správu aplikace, nejsou přístupné klasickým uživatelům, je nutné tento údaj uchovávat v databázi. Atribut `Role` uživatel při registraci nevyplňuje. Je automaticky nastavený na oprávnění normálního uživatele aplikace. Pouze administrátor může registrovat uživatele s vyššími pravomocemi.
- `Heslo` - stejně jako u atributu `E-mail` bude sloužit k přihlášení uživatele do systému. `Django` zajišťuje bezpečnost ukládání hesel do databáze a to tak, že si uchovává pouze jejich `hash`.

Krom řady atributů disponuje entita `Uživatel` i vztahy. Na obrázku 3.2 si můžeme všimnout, že má tato entita dva různé `1:N` vztahy. První vede k entitě `Zpráva`. Slovně bychom tedy mohli říct, že jeden konkrétní uživatel může odeslat několik zpráv, avšak jedna konkrétní odeslaná zpráva náleží pouze jednomu uživateli. Druhý vede k entitě `Objednávka`. V rámci aplikace může jeden uživatel vytvářet řadu objednávek, ale jedna konkrétní objednávka se váže pouze k jednomu uživateli. Entity `Administrátor` a `Správce` dědí atributy z entity `Uživatel`.

Objednávka

Entita `Objednávka` je esenciální pro chod aplikace. V závislosti na ni je nutné ostatní entity aktualizovat. Její atributy jsou vytvořeny automaticky poté, co zákazník provede platbu.

- `objednávka_id` - jedinečný identifikátor, který přiděluje `Django`.
- `Datum` - symbolizuje datum a čas, kdy přesně byla objednávka vytvořena.
- `Celková částka` - jedná se o sumu, kterou uživatel zaplatil. Tato částka zahrnuje všechny položky objednávky.

Objednávka disponuje dvěma různými vztahy. První k entitě `Uživatel`, který už byl popsán v části 3.2. Druhý vztah k entitě `Položka_objednávky` je typem `1:N`. Slovně tedy popisujeme, že jedna objednávka může mít několik položek. Na druhou stranu jedna určitá položka se však váže pouze k jedné objednávce.

Produkt

Entita `Produkt` znázorňuje plodinu, která je vypěstována ve skleníku, a kterou si budou moci uživatelé zakoupit. V rámci korektního zobrazení pro uživatele obsahuje entita `Produkt` několik atributů. Ty jsou při vytvoření instance této entity vyžadovány.

- `produkt_id` - jednoznačný identifikátor.
- `Název` - název produktu, který jasně zákazníkům řekne, o jaký produkt se jedná.
- `Cena` - cena, kterou zákazník zaplatí při koupi příslušného produktu. Tato částka je účtována za jeden kus produktu.
- `Popis` - detailnější popis produktu, ve kterém se uvádí původ plodin a další důležité informace, které mohou zákazníkovi ulehčit výběr.
- `Obrázek` - obrázek produktu.
- `Množství skladem` - hodnota, která určuje, jaké množství produktu je ještě skladem. Tato hodnota se aktualizuje v závislosti na objednávkách, které tento produkt obsahují.

Produkt má několik vztahů, všechny tyto vztahy jsou typu `1:N`. První vztah je s entitou `Skleník`. Vztah popisuje, že jeden skleník může obsahovat několik produktů, ale jeden konkretní produkt náleží pouze jednomu skleníku. Druhý vztah popisuje vazbu mezi produktem a správcem. Jeden produkt je vytvořen pouze jedním správcem, avšak jeden správce má možnost vytvořit produktů několik. V neposlední řadě produkt může být součástí několika položek objednávky, ale jedna konkrétní položka se pojí pouze s jedním produktem.

Skleník

Entita `Skleník` slouží v aplikaci jako kategorie. Každý skleník reprezentuje určitou množinu rozdílných produktů, které si zákazník může zakoupit. Jelikož si produkty zákazník vyzvedává sám, je nutné specifikovat určité atributy.

- `skleník_id` - jednoznačný identifikátor
- `Adresa` - je důležitým atributem, který reprezentuje místo, kde si zákazníci své produkty mohou vyzvednout.
- `Obrázek` - fotografie skleníku, kde si lidé mohou vyzvednout své objednávky. Slouží k rychlejší orientaci uživatelů, kteří podle fotky mohou hned rozeznat, o jaký skleník se jedná.
- `Otevírací doba` - časový interval, ve kterém si lidé mohou objednávky vyzvednout.

Skleník má vztah s entitou `Správce`, který značí, že jeden skleník spravuje pouze jeden správce. V rámci aplikace však jeden správce může spravovat skleníku několik. S entitou `Administrátor` je tomu podobně, skleník se váže k jednomu konkrétnímu administrátorovi, který má však možnost založit skleníků několik. Vazba s entitou `Produkt` již byla popsána výše.

Zpráva

Entita `Zpráva` uchovává komunikaci mezi jednotlivými uživateli.

- `zpráva_id` - jednoznačný identifikátor
- `Příjemce` - e-mail uživatele, kterému má být příslušná zpráva doručena.
- `Odesílatel` - e-mail uživatele, který zprávu vytvořil.
- `Text` - samotný text zprávy, který si uživatelé vyměňují.
- `Titulek` - reprezentuje funkci předmětu v e-mailech.
- `Datum` - časový údaj, který se automaticky vytváří s vytvořením zprávy.

Položka objednávky

Položka objednávky slouží jako pomocná entita k entitě `Objednávka`.

- `Množství` - určuje objednaný počet kusů příslušného produktu.
- `Cena` - částka, která odpovídá množství produktu, které bylo zakoupeno.

Revize

Entita `Revize` symbolizuje poznámky, které si může každý správce zaznačit. Každá revize by měla obsahovat několik náležitostí.

- `revize_id` - jednoznačný identifikátor
- `Doba revize` - časový údaj, který odpovídá době, jakou musel správce vynaložit na údržbu skleníku. Tuto informaci pak může přeposlat administrátorovi, který mu za příslušné hodiny zaplatí.
- `Popis` - text, který slouží správci pro zaznamenání jednotlivých problémů, které musí vyřešit.
- `Titulek` - nadpis pro odlišování jednotlivých revizí.
- `Stav` - určuje, jestli je příslušná revize aktivní nebo již splněna.

V rámci aplikace může příslušná revize náležet pouze jednomu správci, ten má však možnost jich vytvořit několik.

Administrátor a Správce

Jak již bylo řečeno, obě tyto entity dědí atributy z entity `Uživatel`. Administrátor má však vztah se správcem a to takový, že administrátor může zaregistrovat několik správců. Vytvořený správce se zodpovídá jednomu administrátorovi.

3.3 Návrh uživatelského rozhraní

Uživatelské rozhraní by mělo být intuitivní a jednoduše sestaveno, aby maximalizovalo uživatelskou přívětivost. Uživatel by se měl snadno zorientovat, na jaké stránce se aktuálně nachází. Důležitým komponentem je hlavička s navigací, která by uživatelům měla sloužit pro snadný přechod mezi jednotlivými stránkami [10]. V rámci své činnosti by uživatel neměl přemýšlet nad dalšími kroky. Proces by měl být intuitivní a pochopitelný pro uživatelé všech věkových kategorií.

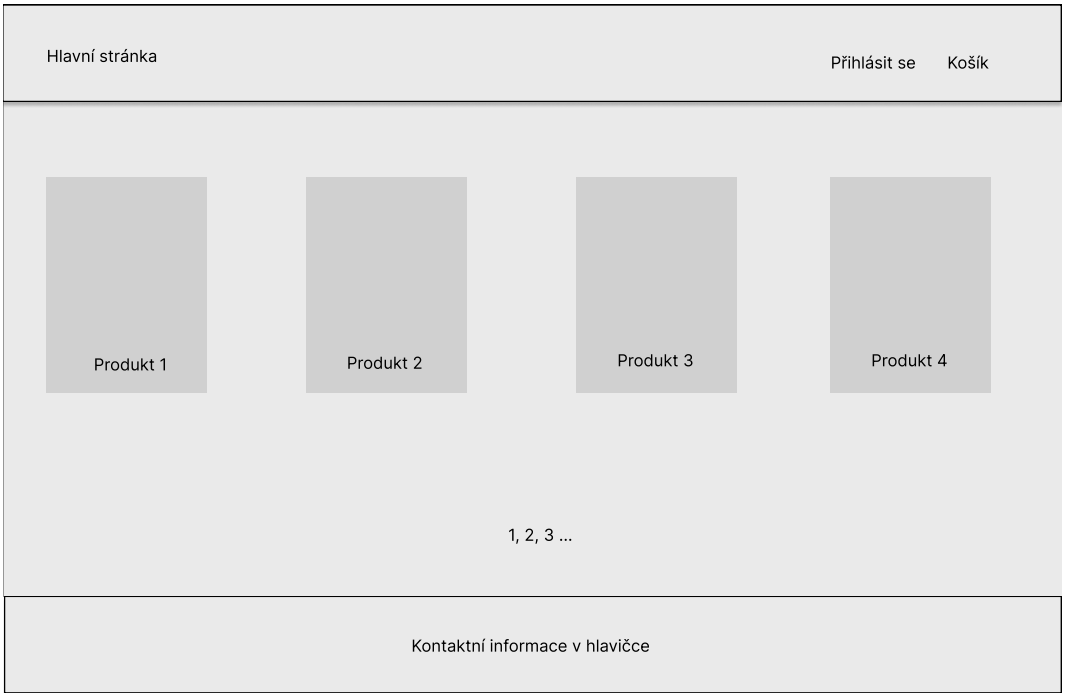
V rámci vývoje uživatelského rozhraní je doporučeno začít s prototypy, které pouze naznačují, jak celkové prostředí bude fungovat a vypadat. Tyto prototypy nám umožňují jednoduché testování zákaznické spokojenosti s rozhraním, aniž bychom se pouštěli do samotného vývoje projektu. Tento proces tedy zlepšuje efektivitu vývoje a náklady spojené s jeho tvorbou. Jako návrhy můžeme využít drátěný model či `mockup`.

3.3.1 Drátěný model

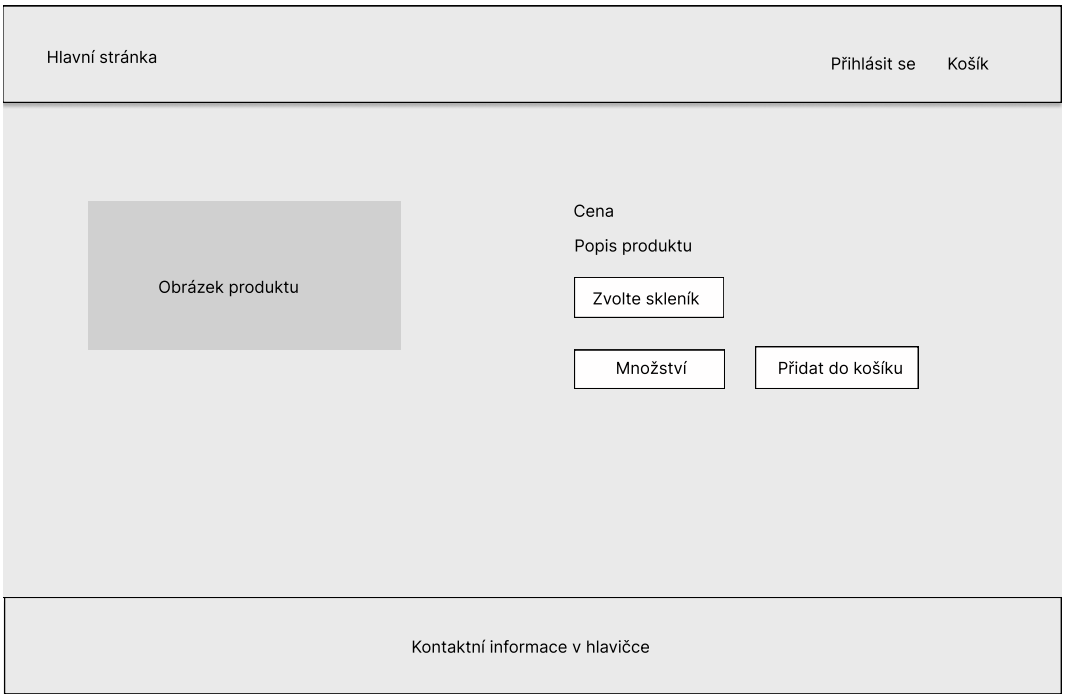
Drátěný model je zjednodušený, schematický náčrt, který se používá při návrhu webových stránek, mobilních aplikací nebo jiných uživatelských rozhraní [5]. Umožňuje designérům a vývojářům diskutovat a uvědomit si různé možnosti uspořádání obsahu a funkcionality předtím, než se přistoupí k detailnějším grafickým prvkům. Užitečný drátěný model by měl obsahovat tyto prvky:

- Rozložení stránky - ukazuje umístění různých částí obsahu, jako jsou hlavička, obsahová oblast a patička.
- Navigace - zobrazuje navigační prvky, jako jsou menu, odkazy a tlačítka, a jejich umístění na stránce.
- Obsahové bloky - identifikuje hlavní oblasti obsahu a zobrazuje, jak jsou uspořádány na stránce.
- Interaktivní prvky - tlačítka, ovládací prvky.

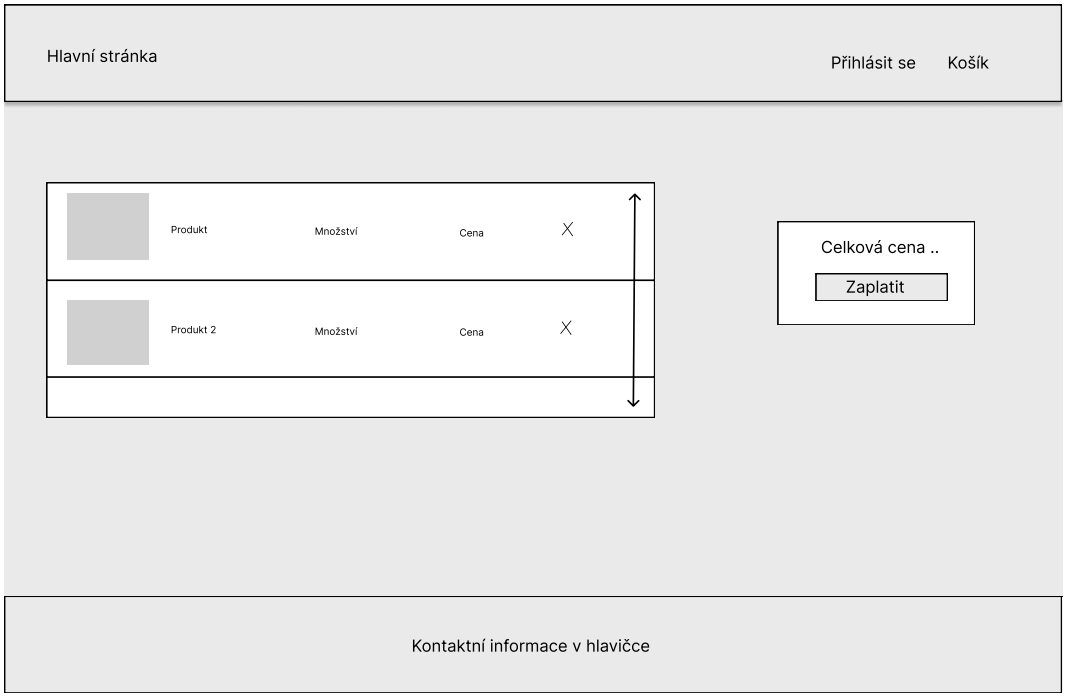
V závislosti na těchto poznatcích byl zhotoven drátěný model pro jednotlivé stránky, který byl následně potencionálními uživateli otestován.



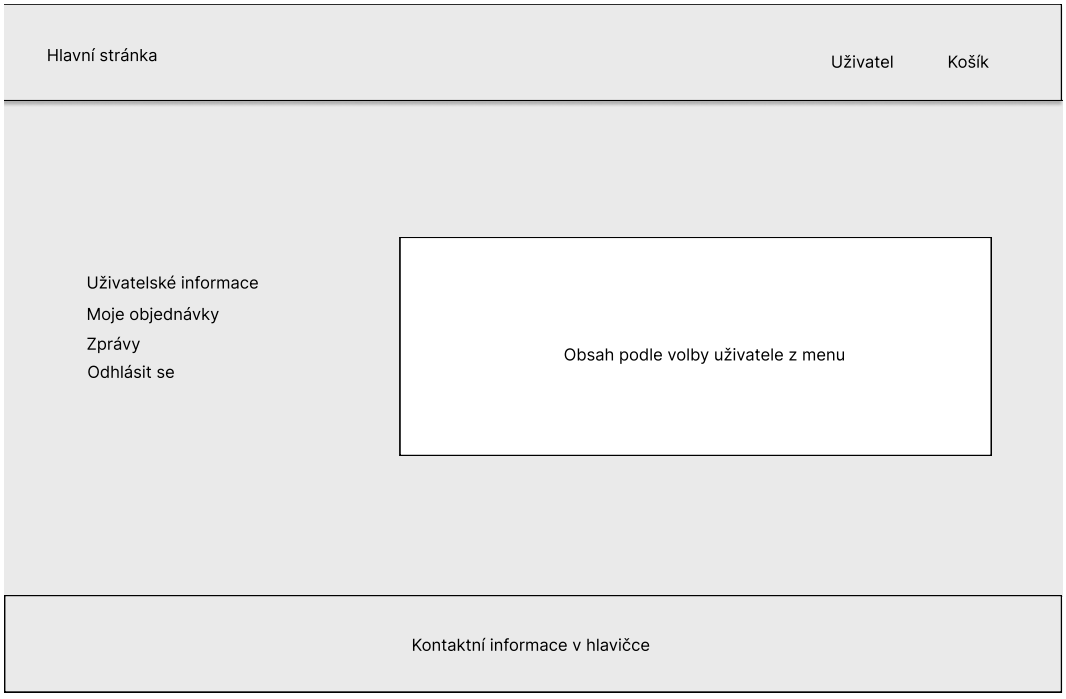
Obrázek 3.3: Stránka s nabídkou produktů, které si uživatel může zvolit.



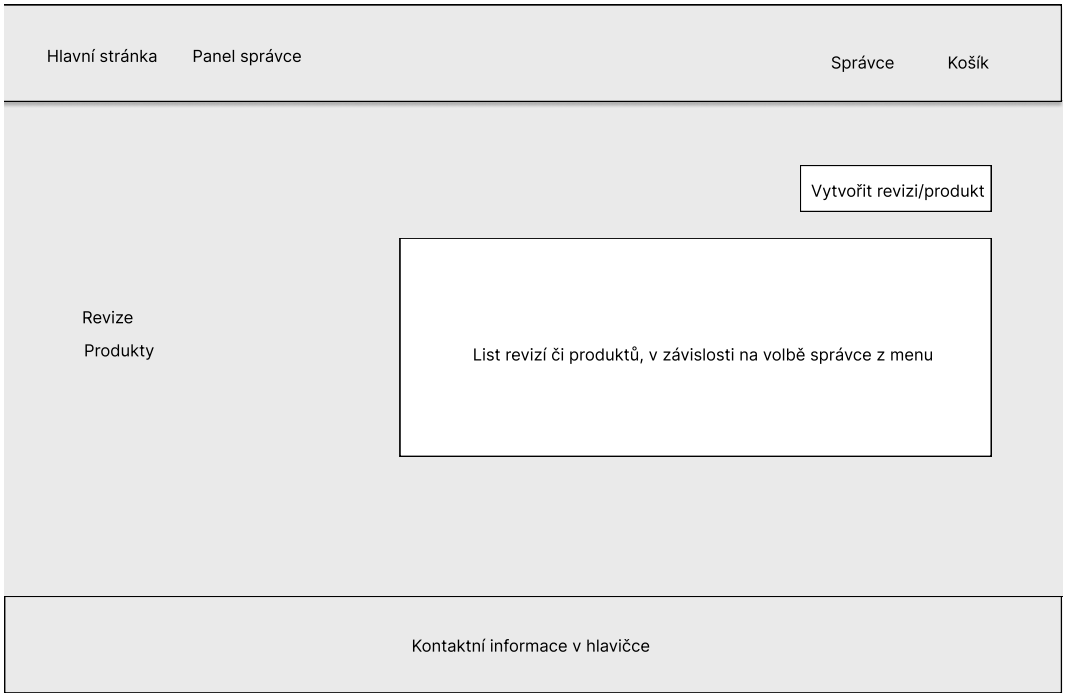
Obrázek 3.4: Detail produktu, který se zobrazí zákazníkovi po kliknutí na příslušný produkt z obrázku 3.3. Uživatel si zvolí počet kusů a skleník, ve kterém se produkt nachází. Poté, co přidá položku do košíku, bude přesměrován.



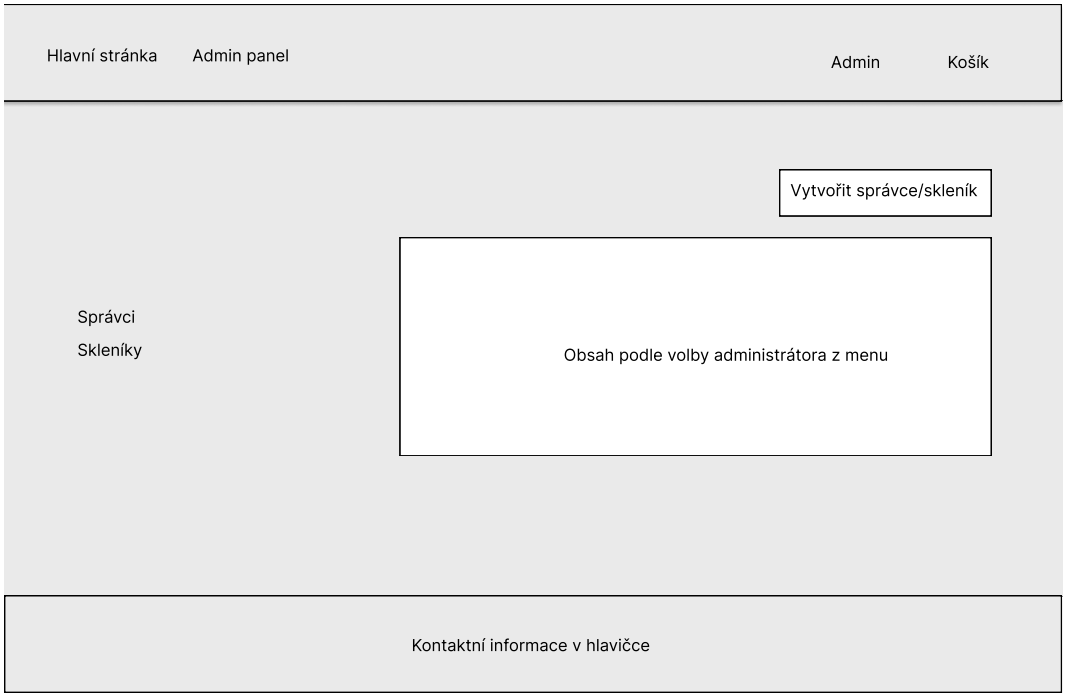
Obrázek 3.5: Košík produktů, které si uživatel zvolil. Uživatel má možnost tyto položky ze svého košíku odstranit, případně objednávku vyřídit.



Obrázek 3.6: Profil uživatele, ke kterému má přístup v případě, že se registroval. Může si zobrazit své objednávky a zprávy. Zároveň má možnost změnit své údaje v panelu uživatelské informace. Hlavička se dynamicky mění, pokud je uživatel přihlášen.

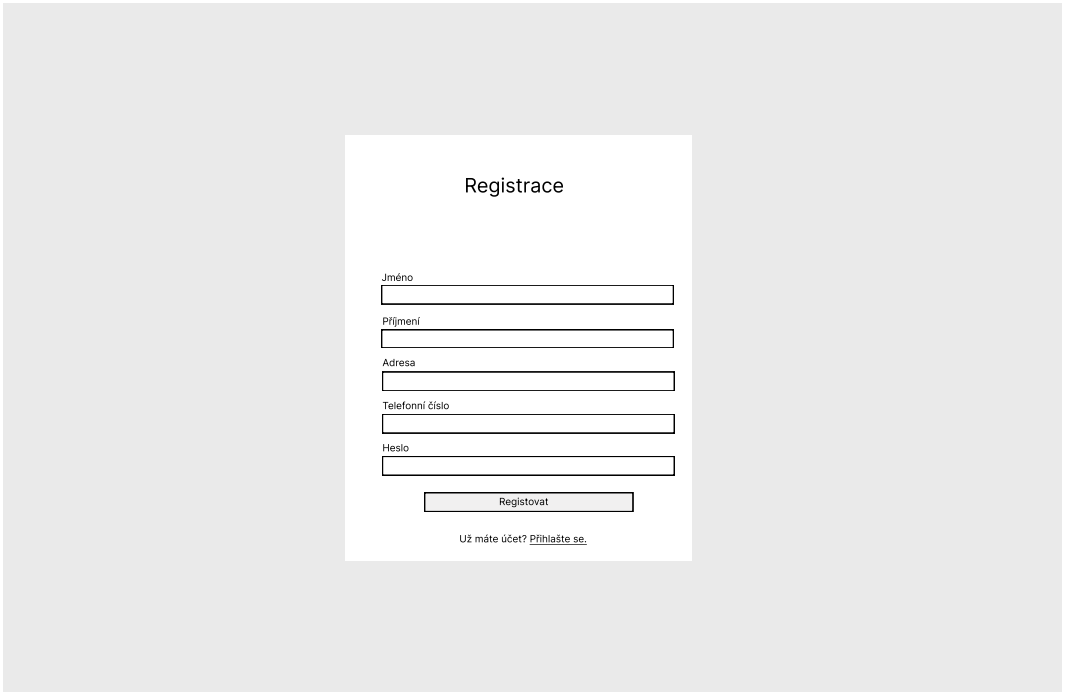


Obrázek 3.7: Stránka, která zobrazuje možnosti správce v rámci aplikace. V případě, že má uživatel oprávnění správce, je mu v hlavičce zpřístupněn odkaz na panel správce, kde si může zobrazit své revize nebo je případně vytvořit. Stejně tak si může zobrazit a vytvořit nové produkty.



Obrázek 3.8: Stránka, která zobrazuje možnosti administrátora v rámci aplikace. Administrátor má možnost vytvářet nové skleníky a registrovat nové správce.

Obrázek 3.9: Stránka pro přihlášení uživatele.



Obrázek 3.10: Návrh formuláře pro registraci uživatele.

Testování drátěného modelu

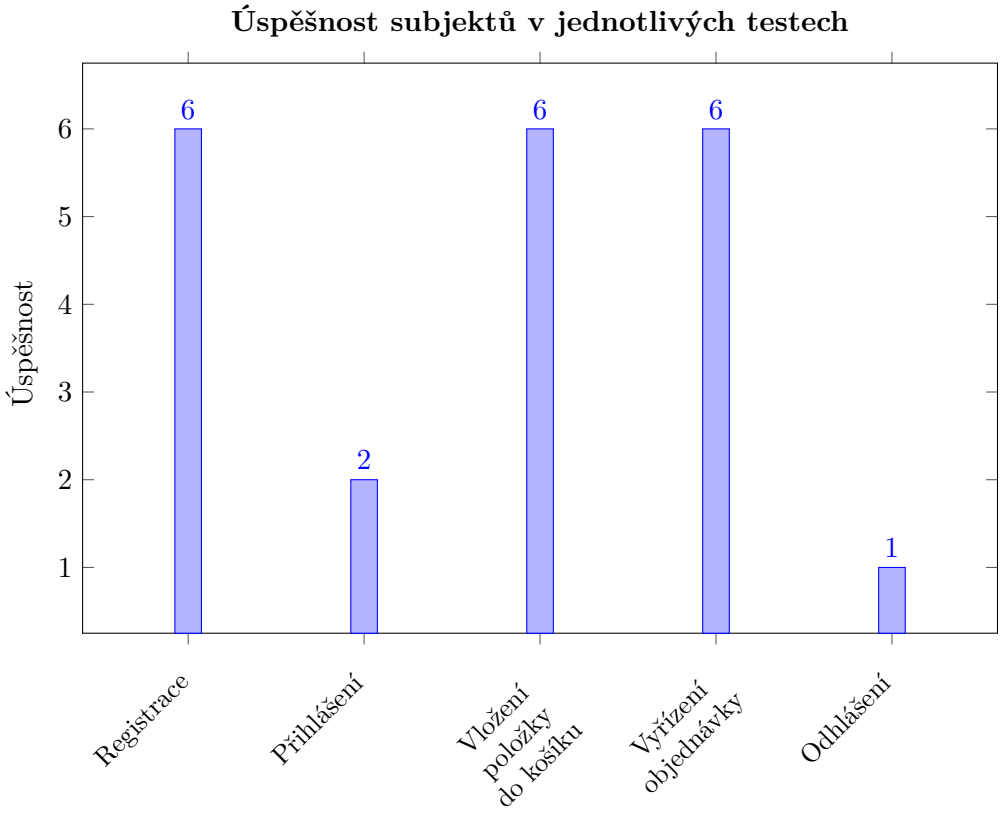
Návrh byl testován uživateli následujících věkových kategorií:

- Mladiství
- Dospělí
- Senioři

Každá kategorie obsahovala dva uživatele. Ti měli za úkol popsat proces, který simuluje zákazníkovu činnost v rámci aplikace. Každý testovaný měl tedy podle drátěného návrhu popsat, jak provede tyto činnosti:

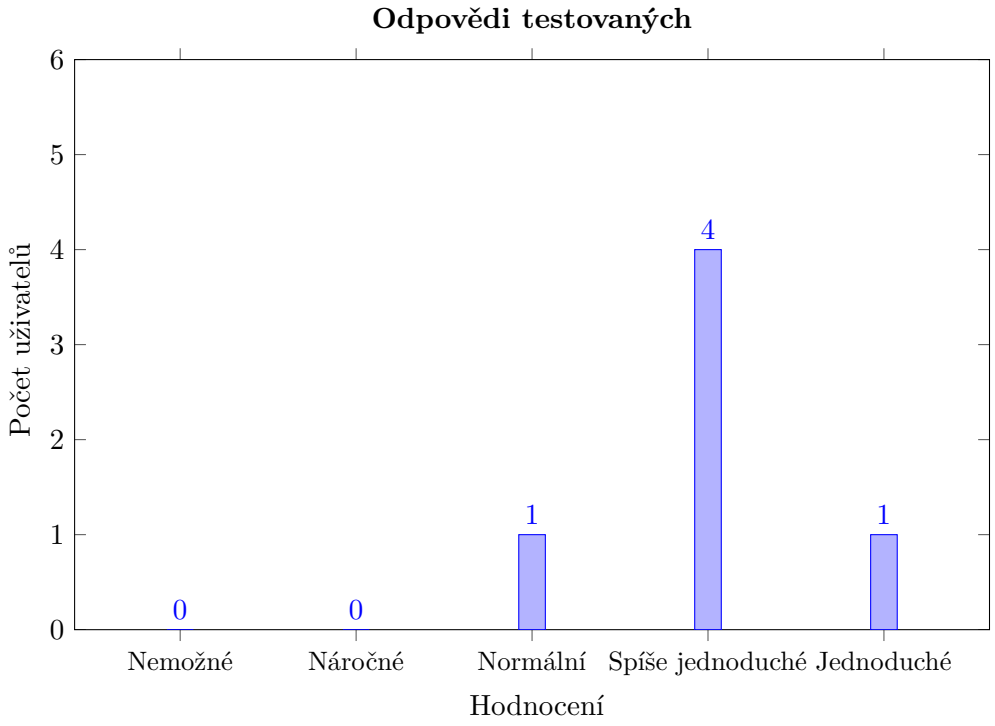
- Registrace
- Přihlášení
- Vložení položky do košíku
- Vyřízení objednávky
- Odhlášení

Výsledky byly převážně pozitivní. Většina subjektů si jednoduše poradila s kompletním průběhem objednávky tj. vložení do košíku a následné zaplacení. Prvním problémem, s kterým se subjekty potýkaly, byla nutnost přihlášení i po registraci. Většina z nich automaticky po registraci přešla hned k nákupu položek. Dalším kamenem úrazu bylo odhlášení. Testovaní měli problém zjistit, jak se k možnosti odhlásit dostat.



Zpětná vazba

Po testování byli uživatelé požádáni o vyplnění krátkého dotazníku, jenž měl zjistit jejich celkovou spokojenost nebo případně odhalit jejich připomínky. Uživatelé z kategorie senioři a dospělí vyplnili, že by očekávali automatické přihlášení po registraci. Zároveň zmínili, že volba skleníku, kterou znázorňuje obrázek 3.5, působí nepřirozeně. Podle jejich představy by měl mít zákazník možnost si zvolit skleník, který je například blízko jeho lokace, a až poté si zobrazit příslušné produkty, které jsou v tomto skleníku k dispozici. Testovaní všech kategorií pak následně vyplnili, že možnost odhlášení je poměrně skrytá a neintuitivně umístěná. V závěru byli uživatelé dotázáni, jak obtížné bylo tyto úkony vykonat.



Po zjištění nových poznatků bylo možné pokračovat v další iteraci návrhu. Ten zohledňuje zpětnou vazbu uživatelů a zároveň poskytuje grafickou podobu samotné aplikace.

3.3.2 Mockup

Mockup je statický vizuální návrh webové stránky nebo aplikace, který simuluje její finální designovou podobu. Neobsahuje funkcionalitu ani interaktivitu. Většinou se tvoří ve speciálních grafických programech jako Glorify nebo Adobe XD. Mockup se využívá až v pozdějších částech vývoje. Je tedy nutné mít před tím stanovenou samotnou funkcionalitu aplikace. Nejedná se však o přesnou podobu finálního produktu. Často zde využíváme provizorní text a obrázky, které nutně nemusí souviset s celkovým konceptem aplikace [6]. Správný mockup by tedy měl obsahovat:

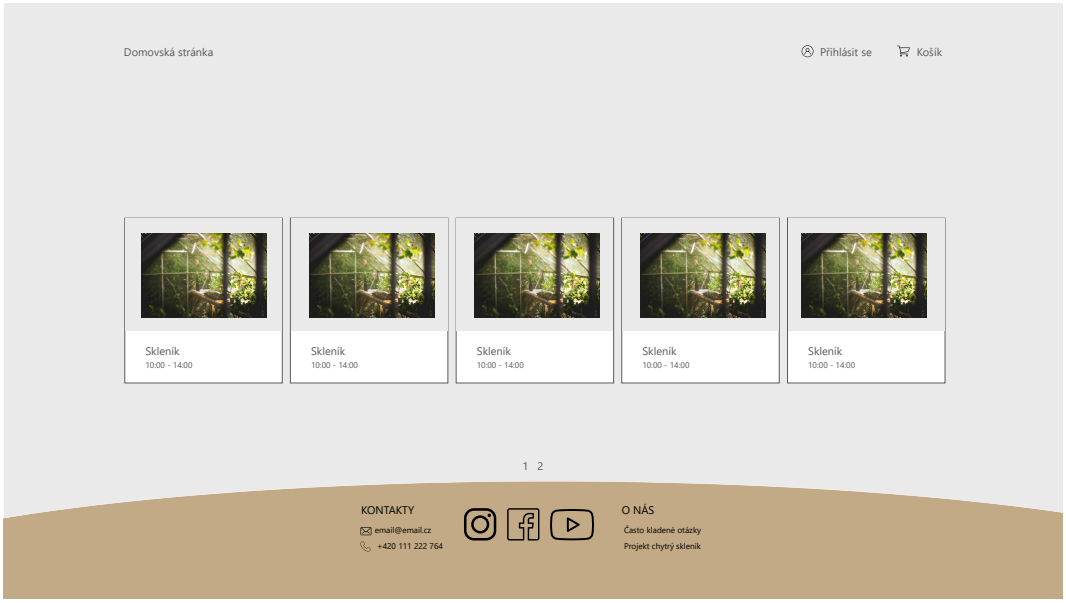
- Rozložení stránky - stejně jako u drátěného modelu. Zobrazuje pozici jednotlivých elementů aplikace.
- Vizuální prvky - tabulky, tlačítka, formuláře a jejich grafické zpracování.
- Typografie a barvy - zvolení adekvátního fontu a jeho velikosti. Přívětivá barevná kombinace elementů, které se na stránce vyskytují.
- Uživatelská interakce - popis interaktivních elementů.

Testování návrhu

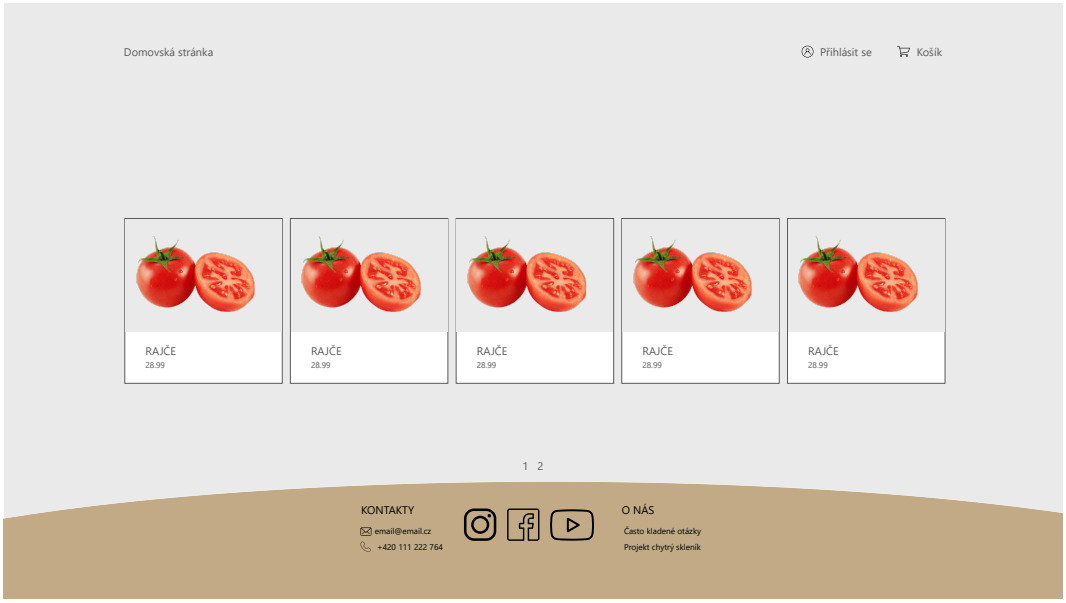
Jelikož se v projektu vyskytují i správci, kteří se starají o technický aspekt aplikace, bylo nutné provést testování i v tomto sektoru. Každý testující uživatel měl vykonat následující úkoly:

- Přejít do panelu správce
- Zobrazit si produkty v různých sklenících
- Vytvořit nový produkt
- Upravit existující revizi
- Označit revizi za hotovu

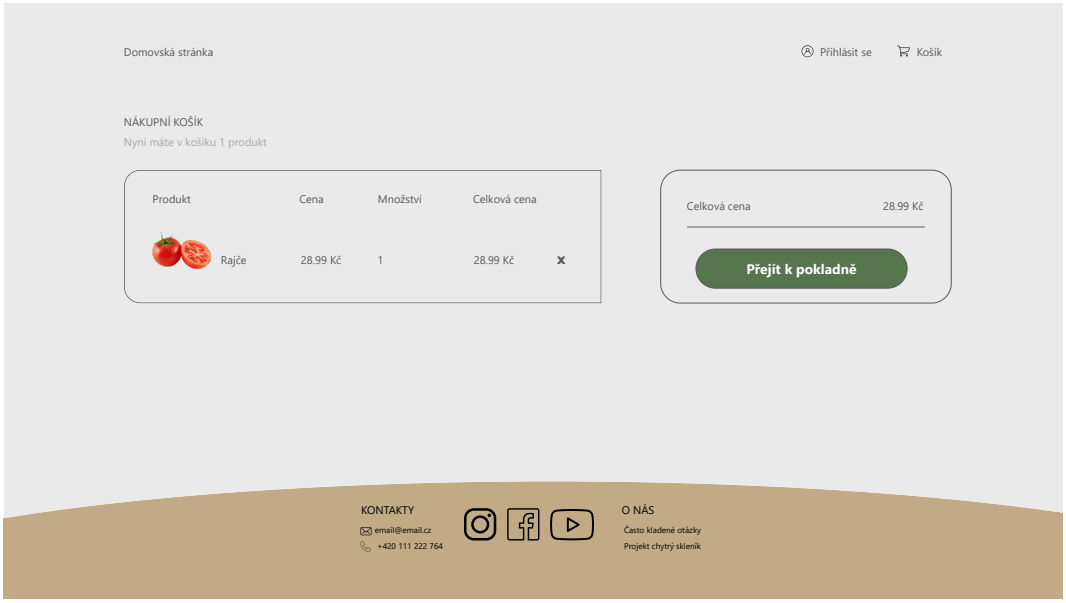
Všem pěti testujícím se podařilo úspěšně provést tyto činnosti. Celý proces označili za intuitivní a lehce vykonatelný. Grafické zpracování označili za vhodné k dané tématice a přehledné. Celkově tedy testování proběhlo úspěšně a mohlo se přejít k dalším krokům.



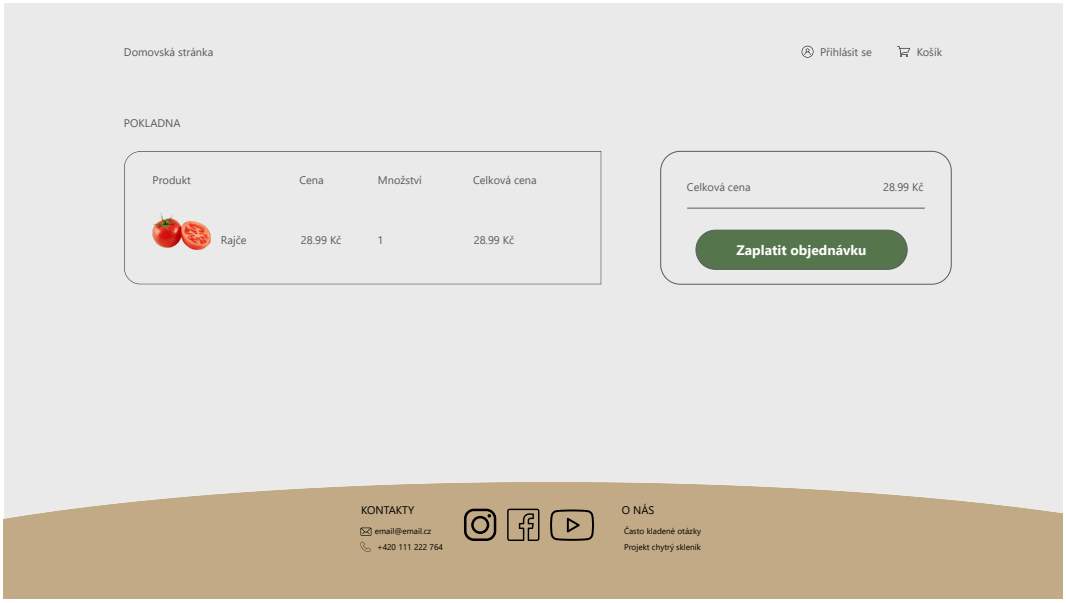
Obrázek 3.11: Stránka, která zobrazuje dostupné skleníky v rámci aplikace.



Obrázek 3.12: Seznam produktů, které jsou dostupné v rámci skleníku, který si uživatel zvolil.



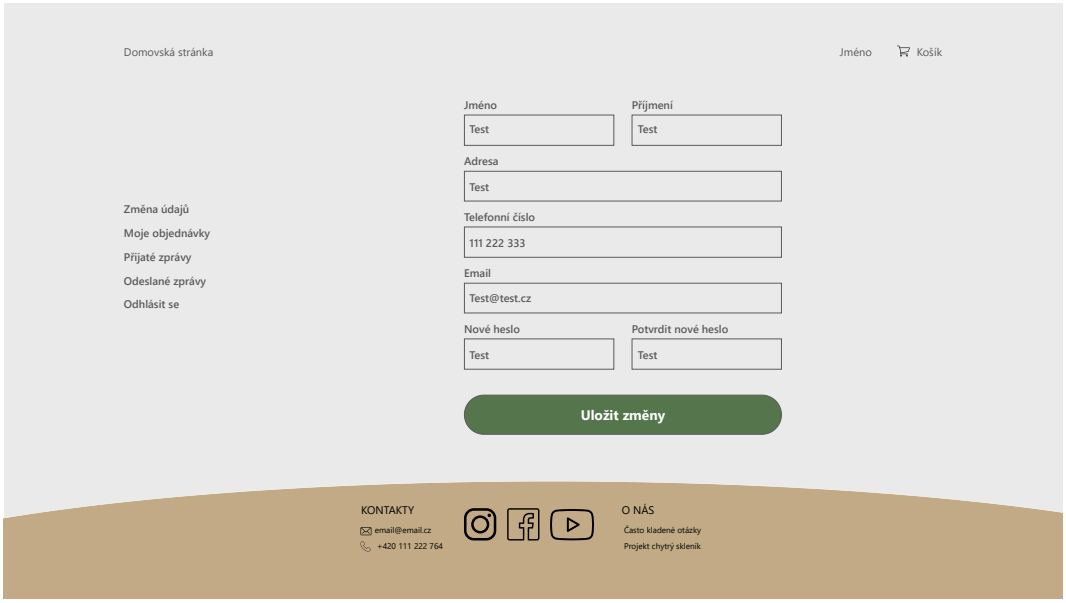
Obrázek 3.13: Košík se zvolenými produkty. Uživatel je sem automaticky přesměrován po každé, co si nějakou položku vybere. Má také možnost jednotlivé položky odstranit.



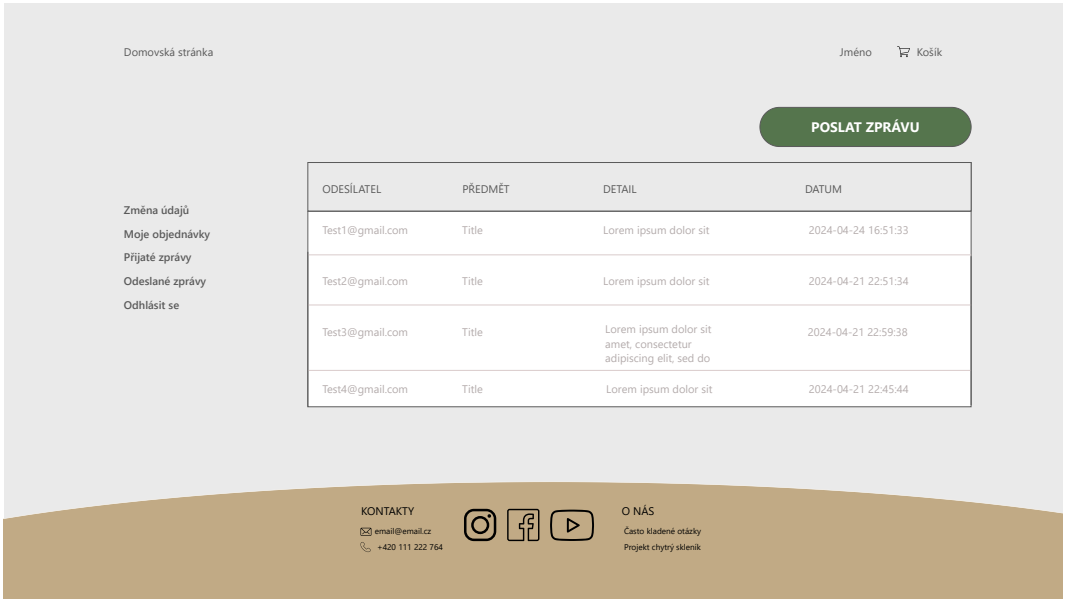
Obrázek 3.14: Stránka s přehledem objednaných produktů. V této fázi uživatel už s produkty nemůže manipulovat. V případě, že klikne na zaplacení, je přesměrován na platební bránu.

Obrázek 3.15: Pohled, který je uživatelovi zobrazen v případě úspešné platby.

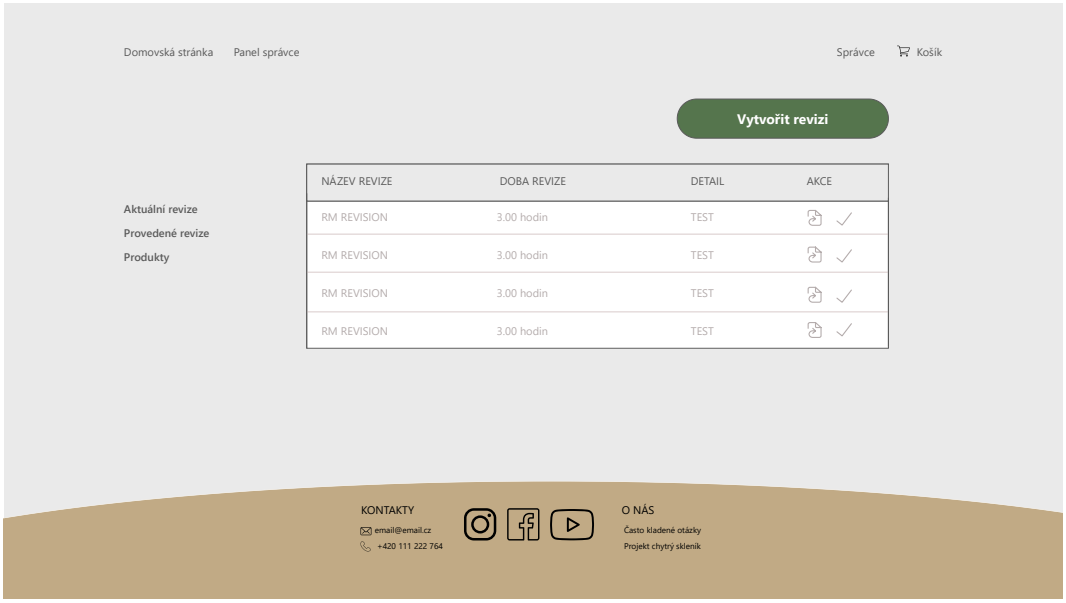
Obrázek 3.16: Detail objednávek příslušného uživatele. Pro tuto možnost je nutné, aby byl uživatel přihlášen. V hlavičce se pak zobrazí jeho jméno. V případě, že přejede přes toto jméno myší, zobrazí se mu možnost přejít na profil, zobrazit objednávky a odhlásit se.



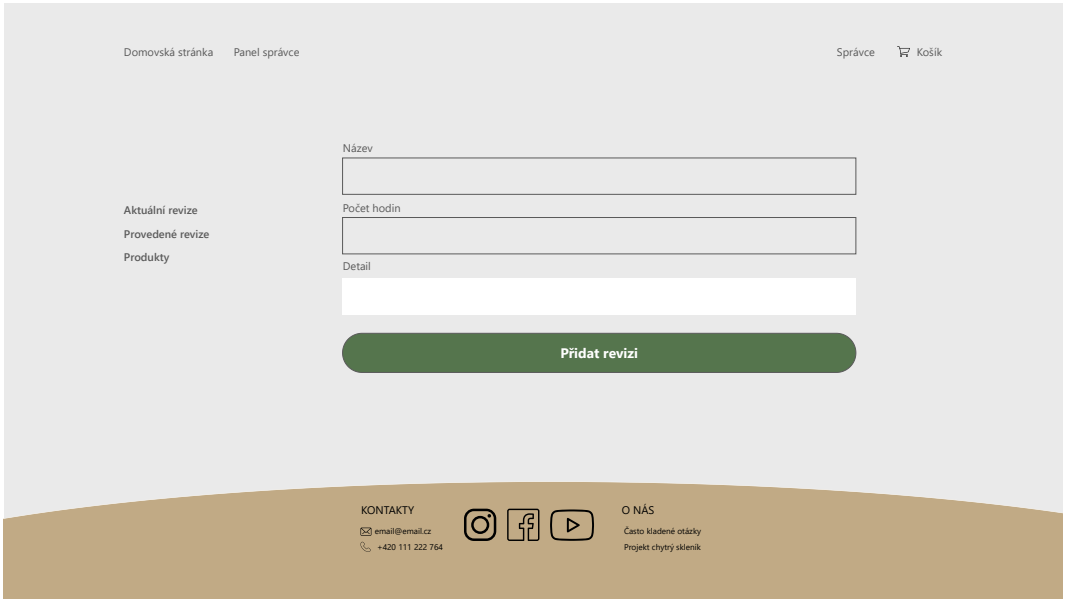
Obrázek 3.17: Formulář s uživatelskými údaji, které jsou změnitelné. V případě, že si uživatel chce změnit nějaký z údajů, stačí jej pouze přepsat a uložit změny. Pokud uživatel nechce změnit heslo, nechává políčka prázdné.



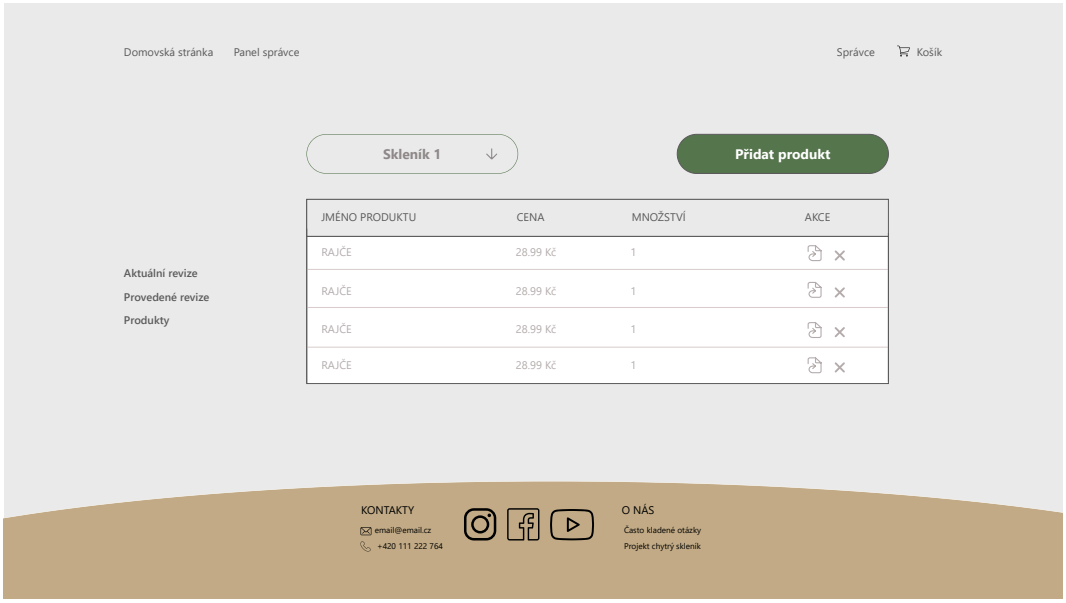
Obrázek 3.18: Tabulka, ve které si uživatel může zobrazit přijaté zprávy. Tato stránka je identická jako odeslané zprávy, jen se liší obsaženými informacemi.



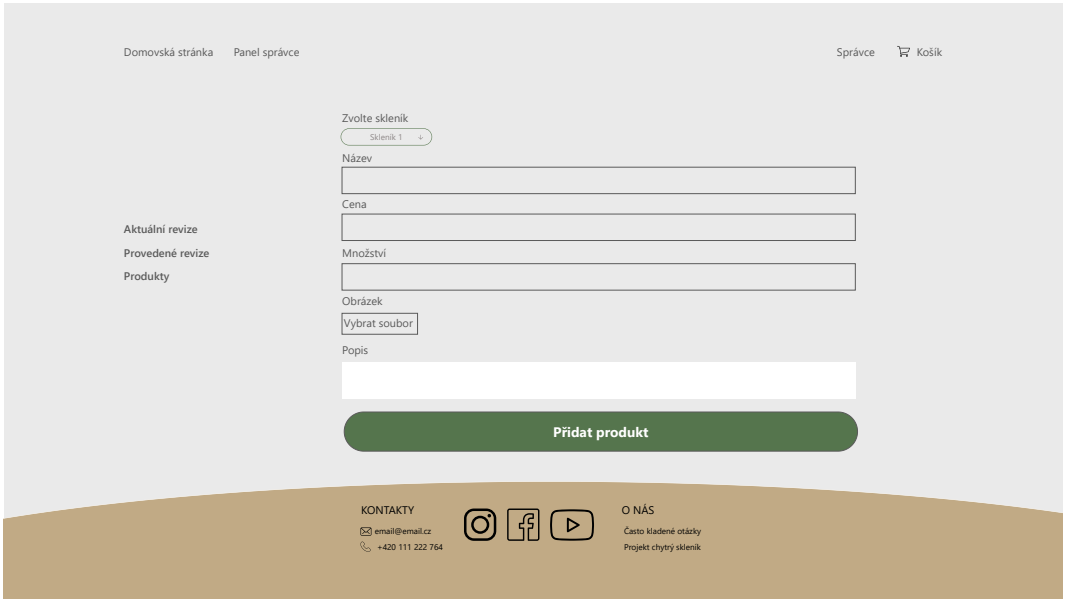
Obrázek 3.19: Tabulka, ve které si správce může zobrazit své aktuální revize. Tyto revize může v sekci akce upravovat, případně označit za hotové.



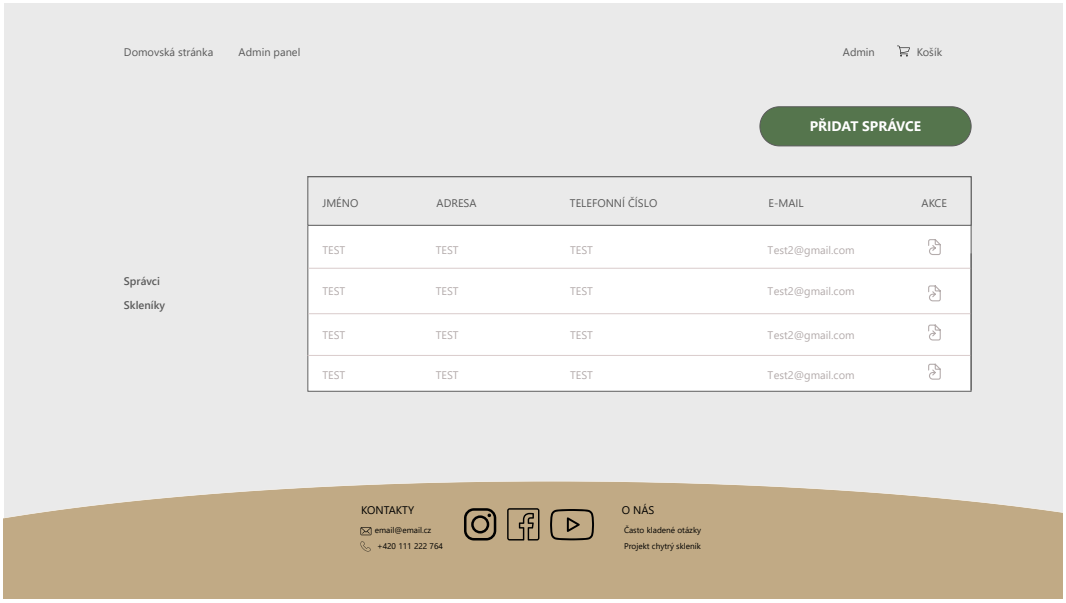
Obrázek 3.20: Formulář, který slouží k vytvoření revize.



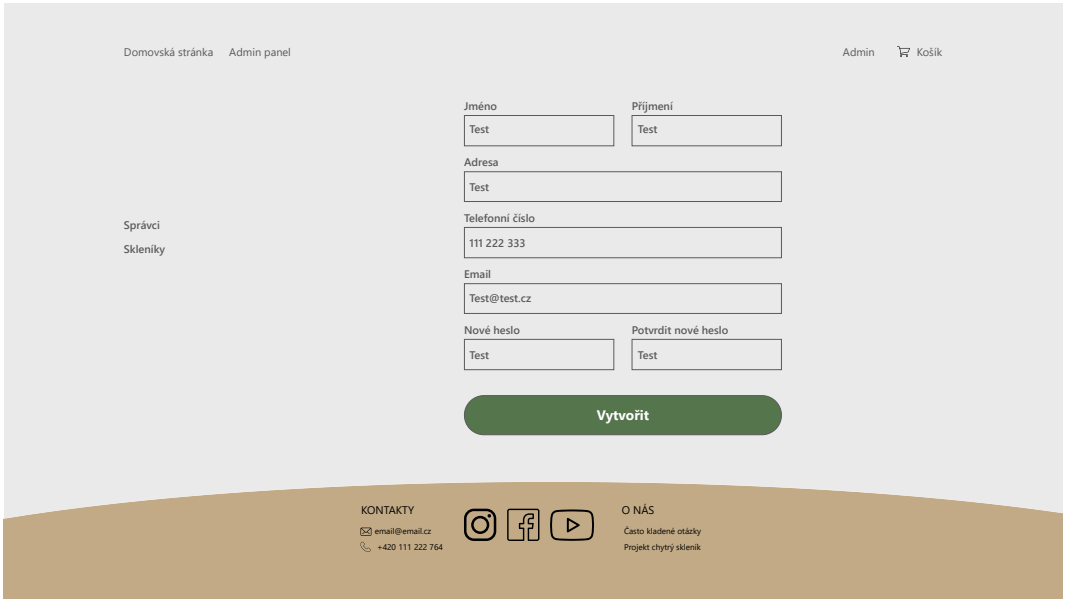
Obrázek 3.21: Tabulka, v niž si může správce zobrazit produkty na základě volby skleníku.



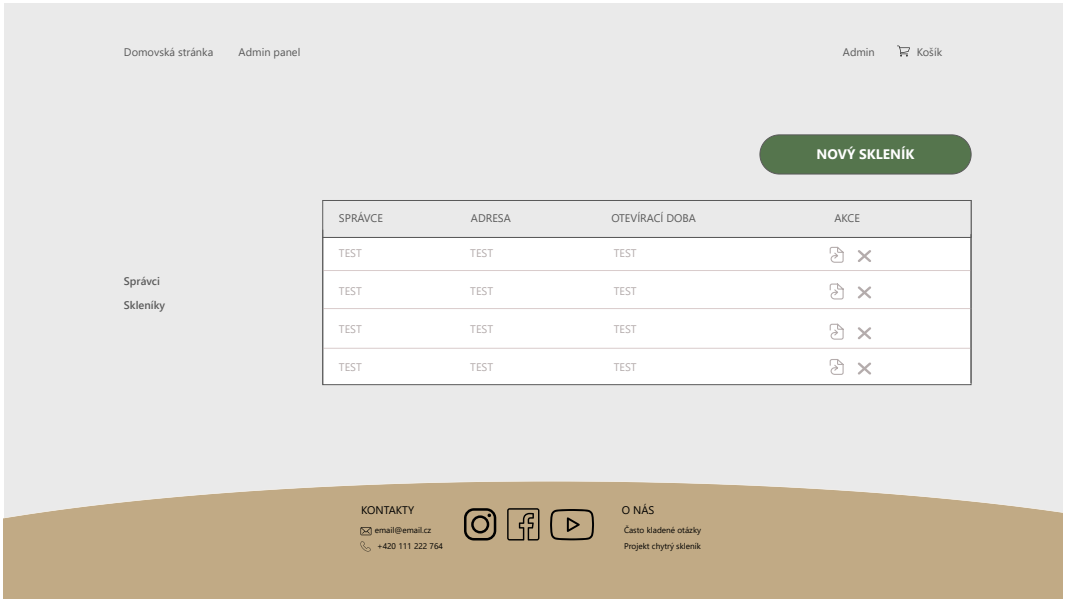
Obrázek 3.22: Formulář, který slouží k vytvoření produktu. Správce mimo zadání informací musí také nahrát obrázek, který bude produkt reprezentovat.



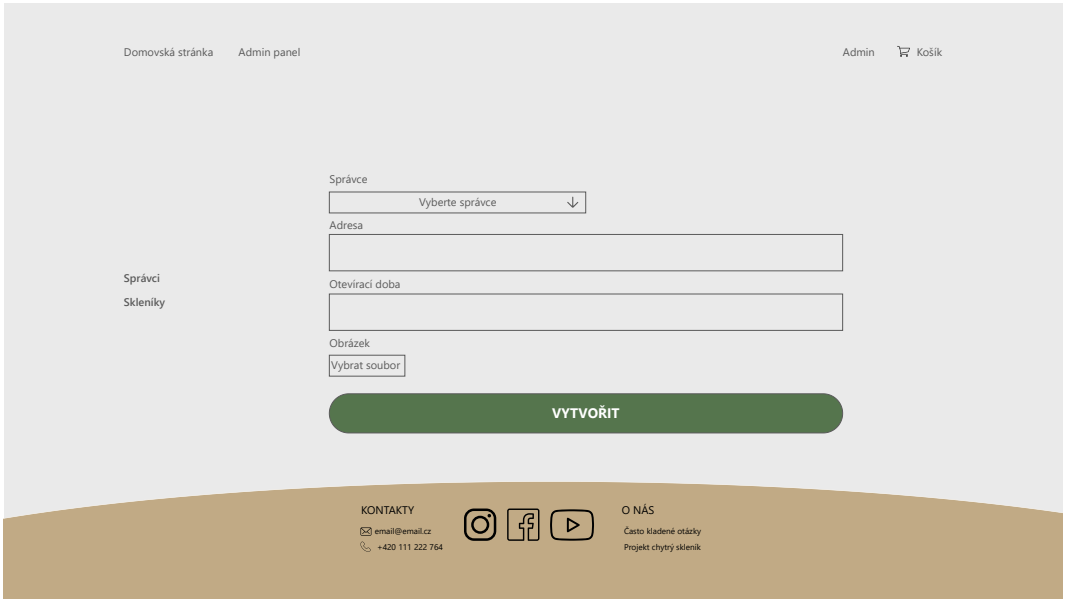
Obrázek 3.23: Přehled existujících správců, které si může administrátor zobrazit a případně jejich profil upravit.



Obrázek 3.24: Formulář pro vytvoření správce. Je totožný s formulářem, přes který se registrují normální zákaznici. Pokud jej však odesílá administrátor z tohoto pohledu, dochází k vytvoření správce nikoliv klasického uživatele.



Obrázek 3.25: Přehled jednotlivých skleníků a možnost jejich úpravy, případně vymazání.



Obrázek 3.26: Formulář pro vytvoření nového skleníku. Administrátor musí zvolit správce, který bude mít příslušný skleník na starost a zároveň nahrát obrázek skleníku, který bude zákazníkům zobrazen.

Kapitola 4

Implementace

V této kapitole je popsán samotný postup řešení. Jsou zde zmíněny knihovny a funkce, které byly využity k implementaci. Kapitola také popisuje integraci platební brány `PayPal` do aplikace.

4.1 Django

4.1.1 Instalace

Před samotnou instalací tohoto frameworku je nezbytným krokem vytvoření virtuálního prostředí. Virtuální prostředí je izolované prostředí, které umožňuje instalaci balíčků a knihoven specifických pro daný projekt. Hlavním důvodem, proč využívat virtuální prostředí je možnost izolace závislostí. V případě, že více projektů využívá stejný balíček, který není nainstalován pouze v rámci virtuálního prostředí, může dojít k nekonzistencím.

Po instalaci `Django` frameworku je nutné vytvoření samotného projektu. Toho můžeme docílit pomocí příkazu `django-admin startproject project_name`. Po spuštění se nám vytvoří pracovní adresář s názvem projektu, v tomto případě `coreApp`, který obsahuje následující soubory:

- `__init__.py` - speciální soubor, který `Pythonu` říká, aby zacházel s celým adresářem jako balíček.
- `wsgi.py` - jedná se o rozhraní, které naslouchá mezi serverem a klientem. Slouží k obsluze webových požadavků.
- `asgi.py` - podobné jako `wsgi.py`, akorát zpracovává asynchroní požadavky.
- `settings.py` - soubor, který obsahuje nastavení celého projektu. Je možné zde registrovat další aplikace, specifikovat ukládání statických souborů, povolit `debuggování` a další.
- `urls.py` - mapuje URL adresy na jednotlivé pohledy. Často se využívá funkce `include`, která umožňuje zahrnout pohledy z různých aplikací.

Krom tohoto adresáře dochází také k vytvoření souboru `manage.py`. Ten umožňuje propojení s příkazovou řádkou. Je možné tedy provádět příkazy spojené s databází či samotné spuštění serveru.

Dalším krokem je vytvoření `Django` aplikace. Ta umožňuje samotné zobrazení webové stránky. Jeden projekt může obsahovat několik aplikací, všechny však musí být registrované

v souboru `settings.py`. V rámci tohoto projektu však byla vytvořena aplikace pouze jedna a to pomocí příkazu `django-admin startapp Api`, kde `Api` je jméno aplikace. Po provedení tohoto příkazu dojde k vytvoření stejnojmenného adresáře, který obsabuje:

- `__init__.py`
- `admin.py` - slouží k registrovaní modelů z databáze.
- `apps.py` - konfigurační soubor, který umožňuje modifikovat chování aplikace.
- `models.py` - popis datového modelu aplikace.
- `tests.py` - obsahuje jednotkové testy.
- `urls.py` - umožňuje mapování jednotlivých URL na `views` v rámci aplikace.
- `views.py` - definuje jednotlivé pohledy, které obsluhují `HTTP` požadavky.

V adresáři byl na víc vytvořen soubor `serializer.py`. Ten umožňuje serializaci dat do `JSON` formy, jež je nezbytná pro přenos dat pomocí `API`[1]. Zároveň přijatá data umí deserializovat do původní formy. Jelikož specifikace těchto formátů může být poměrně náročná, byl využit `Django REST framework`[2], který celý tento proces ulehčuje.

`Django REST framework` je flexibilní nástroj, který slouží k budování webových `APIs`. Obsahuje knihovny, které ulehčují úlohy jako je autentizace a serializace. Zároveň disponuje rozsáhlou dokumentací, která detailně popisuje jednotlivé balíčky a jejich možnosti.

4.1.2 Autentizace

Jako autentizační systém byl využit `Simple JSON Web Token`[3]. Jedná se o modul, který je určen právě pro `REST framework`. Vývojáři si jej mohou sami přizpůsobit a nakonfigurovat podle potřeby aplikace. Rozlišujeme dva druhy `tokenů`:

- Přístupový - slouží k identifikaci uživatele, jeho expirace bývá zpravidla kratší.
- Obnovovací - slouží k obnovení přístupového tokenu. Jeho platnost je výrazně delší.

V případě, že je zavolán `endpoint` pro obnovu `tokenů`, dojde k vytvoření nového přístupového, ale i obnovovacího `tokenu`. Tato skutečnost může být nebezpečná, jelikož starý obnovovací `token` může stále tyto údaje měnit. Proto je nezbytné neplatné obnovovací `tokeny` umístit na černou listinu.

Často je však potřeba z `tokenu` získat další informace, jako uživatelské jméno, případně e-mail. Toho je možné docílit dvěma způsoby.

[1] https://www.ibm.com/topics/api
[2] https://www.django-rest-framework.org/
[3] https://django-rest-framework-simplejwt.readthedocs.io/en/latest/

První možnost je zakódovat tyto informace do samotného `tokenu`. Jedná se o běžný postup, který se často u `JWT` využívá. Nevýhodou však je nutnost dešifrovaní tokenu, abychom se ve frontendové části k datům dostali.

```
1   import TokenObtainPairSerializer
2
3   class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
4   @classmethod
5   def get_token(cls, user):
6       token = super().get_token(user)
7       #add email information to token
8       token['email'] = user.email
9
10      return token
```

Výpis 4.1: Příklad, jak do JWT tokenu zakódovat další informace.

Druhou možností je modifikovat třídu `TokenObtainPairSerializer`, konkrétně funkci `validate`, která vrací příslušná data.

```
1   import TokenObtainPairSerializer
2
3   class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
4   @classmethod
5   def validate(self, attrs):
6       data = super().validate(attrs)
7       data['refresh'] = str(refresh)
8       data['access'] = str(refresh.access_token)
9       #append additional information to JWT
10      data['email'] = self.user.email
11      data['address'] = self.user.address
12
13      return data
```

Výpis 4.2: Modifikace funkce validate, která standardně vrací pouze tokeny.

Pro potřeby aplikace byla zvolena druhá možnost. V případě, že se uživatel přihlásí, je společně s `tokeny` poslána řada dat, které není nutné dešifrovat. Aplikace zatím využívá pouze přístupový `token`, jehož platnost je stanovena na 7 dní. Po každé, co se uživatel přihlásí, je zavolán `endpoint`, jenž vrací přístupový `token` společně s informacemi o uživatelském účtu.

Registraci zajišťuje funkce `registerUser`. V ni se rozlišuje, zda se jedná o běžného uživatele nebo správce, který je registrován administrátorem. V případě, že se jedná o správce, je zasílán na tento `endpoint` parametr `is_staff`, jenž je nastaven na hodnotu `True`. Pro registrací běžného uživatele musí být splněny následující podmínky:

- Vytvoření uživatelské účtu s příslušnými údaji.
- Automatické přihlášení po registraci.
- Možnost přístupu ke stránkám, které vyžaduji autentizaci.

Pro tyto potřeby je v aplikaci využit dočasný přístupový `token` s název `registerToken`, který je serializován s ostatními daty. Nejedná se o atribut modelu uživatele, je pouze přidán pomocí `SerializerMethodField`[4] do těla odpovědi, která reaguje na registrační požadavek.

`Django` samo nabízí svůj výchozí model uživatele, který je dostupný po importování balíčku `django.contrib.auth`[5]. Tento model je však pro potřeby aplikace nedostatečný, jelikož neumožňuje ukládání telefonních čísel, adres apod. Proto bylo nutné jej nahradit vlastním modelem, jenž rozšiřuje model `AbstractUser`[6]. Pozměněno bylo také přihlašovaní, kde bylo uživatelské jméno nahrazeno e-mailem.

4.1.3 Objednávky

Každý uživatel aplikace má možnost vytvořit objednávku, která se skládá z jednotlivých produktů, jež nazýváme položky objednávky. Krom samotného vytvoření si může uživatel své jednotlivé objednávky zobrazit. Na obrázku 3.2 si lze povšimnout poměrně složitých vztahů tohoto modelu. V rámci implementace bylo tedy nutné tento proces rozdělit do tří na sobě závislých kroků.

- Vytvoření instance objednávky včetně vztahu s entitou `Uživatel` a nastavení atributu `Celková cena`.
- Vytvoření jednotlivých položek objednávky, které jsou spojeny vztahy s objednávkou a příslušným produktem, a zadání jejich atributů.
- Upravení celkového množství produktu a jeho následné uložení.

[4] https://www.django-rest-framework.org/api-guide/fields/#serializermethodfield
[5] https://docs.djangoproject.com/en/5.0/ref/contrib/auth/#
[6] https://docs.djangoproject.com/en/5.0/topics/auth/customizing/

V závislosti na těchto krocích byla implementovaná funkce `createOrder`, která se z frontendové částí volá v momentě, kdy dojde ke korektnímu zaplacení zákazníkem.

```
1   @api_view(['POST'])
2   def createOrder(request):
3       #get data from frontend
4       data = request.data
5       orderItems = data["orderItems"]
6       # set user relationship
7       user = MyUser.objects.get(email=data["user"])
8       # create order
9       order = Order.objects.create(
10              user = user,
11              totalPrice = data["totalPrice"]
12      )
13      # create order items and set the relationship
14      for item in orderItems:
15          product = Product.objects.get(id=item["data"]["id"])
16          temp = OrderItem.objects.create(
17              product=product,
18              order=order,
19              name = product.name,
20              amount = item["amount"],
21              price = item["data"]["price"],
22          )
23              # change product inStock
24              product.inStock -= int(temp.amount)
25              product.save()
26      # serialize the response and send it to frontend
27      serializer = OrderSerializer(order, many=False)
28      return Response(serializer.data)
```

Výpis 4.3: Princip fungování funkce createOrder

Aplikace má také implementovanou funkci pro zobrazení všech objednávek, jež měla sloužit správcům pro vyřizování jednotlivých objednávek. Nicméně po konzultaci s vedoucím práce byl vyvozen závěr, že správce, který by tyto objednávky musel v systému vyřizovat, by byl velmi nákladný a v této fázi projektu by tak tyto výdaje byly příliš vysoké. Zákazníci si tedy budou moci po zaplacení sami objednávku vyzvednout.

Mimo jiné každý skleník obsahuje kamery, které by tak eliminovaly možné pochybné chování určitých zákazníků.

4.1.4 Administrace

Jak již bylo zmíněno v aplikaci se vyskytují tři typy uživatelů, kde každý z nich má jiné pravomoci. Každý přihlášený uživatel má možnost zasílat a přijímat zprávy. K odeslání stačí pouze e-mail doručovatele. Uživatel je schopen si ve svém profilu změnit údaje a to i včetně hesla.

Změnu hesla zajišťuje funkce `make_password`, která je importována z modulu `django.contrib.auth.hashers`[7]. Tato funkce přijímá heslo v prosté textové formě a převádí je na `hash`, jenž je následně uložen do databáze.

V případě, že uživatel disponuje oprávněním správce má možnost manipulovat s produkty v databázi. K tomu slouží funkce `createProduct`, `updateProduct` a `deleteProduct`. Tyto funkce jsou obohaceny o dekorátor `@permission_classes([IsAdminUser])`[8], jenž zajišťuje, že k těmto funkcím má přístup pouze uživatel s nastavením proměnné `is_staff` na hodnotu `True`. Správce má taky možnost vytvářet a upravovat své revize. Funkce pro tyto potřeby jsou opět zpřístupněny pouze jemu. Důležitým faktorem je atribut `isCompleted`, který symbolizuje, zda se jedná o již vyřízenou revizi nebo aktuální.

V neposlední řadě je v systému taky administrátor. Ten registruje jednotlivé správce a má pravomoci měnit jejich údaje, případně status správce odebrat. Nemůže však účet úplně vymazat. Registrace správců i uživatelů se nachází na stejném koncovém bodě, ale odlišuje se proměnnou `is_staff`. Administrátor také vytváří a upravuje jednotlivé skleníky, k nimž je povinen přiřadit jednoho z registrovaných správců. Jelikož model `User` v `Djangu` standardně nabízí pouze proměnnou `is_staff` pro určení role, bylo klíčové vytvoření další proměnné, jež označuje status administrátora. Tato proměnná byla označena jako `isAdmin`. V projektu bylo stanoveno, že administrátor bude pouze jeden. Pro každého nového uživatele je tedy tato proměnná nastavena na výchozí hodnotu `False`. Administrátora tedy nelze registrovat. Jeho účet musí tedy být v databázi již před spuštěním aplikace nebo lze případně upravit standardní účet v `Django admin` panelu, kde se dá změnit hodnota atributu `isAdmin`.

[7] https://docs.djangoproject.com/en/2.0/_modules/django/contrib/auth/hashers/
[8] https://www.django-rest-framework.org/api-guide/permissions/

4.1.5 Databáze

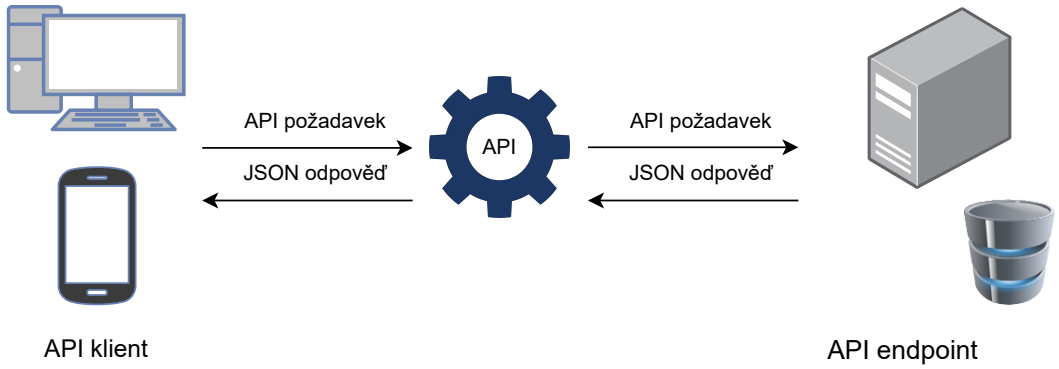
Pro testovací účely bylo využito `PostGreSQL` relační databáze, která je volbou většinou vývojářů pracující s frameworkem `Django`. `Django` využívá mapovač `ORM`, který umožňuje abstrakci spojení mezi tabulkou databáze a modelem v `Djangu` [11]. Vývojáři tedy mohou kódovat `SQL` příkazy v podobě `Python` kódu.

Pro potřeby aplikace a demonstraci použití práce je však využití `PostGreSQL` nevhodné. Databáze by totiž musela mít zaplacený `hosting`, který je poměrně finančně náročný a v testovací fázi zbytečný. Z těchto důvodu byl zvolen výchozí databázový systém, který je již součástí `Django` frameworku, `SQLite`. Jedná se o vestavěný databázový systém, který ukládá data do jednoho souboru na disku.

V případě budoucího vývoje však není problém přejít na databázi `PostGreSQL`. Jediné co stačí změnit je nastavení v konfiguračním souboru `settings.py`.

4.1.6 Koncové body API

Koncový bod je digitální poloha, kam se zasílají požadavky ze strany klienta, které požadují určité zdroje na serverové straně [8]. `Endpoint` je typicky reprezentován jako URL, které označuje určitou polohu na serveru. Aplikace založena na `REST API` má takových bodů několik, kde každý z nich zpravidla označuje nějakou entitu.



Obrázek 4.1: Diagram znázorňující komunikaci klient-server pomocí API.

Každý požadavek, který je směřován na příslušný `endpoint`, musí obsahovat metodu, jež specifikuje, jaká operace se s daty na straně serveru vykoná. Krom metody je součástí také hlavička, v niž jsou dodatečné informace jako typ posílaných dat případně autentizační údaje. V neposlední řadě pak samotné tělo, v němž jsou serializovaná data.

V momentě, kdy je požadavek doručen na příslušný endpoint, dojde k zavolání korespondujícího pohledu. Ten požadavek obslouží a vygeneruje odpověď, která je odeslána v `JSON` nebo `XML` formátu zpět klientovi.

Endpoint	Funkce	Popis
products/create	createProduct	Vytvoření produktu správcem
products/greenhouse/<str:pk>	getProducts	Zobrazí produkty vytvořené správcem
products/<str:pk>	getProduct	Zobrazí detail konkrétního produktu
products/<str:pk>/update	updateProduct	Upravení atributů produktu
products/<str:pk>/delete	deleteProduct	Vymazání produktů z databáze

Tabulka 4.1: Koncové body pracující s modelem `Product`

Endpoint	Funkce	Popis
greenhouses/	getGreenhouses	Zobrazí existující skleníky
greenhouses/create	createGreenhouse	Vytvoření skleníku administrátorem
greenhouses/<str:pk>/update	updateGreenhouse	Upravení atributů skleníku
greenhouses/<str:pk>/delete	deleteGreenhouse	Vymazání skleníku z databáze

Tabulka 4.2: Koncové body pracující s modelem `Greenhouse`

Endpoint	Funkce	Popis
orders/	getOrders	Zobrazí objednávky přihlášeného uživatele
orders/create	createOrder	Vytvoření objednávky, provedeno automaticky po zaplacení

Tabulka 4.3: Koncové body pracující s modelem `Order`

Endpoint	Funkce	Popis
users/	getManagers	Zobrazí všechny správce v systému
users/<str:pk>	getManager	Zobrazí informace o správci
users/login	MyTokenObtainPairView	Modifikuje třídu `Simple JWT` sloužící pro login
users/register	registerUser	Registrace uživatele, případně správce
users/profile	getUser	Zobrazí informace přihlášeného uživatele
users/profile/update	updateUser	Aktualizace údajů, přihlášeného uživatele
users/<str:pk>/update	updateManager	Aktualizace údajů správce, které provádí administrátor

Tabulka 4.4: Koncové body pracující s modelem `MyUser`

Endpoint	Funkce	Popis
revisions/	getRevisions	Zobrazí revize vytvořené konkrétním správcem
revisions/<str:pk>	getRevision	Zobrazí detaily příslušné revize
revisions/create	createRevision	Vytvoření revize
revisions/<str:pk>/update	updateRevision	Změna informací v konkrétní revizi

Tabulka 4.5: Koncové body pracující s modelem `Revision`

Endpoint	Funkce	Popis
messages/	getMessages	Zobrazí doručené zprávy správcem
messages/sent	getSentMessages	Zobrazí odeslané zprávy
messages/create	createMessage	Vytvoření zprávy

Tabulka 4.6: Koncové body pracující s modelem `Revision`

4.2 React

4.2.1 Instalace

Pro vytvoření `React` projektu je možné využít nástroj `Create React App`. Abychom jej mohli použít, je nutné mít nainstalovaný `node.js`. Poté lze pomocí příkazu `npx create-react-app nazevProjektu` tento projekt vytvořit.

Po úspěšném vykonání příkazu dojde k vytvoření adresáře, jenž se v tomto případě nazývá `appfrontend`. Zároveň jsou v tomto adresáři vygenerované soubory nutné pro konfiguraci a samotnou implementaci aplikace. Mezi důležité soubory patří:

- `/src/index.js` - vstupní bod, který renderuje hlavní komponentu `App`.
- `/src/app.js` - základní komponenta, jež obsahuje směrovací logiku.
- `/src/index.css` - `CSS` soubor, jenž umožňuje stylizaci napříč projektem.
- `package.json` - soubor, který slouží pro správu projektu. Obsahuje závislosti, jméno projektu a umožňuje definovat proxy server.

V souboru `package.json` je specifikován proxy server, který reprezentuje adresu a port výše zmíněného `Django` serveru 4.1. Jelikož `Django` standardně běží na adrese `http://127.0.0.1:8000`, je sekce `proxy` nastavena právě na tuto hodnotu. Při posílání požadavků na server tak lze zadat jen příslušný koncový bod.

`React` disponuje řadou funkcionálních komponentů bez potřeby instalace dalších balíčků. Mezi nejdůležitější patří např. `useEffect` a `useState`, které se označují jako `React Hooks`. `Hook useEffect` umožňuje manipulaci se stránkami v závislosti na hodnotách zadaných proměnných. `Hook useState` zase uchovává stav proměnných a dokáže jej měnit.

Pro potřeby projektu je však `useState` nedostačující, protože je nutné uchovávat stavy globálně a mít k ním přístup i v jiných komponentech. Vzhledem k těmto faktům byla do projektu integrována knihovna `React Redux`.

4.2.2 React Redux

`Redux` je `open-source` knihovna, která slouží k správě stavu aplikace. Jeho hlavní výhodou je jeho škálovatelnost [2]. Umožňuje jednoduchý přístup k datům všem komponentům a jejich modifikaci pomocí tzv. `reducers` a `actions`. Redux tak napomáhá udržovat stav i komplexních aplikací předvídatelný. Autoří `Reduxu` Dan Abramov a Andrew Clark popsali předvídatelnost ve třech bodech:

- Veškerý stav aplikace je uchován na jednom místě.
- Stav by měl být pouze pro čtení. Jeho úpravy zprostředkovávají `actions`.
- Samotný vývojář specifikuje finální stav aplikace. Aby nedošlo k nekonzistencím, není stav modifikován, ale využívají se `reducers`, které určí hodnotu stavu v závislosti na příslušné akci.

Redux se skládá ze třech hlavních částí, které mezi sebou komunikují a modifikují jednotlivé stavy.

Actions

Jedná se o prosté JS objekty, které obsahují typ akce a samotné data, jež slouží ke změně stavu. Akce jsou vykonány pomocí metody `dispatch`[9] a následně odeslány do `Store`. Jedná se o jedinou možnost, jak poslat data z aplikace do `Redux store`, kde jsou uchovány. V rámci aplikace akce získávají data pomocí `API calls`, které odesílají `HTTP` požadavky na server.

Reducers

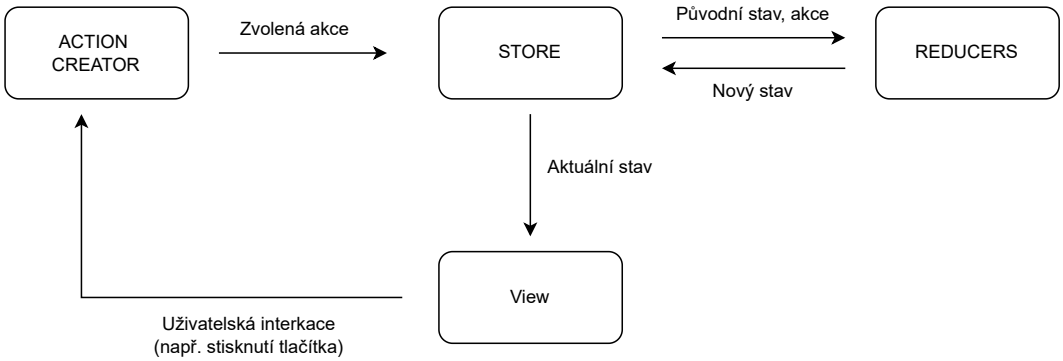
`Reducers` jsou čisté funkce, které na základě stavu aplikace a akce, jež se má vykonat, vrací nový stav. Každý `reducer` se specializuje na určitou část aplikace. Pro zapouzdření jednotlivých `reducers` využíváme funkci `combineReducers`. Ta vrací jeden základní `reducer`, který je pak registrován v objektu `store`.

Store

`Store` je `JS` objekt, který uchovává stav celé aplikace a poskytuje rozhraní pro jeho čtení a aktualizaci. K vytvoření instance tohoto objektu slouží funkce `configureStore`[10]. Tato funkce byla využita při implementaci se dvěma parametry:

- `reducer` - kořenový `reducer`
- `initialState` - definuje počáteční stav. V projektu je využit pro čtení dat z lokálního úložiště prohlížeče.

Aby měl tento objekt přístup do různých částí aplikace je využita komponenta `Provider`. `Provider` zpravidla „obaluje"komponentu nejvyšší úrovně, která se nazývá `App`. Díky tomu je pak `store` schopný znovu vykreslit příslušnou stránku v případě, že dojde k určitým změnám.



Obrázek 4.2: Model znázorňující komunikaci mezi komponenty `Reduxu`. Akce vyvolaná uživatelem změní stav komponentě `reducer`, která následně zasílá aktuální stav do `store`, který pak stránku nově vykreslí.

[9] https://react-redux.js.org/using-react-redux/connect-mapdispatch
[10] https://redux-toolkit.js.org/api/configureStore

4.2.3 Platební systém

V projektu není nutné uchovávat platební informace jednotlivých uživatelů, proto jsou platby implementovány pouze na klientské straně. Mezi nejpopulárnější platební poskytovatelé se v posledních letech řadí `Stripe` a `PayPal` [7].

`Stripe`

`Stripe` nabízí `API`, které lze snadno integrovat do většiny populárních programovacích jazyků. Disponuje rozsáhlou dokumentací, jež detailně popisuje, jak jednotlivé komponenty nakonfigurovat. `Stripe` funguje jako platební brána i jako platební procesor. Jeho hlavní výhodou je možnost použití různých platebních metod jako `Apple Pay`, `Google Pay` a další. `Stripe` také umožňuje vývojářům jednoduché přizpůsobení platebního formuláře.

`PayPal`

`PayPal` je nejpopulárnější platební brána, která je podporována ve více než 203 zemích. Nabízí vývojářům rozsáhlou dokumentaci a také testovací režim plateb, který lze využít při samotném vývoji aplikace. Na rozdíl od brány `Stripe` funguje na bázi účtu, v němž si uživatelé můžou přidávat různé platební či debetní karty, které mohou využít k placení.

Vzhledem k výborným testovacím možnostem a jednoduché integraci do `React` projektu byl pro tuto aplikaci jako platební systém vybrán právě `PayPal`.

Implementace `PayPal` platební brány

Před samotnou implementací bylo nutné vytvořit dva `PayPal` účty. První účet reprezentuje byznys účet, na které jsou odeslány částky z vyřízených objednávek. Druhý účet slouží pro simulaci odeslání platby za objednávku.

K integraci `PayPal` brány do projektu je nutné importovat knihovnu `@paypal/react-paypal-js`. Ta se skládá ze dvou hlavních částí:

- PayPalScriptProvider - slouží k načtení `JS SDK` skriptu do projektu. Stejně jako `Redux Provider` 4.2.2 obaluje hlavní komponentu `App`, aby mohl detekovat změny v podřadných komponentech. Obsahuje parametr `options`, v němž je nutné specifikovat `client-id` a měnu, ve které budou platby účtovány. `Client-id` reprezentuje identifikátor účtu, na který se budou odesílat platby za objednávky.
- PayPalButtons - slouží k vykreslení `PayPal` uživatelského rozhraní.

Komponenta `PayPalButtons` obsahuje parametry `createOrder` a `onApprove`. Parametru `createOrder` je přiřazena `Arrow` funkce `onCreateOrder`, jež vytvoří samotný platební formulář s celkovou cenou pro jednorázovou platbu. Stejně tak je přiřazena `Arrow` funkce i paramtru `onApprove`. Ta symbolizuje úspěšné provedení platby. Také nastavuje hodnotu proměnné `created` na `True`. Tato proměnná je pak použita v `hooku useEffect`, který vykoná akci pro vytvoření objednávky na serveru a přesměruje uživatele na stránku o úspěšném provedení objednávky. Kód byl převzat a modifikován pro potřeby aplikace z oficiálního `PayPal` blogu[11].

[11] https://developer.paypal.com/community/blog/how-to-add-paypal-checkout-payments-to-your-react-app/

4.2.4 Stránky

Přihlášení uživatele

Pro přihlášení je uživateli nabídnuta stránka s formulářem, ve kterém musí specifikovat svůj e-mail a heslo. V případě, že tyto údaje nejsou korektní, je uživatel s touto skutečností obeznámen ve formě upozornění. Formulář také obsahuje odkaz na stránku, kde je možné se registrovat. Všechny údaje jsou povinné, tudíž v případě jejich nevyplnění není registrace možná. Registrace také vyžaduje unikátní e-mail, se kterým se uživatel následně přihlašuje.

Po úspěšné registraci je uživatel automaticky přihlášen a je mu zpřístupněn panel, který má hodnotu jeho jména. Pokud přejede myší přes tento panel jsou mu zobrazeny dvě možnosti:

- Profil - po kliknutí na tuto sekci se uživatel dostává do svého menu, ve kterém si může zobrazit objednávky, napsat zprávu, případně upravit své informace.
- Odhlásit se - tato možnost je k nalezení i v sekci profil, nicméně vzhledem k výsledkům testování bylo nutné tuto možnost více zviditelnit.

V momentě, kdy je uživatel přihlášen, začíná odpočet platnosti přístupového `tokenu` uživatele. Platnost tohoto `tokenu` byla nastavena v `Djangu` na 7 dní, proto je nutné v klientské části tuto hodnotu sledovat a v případě jejího vypršení uživatele odhlásit. K tomuto monitorování slouží funkce `AuthVerify`, která porovnává datum expirační doby `tokenu` s aktuálním časem. Tato funkce byla převzata z již existujícího článku[12] a modifikována pro potřeby aplikace.

Domovská stránka

Domovská stránka zobrazuje aktuálně dostupné skleníky a jejich detaily. Je zde využito komponenty `Pagination`, která definuje celkový počet stránek v závislosti na celkovém množství skleníků. Pro potřeby aplikace je počet skleníku na stránku stanoven na 5. V případě, že těchto skleníku existuje více, je nutné se přepnout na další stránku, čehož lze docílit kliknutím na číslo příslušné stránky v zápatí.

Zobrazené skleníky fungují jako odkaz, po jehož kliknutí je uživatel přesměrován na stránku, která zobrazuje produkty dostupné v příslušném skleníku.

Nabídka produktů

Stránka s nabídkou produktů se vztahuje k jednotlivým skleníkům. Zákazník si tak jednoduše může vybrat skleník blízko jeho lokace a zakoupit si produkty, kterými skleník disponuje. Opět je využito komponenty `Pagination`, která produkty rozděluje po pěti na příslušné stránky. Zákazníkovi je představen obrázek produktu, jeho název a cena. Celý tento objekt funguje jako odkaz, který uživatele po kliknutí přesměruje na detail produktu, ve kterém si může zvolit množství a následně jej vložit do košíku.

Košík

Košík je implementován za využití lokálního úložiště v prohlížeči. Poté, co zákazník klikne na tlačítko přidat do košíku, je vyslána akce, která získá data ze serveru o příslušném produktu. Tyto data jsou následně pomocí `reduceru` uloženy do příslušného stavu a lokálního úložiště. Dokud je tedy uživatel přihlášen, položky v košíku zůstávají.

[12] https://www.bezkoder.com/handle-jwt-token-expiration-react/

Zákazník může také jednotlivé položky z košíku odebrat. V případě, že je se svou selekcí spokojen je přesměrován na stránku s objednávkou, kde dochází k následnému zaplacení.

Objednávka a platba

Stránka s objednávkou slouží jako prostředník mezi košíkem a samotným zaplacením. Zákazník již nemůže odstraňovat jednotlivé položky. Je mu přehledně zobrazena rekapitulace objednávky společně s celkovou cenou.

Po kliknutí tlačítka zaplatit je zákazníkovi zobrazena `PayPal` brána, která přehledně zobrazuje celkovou cenu v českých korunách. V momentě, kdy je objednávka zaplacena, dochází k jejímu samotnému vytvoření pomocí akce `createOrder`. V případě, že je zákazník přihlášen, může si tuto objednávku zpětně zobrazit ve svém profilu. Po úspěšném zaplacení je zákazník přesměrován na stránku, jež oznamuje úspěšné zaplacení.

Administrace

Jelikož v systému rozlišujeme krom běžných uživatelů také správce a administrátora, bylo nutné stránky pro tyto speciální uživatelé ošetřit a to tak, aby pro regulérního uživatele nebylo možné se k těmto pohledům dostat.

Pro stránky správce dochází k využití komponenty `ProtectedAdminRoute`. Dle údajů uložené v lokálním úložišti tato komponenta rozlišuje, jestli je uživatel správcem, regulérním uživatelem nebo není vůbec přihlášený. Pokud je uživatel správce, je mu umožněn přístup. Pokud by se normální uživatel snažil dostat na tuto URL adresu, bude mu zobrazena zpráva, která oznamuje, že k tomuto pohledu nemá oprávnění. V neposlední řadě pokud není uživatel vůbec přihlášen, bude přesměrován na přihlašovací stránku.

Stejný přístup je pak využit pro pohledy administrátora, kde je využito komponenty `ProtectedSuperuserRoute`, která funguje na stejném principu jako `ProtectedAdminRoute` s rozdílem, že ověřuje hodnotu proměnné `isAdmin`.

4.2.5 Úprava `UI` pro mobilní zařízení

Aplikace byla primárně vyvíjena jako desktopová aplikace, avšak pro využití v praxi je nutné zajistit funkčnost a přehlednost `UI` i na mobilních zařízeních. Většina stylizace byla vytvořena bez pomocí externích knihoven za použití relativních jednotek. Pokud nejsou hodnoty jednotlivých objektů pevně určené, ale odvíjí se od šířky obrazovky, je pak jednoduché modifikovat rozložení na míru různým zařízením.

Pro definování pravidel bylo využito `media query`, které reaguje na změnu šířky obrazovky. Uživatelské rozhraní je tak pozměněno v případě, že šířka obrazovky klesne pod 600 pixelů. V tomto případě dochází ke změně polohy prvků do sloupců a úpravě velikostí jednotlivých elementů tak, aby mobilní uživatel mohl přehledně vykonávat své úkony.

4.2.6 Výsledky a možnosti rozšíření

Výsledným produktem je aplikace, která uživatelům nabízí intuitivní prostřední pro vytváření a zaplacení objednávek. Aplikace také motivuje, aby se noví uživatelé registrovali, nabízí totiž řadu funkcí jako posílání zpráv a zobrazení historie objednávek. Správcům je umožněna jednoduchá manipulace s produkty a také vytvoření vlastních poznámek v podobě revize.

V budoucím vývoji by se tato práce dala propojit s projektem chytrý skleník, kde by například každý uživatel, který si ve skleníku pronajal plochu, dostal identifikátor k příslušnému skleníku. Tento identifikátor by se pak mohl využít k omezení komunikace a to tak, aby příslušného správce mohli kontaktovat pouze uživatelé, kteří mají v skleníku pronajatou plochu. Dále by bylo vhodné při autentizaci využívat i obnovovací `token`, který by prodlužoval platnost přístupového `tokenu`, pokud by byl uživatel na stránce aktivní. V případě nasazení aplikace do provozu by bylo nutné přejít na plnohodnotný databázový systém a zaplatit jeho hostování.

Kapitola 5

Závěr

Hlavním cílem projektu bylo vytvořit aplikaci, která je jednoduchá a intuitivní na použití. Pro splnění těchto cílů bylo nutné se v první řadě seznámit se samotnou problematikou práce a cílovými uživateli, kteří by tuto aplikaci mohli využívat.

Před samotnou implementaci byl proveden důkladný průzkum existujících technologií pro tvorbu webových aplikací. Znalosti nabyté z tohoto průzkumu pak byly využity pro vybrání vhodné kombinace, která pak ulehčila samotný proces implementace.

Během vývoje celé práce byl kladen velký důraz na testování a zpětnou vazbu uživatelů. V rámci práce byl vytvořen drátěný model, ve kterém byly ujasněny potřeby normálních zákazníků, kteří nebyli vzdělaní v technických oborech. Jelikož se v aplikaci vyskytují i správci, kteří mají na starost technickou část chodu aplikace, bylo nutné provést testování i na tomto sektoru uživatelů. K tomu posloužil vytvořený `mockup`, který simuloval grafické rozhraní aplikace.

Po zohlednění zpětné vazby uživatelů a úpravě návrhů jsem přistoupil k samotné implementaci projektu. Výsledkem je intuitivní, responzivní aplikace, která je naprogramovaná tak, aby byla lehce rozšířitelná pro budoucí potřeby. Aplikace splňuje veškerou funkcionalitu, která byla na začátku projektu stanovena.

V budoucnu by aplikace mohla být rozšířena o různé možnosti plateb. Zároveň by uživatelům mohly být zasílány notifikace s různými aktualizacemi, například jaký produkt je ve slevě apod. Možností vylepšení je také přidání sekcí s novinkami nebo případně nástěnku s různými obrázky.

Literatura

[1] Awati, R. *Spring Framework (Spring)* [online]. 2024 [cit. 2024-03-10]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework.

[2] Chinnathambi, K. *Learning React*. 1st. Addison-Wesley Professional, 2016. ISBN 0134546318.

[3] Crockford, D. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008. ISBN 0596517742.

[4] EDUCBA. *Struts in Java* [online]. 2023 [cit. 2024-03-07]. Dostupné z: https://www.educba.com/struts-in-java/.

[5] Figma. *What is Wireframing?* [online]. [cit. 2024-03-21]. Dostupné z: https://www.figma.com/resource-library/what-is-wireframing/.

[6] Hufford, B. *What is a Mockup? (+How to Create a Mockup in 2022)* [online]. 2022 [cit. 2024-03-21]. Dostupné z: https://cliquestudios.com/mockups/.

[7] Juillet, R. *Online Payment Gateways: Best Options* [online]. 2020 [cit. 2024-04-21]. Dostupné z: https://www.bocasay.com/online-payment-gateways-best-options/.

[8] Juviler, J. *What Is an API Endpoint? (And Why Are They So Important?)* [online]. 2024 [cit. 2024-04-05]. Dostupné z: https://blog.hubspot.com/website/api-endpoint.

[9] Kareliya, A. *Decoding the Java vs Python Debate: Which Programming Language is Better?* [online]. 2023 [cit. 2024-03-07]. Dostupné z: https://radixweb.com/blog/python-vs-java.

[10] Krug, S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3rd. New Riders Publishing, 2014. ISBN 0321965515.

[11] Makai, M. *Object-relational Mappers (ORMs)* [online]. 2022 [cit. 2024-03-26]. Dostupné z: https://www.fullstackpython.com/object-relational-mappers-orms.html.

[12] Mannotra, V. *Guide to Web Application Testing* [online]. 2023 [cit. 2024-03-19]. Dostupné z: https://www.browserstack.com/guide/web-application-testing.

[13] Nedbálek, S. *Uživatelské rozhraní mobilních aplikací*. Brno, CZ, 2017. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Dostupné z: https://is.muni.cz/th/exrqj/.

[14] Spring. *Spring Framework* [online]. [cit. 2024-03-07]. Dostupné z: https://spring.io/projects/spring-framework.

[15] Stefanov, S. *React: Up & Running: Building Web Applications*. 1st. O'Reilly Media, 2016. ISBN 1491931825.

[16] Vincent, W. S. *Django for Beginners: Build Websites with Python and Django*. WelcomeToCode, 2020. ISBN 1735467200.

[17] Zeman, F. *Úvod do testování v JavaScriptu* [online]. 2021 [cit. 2024-03-19]. Dostupné z: https://www.itnetwork.cz/javascript/testovani/uvod-do-testovani-v-javascriptu.

Příloha A

Struktura obsahu přiložené SD karty

root
- `djangoApp` ..... Zdrojové kódy aplikace
- `latex` ..... Zdrojové soubory textu ve formátu LaTeX
- `app_video.mkv` ..... Demonstrační video
- `Poster.pdf` ..... Plakát k webové aplikaci
- `HowToRun.pdf` ..... Návod pro spuštění aplikace
- `SmartMarketThesis.pdf` ..... Bakalářská práce ve formátu PDF