FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Webová aplikace pro správu domácího inventáře |
| **Student:** | Nicolas Stefan Maskaľ |
| **Vedoucí:** | Ing. Zdeněk Rybola, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové a softwarové inženýrství, zaměření Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2023/2024 |

## Pokyny pro vypracování

Cílem práce je navrhnout a realizovat část jednotného systému pro správu domácnosti umožňující správu domácího inventáře. Mezi klíčové vlastnosti patří evidence předmětů, jejich vlastností, množství, míry spotřeby, umístění, trvanlivosti (včetně jejího hlídání), apod. Systém by měl také umožnit pohodlné vyhledávání a kategorizaci předmětů s jejich agregací.

Při realizaci postupujte podle technik softwarového inženýrství. Zejména pak:
- Analyzujte potřeby v této oblasti a proveďte rešerši existujících řešení pro tuto agendu
- Společně s vedoucím práce stanovte konkrétní požadavky na tuto část systému
- Společně s kolegou navrhněte jádro systému umožňující jednotnou správu uživatelů a domácností a autentizaci
- Navrhněte architekturu a konkrétní řešení této části systému s ohledem na snadnou integraci s jádrem systému a budoucí rozšiřitelnost
- Dle provedeného a schváleného návrhu řešení implementujte a řádně otestujte
- Při řešení aplikujte techniky zajištění kvality kódu a kontinuální integrace
- Řešení odpovídajícím způsobem zdokumentujte a připravte k produkčnímu nasazení

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. ledna 2023 v Praze.*

Bakalárska práca

# WEBOVÁ APLIKACE PRO SPRÁVU DOMÁCÍHO INVENTÁŘE

**Nicolas Stefan Maskaľ**

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Zdeněk Rybola, Ph.D.
10. januára 2024

# Obsah

# Zoznam obrázkov



# Zoznam tabuliek

# Zoznam výpisov kódu

*Chcel by som poďakovať svojmu vedúcemu práce, Ing. Zdeňku Rybolovi, Ph.D., za jeho čas, odborné vedenie práce a užitočné rady. Ďalej by som chcel poďakovať svojej rodine a priateľke za neustálu podporu počas celej doby štúdia.*

# Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Prahe dňa 10. januára 2024 .....

# Abstrakt

Táto bakalárska práca sa zaoberá návrhom a vývojom aplikácie HouseKeeper – aplikácie na komplexnú správu domácností, ich inventára a energií. Aplikácia umožňuje sledovanie zásob rôznych produktov a kusov, ich členenie do kategórií a lokácií. Medzi ďalšie funkcionality aplikácie patrí mimo iné i správa domácnosti. Na implementáciu serverovej časti bol použitý programovací jazyk PHP a framework Laravel. Pri implementácii klientskej časti bol zase použitý programovací jazyk Typescript a framework React. Najprv bola vykonaná rešerš už existujúcich podobných aplikácií, potom bol vytvorený návrh vlastného riešenia. Ďalej bolo implementované jadro aplikácie, obsahujúce funkcionalitu na správu domácnosti a jej členov. Súčasťou implementácie bol aj modul na správu inventára domácnosti. Táto aplikácia bola vytvorená v spolupráci s kolegom Dávidom Jenčom, ktorý mal na starosti klientskú časť jadra aplikácie a modul pre sledovanie energií v domácnosti. Na záver bola aplikácia otestovaná a pripravená na produkčné nasadenie.



**Kľúčové slová** webová aplikácia, inventár, domácnosť, kategorizácia produktov, agregácia kusov, PHP, Laravel



# Abstract

This bachelor thesis focuses on the design and development of the application HouseKeeper – an application for complex management of households, their inventory and energy. The application enables inventory tracking of various products and items, as well as their classification into categories and locations. Other functionalities of the application include household management. For the server-side implementation, the PHP programming language and the Laravel framework were used. For the client-side implementation, the Typescript programming language and the React framework were utilized. First, a research of already existing similar applications was carried out, followed by the creation of a custom solution design. Subsequently, the core of the application was developed, incorporating features to manage households and their members. The implementation also included the module for inventory management. This application was created in collaboration with colleague David Jenčo, who was responsible for the client part of the core application and the module for energy tracking. Finally, the application was tested and prepared for production deployment.



**Keywords** web application, inventory, household, product categorization, item aggregation, PHP, Laravel

# Zoznam skratiek

| | |
|---|---|
| API | Application Programming Interface |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| IDE | Integrated Development Environment |
| JSON | JavaScript Object Notation |
| NoSQL | Not only SQL |
| ORM | Object-Relational Mapping |
| OWASP | Open Web Application Security Project |
| PHP | PHP: Hypertext Preprocessor |
| REST | Representational state transfer |
| SQL | Structured Query Language |
| XSS | Cross site scripting |

# Kapitola 1
# Úvod

Koncept správy domáceho inventára vychádza zo základnej ľudskej potreby organizovať svoj osobný majetok. Inventarizácia je základným aspektom života v domácnosti, ktorý sa v priebehu storočí výrazne vyvíjal. Spisovanie si evidencie vlastného majetku siaha hlboko do minulosti, k starovekým civilizáciám. Kumulácia ľudského majetku si vyžadovala systém, ktorý by evidoval, čo ľudia vlastnia, čo spotrebovali a čo ešte potrebujú. Ľudia si zaznamenávali údaje o svojom vlastníctve najprv do hlinených tabuliek, či ich zapisovali na papyrus. S postupom času je možné vidieť vývoj tejto praktiky od jednoduchých papierových zoznamov až po sofistikované digitálne databázy, ktoré odrážajú technologický a spoločenský pokrok.

V dnešnej dobe na trhu existuje veľa riešení pre správu domáceho inventára, vo forme rôznych mobilných, či webových aplikácií. Avšak, mnoho aplikácií predkladá často nepostačujúce funkcie so zložitým a neintuitívnym užívateľským rozhraním. Lepšie aplikácie s prepracovanými funkcionalitami sú mnohokrát spoplatnené nemalými čiastkami. Stále je tu vidieť priestor na pokrok, preto bolo opodstatnené, aby vznikla aplikácia, ktorá by predstavila pokročilé funkcie, a to zadarmo. Táto bakalárska práca sa preto zaoberala vývojom takejto aplikácie. Zamerala sa na vyplnenie medzier, ktoré sa v súčasných riešeniach objavujú.

Aplikácia HouseKeeper, vyvinutá v tejto bakalárskej práci, je aplikácia pre komplexnú správu domácností. Súčasťou jej systému je modul pre správu inventára domácnosti a modul pre sledovanie energií domácnosti. Jadro aplikácie bolo vyvinuté spoločne s kolegom Dávidom Jenčom, ktorý taktiež vyvinul modul pre správu energií.

V druhej kapitole sa táto bakalárska práca venovala analýze požiadaviek na aplikáciu a analýze problematiky, pričom bola vykonaná rozsiahla rešerš podobných aplikácií. V tretej kapitole je opísaný návrh riešenia, diskusia nad voľbou programovacieho jazyka, frameworku, architektúry a databázy. Kapitola 4 sa venovala opisu použitého spôsobu riadenia softvérového projektu a implementácie riešenia aplikácie. V piatej kapitole je zhrnuté testovanie a nasadenie aplikácie.

## 1.1 Ciele práce

Hlavným cieľom tejto záverečnej práce je navrhnutie a vytvorenie systému pre správu domácností s názvom HouseKeeper. Tento systém bude mimo iné umožňovať správu domáceho inventára.

Prvým cieľom práce je určenie konkrétnych požiadaviek na aplikáciu spoločne s vedúcim práce. Ďalším cieľom je analýza už existujúcich riešení podobných aplikácií. Ďalej je cieľom spolu s kolegom Dávidom Jenčom navrhnutie jadra systému, pričom bude mať kolega na starosti klientskú časť, a autor práce serverovú časť. Jadro bude umožňovať autentizáciu, a jednotnú správu užívateľov a domácností.

Dôležitým cieľom vyplývajúcim z dohodnutých požiadaviek je návrh architektúry a návrh konkrétneho riešenia systému inventára. Dôraz bude kladený na jeho integráciu s jadrom systému a budúcu rozšíriteľnosť. Na základe schváleného návrhu je cieľom implementácia riešenia. V neposlednom rade je cieľom otestovanie, zdokumentovanie riešenia a jeho príprava k nasadeniu.

# Kapitola 2
# Analýza

Nevyhnutnou súčasťou správneho vývoja softvéru je analýza riešenej problematiky. Keďže aplikácia je vyvíjaná na mieru pre zadávateľa (Ing. Zdeněk Rybola, Ph.D.), začiatok kapitoly je venovaný opisu funkcionalít, ktoré by mala daná aplikácia podľa zadávateľa spĺňať. Súčasťou kapitoly je aj vymedzenie funkčných a nefunkčných požiadaviek, analýza existujúcich riešení, model prípadov použitia a doménový model. Analýza je sústredená hlavne na funkcionality umožňujúce správu domáceho inventára.

## 2.1 Opis požadovaných funkcionalít aplikácie

Od aplikácie HouseKeeper sa požaduje umožnenie komplexnej správy domácností. Aplikácia by mala byť členená na viacero častí – na jadro, inventár domácnosti a sledovanie energií domácnosti. Jadro aplikácie vznikne v spolupráci s kolegom Dávidom Jenčom.

Jadro aplikácie by malo podporovať funkcionalitu pre registráciu nových užívateľov, ich prihlasovanie a odhlasovanie, ako aj možnosť zmeny hesla. Registrácia užívateľa v aplikácii je podmienená odoslaním verifikačného e-mailu, a až do jeho potvrdenia nebude užívateľ môcť využívať aplikáciu.

Po úspešnej registrácii poskytne aplikácia nástroje na správu domácností – umožní vytvorenie novej domácnosti, jej úpravu alebo odstránenie, či prehľad všetkých domácností užívateľa. Pre praktické používanie aplikácie je rovnako potrebné, aby mohol užívateľ zdieľať svoje domácnosti aj s inými užívateľmi. Tým sa pozvaní užívatelia stanú členmi danej domácnosti, pričom im bude možné nastaviť práva na jednotlivé moduly aplikácie. Členom domácnosti bude možné prideliť neobmedzený prístup k modulu, alebo im bude možné prístup obmedziť. Obmedzenie spočíva v tom, že nebudú môcť daný modul upravovať, iba čítať, prípadne bude možné prístup úplne obmedziť, aby daný modul nemohli ani vidieť.

Medzi hlavné funkcionality inventára domácnosti by malo patriť sledovanie zásob rôznych produktov. Produkty môžu súvisieť s potravinami, s osobnou hygienou, s materiálom na rekonštrukciu domu, alebo s čímkoľvek iným podľa špecifických potrieb užívateľa. Jednotlivé produkty by sa mali dať členiť do kategórií definovaných užívateľom. Tieto produkty môžu mať atribúty súvisiace s daným produktom, či s jeho kusmi. Od aplikácie sa očakáva i možnosť vyhľadávania produktov a kusov na základe ich názvu, popisu, kategórie a lokácie. Je potrebné umožniť sledovanie rôznych kusov produktov rovnakého typu oddelene, vrátane možnosti zaznamenávania zostávajúceho množstva či kvality jednotlivých kusov. Pre každý kus musí byť možné nastaviť dátum expirácie a upozornenie na tento dátum, aby bolo možné daný kus vyhodiť alebo doskladniť. Kusy produktu sa môžu nachádzať v rôznych lokáciách, ktoré môžu, no nemusia byť spoločné pre viacero kusov. Každá takáto lokácia môže mať navyše ďalšie podlokácie.

Ďalšou časťou aplikácie by malo byť sledovanie energií danej domácnosti. Malo by byť možné sledovať spotrebu a výdaje za elektrinu, vodu či plyn. Očakáva sa, že aplikácia bude schopná vykresliť rôzne grafy zobrazujúce sledované údaje. Na tejto časti bude pracovať kolega Dávid Jenčo v rámci jeho bakalárskej práce.

Medzi ďalšie rozšírenia aplikácie, ktorým by sa v budúcnosti mohli zaoberať ďalší študenti v rámci svojich bakalárskych prác, by mohol patriť modul pre správu úloh. Tento modul by umožnil vytváranie úloh a ich prideľovanie ďalším členom domácnosti. Ďalším rozšírením by mohol byť modul pre plánovanie údržby, teda pravidelných opráv, revízií, apod.

## 2.2 Požiadavky

Cieľom analýzy požiadaviek je vymedziť hranice aplikácie, vyjasniť zadanie so zadávateľom a zachytiť prípadne obmedzenia, ktoré budú kladené na aplikáciu. Požiadavky musia byť jasné a ľahko overiteľné. Požiadavky v tejto podkapitole sú zamerané hlavne na správu domáceho inventára.

### 2.2.1 Funkčné požiadavky

Sekcia obsahuje popis funkčných požiadaviek, ktoré sú kladené na aplikáciu. Funkčné požiadavky definujú očakávané funkcionality a chovanie aplikácie.

- **F1. Správa kategórií**

  Aplikácia bude umožňovať užívateľovi vytvárať, spravovať, upravovať a vymazávať kategórie produktov v domácnosti. Kategória môže mať niekoľko podkategórií. Do každej kategórie bude možné priradiť produkty. Pri tvorbe novej kategórie bude možné špecifikovať aj predvolený typ a jednotku merania produktov. Príkladom kategórie môže byť kategória *Nápoje*, ktorá by mohla mať podkategórie *Alkoholické nápoje* a *Nealkoholické nápoje*.

- **F2. Správa lokácií**

  Aplikácia bude umožňovať užívateľovi vytvárať, spravovať, upravovať a vymazávať lokácie v jeho domácnosti. Do jednotlivých lokácií bude možné priradiť jednotlivé kusy produktov. Každá lokácia bude môcť obsahovať akýkoľvek počet podlokácií, v ktorých sa taktiež budú môcť nachádzať kusy produktov. Príkladom lokácie môže byť lokácia *Kuchyňa*, ktorá by mohla mať podlokáciu *Chladnička*.

- **F3. Správa atribútov**

  Aplikácia bude umožňovať užívateľovi vytvárať, spravovať, upravovať a vymazávať atribúty. Pri každej kategórii bude možné vytvoriť definície atribútov, ktoré sa budú viazať na produkty týchto kategórií. Každá podkategória bude takisto obsahovať tieto atribúty. Pri každom produkte bude rovnako možné vytvoriť definície atribútov, no tieto atribúty sa budú vzťahovať iba na kusy týchto produktov. Príkladom atribútu môžu byť *Kalórie*.

- **F4. Správa produktov**

  Aplikácia bude umožňovať užívateľovi vytvárať, spravovať, upravovať a vymazávať produkty v domácnosti. Pri tvorbe nových produktov si užívateľ zvolí kategóriu, typ merania a jednotku merania. Podporované typy merania budú množstvo, dĺžka, objem, opotrebenie a hmotnosť. Podporované jednotky merania budú kusy, centimetre, metre, mililitre, litre, percentá opotrebenia, gramy a kilogramy. Užívateľ si bude môcť zvoliť aj predvolenú lokáciu kusov produktu. Príkladom produktu môže byť *Pomarančový džús*.

- **F5. Správa kusov**

  Aplikácia bude umožňovať užívateľovi vytvárať, spravovať, upravovať a vymazávať kusy konkrétneho produktu. Každý kus bude mať atribúty podľa toho, akého produktu je exemplárom. Pri jednotlivých kusoch bude možné zaznamenávať dátum spotreby, lokáciu, zostávajúce množstvo či opotrebovanie daného kusu. Aplikácia musí umožňovať aj tvorbu a vymazanie viacerých kusov naraz. Kusom produktu sa môže rozumieť napríklad konkrétna fľaša pomarančového džúsu.

- **F6. Pokročilé vyhľadávanie produktov a kusov**

  Aplikácia musí umožniť vyhľadávať produkty podľa ich názvu, typu merania, mernej jednotky, kapacity, atribútov, kategórie a podľa lokácie kusov. Ďalej musí ponúknuť možnosť vyhľadávať kusy podľa názvu produktu, popisu kusu, dátumu expirácie, typu a jednotky merania, zostávajúceho množstva, kategórie a lokácie.

- **F7. Upozornenia na expiráciu kusov**

  Aplikácia bude umožňovať užívateľovi nastavenie upozornenia pred expiráciou jednotlivých kusov. Pre každý produkt bude možné nastaviť, koľko dní pred expiráciou dôjde k upozorneniu. Upozornenia vo forme emailu budú hromadne posielané o polnoci všetkým členom domácnosti s právom modul inventára upravovať.

- **F8. Upozornenia na nízke zostávajúce množstvo produktu**

  Aplikácia bude umožňovať užívateľovi nastavenie upozornenia o nízkom zostávajúcom množstve produktu. Pre každý produkt bude možné nastaviť, pri akom množstve dôjde k upozorneniu. Upozornenia vo forme emailu budú hromadne posielané o polnoci všetkým členom domácnosti s právom modul inventára upravovať.

- **F9. Agregácia kusov produktu**

  Aplikácia umožní užívateľovi zobraziť si informáciu o zostávajúcom množstve produktu. Od aplikácie sa očakáva i agregácia viacerých produktov naraz, aby si mohol užívateľ zobraziť zostávajúce množstvo všetkých produktov patriacich do istej kategórie.

- **F10. História zmien**

  Aplikácia umožní užívateľovi zobraziť históriu zmien týkajúcich sa použitého a zostávajúceho množstva produktu, ktoré vykonal užívateľ alebo ostatní členovia danej domácnosti.

- **F11. Správa užívateľov**

  Aplikácia umožní užívateľovi tvorbu nového účtu, do ktorého sa potom bude môcť prihlásiť.

- **F12. Správa domácnosti**

  Aplikácia umožní vytváranie, úpravu alebo vymazanie domácností. Taktiež bude možné pridávať ďalších členov do domácnosti a nastaviť ich oprávnenia pre každý modul, ako sú právo na úpravu modulu alebo čítanie modulu. Rovnako bude možné úplne obmedziť prístup k modulu.

### 2.2.2 Nefunkčné požiadavky

Sekcia nefunkčných požiadaviek obsahuje popis všetkých požiadaviek, ktoré súvisia s použitými technológiami, vlastnosťami a charakteristikami aplikácie.

- **N1. Integrácia do jadra systému HouseKeeper**

  Modul inventára domácnosti musí byť integrovaný do jadra systému HouseKeeper a musí s ním byť kompatibilný. Serverová časť aplikácie by mala byť písaná v programovacom jazyku PHP a taktiež musí ponúkať rozhranie REST.

- **N2. Lokalizácia**

  Aplikácia musí podporovať český jazyk, no musí umožnovať i jednoduché pridanie iných jazykov iba zmenou príslušných prekladových súborov bez nutnosti zmeny kódu.

- **N3. Funkčnosť vo webových prehliadačoch**

  Aplikácia musí byť použiteľná vo webových prehliadačoch. Garantovaným prehliadačom je Google Chrome verzie 119.0.6045.123 a Firefox verzie 119.0.1.

## 2.3 Existujúce podobné riešenia

Pred vyvinutím aplikácie je potrebné urobiť rešerš ohľadom existujúcich riešení danej problematiky. Manažment inventára domácnosti nie je žiadna novinka, podobných aplikácii už existuje mnoho. Z rešerše sú vynechané aplikácie, ktoré sú šité na mieru pre správu inventára podnikov, pretože nie sú vhodným riešením pre domácnosť. Tieto aplikácie majú väčšinou predpripravenú integráciu iných služieb, ako sú platobné metódy, kuriérske spoločnosti, či e-shopy ako Ebay[1] a Amazon[2].

### 2.3.1 Sortly

Sortly [1] je webová aplikácia na sledovanie inventára a majetku, ktorá umožňuje sledovať a zaznamenávať produkty a všetky ich podrobnosti. Ponúka moderné grafické rozhranie vo webovej verzii (Obr. 2.1) s možnosťou nahrávania vlastných snímkov produktov.

Aplikácia ponúka rôzne stupne predplatného – verziu *Advanced* za 29$ mesačne, či verziu *Ultra* za 59$ mesačne [2]. Tieto verzie sa od seba líšia v počte obmedzení. Aplikácia ponúka taktiež neplatenú verziu, ale jej nevýhodou je to, že je príliš obmedzená a ponúka málo funkcionalít. Aplikácia Sortly taktiež ponúka predplatné *Enterprise*, ktoré ale nebolo do porovnania zahrnuté, pretože sa jedná o stupeň predplatného vyrobeného na zakázku.

Sortly umožňuje vytvárať adresáre a ich podadresáre, ktoré sú funkcionalitou podobné požadovaným kategóriám. Pre každý adresár je možné pridať atribút, a tento atribút potom bude mať každý produkt nachádzajúci sa v tomto adresári. Bohužiaľ táto funkcionalita v neplatenej verzii ponúka užívateľovi vytvoriť iba jeden vlastný atribút pre každý produkt. Podľa stupňa plánu predplatného je toto obmedzenie zvýšené na 10, alebo až 25 vlastných atribútov pre každý produkt. Toto obmedzenie je ale z pohľadu stanovených funkčných požiadaviek na systém stále nepostačujúce.

Podobne ako s atribútmi, i funkcionalita pozvania ďalších členov je obmedzená podľa stupňa predplatného. V neplatenej verzii nie je možné pozývať ďalších členov, v platených verziách je možné pozvať 5 alebo 9 členov, čo je rovnako nepostačujúce.

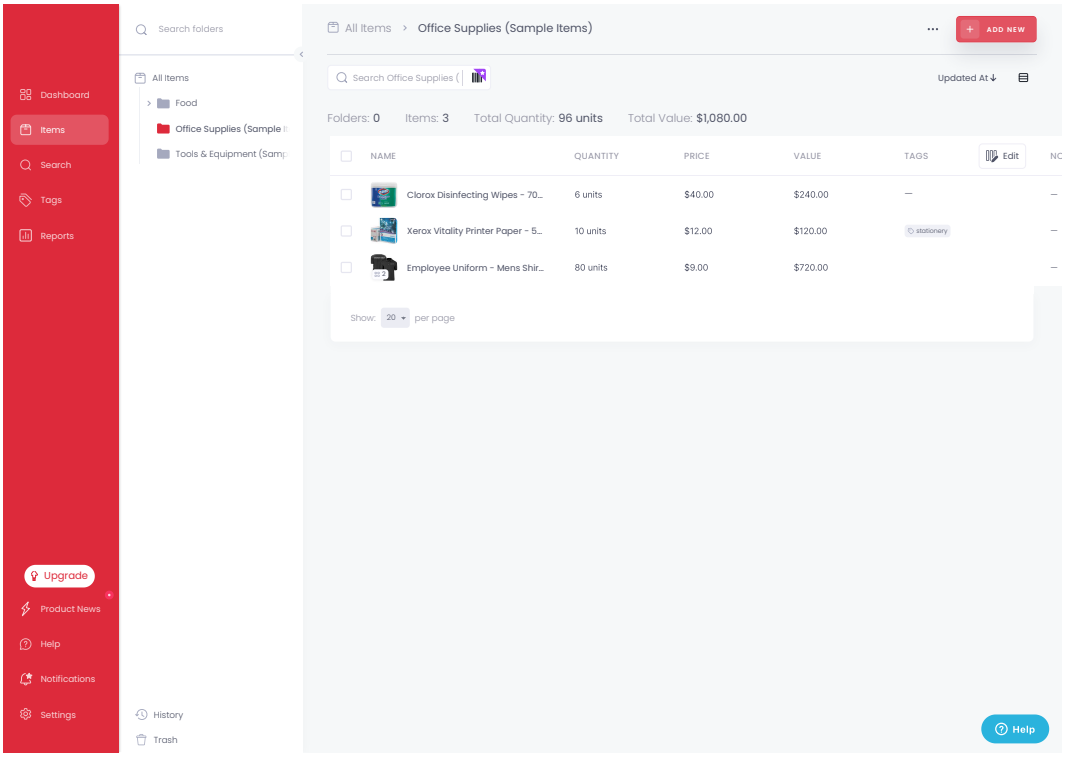
Aplikácia Sortly ako jediná z porovnávaných aplikácií ponúka informáciu o množstve produktov v adresári. Taktiež ako jediná umožňuje funkcionalitu jednotiek merania. V neplatenej verzii je však možné zvoliť iba jednu jednotku merania, a tou sú kusy. Aplikácia taktiež užívateľovi umožňuje nahliadnuť do histórie zmien.

Čo sa týka upozornení, Sortly ponúka i v neplatenej verzii možnosť nastavenia upozornenia na nízke množstvo produktu. Aplikácia však ponúka možnosť upozornenia na expiráciu iba v platených verziách.

Sortly má okrem webovej aplikácie i mobilné aplikácie na operačné systémy Android a iOS. Medzi všetkými týmito aplikáciami ponúka pohodlnú synchronizáciu. Užívateľ takto môže správu jednoduchšieho charakteru vykonať pomocou mobilnej aplikácie, a na tú zložitejšiu použiť webovú aplikáciu.

[1] https://www.ebay.com/
[2] https://www.amazon.com/

Nevýhodou aplikácie je ale úplne chýbajúca správa lokácii a domácnosti a nemožnosť rozlíšiť jednotlivé kusy daného produktu, a to ani v platenej verzii. Tým pádom si nie je možné zaznamenať, že jeden kus nejakého produktu je už čiastočne spotrebovaný, a ostatné sú ešte nepoužité.




**Obr. 2.1** Ukážka vzhľadu webovej aplikácie Sortly

### 2.3.2 Memento Database

Memento Database [3] je mobilná aplikácia pre Android a iOS, dostupná aj v desktopovej verzii. Podobne, ako aj Sortly, umožňuje spravovať produkty v domácnosti, a tie triediť do knižníc a skupín. Jednotlivé produkty je potom možné filtrovať podľa atribútov, pričom je potrebné vyzdvihnúť možnosť špecifikovania typu atribútu, ktorú ponúka Memento Database ako jediná z porovnávaných aplikácií. Je teda možné určiť, či bude daný atribút text, číslo, dátum alebo iný vlastný typ s vlastnými obmedzeniami. Taktiež je možné zobraziť si históriu vykonaných zmien.

V aplikácii bohužiaľ chýba správa domácností, lokácií a jednotlivých kusov. Rovnako chýba možnosť pozývania ďalších členov. Aplikácia síce umožňuje funkcionalitu kategórií, avšak v obmedzenom režime. Nie je možné vytvoriť si vlastné kategórie, iba použiť kategórie predvytvorené.

V rámci aplikácie je možné zoskupiť produkty, a to do knižníc alebo skupín (v kontexte aplikácie HouseKeeper sa jedná o kategórie a podkategórie). Bohužiaľ je táto funkcionalita zoskupenia obmedzená, pretože nie je možné vytvoriť podknižnice a podskupiny.

Medzi ďalšie nevýhody patrí nemožnosť nastaviť si upozornenia na expiráciu a nízke množstvo produktu.

Pozitívom je pokročilá funkcia vloženia vlastného Javascript kódu, ktorý sa spustí, ak nastane nejaká užívateľom vybraná udalosť.

### 2.3.3 Smart Inventory System

Smart Inventory System [4] je mobilná aplikácia pre Android. Existuje aj webová aplikácia, avšak tá je spoplatnená. Smart Inventory System užívateľovi ponúka rôzne funkcionality, avšak mnohé z nich sú v obmedzenom režime vzhľadom k definovaným požiadavkám, alebo sú spoplatnené.

V bezplatnej verzii aplikácie je možné napríklad spravovať a vytvárať produkty, a taktiež ich triediť do skupín (kategórií). Táto funkcionalita je ale neúplná, pretože sa nedajú vytvárať podskupiny (podkategórie). Existuje aj možnosť pridávať vlastné atribúty, avšak nie je možné vytvoriť atribút výhradne pre produkty patriace do istej skupiny. Atribúty sa môžu vzťahovať buď na jeden konkrétny produkt, alebo na úplne všetky produkty.

Predplatená verzia ponúka možnosť sledovať históriu zmien a nastaviť si upozornenia na nízke zostávajúce množstvo produktu.

Táto aplikácia však nespĺňa veľa požiadaviek kladených na aplikáciu. Nie je možné spravovať domácnosti, pozývať ďalších členov ani vytvárať a spravovať lokácie. Tiež nie je možné rozlišovať jednotlivé kusy produktu, sledovať zostávajúce množstvo produktu ani nastaviť upozornenia na expiráciu kusov.

### 2.3.4 Itemtopia

Itemtopia [5] je mobilná aplikácia pre Android a iOS určená na organizáciu majetku. Táto aplikácia existuje v bezplatnej aj v platenej verzii (220 czk/mesiac).

V bezplatnej verzii ponúka možnosť vytvárať lokácie a podlokácie pod názvom *spaces* a *subspaces*. Do týchto lokácií je možné pozývať ďalších členov a nastaviť im role. Teoreticky by teda bolo možné využiť prvú vrstvu lokácií ako domácnosti, a ich podlokácie chápať ako lokácie danej domácnosti. Pozývanie členov je ale v bezplatnej verzii limitované a pre neobmedzené pridávanie členov je potrebné zakúpiť si predplatné.

Čo sa týka kategórií a podkategórií, v bezplatnej verzii taktiež dochádza k obmedzeniam. Kategórie sú v aplikácii predvytvorené, a pre vytváranie vlastných kategórií je nutné mať platenú verziu. Taktiež neexistuje možnosť vytvárať podkategórie, a to ani v platenej verzii.

Aplikácia má veľmi kvalitné spracovanie funkcionality atribútov a ponúka veľkú variabilitu v ich tvorbe – užívateľ si preto môže atribúty vyrobiť na mieru a prispôsobiť ich svojim požiadavkám. Je možné vytvoriť textové atribúty, číselné atribúty alebo atribúty na dátum a čas. Atribúty sa vytvárajú v rámci kategórií a vzťahujú sa na produkty danej kategórie.

Pridávanie vlastných produktov do lokácie je bez obmedzenia možné až v platenej verzii. Aplikácia neumožňuje rozlíšiteľnosť kusov produktu a sledovanie zostávajúceho množstva produktov.

Čo sa týka upozornení, nie je vyslovene možné vytvoriť upozornenie na nízke množstvo a expiráciu. Aplikácia však umožňuje tvorbu vlastných upozornení prispôsobených potrebám užívateľa. Je možné zvoliť si, na čo konkrétne chce byť užívateľ upozornený, ďalej určiť dátum, čas a periodicitu upozornenia. V aplikácii je možné zobraziť si históriu vykonaných zmien.

### 2.3.5 Home Inventory+

Home Inventory+ [6] je aplikácia, ktorá umožňuje organizovať produkty v domácnosti. Táto aplikácia je určená iba pre operačný systém iOS. Aplikácia má bezplatnú verziu a taktiež ponúka možnosť zakúpenia členstva za symbolickú cenu, 49 czk.

Home Inventory+ ako jedna z mála umožňuje tvorbu lokácií a podlokácií, rovnako aj sledovanie množstva produktov. Taktiež umožňuje vytvárať kategórie a triediť do nich produkty, avšak tvorbu podkategórií už užívateľovi nedovoľuje. Tvorba kategórií a lokácií je v bezplatnej verzii dosť obmedzená, je možné mať iba dve kategórie a dve lokácie.

Čo sa týka ostatných požadovaných funkcionalít, tie už aplikácia neponúka. Nie je možné spravovať a vytvárať domácnosti, ani pozývať ďalších členov. Aplikácia nerozlišuje kusy produktov a neumožňuje ani funkcionalitu jednotiek merania. Užívateľa taktiež neupozorní na expiráciu a nízke množstvo produktu. Neexistuje ani možnosť pridávania vlastných atribútov a sledovania histórie zmien.

### 2.3.6 Záver analýzy existujúcich riešení

Ako je vidieť v tabuľke 2.1, žiadna z porovnávaných aplikácii neponúka funkcionality v takom rozsahu, v akom sú požadované. Ani jedna aplikácia neumožňuje správu energií a úloh, taktiež chýba podpora českého jazyka. Je nutné podotknúť, že žiadna z aplikácií neponúka možnosť rozlíšenia kusov produktu. Aby bolo možné využiť funkcionality aplikácií naplno, je taktiež často potrebné zakúpiť si členstvo.

Z týchto dôvodov je preto žiadúce vyvinúť vlastný systém s požadovanými funkcionalitami.

| | Sortly | Memento Database | Smart Inventory System | Itemtopia | Home Inventory+ | **HouseKeeper** |
|---|---|---|---|---|---|---|
| Správa domácností | x | x | x | ✓ | x | ✓ |
| Pozývanie ďalších členov | ✓* | x | x | ✓* | x | ✓ |
| Správa lokácií | x | x | x | ✓ | ✓* | ✓ |
| Kategorizovanie produktov | ✓ | ✓** | ✓** | ✓** | ✓** | ✓ |
| Správa produktov | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Funkcionalita jednotiek merania | ✓* | x | x | x | x | ✓ |
| Rozlíšiteľnosť kusov | x | x | x | x | x | ✓ |
| Agregácia zostávajúceho množstva | ✓ | x | x | x | ✓ | ✓ |
| Upozornenia na expiráciu | ✓* | x | x | x | x | ✓ |
| Upozornenia na nízke množstvo | ✓ | x | ✓* | x | x | ✓ |
| Pridávanie vlastných atribútov | ✓* | ✓ | ✓** | ✓ | x | ✓ |
| História zmien | ✓ | ✓ | ✓* | ✓ | x | ✓ |
| Správa energií | x | x | x | x | x | ✓ |
| Správa úloh | x | x | x | x | x | ✓ |
| Podpora českého jazyka | x | x | x | x | x | ✓ |
| Webová aplikácia | ✓ | x | ✓* | x | x | ✓ |
| Windows desktopová aplikácia | x | ✓ | x | x | x | x |
| Android aplikácia | ✓ | ✓ | ✓ | ✓ | x | x |
| iOS aplikácia | ✓ | ✓ | x | ✓ | ✓ | x |

**Tabuľka 2.1** Zhrnutie funkcionalít porovnávaných aplikácií (*platená funkcionalita, **obmedzená/neúplná funkcionalita)

## 2.4 Model prípadov použitia

Model prípadov použitia slúži na zachytenie správania systému z pohľadu užívateľa. Na základe tohto modelu je neskôr možné vykonať akceptačné testy, a slúži taktiež k podrobnejšej špecifikácii funkčnosti pre vývoj. Model sa skladá z aktérov, prípadov použitia a z diagramov prípadov použitia. Diagramy prípadov použitia slúžia na prehľadné zobrazenie vzťahov medzi aktérmi a prípadmi použitia. [7]

### 2.4.1 Aktéri

Aktér je rola, ktorú nejaká entita prijíma pri interakcii so systémom. Vo väčšine prípadov je aktér osoba, no aktérom môže byť aj iný systém alebo čas. [8]

V module domáceho inventára aplikácie HouseKeeper sú relevantní traja aktéri – *systém*, *prihlásený užívateľ s právami modul iba čítať* a *prihlásený užívateľ s právami modul upravovať*.

Neprihlásený užívateľ a prihlásený užívateľ, ktorý nemá práva modul čítať ani upravovať, aktérmi nie sú. Je to z toho dôvodu, že nemajú prístup k modulu aplikácie.

### 2.4.2 Prípady použitia

Každý prípad použitia odpovedá cieľu, ktorý je možné v aplikácii dosiahnuť. Nižšie sa nachádza zoznam všetkých prípadov použitia modulu inventára domácnosti.

- **UC1. Zobrazenie kategórií**

  Užívateľ si kliknutím na tlačítko *Kategórie* zobrazí stromový prehľad všetkých kategórií domácnosti. Kliknutím na konkrétnu kategóriu sa zobrazia jej podkategórie a jej detail. Na stránke s detailom kategórie je informácia o zostávajúcom množstve kusov produktov patriacich do tejto kategórie.

- **UC2. Správa kategórií**

  Užívateľ si môže vytvoriť, upraviť alebo vymazať kategórie. Pri vytvorení a úprave kategórie užívateľ vyplní názov a popis. Môže si taktiež zvoliť nadradenú kategóriu, predvolený typ merania a predvolenú jednotku merania. Pri výbere nadradenej kategórie systém zobrazí už spomínaný stromový prehľad všetkých kategórií. Pri zmene nadkategórií systém zobrazí dialógové okno, ktoré upozorňuje na to, že môže dôjsť k zmazaniu niektorých atribútov. Pri pokuse o zmazanie kategórie systém zobrazí dialógové okno, ktoré užívateľa upozorní na to, že po potvrdení dôjde k zmazaniu všetkých podkategórií, produktov kategórie, a kusov daných produktov. Potom, čo užívateľ potvrdí okno, prebehne spomínaný výmaz.

- **UC3. Zobrazenie lokácií**

  Užívateľ si kliknutím na tlačítko *Lokácie* zobrazí stromový prehľad všetkých lokácií danej domácnosti. Kliknutím na konkrétnu lokáciu sa zobrazia jej podlokácie a jej detail.

- **UC4. Správa lokácií**

  Užívateľ si môže vytvoriť, upraviť alebo vymazať lokácie. Pri tvorbe a úprave lokácií užívateľ vyplní názov a popis. Môže si taktiež zvoliť nadradenú lokáciu. Systém pri pokuse o vymazanie lokácie užívateľovi zobrazí dialógové okno s upozornením, že vymazanie spôsobí aj zmazanie všetkých podlokácií. Potom, čo užívateľ potvrdí okno, prebehne spomínaný výmaz.

- **UC5. Zobrazenie definícií atribútov pre produkty**

  Definície atribútov pre produkty si užívateľ zobrazí v detaile kategórie, v sekcii *Atribúty*. Na tejto stránke sa nachádzajú dve tabuľky atribútov. Prvá tabuľka obsahuje priame atribúty danej kategórie, a druhá tabuľka nepriame atribúty, teda atribúty nadkategórií.

- **UC6. Správa definícií atribútov pre produkty**

  Definície atribútov pre produkty si môže užívateľ vytvoriť, upraviť alebo zmazať v detaile kategórie, v sekcii *Atribúty*. Pri tvorbe atribútu vyplní názov, popis, predvolenú hodnotu atribútu a v prípade, že chce aplikovať predvolenú hodnotu aj na už existujúce produkty, zaklikne túto možnosť. Vytvorené atribúty môže užívateľ vymazať, a to aj hromadne.

- **UC7. Vyhľadávanie a zobrazenie produktov**

  Užívateľ si môže vyhľadávať a zobrazovať produkty v prehľadnej tabuľke. Danú tabuľku užívateľ nájde na stránke s prehľadom všetkých produktov, alebo v detaile kategórií, kde sú zobrazené iba produkty patriace do danej kategórie. V tabuľke môže užívateľ filtrovať produkty podľa ich názvu, mernej jednotky, kapacity, atribútov, kategórie a podľa lokácie kusov.

- **UC8. Zobrazenie detailu produktu**

  Užívateľ sa k stránke s detailom produktu dostane z tabuľky produktov, a to po kliknutí na názov produktu, ktorého detail si chce zobraziť. Táto stránka obsahuje všeobecné informácie, ako je popis, typ a jednotka merania, kapacita, množstvo kusov, kategória, predvolená lokácia kusov, spodná hranica rezervy a spodná hranica expirácie. Ďalej obsahuje stránka atribúty produktu, zoznam kusov, atribúty kusov a históriu zmien. Obsahuje i tlačítka na úpravu a vymazanie produktu.

- **UC9. Tvorba produktu**

  Užívateľ sa na stránku s formulárom pre tvorbu produktu dostane buď z prehľadu všetkých produktov, ako aj zo stránky detailu kategórie. Užívateľ pri tvorbe nového produktu vyplní jeho názov a popis, zvolí si kategóriu, do ktorej bude produkt zaradený, typ merania, jednotku, kapacitu produktu a predvolenú lokáciu kusov. Výber kategórie ovplyvní typ a jednotku merania podľa predvolených hodnôt kategórie. Zvolením kategórie systém aplikuje preddefinované atribúty kategórie, pri ktorých užívateľ vyplní hodnoty pre daný produkt. V prípade, že si chce užívateľ nastaviť oznámenia na e-mail pri blížiacej sa expirácii alebo pri nízkom množstve produktu, nastaví si ich zvolením spodnej hranice.

- **UC10. Editácia produktu**

  Užívateľ sa k editácii produktu dostane z detailu produktu, po kliknutí na tlačítko *Upraviť produkt*. Systém užívateľa presmeruje na stránku s formulárom pre úpravu produktu. Editácia prebieha podobne ako tvorba nového produktu, avšak s dvoma rozdielmi. Prvý rozdiel spočíva v tom, že ak má produkt aspoň jeden kus, tak pri pokuse o zníženie kapacity produktu systém zobrazí dialógové okno. Obsah okna upozorňuje na to, že daný akt môže spôsobiť zmenu zostávajúceho množstva kusov. Druhý rozdiel je ten, že ak má produkt nejaké kusy, nie je možné meniť typ merania produktu. V prípade, že dôjde k zmene zostávajúceho množstva kusov, vytvorí systém záznam o tejto zmene.

- **UC11. Vymazanie produktu**

  Užívateľ sa k výmazu produktu dostane z detailu produktu kliknutím na tlačítko *Zmazať produkt*. Po kliknutí systém zobrazí dialógové okno s upozornením na to, že po potvrdení budú zmazané aj všetky kusy tohto produktu. Potom, čo užívateľ potvrdí okno, prebehne spomínaný výmaz.

- **UC12. Zobrazenie definícií atribútov pre kusy**

  Na stránke detailu produktu, v sekcii *Atribúty kusov*, si užívateľ zobrazí atribúty pre kusy daného produktu. Dané atribúty sa nachádzajú v tabuľke.

- **UC13. Správa definícií atribútov pre kusy**

  Na stránke detailu produktu, v sekcii *Atribúty kusov*, môže užívateľ vytvárať, upravovať alebo mazať atribúty pre kusy daného produktu. Pri tvorbe atribútu vyplní názov, popis, predvolenú hodnotu atribútu a v prípade, že chce aplikovať predvolenú hodnotu aj na už existujúce kusy, zaklikne túto možnosť. Vytvorené atribúty užívateľ môže vymazať, a to aj hromadne.

- **UC14. Zobrazenie histórie zmien**

  Užívateľ si v sekcii *História zmien*, ktorá sa nachádza v detaile produktu, zobrazí vykonané zmeny zostávajúceho množstva produktu. V tabuľke histórie zmien sa nachádzajú údaje o dátume a čase zmeny, o počte ovplyvnených kusov, o predchádzajúcom a novom množstve, a taktiež informácie o autorovi zmeny.

- **UC15. Vyhľadávanie a zobrazenie kusov**

  Užívateľ si môže vyhľadávať a zobrazovať kusy v prehľadnej tabuľke. Danú tabuľku je možné nájsť na stránke s prehľadom všetkých kusov, alebo v detaile produktu, kde sú zobrazené iba kusy daného produktu. V tabuľke môže užívateľ filtrovať kusy podľa názvu produktu, popisu kusu, dátumu expirácie, typu a jednotky merania, zostávajúceho množstva, kategórie a lokácie.

- **UC16. Tvorba kusov**

  Užívateľ sa na stránku s formulárom pre tvorbu kusov dostane z prehľadu všetkých kusov, zo stránky detailu produktu alebo zo stránky detailu lokácie. Užívateľ si pri tvorbe nového kusu zvolí, pod aký už existujúci produkt bude patriť nový kus, nastaví si jeho dátum expirácie, vyplní popis, prípadné atribúty od produktu a počet kusov. Taktiež si zvolí jeho lokáciu v domácnosti. Systém po vytvorení kusov vytvorí aj záznam o zmene zostávajúceho množstva produktu.

- **UC17. Editácia kusu**

  Užívateľ sa na stránku s formulárom pre úpravu kusu dostane z tabuľky kusov po kliknutí na ikonku ceruzky v stĺpci *Akcie*. Užívateľ môže meniť zostávajúce množstvo kusu, dátum expirácie, popis, atribúty a lokáciu. Čo však meniť nemôže, je produkt. Systém v prípade zmeny zostávajúceho množstva vytvorí záznam o zmene zostávajúceho množstva produktu.

- **UC18. Vymazanie kusov**

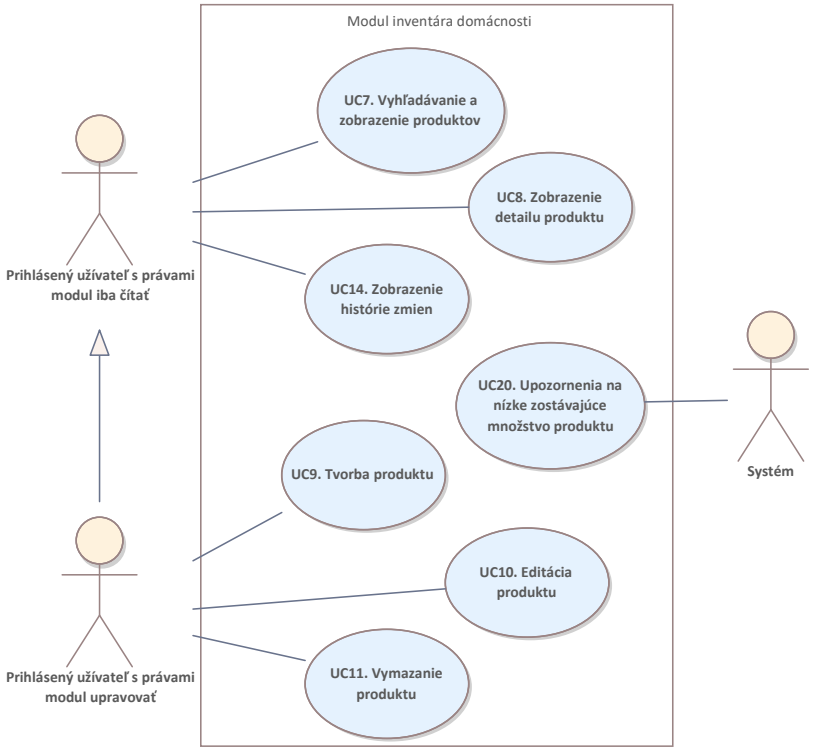
  Užívateľ si v tabuľke kusov označí jeden či viacero kusov, ktoré chce zmazať. Po kliknutí na tlačítko *Zmazať vybrané* a po potvrdení dialógu systém označené kusy vymaže a vytvorí záznam o zmene zostávajúceho množstva produktu.

- **UC19. Upozornenia na expiráciu kusov**

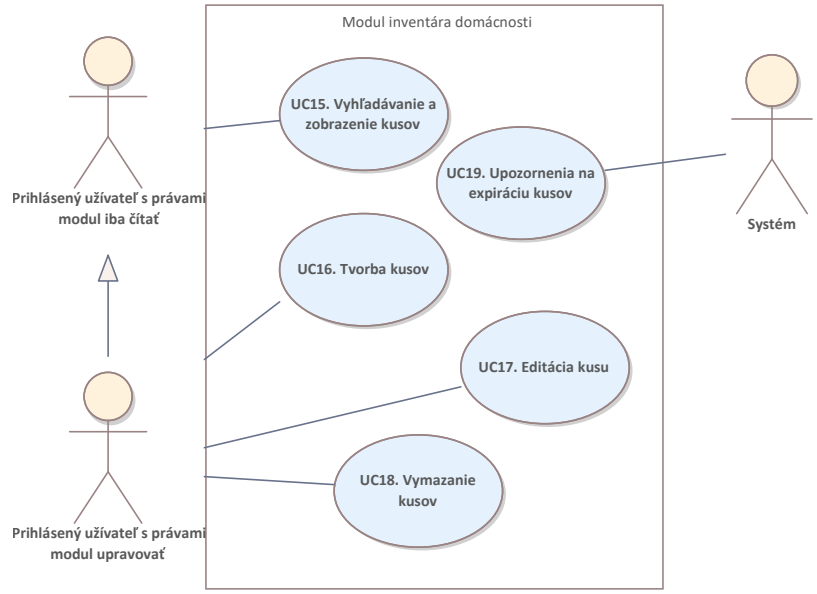
  Systém každý deň v čase 00:00 odošle e-mail s upozornením na expiráciu kusov všetkým členom domácnosti, ktorí majú práva na úpravu modulu inventára domácnosti. Súčasťou upozornenia sú tie kusy produktov, ktorých počet dní do expirácie je nižší, než ich nastavená spodná hranica expirácie.

- **UC20. Upozornenia na nízke zostávajúce množstvo produktu**

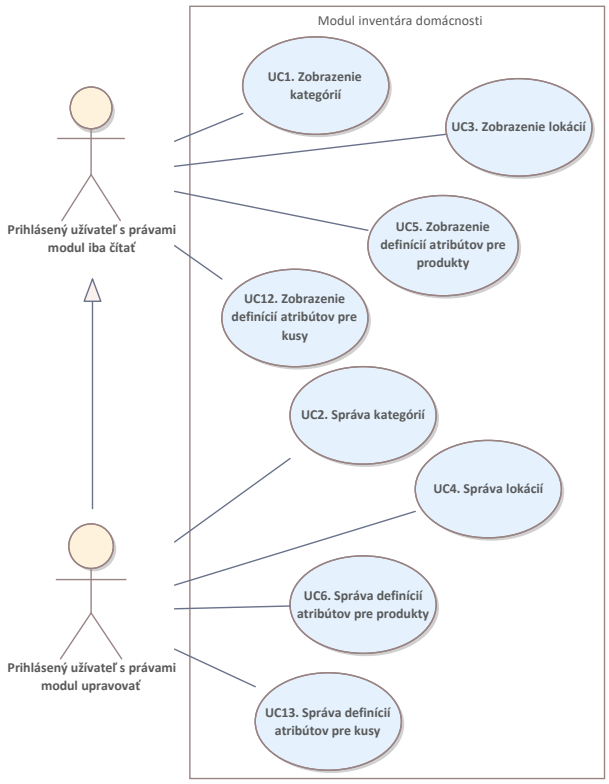
  Systém každý deň v čase 00:00 odošle e-mail s upozornením na nízke zostávajúce množstvo produktu všetkým členom domácnosti, ktorí majú práva na úpravu modulu inventára domácnosti. Súčasťou upozornenia sú tie produkty, ktorých zostávajúce množstvo produktu je nižšie, než ich stanovená spodná hranica.




**Obr. 2.2** Diagram prípadov použitia produktov




**Obr. 2.3** Diagram prípadov použitia kusov




**Obr. 2.4** Diagram ostatných prípadov použitia

### 2.4.3 Pokrytie funkčných požiadaviek prípadmi použitia

Tabuľka 2.2 zobrazuje pokrytie funkčných požiadaviek jednotlivými prípadmi použitia. Týmto sa overí, že každý prípad použitia je potrebný, a nevynechal sa žiaden relevantný funkčný požiadavok. Jediné vynechané funkčné požiadavky boli tie, ktoré sa týkali jadra aplikácie.

## 2.5 Doménový model

Doménový model zachytáva sieť vzájomne prepojených objektov, pričom každý objekt stelesňuje entitu s určitým významom [9]. Tento model slúži na prehľadné zachytenie pojmov, vzťahov a pravidiel v určitej doméne, neskôr slúži ako východisko pre implementáciu. Obrázok 2.5 reprezentuje doménový model inventára domácnosti v aplikácii HouseKeeper.

- **Domácnosť**

  Domácnosť je základná stavebná jednotka aplikácie a je súčasťou jadra. Domácnosť predstavuje priestor, v ktorom prebiehajú všetky úkony. V domácnosti sa môže nachádzať ľubovoľný počet členov. Pozývať nových členov a upravovať údaje o domácnosti môže iba jej majiteľ.

- **Členstvo**

  Členom domácnosti sa môže stať užívateľ buď založením svojej novej domácnosti, alebo prijatím pozvánky od majiteľa inej domácnosti.

| | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 |
|---|---|---|---|---|---|---|---|---|---|---|
| UC1 | ✓ | | | | | | | | ✓ | |
| UC2 | ✓ | | | | | | | | | |
| UC3 | | ✓ | | | | | | | | |
| UC4 | | ✓ | | | | | | | | |
| UC5 | | | ✓ | | | | | | | |
| UC6 | | | ✓ | | | | | | | |
| UC7 | | | | ✓ | | ✓ | | | | |
| UC8 | | | | ✓ | | | | | ✓ | |
| UC9 | | | ✓ | ✓ | | | ✓ | ✓ | | |
| UC10 | | | ✓ | ✓ | | | ✓ | ✓ | | |
| UC11 | | | | ✓ | | | | | | |
| UC12 | | | ✓ | | | | | | | |
| UC13 | | | ✓ | | | | | | | |
| UC14 | | | | | | | | | | ✓ |
| UC15 | | | | | ✓ | ✓ | | | | |
| UC16 | | | ✓ | | ✓ | | | | | ✓ |
| UC17 | | | ✓ | | ✓ | | | | | ✓ |
| UC18 | | | | | ✓ | | | | | ✓ |
| UC19 | | | | | | | ✓ | | | |
| UC20 | | | | | | | | ✓ | | |

**Tabuľka 2.2** Tabuľka zobrazujúca pokrytie funkčných požiadaviek prípadmi použitia

- **Užívateľ**

  Užívateľ môže byť členom akéhokoľvek počtu domácností. Užívateľ musí byť pre realizáciu úkonov v aplikácii HouseKeeper registrovaný a prihlásený. Považuje za súčasť jadra.

- **Práva k modulom**

  K členstvu sa vzťahujú rôzne užívateľské práva na jednotlivé moduly aplikácie. Člen môže mať v súvislosti s daným modulom právo modul čítať, alebo aj upravovať. V prípade, že užívateľ nemá k danému modulu žiadne práva, tak k nemu nemá ani prístup.

- **Kategória**

  Kategórie slúžia na zoskupenie produktov a môžu mať nadkategórie i podkategórie. Každá kategória má svoj názov, popis, predvolenú jednotku produktov a predvolený typ produktov. Kategórie môžu mať neobmedzený počet definícií atribútov, pričom tieto atribúty budú aplikované na všetky produkty tejto kategórie i podkategórií.

- **Produkt**

  Základnou stavebnou jednotkou inventára domácnosti je produkt. Každý produkt má svoj názov, popis, kapacitu, typ merania a jednotku merania. Všetky produkty sú zaradené do nejakej kategórie. Taktiež majú atribúty podľa toho, v akej kategórii sa nachádzajú. Všetky produkty môžu mať kusy. Produkty môžu mať svoje nastavené upozornenia na nízke zostávajúce množstvo produktu a na expiráciu kusov. Produkty môžu mať neobmedzený počet definícií atribútov, pričom tieto atribúty budú aplikované na všetky kusy tohto produktu.

- **Upozornenie**

  Každé upozornenie sa viaže práve k jednému produktu. Existujú dva typy upozornení, a tými sú upozornenie na nízke zostávajúce množstvo produktu a upozornenie na expiráciu kusov. Každé upozornenie má spodnú hranicu, pri prekročení ktorej dôjde k upozorneniu užívateľa.

- **Definícia atribútu**

  Každá definícia atribútu má názov, popis a predvolenú hodnotu. Definícia atribútu má vzťah buď ku kategórii alebo k produktu. Ak sa vzťahuje ku kategórii, je aplikovaná na všetky produkty tejto kategórie. Ak sa vzťahuje k produktu, bude aplikovaná na všetky kusy tohto produktu.

- **Hodnota atribútu**

  Hodnota atribútu má vzťah buď k produktu alebo kusu. Ku každej hodnote existuje definícia atribútu, ktorá nesie jeho názov.

- **Kus**

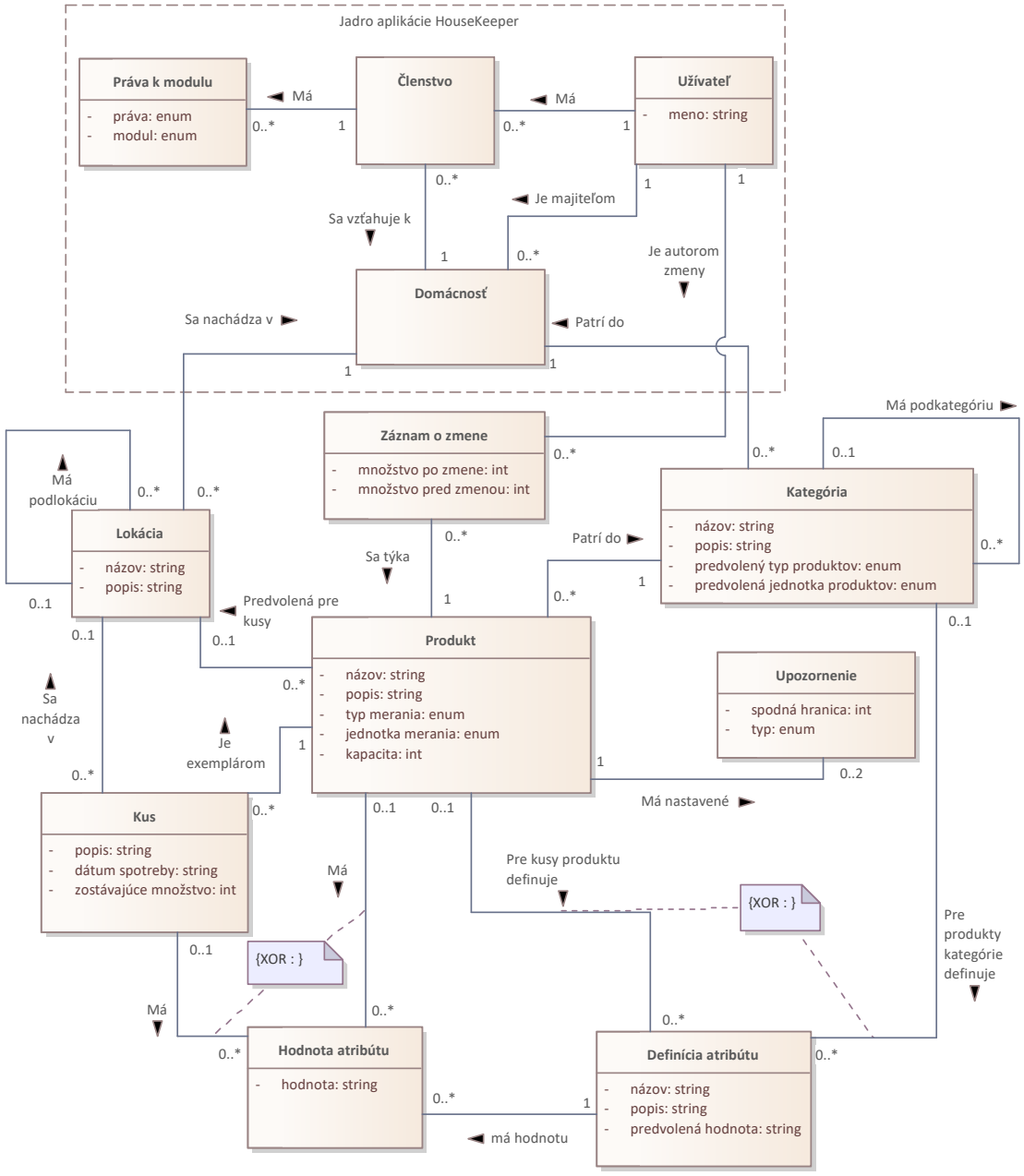
  Každý kus je exemplárom práve jedného produktu. U kusov sa eviduje popis, zostávajúce množstvo, prípadne dátum expirácie. Každý kus sa môže nachádzať v nejakej lokácii. Kusy majú atribúty podľa toho, pod aký produkt patria.

- **Lokácia**

  Lokácie slúžia na zoskupenie kusov, pričom môžu mať nadlokácie i podlokácie. Každá lokácia môže mať svoj názov a popis.

- **Záznam o zmene**

  Záznam o zmene je jednoduchá štruktúra, ktorá obsahuje údaj o množstve pred jeho zmenou a údaj o množstve po danej zmene. Taktiež obsahuje informáciu o autorovi zmeny.




**Obr. 2.5** Doménový model inventára domácnosti v aplikácii HouseKeeper

# Kapitola 3
# Návrh

Táto kapitola sa zaoberá návrhom jadra a modulu inventára domácnosti systému HouseKeeper. Dôraz je kladený na návrh serverovej časti aplikácie, pretože návrh klientskej časti má na starosti kolega Dávid Jenčo vo svojej bakalárskej práci.

Súčasťou tejto kapitoly je opis použitých technológií, teda programovacieho jazyka a frameworku, pričom je taktiež obsiahnuté porovnanie frameworkov. Ďalej sa kapitola zaoberá perzistenciou dát s návrhom databázového modelu a návrhom architektúry aplikácie HouseKeeper. Súčasťou kapitoly je aj návrh riešenia prípadu použitia na výmaz viacerých kusov, ktorý obsahuje diagram tried a sekvenčné diagramy.

Modul inventára domácnosti bude používať rovnaké technológie, databázu a architektúru ako jadro, preto bude ľahko integrovateľný do jadra systému HouseKeeper.

## 3.1 Programovací jazyk

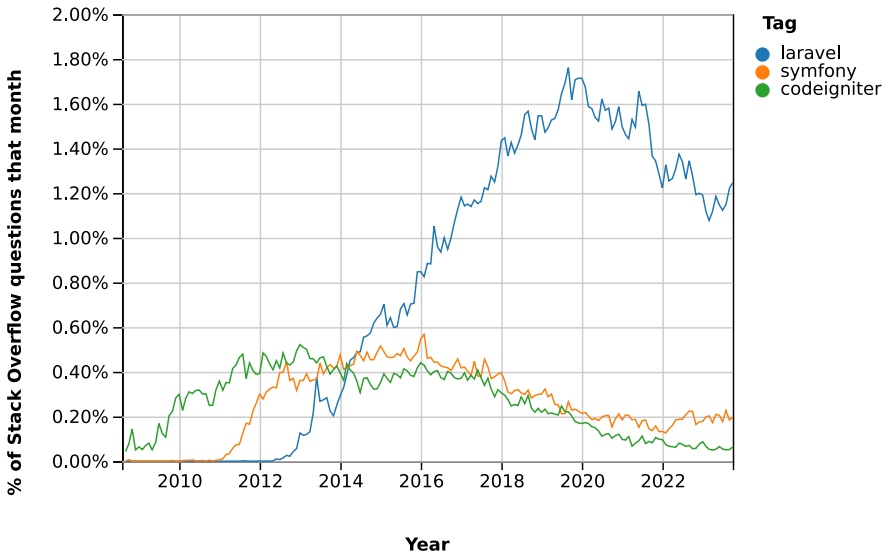
Z nefunkčných požiadaviek v sekcii 2.2.2 plynie, že programovacím jazykom serverovej časti musí byť jazyk PHP. PHP [10] je jeden z najviac používaných programovacích jazykov na svete. Prvá verzia jazyka vznikla v roku 1994 a za jej vytvorením stojí dánsko-kanadský programátor, Rasmus Lerdorf. PHP je rekurzívna skratka pre *PHP: Hypertext Preprocessor*. Je to dynamický, slabo typovaný skriptovací programovací jazyk, ktorý umožňuje objektovo orientované programovanie. Syntax jazyka je inšpirovaná jazykom Perl a podporuje jednoduchú integráciu so značkovacím jazykom HTML. Jazyk sa stále vyvíja, pravidelne vychádzajú nové verzie. Aplikácia HouseKeeper bude používať jednu z najnovších verzií PHP, ktorou je verzia 8.2.

Kolega Dávid Jenčo mal v klientskej časti slobodu výberu jazyku, kde bol vybraný jazyk Typescript [11].

## 3.2 Porovnanie frameworkov

*Framework* je sada znovu použiteľných tried a funkcionalít, ktoré značne uľahčujú vývoj implementovaním abstraktných funkcionalít a určením architektúry. [12]

Táto sekcia sa venuje porovnaniu a následnému výberu frameworku serverovej časti aplikácie. Keďže jazykom serverovej časti je jazyk PHP, porovnanie sa sústredí iba na jeho frameworky. Porovnávanie kladie dôraz na popularitu frameworku medzi programátormi, modernosť, dokumentáciu a podporu tvorby REST API. Taktiež prihliada na učiacu krivku, pretože na aplikácii budú pracovať i ďalší študenti pri rozširovaní tejto aplikácie. Samozrejme, rolu hrá aj subjektivita autora. Porovnávané frameworky sú Symfony, Laravel a CodeIgniter.




**Obr. 3.1** Graf vývoja popularity porovnávaných frameworkov na známom fóre Stack Overflow [13]

### 3.2.1 Symfony

Symfony [14] je PHP open source framework vydaný pod licenciou MIT. Prvá verzia tohto frameworku vznikla v roku 2005.

Symfony má dobrú komunitu a dokumentáciu na vysokej úrovni. Na základe dotazníka od Stack Overflow sa v súčasnosti jedná o druhý najpopulárnejší PHP framework [15]. Symfony má bohužiaľ vysokú učiacu krivku a neponúka dostatočne vhodné možnosti na tvorbu REST API. Populárnou, ale už zastaralou možnosťou na tvorbu REST API v ekosystéme Symfony je balíček FOSRestBundle a API Platform. API Platform je skvelý pre jednoduché aplikácie, no keď sa stane aplikácia zložitejšou, je náročnejšie ju potom rozširovať. Okrem toho je dokumentácia API Platform celkom zastaralá, čo ho činí nevhodnou pre požiadavky aplikácie HouseKeeper.

### 3.2.2 Laravel

Laravel [16] je moderný PHP framework, ktorý vyvinul programátor Taylor Otwell v roku 2012. Znovu sa jedná o open source framework poskytovaný pod licenciou MIT. Podľa porovnaní od Stack Overflow [15] sa jedná o najpopulárnejší PHP framework.

Laravel je postavený na frameworku Symfony, a abstrahuje veľké množstvo jeho funkcionalít. Kvôli tomu je učiaca krivka nižšia. Laravel ponúka mnoho výhod, ako je veľká komunita, či dokumentácia na naozaj veľmi vysokej úrovni. Ponúka aj natívnu podporu tvorby REST API a serializáciu dát bez nutnosti inštalácie ďalších balíčkov. Tvorba REST API je rýchla a ľahko rozšíriteľná, jednoducho sa pridáva komplexnejšia logika. Veľká výhoda Laravelu je aj možnosť využiť predpripravenú implementáciu funkcionalít – starter kit. Napríklad, starter kit *Breeze* už obsahuje plne funkčnú autentizáciu aj v podobe REST API, čo pri vývoji ušetrí čas.

### 3.2.3 CodeIgniter

CodeIgniter [17] je open source framework vyvíjaný americkou spoločnosťou EllisLab. Jeho prvá verzia bola vydaná v roku 2006.

Jedná sa o framework s najmenšou učiacou krivkou z vyššie spomínaných frameworkov, čo je jednoznačne pozitívom. Oproti Symfony a Laravelu je CodeIgniter oveľa menší framework. Svojou jednoduchosťou tak dosiahne lepší výkon z hľadiska rýchlosti. Je navrhnutý tak, aby bol jasný a zrozumiteľný aj pre začiatočníkov. Avšak fakt, že je menší, môže byť aj záporom – veľké množstvo základných funkcionalít je nutné implementovať. Je vhodný skôr na malé projekty, a z tohto dôvodu je nepostačujúci pre potreby vývoja aplikácie HouseKeeper.

### 3.2.4 Záver porovnania a výber

Po porovnaní vyššie spomínaných frameworkov bol zvolený Laravel, pretože spĺňa všetky stanovené požiadavky. Ako je možné vidieť aj na Obrázku 3.1, Laravel od svojho vzniku v roku 2012 postupne naberal na popularite. V súčasnosti je z porovnávaných frameworkov najviac používaný. Tým sa môže považovať výber Laravelu za bezpečnú voľbu.

V práci Dávida Jenča, na klientskej časti aplikácie, bolo vykonané podobné porovnanie frameworkov, ako v tejto práci. Vybraný framework klientskej časti bol *React*. [11]

## 3.3 Perzistencia dát

Z funkčných požiadaviek v sekcii 2.2.1 jednoznačne vyplýva, že aplikácia musí isté množstvo údajov perzistovať na strane serveru na to, aby tieto požiadavky splnila. V dnešnej dobe je štandardom uchovávať dáta v databáze. V aplikácii HouseKeeper sa preto budú dáta ukladať, aby bolo umožnené správne fungovanie správy domácností, kategórií, lokácií, atď.

### 3.3.1 Výber databázy

Pri výbere typu databázy je najprv nutné určiť, či bude typom databázy NoSQL databáza, teda nerelačná databáza, alebo klasická, relačná databáza.

NoSQL databázy sa vyznačujú tým, že nemusia mať pevnú štruktúru dát, čo umožňuje veľkú flexibilitu a pružnosť. Vynikajú najmä svojou škálovateľnosťou. Dátové modely môžu byť rôzne, od jednoduchých modelov kľúč-hodnota, až po komplexné grafové modely. Vďaka tomu sú vhodné pre širokú škálu aplikácií, hlavne pre tie, ktoré pracujú s veľkými množstvami neštruktúrovaných dát. Nevýhodou týchto databáz je už ale horšia práca s relačnými dátami, a zároveň fakt, že sa jedná o pomerne novú technológiu, teda nie tak overenú v praxi, ako sú relačné databázy. [18]

V prípade aplikácie HouseKeeper je ale štruktúra dát pevná a relačná, preto je možné použiť relačnú databázu. Tento typ databázy bol nakoniec vybraný kvôli chýbajúcej natívnej podpore NoSQL databáz vo frameworku Laravel. Rolu taktiež hrali preferencie autora práce, kolegu Dávida Jenča a vedúceho práce. Schéma dát bude tým pádom predurčená, a databáza bude dáta ukladať do preddefinovaných tabuliek. K dotazovaniu nad dátami sa bude používať jazyk SQL. Výhodou relačných databáz je hlavne ich konzistencia, čo znamená že každý dotaz nad dátami vráti vždy najaktuálnejšie dáta [18].

Relačných databáz existuje niekoľko, aplikácia bude ale používať databázu PostgreSQL. PostgreSQL [19] je open source databáza vyvíjaná už od roku 1986, ktorá má veľmi silnú reputáciu, skvelú komunitu a veľké množstvo funkcionalít oproti iným relačným databázam. Je potrebné dodať, že ak by v budúcnosti nastala situácia, pri ktorej by boli funkcionality databázy nepostačujúce, bolo by možné ju jednoducho vymeniť za inú relačnú databázu vďaka *objektovo relačnému mapovaniu*.

### 3.3.2 Objektovo relačné mapovanie

Objektovo relačné mapovanie (ORM) slúži na prevod dát medzi objektovo orientovanými programovacími jazykmi a relačnými databázami. Umožňuje tak vývojárom namiesto písania zložitých dotazov SQL komunikovať s databázou pomocou vysoko-úrovňového programovacieho jazyka. Tým sa abstrahuje od interakcií s databázou, a kód sa stane ľahšie udržiavateľným a viac čitateľným. Mapovanie spravidla prebieha tak, že sa mapujú databázové tabuľky na triedy, a riadky na objekty. [20]

Framework Laravel má už svoje vstavané ORM, pod názvom *Eloquent*. Eloquent podporuje niekoľko databáz, ako už spomínaný PostgreSQL, ale aj MySQL, MariaDB a ďalšie. Poskytuje jednotné rozhranie pre všetky tieto podporované databázy. Tým pádom je možné databázu vymeniť iba zmenou odpovedajúceho riadku v konfiguračnom súbore, bez nutnosti zasahovať do kódu. [21]

Eloquent ORM je postavený na návrhovom vzore *active record* [22]. Active record je návrhový vzor, v ktorom objekt zapuzdruje dáta reprezentované riadkom v tabuľke databázy. Objekt obsahuje logiku na prístup k dátam, aj na ich úpravu či tvorbu. Pomocou tohto objektu je možne manipulovať priamo s celou tabuľkou, teda i vyhľadávať a mazať iné riadky v nej. Táto vlastnosť je svojou jednoduchosťou výhodou najmä v malých aplikáciach, no vo väčších aplikáciách môže spôsobovať problémy. Je to kvôli tomu, že môže ľahko dôjsť k porušeniu zodpovedností manipuláciou dát na nevhodnom mieste v kóde. [9]

Ďalšou charakteristikou návrhového vzoru active record je to, že dané objekty obsahujú i biznis logiku. To však v prípade aplikácie HouseKeeper nebude pravda. Biznis logiku bude mať na starosti biznis vrstva, a nie tieto objekty, ktoré budú súčasťou dátovej vrstvy. Použitie týchto objektov bude teda skôr pripomínať návrhový vzor *row data gateaway*, ktorý sa líši od active record len tým, že objekty na prístup k dátam neobsahujú biznis logiku. Tým sa zníži komplexita a bude možné biznis logiku testovať zvlášť. [9]

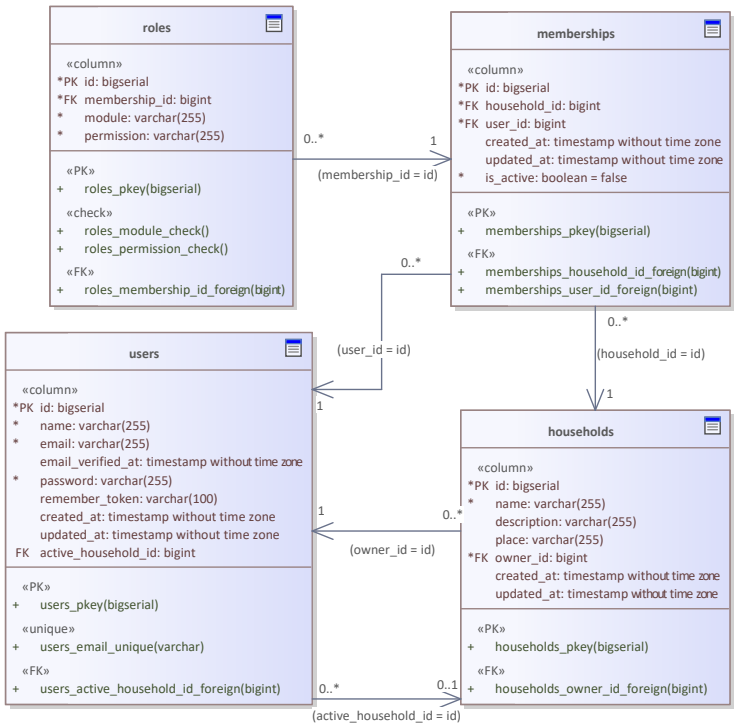
### 3.3.3 Návrh databázového modelu

Keďže aplikácia bude používať relačnú databázu s pevnou štruktúrou dát, je nutné navrhnúť ich štruktúru. Návrh databázového modelu zachycujúci štruktúru dát a ich vzťahy sa nachádza na obrázkoch 3.2 a 3.3. Tento návrh priamo vychádza z doménového modelu v podkapitole 2.5.

Pri procese tvorby návrhu bolo nutné rozhodnúť, ako budú ukladané hodnoty kapacity produktu a zostávajúceho množstva kusov. Najväčšou prekážkou bolo to, že aplikácia musí podporovať ukladanie týchto hodnôt v rôznych jednotkách, a taktiež ich musí byť schopná agregovať. Preto sa zvolilo riešenie, v ktorom sa budú dané hodnoty ukladať v najmenšej jednotke daného typu merania. Napríklad pre objem sa budú hodnoty ukladať v mililitroch, a pre dĺžku zase v centimetroch. Ku každému produktu sa bude taktiež ukladať typ merania a jednotka merania. Týmto spôsobom bude síce nutné vždy jednotky premieňať pred ich zobrazením užívateľovi, no agregácia bude jednoduchá a priamočiara. Medzi ďalšie výhody taktiež patrí bezproblémové radenie podľa kapacity alebo množstva pre istý typ merania, keďže všetky hodnoty sú pre daný typ v rovnakej jednotke.

Zaujímavosťou je, že produkt nemusí mať kapacitu, teda že kapacita produktu môže byť *null*. To poskytne možnosť pridávania abstraktných produktov, ako je *piesok* alebo *vajce*. Využitie môže byť napríklad vtedy, keď si užívateľ nechce rozdeľovať produkty podľa veľkosti obalu, a stačí mu iba jeden produkt pre všetky možné kapacity. Tak bude mať užívateľ iba jeden produkt, *piesok*, a každý kus bude predstavovať jeden konkrétny obal, pričom obaly môžu mať rôzne veľkosti.


**Obr. 3.2** Databázový model modulu inventára domácnosti




**Obr. 3.3** Databázový model jadra

## 3.4 Architektúra

Architektúra aplikácie určuje rozdelenie aplikácie do určitých logických celkov. Vhodne zvolená architektúra uľahčuje rozšíriteľnosť a udržiavateľnosť aplikácie v budúcnosti, i počas vývoja. Je dôležité poznamenať, že je možné aplikovať niekoľko architektúr v aplikácii zároveň. Dôvodom je, že každá architektúra reprezentuje inú úroveň pohľadu na aplikáciu. V tejto kapitole bude opísaná architektúra serverovej a klientskej časti, rovnako ako aj celej aplikácie.

### 3.4.1 Serverová časť

Jednou z najznámejších architektúr vo svete softvérového vývoja je *trojvrstvová* architektúra[1]. Ako vyplýva z názvu, v tejto architektúre je aplikácia rozdelená do troch vrstiev – do prezentačnej, biznisovej a dátovej vrstvy. Každá z týchto vrstiev ma iné zodpovednosti. [9]

Prvou vrstvou je prezentačná vrstva, ktorá má na starosti zobrazenie informácii užívateľovi, spracovanie užívateľského vstupu, prípadne predanie tohto vstupu ďalej do ďalších vrstiev. V kontexte serverovej časti aplikácie bude za prezentačnú vrstvu brané REST API. Ďalšia, biznisová vrstva, sa zaoberá všetkou logikou, ktorú musí aplikácia uskutočniť v rámci danej domény. Zahrnuté sú v nej všetky zložitejšie procesy a výpočty s dátami na základe užívateľského vstupu predaného z prezentačnej vrstvy. Poslednou, najnižšou vrstvou, je dátová vrstva, ktorá má na starosti ukladanie a vyhľadávanie dát. [9]

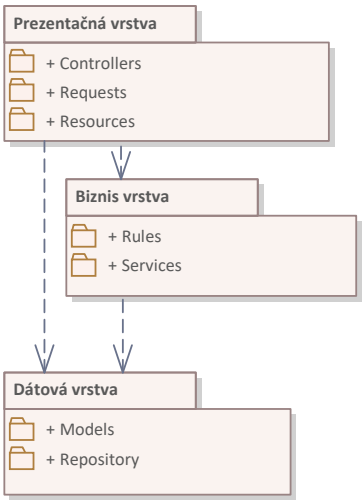
Ďalšou charakteristikou trojvrstvovej architektúry je to, ako sú jednotlivé vrstvy na sebe závislé. Vo svojej pôvodnej podobe trojvrstvová architektúra požadovala, aby prezentačná vrstva závisela iba na biznisovej vrstve, a biznisová vrstva iba na dátovej. Táto podmienka sa ale používa najmä v obrovských systémoch veľkých podnikov, a preto sa môže zdať v súvislosti

[1] Anglicky – three-layer architecture (jedná sa o rozdelenie na logickej úrovni)

s aplikáciami menších veľkostí až príliš prísna. Preto sa v praxi taktiež používa voľnejšia varianta tejto architektúry, ktorá povoľuje aj závislosť prezentačnej vrstvy na dátovej vrstve. [9]

Aplikácia HouseKeeper bude používať relaxovanú variantu trojvrstvovej architektúry. Pomocou tejto architektúry bude zaistená ľahká rozšíriteľnosť aplikácie aj v budúcnosti. Diagram zobrazujúci závislosti prezentačnej, biznisovej a dátovej vrstvy je možné vidieť na obrázku 3.4. Na diagrame je taktiež možné vidieť, aké typy tried budú patriť do jednotlivých vrstiev.

Ďalším komplementárnym pohľadom k trojvrstvovej architektúre je architektúra *MVC*. Na tejto architektúre je postavených mnoho webových frameworkov. Medzi nimi sú aj v podkapitole 3.2 spomínané frameworky Symfony, CodeIgniter, a zvolený Laravel. Táto architektúra sa skladá z troch častí, a to z modelu (M), view (V) a controlleru (C). View má na starosti vykreslenie rozhrania užívateľovi. Serverová časť má ako rozhranie REST, tým pádom sa za view pokladá JSON[2]. Controller má na starosti spracovanie užívateľského vstupu a taktiež určuje, aké rozhranie sa má užívateľovi zobraziť. Tieto dve časti sú z pohľadu trojvrstvovej architektúry súčasťou prezentačnej vrstvy. Posledná časť, model, má na starosti doménovú logiku aplikácie, spolu so správou dát. To zahŕňa vyhľadávanie, úpravu a ukladanie dát. Model je preto z pohľadu trojvrstvovej architektúry súčasťou biznis a dátovej vrstvy. [9]




**Obr. 3.4** Závislosti vrstiev v relaxovanej trojvrstvovej architektúre

### 3.4.2 Klientská časť

Návrh architektúry klientskej časti aplikácie uskutočnil kolega Dávid Jenčo. Podrobnejší opis tejto architektúry je preto možné nájsť v jeho práci. Klientská časť aplikácie bude postavená na princípoch architektúry *Flux* s drobnými rozdielmi.

Flux [23] je zložený zo 4 častí. Prvou časťou je *store*, ktorý obsahuje údaje o aktuálnom stave aplikácie, i logiku k jej zmene. Nasledujúcou časťou je *view*, ktorá má na starosti zobrazenie rozhrania užívateľovi na základe stavu, a spracovanie vstupu. V neposlednom rade je súčasťou *action*, alebo akcia, ktorá sa vytvára po interakcii užívateľa s view. Zmyslom tejto komponenty je špecifikovanie, ako sa má zmeniť určitý store. Akcie sú po vytvorení predané ďalšej časti, ktorá sa volá *dispatcher*. Táto časť rozhodne, k akému store má danú akciu doručiť. V danom store sa potom na základe doručenej akcie zmení stav. O zmene stavu sa neskôr dozvie view, ktorý upraví užívateľské rozhranie na základe nových dát z upraveného stavu.

[2] JSON – JavaScript Object Notation

Aplikácia HouseKeeper bude ale k spravovaniu stavu klientskej časti využívať knižnicu *Redux* [24], čím sa mierne zmení architektúra. Hlavným rozdielom bude to, že store v aplikácii bude iba jeden. K tomuto store má potom prístup celá aplikácia. To zjednodušuje hľadanie chýb a rozširovanie aplikácie. Ďalšou odlišnosťou bude to, že stav aplikácie sa v tejto upravenej architektúre nebude meniť v store, ale v inej časti architektúry, známej ako *reducer*. Keďže bude store v tejto architektúre iba jeden, existencia spomínaného komponentu dispatcher už nie je potrebná. Akcie sa tým pádom predajú priamo časti reducer. Store takto obsahuje iba stav aplikácie, čím je zabezpečené lepšie oddelenie zodpovedností jednotlivých častí.

### 3.4.3 Architektúra aplikácie ako celku

Na záver je potrebné sa pozrieť i na architektúru celej aplikácie a na to, ako budú jednotlivé časti spolu komunikovať. V tomto prípade sa zase bude jednať o trojvrstvovú architektúru[3], kde ale dané vrstvy nebudú určené logickým rozdelením aplikácie, ale rozdelením fyzickým. V danom kontexte budú jednotlivými vrstvami klientská časť, serverová časť a databáza, ktorá je podrobnejšie opísaná v podkapitole 3.3.

Vstupné rozhranie pre užívateľa bude v aplikácii pochopiteľne tvoriť klientská časť. Tá bude mať na starosti správne zobrazenie užívateľského rozhrania a spracovanie vstupu. Klientská časť ale z bezpečnostných dôvodov nemá priamy prístup k dátam uloženým v databáze. Prístup k perzistovaným dátam dosiahne kontaktovaním serverovej časti prostredníctvom jeho rozhrania REST, a to pomocou HTTP požiadavku. Serverová časť spracuje túto HTTP požiadavku a zostaví SQL dotaz na prípadné získanie alebo úpravu dát, ktorý pošle databáze. Databáza získa alebo upraví dáta na základe zostaveného dotazu, ktoré pošle serverovej časti. Tá ich potom spracuje, a v prípade potreby ich priloží k HTTP odpovedi odoslanej klientskej časti. Klientská časť tak na základe odpovedi prekreslí rozhranie a pripraví sa na spracovanie ďalšieho vstupu od užívateľa.

Na obrázku 3.5 je možné vidieť diagram nasadenia aplikácie HouseKeeper. Klientská a serverová časť aplikácie budú nasadené spoločne na jednom webovom serveri. Po prvej HTTP požiadavke od prehliadača užívateľa pošle webový server užívateľovi klientskú časť. Klientská časť bežiaca v prehliadači užívateľa potom bude pomocou protokolu HTTP a rozhrania REST komunikovať so serverovou časťou aplikácie.

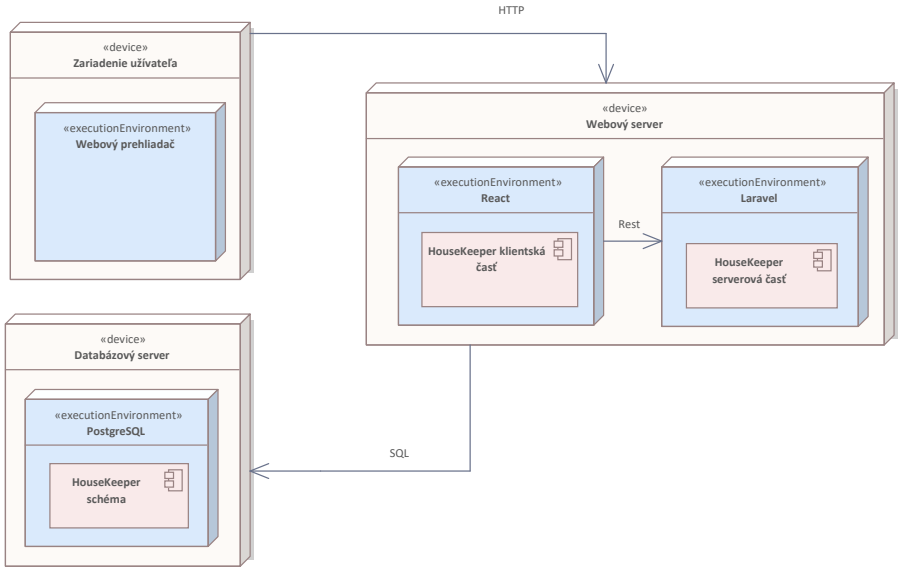
## 3.5 Návrh riešenia výmazu viacerých kusov

V tejto podkapitole sa nachádza návrh tried a ich komunikácie. Tento návrh sa sústredí na realizáciu prípadu použitia výmazu viacerých kusov zároveň, spomínaný v sekcii 2.4.2. Pomocou tohto návrhu je možné prehľadne vysvetliť zodpovednosti jednotlivých typov tried, ktoré budú používané v aplikácii. Návrhy riešení pre iné prípady použitia by vyzerali podobne, no bolo by príliš náročné vykonať takýto detailný návrh pre každý z nich.

### 3.5.1 Návrh tried

Diagram 3.6 zobrazuje návrh tried serverovej časti. Klientská časť nebude obsahovať žiadne triedy, no pre úplnosť zobrazuje diagram 3.7 návrh komponentov a typov objektov. Inšpiráciou návrhu tried serverovej časti bol vybraný framework Laravel, ktorý ponúka rozhrania pre niektoré balíčky tried. Triedy v tomto diagrame obsahujú iba metódy, či atribúty, ktoré sú nutné k realizácii výmazu viacerých kusov. Iné metódy, atribúty a taktiež i triedy implementované Laravelom sú pre prehľadnosť diagramu vynechané. Pred názvom každej triedy sa nachádza názov

[3] Anglicky – three-tier architecture (jedná sa o rozdelenie na fyzickej úrovni)




**Obr. 3.5** Diagram nasadenia aplikácie HouseKeeper

balíčku, do ktorého daná trieda patrí. V diagrame 3.4 je možné vidieť vzťahy medzi jednotlivými balíčkami a vrstvami v trojvrstvovej architektúre.

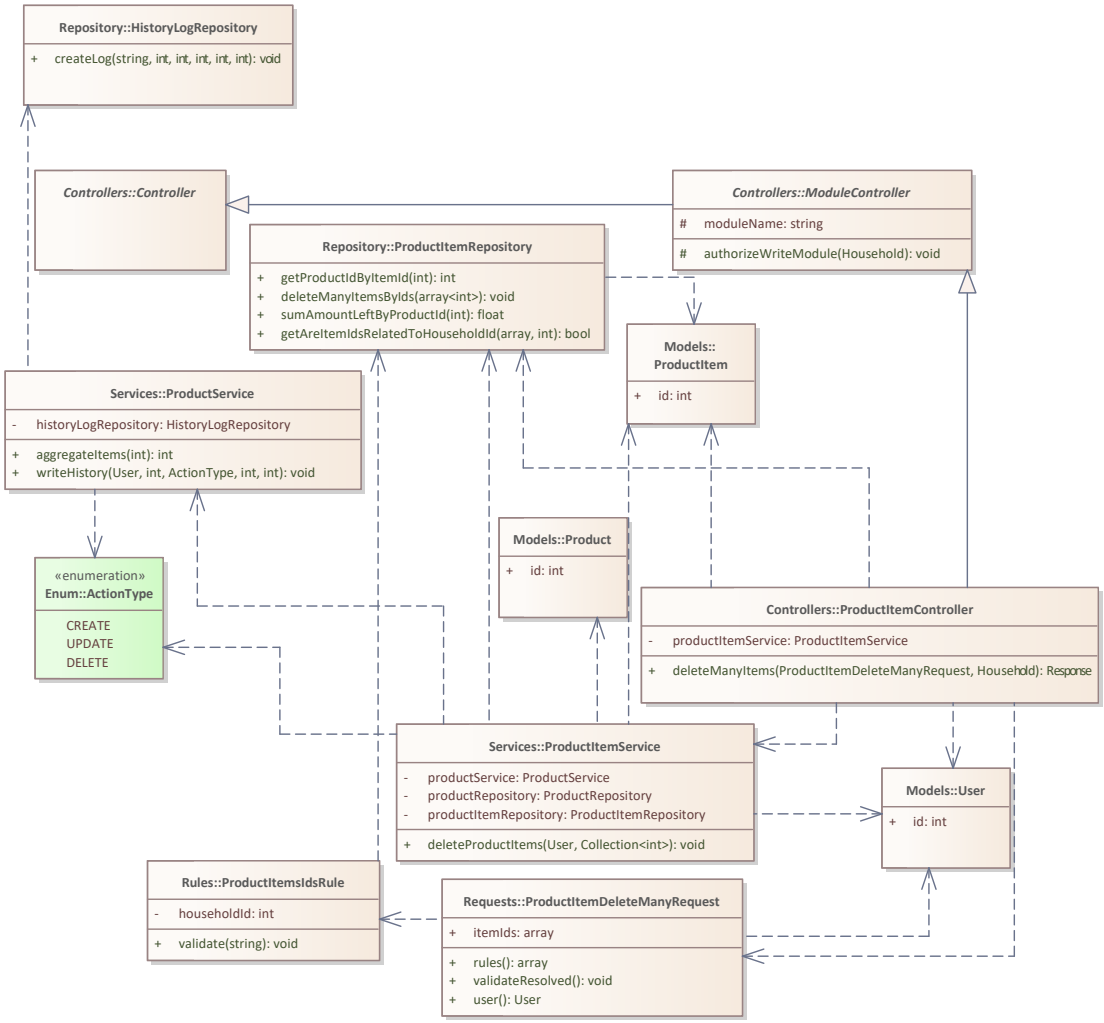
Triedy v balíčku *Request* sú súčasťou prezentačnej vrstvy. Tieto triedy majú na starosti jednoduchšiu validáciu typov alebo krajných hodnôt dát. V prípade nutnosti komplexnejšej validácie sa používajú triedy z balíčku *Rules*, ktoré sú už súčasťou biznis vrstvy. Po validácii sa predáva vstup do jednej z tried v balíčku *Controllers*, ktoré sú súčasťou prezentačnej vrstvy. Tieto triedy rozhodujú, akú doménovú či dátovú logiku využijú k splneniu potrebnej funkcionality. Za vykonanie doménovej logiky sú zodpovedné triedy v balíčku *Services*, ktoré ale na komunikáciu s databázou musia použiť triedy z balíčka *Repository*.

Balíček *Repository* obsahuje triedy postavené na návrhovom vzore s rovnakým názvom. Tieto triedy sú chápané ako súčasť dátovej vrstvy. Použitím tohto návrhového vzoru sa abstrahuje logika prístupu k dátam a manipulácie s nimi. Týmto sa zaistí oddelenie zodpovedností. Tento prístup neskôr zjednoduší proces jednotkového testovania, keďže doménovú logiku bude možné testovať zvlášť. Ďalším balíčkom dátovej vrstvy je *Models*. Spomínaný balíček obsahuje triedy reprezentujúce jednotlivé tabuľky v databáze. Tieto triedy obsahujú atribúty podľa toho, aké stĺpce má daná tabuľka. O prevod riadkov tabuliek databázy do objektov týchto tried sa stará už spomínaný ORM (podkapitola 3.3.2).

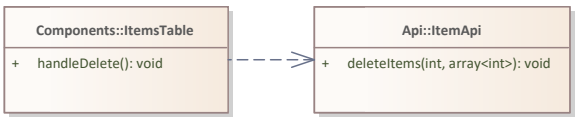
Posledným balíčkom je *Resources*. Balíček obsahuje triedy, ktoré sú zodpovedné za prevod objektov do formátu JSON. V diagrame 3.6 sa ale nenachádza žiadna trieda z tohto balíčka. Je to z dôvodu, že serverová časť pri výmaze kusov neposiela žiadne dáta klientskej časti, a preto nie je daný prevod nutný.

### 3.5.2 Návrh komunikácie tried

K zobrazeniu komunikácie tried z návrhu na výmaz kusov boli použité sekvenčné diagramy 3.8 a 3.9. Celý proces začína stlačením tlačítka pre výmaz kusov užívateľom po tom, čo v tabuľke označil kusy, ktoré chce zmazať. Framework klientskej časti React týmto spustí funkciu *handleDelete* v komponente *ItemsTable*. Tento komponent je zodpovedný za zobrazenie tabuľky, a z pohľadu architektúry klientskej časti sa pokladá za view. V tejto funkcii sa zavolá funkcia *deleteItems* objektu *itemApi* s identifikátormi kusov, ktoré majú byť zmazané. V kontexte archi-




**Obr. 3.6** Diagram tried serverovej časti nutných k realizácii prípadu použitia výmazu kusov



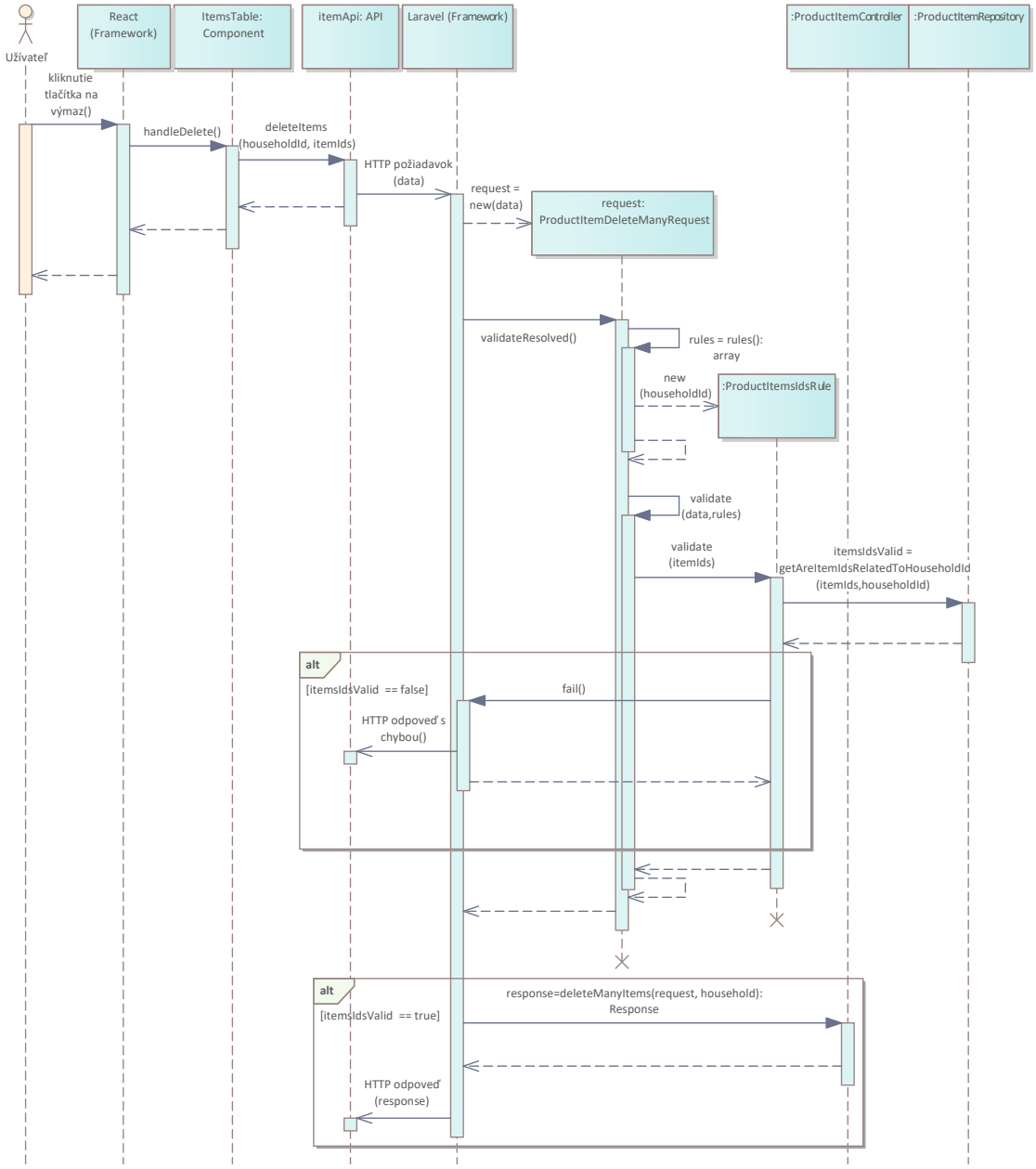

**Obr. 3.7** Diagram komponentov a typov objektov klientskej časti nutných k realizácii prípadu použitia výmazu kusov (pre jednoduchšie zakreslenie reprezentované ako triedy)

tektúry klientskej časti je táto funkcia akcia, a objekt *itemApi* je reducer. Funkcia odošle HTTP požiadavku na rozhranie REST serverovej časti, obsahujúcu identifikátory kusov a domácnosti. Túto požiadavku prevezme framework serverovej časti Laravel, a vytvorí inštanciu vhodnej Request triedy, ktorá je v tomto prípade *ProductItemDeleteManyRequest*. Pri vytvorení jej predá spomínané identifikátory a zavolá jej metódu *validateResolved*.

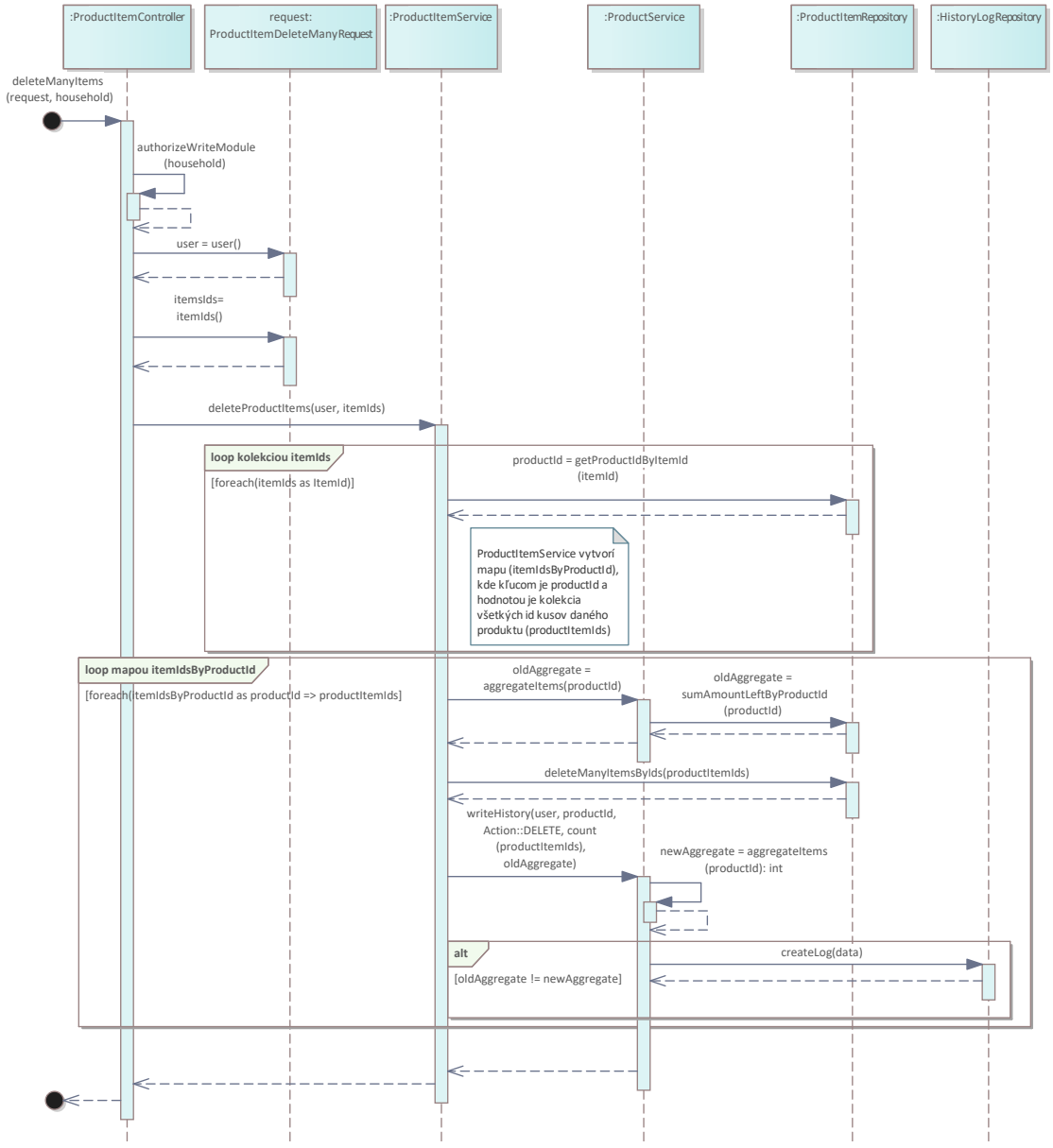
Zavolaním metódy *validateResolved* sa spustí validácia dát, ktoré prišli na server. Pre vyzdvihnutie všetkých pravidiel, ktoré sú relevantné, si zavolá trieda *ProductItemDeleteManyRequest* svoju ďalšiu metódu *rules*, v ktorej sa vytvorí inštancia triedy *ProductItemsIdsRule*. Táto trieda

je zodpovedná za overenie, že identifikátory kusov patria do danej domácnosti. Po dokončení metódy *rules* zavolá metóda *validateResolved* metódu *validate* na všetky vrátené pravidlá. Tým sa spustí validácia v triede *ProductItemsIdsRule*, v ktorej sa zavolá metóda *getAreItemIdsRelatedToHouseholdId* triedy *ProductItemRepository*, pomocou ktorej zistí, či patria dané identifikátory kusov do domácnosti. Ak nie, tak zavolá trieda *ProductItemsIdsRule* funkciu *fail*, čím sa vyhodí výnimka. Laravel túto výnimku spracuje, odošle HTTP odpoveď klientskej časti o zlyhaní validácie a celý proces týmto končí. V opačnom prípade framework zavolá metódu *deleteManyItems* triedy *ProductItemController*, čím končí sekvenčný diagram 3.8.

Sekvenčný diagram 3.9 začína zavolaním už spomínanej metódy *deleteManyItems*. *ProductItemController* si pomocou svojej metódy *authorizeWriteModule* overí, že daný užívateľ má práva upravovať inventár domácnosti. Následne si z predaného *Request* objektu vyzdvihne užívateľa a identifikátory kusov, a predá ich do metódy *deleteProductItems* triedy *ProductItemService*. Daná metóda potom začne cyklicky prechádzať všetky identifikátory kusov za účelom vytvorenia mapy, kde kľúčom je identifikátor produktu a hodnotou je kolekcia všetkých identifikátorov kusov daného produktu. K zisku identifikátoru produktu je využívaná metóda *getProductIdByItemId* triedy *ProductItemRepository*. Po dokončení cyklu začne v metóde *deleteManyItems* ďalší cyklus cez mapu, ktorá bola vytvorená v predošlom cykle. Cieľom každej iterácie v tomto cykle je výmaz kusov produktov a v prípade, že sa zmení agregácia množstva produktu, i vytvorenie záznamu o zmene množstva produktu. K agregácii a zápisu histórie sú využité metódy *aggregateItems* a *writeHistory* triedy *ProductService*. Na samotný výmaz kusov je zase použitá metóda *deleteManyItemsByIds* triedy *ProductItemRepository*. Po skončení cyklu daná metóda končí, a serverová časť pošle HTTP odpoveď klientskej časti o úspešnom výmaze.




**Obr. 3.8** Sekvenčný diagram začínajúci stlačením tlačítka na výmaz kusov užívateľom, končiaci zavolaním metódy kontroléra




**Obr. 3.9** Sekvenčný diagram zavolania metódy *deleteManyItems* triedy *ProductItemController*

# Kapitola 4
# Implementácia

Implementácia je akt, pri ktorom sa premieňa návrh aplikácie na funkčnú aplikáciu. Zahŕňa písanie kódu, integráciu rôznych systémov a zabezpečenie funkčnosti softvéru. V tejto kapitole je opísaná implementácia jadra a modulu inventára domácnosti aplikácie HouseKeeper. Najprv sú popísané techniky riadenia softvérového projektu, potom použité nástroje a štruktúra súborov. Ďalej je opísaná implementácia autentizácie, modularizácie projektu a tvorba REST API. Kapitola končí opisom implementácie klientskej časti.

## 4.1 Riadenie softvérového projektu

V dnešnej dobe je vývoj softvéru nie len o naprogramovaní riešenia, ale i o celkovom riadení softvérového projektu. Správne riadenie projektu zaisti omnoho vyššiu kvalitu vyvinutého softvéru, jasnejší prehľad počas jeho realizácie a zvýšenie efektivity počas implementácie. Tým, že vývoj jadra aplikácie bola tímová práca a následný vývoj modulov aplikácie prebiehal paralelne, je dôležitosť správneho riadenia projektu ešte väčšia. Všetky spomínané procesy v tejto podkapitole boli zavedené spolu s kolegom Dávidom Jenčom a vedúcim práce.

### 4.1.1 Verzovanie kódu

Verzovanie kódu je jeden z najvýznamnejších aspektov pre správne riadenie projektu. Verzovanie umožní tímom sledovať a spravovať zmeny. V prípade nutnosti dáva taktiež možnosť vrátiť sa naspäť v histórii za účelom odvrátenia nechcených zmien.

Pre verzovanie kódu bol používaný systém *Git*. Git [25] je distribuovaný nástroj pre verzovanie kódu, ktorý umožňuje viacerým vývojárom pracovať na jednom projekte súčasne. To, že je distribuovaný, znamená, že každý člen tímu má svoju lokálnu kópiu celého projektu i históriu jeho zmien. To umožní vývojárom nezávislo na sebe uskutočňovať zmeny bez nutnosti spoliehať sa na centrálny server, ako to je v prípade centralizovaných verzovacích systémov.

Medzi kľúčové pojmy systému Git patrí *commit* (záznam o zmene) a *vetva* (línia zmien). Väčšinou predstavuje jedna vetva nejakú vyvíjanú funkcionalitu aplikácie, a jednotlivé commity predstavujú úpravy, ktorými sa daná funkcionalita implementuje. Po dokončení funkcionality sa môže previesť *merge* (zlúčenie) vetvy s vyvíjanou funkcionalitou do hlavnej vetvy. Všetky zmeny by sa potom mali posielať na spoločný repozitár, aby si ich mohli stiahnuť i ďalší členovia tímu.

Pri vývoji aplikácie HouseKeeper bol ako spoločný repozitár použitý fakultný GitLab repozitár pod správou vedúceho práce. GitLab [26] ponúka množstvo nástrojov na riadenie projektu. Medzi najväčšie výhody týchto nástrojov patrí bohatá ponuka funkcionalít, priama integrácia s repozitárom a jednotné webové rozhranie pre všetky nástroje.

### 4.1.2 Sledovanie práce

Kľúčovú rolu v oblasti riadenia softvérových projektoch bezpochyby hrá správa a sledovanie práce a úloh vývojárov. Pre sledovanie práce a úloh v aplikácii HouseKeeper bol využívaný nástroj *GitLab issues*. Tento nástroj poskytuje komplexný prehľad o prebiehajúcich *issues* (úlohách) a umožňuje tak tímom efektívne organizovať ich prácu. Každú úlohu je možné priradiť konkrétnemu vývojárovi, ktorý potom bude zodpovedný za jej vykonanie.

Dôležitou funkcionalitou tohto nástroja je možnosť pridávať *labels* (štítky) ku každej úlohe. Každá úloha môže mať týchto štítkov aj niekoľko zároveň. Tieto štítky slúžia na kategorizáciu úloh, ktorá neskôr umožní ich filtrovanie. V projekte aplikácie HouseKeeper boli štítky používané na rozlíšenie toho, či sa úlohy týkajú klientskej alebo serverovej časti jadra, alebo konkrétneho modulu. Taktiež boli použité štítky na určenie priority úlohy, a na to, či sa daná úloha týka opravy existujúcej funkcionality, alebo pridania novej. Za zmienku stoja určite aj štítky na určenie aktuálneho stavu danej úlohy. Pomocou týchto štítkov je možné určiť, či sa na úlohe pracuje, alebo je už úloha hotová a čaká na záverečnú kontrolu od vedúceho práce.

### 4.1.3 Vývojový proces

Súčasťou riadenia softvérového projektu je aj určenie a dodržanie vývojového procesu, ktorý zaručí vysokú kvalitu výsledného softvéru. V projekte aplikácie HouseKeeper bol zavedený vývojový proces, ktorý začína tvorbou *issue* (úlohy). Táto úloha spravidla obsahuje opis požadovanej práce, relevantné štítky a priradeného vývojára, ktorý je zodpovedný za vykonanie úlohy.

Keď na úlohe začne vývojár pracovať, zmení štítky o stave úlohy. Týmto bude v prehľade úloh jasné, že sa na úlohe pracuje. Potom vytvorí novú vetvu z hlavnej vetvy projektu, v ktorej implementuje potrebné zmeny na svojom lokálnom zariadení. Počas implementácie pravidelne nahráva danú vetvu do spoločného repozitára, aby v prípade poškodenia lokálneho zariadenia neprišiel o svoje zmeny.

Po dokončení implementácie je ďalším krokom vytvorenie *Merge Requestu* (žiadosti o zlúčenie zmien). V danej žiadosti označí riešenú úlohu, čím sa v systéme GitLab prepojí daná žiadosť s úlohou. Systém k žiadosti zobrazí prehľadné webové rozhranie, v ktorom sú viditeľné vykonané zmeny v kóde. Táto žiadosť o zlúčenie je potom podrobená *code review* (posúdeniu kódu) vedúcim práce, čím sa zabezpečí kvalita a správnosť kódu. Po schválení sa zmeny zlúčia do hlavnej vetvy, a úloha sa v systéme GitLab označí ako vyriešená. Tento proces zabezpečuje štruktúrovaný a efektívny prístup k správe a integrácii zmien v projekte.

### 4.1.4 Kvalita kódu

Veľmi dôležitou zložkou vývoja softvéru je aj zaistenie vysokej kvality kódu, pretože kvalita kódu priamo ovplyvňuje udržiavateľnosť, rozšíriteľnosť a spoľahlivosť aplikácie.

V projekte aplikácie HouseKeeper bol pre zaistenie kvality kódu použitý SonarQube. SonarQube [27] je nástroj na statickú analýzu, ktorý hodnotí kvalitu kódu s cieľom odhaliť chyby, zraniteľnosti a nečitateľný kód. Vo svojom webovom rozhraní poskytuje podrobný prehľad o stave kódu, v ktorom taktiež ponúka návrhy na zlepšenie. V tomto prehľade sú zobrazené informácie, ako počet chýb, zraniteľností a *code smells* (nečitateľný kód), s odhadom pracnosti ich opravy. Projekt je za každú túto kategóriu nedostatkov ohodnotený známkou. Pomocou týchto známok je možné vidieť, či je vyvíjaný kód v projekte dostatočne kvalitný. Po vyvinutí aplikácie HouseKeeper bol projekt ohodnotený vo všetkých kategóriách najlepšou možnou známkou, „A“.

Ďalším aspektom zaistenia kvality kódu je použitie automatických formátovačov kódu. Tieto nástroje sú schopné automaticky naformátovať kód tak, aby dodržiaval konkrétny štýl kódu. Tým sa zabezpečí jeho konzistentnosť v celom projekte. Dodržaním konzistentnosti sa mno-

honásobne zvyšuje čitateľnosť a udržiavateľnosť kódu. Spomínané nástroje taktiež zvyšujú efektivitu vývojárov, ktorí nemusia strácať čas ručným formátovaním kódu, a môžu sa tak sústrediť na samotnú logiku a funkčnosť. V projekte aplikácie HouseKeeper boli pre tento účel použité nástroje Prettier[1] a Pint[2].

### 4.1.5 Dokumentácia

Pre vytvorenie dokumentácie projektu bol použitý nástroj *Enterprise Architect*. Enterprise Architect [28] je nástroj, ktorý sa špecializuje na podporu softvérových činností, ako je analýza a návrh. Pre účely aplikácie HouseKeeper bol vedúcim práce založený nový projekt, do ktorého boli nahrané vytvorené diagramy a dokumentácia. Takto môžu ďalší vývojári jednoducho vychádzať z už vytvorených diagramov, bez nutnosti ich prekreslenia.

Na vytvorenie dokumentácie kódu sa využíval nástroj phpDocumentor[3]. Pomocou tohto nástroja je možné vytvoriť dokumentáciu kódu jednoduchým spustením príkazu. Vďaka tejto vlastnosti našiel využitie v procese kontinuálnej integrácie.

### 4.1.6 Kontinuálna integrácia

Kontinuálna integrácia je jedným z ďalších kľúčových procesov pri vývoji softvéru. Daný proces sa spúšťa automaticky pri nahraní zmien kódu vývojárom do spoločného repozitára. Týmto podporuje pravidelné nahrávanie zmien. V prípade aplikácie HouseKeeper sa tento proces odohráva v systéme GitLab, ktorý obsahuje nástroje na jeho podporu.

Pri nahraní zmien sa proces kontinuálnej integrácie začína zostavením aplikácie, čím sa môžu zachytiť základné chyby v kóde. Táto zostavená aplikácia neskôr slúži ako východisko pre ďalšie kroky tohto procesu. Po zostavení sa spustia automatizované jednotkové testy, ktoré sú podrobnejšie opísané v podkapitole 5.1, čím sa zachytia chyby na logickej úrovni kódu. Kód sa potom odošle na už spomínaný SonarQube za účelom statickej analýzy kódu. Ďalším krokom je automatizovaná tvorba dokumentácie kódu pomocou nástroju phpDocumentor. Posledným krokom je potom vytvorenie obrazu Docker, čím je aplikácia pripravená na nasadenie (podkapitola 5.3).

Kontinuálna integrácia takto zefektívňuje celý proces implementácie. Je to preto, že znižuje počet chýb a zvyšuje efektivitu automatizovaním krokov, ktoré by bolo inak nutné vykonať ručne.

## 4.2 Použité nástroje

Vytváranie webovej aplikácie je komplexný proces, ktorý zahŕňa kombináciu technických znalostí a efektívneho využívania nástrojov. V oblasti vývoja webových aplikácií zohrávajú nástroje kľúčovú úlohu pri zefektívňovaní procesu vývoja.

Jeden z najdôležitejších nástrojov pri programovaní je určite editor, alebo IDE. Dobrý editor umožní efektívne realizovať zmeny a meniť kód na niekoľkých miestach zároveň, čo má neskôr pozitívny dopad na celý proces vývoja. Pri vývoji serverovej časti, napísanej v jazyku PHP, bolo použité IDE PHPStorm[4] a nástroj na správu závislostí Composer. Pri vývoji klientskej časti, napísanej v jazyku Typescript, bolo na druhú stranu použité IDE WebStorm[5] a nástroj na správu závislostí npm. Všetky spomínané nástroje sú štandardnou voľbou pre dané programovacie jazyky.

[1] https://prettier.io/
[2] https://laravel.com/docs/10.x/pint
[3] https://www.phpdoc.org/
[4] www.jetbrains.com/phpstorm
[5] www.jetbrains.com/webstorm

## 4.3 Štruktúra súborov serverovej časti

Pri vývoji webovej aplikácie je výber vhodnej štruktúry súborov veľmi dôležitý. Laravel poskytuje štandardnú štruktúru súborov, ktorá je intuitívna a efektívna, a zároveň jej dodržanie umožní lepšiu navigáciu v súboroch. Štandardná štruktúra uľahčí vývojárom, ktorí už majú skúsenosť s Laravelom, rýchlejšie sa adaptovať na nový projekt. Podporuje taktiež vstavané funkcionality Laravelu.

Štandardnú štruktúru Laravelu bolo nutné obohatiť o niekoľko adresárov, ako *Modules* či *Repository*. Adresár *Modules* obsahuje moduly vytvorené pomocou knižnice na modularizáciu (podkapitola 4.6). Adresár *Repository* obsahuje triedy na komunikáciu s databázou, spomínané v návrhu tried v sekcii 3.5.1. Ďalšie adresáre nebolo nutné vytvárať, keďže Laravel už vytvoril adresáre pre všetky ďalšie balíčky spomenuté v návrhu tried.

- `app` — zdrojové kódy jadra
  - `Console` — vlastné konzolové príkazy
  - `Data` — dátové triedy na presun dát medzi vrstvami
  - `Http` — časť, ktorá má na starosti komunikáciu cez protokol HTTP
    - `Controllers` — triedy na spracovanie HTTP požiadaviek a vrátenie odpovedí
    - `Middleware` — triedy na filtrovanie HTTP požiadaviek
    - `Requests` — validačná logika vstupných dát z REST API
    - `Resources` — triedy na formátovanie dát v odpovedi serveru
  - `Models` — modely aplikácie
  - `Repository` — triedy abstrahujúce logiku prístupu k dátam
  - `Rules` — komplexnejšia validácia vstupu
  - `Services` — doménová logika aplikácie
- `config` — konfiguračné súbory aplikácie
- `database` — migračné súbory aplikácie
- `lang` — súbory obsahujúce preklady reťazcov
- `Modules` — moduly aplikácie
  - `Inventory` — modul s rovnakou štruktúrou súborov, ako zložka *app*
- `public` — verejná zložka aplikácie, po zostavení aplikácie obsahuje klientskú časť
- `composer.json` — zoznam závislostí PHP knižníc

**Obr. 4.1** Štruktúra súborov serverovej časti

## 4.4 Autentizácia

Autentizácia je základným bezpečnostným prvkom webových aplikácií, ktorý slúži na overenie identity užívateľov. Zabezpečuje, aby k citlivým údajom mali prístup len oprávnené osoby. Z tohto dôvodu tvorí autentizácia základnú časť jadra aplikácie HouseKeeper.

### 4.4.1 Využitie Breeze

Ako bolo spomínané v sekcii 3.2.2, Laravel ponúka možnosť využiť predpripravené balíčky funkcionalít pre zrýchlenie vývoja. Po spustení príkazu na inicializáciu základného Laravel projektu bol využitý práve balíček *Breeze*. Súčasťou tohto balíčku je implementácia prihlásenia, registrácie, emailovej verifikácie a obnovy hesla. Použitie balíčku značne urýchlilo vývoj aplikácie, avšak stále boli potrebné isté úpravy. Breeze vygenerovalo všetku logiku spojenú s týmito akciami do kontrolérov, ktoré sú súčasťou prezentačnej vrstvy. Pre oddelenie zodpovedností vrstiev bolo nutné danú logiku upraviť a premiestniť do tried biznis vrstvy.

### 4.4.2 Cookies

Pre podrobnejší opis funkčnosti autentizácie jadra je nutné vysvetliť, čo je to *cookie*. Cookie je malé množstvo dát, ktoré užívateľovi posiela webový server za účelom ich uchovania v jeho prehliadači. Neskôr, pri posielaní HTTP požiadavku na server, ich môže prehliadač klienta priložiť k tomuto požiadavku. Pomocou týchto dát potom dokáže server autentizovať užívateľa. [29]

*HTTP-only cookies*, odporučené uznávanou organizáciou OWASP [30], zvyšujú úroveň zabezpečenia. Nastavením príznaku *HttpOnly* v cookie sa zabráni skriptom na strane klienta v prístupe k údajom súboru cookie. Väčšina útokov XSS (cross-site scripting) je zameraná práve na krádež cookies. Ak prehliadač podporuje funkciu *HttpOnly* a zistí súbor cookie s týmto príznakom, zabráni skriptom na strane klienta v ich čítaní. Rozhodlo sa preto, že pre ich vysokú úroveň bezpečnosti bude jadro aplikácie používať tento typ cookies.

Alternatívou použitia cookies je použitie autentizačných tokenov a ich uloženie do *browser local storage* (lokálneho úložiska prehliadača). V prípade úspešného prihlásenia, pošle serverová časť klientskej časti token v podobe reťazca. Tento token si potom uloží klientská časť do lokálneho úložiska prehliadača. Organizácia OWASP ale neodporúča použitie tohto lokálneho úložiska [31]. Je to z dôvodu, že dané úložisko je náchylné na útoky XSS, pretože prehliadač nezabraňuje prístupu k tomuto úložisku, ako v podobe *HTTP-only* cookies.

## 4.5 REST API

REST je štandard používaný pri vytváraní webových aplikácií. Pomocou tohto štandardu komunikuje klientská a serverová časť aplikácie HouseKeeper. REST umožňuje systémom komunikovať cez internet jednoduchým spôsobom pomocou požiadaviek HTTP za účelom prístupu k dátam. Pri komunikácii sa používajú metódy HTTP (GET, POST, PUT, DELETE). Dáta sa k jednotlivým požiadavkám prikladajú vo formáte JSON. [32]

Laravel pri vývoji rozhrania REST potvrdil svoju silnú stránku v tejto oblasti, a značne tak urýchlil implementáciu. Pri vývoji bol pravidelne využívaný efektívny spôsob tvorby REST kontrolérov pomocou jedného príkazu. Spomínaný príkaz vytvorí šablónu triedy REST kontroléru. Táto trieda má predpripravené metódy na jednotlivé akcie REST. Ukážka tejto vygenerovanej triedy sa nachádza vo výpise kódu 4.1.

Ďalšou zo silných stránok Laravelu pri tvorbe rozhrania REST je aj kompaktný zápis deklarácie ciest na jednotlivé akcie REST kontroléru. Vďaka dodržaniu pomenovania metód vo vygenerovanej triede, stačí na prepojenie ciest k akciám REST jeden riadok kódu. To eliminuje potrebu deklarovať každú cestu a akciu REST s HTTP metódou zvlášť. Týmto sa značne znižuje chybovosť, zvyšuje čitateľnosť kódu a rýchlosť vývoja. Ukážka tejto deklarácie ciest k metódam REST kontroléru sa nachádza vo výpise 4.2.

## 4.6 Modularizácia

Aplikácia HouseKeeper má okrem jadra niekoľko modulov, preto je potrebné štruktúru projektu podľa nich rozdeliť. Výhodou tohto prístupu je to, že je jasne rozpoznateľné, ktoré súbory sú súčasťou ktorého modulu, čo výrazne zlepšuje orientáciu v kóde.

Pri implementácii bola použitá knižnica *Laravel Modules*, ktorá ponúka i množstvo príkazov na prácu s jednotlivými modulmi. Napríklad, príkaz na spustenie databázových migrácií konkrétneho modulu, či príkaz pre generovanie tried pre konkrétny modul. Knižnica umožňuje aj tvorbu nového modulu, pričom štruktúra súborov vo vytvorenom module bude veľmi podobná štruktúre súborov jadra (Obrázok 4.1). Kolega Dávid Jenčo zvolil pri implementácii klientskej časti podobné riešenie, tým pádom je aplikácia ako celok konzistentná. Týmto prístupom je zaistená jednoduchá rozšíriteľnosť aplikácie o prípadné ďalšie moduly.

```php
class HouseholdController
{
    public function index()
    {
        // HTTP Metóda: GET, Url: "/households"
    }

    public function store(Request $request)
    {
        // HTTP Metóda: POST, Url: "/households"
    }

    public function show(Household $household)
    {
        // HTTP Metóda: GET, Url: "/households/{id}"
    }

    public function update(Request $request, Household $household)
    {
        // HTTP Metóda: POST, Url: "/households/{id}"
    }


    public function destroy(Household $household)
    {
        // HTTP Metóda: DELETE, Url: "/households/{id}"
    }
}
```

**Výpis kódu 4.1** Ukážka vygenerovaného REST kontroléru s pridanými komentármi pre lepšie pochopenie

```php
Route::apiResource('households', HouseholdController::class);
```

**Výpis kódu 4.2** Ukážka kompaktnej deklarácie ciest k jednotlivým akciám REST kontroléru

```
$category->ancestors // Kolekcia podkategórií danej kategórie
$category->children  // Kolekcia iba priamych podkategórií danej kategórie
$category->children  // Kolekcia nadkategórií danej kategórie
```

**Výpis kódu 4.3** Ukážka použitia knižnice Laravel Adjacency List

## 4.7 Rekurzívne vzťahy

Pri implementácii modulu inventára domácnosti boli jednou z výziev rekurzívne vzťahy medzi dátami. Tento typ vzťahu sa v module vyskytuje niekoľkokrát. Konkrétne sa jedná o vzťah medzi kategóriou a nadkategóriou, lokáciou a nadlokáciou. Možností, ako sa vysporiadať s týmto problémom, bolo niekoľko. Buď všetku logiku, ako získavanie podkategórií a nadkategórií danej kategórie implementovať manuálne, alebo k tomu využiť už existujúce riešenie. Medzi výhody použitia existujúceho riešenia patrí nižšia chybovosť, rýchlejší a bezpečnejší kód, a efektívnejší vývoj.

Na daný problém sa preto zvolila knižnica *Laravel Adjacency List*[6], vytvorená jedným z vývojárov Laravelu. Táto knižnica sa špecializuje na riešenie rekurzívnych vzťahov, a je pravidelne aktualizovaná. Ukážku použitia tejto knižnice je možne vidieť vo výpise kódu 4.3. Ako je vidieť v ukážke, použitie je jednoduché a priamočiare. Zvolením tejto knižnice sa ušetrilo nemalé množstvo času pri implementácii.

## 4.8 Posielanie emailov

Ďalšou z prekážok pri vývoji bolo vyriešenie problematiky posielania emailov. Serverová časť aplikácie musí posielať emaily pri registrácii za účelom overenia emailovej adresy užívateľa. Taktiež posiela emaily pri resetovaní zabudnutého hesla, alebo na upozornenie užívateľa o nízkom množstve produktu či expirujúcom kuse produktu.

Laravel našťastie ponúka rozhranie na komunikáciu s emailovými službami. Na nastavenie stačí upraviť relevantné riadky v konfiguračnom súbore Laravelu, podobne ako pri nastavovaní databázy. Tento prístup umožňuje neskôr vymeniť emailovú službu za inú, bez nutnosti zasahovať do kódu. Ako emailová služba, cez ktorú sa posielajú všetky emaily, bola zvolená služba Gmail[7] od spoločnosti Google. Pre jej využitie stačí vytvorenie obyčajného Google účtu, a vygenerovanie API kľúču, ktorý slúži na autentizáciu aplikácie voči službe. Tento kľúč sa potom spolu s identifikačnými údajmi emailovej služby zadal do konfiguračného súboru Laravelu.

Využitie našli aj predpripravené vzhľady emailov s obsahom, ktoré boli súčasťou predpripraveného balíčku funkcionalít Breeze, a vyžadovali iba drobné úpravy. Jediné emaily, ktorých obsah bolo nutné napísať ručne, boli emaily týkajúce sa upozornení. Vďaka využitiu abstrakcie Laravelu na tvorbu obsahu emailu, bola ale ich implementácia jednoduchá a priamočiara.

Posledným krokom bolo nastavenie automatického posielania upozornení o polnoci ohľadom nízkeho zostávajúceho množstva produktu a expirujúcich kusov. I k tomu ponúka Laravel vhodnú abstrakciu, kde v triede *Kernel*, nachádzajúcej sa v súbore *Kernel.php* v zložke *app/Console*, stačilo konkretizovať metódu, a ako často sa má daná metóda volať. Použitie je možné vidieť vo výpise kódu 4.4. Výsledkom je to, že sa každý deň o polnoci spustí metóda *checkNotifications* triedy *NotificationService*, ktorá má na starosti kontrolu nastavených notifikácií a odosielanie emailov.

[6] https://github.com/staudenmeir/laravel-adjacency-list
[7] https://www.google.com/intl/sk/gmail/about/

```
 class Kernel extends ConsoleKernel
{
    protected function schedule(Schedule $schedule): void
    {
        $schedule->call(function (NotificationService $notificationService) {
            $notificationService->checkNotifications();
        })->daily();
    }
}
```

**Výpis kódu 4.4** Ukážka nastavenia automatického každodenného spúšťania metódy

## 4.9 Klientská časť aplikácie

V súvislosti s vývojom modulu inventára bolo samozrejme potrebné vyvinúť aj klientskú časť. Architektúru a použité technológie tejto časti navrhol kolega Dávid vo svojej bakalárskej práci. Implementácia klientskej časti teda vychádza z daného návrhu, a preto je potrebné spomenúť použité technológie.

### 4.9.1 TypeScript

TypeScript [33] je open source programovací jazyk vyvinutý spoločnosťou Microsoft. Je určený na vývoj rozsiahlych aplikácií a prekladá sa do jazyka JavaScript. TypeScript je staticky typovaný jazyk, vďaka čomu umožňuje včasné zachytávanie chýb počas procesu vývoja. Jeho kompatibilita s existujúcimi knižnicami jazyka JavaScript a schopnosť spustiť ho v ľubovoľnom prehliadači z neho robí univerzálnu voľbu pre vývoj.

### 4.9.2 React

React [34] je populárny open source framework, ktorý vyvinula spoločnosť Facebook. React sa používa na vytváranie užívateľských rozhraní, najmä jednostránkových aplikácií. Architektúra Reactu je založená na znovu použiteľných komponentoch, ktoré podporujú znovuvyužitie kódu.

Veľkou výhodou Reactu je možná asynchrónna komunikácia so serverom, bez nutnosti obnovovania stránky. Tým je stránka omnoho viac užívateľsky prívetivá, oproti klasickým multistránkovým aplikáciám, pri ktorých sa obnovuje celá stránka pri zmene dát. Špecifickou vlastnosťou Reactu je aj virtuálny DOM[8]. Virtuálny DOM sa mení pri každej zmene vnútorného stavu aplikácie. React pravidelne porovnáva obsah virtuálneho a reálneho DOM, a v prípade, že sa líšia, nahradí obsah reálneho DOM tým virtuálnym. Tento proces značne optimalizuje výkon, pretože úpravy reálneho DOM sú spravidla príliš časovo náročné.

### 4.9.3 Vzhľad stránky

Atraktívny vzhľad stránky bol dosiahnutý použitím knižnice MUI a CSS[9] frameworku Tailwind. MUI [35] je knižnica, ktorá obsahuje bohatú ponuku komponentov používateľského rozhrania, ako sú tlačidlá, tabuľky, dialógové okná, ikonky, či textové polia. Tieto komponenty sa riadia

[8] DOM – Document Object Model
[9] CSS – Cascading styling sheets

princípmi Material Design[10] a umožňujú tak vývojárom vytvárať vizuálne atraktívne aplikácie. Použitím týchto komponentov sa zaručil konzistentný vzhľad aplikácie a ušetrený čas pri vývoji.

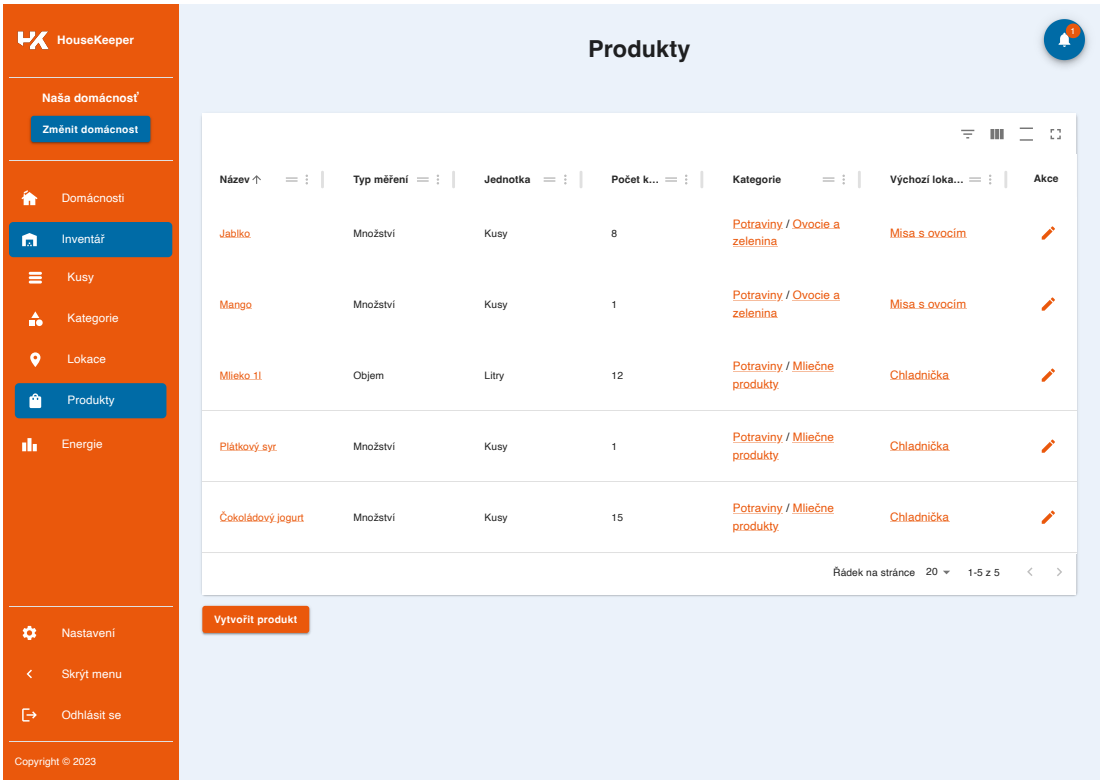
Pre samotné rozloženie stránky a umiestnenie jednotlivých komponentov bol použitý CSS framework Tailwind. Tailwind [36] obsahuje preddefinované CSS triedy, čím urýchľuje vývoj. Kladie dôraz na funkčnejší prístup k štýlovaniu, pri ktorom sa tieto CSS triedy aplikujú priamo do predpisu komponentov, bez nutnosti vytvárania ďalšieho CSS súboru.

### 4.9.4 Pokročilé vyhľadávanie dát

Jednou z funkčných požiadaviek na aplikáciu v sekcii 2.2.1 bolo pokročilé vyhľadávanie produktov a kusov. Táto požiadavka si vyžadovala implementáciu filtrovania a stránkovania dát na strane serveru. Taktiež bolo nutnosťou užívateľovi umožniť zobrazenie týchto dát v kompaktnej tabuľke.

Na implementáciu tabuľky nepostačovalo riešenie od knižnice MUI, ako v ostatných častiach aplikácie, pretože daná knižnica povoľuje filtrovanie iba podľa jedného stĺpca naraz. Existuje aj platená verzia tejto tabuľky, no pre účely aplikácie HouseKeeper by bolo toto riešenie nevhodné. Z tohto dôvodu bolo nutné nájsť alternatívu, a tou je knižnica *Material React Table*.

Material React Table[11] poskytuje mnoho funkcií. Umožňuje napríklad filtrovanie dát podľa ľubovoľného počtu stĺpcov naraz, či prehadzovanie a skrytie jednotlivých stĺpcov tabuľky. Ďalej obsahuje možnosť zobrazenia tabuľky na celú obrazovku, čo ocenia najmä užívatelia mobilných telefónov. Táto knižnica interne používa iné komponenty MUI, a tak prirodzene zapadá do vzhľadu aplikácie. Na obrázku 4.2 je možné vidieť vzhľad tabuľky vytvorenej pomocou tejto knižnice.




**Obr. 4.2** Vzhľad tabuľky produktov vytvorenej pomocou knižnice Material React Table

[10] Známy designový štandard vyvinutý spoločnosťou Google
[11] https://www.material-react-table.com/

# Kapitola 5
# Testovanie a nasadenie

Testovanie vyvinutého softvéru a kódu je základnou fázou vývoja softvéru, ktorá zabezpečuje spoľahlivosť, funkčnosť a rozšíriteľnosť aplikácie. Táto kapitola sa zaoberá testovaním a prípravou na nasadenie aplikácie HouseKeeper. Na začiatku kapitoly sa nachádza opis implementovaných jednotkových testov v kóde. Kapitola pokračuje výpisom užívateľských testov, ktoré pokrývajú základnú funkcionalitu. V závere je opísaný proces prípravy na nasadenie aplikácie.

## 5.1 Jednotkové testy

Jednotkové testy majú za úlohu testovať individuálne jednotky aplikácie, aby sa zabezpečilo správne fungovanie izolovaných funkcionalít. Najväčším prínosom týchto testov je ľahšie budúce rozširovanie aplikácie. Je to preto, že v prípade nesprávnych úprav existujúceho kódu dôjde k zlyhaniu jednotkových testov, čo pomôže zachytiť chyby.

Serverová časť aplikácie bola otestovaná pomocou frameworku Pest[1] a klientská časť pomocou frameworku Vitest[2]. Otestované boli všetky dôležité časti aplikácie obsahujúce zložitú doménovú logiku. V serverovej časti boli otestované všetky triedy typu *Service*, ktoré tvoria biznis vrstvu. Významnú rolu pri testovaní hralo to, že sa pri implementácii aplikácie použili na komunikáciu s databázou triedy typu *repository*. To umožnilo počas testovania využitie *mocks* (simulácií) týchto tried a iných závislostí, čo zaistilo izolovanie doménovej logiky od databázy. V klientskej časti bola zase otestovaná logika premeny jednotiek.

Jednotkové testy je samozrejme možné spúšťať počas vývoja manuálne. Spolu s kolegom Dávidom sa ale pridal ešte krok spustenia týchto testov ako súčasť kontinuálnej integrácie (sekcia 4.1.6). Týmto je zaistené, že kód, ktorý neprechádza napísanými jednotkovými testami, neprejde ani kontinuálnou integráciou. Vývojár takto zistí, že má v kóde ešte chyby, ktoré je nutné opraviť, predtým ako sa pokúsi o zlúčenie svojej práce do hlavnej vetvy projektu.

## 5.2 Užívateľské testy

Užívateľské testy slúžia na otestovanie aplikácie a overenie funkčnosti jej základných funkcionalít. Skladajú sa z užívateľských scenárov, pričom každý scenár pokrýva istú funkcionalitu aplikácie. Všetky scenáre, okrem prvých dvoch, vyžadujú prihlásenie. Užívateľské scenáre aplikácie sú opísané v nasledujúcich bodoch, pričom prvých 7 scenárov sa týka jadra, a ďalších 21

[1] https://pestphp.com
[2] https://vitest.dev

sa už týka modulu inventára domácnosti. Všetky scenáre modulu inventára domácnosti vyžadujú aspoň práva čítania na modul.

- **1. Registrácia nového užívateľa**

  Na úvodnej stránke pre registráciu zadať meno, emailovú adresu, heslo a potvrdenie hesla. Po zadaní hesla, pozostávajúceho iba z malých písmen, systém upozorní na to, že heslo musí obsahovať minimálne jedno veľké písmeno, jedno malé písmeno a jedno číslo. Systém taktiež upozorní na zadanie už zaregistrovanej emailovej adresy. Po úspešnej registrácii systém pošle na zadanú emailovú adresu email pre overenie emailovej adresy. Po overení nastane presmerovanie do aplikácie na stránku s oznámením, že overenie prebehlo úspešne. Týmto je užívateľ prihlásený do aplikácie.

- **2. Prihlásenie užívateľa**

  Ísť na prihlasovaciu stránku, zadať registrovanú emailovú adresu a heslo. Po zadaní nesprávneho hesla systém zobrazí chybovú hlášku. Po úspešnom prihlásení sa zobrazí úvodná stránka.

- **3. Obnova hesla**

  Na prihlasovacej stránke kliknúť na tlačítko *Zabudli ste heslo?*. Zadať emailovú adresu, ktorá bola registrovaná. Po zadaní nezaregistrovanej adresy sa zobrazí chybová hláška upozorňujúca na to, že sa nepodarilo nájsť užívateľa s touto emailovou adresou. Po zadaní registrovanej emailovej adresy a potvrdení formulára, systém odošle email na danú adresu. Zobraziť si daný email a kliknúť na tlačítko *Obnoviť heslo*. Vyplniť heslo, ktoré spĺňa požiadavky, potvrdiť zmenu kliknutím na tlačítko *Zmeniť heslo*. Následne vyskúšať prihlásenie s novým heslom a overiť nemožnosť prihlásenia sa so starým heslom.

- **4. Zmena hesla a mena**

  Kliknúť na *Nastavenie* v bočnom menu. Zmeniť heslo zadaním starého hesla, nového hesla a potvrdením nového hesla. Po zadaní nesprávneho starého hesla sa zobrazí chybová hláška. Zmeniť užívateľské meno zadaním nového mena.

- **5. Vytvorenie, editácia a zobrazenie domácnosti**

  Kliknúť na tlačítko *Domácnosti* v bočnom menu. Pred vytvorením prvej domácnosti systém upozorňuje v bočnom menu na fakt, že užívateľ nemá žiadnu aktívnu domácnosť. Novú domácnosť vytvoriť kliknutím na tlačítko *Vytvoriť novú domácnosť*. Zadať názov domácnosti, obec, kde sa domácnosť nachádza a podrobnejší popis. Upraviť vytvorenú domácnosť kliknutím na symbol ceruzky pri danej domácnosti v prehľade všetkých domácností, respektíve v detaile domácnosti po kliknutí na tlačítko *Upraviť domácnosť*. Overiť, že upravovať domácnosť môže iba majiteľ domácnosti a iní členovia nie. Systém v prehľade domácností zobrazuje všetky domácnosti, vlastné domácnosti a cudzie domácnosti.

- **6. Pridávanie nových členov do domácnosti**

  Pre vykonanie daného scenára, je nutné byť majiteľom danej domácnosti. Pridať nových členov do domácnosti kliknutím na symbol *Pridať člena* pri danej domácnosti v prehľade všetkých domácností, respektíve po kliknutí na detail domácnosti pomocou tlačítka *Pozvať nového člena*. Zadať emailovú adresu nového člena, vybrať mu oprávnenia na moduly. Je možné nevybrať žiadne oprávnenia, alebo určiť práva na čítanie alebo editáciu. Stlačiť tlačítko *Odoslať pozvánku*. V prípade zadania neregistrovanej emailovej adresy, systém upozorní na to, že sa nepodarilo nájsť užívateľa s danou emailovou adresou. V prípade, že emailová adresa je registrovaná, odošle sa pozvánka.

- **7. Pozvánka do domácnosti**

  V pravej hornej časti obrazovky sa zobrazí ikonka zvončeka. Rozkliknúť ikonku, po rozkliknutí sa zobrazí oznámenie o pozvaní do domácnosti. Oznámenie prijať alebo odmietnuť. V prípade prijatia pozvánky sa daná domácnosť pridá do sekcie cudzích domácností a užívateľ sa stane členom. V prípade odmietnutia pozvánky sa oznámenie vymaže.

- **8. Vytvorenie kategórie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Kliknúť na tlačítko *Inventár* v bočnom menu, potom na tlačítko *Kategórie*. Kliknúť na symbol + vedľa nadpisu *Kategórie*, respektíve na tlačítko *Vytvoriť kategóriu*. Zadať názov kategórie, popis, predvolený typ merania produktov, prípadne aj jednotku merania produktov. V prípade už nejakých existujúcich kategórií, možnosť zvoliť jednu z týchto kategórií ako nadradenú. Kliknúť na tlačítko *Vytvoriť kategóriu*.

- **9. Editácia kategórie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. V prehľade kategórií, po zvolení kategórie, kliknúť v detaile na tlačítko *Upraviť kategóriu*. Zmeniť názov, popis, nadradenú kategóriu, predvolený typ merania produktov, prípadne aj jednotku merania produktov. Pri zmene nadkategórií systém zobrazí dialógové okno, ktoré upozorňuje na to, že môže dôjsť k zmazaniu niektorých atribútov. Na záver kliknúť na tlačítko *Upraviť kategóriu*.

- **10. Výmaz kategórie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. V prehľade kategórií, po zvolení kategórie, kliknúť na tlačítko *Zmazať kategóriu*. Systém zobrazí upozornenie, že zmazaním kategórie budú zmazané aj všetky podkategórie a produkty tejto kategórie. Potvrdiť výmaz kliknutím na tlačítko *Zmazať*.

- **11. Detail kategórie**

  V bočnom menu, kliknutím na tlačítko *Kategórie*, si po zvolení kategórie zobraziť všeobecné informácie o kategórii, jej produktoch a atribútoch.

- **12. Vytvorenie lokácie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Kliknúť na tlačítko *Lokácie* v bočnom menu. Kliknúť na symbol + vedľa nadpisu *Lokácie*, respektíve na tlačítko *Vytvoriť lokáciu*. Zadať názov a popis. V prípade už nejakých existujúcich lokácií, možnosť zvoliť jednu z týchto lokácií ako nadradenú. Kliknúť na tlačítko *Vytvoriť lokáciu*.

- **13. Editácia lokácie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. V prehľade lokácií si zvoliť lokáciu a kliknúť na tlačítko *Upraviť lokáciu*. Zmeniť požadované políčka, napríklad názov, a potvrdiť kliknutím na tlačítko *Upraviť lokáciu*.

- **14. Výmaz lokácie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. V prehľade lokácií, po zvolení lokácie, kliknúť na tlačítko *Zmazať lokáciu*. Systém zobrazí upozornenie, že zmazaním lokácie budú zmazané aj všetky podlokácie tejto kategórie. Potvrdiť výmaz kliknutím na tlačítko *Zmazať*.

- **15. Detail lokácie**

  V sekcii *Lokácie* si zobraziť všeobecné informácie o lokácii a kusoch produktov, ktoré sa v tejto lokácii nachádzajú.

- **16. Vytvorenie produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Kliknúť na tlačítko *Produkty* v bočnom menu. Kliknúť na tlačítko *Vytvoriť produkt*. Zadať názov, popis, zvoliť kategóriu. Typ a jednotka merania sa automaticky predvyplnia, podľa predvoleného typu a jednotky merania zvolenej kategórie. Vyskúšať aj možnosť zmeniť typ a jednotku merania pre daný produkt. Ďalej zvoliť kapacitu produktu a predvolenú lokáciu kusov. Nastaviť si oznámenia na email zadaním spodnej hranice rezervy a spodnej hranice expirácie. Kliknúť na tlačítko *Vytvoriť produkt*.

- **17. Editácia produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Upraviť produkt kliknutím na symbol ceruzky v prehľade všetkých produktov, respektíve po kliknutí na názov produktu a tlačítko *Upraviť produkt*. Zmeniť požadované políčka, napríklad popis, a kliknúť na tlačítko *Upraviť produkt*. Overiť, že v prípade, že produkt už má nejaké kusy, nie je možnosť editovať typ merania. Ďalej vyskúšať zníženie kapacity produktu, ktorý už má existujúce kusy – zobrazí sa dialógové okno s upozornením, že môže dôjsť k zmene zostávajúceho množstva kusov.

- **18. Výmaz produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Vymazať produkt kliknutím na názov produktu a na tlačítko *Zmazať produkt*. Systém upozorní na to, že po zmazaní produktu budú zmazané aj všetky kusy tohto produktu. Potvrdiť výmaz kliknutím na tlačítko *Zmazať*.

- **19. Detail produktu**

  V sekcii *Produkty* si po kliknutí na daný produkt zobraziť všeobecné informácie o produkte, atribúty produktu, kusy produktu, atribúty týchto kusov a históriu zmien.

- **20. Tabuľka produktov**

  Kliknutím na tlačítko *Produkty* v bočnom menu si zobraziť všetky vytvorené produkty v domácnosti, a informácie o nich. Možnosť zobrazenia si tabuľky produktov aj z detailu ich kategórie. V tabuľke vyskúšať filtrovanie podľa rôznych stĺpcov, napríklad podľa kapacity či podľa umiestnenia kusov. Ďalej overiť správnu funkciu zoradenia podľa stĺpca, napríklad podľa názvu.

- **21. Vytvorenie kusov produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Vytvoriť kusy produktu po kliknutí na tlačítko *Kusy* v bočnom menu a následnom kliknutí na tlačítko *Vytvoriť kusy*. Zvoliť produkt, ktorého kus bude vytvorený. Po zvolení produktu sa automaticky vyplní typ merania a kapacita. Ďalej vyplniť zostávajúce množstvo v ks, dátum expirácie, popis kusu a lokáciu kusu. Zvoliť počet vytvorených kusov. Potvrdiť vytvorenie kusov stlačením tlačítka *Vytvoriť kusy*. Overiť, že sa vytvorí správny počet kusov.

- **22. Editácia kusov produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Editovať kus produktu z detailu produktu v sekcii *Kusy* po kliknutí na symbol ceruzky pri danom kuse. Po vykonaných úpravách kliknúť na tlačítko *Upraviť kus*.

- **23. Výmaz kusov produktu**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Vymazať jeden alebo viac kusov produktu po označení daných kusov v detaile produktu, v sekcii *Kusy*. Kliknúť na tlačítko *Zmazať vybrané* a potvrdiť akciu stlačením tlačítka *Zmazať*.

- **24. Prehľad kusov produktu**

  Zobraziť si prehľad kusov produktu z detailu produktu v sekcii *Kusy*, respektíve po kliknutí na tlačítko *Kusy* v bočnom menu. V tabuľke vyskúšať filtrovanie podľa rôznych stĺpcov, napríklad podľa množstva či podľa lokácie. Ďalej overiť správnu funkciu zoradenia podľa stĺpca, napríklad podľa názvu produktu.

- **25. Vytvorenie atribútu kategórie**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Vytvoriť atribút po kliknutí na tlačítko *Kategórie* v bočnom menu, kliknutím na sekciu *Atribúty* v detaile kategórie, a následnom kliknutí na tlačítko *Vytvoriť atribút*. Zadať názov nového atribútu, jeho popis a predvolenú hodnotu. Možnosť aplikovať predvolenú hodnotu na všetky už existujúce produkty kategórie. Potvrdiť vytvorenie atribútu kliknutím na tlačítko *Vytvoriť atribút*.

- **26. Vytvorenie atribútu pre kusy produktov**

  Pre vykonanie daného scenára je nutné mať oprávnenie na úpravu modulu inventára v danej domácnosti. Kliknúť na tlačítko *Produkty* v bočnom menu, zvoliť si produkt a kliknúť na sekciu *Atribúty kusov*. Kliknúť na tlačítko *Vytvoriť atribút* a zadať názov, popis a predvolenú hodnotu atribútu. Taktiež možnosť aplikovať predvolenú hodnotu na všetky už existujúce kusy produktu. Potvrdiť vytvorenie atribútu kliknutím na tlačítko *Vytvoriť atribút*.

- **27. História zmien**

  Kliknúť na *Produkty* v bočnom menu, a po zobrazení detailu daného produktu si zobraziť históriu zmien kliknutím na sekciu *História zmien*. Zobraziť si vykonané zmeny v množstve produktu. Overiť, že zoznam je zoradený od najnovšieho po najstarší záznam.

- **28. Upozornenia na nízke zostávajúce množstvo produktu a expiráciu**

  Nastaviť upozornenie na nízke zostávajúce množstvo produktu a blížiacu sa expiráciu úpravou alebo vytvorením nového produktu. Vytvoriť kus daného produktu s množstvom a expiráciou nižšou, než je nastavená spodná hranica produktu. Počkať do polnoci a v emailovej schránke skontrolovať, že sa správne odoslalo upozornenie.

## 5.3 Nasadenie

Finálnou časťou práce je príprava k produkčnému nasadeniu. Nasadenie aplikácie je proces, ktorého cieľom je dostať aplikáciu do verejne prístupného stavu. V rámci kontinuálnej integrácie (sekcia 4.1.6) sa ako posledný krok spolu s kolegom Dávidom pridalo zostavenie obrazu Docker. Obraz Docker je balík, ktorý obsahuje zostavenú aplikáciu, stiahnuté závislosti a rôzne nastavenia. Pomocou neho je potom možné spustiť aplikáciu i na iných zariadeniach, bez nutnosti inštalácie programovacích jazykov a ďalších závislosti. Zostavenie aplikácie HouseKeeper prebieha tak, že sa najprv zostaví klientská časť aplikácie, potom serverová, a nakoniec sa do verejnej zložky serverovej časti vloží zostavená klientská časť. Serverová časť je tak počas behu aplikácie zodpovedná za odosielanie klientskej časti užívateľom. Vedúci práce bude schopný pomocou tohto zostaveného obrazu nasadiť aplikáciu na webový server.

# Kapitola 6
# Záver

Hlavným cieľom tejto bakalárskej práce bolo navrhnutie a vytvorenie systému pre správu domácností, ktorý by ponúkal aj správu domáceho inventára. Na začiatku boli určené funkčné a nefunkčné požiadavky kladené na aplikáciu. Ďalej bola vykonaná analýza existujúcich podobných riešení, na základe ktorej bolo zistené, že žiadna z aplikácií nespĺňa určené požiadavky. Potom bol vytvorený model prípadov použitia aplikácie a doménový model. V ďalšej časti práce bol vytvorený návrh budúcej aplikácie a boli zvolené vhodné technológie. Na implementáciu serverovej časti bol zvolený programovací jazyk PHP a framework Laravel. Na implementáciu klientskej časti bol zas zvolený jazyk Typescript a framework React. Neskôr bola navrhnutá architektúra aplikácie, pričom bol vytvorený aj databázový model. Dôležitou časťou práce bola implementácia jadra a modulu domáceho inventára. Zároveň bola aplikácia riadne otestovaná a pripravená k produkčnému nasadeniu. Všetky určené ciele práce boli teda splnené.

Vyvinutá aplikácia s názvom HouseKeeper ponúka riešenie pre komplexnú správu domácnosti – umožňuje správu domáceho inventára a zároveň ponúka možnosť sledovania energií domácnosti. Aplikácia umožňuje vytváranie ľubovoľného počtu domácností a ich zdieľanie s inými užívateľmi. V aplikácii si môžu užívatelia zaznamenávať zostávajúce množstvo a zásoby akýchkoľvek produktov a kusov. Je možné členiť tieto produkty do kategórií a zobrazovať si ich v prehľadnej tabuľke. Aplikácia taktiež ponúka rôzne funkcionality, ako možnosť vytvorenia vlastných atribútov pre produkty a kusy, nastavenie emailových notifikácií pri blížiacom sa dátume expirácie kusov, a taktiež upozornenie na nízke množstvo produktu.

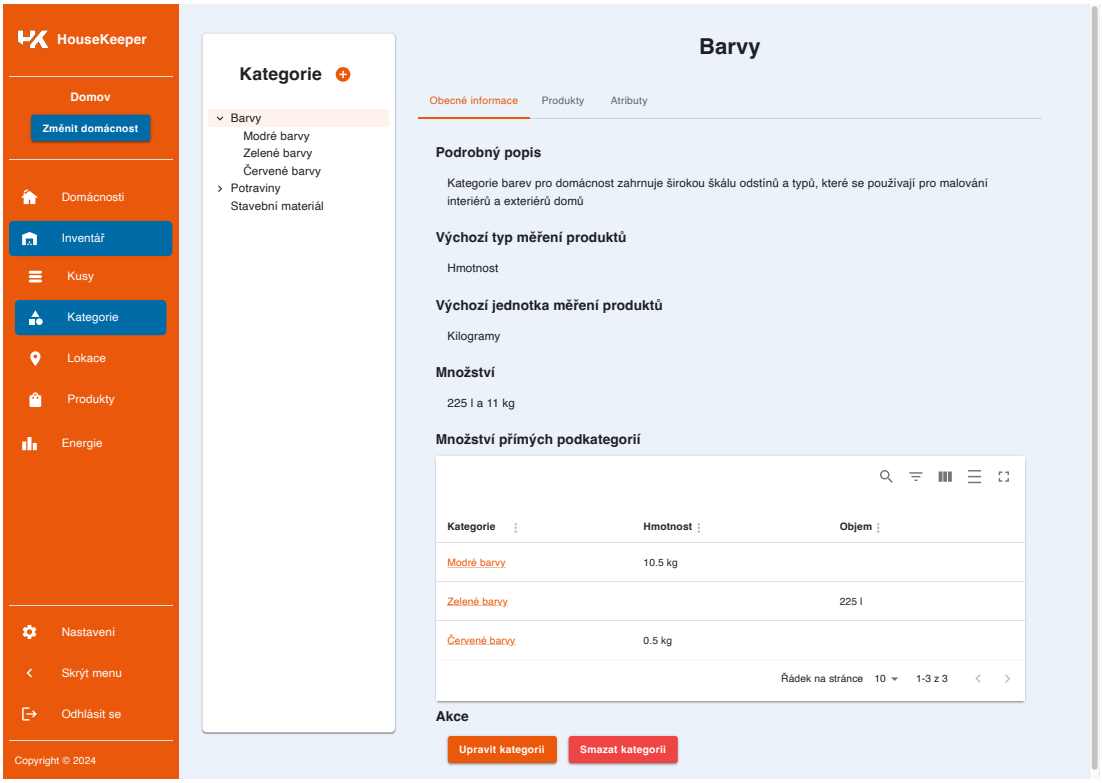
Možným rozšírením aplikácie by mohol byť modul pre správu úloh, či modul pre plánovanie údržby. V module domáceho inventára by bolo možné pridať funkcionalitu, ktorá by v prípade výmazu kategórie umožnila ľahko presunúť jej produkty do inej kategórie. Taktiež by bola žiadaná možnosť pridávať vlastné fotografie pre produkty a kusy.

Ďalším možným vylepšením by mohlo byť zvýšenie bezpečnosti aplikácie pri obnove hesla. V súčasnosti aplikácia pri zadaní nezaregistrovanej adresy informuje o tom, že daná adresa nebola nájdená. Vylepšenie by spočívalo v zmene hlášky na hlášku: „Na daný e-mail boli odoslané inštrukcie pre obnovu hesla, ak je táto adresa registrovaná“.

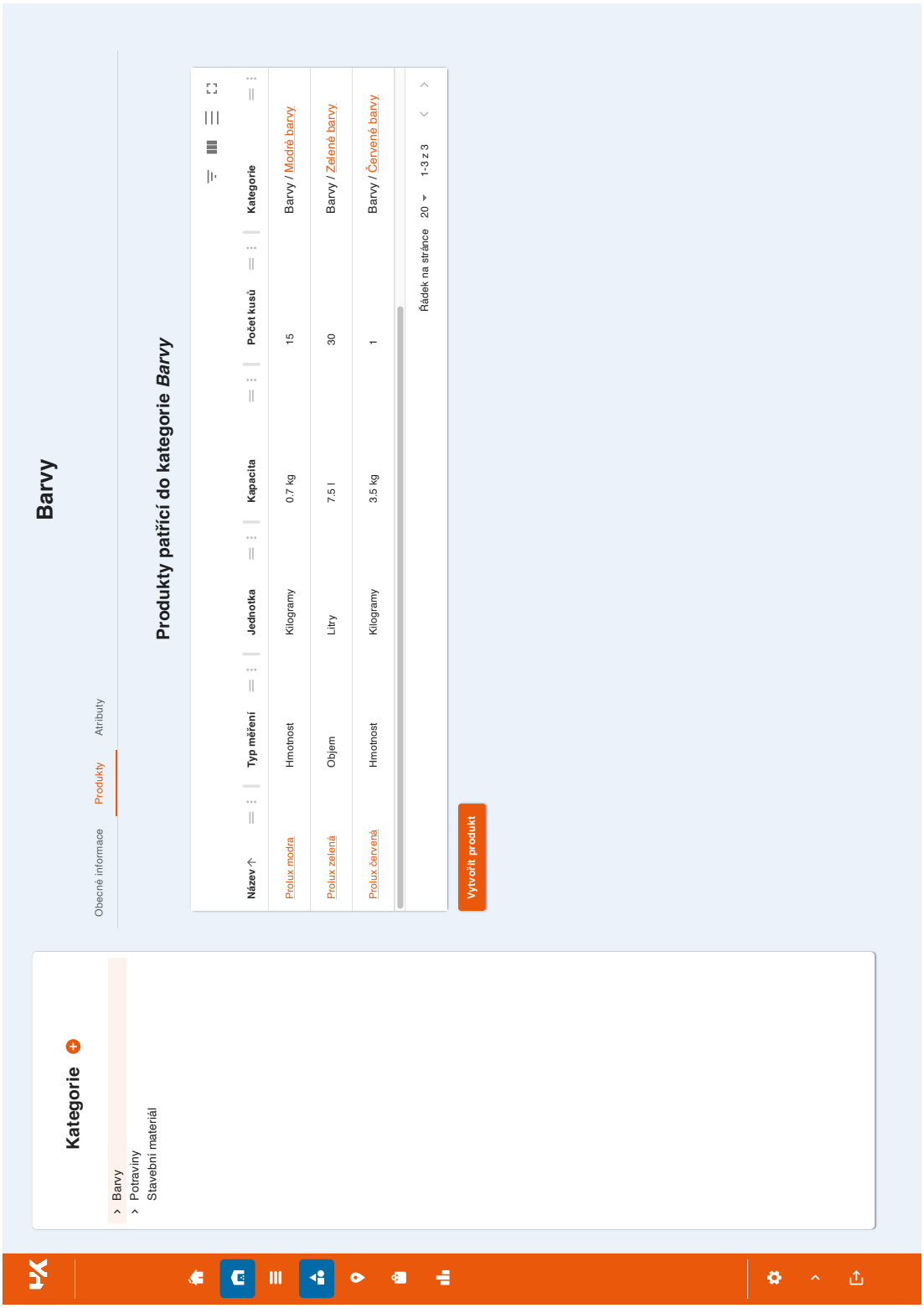
V neposlednom rade by bolo vhodné vytvoriť aj mobilnú aplikáciu, ktorá by umožnila jednoduchú každodennú správu inventára – komplexnejšiu správu by naďalej ponúkala webová aplikácia. V mobilnej aplikácii by bolo možné aj priamo zhotovovať fotografie produktov a ukladať ich do aplikácie.

# Dodatok A

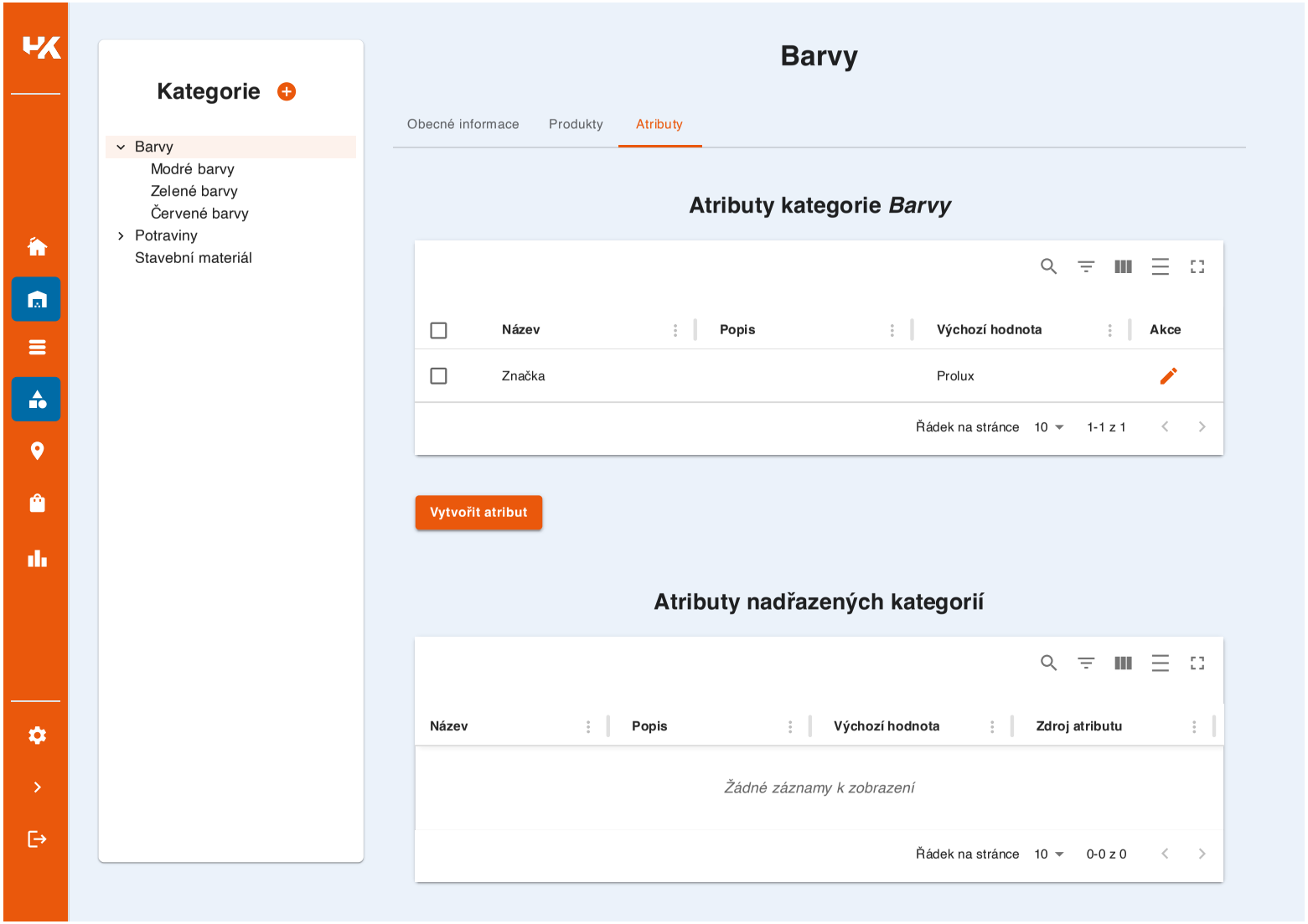
# Vybrané ukážky aplikácie




**Obr. A.1** Ukážka detailu kategórie

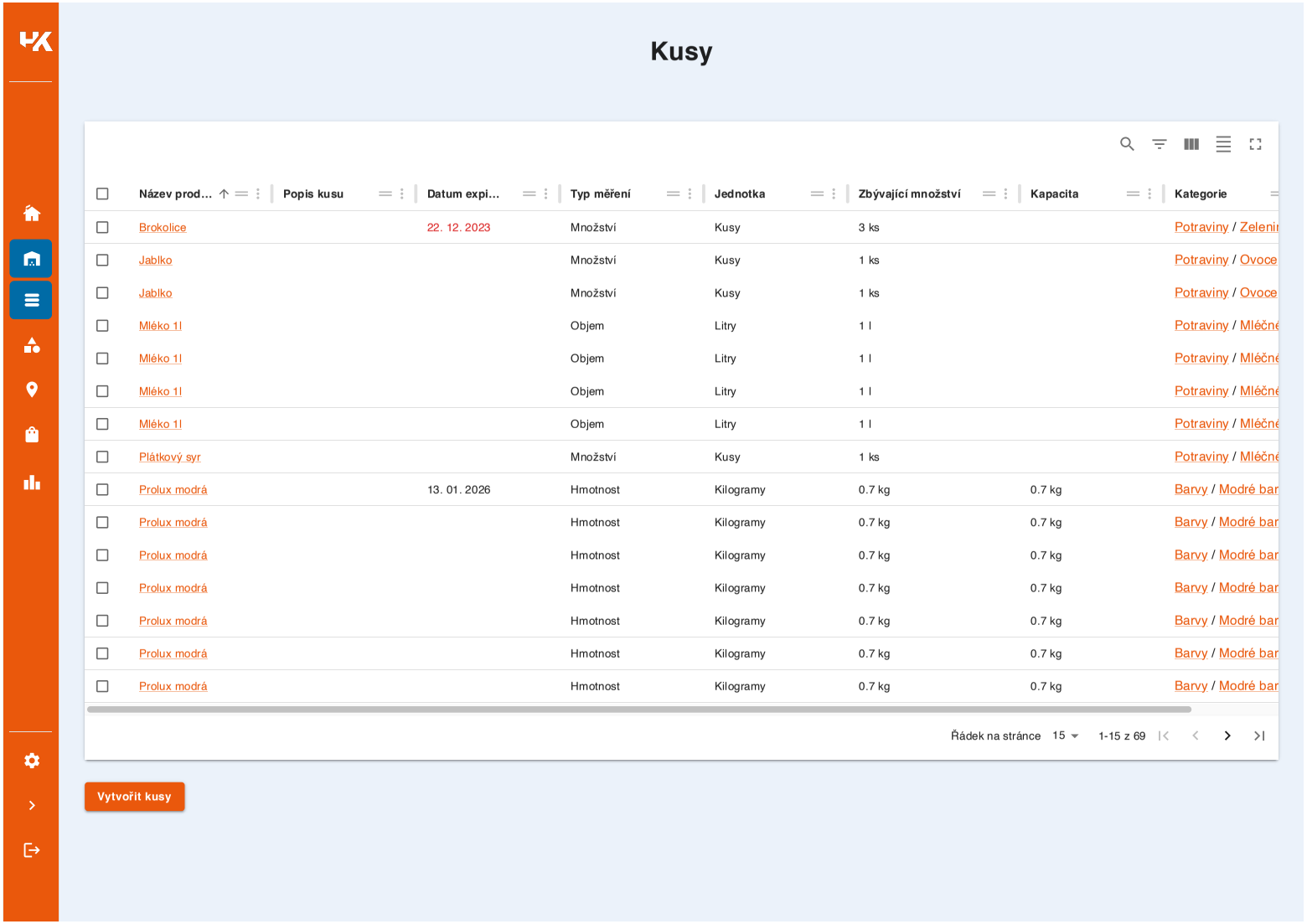



**Obr. A.2** Ukážka tabuľky produktov konkrétnej kategórie




**Obr. A.3** Ukážka tabuliek atribútov kategórie

**Obr. A.4** Ukážka formuláru pre úpravu produktu




**Obr. A.5** Ukážka tabuľky všetkých kusov v domácnosti

# Bibliografia


1. SORTLY INC. *Sortly: Inventory Simplified* [online]. © 2023. [cit. 2023-10-28]. Dostupné z : `https://www.sortly.com`.
2. SORTLY INC. *Pricing Plans - Sortly — Free 14 Day Trials Available!* [online]. © 2023. [cit. 2023-10-28]. Dostupné z : `https://www.sortly.com/pricing`.
3. MEMENTODB INC. *Memento Database* [online]. © 2020. [cit. 2023-10-28]. Dostupné z : `https://mementodatabase.com`.
4. NONZEROAPPS. *Smart Inventory — Smart Inventory Web* [online]. © 2021. [cit. 2023-10-28]. Dostupné z : `https://smartinventory.nonzeroapps.com`.
5. ITEMTOPIA INC. *Itemtopia* [online]. [B.r.]. [cit. 2023-10-28]. Dostupné z : `https://www.itemtopia.com`.
6. BAZZANI, Daniel. *Home Inventory+ on the App Store* [online]. 2021. [cit. 2023-10-28]. Dostupné z : `https://apps.apple.com/us/app/home-inventory/id1537446653`.
7. RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. *The Unified Modeling Language Reference Manual*. Second. Boston: Addison-Wesley, 2005. ISBN 0-321-24562-8.
8. ARLOW, Jim; NEUSTADT, Ila. *UML 2 And The Unified Process: Practical Object-Oriented Analysis And Design*. Second. Boston: Addison-Wesley, 2006. ISBN 0-321-32127-8.
9. FOWLER, Martin; RICE, David; FOEMMEL, Matthew; HIEATT, Edward; MEE, Robert; STAFFORD, Randy. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2002. ISBN 0-321-12742-0.
10. PHP GROUP. *PHP: Hypertext Preprocessor* [online]. © 2001-2023. [cit. 2023-11-07]. Dostupné z : `https://www.php.net`.
11. JENČO, Dávid. *Webová aplikace pro domácí evidenci spotřeby energií*. Praha, 2024.
12. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns Elements of Reusable Object-Oriented Software*. Westford, Massachusetts: Addison-Wesley, 1995. ISBN 0-201-63361-2.
13. STACK EXCHANGE INC. *Stack Overflow Trends* [online]. © 2023. [cit. 2023-11-11]. Dostupné z : `https://insights.stackoverflow.com/trends?utm_source=so-owned&utm_medium=blog&utm_campaign=trends&utm_content=blog-link&tags=laravel%2Csymfony%2Ccodeigniter`.
14. SYMFONY SAS. *Symfony, High Performance PHP Framework for Web Development* [online]. [B.r.]. [cit. 2023-11-11]. Dostupné z : `https://symfony.com`.
15. STACK EXCHANGE INC. *Stack Overflow Developer Survey 2023* [online]. 2023. [cit. 2023-11-07]. Dostupné z : `https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies`.

16. LARAVEL LLC. *Laravel - The PHP Framework For Web Artisans* [online]. © 2011-2023. [cit. 2023-11-11]. Dostupné z : https://laravel.com.
17. CODEIGNITER FOUNDATION. *Welcome to CodeIgniter* [online]. © 2023. [cit. 2023-11-12]. Dostupné z : https://codeigniter.com.
18. VETTOR, Rob; SMITH, Steve. *Cloud native .net apps for azure*. Redmond, Washington: Microsoft Developer Division, © 2023.
19. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: The world's most advanced open source database* [online]. © 1996-2023. [cit. 2023-11-13]. Dostupné z : https://www.postgresql.org.
20. ELLINGWOOD, Justin. *What is an ORM (Object Relational Mapper)?* [online]. [B.r.]. [cit. 2023-11-13]. Dostupné z : https://www.prisma.io/dataguide/types/relational/what-is-an-orm.
21. LARAVEL LLC. *Database: Getting Started - Laravel 10.x - The PHP Framework For Web Artisans* [online]. © 2011-2023. [cit. 2023-11-13]. Dostupné z : https://laravel.com/docs/10.x/database#introduction.
22. LARAVEL LLC. *Eloquent - laravel-docs* [online]. [B.r.]. [cit. 2023-11-14]. Dostupné z : https://laravel-docs.readthedocs.io/en/latest/eloquent.
23. META PLATFORMS, INC. *In-Depth Overview — Flux* [online]. 2023. [cit. 2023-12-12]. Dostupné z : https://facebookarchive.github.io/flux/docs/in-depth-overview.
24. DAN ABRAMOV AND THE REDUX DOCUMENTATION AUTHORS. *Redux Fundamentals, Part 2: Concepts and Data Flow — Redux* [online]. © 2015–2023. [cit. 2023-12-12]. Dostupné z : https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow.
25. SOFTWARE FREEDOM CONSERVANCY. *Git* [online]. [B.r.]. [cit. 2023-11-18]. Dostupné z : https://git-scm.com.
26. GITLAB B.V. *Plan and track work — GitLab* [online]. 2023. [cit. 2023-11-26]. Dostupné z : https://docs.gitlab.com/ee/topics/plan_and_track.html.
27. SONARSOURCE S.A. *Code Quality, Security & Static Analysis Tool with SonarQube — Sonar* [online]. © 2008-2023. [cit. 2023-11-18]. Dostupné z : https://www.sonarsource.com/products/sonarqube.
28. SPARX SYSTEMS PTY LTD. *UML modeling tools for Business, Software, Systems and Architecture* [online]. © 2000-2023. [cit. 2023-11-26]. Dostupné z : https://sparxsystems.com/.
29. MDN CONTRIBUTORS. *Using HTTP cookies - HTTP — MDN* [online]. 2023. [cit. 2023-11-19]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.
30. OWASP FOUNDATION, INC. *HttpOnly — OWASP Foundation* [online]. 2023. [cit. 2023-11-19]. Dostupné z : https://owasp.org/www-community/HttpOnly.
31. CHEATSHEETS SERIES TEAM. *HTML5 Security - OWASP Cheat Sheet Series* [online]. 2022. [cit. 2023-11-27]. Dostupné z : https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html#local-storage.
32. IBM. *What is a REST API? — IBM* [online]. [B.r.]. [cit. 2023-11-19]. Dostupné z : https://www.ibm.com/topics/rest-apis.
33. MICROSOFT. *TypeScript: JavaScript With Syntax For Types.* [online]. © 2012-2023. [cit. 2023-11-20]. Dostupné z : https://www.typescriptlang.org.
34. META OPEN SOURCE. *React* [online]. © 2023. [cit. 2023-11-20]. Dostupné z : https://react.dev.

35. MATERIAL UI SAS. *MUI: The React component library you always wanted* [online]. © 2023. [cit. 2023-11-27]. Dostupné z : `https://mui.com`.

36. TAILWIND CSS. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.* [online]. [B.r.]. [cit. 2023-11-27]. Dostupné z : `https://tailwindcss.com`.

# Obsah prílohy

- `README.rtf` ..... stručný popis obsahu
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `BE` ..... zdrojové kódy serverovej časti
    - `FE` ..... zdrojové kódy klientskej časti
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
- `docs` ..... dokumentácia kódu a projekt Enterprise Architect