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.

5. května 2019

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 5. května 2019 .....

Odkaz na tuto práci

Kabela, Jan. *EasyPacking*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato diplomová práce se zabývá aplikací EasyPacking pro OS Android. Tato aplikaci by měla napomáhat uživatelům nezapomenout si žádnou věc při cestách. V této aplikaci bude moci uživatel vytvářet seznamy věcí, které si na cestu bere. Zároveň bude tyto seznamy uživatel popisovat za pomoci cílové destinace, aktivit s cestou spojených, věku a pohlaví. Takto vytvořené seznamy budou ostatní uživatelé moci vyhledávat a předměty si sbalit. Jednotlivé předměty budou rozděleny podle místností, kde se obvykle v bytě nacházejí. Pro realizaci aplikace bude nutné vytvořit aplikační server. Na tomto serveru se budou zpracovávat data potřebná pro chod aplikace. Zároveň server bude 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.

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

Abstract

This diploma thesis's goal is to develop and aplication EasyPacking for OS Android. This application should help users with packing. In this aplication will users 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 destination, activities that users will be doing on there vacation, age and sex. Users will be able to search packing list based on these criteria. Each item in packing list will be devided into rooms in flat where thet can be usually found. It will be necessary to develop an aplication server for this aplication. This aplication server will be will be providing the aplication with data. Also it is necessary to create a databse for storing this data. For communication between aplication server and aplication will be used HTTPS protocol. The data will be transferred in JSON structure. Database will be created using PostgreSQL. Aplication will be developed in Kotlin. The result of this thesis will be tested aplication EasyPacking with aplication server and database.

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

Obsah

Seznam obrázků

Úvod

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

Přijde mi zbytečné vytvářet si pro každou cestu nový seznam. Je pravděpodobné, že pokud se rozhodnu pro jakoukoli cestu a s ní spojené aktivity, nebudu první, kdo takovou tu cestu podnikl. Rozhodl jsem se tedy pro vytvoření aplikace, do níž budou uživatelé moci seznamy ukládat 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 matematické modely pro řešení aplikace EasyPacking. Zaměřuji se zde na počet dat potřebných pro plnou funkčnost aplikace. Nalezené matematické modely následně používám pro zjištění počtu potřebných seznamů pro naplnění aplikace.

Dále jsem v této práci řešil možnosti, jak jednotlivé části implementovat a jaké technologie k tomu použít. Řešil jsem zde možnosti pro aplikační server, databázi a samotnou aplikaci. Zde jsou hodnoceny technologie a postupy, které se v dnešní době používají. Na základě jejich hodnocení a vhodnosti pro aplikaci jsem si jednotlivé technologie zvolil.

V neposlední řadě jsme popisoval samotnou implementaci jednotlivých částí. Tato podkapitola obsahuje návrh databáze zohledňující předchozí průzkum požadavků. Dále popis implementace aplikačního serveru v Ruby on Rails a vývoj aplikace v jazyku Kotlin. Spolu s popisem návrhu aplikace jsou zde

návrhy jednotlivých obrazovek aplikace.

Na konci práce je popsáno, jak byly jednotlivé části testovány. Pro testování aplikace jsou zde uvedeny testovací scénáře. Aplikační server byl testován pomocí testování chovaní. V závěru jsem uvedl možné další funcionality, které by mohly být do aplikace implementovány.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit aplikace pro OS Android s názvem EasyPacking. Tato aplikace by měla usnadnit uživatelům balení na cesty. Základ aplikace tvoří seznamy věcí, které uživatelé v rámci aplikace 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 vlastním seznamů, ale využívání již vytvořených. Jelikož se jednotlivé seznamy vážou k daným prázdninovým lokacím, u často navštěvovaných destinací je předpokládáno, že cíl opakovaného používání již vytvořených nastane dříve než u ne tolik oblíbených turistických míst.

Kromě dané cílové destinace budou jednotlivé seznamy blíže specifikovány pomocí, plánovaných aktivit, věku a pohlaví. Ohledně věku je jasné, že například v intervalu 10 let se seznam skoro lišit nebude, ale pokud budeme brát například rozpětí 30 let může nastat již velká změna. Pro co největší usnadnění balení budou seznamy přirazovat jednotlivé předměty k jejich pravděpodobným lokacím přímo v obydlí uživatele. V ideálním případě pak uživatel projde svůj domov právě jednou, v každé místnosti si sbalí potřebný předmět a bude připraven odcestovat.

Jedním z cílů je naleznout matematický model, který bude diskutovat funkcionalitu aplikace.

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

1. Vytvoření databáze

2. Vytvoření serveru napojeného na databázi s následující funkcionalitou

a) Uživatel
   i. Vytvoření účtu
   ii. Vytváření seznamů
   iii. Editace svých seznamů
   iv. Mazání svých seznamů
   v. Vyhledávání seznamů na základě věku, pohlaví, cílové destinace a aktivit

Kapitola 2

Analýza a návrh

2.1 Současná řešení

2.1.1 PackPoint travel packing list

V aplikaci uživatel na začátku zadá lokalitu kam cestuje, datum odjezdu, délku pobytu a typ cesty (dovolené nebo pracovní cesta), zda se jedná o dovolenou nebo pracovní cestu. Následně vybere aktivity, které plánuje provádět v cílové destinaci. Na výběr je 15 aktivit. Pokud uživatel použije premium verzi za 90 Kč, tak může zadat i vlastní aktivitu. Na základě tohoto výběru je uživateli vygenerován seznam doporučených věcí, které si s sebou má vzít. Uživatel postupně projde seznam a buď se rozhodne předmět vzít s sebou nebo pomocí posunutí na stranu předmět ze seznamu odstraní. V premium verzi může uživatel přidat i nové věci. Uživatel může takto upravený seznam sdílet pomocí e-mailu, sociálních sítí nebo cloudových úložišť.

Nevýhoda aplikace je, že uživatel musí vždy seznam vytvořit. Další nevýhoda 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 na začátku 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 věci jsou rozděleny do kategorií, jako například oblečení, hygiena nebo dokumenty. Uživatel se následně rozhodne, zda si daný předmět vzít s sebou nebo nikoliv. Je možné editovat jednotlivé věci, 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, kterým musí uživatel 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.

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. Vše si uživatel musí napsat sám. Jednotlivé seznamy není možné sdílet. Z uvedených aplikací je tato nejslabší.

2.1.4 Shrnutí současných řešení

Budu se věnovat pouze prvním dvou zmíněným aplikacím. V rámci komentářů uživatelů je hlavní nedostatek absence jiných jazyků než angličtiny. Z mého pohledu je také nedostatkem, že uživatel musí seznam pro každou novou destinaci vytvářet. Popřípadě 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 na aktivity. I přes tyto nedostatky jsou tyto aplikace jedny z nejlepších v této době. Z tohoto důvodu se velmi hodí pro inspiraci vytváření nové aplikace.

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 možnosti. Z tohoto údaje je pak možné odhadnout problémy, které by mohly nastat.

Kritéria můžeme rozepsat jako:

$K_1, K_2, K_3, K_4....K_n, J_1, J_2, J-3, ..., J_n$

Pro každé kritérium Ki bude zvolená konkrétní hodnota kritéria představována 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...r možných hodnot, pak použiji vzorec

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

Druhá skupina kritérií $J_1, J_2, J_3, ..., J_m$ představuje taková, kde z nabízené množiny hodnot J_i mohu vždy zvolit pouze jedinou, mám tedy přesně $|J_i|$ možností. Tak celkový počet možných kombinací lze zapsat jako:

$$[\binom{|K_i|}{1} + \binom{|K_i|}{2} + \binom{|K_i|}{3} + ... + \binom{|K_i|}{r}] * |J_1| * |J_2| * |J_3| * ... * |J_m|$$

Tímto vzorce jsem schopen zjistit jak velký počet seznamů je potřeba k obsáhnutí všech možných kombinací kritérií.

Je nutné ovšem počítat s tím, že některé zadané seznamy budou mít stejná kritéria a tím se zvýší počet potřebných seznamů pro obsažení všech kritérií. Tento problém popisuje Cupon collector problem.

Cupon collector problem se zabývá otázkou, kolik je potřeba zakoupit unikátních předmětů, abychom získali všechny. Příkladem může být zakupování sběratelských kartiček, které jsou prodávány v balíčkách po několika kartách. Karty jsou v těchto balíčcích náhodně umístěny. Může se stát, že při zakoupení dvou balíčků budou některé karty stejné. Řešením tohoto problému mohou sběratelé určit, kolik takovýchto balíčků budou muset koupit, aby získali celou kolekci. Jak vyplývá z tohoto zdroje [1]

Díky tomuto problému lze určit, kolik seznamů je skutečně potřeba pro naplnění všech kritérií. Pomocí počtu možných kombinací kritérií a použití aproximačního vzorce pro výpočet tohoto problému, který je následující,

$$E[X] = y(ln(y) + \Gamma) + 1/2$$

,se tento počet potřebných seznamů dozvím.

Do tohoto vzorce za y dosadím vypočtené číslo možných kombinací kritérií. Za Γ dosadím Euler-Mascherovu konstantu, která je rovna přibližně 0.577216.

Pro objasnění jak je možné, že některé seznamy budou stejné z pohledu matematiky, použiji Narozeninový paradox.

Narozeninový paradox se zabývá otázkou, při jakém počtu osob v místnosti je pravděpodobnost pro náhodné dvě osoby, aby měli narozeniny ve stejný den. Pokud například osob v místnosti je 23, tak pravděpodobnost, že tento jev nastane je rovna 50%. Tento paradox se využívá například v bezpečnosti. [2]

K získání pravděpodobnosti shody je použit opačný numerický proces. Nejprve vypočtu, jaká je pravděpodobnost, že všechny seznamy jsou odlišné. Následně zbylá pravděpodobnost do 100%, je pravděpodobnost pro shodu alespoň dvou seznamů. V této práci budu uvažovat jaká je pravděpodobnost, že shoda je 50%.

První je nutné vypočítat počet možných dvojic seznamů. Pro tento výpočet využiji následující vzorec.

$$z = s * (s–1)$$

kde s je počet seznamů. Dále vypočtu pravděpodobnost, že jeden pár je unikátní. Za pomoci následujícího vzorce

$$p(t) = c * (c–1)$$

Promněná c zde označuje počet všech možných kritérií. Šance, že všechny seznamy budou unikátní, je:

$$p(u) = p(t)^z$$

P(u) je pravděpodobnost, že všechny páry budou unikátní, jelikož se snažím zjistit počet nutných seznamů, pro určitou pravděpodobnost. P(u) se bude rovna této pravděpodobnosti. Jak jsem dříve zmínil tato pravděpodobnost se bude rovnat 50%. Znám tedy p(u) a mohu si také spočítat p(t). Z posledního vzorce spočítám z, z dosadím do prvního vzorce a vypočtu s. S pak označuje mnou hledaný počet seznamů, který je potřeba pro 50% pravděpodobnost, alespoň 2 stejných seznamů.

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.

Můžu počítat, že na světě je přibližně 200 států. Dále pro utvoření představy o počtu potřebných seznamů uvažujme 50 možných aktivit, které uživatelé aplikace budou provozovat na svých cestách. Posledními dvěma kritérii je věk a pohlaví. Pohlaví tvoří dvě možnosti. Co se týče věku, aplikace vyhledává v rozsahu +-5 let od uživatelem zadaného věku. Pokud budu uvažovat uživatele v rozpětí věku 10 – 60 let. Tedy uživatele co již cestují a používají chytrý telefon. Mělo by postačit díky intervalu +-5 let 6 záznamů, které budou rovnoměrně rozmístěny v intervalu 10-60 let.

Z předešlé úvahy vychází následující počty jednotlivých kritérií

- 50 aktivit
- 200 států
- 2 pohlaví
- 6 odlišných věkových kategorií

Není pravděpodobné, že by uživatel zaškrtl všechny aktivity. Budu počítat s tím, že rozumné číslo pro maximální počet aktivit pro jednu cestu je 4. Takto se dostaneme, pomocí dosazení do dříve zmiňovaného vzorce na

$$[\binom{50}{1} + \binom{50}{2} + \binom{50}{3} + \binom{50}{4}] * 200 * 2 * 6 = 602820000$$

Dále se použiji Cupon collector problem pro zjištění počtu potřebných seznamů, tak aby byly všechny kritéria obsáhnuta. Dosadím do předešlého vzorce.

$$602820000 * (ln(602820000) + 0.577216) + 1/2 = 1.2535247 * 10^{10}$$

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

Jak je vidět z výsledku výpočtu, takovýto počet záznamů je skoro nemožné uchovávat v databázi. Zároveň vyhledávání v takovém to objemu dat by bylo opravdu pomalé. Je nutné omezit počet těch největších kriterií. Tím je jak destinace, tak aktivity.

Ohledně destinací můžu počítat s tím, že některé destinace nejsou navštěvovány, například s ohledem na situaci v těchto zemích. Dále ne v každé destinaci lze provádět všech 50 aktivit. Ovšem ani tento předpoklad nesníží počet seznamů dostatečně.

Můžu ovšem říci, že nějaké seznamy budou velice podobné popřípadě dokonce shodné. Příkladem může být lyžování v Alpách nebo dovolená u moře v jižní Evropě. Je tedy zbytečné uchovávat takto shodné seznamy. Bylo by tedy výhodné vytvořit "slovník"společných destinací a aktivit, kde je předpoklad, že seznamy budou stejné. Na základě tohoto omezit počet podobných seznamů.

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, ale stále je zajímavé zanalyzovat možnosti.

2.3.1 Databáze

První bych se zaměřil na řešení databáze 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 mnoho 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. [3].

V aplikace 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 klade na databázi, splňují relační databáze. Jak je popsáno v tomto zdroji [4] Pro rychlé načítání z databáze je nutné použít indexy.

Důvod pro vytvoření indexů na dané tabulce v databázi je zrychlení prohledávání tabulky a nalezní 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, následně vloží nový záznam do každého indexu v tabulce. Vložení indexu musí být na správné místo [5].

Hlavní funkcí aplikace je vyhledávání seznamů. Ohledně vkládání a mazání

záznamů a s tím spojená práce s indexy není tak důležitá. Důvodem nedůležitosti vkládání je, že uživatel bude spíše vyhledávat seznamy v aplikaci, než je ukládat.

Ohledně distribuce databáze, toto rozhodnutí nechám až po analýze serverové části. Pro různé 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 nezávislí na platformě i jazyku, REST umí používat pouze HTTP. U SOAP je požadavek i odpověď zabalená 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. [6]

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 menšími 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. [7]

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. [7]

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.

2.3.2.3 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 nároční 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 nějakými problémy. S tím také souvisí menší počet řešení problému na internetových fórech nebo blocích. Toto ovšem netvoří 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.

2.3.2.4 Zabezpečení

Je důležité kontrolovat přístup uživatelů na server a to ať už ověřením, zda je uživatel přihlášen a jím zaslaný požadavek na server má být zpracován, tak ochrana proti útokům na databázi a to zejména útoky typu SQL Injection.

2.3.2.5 Autentikace

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

JWTs carry information (called "claims") via JSON, hence the name JSON Web Tokens. JWT is a standard and has been implemented in almost all popular programming languages. Hence, they can be easily used or exchanged in systems implemented in diverse platforms. JWTs are comprised of plain strings, so they can be easily exchanged in a URL or a HTTP header. They are also self-contained and carry information such as payload and signatures.[8]

JsonWebToken je text, který se dá poslat v hlavičce HTML. Samotný token se pak skládá ze tří částí hlavička, obsah a podpis. Hlavička obsahuje typ tokenu, tady je to JWT a hashovací algoritmus, který byl použit. 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. [8]

Tuto možnost jsem zvolil kvůli její jednoduchosti použití. Ale zároveň je velice jednoduchá na implementaci v Ruby on Rails. Ovšem mnoho ostatních jazyků tuto metodu autentikace podporuje a ani tam není složité ji implementovat. Jako hashovací algoritmus jsem použil Bcrypt.

bcrypt was designed by Niels Provos and David Mazières based on the Blowfish cipher: b for Blowfish and crypt for the name of the hashing function used by the UNIX password system.[9]

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

Hlavicka eyJhbGciOiJIUzI1NiJ9

Obsah eyJ1c2VyX2lkIjoxLCJleHAiOjE1NTQ3MzMwMDB9

Podpis VqJRuX7x6b9ExKkGNyLcSdD6zQaJ_cy-Iz6tz2p3rRk

2.3.2.6 SQL Injection

SQL injection is a code injection technique that might destroy your database. SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.[10]

Co se týče této práce, nejedná se o webovou stránku, ale o aplikaci. Ovšem ať už řešíme webovou stránku, aplikaci nebo jakýkoliv frontend, který komunikuje se serverem potažmo s databází. A to na základě uživatelského inputu. Je nutné tento problém nepřehlížet, ale zamyslet se zda nemůže nastat.

Příkladem může být:

”Select * FROM Users WHERE UserId =”

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

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

Ohledně této práce problém může nastat pouze při přihlašování nebo registraci uživatele. Jelikož je to jediné místo, kde uživatel zadává text, který je následně kontrolován s databází nebo propsán do databáze.

Možné řešení je používat SQL parametry. Kde pokud například zjišťuji zda uživatel s daným id existuje v databázi 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.[10]

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]]”

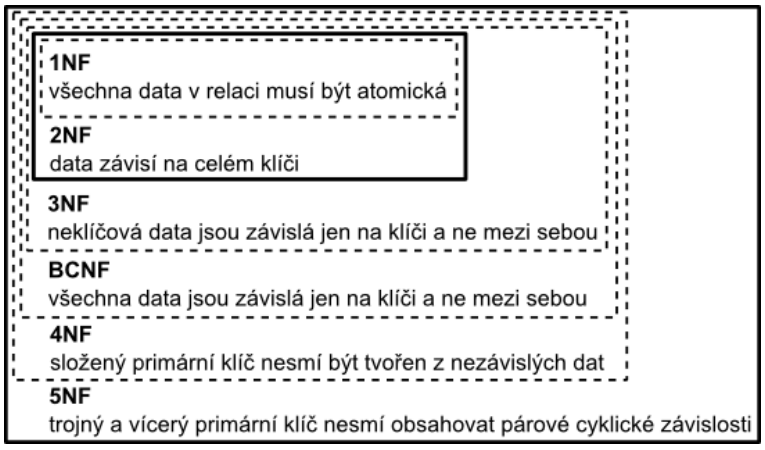
Kapitola 3

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. Normálních forem je celkem 6.

„*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.*“ [11]



Obrázek 3.1: Normální formy.

Z tohoto článku a z 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 deromlizace jsou složitá, jelikož je nutné zamyslet se, nad dopadem zrychlení výběru dat nad jejich vkládáním. Zároveň je možné, že dojde k redundanci dat.

Ale 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. Následně použiji příklad týkající se databáze, která je součástí této diplomové práce.

Pokud chci ukládat překlady k jednotlivým záznamům, musím u nich uvádět, kterého jazyka se týkají. Jazyk dokáži definovat pomocí tří znaků. Ovšem všechny záznamy v jednom jazyce budou mít stejné tyto 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 to podívám z hlediska rychlosti výběrů, vkládání a velikosti uložených dat. Tak mohu diskutovat tyto zlepšení. Pro uložení cizího klíče, který bude u každého záznamu budu používat integer, který má velikost je 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 - User
- Seznamy - PackingList
- Předměty – Item
- Lokality předmětů - Locations
- Destinace - Destination
- Překlad aktivit - Translation_Criterion
- Překlad předmětů – Translation_Item
- Překlad lokalit – Translation_Location
- Překlad destinací – Translation_Destination

Atributy tabulky Uživatel

- Primární klíč – id – bigint

- Heslo – password – varchar
- e-mailová adresa – email – varchar
- rozpoznání mezi administrátorem a běžným uživatelem – role – varchar

Atributy tabulky seznam

- Primární klíč – id – bigint
- Uživatelské hodnocení – rating – integer
- Pohlaví – sex – varchar
- Věk – age – integer

Atributy tabulky destinace

- Primární klíč – id – bigint

Atributy tabulky předmět

- Primární klíč – id – bigint

Atributy tabulky lokality předmětů

- Primární klíč – id – bigint

Atributy tabulky aktivit

- Primární klíč – id – bigint

Atributy tabulky překlad aktivit

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

Atributy tabulky překlad předmětů

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

Atributy tabulky překlad lokalit

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

- Překlad – name - varchar

Atributy tabulky překlad destinací

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

3.1.2 Relace

Propojení jednotlivých tabulek bude následující.

- Uživatel - seznam

  Uživatel má své seznamy, které vytvořil. Uživatel může mít více seznamů, proto spojení Uživatele a seznamu bude relací n:1. Kvůli tomu bude tabulka seznamů obsahovat cizí klíč uživatele.

- 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_Criterion. Tato tabulka bude obsahovat dva atributy. První bude cizí klíč daného seznamu. Druhý atribut bude cizí klíč aktivity.

- Aktivita - překlad aktivity

  Každá aktivita bude spojena se svými překlady do různých jazyků. Každá aktivita bude mít více překladů. To znamená spojení 1:n. Proto bude do tabulky překlad aktivity vložen cizí klíč z tabulky aktivita.

- 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.

- Destinace - překlad destinace

  Destinace má také svůj překlad, stejně jako bylo u aktivity. Bude stejně jako tam přidán cizí klíč destinace do tabulky překlad destinace.

- Seznam - předmět

  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 předmět.

- Předmět - překlad předmětu

  Předměty musí být navázány na svoje překlady. Navázání bude realizovat relace 1:n. Bude tedy opět přidán cizí klíč z tabulky předmět do tabulky překlad předmětů.

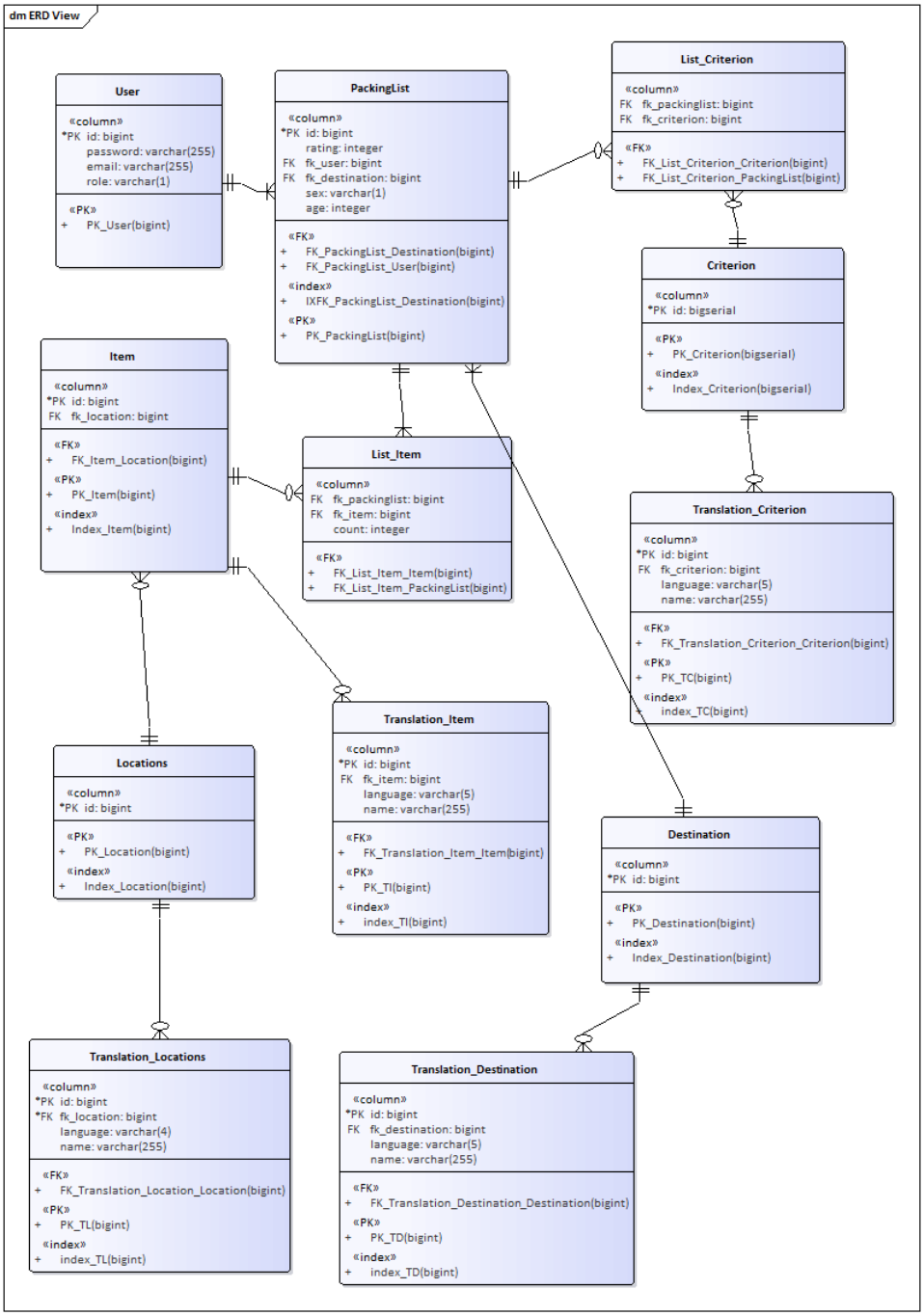
- Předmět - lokalita

  Jak bylo v analýze zmíněno, každý předmět má svou lokalizaci, kde se nachází. Z tohoto důvodu je nutná relace mezi předmětem a lokací. Na jedné lokaci se může nacházet více předmětů. Proto se tato relace bude typu 1:n. Bude tedy přidán cizí klíč z tabulky lokace do tabulky předmět.

- Lokalita - překlad lokality

  Poslední z relací bude relace mezi lokalitami a překladem lokalit. Opět jako u předchozích případů se bude jednat o relaci 1:n. Což zařídí přidání cizího klíče z tabulky lokace do tabulky překlad lokace.

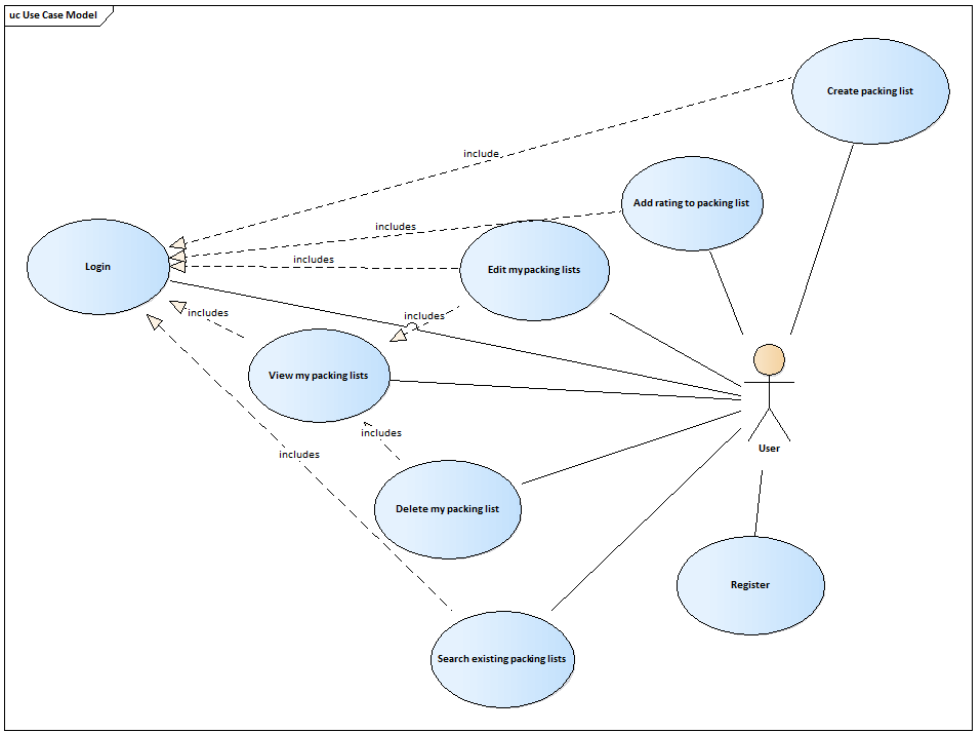
3.1.3 Schéma



Obrázek 3.2: Schéma databáze.

3.2 Diagramy

3.2.1 UseCase



Obrázek 3.3: Use case.

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

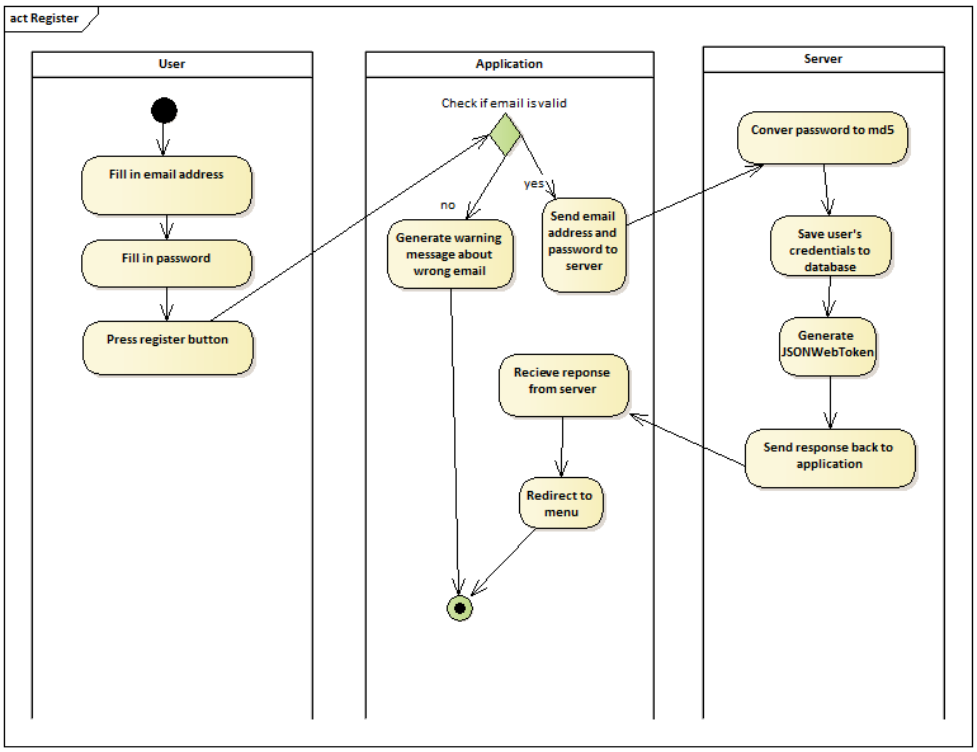
- Registrace
- Přihlášení
- Vytvoření nového seznamu
- Vyhledávání seznamů
- Přidání hodnocení k seznamu
- Zobrazení svých seznamů
- Editovat své seznamy
- Mazat své seznamy
- Zobrazit si své seznamy

3.2.2 Popis jednotlivých aktivit

3.2.2.1 Registrace

Při otevření aplikace se uživatel dostane na obrazovku přihlášení. Zde vyplní email. Dále zadá své heslo. Nakonec stiskne tlačítko registrovat. Aplikace zkontroluje, zda je email ve správném tvaru, popřípadě neobsahuje žádné speciální znaky.

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



Obrázek 3.4: Activity diagram registrace.

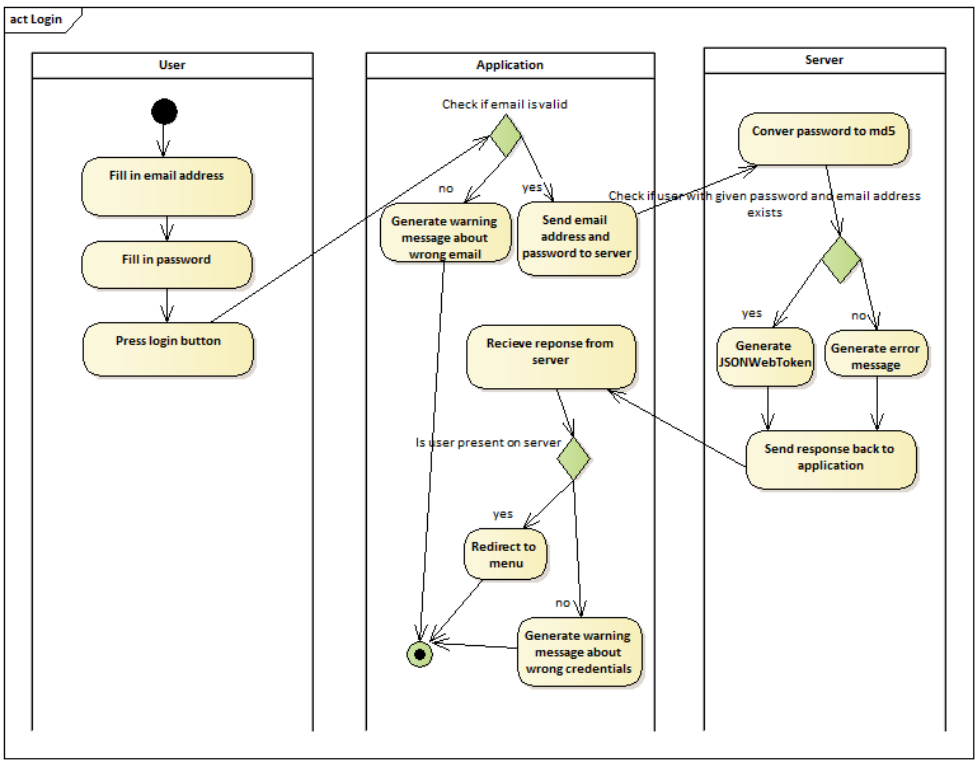
3.2.2.2 Přihlášení

Při otevření aplikace se uživatel dostane na obrazovku přihlášení. Zde vyplní email, kterým se dříve registroval. Dále zadá své heslo. Nakonec stiskne

tlačítko přihlášení. Aplikace zkontroluje, zda je email ve správném tvaru, popřípadě neobsahuje žádné speciální znaky.

Aplikace pošle požadavek na server s uživatelem 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 authorizaci 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 3.5: Activity diagram přihlášení.

3.2.2.3 Menu

V menu je seznam možných aktivit v aplikaci. Vyhledat seznam, vytvořit seznam, zobrazit mé seznamy a zobrazit informace o aplikaci.

3.2.2.4 Vytvoření nového seznamu

Uživatel se přihlásí nebo registruje. Následně v aktivitě menu zvolí Vytvořit nový.

Zobrazení aktivit Aplikace zašle požadavek na obdržení všech aktivit, které může uživatel na své cestě uskutečnit. V rámci tohoto požadavku je připojen JSONWebToken, který dříve obdržel. Pokud daný JSONWebToken na serveru existuje a je stále validní, server vybere všechny aktivity z databáze a zašle je zpět do aplikace. Pokud JSONWebToken neexistuje nebo není již validní, server zašle informaci o nepovedeném ověření.

Pokud aplikace obdržela seznam aktivit, přesměruje uživatele na zobrazení aktivit. V případě, že se ověření nezdařilo, je uživateli zobrazena zpráva o chybě a následně je přesměrován na aktivitu přihlášení.

V seznamu aktivit uživatel vybere všechny aktivity spojené s jeho cestou. Pak svůj výběr odsouhlasí stisknutím tlačítka ok. Aplikace si zapamatuje aktivity, které uživatel zvolil. Následně je přesměrován na zobrazení destinací.

Zobrazení destinací Aplikace zašle požadavek na server ohledně získání všech destinací. Tento požadavek opět obsahuje JSONWebToken pro ověření uživatele. Pokud je JSONWebToken validní, server vybere z databáze všechny destinace a zašle je zpět do aplikace.

V opačném případě zašle zprávu o chybném ověření. Pokud ověření proběhlo v pořádku a aplikace obdržela seznam destinací, je uživatel přesměrován na aktivitu zobrazení destinací. V případě opačném se uživateli zobrazí zpráva o neúspěšném ověření a je přesměrován na aktivitu přihlášení.

V aktivitě zobrazení destinací si uživatel vybere jednu ze zobrazených destinací, podle toho, kam cestuje. Svůj výběr potvrdí stisknutím tlačítka Ok. Aplikace si jeho výběr přidá k zapamatovaným aktivitám. Následně je uživatel přesměrován na aktivitu zadání doplňujících informací.

Doplňující informace Při zadávání doplňujících informací uživatel vypní svůj aktuální věk a vybere zda je muž nebo žena. Po vyplnění těchto informací uživatel stiskne tlačítko Ok. Aplikace si zadané údaje přidá k již zapamatovaným aktivitám a destinaci. Uživatel je přesměrován na zobrazení místností.

Zobrazení místností Aplikace se dotáže na server pro získání všech místností. Požadavek opět obsahuje JSONWebToken. Pokud je JSONWebToken validní, server vybere všechny místnosti z databáze a zašle je zpět do aplikace. V opačném případě zašle zprávu o chybném JSONWebTokenu.

V případě, že ověření bylo úspěšné a aplikace získala seznam místností, je uživatel přesměrován na aktivitu zobrazení místností. Pokud ověření úspěšné nebylo, je uživateli zobrazena chybová zpráva o neúspěšném ověření a je uživatel přesměrován na aktivitu přihlášení.

Zobrazení předmětů Při zobrazení místností uživatel projde jednotlivé místnosti. Po kliknutí na danou místnost se aplikace dotáže na server pro získání všech předmětů, které jsou s danou místností svázány. Opět je spolu s požadavkem zaslán JSONWebToken pro ověření. Pokud je JSONWebToken správný, server vybere z databáze všechny předměty svázané s danou

místností a odešle je zpět do aplikace. V opačném případě je zpět odeslána chybová zpráva.

V případě úspěšného ověření je uživatel přesměrován na aktivitu vkládání předmětů. V opačném případě se uživateli zobrazí chybové upozornění a je přesměrován na aktivitu přihlášení.

U zobrazení předmětů jsou uživateli zobrazeny všechny předměty z dané místnosti spolu s číslem označujícím počet daných předmětů potřebných pro cestu. Pokud uživatel předmět nepotřebuje, nic nevyplňuje a ponechá výchozí hodnotu 0. Pokud uživatel tuto místnost již navštívil, aplikace si dříve vyplněné údaje pamatuje a při načítání předmětů změní výchozí hodnotu počtu na jím již vyplněnou.

Vyplnění celého seznamu Uživatel projde všechny místnosti a postupně si přidá do seznamu všechny předměty potřebné pro cestu. Když je uživatel spokojen se všemi jím přidanými předměty, stiskne tlačítko Save list.

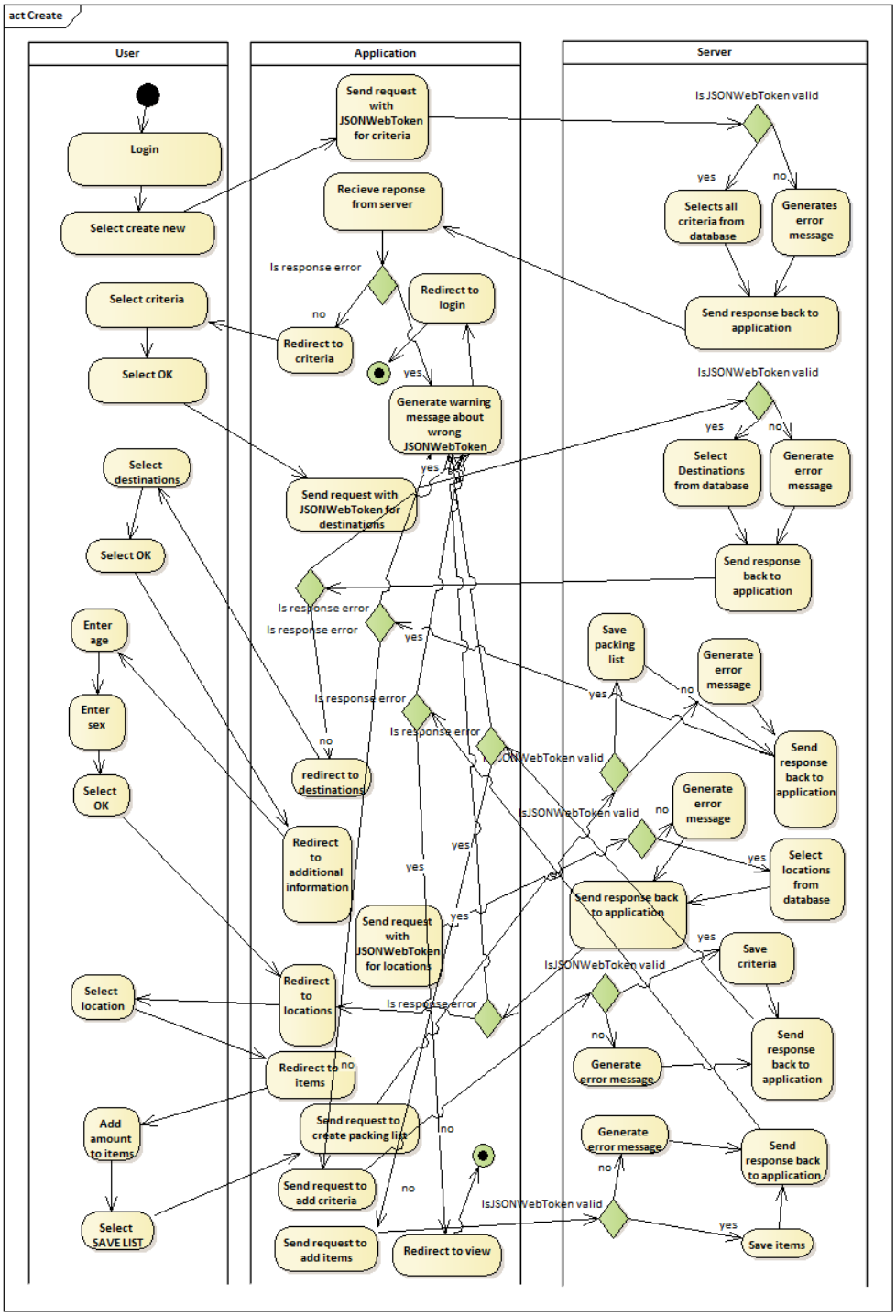
Následně pak aplikace zašle požadavek na získání id uživatele. Ten se na serveru dekóduje z jeho JSONWebTokenu a zašle se zpět do aplikace. Poté aplikace zašle požadavek na vytvoření seznamu včetně id uživatele, pohlavím a věku uživatele, id destinace a JSONWebTokenu. Na serveru se vytvoří daný seznam a do aplikace je zasláno id takto vytvořeného seznamu.

Dalším krokem je přidání všech aktivit k seznamu. Zde aplikace zašle na server požadavek pro přidání aktivit s id seznamu, id všech zvolených aktivit a JSONWebTokenu. Server podle id seznamu přidá všechny obdržené aktivity.

Nakonec pak aplikace pošle požadavek na přidání všech předmětů s id seznamu, id všech předmětů a JSONWebTokenem. Server opět pomocí id seznamu přidá jednotlivé předměty.

Při každém požadavku se opět ověřuje JSONWebToken stejně, jak tomu bylo u předchozích požadavků. Pokud není platný je uživatel přesměrován na aktivitu přihlášení.

Po úspěšném uložení všech částí seznamu je uživatel přesměrován na zobrazení jím vytvořeného seznamu.



Obrázek 3.6: Activity diagram Vytvoření nového seznamu.

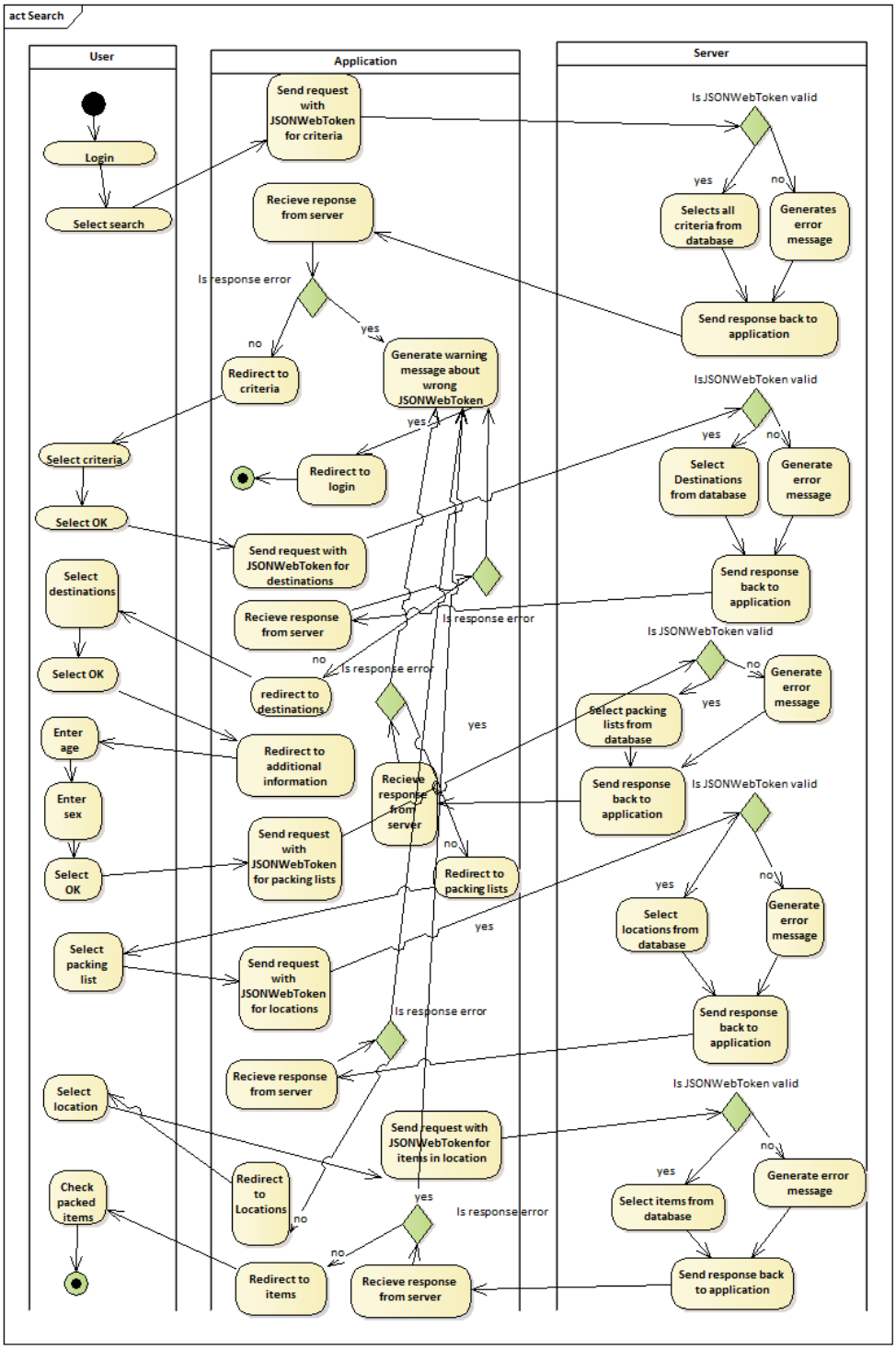
3.2.2.5 Vyhledávání

Vyhledávání již existujícího seznamu je velice podobné vytváření nového seznamu. Uživatel se přihlásí nebo registruje. Následně zvolí v menu Vyhledat. Poté, stejně jako při vytváření, zadá aktivity spojené s cestou a cílovou destinaci. Dále vyplní doplňující informace.

Zobrazení seznamů Aplikace následně zašle požadavek na získání seznamů dle zadaných informací. Server z databáze vybere všechny seznamy splňující kritéria. Akorát věk si server upraví na interval zadaný věk +-5 let. Takto nalezené seznamy seřadí podle jejich hodnocení a odešle je zpět.

Opět server kontroluje, zda má uživatel platný JSONWebToken. Pokud tomu tak není je s chybou přesměrován na přihlášení. V opačném případě se mu zobrazí nalezené seznamy.

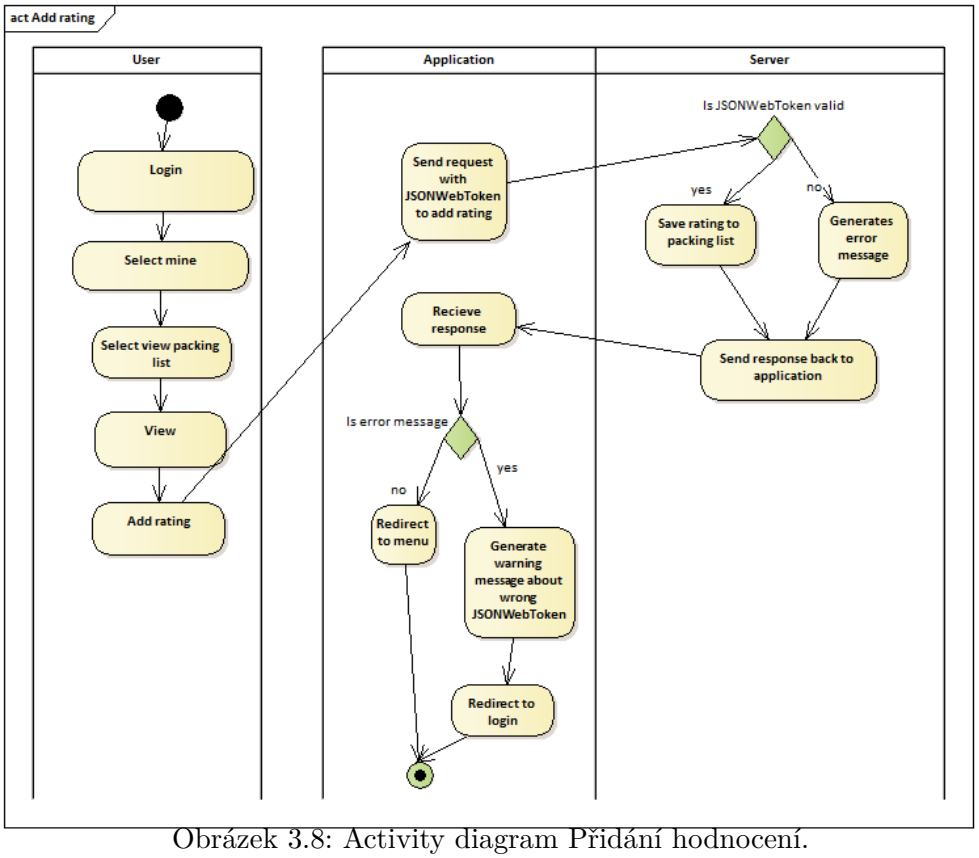
Uživatel si zvolí jeden z uvedených seznamů. Stejně jako při vytváření je přesměrován na seznam místností, pod kterými nalezne předměty. Které se tam nacházejí. Rozdíl oproti aktivitě vytváření je v tom, že se při rozkliknutí zobrazí pouze předměty, jenž jsou v seznamu přidané a ne všechny předměty, které se v místnosti nacházejí. Další rozdíl je, že předměty nelze editovat, ale pouze označit jako již sbalené. Aplikace si opět pamatuje, jaké předměty uživatel již označil jako sbalené a při opětovném navštívení místnosti se mu dané předměty označí.



Obrázek 3.7: Activity diagram Vytvoření nového seznamu.

3.2.2.6 Přidání hodnocení

Po dokončení sbalení všech předmětů v aktivitě vyhledávání. Je uživatel dotázán na zvolení hodnocení daného seznamu. Uživatel vybere hodnocení od 1 do 5 a odsouhlasí ho. Aplikace pak následně zašle požadavek na přidání hodnocení. Server přidá hodnocení k danému seznamu a odešle odpověď o úspěchu. Následně je uživatel přesměrován do menu.



Obrázek 3.8: Activity diagram Přidání hodnocení.

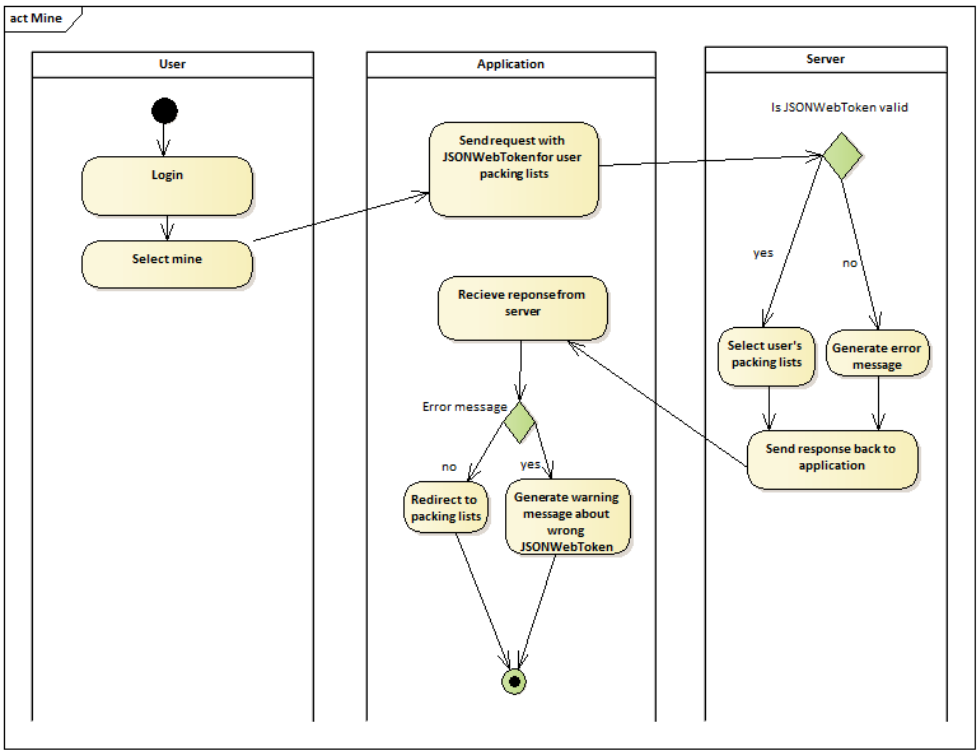
3.2.2.7 Správa vlastních seznamů

Seznamy jsou uložené s návazností na uživatele, který je vytvořil. Tím pádem má vlastník možnost své seznamy spravovat.

3.2.2.8 Zobrazení vlastních seznamů

Pokud uživatel po přihlášení zvolí v menu tlačítko Mé, aplikace odešle požadavek na server k získání všech seznamů, které kdy daný uživatel vytvořil. Server vyhledá v databázi seznamy, které tomuto vyhovují, a odešle je zpět do aplikace. Opět zde probíhá kontrola JSONWebTokenu. Pokud je úspěšná, jsou

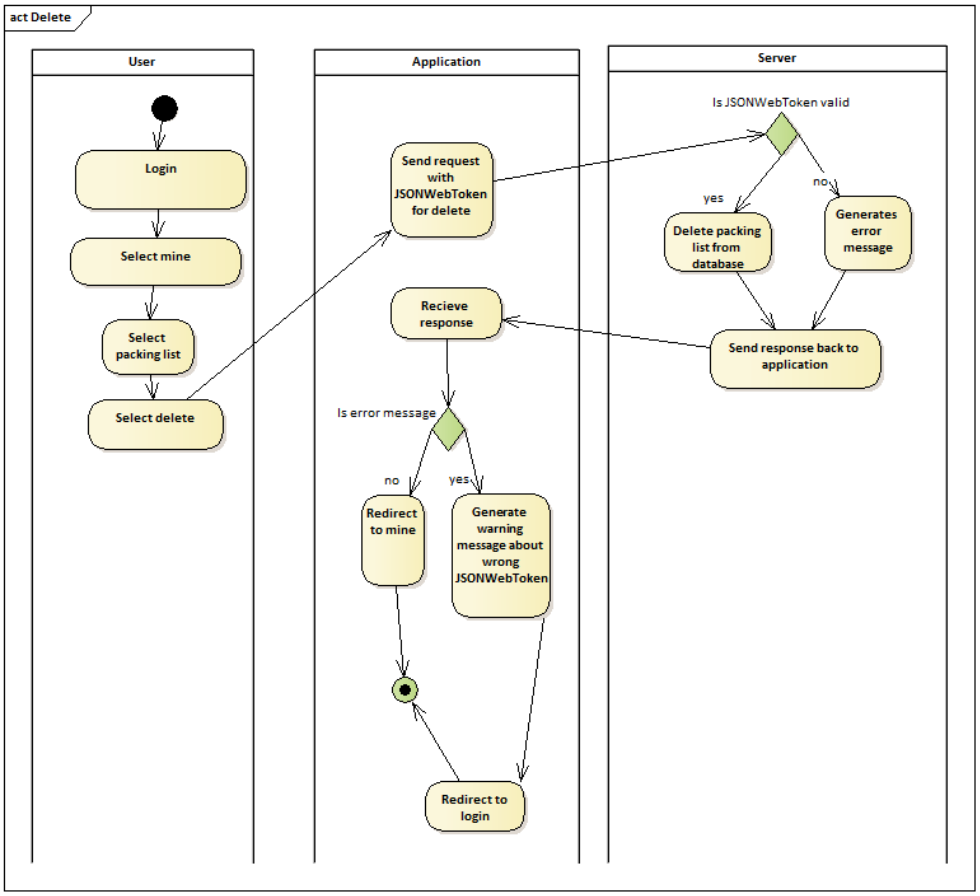
uživateli zobrazeny všechny jeho seznamy. V opačném případě je odkázán na přihlášení.



Obrázek 3.9: Activity diagram zobrazení vlastních seznamů.

3.2.2.9 Smazání

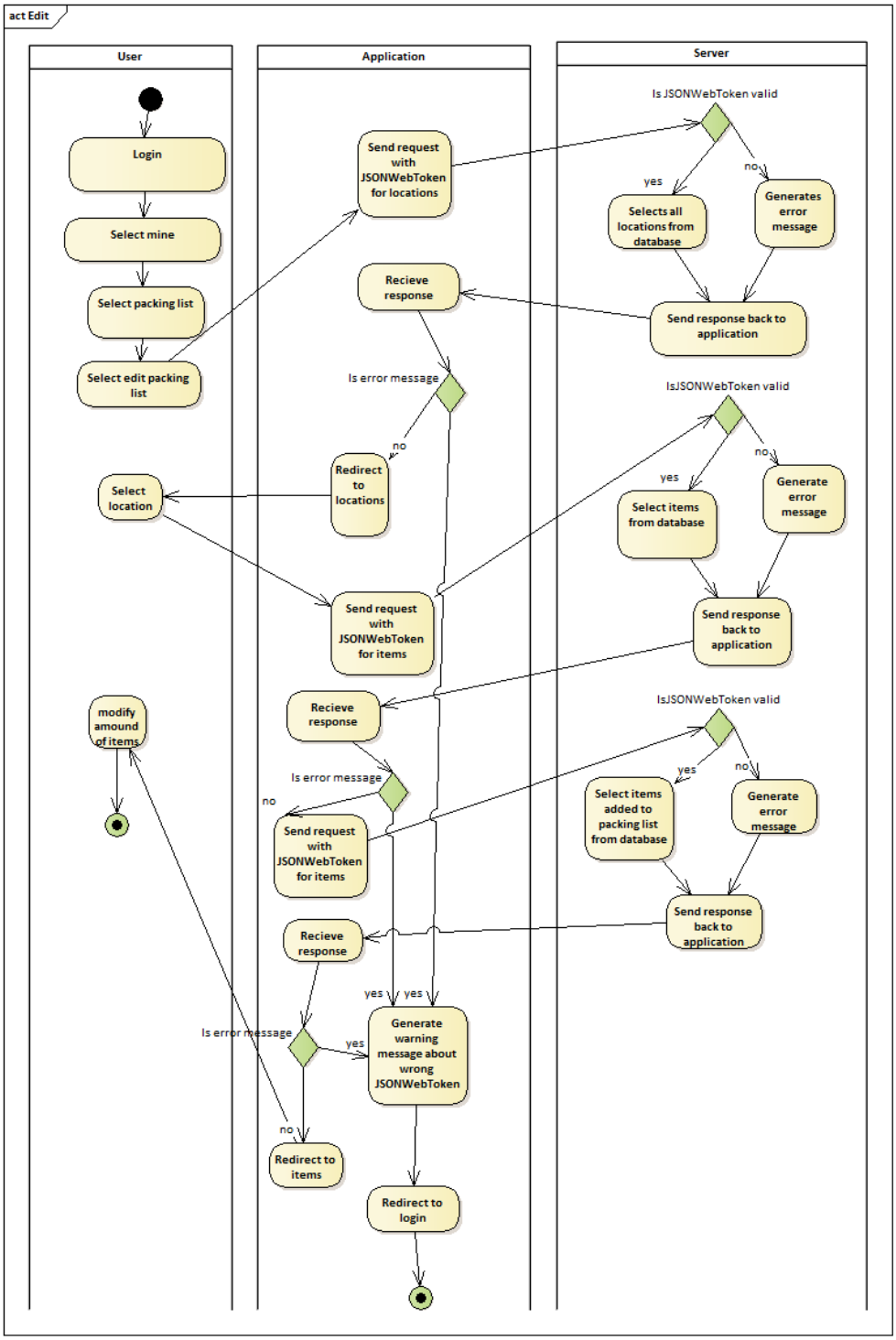
Po přihlášení, zvolení zobrazení svých seznamů, vybrání některého ze seznamů a kliknutí smazání v menu. Je odeslán požadavek na server pro smazání daného seznamu. Server daný seznam smaže a odešle odpověď o úspěchu. Probíhá zde kontrola JSONWebTokenu. Při neúspěchu je uživatel odkázán na přihlášení. Po úspěšném smazání aplikace přesměruje uživatele zpět do aktivity svých seznamů.



Obrázek 3.10: Activity diagram smazání.

3.2.2.10 Editace

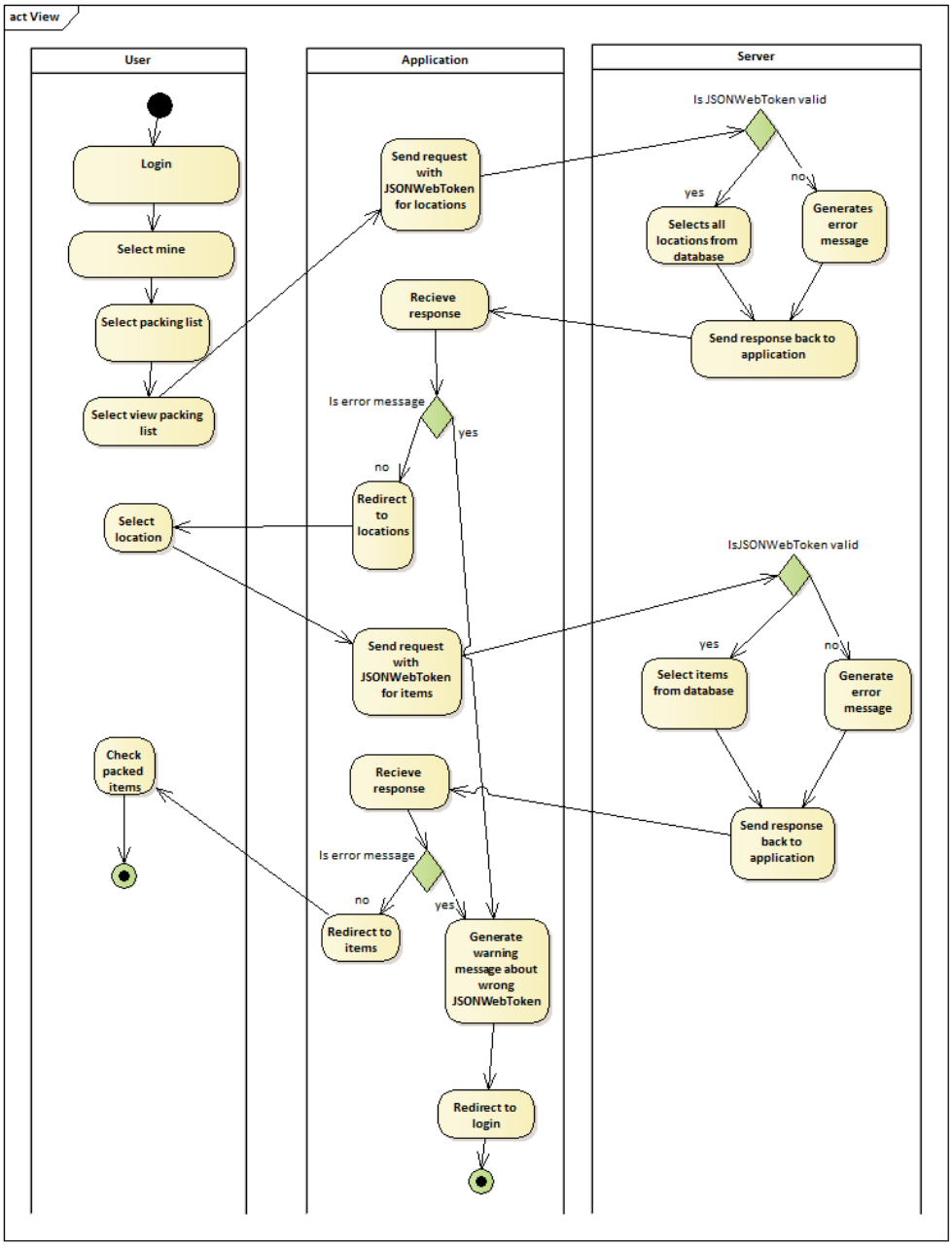
Uživatel se přihlásí, nechá si zobrazit své seznamy, zvolí seznam, který chce editovat, a klikne editovat. Následně je aplikací přesměrován do aktivity zobrazení místností se stejnou funkcionalitou, jako při vytváření nového seznamu. Ovšem rovnou si aplikace od serveru vyžádá již přidané předměty, takže po navštívení místnosti nalezne uživatel u již přidaných předmětů jejich počet a nikoliv 0. Postupně zedituje všechny předměty, které si přeje. Nakonec změny odsouhlasí kliknutím uložit seznam. Jako při vytváření je poté přesměrován na zobrazení daného seznamu. Opět zde probíhá kontrola pomocí JSONWebTokenu.



Obrázek 3.11: Activity diagram editace.

3.2.2.11 Zobrazení

Je možné pro uživatele si pouze zobrazit některý ze svých seznamů. Zejména, aby nemusel procházet celou aktivitou vyhledat, ale mohl rovnou použít svůj seznam. Pokud tedy namísto smazání či editace uživatel zvolí zobrazit. Dostane se do aktivity zobrazení místností, která je shodná s aktivitou zobrazení místností při vyhledávání a následném zobrazení předmětů, kde uživatel zaklikává již sbalené předměty. Opět v jednotlivých krocích probíhá kontrola JSONWebTokenu.



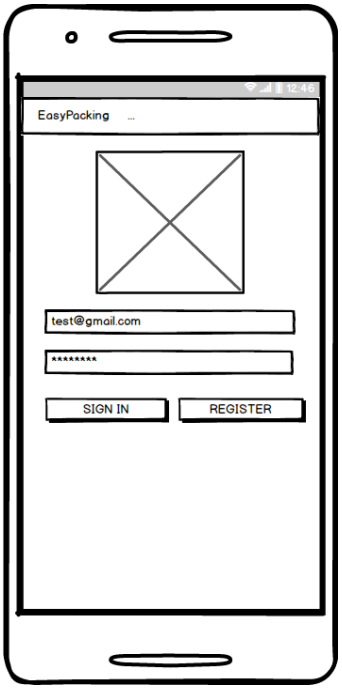
Obrázek 3.12: Activity diagram zobrazení.

3.3 MockUps

Každá obrazovka obsahuje záhlaví. Záhlaví obsahuje název aplikace. Jak bude vidět později kromě obrazovky pro přihlášení/registrace obsahuje navíc ještě tlačítko menu. Toto tlačítko odkazuje na obrazovku menu. Tlačítko tam je proto, aby se uživatel mohl z jakékoliv aktivity dostat zpět na obrazovku menu a tím započít jinou aktivitu.

3.3.1 Přihlášení

Obrazovka přihlášení obsahuje logo aplikace. Pod tímto 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. Jejich zobrazení závisí na tom, zda je uživatel již registrován nebo ne.

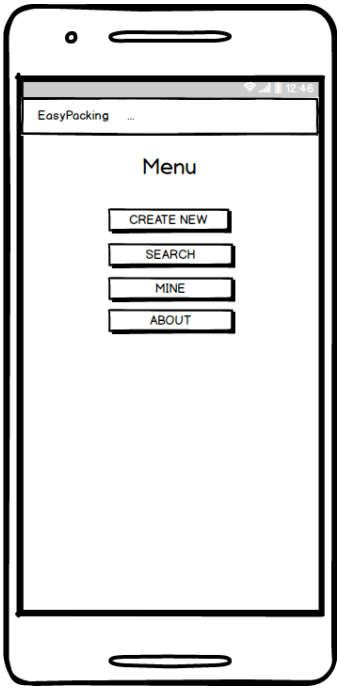


Obrázek 3.13: Mockup přihlášení.

3.3.2 Menu

Menu je hlavním rozcestníkem aplikace. Zde se uživatel rozhoduje za jakým účelem do aplikace přišel. Pro jasnost aktivity je tu nadpis Menu. Dále se menu skládá ze čtyř tlačítek. První je Vytvořit nový seznam. Následuje Vyhledat

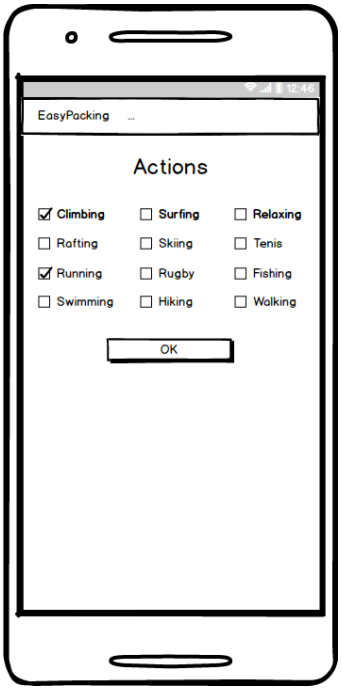
seznam. V neposlední řadě zde uživatel najde Moje seznamy. V případě, že by se uživatel chtěl dozvědět něco o aplikaci tak je zde tlačítko O aplikaci.



Obrázek 3.14: Mockup menu.

3.3.3 Aktivity

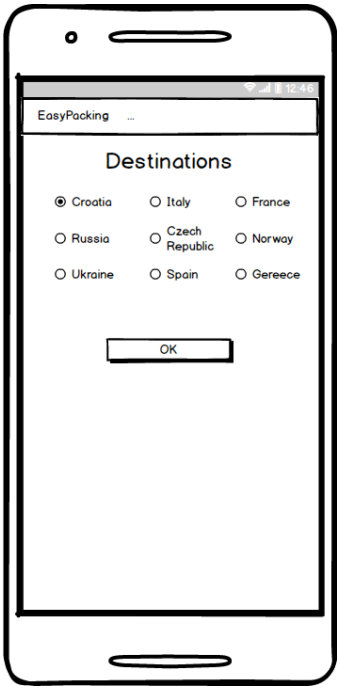
Zde se uživatel rozhoduje, jaké činnosti bude v rámci své cesty podnikat. Ilustrativně je zde zobrazeno 16 činností. Pole s činnostmi bude možné posouvat. To hlavně proto, že činností bude pravděpodobně více než se na obrazovku vejde. Každá činnost je opatřena zaškrtávacím polem s popiskem činnosti. Nakonec tato obrazovka obsahuje tlačítko Ok pro potvrzení výběru.



Obrázek 3.15: Mockup aktivity.

3.3.4 Destinace

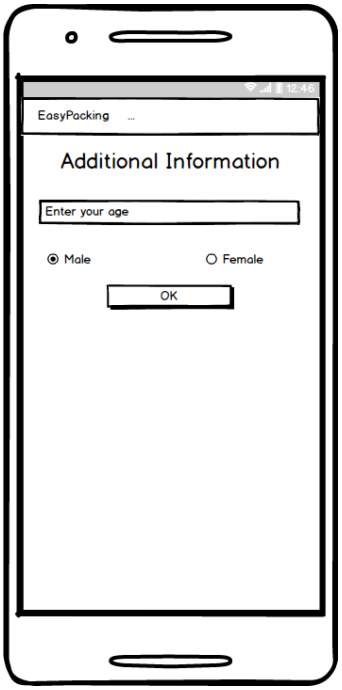
Zde uživatel zadá do jaké destinace bude cestovat. Tato obrazovka se podobá obrazovce Aktivity. Rozdíl je v tom, že namísto zaškrtávacích polí, může uživatel zvolit pouze jednu destinaci. Opět bude pole s destinacemi posouvací, jelikož nyní je ve světě okolo 200 zemí.



Obrázek 3.16: Mockup destinace.

3.3.5 Doplňující informace

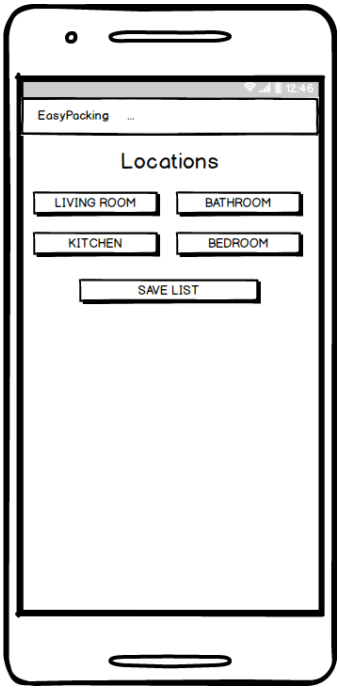
Pro přesnější vyhledávání je nutné, aby uživatel zadal svůj věk a pohlaví. Proto na této obrazovce je jak pole pro vyplnění věku, jenž je omezené pouze na číselný vstup, tak dva přepínače a to buď muž nebo žena. Opět je zde tlačítko pro potvrzení výběru.



Obrázek 3.17: Mockup doplňující informace.

3.3.6 Místnosti nový seznam

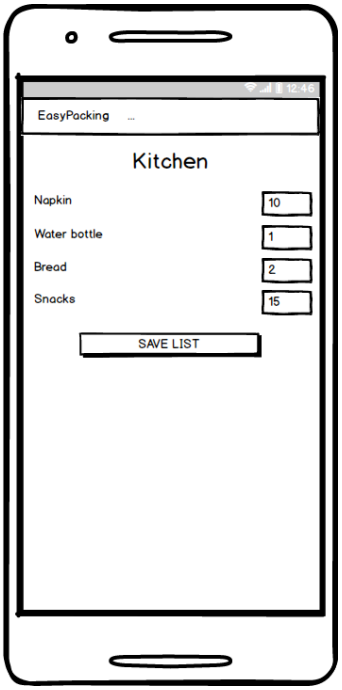
Zde již uživatel nalezne jednotlivé místnosti, pod kterými se ukrývají jednotlivé předměty. Jsou zde tedy místnosti v podobě tlačítka ve dvou sloupcích. A následně tlačítko pro uložení seznamu.



Obrázek 3.18: Mockup Místnosti nový seznam.

3.3.7 Předměty nový seznam

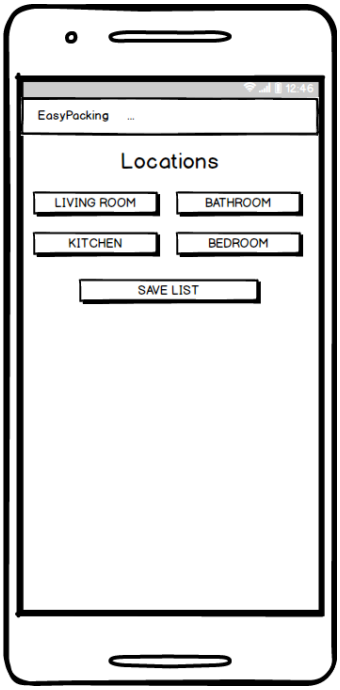
V této obrazovce uživatel přidává do seznamu věci z určité místnosti, které si bude brát s sebou. Je zde vidět seznam všech věcí, které si uživatel může sbalit. Vedle každého předmětu je pole pro zadání počtu jednotlivých předmětů. Toto pole bude na začátku inicializované na 0. Nakonec je zde potvrzovací tlačítko. Opět je pole s předměty posuvné.



Obrázek 3.19: Mockup předměty nový seznam.

3.3.8 Místnosti zobrazit

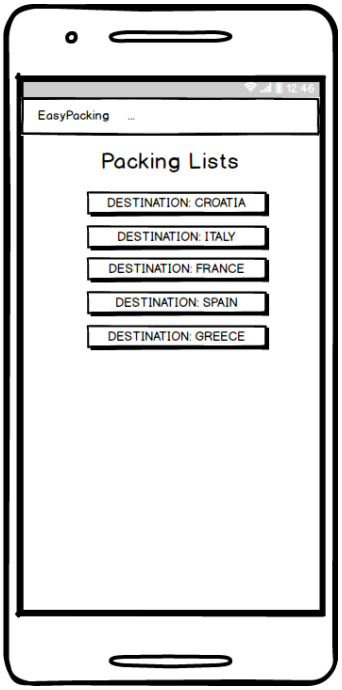
Tato obrazovka je stejná, jako obrazovka místnosti pro vytvoření seznamu, až na ten rozdíl, že zde nenajdeme tlačítko pro uložení seznamu.



Obrázek 3.20: Mockup místnosti zobrazit.

3.3.9 Zobrazení vlastních seznamů

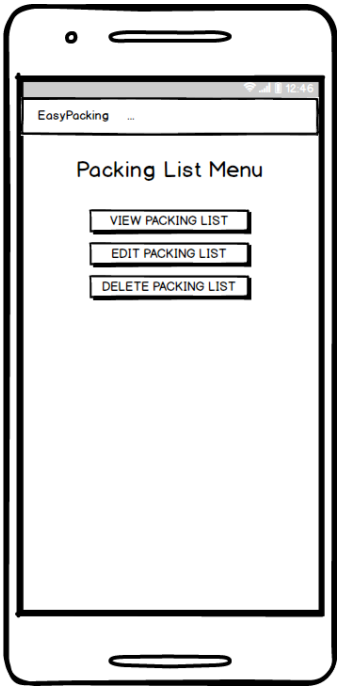
Zde se uživateli zobrazí jeho vlastní seznamy. Jedná se o tlačítka pod sebou, které jsou označena názvem destinace, pro níž je seznam vytvořen.



Obrázek 3.21: Mockup zobrazení vlastních seznamů.

3.3.10 Menu seznamy

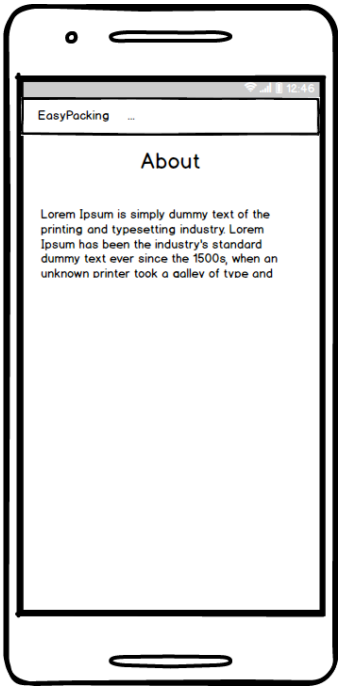
Na této obrazovce si uživatel vybírá, co chce se svým vybraným seznamem dělat. Jedná se o tři tlačítka a to zobrazit seznam, upravit seznam nebo smazat seznam.



Obrázek 3.22: Mockup menu seznamy.

3.3.11 O aplikaci

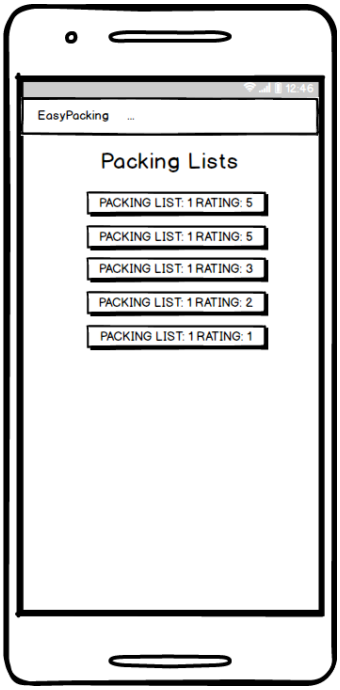
Zde uživatel nalezne informace o aplikaci



Obrázek 3.23: Mockup zobrazení vlastních seznamů.

3.3.12 Vyhledávání seznamů

Při vyhledávání seznamů se uživateli zobrazí seznam seznamů, které splňují jím zadaná kritéria. Jednotlivé seznamy jsou seřazeny podle hodnocení



Obrázek 3.24: Mockup vyhledávání seznamů.

3.4 Testování

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

Heroku focuses relentlessly on apps and the developer experience around apps. Heroku lets companies of all sizes embrace the value of apps, not the distraction of hardware, nor the distraction of servers - virtual or otherwise. [12]

Heroku je tedy 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 databaze 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 tyto data do databáze.

Jelikož i tak toto nahrání nějakou dobu trvá, používal jsem zároveň program Insomnia. Jedná se REST api clienta. Tímto programem jsem testoval aplikační server pro jednotlivé dotazy, před implementováním funkcionalit využívající tyto metody do aplikace.

3.4.1 Testování aplikačního serveru

Pro tetování aplikačního serveru jsem využil následující RubyGems. Zároveň 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.

- 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 zde je 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 generovat jednoduše kolik chceme objektů translation_criterion pomocí zavolání funkce create nebo create_list pokud chceme objektů vygenerovat více v kterémkoli testovacím souboru.

3.4.1.1 Testování jednotlivých částí aplikačního serveru

Test Databáze V testu databáze jsem testoval jestli každá tabulka zde model 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(:rating) }
    it { should validate_presence_of(:age) }
    it { should validate_presence_of(:sex) }
end
```

V 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 hodnocení, věk a pohlaví.

Test směřování U 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 #create' 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 create 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. 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, Role cant́ be blank')
    end
  end
```

```
end
```

V 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 je návratová zpráva Account created successfully a zároveň obsahuje JSONWebToken. V druhé části můžeme vidět, test zaslání špatných dat v požadavku. Kontroluje se, zda návratový kód jestli je roven 422 a zda obsahuje zprávu o špatných údajích.

3.4.2 Testovací scénáře aplikace

3.4.2.1 Vytvoření seznamu

V tomto scénáři bude otestováno zda správně funguje vytvoření nového seznamu.

1. Tester se přihlásí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vyberte možnost create new.
3. V obrazovce actions vyberte jím zvolené aktivity. Jeho výběr odsouhlasí stisknutím tlačítka ok.
4. V destinations vybere některou z uvedených destinací. Následně stiskne tlačítko ok.
5. V aktivitě additional informations vybere pole označené „Enter your age“ a vyplní jeho věk. Po té zvolí zda je muž či žena. Opět zadané informace odsouhlasí kliknutím ok.
6. Následně v locations vybere zmáčknutím některou místnost.
7. Ve zvolené místnosti přepíše u některých předmětů původní hodnotu 0 na jiné číslo. Následně odsouhlasí stisknutím ok.
8. V navrácené obrazovce locations zvoltý opět místnost z kroku 6 a zkontroluje, zda jím zadaná čísla u předmětů jsou stále vyplněna.
9. Opakuje kroky 6-8 pro všechny zobrazené místnosti. Následně stiskne tlačítko save list
10. Po uložení seznamu a přesměrování na novou obrazovku locations, postupně projde jednotlivé místnosti a zkontroluje, zda obsahují jím přidané předměty spolu s množstvím.

3.4.3 Vyhledávání seznamu

Tento testovací scénář slouží k ověření správného chování aplikace při vyhledávání seznamů.

1. Tester se přihlasí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost search.
3. V actions vybere stejné aktivity jako vyplnil ve scénáři vytvoření seznamu. Odsouhlasí výběr tlačítkem ok.
4. V destinations vybere stejné aktivity jako vyplnil ve scénáři vytvoření seznamu. Následně stiskněte ok.
5. Na obrazovce additional informations vyplní věk a pohlaví jako vyplnil ve scénáři vytvoření seznamu. Pokračuje stisknutím tlačítka ok.
6. V aktivitě packing lists stiskne tlačítko s názvem packing list: 1 rating 5. Toto je jím vytvořený seznam ze scénáře vytvoření nového seznamu. Stiskne toto tlačítko.
7. Na obrazovce locations postupně projde jednotlivé místnosti pomocí stisknutím jejich názvu.
8. V jednotlivých místnostech zkontroluje, zda obsahuje všechny předměty spolu s jejich počtem, které byly v jeho vytvořeném seznamu.

3.4.3.1 Zobrazení seznamu pomocí navigace přes moje seznamy

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

1. Tester se přihlasí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost mine.
3. Na následující obrazovce packing lists nalezne seznam, který dříve vytvořil. Vybere tento seznam pomocí kliknutí na něj.
4. V packing list menu vybere tlačítko view packing list.
5. Následně v locations projde, jako v minulém scénáři vyhledání seznamu, jednotlivé seznamy a zkontroluje, zda obsahují všechny jím přidané předměty spolu s jejich počtem

3.4.3.2 Editace seznamu pomocí navigace přes moje seznamy

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

1. Tester se přihlasí pomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost mine.
3. Na následující obrazovce packing lists nalezne seznam, který dříve vytvořili. Vybere tento seznam pomocí kliknutí na něj.
4. V packing list menu vybere tlačítko edit packing list.
5. V locations zvolí některou místnost pomocí kliknutí na její název.
6. Následně ve zvolené místnosti odebere některý předmět přepsáním jeho množství na 0, přidá předmět upravením jeho počtu z 0 na jiné číslo a upraví množství některého předmětu. Potvrďí úpravy stisknutím ok.
7. V navrácené aktivitě locations uloží úpravy stiskem tlačítka save list.
8. Na nové obrazovce locations navštiví místnost, ve které upravil předměty. Stisknutím jejího názvu
9. V místnosti zkontroluje, zda všechny předměty které jste upravili v bodě 6 jsou správně zobrazeny.

3.4.3.3 Smazání seznamu pomocí navigace přes moje seznamy

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

1. Tester se přihlasípomocí emailové adresy user@gmail.com a hesla 12345.
2. Z menu vybere možnost mine.
3. Na následující obrazovce packing lists nalezne seznam, který dříve vytvořil. Vyberte tento seznam pomocí kliknutí na něj.
4. V packing list menu vybere tlačítko delete packing list.
5. Zkontroluje zda v navrácené aktivitě packing lists se nezobrazuje žádný seznam.

3.4.4 Testování chybových zpráv

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

- Přihlášení nesprávnými údaji jak email tak heslo. Měla by se zobrazit zpráva invalid credentials. Zůstání na přihlašovací stránce.
- Registrace již registrivané emailové adresy. Kontrola zobrazení zprávy This email is already in use. Zůstání na přihlašovací stránce.
- Registrace uživatele pomocí email adresy ve špatném formátu. Zobrazení zprávy This email address is not in valid format.
- Vyhledání seznamu pomocí kritérií, kterým žádný seznam neodpovídá. Na obrazovce packing lists po zadání kriterií by se měla zobrazit zpráva There are no packing lists that meets your criteria.
- Zobrazení uživatelových seznamů pokud ještě žádné nevytvořil. Na obrazovce packing lists by se měla zobrazit chybová zpráva you do not have any packing lists yet
- Volání jednotlivých funkcionalit serveru bez validního JSONWebTokenu. Zobrazení chybové zprávy Your are not loged in or your login has expired. Přesměrování na aktivitu login.

Závěr

Výsledkem této diplomové práce je funkční aplikace EasyPacking spolu s aplikačním serverem a databází. Jednotlivé části byly řádně otestovány.

Zhodnotil jsem v práci matematické modely a problémy spojené s vývojem takovéto databáze. Příkladem je Narozeninový paradox nebo Problém průvodčího. V této části je vidět, že i v takto jednoduché aplikaci se lze setkat s problémy, které ji mohou omezovat.

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.

Během práce jsem se seznámil s vývojem aplikací pro OS Android v jazyce Kotlin.

Aplikaci lze dále upravovat a zlepšovat. Lze například přidat možnost volby typu cestování, zda uživatel bude cestovat autem nebo letadlem. Dnes se také řeší u typu cestování, zda se jedná například o Light packing, tedy cestování s minimem věcí. Dále je možnost přidat volbu termínu a s ním spojeným předpokládaným počasím. Pro omezení vytváření nových seznamů je pak možné zavést pro uživatele možnost vytvořit si seznam věcí, které si berou s sebou vždy, například léky. Tyto předměty by se pak přidaly ke každému uživatelem otevřenému seznamu. V neposlední řadě by bylo možné uživatelům umožnit nalezený seznam si převzít a upravit si ho podle sebe.

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

[4] 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/

[5] 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

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

[7] 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

[8] 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/

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

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

[12] 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
- `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