FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	EasyPacking
Student:	Bc. Jan Kabela
Vedoucí:	doc. RNDr. Josef Kolář, CSc.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

1. **Seznamte se** s existujícími aplikacemi, které umožňují záznam, doporučování a hodnocení různých variant "balicích" seznamů (typicky např. pro účely cestování).
2. **Vytvořte** vhodný matematický model, formulujte smysluplné problémy nad tímto modelem a určete složitost jejich řešení.
3. **Navrhněte a realizujte** mobilní aplikaci pro Android (včetně backend serveru) podporující záznam "balicích" seznamů s těmito funkcionalitami:
Běžný uživatel - vytvoření účtu, vytvoření a úpravy "balicích" seznamů, vyhledávání v seznamech ostatních uživatelů na základě věku, pohlaví, destinace a činností spojené s cestou (chození po horách, dovolená u moře atd.). Při zobrazení seznamu se jednotlivé položky zobrazují seskupeny podle svého typického umístění v domě.
Správce - přidávání položek do číselníků destinací, předmětů, lokalit předmětů a činností spojených s cestou, dále vše, co může běžný uživatel.
Jednotlivé funkcionality řádně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 16. dubna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

EasyPacking

Bc. Jan Kabela

Katedra softwarového inženýrství
Vedoucí práce: doc. RNDr. Josef Kolář, CSc.

31. května 2020

Poděkování

Rád bych poděkoval panu doc. RNDr. Josef Kolářovi, CSc. za konzultace a pomoc při tvorbě této diplomové práce

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 31. května 2020 .....

Odkaz na tuto práci

Kabela, Jan. *EasyPacking*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato diplomová práce se zabývá vývojem aplikace EasyPacking pro OS Android. Aplikace by měla pomáhat uživatelům při balení na cesty. V aplikaci může uživatel vytvářet seznamy věcí, které si na cestu plánuje vzít. Tyto seznamy budou parametrizovány pomocí těchto kritérií: věk, pohlaví, cílová destinace, aktivity, jež uživatel plánuje v cílové destinaci provádět, způsob dopravy a předpokládané počasí. Vytvořené seznamy budou k dispozici i ostatním uživatelům, kteří je mohou použít pro svou cestu nebo se jimi mohou inspirovat. V rámci vyhledávání budou uživateli doporučovány předměty, na základě vytvořených seznamů ostatních uživatelů. Doporučování bude probíhat za pomoci statistického modelu a za pomoci neuronové sítě. Jednotlivé předměty budou rozděleny podle kategorií (oblečení, hygiena, ...). Pro realizaci aplikace bude nutné vytvořit aplikační server. Na tomto serveru se budou zpracovávat data potřebná pro chod aplikace. Zároveň bude server spolupracovat s databází pro ukládání jednotlivých seznamů. Pro komunikaci mezi serverem a aplikací je použit protokol HTTPS. Data se přenášejí ve struktuře JSON. Pro databázi je zvolena technologie PostgreSQL. Aplikace je vytvořená pomocí jazyku Kotlin. Server je vytvořen v jazyku Ruby on Rails. Výsledkem této práce je otestovaná aplikace EasyPacking spolu s aplikačním serverem a databází. Zároveň budou v rámci práce porovnány obě doporučovací metody.

Klíčová slova Kotlin, Ruby on Rails, Postgresql, JSON, JSONWebToken, Neural network

Abstract

The goal of the present Diploma thesis is to develop an application EasyPacking for OS Android. The application should help users with packing when planning to travel. Users will be able to create packing lists with items they are going to pack. Users will add criteria to these packing lists. The criteria consist of age, gender, destination, activities that users will be doing during their vacation, means of transport and expected weather. Users will be able to search for other's packing lists based on their criteria. During the search a user will be able to choose between two suggesting methods. First method will be statistical, second will be neural network. The items in packing list will be split into categories (clothing, hygiene,...). It will be necessary to develop an application server. The application server will be providing the application with data. Also it is necessary to create a database for storing the data. The communication between the application server and the application will be done by HTTPS protocol. The data will be transferred in JSON structure. The database will be created using PostgreSQL. The application will be developed in Kotlin. The functionality of the application will be tested. The result of this thesis will be the application EasyPacking with the application server and the database. Also there will be a summary of success of the two suggesting methods.

Keywords Kotlin, Ruby on Rails, Postgresql, JSON, JSONWebToken, Neural network

Obsah

Seznam obrázků

Úvod

Tato diplomová práce se týká aplikací pro vytváření seznamů předmětů, které si uživatelé berou s sebou na cesty. Zároveň tato práce obsahuje návrh, implementaci a testování takovéto aplikace.

Přijde mi zbytečné vytvářet si pro každou cestu nový seznam věcí. které si sbalit s sebou. Je pravděpodobné, že pokud se rozhodnu pro nějakou cestu a s ní spojené aktivity, nebudu první, kdo takovou tu cestu podnikl. Rozhodl jsem se tedy vytvořit aplikaci, do níž budou uživatelé moci ukládat seznamy sbalených věcí a vzájemně je sdílet.

Dnes existuje několik aplikací s tímto účelem. V části Analýza dosavadních řešení některé z těchto aplikací zkouším a popisuji jejich výhody a nevýhody. Aplikace jsem vybral podle jejich počtu stažení a hodnocení. Na základě toho pak analyzuji, jak aplikaci EasyPacking navrhnout.

Dále v této práci analyzuji možná řešení aplikace EasyPacking z pohledu matematiky. Zaměřuji se zde na počet dat potřebných pro plnou funkčnost aplikace. Vybrané matematické modely následně používám pro zjištění počtu potřebných seznamů pro naplnění aplikace. V práci se zaměřuji i na matematické modely pro doporučování předmětů na základě zadaných kritérií. Jedná se o statistickou metodu na základě relativní četnosti a neuronovou síť. Dále se pak zaměřuji na tyto dvě metody, analýzu jejich řešení, implementaci a porovnání úspěšnosti.

V této práci jsem také přemýšlel nad možnostmi, jak jednotlivé části implementovat a jaké technologie k tomu použít. Řešil jsem možnosti pro aplikační server, databázi a samotnou aplikaci. Nejprve jsem analyzoval technologie a postupy, které se v dnešní době používají. Na základě jejich hodnocení jsem zvolil nejvhodnější technologie pro mou aplikaci.

V neposlední řadě jsme popsal samotnou implementaci jednotlivých částí. Kapitola Realizace obsahuje návrh databáze zohledňující předchozí průzkum požadavků. Dále obsahuje popis implementace aplikačního serveru v Ruby on Rails a vývoj aplikace v jazyce Kotlin. Spolu s popisem návrhu aplikace jsou zde zmíněny také návrhy jednotlivých obrazovek aplikace.

V závěru práce je popsáno, jak byly jednotlivé části testovány. V časti Testování jsou uvedeny testovací scénáře, dle kterých byla aplikace testována. Aplikace byla dále testována také vybraným vzorkem potenciálních koncových uživatelů. Aplikační server byl testován již v průběhu vývoje, jelikož se jednalo o test driven development.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit aplikaci EasyPacking pro OS Android. Tato aplikace by měla svým uživatelům usnadnit balení na cesty. Datovou základnu aplikace tvoří seznamy předmětů,které by si cestovatel měl na specifickou cestu vzít - uživatelé tyto seznamy vytvářejí svým potřebám na míru. Následně jsou seznamy archivovány, takže jsou nadále dostupné jak autoru balicího seznamu, tak i ostatním uživatelům aplikace. Konečným cílem je pak postupné ustupování od vytváření vlastních seznamů a využívání seznamů již vytvořených. Jelikož se jednotlivé seznamy vážou k daným prázdninovým lokacím, u často navštěvovaných destinací je předpokládáno, že k opakovanému používání již vytvořených seznamů dojde dříve, než u méně oblíbených turistických míst.

Kromě dané cílové destinace budou jednotlivé seznamy blíže specifikovány pomocí věku, pohlaví, plánovaných aktivit, dopravního prostředku a počasí. Předměty se budou zobrazovat uživatelům podle kategorie, do které spadají, a následně podle aktivity, pro kterou jsou určeny.

Jedním z cílů je nalézt matematické modely, které lze použít při doporučování předmětů podle zadaných kritérií. Pokusím se popsat tyto modely, implementovat je a následně porovnat jejich funkčnost.

Kvůli zmíněné archivaci a vyhledávání bude nutné vytvořit pro aplikaci i serverovou část. Celá implementace aplikace se bude skládat z těchto částí:

- Vytvoření databáze
- Vytvoření serveru napojeného na databázi s následujícími funkcionalitami:
  - Vytvoření účtu
  - Vytváření seznamů

- Editace mých seznamů
- Mazání mých seznamů
- Vyhledávání seznamů na základě věku, pohlaví, cílové destinace, aktivit, typu dopravy a počasí

Kapitola 2

Analýza a návrh

Zprvu se zaměřím na již existující aplikace, které budu brát jako vzor, a pokusím se jejich funkcionalitu vylepšit. Díky uživatelským komentářům se mohu inspirovat funcionalitami, které existujícím aplikacím chybí, ale uživatelé by je uvítali. O tyto funkcionality se pak pokusím aplikaci EasyPacking obohatit.

2.1 Současná řešení

2.1.1 PackPoint travel packing list

Na začátku uživatel zadá lokalitu, kam cestuje, datum odjezdu, délku pobytu a typ cesty (dovolená nebo pracovní cesta). Následně vybere aktivity, které plánuje provádět v cílové destinaci. V základní verzi aplikace má na výběr z 15 aktivit. Pokud si uživatel připlatí 90 Kč za verzi premium, tak může zadat i vlastní aktivity. Po vyplnění požadovaných kritérií je vygenerován seznam doporučených věcí, které by se uživateli mohly na zahraniční cestě hodit. Následně má uživatel možnost postupně seznam překontrolovat a rozhodnout, zda si předmět opravdu s sebou zabalí. Pokud se z nějakého důvodu rozhodne si danou věc s sebou nevzít, jednoduše ji pomocí posunutí na stranu ze seznamu odstraní. V premium verzi může uživatel do seznamu věci také přidávat. Upravený seznam je možné sdílet pomocí e-mailu, sociálních sítí nebo cloudových úložišť.

Nevýhodou aplikace je, že uživatel musí vždy seznam vytvořit. Další nevýhodou je absence jiných jazyků než angličtiny. Mnoho funkcionalit je možno využít až při koupi premium verze.

2.1.2 PacKing

Pro vytvoření seznamu v této aplikaci uživatel zadá délku pobytu, pohlaví, dopravní prostředek, kterým bude cestovat, počasí, které předpokládá během pobytu a nakonec vybere aktivity, jež bude během pobytu provádět. Aplikace na základě těchto parametrů vygeneruje seznam věcí, které si uživatel má sbalit s sebou. Jednotlivé předměty jsou rozděleny do kategorií (například oblečení, hygiena nebo dokumenty). Uživatel se následně rozhodne, zda si daný předmět vzít s sebou nebo nikoliv. Jednotlivé věci je možné editovat, měnit jejich počet nebo změnit kategorii.

Nevýhodou je, že seznam není možné sdílet. Sdílení je možné pouze v premium verzi. Dále aplikace generuje ohromné množství předmětů, které uživatel musí projít a rozhodnout se, zda si je sbalit či nikoliv. Toto procházení zabere tolik času, že by dle mého názoru bylo rychlejší si seznam napsat ručně. Uživatel si může v základní verzi vytvořit jen jeden seznam.

Tato aplikace obsahuje podobná kritéria, která jsem zvolil pro aplikaci EasyPacking. S kritérii jsou spojené i jednotlivé předměty. Je tedy možné použít předměty z aplikace PacKing pro naplnění číselníků aplikace EasyPacking.

2.1.3 Packing List Lite

V této aplikaci si uživatel může vytvořit seznamy tak, jako by to dělat na papíře. Je možné přidat kategorie. Do kategorií se pak vkládají jednotlivé položky. Takto vytvořené seznamy jsou uloženy v aplikaci.

Aplikace nijak nenapomáhá vytváření seznamů. Uživatel si musí vše napsat sám. Jednotlivé seznamy není možné sdílet. Z uvedených aplikací je tato nejslabší.

2.1.4 Shrnutí současných řešení

Budu se věnovat pouze prvním dvěma zmíněným aplikacím. V rámci uživatelských komentářů jsem objevil jeden hlavní nedostatek, a tím byla absence jiných jazyků než angličtiny. Z mého pohledu je také nedostatkem, že uživatel musí seznam pro každou novou destinaci vytvářet. Stejně tak i pro již navštívenou destinaci, ale s jinými aktivitami. Jelikož se předměty, které si uživatelé berou na dovolenou, můžou lišit, tak se v obou aplikacích generuje velké množství doporučených předmětů, a to i s ohledem na rozlišení podle aktivit. I přes tyto nedostatky aktuálně patří uvedené aplikace mezi jedny z nejvíce používaných aplikací pro balení na cesty. Z tohoto důvodu se hodí pro inspiraci vytváření aplikace EasyPacking.

2.2 Matematický model

2.2.1 Počet záznamů v databázi

Podle počtu jednotlivých kritérií je možné přibližně určit, kolik seznamů by mělo minimálně existovat, aby byly pokryty všechny možné kombinace. Z tohoto údaje je pak možné odhadnout problémy, které by mohly nastat.

Kritéria používaná pro specifikaci cesty můžeme rozdělit do dvou skupin a formálně vyjádřit následujícím seznamem:

$$K_1, K_2, ...K_m, J_1, J_2, ...J_n$$

Pro každé kritérium K_i z první skupiny bude specifikace vyjádřena podmnožinou nabízených hodnot, tedy $k_i \subseteq K_i$. Pokud je znám celkový počet možností $|K_i|$ kritéria K_i, je celkový počet možných hodnot tohoto kritéria dán celkovým počtem podmnožin, tedy mohutností potenční množiny $2^{|K_i|}$.

Pokud neuvažuji všechny podmnožiny, ale například jen podmnožiny obsahující $1, 2, ..., r_i$ možných hodnot, pak použiji vzorec:

$$\binom{|K_i|}{1} + \binom{|K_i|}{2} + \binom{|K_i|}{3} + ... + \binom{|K_i|}{r_i}$$

Druhá skupina kritérií $J_1, J_2, ...J_m$ představuje taková kritéria, kde z nabízené množiny hodnot J_i mohu vždy zvolit právě jednu. Mám tedy přesně $|J_i|$ možností. Celkový počet možných kombinací kritérií N je dán výrazem:

$$N = \prod_{i=1}^{m}[\binom{|K_i|}{1} + \binom{|K_i|}{2} + \ldots + \binom{|K_i|}{r_i}] * |J_1| * |J_2| * \ldots * |J_n|$$

Tímto vzorcem jsem schopen zjistit, jak velký počet seznamů je minimálně potřeba k obsažení všech možných kombinací kritérií.

V navrhované aplikaci předpokládám postupně archivovat v databázi konkrétní kombinace kritérií a jim odpovídající seznamy věcí doporučených pro danou cestu. Můžeme si tedy položit otázku, kolik bude třeba archivovat takových záznamů, abychom měli v databázi alespoň jeden záznam pro každou možnou kombinaci kritérií. Budeme-li předpokládat, že všechny kombinace kritérií mají stejnou pravděpodobnost výskytu, můžeme využít řešení základní varianty úlohy označované jako Coupon collector's problem. [1].

Tuto úlohu lze ilustrovat na situaci, kdy určitý řetězec prodejen přibalí ke každému nákupu nad např. 100 Kč náhodně vybranou kartičku z pevně stanovené sady obsahující n různých typů kartiček.

Typickými "sběrateli"takových kartiček jsou děti, které pak nutí rodiče k nákupům v daném řetězci. Kolik nákupů je třeba uskutečnit, aby se dítě mohlo před kamarády pochlubit kompletní sadou kartiček?

Řešení tohoto klasického problému kombinatorické statistiky je dáno výrazem pro střední hodnotu počtu nákupů

$$E[X] = n \cdot H_n,$$

kde H_n je tzv. n-té harmonické číslo, neboli n-tý částečný součet tzv. harmonické řady:

$$H_n = 1 + 1/2 + 1/3 + ... + 1/n = \sum_{i=1}^{n} 1/i$$

Postačující aproximaci střední hodnoty $E[X]$ tedy představuje výraz vycházející z aproximace n-tého harmonického čísla

$$n \cdot (\ln(n) + \gamma) + 1/2,$$

kde $\gamma = 0,57721566...$ je tzv. Euler-Mascherova konstanta.

Další otázku, kterou si ohledně archivace jednotlivých seznamů můžu klást, je při jakém počtu uložených seznamů se dosáhne stanovené pravděpodobnosti, že se mezi nimi vyskytne dvojice seznamů s identicky stejnými kombinacemi kritérií.

Zde se můžeme inspirovat řešením tzv. Narozeninového paradoxu, který se zabývá otázkou, při jakém počtu osob v místnosti je alespoň 50% pravděpodobnost, že mezi nimi budou dvě osoby, které slaví narozeniny ve stejný den.

Jako paradox totiž působí skutečnost, že k tomu stačí pouhých 23 osob, přestože (nepřestupný) rok má plných 365 dní. [2].

Pro řešení této úlohy využijeme principu doplňku - určíme pravděpodobnost, že v náhodně vybrané skupině k osob nebudou mít žádné dvě osoby stejné datum narozenin.

Tuto pravděpodobnost určíme jako poměr k-členných variací z 365 prvků ku počtu k-členných variací s opakováním z 365 prvků, tedy

$$p'(t_k) = 365 * 364 * ... * (365 - k + 1)/365^k$$

Pravděpodobnost doplňkového jevu - tedy že v dané skupině osob budou mít alespoň dvě osoby stejné datum narozenin - bude $p(t_k) = 1 - p'(t_k)$. Takto je

možné určit, že hodnotu $p(t_k) > 0,5$ dosáhneme již pro k = 23 a platí pro ni $p(t_{23}) = 0,507$.

Předpokládáme-li opět, že všechny kombinace hodnot kritérií mají stejnou pravděpodobnost výskytu, pak pravděpodobnost shody aspoň dvou kombinací kritérií v nějakém výběru k seznamů z celkového maximálního počtu n bude dána vztahem

$$p(t_n, k) = 1 - V(k, n)/V'(k, n) = 1 - n * (n - 1) * ... * (n - k + 1)/n^k,$$

kde $V(k, n)$, resp. $V'(k, n)$ jsou počty k-členných variací z n prvků, resp. k-členných variací s opakováním z n prvků

Tuto analýzu nyní aplikuji na předpokládané počty kritérií v aplikaci Easypacking.

Na příkladu aplikace EasyPacking se budu snažit ukázat, že díky uvedeným problémům nestačí pouze určit počet kombinací jednotlivých kritérií. Je také nutné uvažovat, jak pravděpodobné je, že uživatelé zadají seznam se stejnými kritérii. Pro jednoduchost nebudu uvažovat ve výpočtech, že některé destinace nebo aktivity jsou více využívány.

Pro určení potřebného počtu seznamů pro splnění všech kombinací kritérií budu zvažovat následující hodnoty. Na světě existuje přibližně 200 států. Řekněme že, v daných lokalitách mohou cestovatelé provozovat 50 různých aktivit. Nejčastější typy transportu do cílové destinace jsou tři - auto, veřejná doprava a letadlo. Počasí je vhodné rozdělit na pět možností: zima, chladno, polojasno, teplo a horko. Pro pohlaví uvažuji dvě možnosti. Posledním kritériem je věk. Aplikace vyhledává v rozsahu +/- 10 let od uživatelem zadaného věku. Pokud budu uvažovat uživatele v rozpětí 10 – 79 let, tedy uživatele, co již cestují a používají chytrý telefon, tak by mělo postačit 7 záznamů, které budou rovnoměrně rozmístěny v intervalu 10 – 79 let.

Z předešlé úvahy vychází následující:

- 50 aktivit
- 200 států
- 2 pohlaví
- 7 různých věkových kategorií
- 3 druhy transportu
- 5 možností pro počasí

V reálné situaci asi nelze očekávat, že by uživatel například zaškrtl všechny aktivity. Budu počítat s tím, že rozumné číslo pro maximální počet aktivit pro jednu cestu je 4. U počasí lze předpokládat, že nenastanou více než dvě možnosti. Takto se dostaneme, pomocí dosazení do dříve zmiňovaného vzorce, na

$$N = [\binom{50}{1} + \binom{50}{2} + \binom{50}{3} + \binom{50}{4}] \cdot 200 \cdot 2 \cdot 7 \cdot 3 \cdot [\binom{5}{1} + \binom{5}{2}] = 3,164805 \cdot 10^{10}$$

Dále použiji Coupon collector's problem pro zjištění počtu potřebných seznamů, tak, aby byla obsáhnuta všechna kritéria. Dosadím do předešlého vzorce.

$$E[X] = N * (\ln(N) * \gamma) + 0,5 = 7,834525 \cdot 10^{11}$$

Z předešlého výpočtu je vidět, že pro naplnění všech kritérií je potřeba přibližně dvacetkrát více záznamů.

Jak je očividné z výsledného výpočtu, takto veliký počet záznamů by bylo skoro nemožné uchovávat v databázi. Zároveň vyhledávání v takovémto objemu dat by bylo opravdu pomalé. Je tedy nutné omezit počet největších kriterií. Jimi jsou destinace a aktivity.

Dále mohu říci, že 50% pravděpodobnost výskytu dvou stejných kombinací kritérií má už množina obsahující 209 459 seznamů.

2.2.2 Doporučování

Pro doporučování použiji dvě metody. První metoda je statistická na základě předpokladů a důsledků. Druhou metodou bude neuronová síť.

2.2.2.1 Statistická metoda

Jak už bylo zmíněno, mám 6 typů kritérií. Všechny jejich kombinace K_i tvoří předpoklady.

Příkladem kombinace je: Potápění, Chorvatsko, Muž, 30-39 let, auto a horko.

Důsledek tvoří výskyt položky d_j v doporučených předmětech. Položky jsou rozděleny do kategorií: oblečení, hygiena, zdraví, apod.

Příklad důsledku je: Spodní prádlo: Ano

2.2.2.2 Model postavený na relativní četnosti

Model je založen na sledování relativní četnosti jednotlivých důsledků. Tento model nehodnotí vazby mezi důsledky ani nesleduje jejich kombinace.

Pro každou kombinaci K_i předpokladů budu zaznamenávat:

- četnost předpokladu n_{K_i} - kolikrát předpoklad nastal
- Pro každou položku důsledku D_j budu udržovat informaci n_{d_j} kolikrát nastala za předpokladu K_i

Relativní četnost r_{d_j} je podíl n_{d_j}/n_{K_i}

Příklad relativní četnosti - Předpoklad: Potápění, Chorvatsko, Muž, 30-39 let, auto a horko nastal celkem 30 krát.
Důsledek: Ponožky: Ano v 20 případech
Relativní četnost je 2/3

Stanovím si práh četnosti, nad kterým budu položku doporučovat za předpokladu K_i, že daná položka důsledku má být doporučena.

Příklad: práh stanovím na 50%. $2/3 > 0,5$. Předmět ponožky bude doporučen.

2.2.2.3 Chybějící hodnoty

Předpoklady budou uspořádány podle podobnosti. Podle podobnosti se dají uspořádat následující předpoklady:

Destinace Destinace se dají uspořádat podle geografické polohy. Provedu to tak, že destinace rozdělím do pěti skupin. Skupiny budou následovné: severní země, mírně severní země, země ve středu, mírně jižní země a jižní země. Pokud nebude existovat záznam pro danou destinaci mohu následně prohledat i destinace ve stejné kategorii.

Věk U předpokladu věk seřadím jednotlivé věkové intervaly vzestupně. V případě, že nebude existovat záznam, použiji okolní intervaly.

Aktivity Aktivity jako takové nelze rozdělit do podobných kategorií. Jelikož ale jsou jednotlivé předměty spojeny přímo s aktivitou, mohu použít podobnost celých seznamů, a to tak, že vyberu seznamy obsahující stejnou aktivitu a z nich pouze předměty pro danou aktivitu. Takto prohledám všechny aktivity zadané v předpokladu a vytvořím nový seznam.

Druh transportu, Počasí, Pohlaví Tyto tři předpoklady není možné rozdělit do podobných kategorií. Bude tedy vždy nutná kompletní shoda.

Postup dohledání V případě, že pro danou kombinaci předpokladů nebude existovat relativní četnost z důvodů zatím nevytvořeného seznamu, tak bude brán průměr ze seznamů s nejpodobnější kombinací kritérií, a to postupně po aktivitě. Podobnost bude brána hierarchicky, a to následovně:

- destinace
- věk

2.2.2.4 Potřebný počet seznamů

Při použití postupu doplnění chybějících záznamů, je možné pro prvotní naplnění databáze použít následující postup. Vždy pro seznam zvolím všechny aktivity i všechny možnosti počasí. Dále postupně vyberu kombinace kategorie destinace, pohlaví, věk a druh transportu. Tímto způsobem jsem schopný vytvořit takovou množinu seznamů, že pro každou kombinaci kritérií bude fungovat doporučení.

Po použití zmíněných úprav počtů jednotlivých kritérií dostanu následující čísla:

- 50 aktivit $->$ vybírám vždy všechny $= 1$
- 200 států $->$ $\binom{5}{1} = 5$
- 2 pohlaví $->$ 2
- 7 odlišných věkových kategorií $->$ $\left\lfloor \frac{7}{2} \right\rfloor = 3$
- 3 druhy transportu $->$ 3
- 5 počasí $->$ vybírám vždy všechny $= 1$

Počet takových to seznamů můžu vyčíslit následovně:

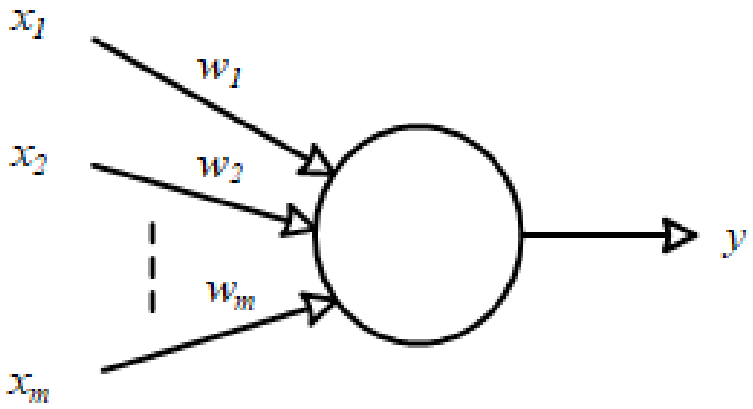
$$1 \cdot 5 \cdot 2 \cdot 3 \cdot 3 \cdot 1 = 90$$

Je tedy potřeba vytvořit 90 seznamů, aby pro každou kombinaci parametrů existoval seznam doporučených předmětů.

Doučení Doučení modelu probíhá automaticky zařazením nového výběru a dopočítáním četností.

2.2.2.5 Neuronová síť

Neuron Neuron je základním stavebním prvkem neuronové sítě. Historickým typem neuronu je perceptron (obrázek 2.1). Perceptron má vstupy $x_1, x_2, ..., x_m$, které jsou váženy váhami $w_1, w_2, ..., w_m$. Výstupem je pak hodnota y:

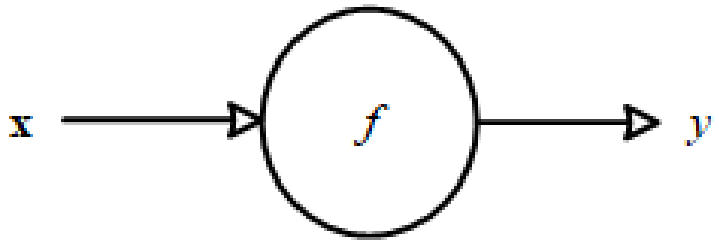


Obrázek 2.1: Perceptor

$$y = \begin{cases} 0, & \sum_j w_j x_j \leq p \\ 1, & \sum_j w_j x_j > p \end{cases}$$

Tj. 0 je výstupem v případě, že vážený vstup je menší nebo roven prahové hodnotě p. Nabývá hodnoty 1 v případě, že prahovou hodnotu přesáhne.

Obecně o neuronu můžeme říci, že aktivační funkce f převádí vektor vstupů $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ na hodnotu $y = f(\mathbf{x})$. Viz obrázek 2.2.



Obrázek 2.2: Obecný neuron

V případě perceptronu pak máme aktivační funkci:

$$f(\mathbf{x}) = \begin{cases} 0, & \sum_j w_j x_j + b \leq 0 \\ 1, & \sum_j w_j x_j + b > 0 \end{cases}$$

Rozdíl oproti předchozímu zápisu spočívá pouze v tom, že místo prahu p používáme posunutí *(bias)* b. Přitom platí, že práh je jen posunutí s opačným znaménkem:

$$b \equiv -p$$

Aktivační funkce perceptronu je skokovou funkcí. Častěji se používají spojité aktivační funkce, které nemají skokový průběh, ale skok aproximují spojitou funkcí.

Neuron označovaný jako **sigmoid** vuyužívá sigmoidní funkci:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Což pro aktivační funkci sigmoidu dává výsledek:

$$f(\mathbf{x}) = \frac{1}{1 + e^{(-1)(\sum_j w_j x_j + b)}}$$

Budu využívat neuronovu síť se skrytými vrstvami, kdy výstup každého neuronu z předchozí vrstvy je přiveden na vstupy všech neuronů v následující vrstvě. Ve skrytých vrstvách se používají ještě další aktivační funkce. Např. aktivační funkce ReLU a Softmax.

Zajímavou aktivační funkcí je **usměrňovač** - *Rectified Linear Units* (ReLU). Používá se ve skrytých vrstvách. Má výstup 0 v případě záporného vstupu. Jinak se výstup rovná vstupu - matematicky to lze vyjádřit jako:

$$f(x) = max(x, 0)$$

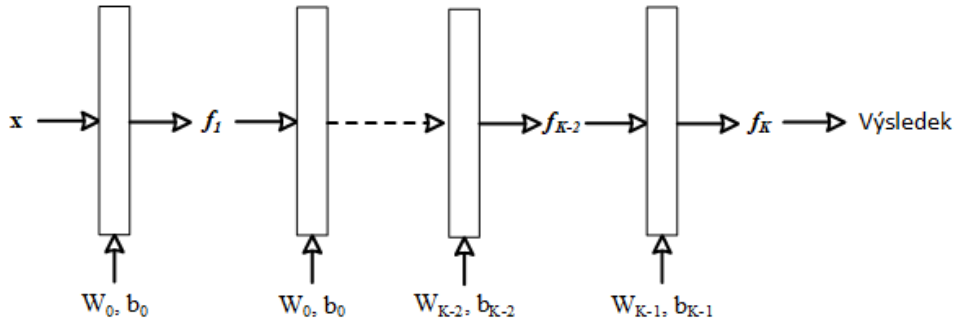
Výstupy perceptronu a sigmoidu byly v rozmezí 0 a 1. Naopak ReLU je v kladném oboru lineární funkcí, tj. může nabývat hodnot 0 až nekonečno.

Aktivační funkce **Softmax**, obdobně jako sigmoid, má obor výstupních hodnot mezi 0 a 1. Avšak v případě Softmax není výstupem pouze jedna hodnota, ale vektor hodnot tak, že se celkový součet těchto hodnot rovná 1. Jedná se o aktivační funkci, která distribuuje pravděpodobnosti vstupů na výstup. Lze to vyjádřit jako:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Kde $\mathbf{z}$ je vektor vstupů a $j = 1, 2, ..., K$ indexuje výstupy.

Propagace



Obrázek 2.3: Vícevrstvá neuronová síť

Na obrázku 2.3 je znázorněna neuronová síť se skrytými vrstvami. Do aktivační funkce f_1 vstupují vektory vstupů $\mathtt{W}_0$ a $\mathtt{b}_0$ (jedná se o vektorovou obdobu váhy w a posunutí b v případě jednoho neuronu).

Neuronová síť provádí složenou funkci:

$$f_K(f_{K-1}(...(f_1(\mathbf{x}))...)) = Vysledek\, vypoctu\, neuronove\, site,$$

kde ve vrstvě i máme aktivační funkci:

$$f_i(x_{i-1}) = \sigma(\mathtt{W}_{i-1}x_{i-1} + \mathtt{b}_{i-1}).$$

Jako aktivační funkci σ je možné využít sigmoid, ReLU apod.

Při výpočtu (dopředné propagace neboli forward propagation) pak platí:

$$f_0 := x$$

$$f_i := \sigma_i(\mathtt{W}_{i-1}x_{i-1} + \mathtt{b}_{i-1}), \qquad i = 1, ...K.$$

Nechť y je výsledek pozorování. Bude nás zajímat, jak se pozorování y liší od výsledku výpočtu neuronové sítě $f_K(f_{K-1}(...(f_1(\mathbf{x}))...))$. Naším cílem bude minimalizovat odchylku výsledku od pozorování.

Zavedeme funkci ztráty (chyby) L, která bude vyjadřovat zjištěnou kvadratickou odchylku pozorování od výpočtu neuronové sítě:

$$L(\Theta) = \| \, y - f_k(\Theta, \mathbf{x}) \, \|^2$$

Cílem bude hledat minimum této funkce. Nikoliv však vůči proměnným x (vektor) ale vůči parametrům $\Theta = \{\mathtt{W}_0, \mathtt{b}_0, ..., \mathtt{W}_{K-1}, \mathtt{b}_{K-1}\}$.

2.2.3 Zpětná propagace (Backward propagation)

Funkce $L(\Theta)$ je mnohoproměnná funkce. Na nalezení jejího minima použiji gradientní metodu, tj. minimum budeme hledat v "nejprudším"směru. Směr budeme určovat parciální derivací. Tu můžeme uskutečnit, protože používáme hladké aktivační funkce (s výjimkou ReLU, která v bodě nula nemá derivaci). Gradientní metoda mě pochopitelně může zavést nikoliv do globálního minima, ale do lokálního minima. Tento problém se řeší různými vylepšeními gradientní metody (např. odskočením z minima a otestováním, zda-li nelze dojít k "lepšímu"minimu). V našem případě avšak nemáme obecnou mnoharozměrnou funkci, ale funkci vycházející z popisu jednotlivých vrstev. Proto uspokojivých výsledků dosahujeme algoritmem zpětné propagace.

Zpětná propagace znamená, že jdeme od poslední vrstvy směrem k první vrstvě a v každém kroku se snažíme optimalizovat $\Theta_j = \{\mathtt{W}_j, \mathtt{b}_j\}$.

Zamyslíme-li se nad parciální derivací

$$\frac{\partial L}{\partial \Theta_{K-1}}$$

pak si musíme uvědomit, že funkce L je složenou funkcí. Vnější funkce je $\| \ ... \ \|^2$ a vnitřní funkce je vnitřek této "závorky". Nyní si vzpomeneme na derivování složených funkcí - derivace složené funkce je derivace vnější funkce krát derivace vnitřní funkce. Přitom derivujeme podle Θ, tj. y ve vnitřní funkci je konstanta, tj. po derivaci nula. Takže v prvním zpětném kroku dostaneme:

$$\frac{\partial L}{\partial \Theta_{K-1}} = \frac{\partial L}{\partial f_K} . \frac{\partial f_K}{\partial \Theta_{K-1}}$$

A můžeme postupovat dále:

$$\frac{\partial L}{\partial \Theta_{K-2}} = \frac{\partial L}{\partial f_K} . \frac{\partial f_K}{\partial f_{K-1}} . \frac{\partial f_{K-1}}{\partial \Theta_{K-2}}$$

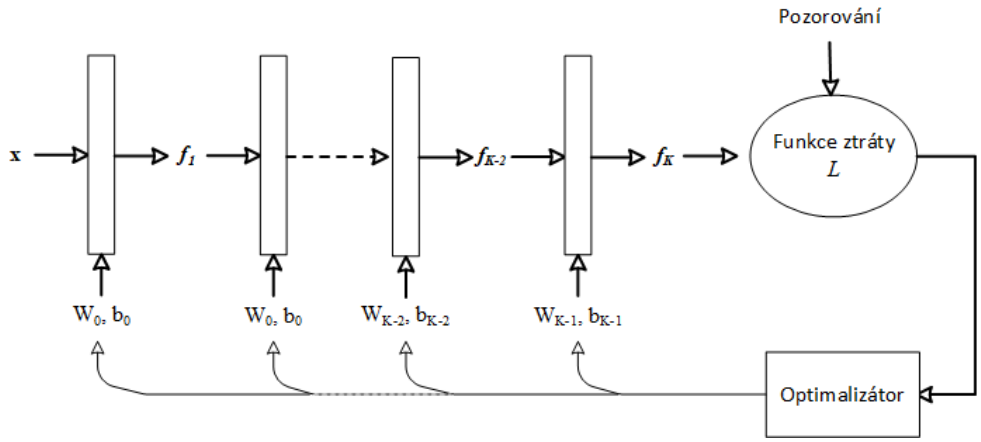
Obecně:

$$\frac{\partial L}{\partial \Theta_i} = \frac{\partial L}{\partial f_K} . \frac{\partial f_K}{\partial f_{K-1}} ..... \frac{\partial f_{i+2}}{\partial f_{i+1}} . \frac{\partial f_{i+1}}{\partial \Theta_i}$$

Prakticky vycházíme z nějaké náhodné hodnoty vektoru x a postupně, porovnáváním s jednotlivými pozorováními, optimalizujeme Θ, tj. učíme neuronovou síť.

Jako funkci ztráty L jsem použil *Categorical Cross-Entropy Loss*. Tato funkce definuje vzdálenost mezi vektory `a` a `b` jako:

$$L(\mathtt{a}, \mathtt{b}) = -\sum_{j=0}^{M} \sum_{i=0}^{M} (a_{ij} \, . \, log(b_{ij})).$$



Obrázek 2.4: Optimálizator

Optimalizátor (obrázek 2.4) je algoritmus procesu učení. To znamená, že na základě porovnání rozdílu mezi pozorováním a výsledkem neuronové sítě upravuje od poslední vrstvy směrem k počáteční parametry $\{\mathtt{W}_j, \mathtt{b}_j\}$. V neuronové síti pro tuto práci jsem použil optimalizátor Adam – *ADAptive Moment estimation.* [3]

2.3 Analýza možností řešení

Pro realizaci aplikace je nutné zvolit typ databáze a její distribuci. Dále je nutné realizovat serverovou část. U serverové části je zapotřebí analyzovat možná řešení a zvolit to nejvhodnější. V rámci realizace aplikace jako takové není mnoho možností volby.

2.3.1 Databáze

První bych se zaměřil na řešení databázové části. Koncept aplikace je takový, že většinu aplikace budou tvořit číselníky. Důvod pro toto řešení je minimalizovat duplicitu a objem dat v databázi. Například předměty se budou u mnoha seznamů opakovat.

Na začátku je důležité se rozhodnout mezi relačními a nerelačními databázemi. Většina dotazů do databáze bude jednoduchých. I když se díky číselníkům nebude jednat o velké množství záznamů, bude potřeba velice rychlé dotazování. Rychlost dotazování je velice důležitá, jelikož v dnešní době rozhoduje, zda aplikace bude úspěšná nebo nikoliv. Žádný uživatel nechce na výsledky čekat dlouho.

Strukturovaná data se skládají z jasně definovaných datových typů a díky jejich schématu se v nich snadno vyhledává. [4].

V aplikaci je jasně dáno, jaká data budou u jednotlivých seznamů zadávána. Typ těchto dat bude předem definovaný. Budou tedy vznikat strukturovaná data. Mohu říci, že předešlé požadavky, které aplikace vyžaduje k dosažení optimálního fungování, splňují relační databáze. [5] Pro rychlé načítání z databáze je nutné použít indexy.

Důvod pro vytvoření indexů na dané tabulce v databázi je zrychlení prohledávání tabulky a nalezení záznamů, které hledáme. Cena za zrychlení vyhledávání pomocí indexů je zpomalení vkládaní záznamů. Vkládání nových záznamů do tabulky bez indexu je jednoduché. Databáze nalezne další prázdný prostor a na něj záznam vloží. Pokud je záznam vkládán do indexované tabulky, databáze vloží nový záznam do tabulky a následně vloží nový záznam do každého indexu v tabulce. Vložení indexu musí být na správné místo [6].

Hlavní funkcí aplikace je vyhledávání seznamů. Vkládání a mazání záznamů, a s tím spojená práce s indexy, není tak důležité. Důvodem nedůležitosti vkládání je, že uživatel bude spíše vyhledávat seznamy v aplikaci, než je ukládat.

Rozhodnutí o distribuci databáze si ponechám až po analýze serverové části. Pro různá řešení serverové části mohou být různé distribuce přívětivější.

2.3.2 Backend

2.3.2.1 API

SOAP využívá formátu XML, zatím co REST může využívat menší formáty, je nezávislý na platformě i jazyku a umí používat pouze HTTP. U SOAP jsou požadavek i odpověď zabalené do obálky, na druhou stranu REST posílá čistá data. SOAP se více hodí do enterprise systémů jako finance nebo telekomunikace. REST je nejužitečnější, pokud se jedná o aplikace, které pouze vybírají data z databáze nebo data v databázi ukládají. V tomto přístupu je rychlejší než SOAP, kde jsou požadavky zabaleny. [7]

Aplikace bude sloužit pouze pro zobrazování dat z databáze, pro jejich ukládání nebo editaci. Jelikož se jedná o přenos dat v mobilním zařízení, je lepší použít REST s menším objemem dat, co se týče požadavků a odpovědí. Zároveň je jednodušší parsování JSON struktury než XML. Aplikace bude komunikovat pomocí HTML, takže s tímto omezením není problém.

2.3.2.2 Framework

V dnešní době jsou nejpoužívanější Node.js a Ruby on Rails.

Ruby od Rails je framework využívající jazyk Ruby. Ruby jako takové má velkou podporu, co se týče testování. Testování je možné pomocí knihoven Minitest nebo pokročilejší knihovny Rspec. Nejen, že se díky těmto modulům jednodušeji píší testy, ale je zde možné jednoduše použít test driven development. V Ruby on Rails je dodržován princip jednoduchosti. Každý kód je lehce čitelný, udržovatelný i testovatelný. Jelikož Ruby on Rails obsahuje mnoho modulů, je možné je znovu využívat, a tak urychlit vývoj. Zároveň komunita okolo Ruby on Rails je velká, díky čemuž se stále objevují nové moduly. [8]

Node.js je dnes na vzestupu. Hlavní podíl na tom má jeho rychlost. Mnoho velkých společností v nedávné době přešlo z Ruby on Rails na Node.js. Node.js má stejně jako Ruby on Rails mnoho balíků, které řeší nejrůznější problémy a vývojář je může jednoduše použít. Node.js je možné použít jak na straně klienta, tak i serveru. [8]

I přes to, že Node.js je dnes preferovanější, rozhodl jsem se použít Ruby on Rails. Jeden z důvodů je, že bych si ho rád vyzkoušel a blíže pochopil. Dále je velice rychlý, co se týká psaní kódu. Neposledním důvodem je jednoduché testování pomocí test driven development.

2.3.3 Neuronová síť

Pro neuronovou síť jsem zvolil knihovnu Tensorflow s module Keras v jazyce Python.

Tensorflow je knihovna pro tvorbu neuronových sítí. Obsahuje mnoho dalších knihoven, které usnadňují práci.

Keras je jedna z knihoven obsažených v Tensorflow. Hlavní přednosti Kerasu jsou tyto:

- Uživatelská přívětivost – má jednoduchý a konzistentní interface pro běžný use case. Přehledně a srozumitelně zobrazuje chybové zprávy.
- Modely v Keras jsou tvořeny z bloků, které se postupně skládají dohromady, a tak tvoří model.
- Jednoduché rozšíření díky použití bloků

[9]

Dalšími užitečnými knihovnami v Tensorflow jsou Hyperas a Hyperopt. Tyto dvě knihovny usnadňují volbu jednotlivých parametrů modelu u neuronové sítě. Uživatel může najednou vyzkoušet více možností volby jednotlivých parametrů. Model je pak automaticky otestován na všechny kombinace takto zvolených parametrů a nakonec zobrazí, při jaké kombinaci parametrů dosahuje nejlepších výsledků.

```
if {{choice(['four', 'five'])}} == 'five':
    model_mlp.add(Dense({{choice([32, 64, 126, 256, 512])}}))
    model_mlp.add(Activation({{choice(['relu', 'sigmoid'])}}))
```

Obrázek 2.5: Hyperas

Zde je příklad použití těchto knihoven. V tomto kroku je testováno, zda má být přidána pátá vrstva. Pro pátou vrstvu knihovna testuje jaký počet neuronových uzlů má být zvolen (32, 64, 126, ...) a jakou aktivační funkci zvolit (ReLU nebo Sigmoid).

2.3.3.1 Aplikace

Pro tuto práci jsem se rozhodoval mezi jazykem Java a Kotlin. Hlavním důvodem byla podpora těchto jazyků v IDE AndroidStudio. AndroidStudio, jak už název napovídá, je vytvořeno hlavně pro vývoj aplikací pro OS Android. Proto zde najdeme nástroje, které s vývojem pro OS Android přímo souvisejí. Jak už emulátor telefonu, který umožní testování aplikace přímo v IDE, tak i možnost vytváření obrazovek pomocí designu. V designu obrazovek je možné vytvářet jednotlivé objekty na obrazovce pomocí přetahování těchto objektů z menu. Emulátor jako takový je vcelku náročný pro systém počítače a bere velké množství paměti. Je tedy stále lepší mít u sebe telefon, který se s IDE propojí, čímž se testování zrychlí.

V dnešní době IDE AndroidStudio již plně podporuje jazyk Kotlin. Není tomu ovšem dlouho, co Kotlin nebyl úplně podporován a vývojáři se mohli setkat s problémy. S tím také souvisí menší počet řešení složitých otázek na internetových fórech nebo blocích. Toto ovšem pro mé využití netvoří zas tak velký problém. Jelikož pokud vyvíjím v IDE AndroidStudio a vyberu, že budu používat Kotlin, tak pokud do tohoto souboru zkopíruji Java kód, tak samo IDE mi nabídne, zda nechci kód převést do jazyku Kotlin. Zároveň, jelikož Kotlin je postaven na jazyku Java, dokáže zpracovat i kód psaný v jazyku Java.

2.3.3.2 Zabezpečení

Je důležité kontrolovat přístup uživatelů na server skrze ověření, zda je uživatel přihlášen a jím zaslaný požadavek na server má být zpracován. Dále je nutné mít zavedenou ochranu proti útokům na databázi, a to zejména útoky typu SQL Injection.

2.3.3.3 Autentikace

Pro ověření, zda je uživatel přihlášen a jím zaslané požadavky se mají zpracovat, jsem použil JSONWebToken.

Název JSONWebToken vznikl díky tomu, že pro předávání tokenu se používá struktura JSON. JSONWebToken je standardem, který byl již implementován do většiny programovacích jazyků. Díky tomu může být použit napříč platformami. Zároveň je reprezentován jako obyčejný text, kvůli tomu může být předán v rámci URL nebo HTTP hlavičky.[10]

JsonWebToken je text, který se dá poslat v hlavičce HTML. Samotný token se pak skládá ze tří částí: hlavičky, obsahu a podpisu. Hlavička obsahuje typ tokenu (v mé aplikaci je to JWT) a použitý hashovací algoritmus. Obsah může mít následující tři typy: soukromý, veřejný a registrovaný. Poslední částí je podpis. Podpis je hash z hlavičky, obsahu a tajemství. Jelikož pouze server zná dané tajemství, není možné, aby s obsahem někdo manipuloval. [10]

Tuto možnost jsem zvolil kvůli jednoduchosti použití. Zároveň je velice snadné ji implementovat v Ruby on Rails. Ovšem mnoho ostatních jazyků tuto metodu autentizace též podporuje a ani v nich není složité ji implementovat. Jako hashovací algoritmus jsem použil Bcrypt.

Autorem Bcryptu jsou Neil Provos a David Mazières. Jméno Bcrypt vzniklo ze šifrovací funkce Blowfish, proto b, a hashovací funkce pro systém ukládání uživatelských hesel v systému UNIX, který se jmenuje crypt.[11]

JSONWebToken pro aplikaci v této práci, kde obsah je id uživatele, pak vypadá například takto:

Hlavička eyJhbGciOiJIUzI1NiJ9

Obsah eyJ1c2VyX2lkIjoxLCJleHAiOjE1NTQ3MzMwMDB9

Podpis VqJRuX7x6b9ExKkGNyLcSdD6zQaJ_cy-Iz6tz2p3rRk

2.3.3.4 SQL Injection

SQL injection je technika vkládání kódu, která může poškodit databázi. Je jednou z nejčastějších technik napadání databázové vrstvy skrze vrstvu aplikační. Jméno injection (česky vsunutí pozn. aut.) si tato technika vysloužila proto, jelikož pracuje se vsunutím části škodlivého kódu do požadavku skrze textové

pole umístěné na webové stránce. [12]

Tato práce nepojednává o webové stránce, ale o aplikaci. Ovšem, ať už řešíme webovou stránku, aplikaci nebo jakýkoliv frontend, který komunikuje se serverem potažmo s databází na základě uživatelského vstupu, je nutné problém SQL injection nepřehlížet, ale zamyslet se, zda nemůže nastat či jak mu předejít.

Příkladem může být:

”Select * FROM Users WHERE UserId =”

jako id na server přijde ”105;DROP TABLE Users”

Příkaz je validní a provede se. Ovšem namísto vyhledání uživatele se celá tabulka vymaže, a to ať už uživatel s id 105 existuje nebo ne.

V rámci aplikace, kterou tato práce popisuje, problém SQL injection může nastat pouze při přihlašování nebo registraci uživatele, jelikož jsou to jediná místo, kde uživatel zadává text, který je následně kontrolován s databází a do databáze propisován.

Možné řešení je používat SQL parametry. Pokud například zjišťuji, zda uživatel s daným id v databázi existuje, nepoužiji:

*„SELECT * FROM Users WHERE UserId = 1“*

ale

*„SELECT * FROM Users WHERE UserId = @0“*

následně při provádění dotazu k němu přidám hodnoty, které jsem obdržel. SQL pak kontroluje každý parametr a zajišťuje, jestli je správný pro daný sloupec. Zároveň každý parametr používá jako celek a ne jakou část dotazu, která by se mohla provést.[12]

Jelikož v této práci používám Ruby on Rails, tak tento problém řešit nemusím. To z důvodu, že Ruby on Rails samo vytváří dotazy a každý dotaz tvoří s parametry. Zde je příklad, jak vypadá dotaz, který Ruby on Rails vygeneruje při hledání uživatele v databázi:

”SELECT ”users”.* FROM ”users”WHERE ”users”.”email-$1 LIMIT $2 [[”email”, ”user@gmail.com”], [”LIMIT”, 1]]”

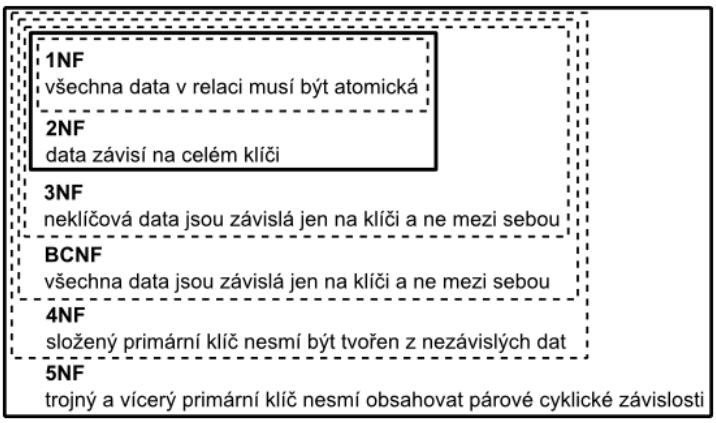
Kapitola 3

Realizace

3.1 Databáze

Aby byla databáze dobře udržovatelná a jednoduše se s ní pracovalo, je nutné dodržet základní principy návrhu databáze s dodržením normálních forem. Těch je celkem šest.

„*Normální formy jsou pak pravidla, která by data v relaci měla splňovat. Čím vyšší normální forma, tím lepší a jednodušší by práce s daty, jejich vybíráním a aktualizacemi měla být. Formy jdou postupně od nižších k vyšším, kdy každá vyšší v sobě zahrnuje formy nižší.*“ [13]



Obrázek 3.1: Normální formy

Z tohoto zdroje a výše připojeného obrázku vyplývá, že správný návrh by měl dodržovat všechny normální formy.

Ovšem v praxi se často používá denormalizace. Denormalizece spočívá

v tom, že v rámci zrychlení dotazů se zanedbají některé normativní formy. Rozhodnutí ohledně denormalizace jsou složitá, jelikož je nutné zamyslet se nad dopadem zrychlení výběru dat a nad jejich vkládáním. Zároveň je možné, že dojde k redundanci dat.

Avšak v příkladu, který jsem použil v návrhu samotné databáze, se může stát, že data nejsou závislá na primárním klíči a zároveň se opakují. Ovšem pokud bychom tato data dali do samostatné tabulky, museli bychom zároveň použít cizí klíč pro navázání dat na danou tabulku.

Pokud chci ukládat překlady k jednotlivým záznamům, musím u nich uvádět, kterého jazyka se týkají. Jazyk dokáži definovat pomocí tří znaků. Všechny záznamy v jednom jazyce budou mít stejné tři znaky. Nyní je otázka, zda nevytvořit novou tabulku nebo číselník, který bude obsahovat jednotlivé zkratky jazyků a tím omezit redundanci dat. Tímto způsobem bych dodržel normální formy pro databázi.

Pokud se na problém podívám z hlediska rychlosti výběrů, vkládání a velikosti uložených dat, tak mohu diskutovat následující zlepšení. Pro uložení cizího klíče, který bude u každého záznamu, budu používat integer, který má velikost 32 bitů. Pokud záznam o jazyce vložím ke každému záznamu, jsou to tři písmena, tedy varchar velikosti 3, a to je 24 bitů. Tímto způsobem nejen ušetřím místo, ale zároveň při výběrech ušetřím čas, jelikož nemusím spojovat tabulky.

3.1.1 Tabulky

Jednotlivé tabulky databáze budou následující.

- Uživatel - Users
- Seznamy - PackingLists
- Předměty – Items
- Kategorie předmětů - Categories
- Aktivity - Criteria
- Destinace - Destinations
- Věk - Ages
- Typ dopravy - Transport_types
- Počasí - Weathers
- Překlad aktivit - Translation_criteria
- Překlad předmětů – Translation_items

- Překlad kategorií – Translation_categories
- Překlad destinací – Translation_destinations
- Překlad počasí – Translation_weathers
- Překlad typu dopravy – Translation_transport_types

Tabulka Uživatel Tabulka Uživatel slouží k uložení informací o daném uživateli.

- Primární klíč – id – bigint
- Heslo – password – varchar
- e-mailová adresa – email – varchar

Tabulka Seznamy Do tabulky Seznamy jsou uloženy informace o specifickém seznamu.

- Primární klíč – id – bigint
- Pohlaví – sex – varchar
- Počet vložených seznamů - total_lists_added – int

Tabulka Destinace V tabulce Destinace jsou k nalezení informace o konkrétní destinaci.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar

Tabulka Předměty Tabulka Předměty slouží k uložení informací ohledně jednotlivých předmětů.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar
- Pohlaví - gender - varchar

Tabulka Kategorie V této tabulce lze najít informace k dané kategorii.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar

Tabulka Aktivity Do tabulky Aktivity se ukládají jednotlivé aktivity.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar

Tabulka Počasí V této tabulce lze nalézt jednotlivá počasí.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar

Tabulka Typ dopravy Zde jsou k dohledání jednotlivé typy dopravy.

- Primární klíč – id – bigint
- Název obrázku – picture_name – varchar

Tabulka Věk V této tabulce jsou uloženy jednotlivé věkové kategorie.

- Primární klíč – id – bigint
- Věkové intervaly – age_range- interval
- Překlad – name - varchar

Tabulky Překlady Obsah těchto tabulek tvoří překlady jednotlivých číselníků.

- Primární klíč – id – bigint
- Jazyk překladu – language – varchar
- Překlad – name – varchar

3.1.2 Relace

Propojení jednotlivých tabulek bude následující:

- Seznam Aktivity

  Seznam je spojen s aktivitami. Každý seznam může mít více aktivit, ale zároveň každá aktivita může být obsažena ve více seznamech. Z těchto důvodů bude mezi těmito tabulkami relace n:m. Toto spojení znamená vytvoření relační tabulky list_criteria. Tato tabulka bude obsahovat dva atributy. První bude cizí klíč daného seznamu. Druhý atribut bude cizí klíč aktivity.

- Seznam Destinace

  Seznam je dále spojen s tabulkou Destinace. Každý seznam má právě jednu destinaci. To zařídí relace 1:n. Pro takovéto spojení, bude nutné, aby tabulka Seznam obsahovala cizí klíč tabulky Destinace.

- Seznam Věk

  Každý seznam je přiřazen k určitému věkovému rozmezí. Ovšem věkové rozmezí má více seznamů. Jedná se tedy o relaci 1:n. Do tabulky Seznamy musí být přidán cizí klíč tabulky Věk.

- Seznam Počasí

  Seznam je vždy svázán s tabulkou Počasí, stejně jako u relace Seznam Aktivit se jedná o relaci n:m. Řešení této relace je nově vzniklá tabulka list_weathers s cizími klíči z tabulky Seznam a Počasí.

- Seznam Předměty

  Každý seznam musí obsahovat jednotlivé předměty, které si uživatel bude brát s sebou. Je zřejmé, že v jednotlivých seznamech bude více předmětů. Zároveň u jednotlivých seznamů se budou předměty opakovat. Situace je tedy stejná jako u relace tabulky Seznam a Aktivity. Z čehož plyne relace n:m. Vznikne nová tabulka list_item obsahující cizí klíče, jak z tabulky Seznam, tak z tabulky Předměty. Následně bude tato tabulka obsahovat sloupec Relativní četnost - support. Tento atribut bude znázorňovat, kolikrát byl tento předmět vložen do seznamu. Je to z důvodu, který byl zmíněn v matematickém modelu.

- Překlady

  Tabulky Kategorie, Kritéria, Destinace, Předměty, Druh transportu a Počasí musí být navázány na svoje překlady. Navázání bude realizovat relace 1:n. Bude tedy opět přidán cizí klíč z uvedené tabulky do Překladu.

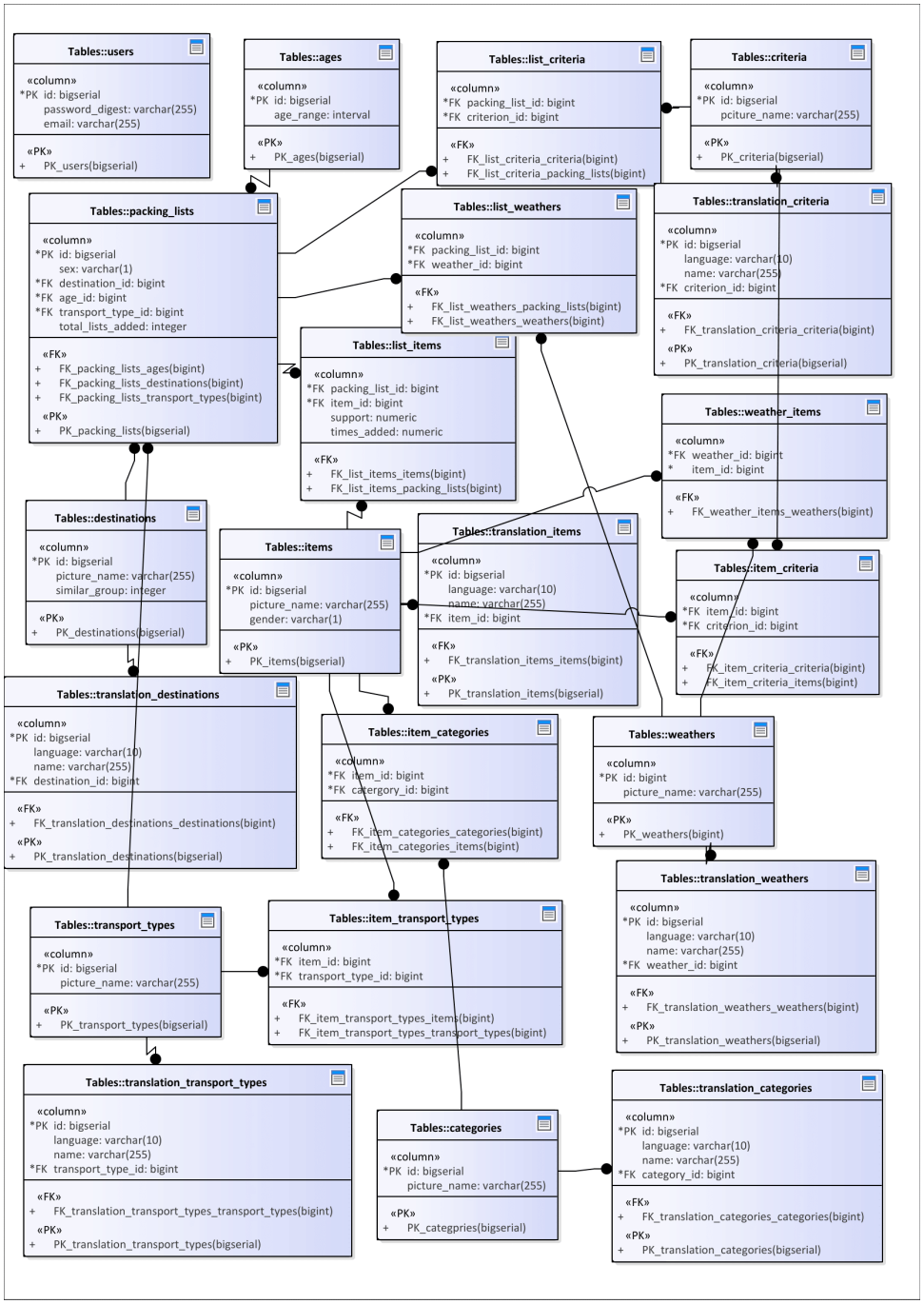
- Předmět - počasí

  Pro každý předmět je důležité, aby se nezobrazoval, pokud pro dané počasí není využitelný. Řešení je vytvořit relaci mezi předmětem a počasím. Tato relace bude m:n. Jeden předmět může být využitý pro více druhů počasí. Realizace spočívá opět ve vytvoření tabulky weather_items s cizími klíči z obou tabulek.

  Pro jednodušší zobrazení, je každý předmět je spojen s aktivitami. Každý předmět může být zobrazen ve více aktivitách a každá aktivita má více předmětů. Jedná se opět o relaci m:n. Realizaci této relace zobrazuje tabulka item_criteria. Tato tabulka obsahuje cizí klíče z obou tabulek.

Poslední z relací se nachází mezi tabulkami Předmět a Kategorie. Opět se jedná o relaci m:n. Uskutečnění této relace probíhá za pomoci tabulky item_category.

3.1.3 Schéma



Obrázek 3.2: Schéma databáze

3.2 Backend

Backend je tvořen REST serverem v Ruby on Rails a Python REST serverem pro doporučování v rámci neuronové sítě.

3.2.1 Ruby on Rails

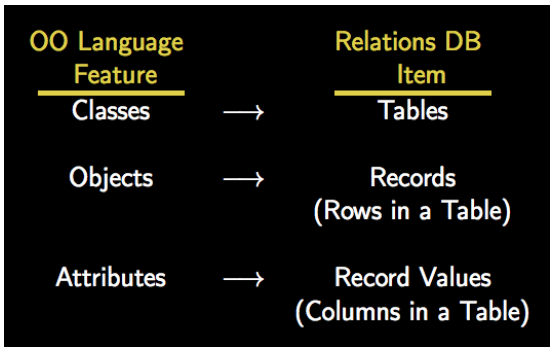
Ruby on Rails, jak už bylo zmiňováno, používá návrhový vzor MVC. Pro realizaci REST serveru nebylo zapotřebí využít prostřední část, tedy zobrazení, jelikož o to se stará frontend, tedy aplikace v Android.

3.2.1.1 Model

Pro popis použiji model PackingList. Model se skládá ze dvou částí.

První částí je definice tabulky. Definice je tvořená vždy třídou pojmenovanou Create a názvem tabulky. Pro PackingList se jedná o třídu CreatePackingLists. Každá takováto třída dědí vlastnosti od třídy ActiveRecords.

ActiveRecords je návrhový vzor využívaný v Ruby i Ruby on Rails. Jedná se o persistenci dat. Ve zkratce to znamená, že v době běhu programu či aplikace se používají třídy, objekty a jejich atributy. Při ukončení běhu jsou všechny tyto záznamy smazány. Toto ale neplatí pro ActiveRecords. Všechny položky jsou za běhu propisovány do databáze, kde jsou uloženy a při následujícím běhu aplikace či programu jsou znovu využity. Zde je obrázek, který ukazuje, jak jsou mapovány jednotlivé části do databáze.



Obrázek 3.3: Funkcionalita ActiveRecords

[14]

Každá třída Create obsahuje metodu Change. Metoda Change obsahuje aktuální definici tabulky, která se má při migraci databáze vytvořit nebo upravit. Třída CreatePackingList pak vypadá následovně:

```ruby
class CreatePackingLists < ActiveRecord::Migration[5.1]
  def change
    create_table :packing_lists do |t|
      t.string :sex, limit: 1
      t.references :destination, foreign_key: true
      t.references :user, foreign_key: true
      t.references :age, foreign_key: true
      t.references :transport_type, foreign_key: true
      t.integer :total_lists_added
      t.timestamps
    end
  end
end
```

Obrázek 3.4: Třída CreatePackingList

Druhou částí je pak samotná třída pro danou tabulku — v tomto případě PackingList. Tato třída obsahuje dvě části. První částí jsou relace tabulky. Druhou částí je validace atributů tabulky. Každá třída dědí od třídy ApplicationRecord, která dědí od již zmíněné třídy ActiveRecords. Díky tomu je možné používat funkce jako belongs_to, has_many a další.

Jak je vidět při porovnání obrázků Třída CreatePackingList a Třída PackingList, je zřejmé, že třída CreatePackingList pouze definuje tabulku a její jednotlivé sloupce, zatím co třída PackingList definuje, kam se odkazují jednotlivé cizí klíče, a zároveň validuje, zda sloupce, které nejsou cizími klíči, existují. Existence sloupců a relací se vždy ověřuje na úrovni třídy.

```ruby
class PackingList < ApplicationRecord
  # model association
  belongs_to :destination
  belongs_to :age
  belongs_to :transport_type

  has_many :list_item, dependent: :destroy
  has_many :list_weather, dependent: :destroy
  has_many :list_criterion, dependent: :destroy

  # validation
  validates_presence_of :sex
  validates_presence_of :total_lists_added
end
```

Obrázek 3.5: Třída PackingList

Pro každou tabulku popsanou v sekci databáze jsou tímto způsobem vytvořeny třídy.

3.2.1.2 Kontrolery

Kontrolery jsou třídy starající se o příchozí požadavky a komunikaci s databází. Server se skládá z 10 kontrolerů. Tyto kontrolery se dělí na dvě skupiny. První skupinou jsou autorizační kontrolery. Jedná se o:

- Aplikační kontroler - kontroluje, zda je připojení JSON Web Token k požadavku validní
- Autorizační kontroler - po přihlášení vytvoří pro uživatele JSON Web Token

Druhou skupinou jsou kontrolery přímo reagující na dotaz a vracející příslušná data. Jsou to:

- Kategorie
- Kritéria
- Destinace
- Předměty
- Seznamy
- Typ transportu
- Počasí
- Uživatel

Každý kontroler kromě Uživatele a Seznamu obsahuje metodu Index. Tato metoda vrací všechny záznamy ze stejnojmenné tabulky. Pro zjednodušení je v rámci výběru dat z databáze navázaná i tabulka překladů dané tabulky. Na základě zadaného kódu jazyka rovnou vrátí jednotlivé záznamy v daném jazyce zpět. Dále obsahují metody Create a Destroy. Následuje popis metod, které jsou v některých kontrolerech navíc z důvodu zjednodušení dotazování.

Kontroler Kategorie — obsahuje navíc metodu index_on_criteria. Tato metoda vybírá pouze kategorie obsahující předměty podle předešlého zadání parametrů Počasí a Kritéria. Tato metoda snižuje počet kategorií zobrazených uživateli.

Kontroler Kritéria — obsahuje metodu index_on_category, jež má podobný význam jako index_on_criteria. Pouze se namísto parametru Kritéria používá Kategorie.

Kontroler Předměty — obsahuje navíc dvě metody get_item_criteria a get_items_transport. Metody jsou skoro totožné, ale kvůli rozdělení typů transportu a kritérií do dvou tabulek v databázi jsou zapotřebí obě. Rozdíl je, že jedna používá zadaný parametr Kritéria pro vyhledání a druhá parametr Typ transportu spolu se všemi ostatními zadanými parametry.

Kontroler Seznamy — zde je metoda Create rozdělena na dva případy. Pokud seznam se zadanými parametry neexistuje, vytvoří ho a pomocí metod add_criteria, add_weathers a add_items přidá do relačních tabulek záznamy o zvolených kritériích, počasí a předmětech.

Pokud seznam se zadanými parametry již existuje, je nutné přepočítat relativní četnost jednotlivých předmětů. Za pomoci metody Recalculate rozdělí vybrané předměty do tří skupin: předměty, které v seznamu vybrané nejsou, předměty existující v seznamu a zároveň vybrány uživatelem a předměty nevybrané uživatelem, ale existující v seznamu.

Metoda Recalculate přidá nové předměty do relační tabulky s relativní četností rovnající se:

1 / počtem již zadaných seznamů se stejnými parametry

Předměty, které již existující a zároveň byly zadané uživatelem, se přepočítají inkrementem hodnoty počtu zadaných seznamů a počtu přidání daného předmětu o 1. Následně se tyto dvě hodnoty podělí a vznikne nová relativní četnost.

Předměty existující, ale nebyly zadány uživatelem, se počítají stejně jako v předchozí situaci, až na to, že počet přidání daného předmětu se neinkrementuje.

Dohledávání seznamů a doporučování je řešeno pomocí metody Lists. Metoda nejprve určí jaký algoritmus byl zvolen. Pokud byla zvolena neuronová síť, je požadavek přesměrován na server starající se o neuronovou síť. V opačném případě metoda zjistí pomocí metody get_packinglist, zda existuje seznam se zadanými parametry. Pokud takovýto seznam existuje vrátí všechny předměty s relativní četností větší než 50%.

Pokud daný seznam neexistuje, postupuje metoda následovně. První rozšíří zadané parametry za pomoci metody set_wither_params. Tato metoda rozšíří zadaný věkový interval o okolní intervaly. Dále uloží číslo podobnosti kategorie do proměnné. Následně pro všechna zadaná kritéria kombinovaná se všemi zvolenými typy počasí zjistí, zda existují seznamy s danou kombinací počasí a kritéria. Pokud ano, přidá do doporučených předmětů všechny ty, které jsou

přiřazené k danému počasí a kritériu.

Pokud ani takovýto seznam pro danou kombinaci neexistuje, zkouší metoda vyhledat seznamy se stejnou kombinací kritérií, počasí a destinací se stejným číslem podobnosti. Při úspěchu opět přidá předměty do doporučených předmětů.

V případě, že ani zde není vyhledávání úspěšné, rozšíří metoda vyhledávání o dříve dohledané okolní věkové kategorie.

Na konci metoda projde všechny doporučené předměty a odstraní z nich ty, které mají relativní četnost menší než 50%. Tento seznam odešle zpět do aplikace.

Kontroler Typ transportu — obsahuje navíc metodu get_transport_if_items. Tato metoda je podobná jako u kategorie či kritéria.

3.3 Frontend

Frontend se skládá z 11 aktivit a 1 třídy pro ukládání seznamu. Každá aktivita odpovídá jedné obrazovce v aplikaci. Každá aktivita je reprezentovaná třídou a XML souborem, který popisuje jednotlivé objekty na obrazovce. Jednotlivé aktivity jsou:

- Přihlášení
- Zobrazení rozcestníku
- Zobrazení kritérií
- Zobrazení destinací
- Zobrazení typů transportů
- Zobrazení doplňujících informací
- Zobrazení seznamu
- Zobrazení rozcestníku kriterií předmětů v seznamu
- Zobrazení předmětů
- Menu vlastních seznamů
- Zobrazení vlastních seznamů

Každá aktivita obsahuje inicializační metodu onCreate. Tato metoda vždy načte jednotlivé parametry předané této aktivitě. Zároveň dokáže načíst uložený stav aplikace. Stav aplikace se ukládá pomocí metody onSaveInstanceState. Tato metoda při změně aplikace (např. rotaci obrazovky nebo minimalizaci aplikace) uloží aktuální stav. Stavem se rozumí jednotlivé parametry.

Postupným procházením aplikace uživatel vyplňuje parametry. Každá aktivita vždy posílá všechny obdržené a zadané parametry následující aktivitě.

Přihlášení

- Přihlášení — se zadaným emailem a heslem zavolá zaslání post requestu pro přihlášení
- Registrace — se zadaným emailem a heslem zavolá zaslání post requestu pro registraci
- Zaslání post requestu — podle parametrů zašle požadavek na server

Zobrazení rozcestníku

- Vyhledat — přesměruje na aktivitu vyhledání
- Zobrazit své seznamy — přesměruje na aktivitu zobrazení vlastních seznamů

Zobrazení destinací, kritérií, typu transportu a doplňující informace

- Získání záznamů ze serveru
- Vytvoření radiobuttonů/checkboxů s jednotlivými záznamy
- Získání vybraných voleb uživatele spuštení následující aktivity

Zobrazení seznamu Do této aktivity vstupuje kromě ostatních parametrů i parametr, který určuje, zda se jedná o vytvoření či editaci seznamu nebo o zobrazení již vytvořeného seznamu. V této aktivitě parametr pouze určuje, zda je možné přejmenovat seznam.

- Získání seznamu podle zadaných kritérií
- Vytvoření předmětů — po získání předmětů ze serveru nebo z uloženého souboru na zařízení vytvoří mapu map. Hlavním klíčem je kategorie, která odkazuje na jednotlivá kritéria, která odkazují na pole vybraných předmětů
- Získání kategorií pro editaci — získá ze serveru všechny kategorie

- Získání kategorií pro zobrazení seznamu — dotáže se na server na kategorie, ve kterých byly vybrány nějaké předměty
- Vytvoření kategorií — vytvoří jednotlivá tlačítka pro kategorie. Po stisknutí přesměruje na zobrazení kritérií
- Uložení seznamu — uloží vytvořený seznam na zařízení za pomoci třídy pro uložení. Následně zašle seznam na server pro přepočítání
- Vrácení z aktivity zobrazení kritérií pro seznam — uloží do mapy předmětů pod danou kategorii všechna vrácená kritéria s jejich přidanými předměty

Zobrazení předmětů Podle toho, zda se jedná o zobrazení stávajícího seznamu nebo tvorbu nového, zobrazí všechny předměty nebo jen přidané.

- Zobrazení předmětů — zobrazí jen přidané předměty
- Vytvoření předmětů — zobrazí všechny předměty s předvolenými, doporučenými a již přidanými předměty
- Potvrzení — zašle do předešlé aktivity přidané předměty

Zobrazení rozcestníku kriterií předmětů v seznamu

- Získání záznamů ze serveru
- Vytvoření radiobuttonu/checkboxů s jednotlivými záznamy
- Vrácení z aktivity zobrazení předmětů pro seznam — uloží do mapy předmětů pod dané kritérium všechny přidané předměty

Zobrazení vlastních seznamů

- Načtení seznamů — ze zařízení načte všechny uložené seznamy
- Vytvoření seznamů — zobrazí všechny načtené seznamy

3.4 Neuronová síť

Neuronová síť se skládá z jednotlivých vrstev. Vrstvy jsou rozděleny do tří skupin.

- Vstupní vrstva
- Skryté vrstvy
- Výstupní vrstva

Samotný výpočet probíhá ve skrytých vrstvách. Každá z vrstev se skládá z neuronů. To je také první argument každé vrstvy. Čím větší je objem dat, tím více neuronů by každá vrstva měla mít. Pro otestování správného počtu neuronů jsem pro každou vrstvu vyzkoušel hodnoty 32, 64, 128, 256 a 512.

Dalším argumentem vrstvy je aktivační algoritmus. Podle zvoleného algoritmu, se pak vrstva rozhoduje, který z jejích neuronů má být aktivován, tedy zda má být jeho výstup propagován do další vrstvy. Pro skryté vrstvy jsem zkoušel dva druhy algoritmů — Sigmoid a ReLu. Pro výstupní vrstvy jsem se rozhodoval mezi Softmax a Linearním.

U modelu je potřeba nastavit optimalizační algoritmus. Po zpracování vstupu neuronovou sítí je nutné rozhodnout, jak využít rozdíl mezi hodnotami, které byly zpracováním vytvořeny, a hodnotami, které jsou známé jako správné ze zadaných dat pro učení sítě. Toto rozhodování umožňuje optimalizační algoritmus.[15]

U optimalizačního algoritmu jsem pro testování toho nejlepšího zvolil SGD, RMSprop a ADAM.

Po vytvoření modelu jsem za pomoci knihoven Hyperas a Hyperopt spustil síť na vytvořených datech a nechal tyto knihovny rozhodnout, která kombinace parametrů dosahuje nejlepších výsledků.

Po provedení výpočtu se ukázalo, že nejlepší nastavení modelu je následující:

- Počet skrytých vrstev - 4
- Vstupní vrstva
  - Počet neuronů - 64
  - Aktivační algoritmus - ReLu
- První skrytá vrstva
  - Počet neuronů - 32
  - Aktivační algoritmus - ReLu
- Druhá skrytá vrstva
  - Počet neuronů - 32
  - Aktivační algoritmus - Sigmoid
- Třetí skrytá vrstva
  - Počet neuronů - 64

  - Aktivační algoritmus - Sigmoid
- Čtvrtá skrytá vrstva
  - Počet neuronů - 64
  - Aktivační algoritmus - Sigmoid
- Výstupní vrstva
  - Aktivační algoritmus - Softmax
- Nastavení modelu
  - Optimalizátor - ADAM
  - Funkce ztráty - Categorical Cross-Entropy

Vytvoření a použití neuronové sítě se skládá ze 3 neprovázaných částí.

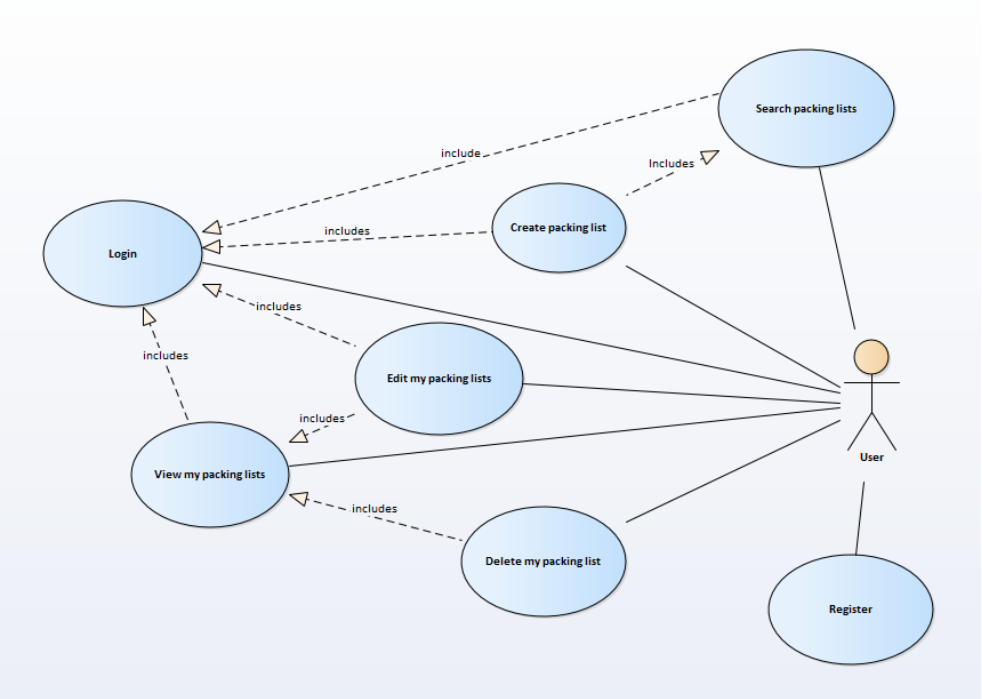
První částí, jak už jsem zmiňoval, je třída pro testování nejvhodnějších parametrů pro neuronovou síť.

Druhou částí je třída pro vytvoření modelů. V této třídě jsou již použity parametry z předchozího testování. Každý model je trénován na 500 000 záznamech. Po natrénování je uložen pro další použití.

Třetí částí je REST server, na který jsou přesměrovány požadavky na doporučení předmětů z hlavního serveru. Server pro každou kombinaci počasí a kritéria načte z databáze předměty, které dané kombinaci odpovídají. Podle Id předmětu si zároveň načte uložené modely z úložiště. Následně na těchto modelech za pomoci všech zadaných parametrů provede predikci. Takto postupuje pro všechny kombinace. Postupně vytváří seznam předmětů, které mají být doporučeny. Nakonec celý tento seznam odešle zpět na původní server.

3.5 Diagramy

3.5.1 UseCase



Obrázek 3.6: Use case

Jak bylo dříve zmíněno, uživatel bude moc provádět následující aktivity:

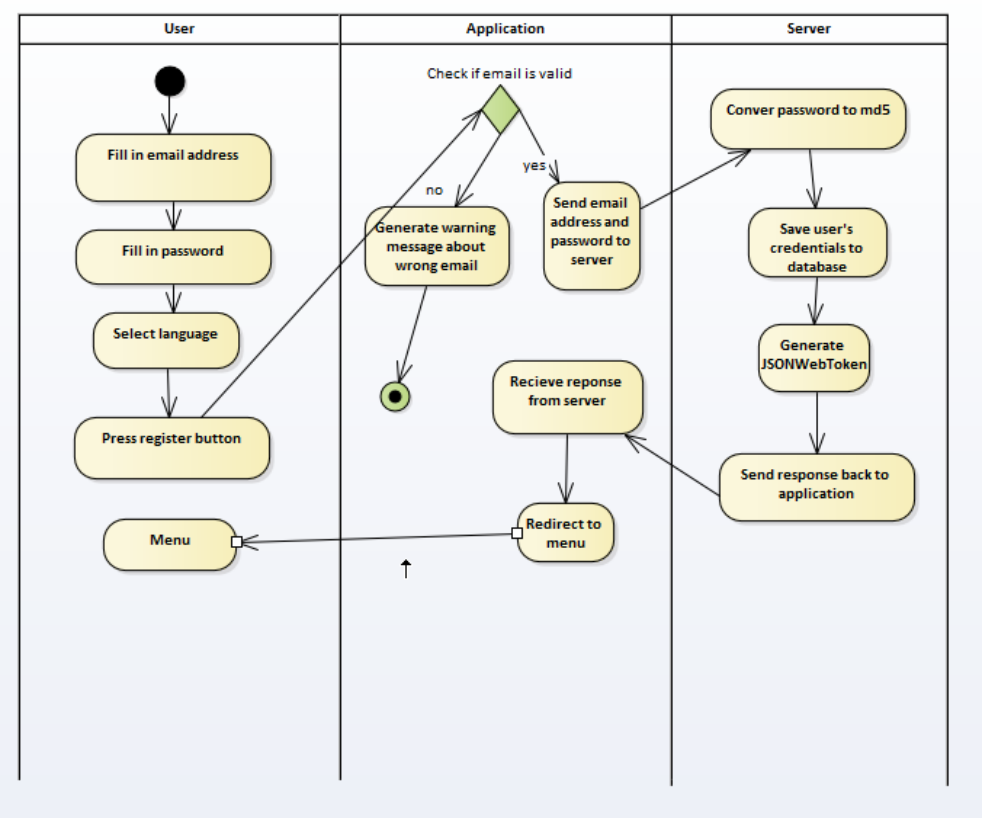
- Registrace
- Přihlášení
- Vytvoření nového seznamu
- Vyhledávání seznamů
- Zobrazení svých seznamů
- Editování své seznamy
- Mazání své seznamy

3.5.2 Popis jednotlivých aktivit

3.5.2.1 Registrace a Přihlášení

Při otevření aplikace se uživatel dostane na obrazovku přihlášení. Zde vyplní email, kterým se dříve registroval. Dále zadá své heslo. Uživatel má možnost zvolit si jazyk aplikace. V případě, že jazyk nezvolí defaultně je zvolena angličtina. Nakonec stiskne tlačítko registrovat nebo přihlásit.

Registrace Aplikace pošle požadavek na server s emailem a heslem uživatele. Server převede heslo na bcrypt hash. Následně uloží uživatelské údaje do databáze a vygeneruje JSONWebToken pro autorizaci uživatele na další požadavky. Tento token následně odešle zpět do aplikace. Následně je uživatel přesměrován do aktivity menu.

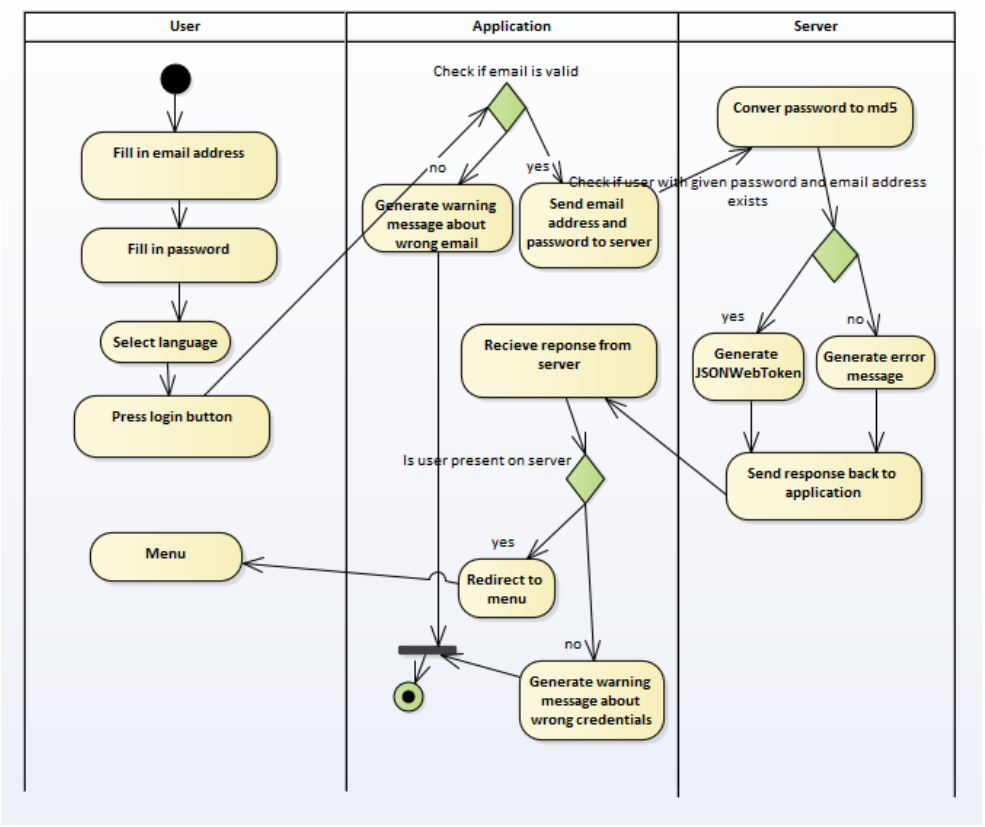


Obrázek 3.7: Registrace

Přihlášení Aplikace pošle požadavek na server se zadaným emailem a heslem. Server převede heslo na bcrypt hash. Následně zkontroluje, zda v databázi existuje uživatel s danou emailovou adresou a heslem. Pokud takovýto

uživatel existuje, vygeneruje JSONWebToken pro autorizaci uživatele na další požadavky. Tento token následně odešle zpět do aplikace.

V případě, že uživatel neexistuje, server odešle zpět zprávu o špatných přihlašovacích údajích. Pokud byl uživatel ověřen, aplikace ho přesměruje do aktivity menu. V opačném případě se uživateli zobrazí upozornění, že zadal špatné přihlašovací údaje.

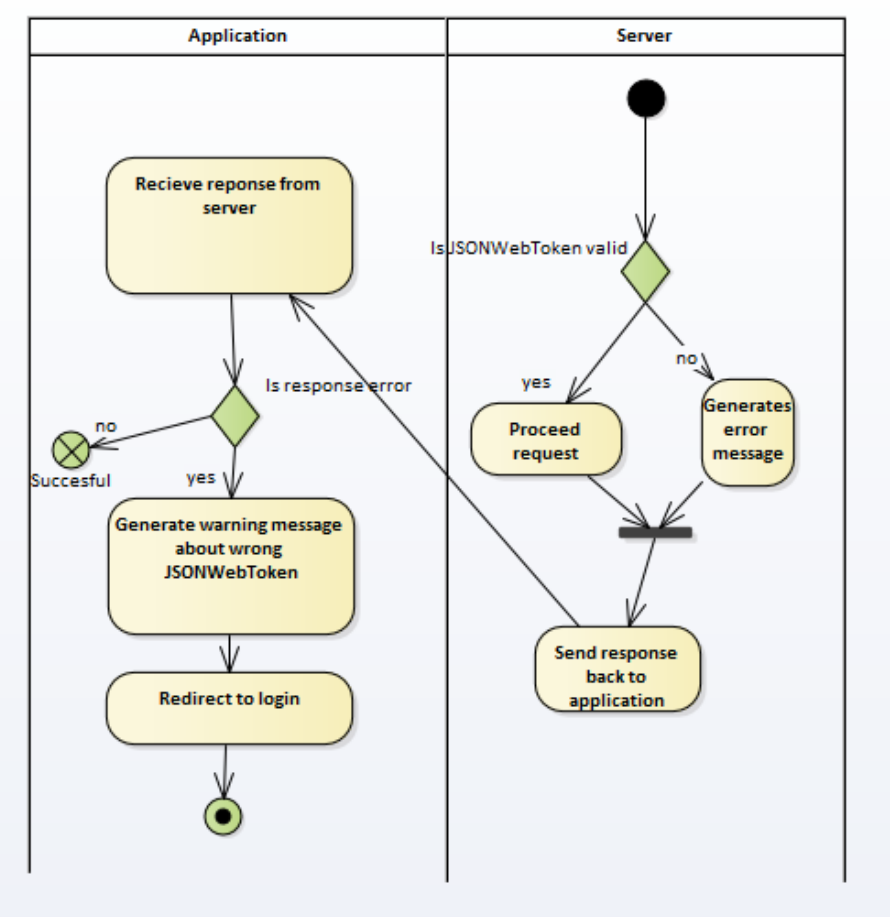


Obrázek 3.8: Přihlášení

3.5.3 Řízení požadavků na serveru

Pro každý požadavek zaslaný na server je jasně stanovený postup jeho vyřízení. Prvně se zkontroluje, zda je JSON Web Token validní. Získání tohoto tokenu je popsáno v rámci aktivit přihlášení a registrace. Následně se při validním tokenu provede požadavek a jeho výsledek je navrácen do aplikace. V opačném případě je vygenerována chybová zpráva, která je opět navrácena do aplikace. V případě chybové hlášky je uživatel vrácen na přihlašovací/registrační akti-

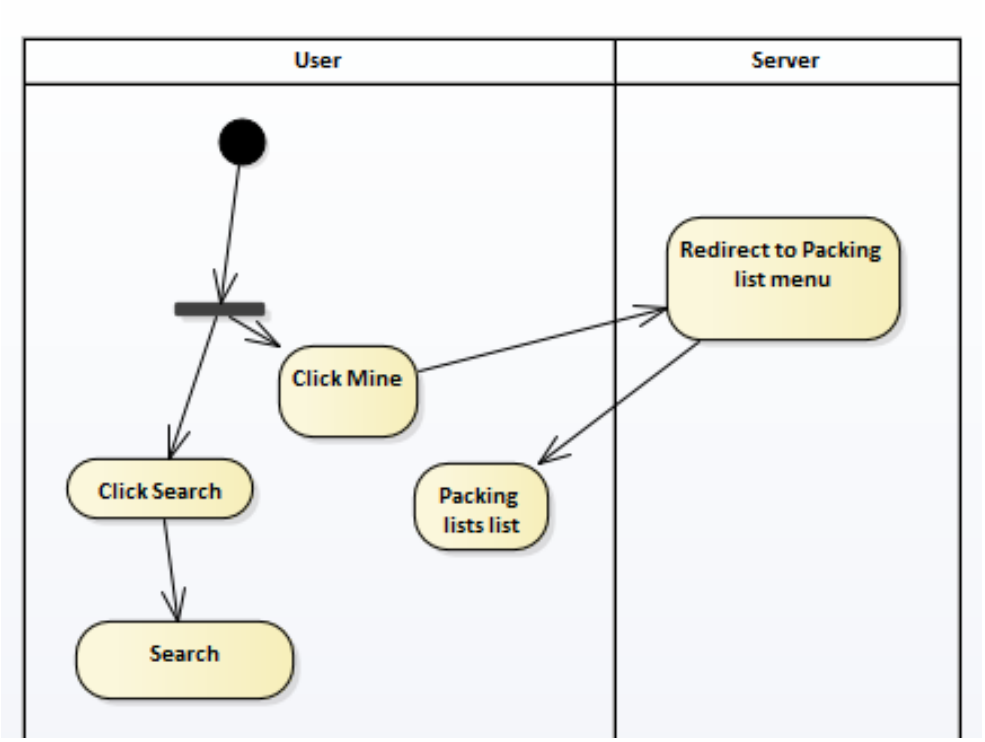
vitu. Tento diagram je použit pro všechny dotazy na stranu serveru. Důvodem je zjednodušení diagramů. Složitější dotazy na server budou popsány níže.



Obrázek 3.9: Procesování požadavků

3.5.3.1 Menu

Po registraci, či přihlášení je uživatel přesměrován na menu. V menu může uživatel zvolit vyhledání seznamu nebo zobrazení svých seznamů. Po zvolení některé z možností je uživatel přesměrován na danou aktivitu.



Obrázek 3.10: Menu

3.5.3.2 Vyhledání seznamu

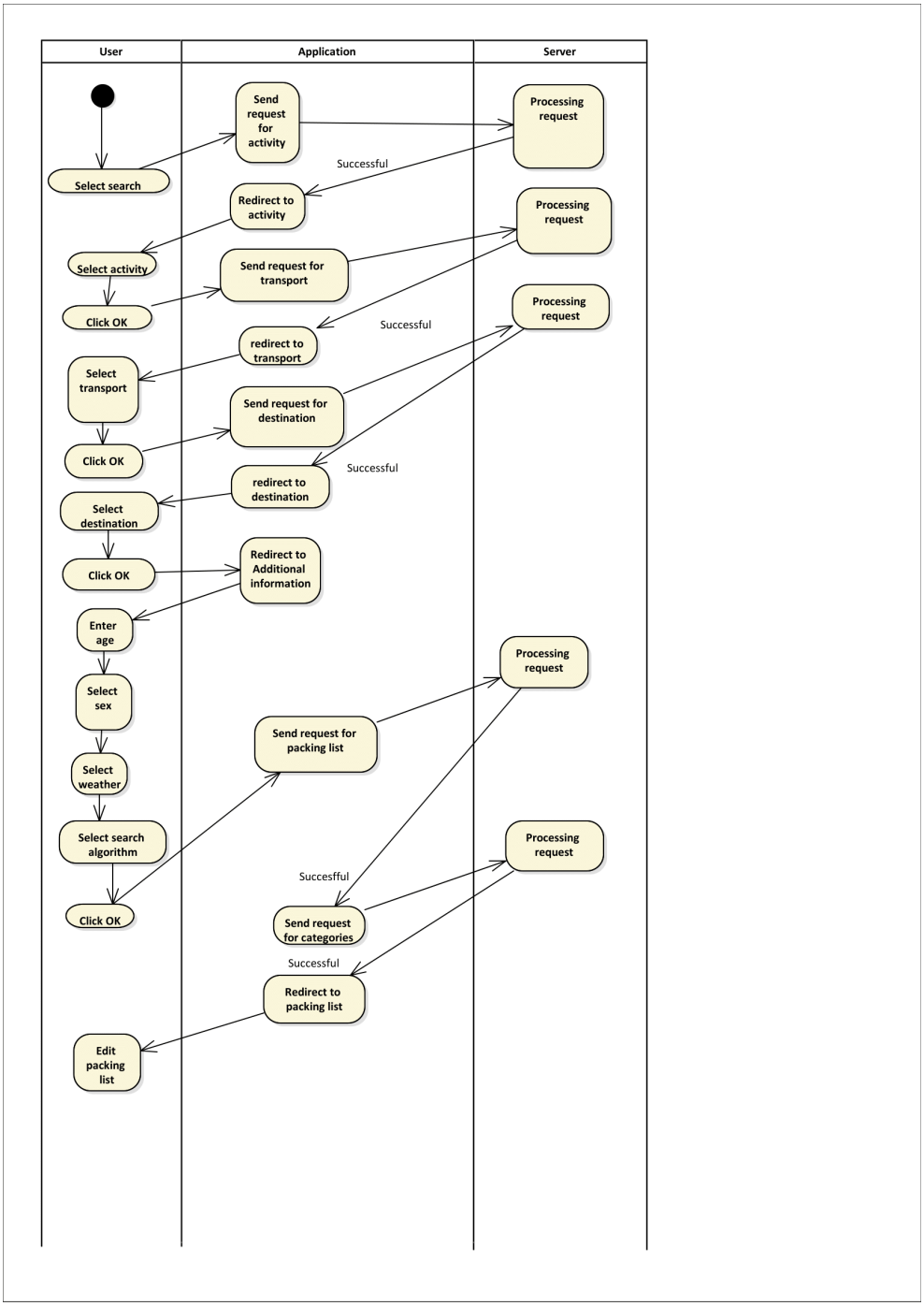
Po zvolení vyhledání v menu, se prvně aplikace dotáže serveru na všechny možné aktivity. Následně je uživatel přesměrován na obrazovku aktivity.

Na této obrazovce si uživatel vybere všechny aktivity, které plánuje provádět během cesty. Tento výběr potvrdí tlačítkem OK.

Dále je mu pomocí dotazu na server zobrazen kompletní seznam možných typů transportu. Zde si vyberej ten, který plánuje využít. Výběr následně opět potvrdí.

Poté je přesměrován na výběr destinace. Uživatel vybere, do které země cestuje a výběr potvrdí.

Poslední z obrazovek je vyplnění nezbytných informací jako je věk, pohlaví, předpokládané počasí a algoritmus pro doporučování. Po zadání všech těchto informací je na serveru vyhledán seznam doporučených věcí pro danou cestu. Uživatel je následně přesměrován do personalizace daného seznamu.



Obrázek 3.11: Vyhledání seznamu

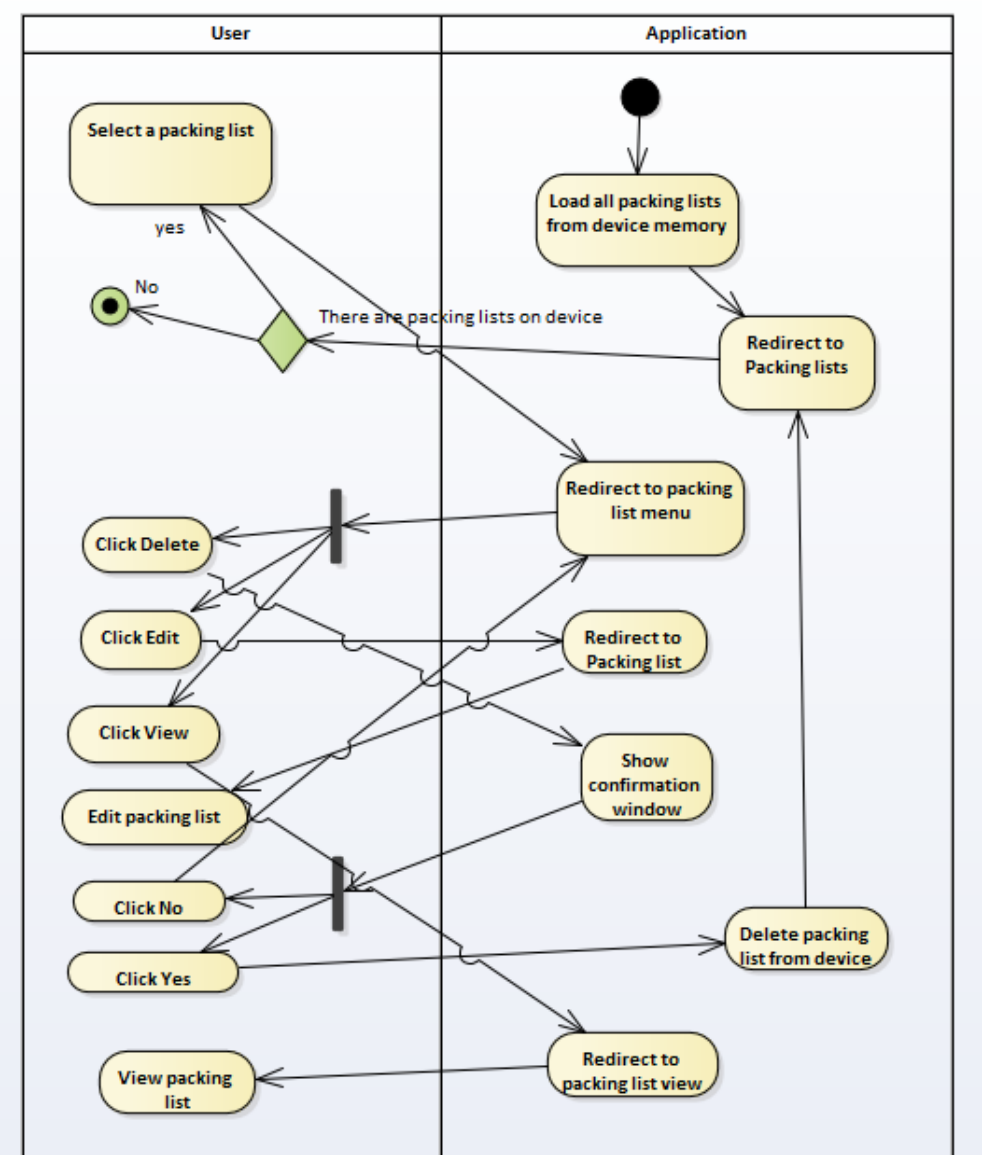
3.5.3.3 Zobrazení vlastních seznamů

Při zvolení zobrazení uživatelových seznamů, je prohledána složka, kam se tyto seznamy ukládají. Pokud žádné seznamy neobsahuje, je zobrazena informace o tomto faktu. V opačné případě jsou zobrazeny jednotlivé seznamy podle názvů, pod nimiž si je uživatel uložil.

Zde uživatel vybere jeden ze zmíněných seznamů. Následně je přesměrován na menu seznamu.

V tomto menu může uživatel vybírat ze tří možností: upravit seznam, zobrazit seznam nebo smazat seznam.

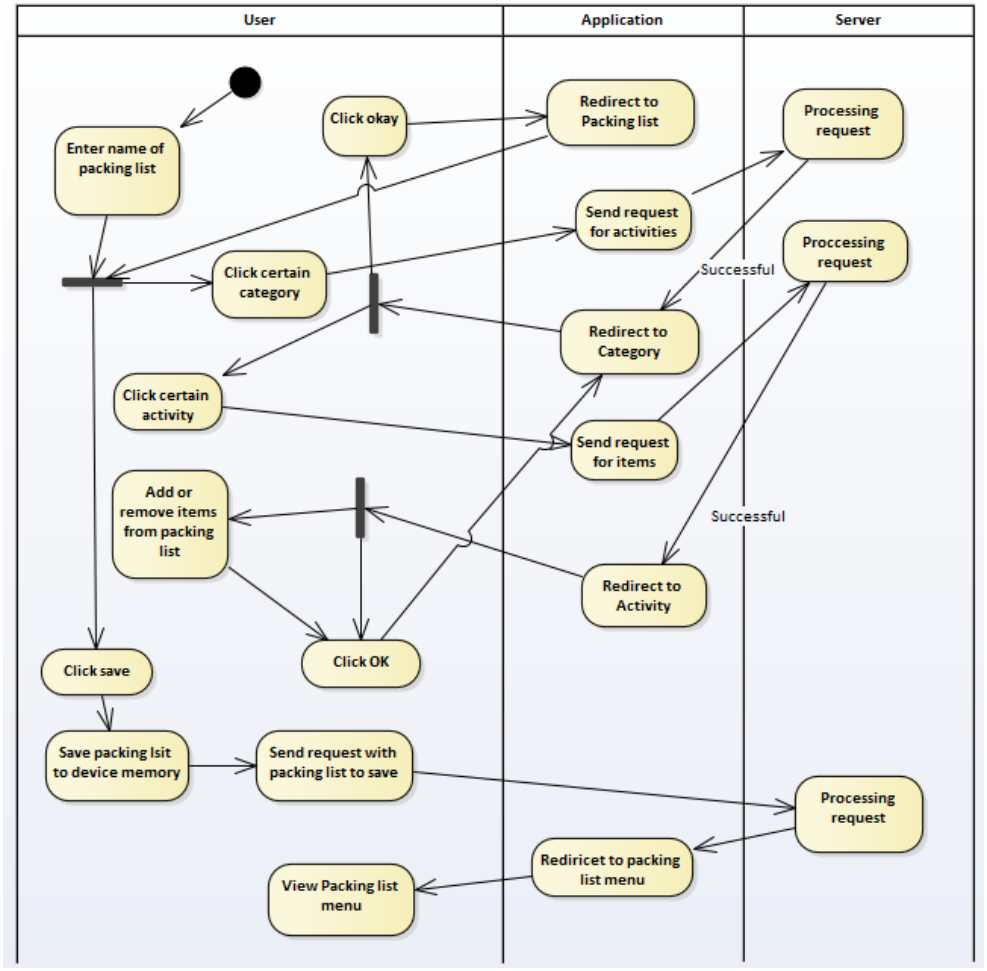
Při zvolení smazání je dotázán, jestli opravdu chce seznam smazat. Pokud uživatel svou volbu nepotvrdí, je přesměrován zpět do menu aktuálního seznamu. V opačném případě je přesměrován na zobrazení stávajících seznamů a jím zvolený seznam je smazán. V ostatních případech je přesměrován na vybranou aktivitu.



Obrázek 3.12: Zobrazení svých seznamů

3.5.3.4 Editace seznamů

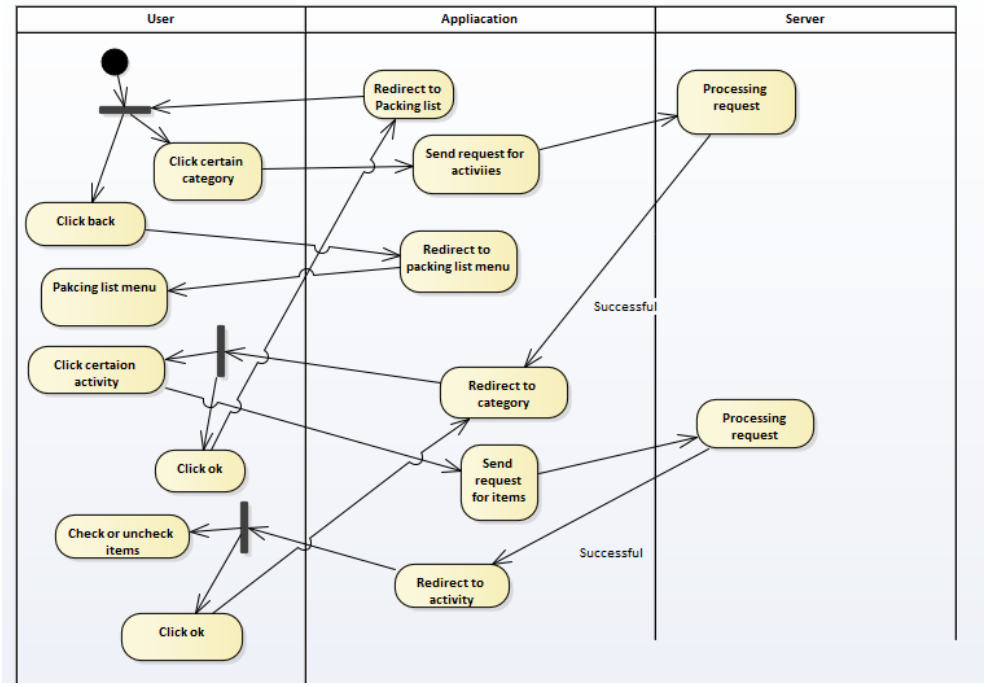
Do editace seznamu se uživatel dostane z vyhledávání podle jednotlivých kritérií nebo skrze úpravu svého již existujícího seznamu. V postupu prvotně zvolí jméno svého seznamu. Následně projde všechny kategorie. U každé kategorie vždy zvolí jednotlivé aktivity. V těchto aktivitách zvolí předměty, které si plánuje vzít s sebou. Nakonec takto upravený seznam uloží. Takto upravený seznam je uložen na použité mobilní zařízení i na server, kde jsou přepočítány jednotlivé relativní četnosti předmětů. Po ukončení je uživatel přesměrován na menu seznamů.



Obrázek 3.13: Editace

3.5.3.5 Zobrazení

Během samotného aktu balení si uživatel může zobrazit detail seznamu vybraného v aktivitě zobrazení vlastních seznamů. Postup je podobný jako u editace, ale namísto zobrazení jak přidaných, tak nepřidaných předmětů, jsou zobrazeny pouze přidané předměty. Uživatel projde všechny kategorie, kritéria a postupně si při balení jednotlivých předmětů dané předměty v seznamu odklikává. Po ukončení aktivity je uživatel přesměrován do menu seznamů.



Obrázek 3.14: Zobrazení

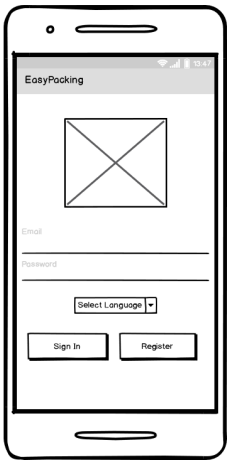
3.6 MockUps

Každá obrazovka, kromě té pro přihlášení a registraci, obsahuje záhlaví, v němž je název aplikace, tlačítko menu a tlačítko zpět. Obrazovka pro přihlášení a registraci obsahuje v záhlaví pouze název aplikace. Tlačítko menu odkazuje na obrazovku menu. Tlačítko zpět odkazuje na předchozí aktivitu.

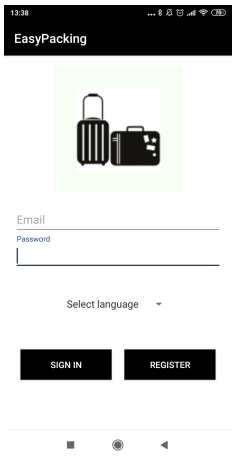
3.6.1 Přihlášení

Obrazovka přihlášení obsahuje logo aplikace. Pod logem je umístěno pole pro zadání emailu uživatele a následuje pole pro zadání hesla. Pak jsou zde vidět

dvě tlačítka, a to Přihlásit nebo Registrovat. V neposlední řadě obsahuje možnost volby jazyka.



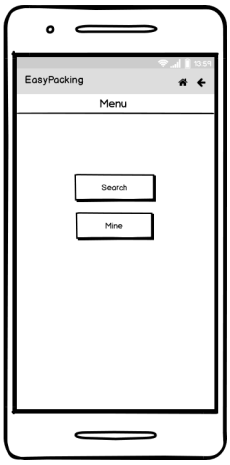
Obrázek 3.15: Mockup přihlášení



Obrázek 3.16: Prototyp přihlášení

3.6.2 Menu

Menu je hlavním rozcestníkem aplikace. Zde se uživatel rozhoduje za jakým účelem do aplikace přišel. Menu se skládá ze dvou tlačítek. Prvním je Vyhledat. Tímto tlačítkem se uživatel dostane do aktivity pro vyhledání seznamu podle jím stanovených kritérií. Druhé tlačítko se jmenuje Moje seznamy. Pokud uživatel zvolí Moje seznamy, dostane se do možnosti správy již vytvořených seznamů.

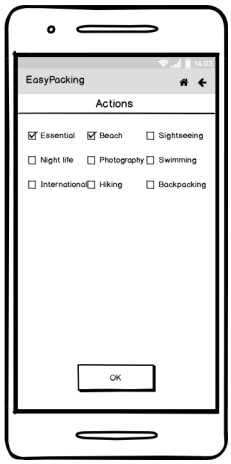


Obrázek 3.17: Mockup menu

Obrázek 3.18: Prototyp menu

3.6.3 Kritéria

Zde se uživatel rozhoduje, jaké činnosti bude v rámci své cesty podnikat. Pole s činnostmi je možné posouvat. To hlavně proto, že činností bude pravděpodobně více než se na obrazovku vejde. Každá činnost je opatřena zaškrtávacím polem s popiskem činnosti. Nakonec tato obrazovka obsahuje tlačítko OK pro potvrzení výběru.

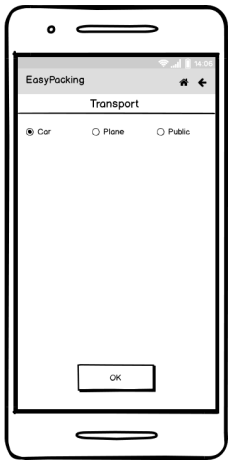


Obrázek 3.19: Mockup kritéria

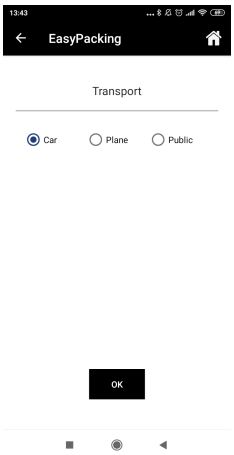
Obrázek 3.20: Prototyp kritéria

3.6.4 Typ transportu

Obrazovka typ transportu obsahuje všechny možnosti dopravy, které uživatel může v aplikaci zvolit. Každý typ transportu je reprezentován radiobuttonem. V dolní části obrazovky se nachází potvrzovací tlačítko OK.



Obrázek 3.21: Mockup typ transportu



Obrázek 3.22: Prototyp typ transportu

3.6.5 Destinace

Zde uživatel zadá do jaké destinace bude cestovat. Tato obrazovka se podobá obrazovce Aktivity. Rozdíl je v tom, že namísto zaškrtávacích polí, může uživatel zvolit pouze jednu destinaci. Opět bude pole s destinacemi posouvací, jelikož nyní je ve světě okolo 200 zemí.

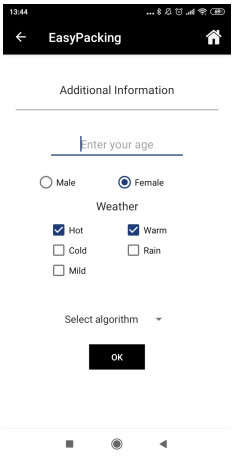
Obrázek 3.23: Mockup destinace

Obrázek 3.24: Prototyp destinace

3.6.6 Doplňující informace

Pro přesnější vyhledávání je nutné, aby uživatel zadal svůj věk, pohlaví a předpokládané počasí. Proto na této obrazovce je jak pole pro vyplnění věku, jenž je omezené pouze na číselný vstup, tak dva přepínače, a to buď muž, nebo žena. Následně jsou zde možnosti vyplnění předpokládaného počasí. Navíc je zde možnost výběru doporučujícího algoritmu. Opět je zde tlačítko pro potvrzení výběru.

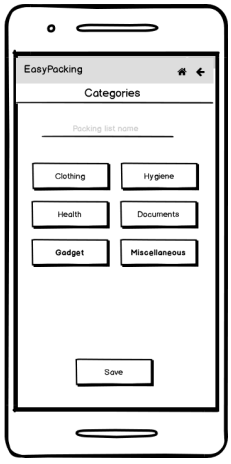
Obrázek 3.25: Mockup doplňují informace



Obrázek 3.26: Prototyp doplňující informace

3.6.7 Kategorie

Na této obrazovce uživatel nalezne jednotlivé typy kategorií předmětů, spolu s polem pro zadání názvu seznamu a tlačítkem pro uložení seznamu.

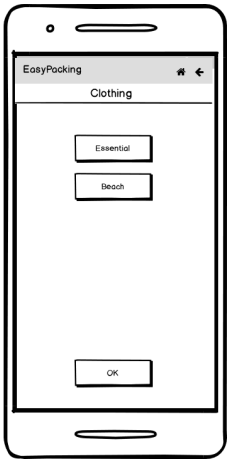


Obrázek 3.27: Mockup kategorie

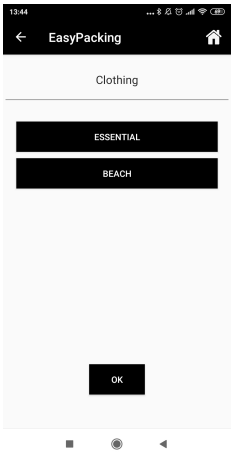
Obrázek 3.28: Prototyp kategorie

3.6.8 Seznam

Zde uživatel nalezne aktivity, které si dříve zvolil a tlačítko OK pro návrat na obrazovku kategorií.



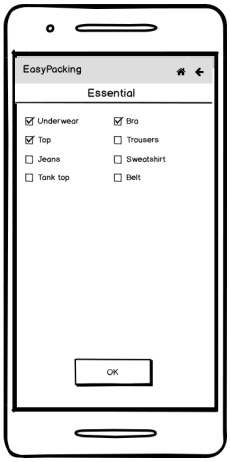
Obrázek 3.29: Mockup aktivity



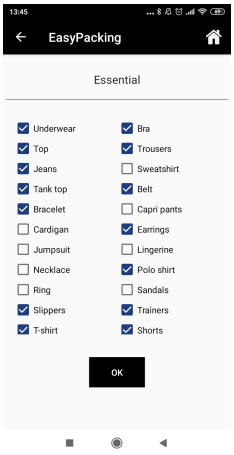
Obrázek 3.30: Prototyp aktivity

3.6.9 Předměty

Na této obrazovce uživatel přidává do seznamu předměty z jednotlivých kombinací kategorií a aktivit, které si bude brát s sebou. Je zde vidět seznam všech věcí, které si uživatel může sbalit. Nakonec je zde potvrzovací tlačítko. Opět je pole s předměty posuvné.



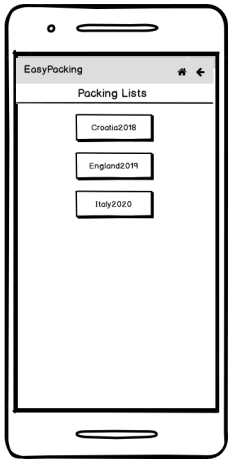
Obrázek 3.31: Mockup předměty



Obrázek 3.32: Prototyp předměty

3.6.10 Zobrazení vlastních seznamů

Zde se uživateli zobrazí jeho vlastní seznamy. Jedná se o tlačítka pod sebou, které jsou označena názvem, který uživatel zadal při ukládání.

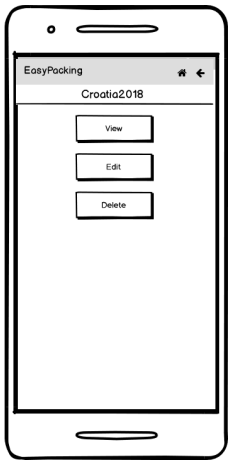


Obrázek 3.33: Mockup vlastní seznamy

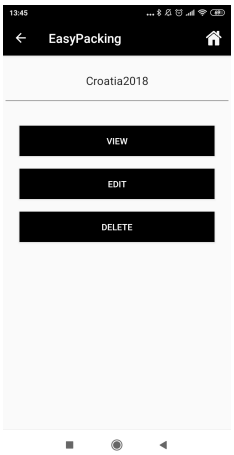
Obrázek 3.34: Prototyp vlastní seznamy

3.6.11 Menu seznamy

Na této obrazovce si uživatel vybírá, co chce se svým vybraným seznamem dělat. Jedná se o tři tlačítka: zobrazit seznam, upravit seznam nebo smazat seznam.



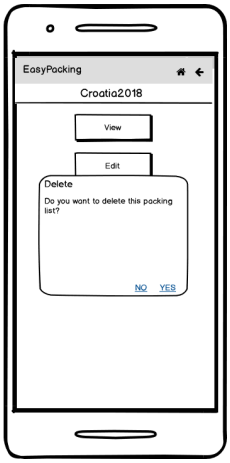
Obrázek 3.35: Mockup menu seznamy



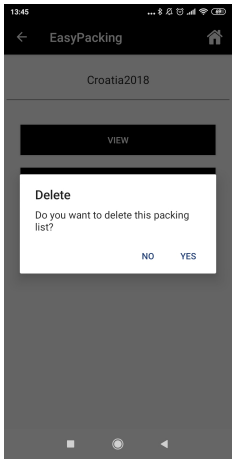
Obrázek 3.36: Prototyp menu seznamy

3.6.12 Smazání vlastního seznamu

Pokud uživatel zvolí v menu seznamu smazání onoho seznamu zobrazí se mu potvrzení o smazání.



Obrázek 3.37: Mockup smazání seznamu



Obrázek 3.38: Prototyp smazání seznamu

3.7 Testování

Pro testování bylo nutné použít server, abych mohl aplikaci testovat na mobilním telefonu namísto emulátoru v Android studiu. Ten, jak jsem již dříve popisoval, je pomalý a náročný na výkon. Z tohoto důvodu jsem použil server Heroku.

Heroku se soustavně zaměřuje na aplikace a zkušenosti vývojářů s aplikacemi. Umožňuje společnostem všech velikostí vyvíjet aplikace, bez nutnosti řešit hardwarové a serverové požadavky.[16]

Heroku je cloudové řešení pro vývoj i produkci aplikačních serverů. Podporovány jsou tyto jazyky:

- Node
- Ruby/ Ruby on Rails
- Java
- PHP
- Python
- Go
- Scala
- Closure

Pro tuto práci jsem využil podporu Ruby on Rails spolu s podporou databáze PostgreSQL. Při používání Heroku stačí pouze nahrát aplikační server na Heroku git. Server se následně přeloží a spustí. Pomocí jednoho příkazu pak lze vytvořit databázi. Pokud aplikační server obsahuje i data pro vložení do databáze, tak pomocí dalšího příkazu Heroku nahraje všechny tato data do databáze.

Jelikož i tak nahrání nějakou dobu trvá, používal jsem zároveň program Insomnia. Jedná se o REST api klienta. Tímto programem jsem testoval jednotlivé dotazy na aplikační server.

3.7.1 Testování aplikačního serveru

Pro testování aplikačního serveru jsem použil test driven development. Před implementací nové funkcionality jsem první napsal testy, jak by se měla daná funkcionalita chovat, a pak postupně funkcionalitu implementoval, aby splňovala všechny dané testy. Pro testování jsem použil následující Ruby knihovny:

- Rspec
- FactoryGirl
- Faker

Rspec je testovací framework. Jedná se o testování chování aplikace. Nezaměřuje se tedy na to, jak aplikace funguje, ale na to, jak se má chovat.

Na tomto příkladě je vidět použití FactoryGirl a Faker. FactoryGirl definuje, jak se má daný objekt vytvořit. Máme zde objekt translation_criterion. První je zde definován jazyk, jenž je pro všechny následně vytvořené instance nastaven na angličtinu. Následně je zde název kritéria. Zde je použit Faker. Aby byl každý vytvořený objekt odlišný, je zde nastaveno vytvoření náhodného slova pomocí Faker. Posledním parametrem je id kritéria, to je pro testovací účely nastaveno na nil.

```
FactoryGirl.define do
    factory :translation_criterion do
        language { 'en' }
        name { Faker::Lorem.word }
        criterion_id nil
    end
end
```

Po takto nadefinované FactoryGirl můžeme jednoduše generovat, kolik chceme objektů translation_criterion, pomocí zavolání funkce create nebo create_list, pokud chceme vygenerovat více objektů v kterémkoli testovacím souboru.

3.7.1.1 Testování jednotlivých částí aplikačního serveru

Test Databáze V testu databáze jsem testoval, jestli každá tabulka (zde PackingList) obsahuje všechny atributy a vazby na ostatní tabulky.

```
RSpec.describe PackingList, type: :model do
    it { should belong_to(:user) }
    it { should have_many(:list_item).dependent(:destroy) }
    it { should have_many(:list_criterion).dependent(:destroy) }
    it { should belong_to(:destination) }
    it { should validate_presence_of(:age) }
    it { should validate_presence_of(:sex) }
end
```

Na příkladu testování modelu PackingList je vidět testování, zda PackingList má cizí klíč pro model User a Destination. Jestli je PackingList navázán na tabulky list_item a list_criterion a zda, při smazání PackingListu, se zároveň smažou i záznamy v těchto tabulkách navazující na daný PackingList. Nakonec je testováno, zda PackingList obsahuje věk a pohlaví.

Test směřování Během testu směrování jsem testoval, zda jednotlivé cesty, které se budou volat z aplikace, odkazují na správné funkce.

```
RSpec.describe UsersController, type: :routing do
    describe 'routing' do
        it 'routes to #signup' do
          expect(post: '/signup').to route_to('users#create')
        end

        it 'routes to #login' do
           expect(post: '/auth/login').to route_to('authentication#authenticate')
        end
    end
end
```

Zde je vidět testování směrování pro třídu User. Najdeme zde dva testy. První test je pro vytvoření uživatele. Testuje se, jestli požadavek post na /signup směřuje do funkce Signup v třídě User. Druhý test pro přihlášení

uživatele ověřuje, zda post požadavek na /auth/login odkazuje na metodu authenticate ve třídě Authentication.

Test zasílání požadavků Nakonec jsem tetoval jednotlivé požadavky na funkcionality, a to jak se správnými, tak špatnými požadavky a jejich návratové hodnoty.

```
describe 'POST /signup' do
  context 'when valid request' do
    before do
      post '/signup', params: valid_attributes.to_json, headers: headers
    end
    it 'creates a new user' do
      expect(response).to have_http_status(200)
    end
    it 'returns success message' do
      expect(json['message']).to match(/Account created successfully/)
    end
    it 'returns an authentication token' do
      expect(json['auth_token']).not_to be_nil
    end
  end
  context 'when invalid request' do
    before  post '/signup', params: , headers: headers

    it 'does not create a new user' do
      expect(response).to have_http_status(422)
    end
    it 'returns failure message' do
      expect(json['message'])
        .to match('Validation failed: Password cant́ be blank, '
          'Username cant́ be blank
          'Email cant́ be blank)
    end
  end
end
```

Na příkladu testování metody registrace uživatele je vidět následující; v první polovině je testováno chování funkcionality při zaslání správných dat. Kontroluje se, zda je navrácen kód 200, jestli návratová zpráva obsahuje hlášku "Account created successfully"a zároveň jestli obsahuje JSONWebToken. V druhé části můžeme vidět test zaslání špatných dat v požadavku. Kontroluje se, zda návratový kód je roven 422 a zda obsahuje zprávu o špatných údajích.

3.7.2 Testovací scénáře aplikace

3.7.2.1 Vyhledání a vytvoření seznamu

V tomto scénáři bude otestováno, zda správně funguje vyhledání, upravení a uložení seznamu.

1. Tester se přihlásí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost vyhledat.
3. V obrazovce Aktivity vybere aktivity. Výběr odsouhlasí stisknutím tlačítka OK.
4. Na následující obrazovce Typ transportu vybere jeden z možných typů dopravy a potvrdí svůj výběr tlačítkem OK.
5. V destinacích vybere některou z uvedených destinací. Následně stiskne tlačítko OK.
6. V aktivitě Dodatečné informace vybere pole označené „Enter your age“ a vyplní svůj věk. Poté zvolí, zda je muž či žena. Vybere některý z možných typů počasí. Nakonec zadané informace odsouhlasí kliknutím OK.
7. Následně v kategoriích vybere některou kategorii.
8. Ve zvolené kategorii vybere některou z aktivit.
9. Ve zvolené aktivitě si kliknutím některé předměty přidá a některé odebere. Následně odsouhlasí stisknutím OK.
10. V obrazovce aktivit stiskne OK pro návrat do seznamu kategorií.
11. V navrácené obrazovce kategorie zvolí opět kategorii z kroku 7, následně aktivitu z kroku 8 a zkontroluje, zda jím zadané nebo odebrané předměty odpovídají jeho výběru.
12. Opakuje kroky 7-11 pro všechny zobrazené kategorie a aktivity. Následně vyplní v obrazovce kategorie název seznamu a stiskne tlačítko uložit.

3.7.2.2 Zobrazení seznamu pomocí navigace přes Moje seznamy

Pomocí tohoto testovacího scénáře tester ověří, zda vše funguje při zobrazování uživatelovo seznamu pomocí navigace přes uživatelovi seznamy.

1. Tester se přihlásí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost Moje seznamy.

3. Na následující obrazovce Moje seznamy nalezne seznam, který dříve vytvořil. Tento seznam vybere pomocí kliknutí.
4. V Menu seznamu vybere tlačítko Zobrazit.
5. Poté postupuje podle kroku 11 z předešlého scénáře. Následně na obrazovce Kategorie stiskne tlačítko OK. Takto se navrátí do obrazovky Menu seznamu.

3.7.2.3 Editace seznamu pomocí navigace přes Moje seznamy

V tomto scénáři tester ověří, zda se aplikace chová správně při editaci uživatelských seznamů.

1. Tester se přihlásí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost Moje seznamy.
3. Na následující obrazovce Moje seznamy nalezne seznam, který dříve vytvořil. Vybere tento seznam pomocí kliknutí.
4. V Menu seznamu vybere tlačítko Upravit.
5. Poté postupuje podle kroku 12 z předešlého scénáře. Následně na obrazovce kategorie přejmenuje seznam a stiskne tlačítko Uložit. Takto se vrátí do obrazovky Menu seznamu.

3.7.2.4 Smazání seznamu pomocí navigace přes Moje seznamy

Tento testovací scénář ověřuje, zda smazání uživatelovo seznamu proběhne v pořádku.

1. Tester se přihlásí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost Moje seznamy.
3. Na následující obrazovce nalezne dva seznamy. První seznam je ten, který byl vytvořen ve scénáři Vyhledání a vytvoření seznamu. Druhý seznam je ten, který byl vytvořen za pomocí scénáře Editace.
4. Uživatel vybere jeden ze zobrazených seznamů.
5. Na obrazovce Menu seznamu vybere Odstranit seznam.
6. Zobrazí se potvrzovací okno, kde vybere možnost Ano.
7. Uživatel je následně přesměrován zpět do Moje seznamy, kde pro druhý seznam zopakuje kroky 3-6.
8. Po odstranění obou seznamů zkontroluje, zda se na obrazovce objevila zpráva o neexistujících seznamech.

3.7.3 Testování chybových zpráv

Dále jsem testoval, zda jsou správně ošetřeny chybové stavy aplikace. Chybové stavy a jejich ošetření jsou následující:

- Přihlášení nesprávnými údaji — zobrazí se zpráva "Špatně zadané údaje"
- Registrace již registrované emailové adresy — zobrazí se zpráva "Tento email je již používán"
- Zobrazení uživatelových seznamů, pokud ještě žádné nevytvořil — na obrazovce Moje seznamy se zobrazí chybová zpráva "Nemáte ještě vytvořené žádné seznamy"
- Volání jednotlivých funkcionalit serveru bez validního JSONWebTokenu — zobrazení chybové zprávy "Nejste přihlášen nebo Váš token expiroval"

3.7.4 Testování uživatelského rozhraní

Pro testování uživatelského rozhraní jsem využil budoucích uživatelů. Nejasnosti v UI, se kterými se setkali, byly velmi podobné. Zde jsou jednotlivé problémy:

- Není jasné, jak vytvořit seznam
- Při vyhledání seznamu jsou některé položky již označené
- Po uložení seznamu není jasné, jak dále postupovat
- Aplikace padá při uložení seznamu

Řešení problémů:

- Vytvoření tlačítka Vytvořit seznam v sekci mé seznamy
- Přejmenování tlačítka Vyhledat seznam na Vytvořit seznam v sekci menu
- Přejmenování tlačítka Zobrazit seznam na Zabalit
- Přidání práva pro čtení externího úložiště

V rámci testování uživatelského rozhraní byli testující uživatelé s aplikací spokojeni. Zjistil jsem, že některé postupy nejsou zcela intuitivní a opravil jsem je podle doporučení uživatelů. V rámci uživatelského testování jsem zjistil, že je nutné testovat aplikaci na více zařízeních, jelikož chyba pádu aplikace při ukládání seznamu na mnou testovaném zařízení nenastala.

3.7.5 Porovnání statistické metody a neuronové sítě

Pro porovnání metod doporučování vyberu některé kombinace kritérií. Pro dané kombinace určím, které předměty by měly být doporučeny. Následně použiji obě metody doporučení a zjistím jejich procentuální úspěšnost. Pro porovnání zvolím 3 různé kombinace předpokladů. Pro každou kombinaci určím procentuální úspěšnost a následně rozhodnu, která metoda je úspěšnější.

Kombinace zvolím tak, aby neodpovídaly kombinacím, které jsem použil pro prvotní plnění databáze. Tímto způsobem mohu porovnat i doplňování chybějících hodnot.

Pro první výběr zvolím následující kritéria:

- Aktivity — základní vybavení, pláž, potápění
- Druh dopravy — auto
- Destinace — Itálie
- Věk — 25 let
- Pohlaví — muž
- Počasí — horko

Předmětů má být doporučeno 72.

Statistická metoda Metoda doporučila 75 předmětů. Z těchto předmětů bylo 14 předmětů zbytečných a 11 předmětů scházelo. Metoda doporučila správně 61 předmětů.

Úspěšnost je v tomto případě 85%.

Neuronová síť Síť doporučila 53 předmětů. 7 předmětů bylo navíc. 35 předmětů scházelo.

Úspěšnost je v tomto případě 64%.

Pro druhý výběr zvolím následující kritéria:

- Aktivity — základní vybavení, cyklistika, běh
- Druh dopravy — veřejná doprava
- Destinace — Velká Británie
- Věk — 40 let

- Pohlaví — žena
- Počasí — polojasno, zima, déšť

Správný počet předmětů je 88.

Statistická metoda V tomto případě bylo doporučeno 77 předmětů. 13 z nich bylo navíc a 24 jich chybělo.

Úspěšnost je v tomto případě 73%.

Neuronová síť Síť doporučila 42 předmětů; z toho 4 navíc a 49 chybělo.

Úspěšnost je v tomto případě 43%.

Pro třetí výběr jsem zvolil kritéria:

- Aktivity — základní vybavení, turistika, vaření
- Druh dopravy — letadlo
- Destinace — Španělsko
- Věk — 60 let
- Pohlaví — muž
- Počasí — teplo, horko

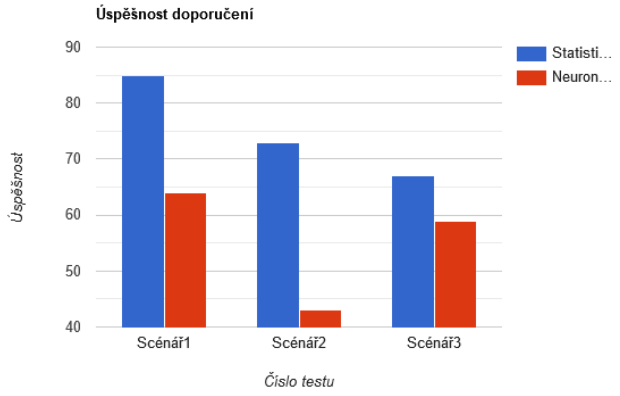
Počet doporučených předmětů má být 58.

Statistická metoda Doporučeno bylo 43 předmětů. Z toho 4 zbytečné. Scházelo 19 předmětů.

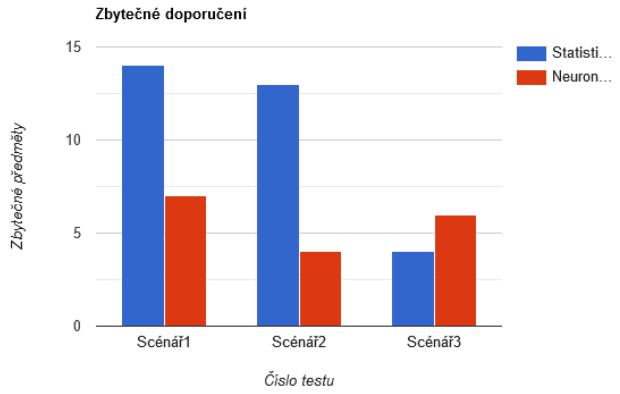
Úspěšnost je v tomto případě 67%.

Neuronová síť Doporučených předmětů je 40 předmětů; z toho 6 navíc a 24 chybí.

Úspěšnost je v tomto případě 59%.



Obrázek 3.39: Znázornění úspěšnosti



Obrázek 3.40: Znázornění počtu zbytečných předmětů

Výsledek Z porovnání je vidět, že průměrná úspěšnost statistické metody je 75%, zatím co u neuronové sítě je průmerná úspěšnost jen 55%. Můžeme tedy předpokládat, že statistická metoda by měla být lepší. Pokud se ovšem podíváme na průměrný počet špatně doporučených předmětů jedná se o 11 předmětů proti 6. Zde je úspěšnější neuronová síť.

Závěr

Výsledkem této diplomové práce je funkční aplikace EasyPacking spolu s aplikačním serverem a databází. Serverová část se skládá ze dvou serverů. Prvním je server v Ruby on Rails, který řeší většinu požadavků. Druhým je server napsaný v jazyce Python. Tento server se stará o neuronovou síť. Jednotlivé části byly otestovány. Serverová část byla testována za pomoci test driven developmentu. Aplikaci jsem testoval pomocí testovacích scénářů a uživatelského testování.

V práci jsem zhodnotil matematické modely a problémy spojené s vývojem databáze. Příkladem je Narozeninový paradox nebo Coupon collector's problem. V této části je vidět, že i v takto jednoduché aplikaci se lze setkat s problémy, které ji mohou omezovat. Zabýval jsem se také doporučujícími algoritmy. Jako doporučující algoritmy jsem zvolil statistickou metodu, založenou na relativní četnosti, a neuronovou síť.

Dále jsem uvažoval nad problémy se zabezpečením aplikace, zejména nad problémy SQL Injection a autentifikace. SQL Injection nakonec nebylo nutné řešit. Autentifikaci jsem vyřešil za pomoci JSONWebTokenu.

Pro samotnou realizaci jsem použil Ruby on Rails spolu s test driven developmentem. Pro databázi jsem použil PostgreSQL. Aplikaci jsem vyvinul v jazyce Kotlin. Komunikace aplikačního serveru s aplikací probíhá za pomoci JSON struktur.

V rámci práce jsem implementoval obě doporučující metody. Na základě vytvořených dat pro statistickou metodu, jsem vytvořil modely pro neuronovou síť. Poté jsem porovnával výsledky obou metod. Ukázalo se, že statistická metoda má větší úspěšnost v oblasti počtu doporučených předmětů v poměru k potřebným předmětům. Zároveň ale neuronová síť měla navrch v rámci omezení doporučení nepotřebných předmětů. Nelze jasně určit, která metoda je

lepší.

Literatura

[1] Hayes, A.; Pilling, G.: Coupon Collector Problem. Dostupné z: https://brilliant.org/wiki/coupon-collector-problem/

[2] Understanding the Birthday Paradox. Dostupné z: https://betterexplained.com/articles/understanding-the-birthday-paradox

[3] Diederik P.Kingma, J. L. B.: ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. Jan 2017.

[4] Taylor, C.: Structured vs. Unstructured Data. Mar 2018. Dostupné z: https://www.datamation.com/big-data/structured-vs-unstructured-data.html

[5] Foote, K. D.: A Review of Different Database Types: Relational versus Non-Relational. Apr 2018. Dostupné z: http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/

[6] Farrell, J.: The Basics of Database Indexes For Relational Databases. Aug 2017. Dostupné z: https://medium.com/jimmy-farrell/the-basics-of-database-indexes-for-relational-databases-bfc634d6bb37

[7] SOAP vs REST 101: Understand The Differences. Dostupné z: https://www.soapui.org/learn/api/soap-vs-rest-api.html

[8] NodeJs Vs Ruby On Rails : Analysis Worth Doing ! – codeburst. Nov 2017. Dostupné z: https://codeburst.io/nodejs-vs-ruby-on-rails-analysis-worth-doing-b1af8632c092

[9] Abadi, M.; Agarwal, A.; andEugene Brevdo, P. B.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Dostupné z: https://www.tensorflow.org/

[10] Kane, D.; Kane, D.: An Introduction to Using JWT Authentication in Rails. Jul 2016. Dostupné z: https://www.sitepoint.com/introduction-to-using-jwt-in-rails/

[11] Arias, D.: Hashing in Action: Understanding bcrypt. May 2018. Dostupné z: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/

[12] Kane, D.; Kane, D.: An Introduction to Using JWT Authentication in Rails. Jul 2016. Dostupné z: https://www.sitepoint.com/introduction-to-using-jwt-in-rails/

[13] Kulhan, J.: Normalizace relačních databází. Jul 2008. Dostupné z: http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/

[14] The Active Record Design Pattern. Dostupné z: http://researchhubs.com/post/computing/web-application/the-active-record-design-pattern.html

[15] Residentmario: Keras optimizers. Dec 2018. Dostupné z: https://www.kaggle.com/residentmario/keras-optimizers

[16] What is Heroku. Dostupné z: https://www.heroku.com/what

Příloha A

Seznam použitých zkratek

SOAP Simple Object Access Protocol

REST Representational State Transfer

JSON JavaScript Object Notation

XML eXtensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `EasyPacking.apk` ..... Instalační soubor aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS