Gymnázium, Praha 6, Arabská 14

Obor programování

Ročníkový projekt

E - KUCHAŘKA

Martin Voplakal, 2.E

duben 2022

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne: Martin Voplakal

Anotace a zadání projektu

Cílem projektu bylo vytvořit přehlednou a jednoduše použitelnou mobilní aplikaci v Javě, do které si uživatel uloží recepty (včetně surovin, postupu a obrázků) a následně si bude v receptech moci vyhledávat nejen podle názvu, ale také podle ingrediencí, což může být užitečné, když chce vařit ze surovin, které doma najde. Bude tak sloužit jako inspirace při vaření. Jako případné rozšíření, pokud zbyde čas, může následovat serverový systém online ukládání a sdílení receptů mezi uživateli, či zálohování.

Obsah

Anotace a zadání projektu 1
1. Úvod 3
2. Zvolené technologie 3
2.1 AndroidX 3
2.1.1 Manifest.xml 4
2.1.2 Aktivity a fragmenty 4
2.1.3 Lifecycle fragmentů a aktivit 4
2.1.3 SQLite 5
2.1.4 HttpURLConnection 6
2.2 Serverová část 6
2.2.1 MongoDB 6
2.2.2 Node.js 7
3. Použité nástroje 8
3.1 Android Studio 8
3.2 Server a API 8
4. Uživatelské rozhraní (GUI) 8
4.1 Recepty (vyhledávání) 8
4.2 Nový recept 9
4.3 Nastavení 9
Závěr 10
Seznam obrázků 10
Reference 10
Další použité materiály 11

1. Úvod

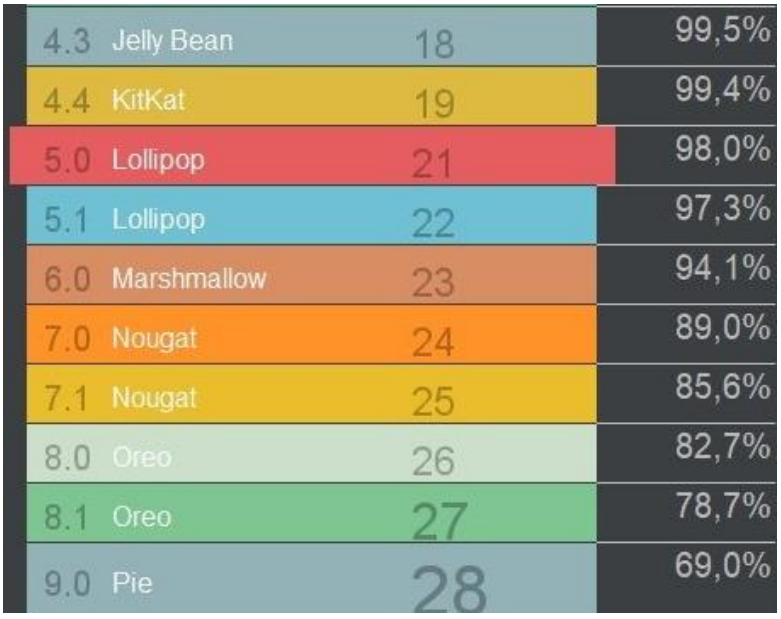
Když jsem si vybíral zadání práce, chtěl jsem si zvolit něco, co by mělo užitný potenciál a zároveň jsem mohl objevit nějakou novou technologii a něco nového se naučit. Jeden z cílů tohoto projektu bylo tedy naučit se pracovat s nativním grafickým frameworkem od Googlu AndroidX a později jsem přidal i serverovou část pro sdílení receptů mezi uživateli a zálohování dat napsanou v Node.js.

2. Zvolené technologie

2.1 AndroidX

Jelikož aplikaci píši v Javě, zvolil jsem nativní grafické prostředí pro Android od Googlu. Mimo systémového Android API [1] využívám také knihovnu AndroidX, která je v tomto ohledu standardem. AndroidX [2] je aktivně vyvíjena a má velkou komunitu s množstvím dostupných materiálů na internetu, o čemž jsem se sám několikrát přesvědčil, když jsem přirozeně musel řešit chyby a dílčí problémy.

Pro vývoj aplikace je nutné mít nainstalovaný Software Development Kit (SDK) [3], jehož verze se odvíjí od nejnižší verze androidu, pro který aplikaci vyvíjíme. Toto tzv. Minimum SDK volíme při vytváření projektu a Android Studio nám poskytne odhad kolik % uživatelů bude mít k aplikaci přístup. Ostatním uživatelům s nižší verzí Androidu se aplikace v Google Play vůbec nezobrazí.



Obrázek 1 Procentuální zastopupení uživatlů systému Android podle úrovně Anroid API (zdroj: Android Studio IDE)

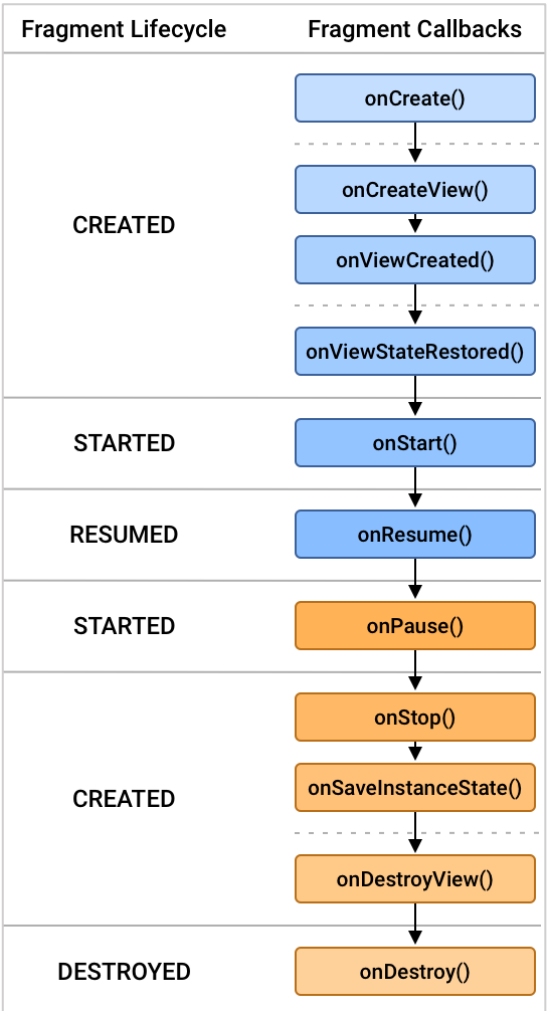
2.1.1 Manifest.xml

Manifest.xml je hlavní XML soubor s informacemi a specifikacemi o aplikaci, který Android obdrží spolu s aplikací při instalaci a obsahuje informace o ikoně, názvu aplikace, vyžadovaných oprávněních, Aktivitách, hardwarových či softwarových požadavcích apod. Konkrétně u mé aplikace je to oprávnění k práci se soubory a hardwarový požadavek na fotoaparát. Zařízením, které specifikacím nevyhovují, Google Play instalaci neumožní

2.1.2 Aktivity a fragmenty

Hlavním prvkem každé aplikace je tzv. aktivita. Je to operačním systémem řízené paradigma a prostředí pro aplikaci. Každá aplikace může mít více aktivit, které je možné aby sytém volal za různých okolností spuštění. Moje aplikace má pouze jednu aktivitu, protože do aplikace mi stačí pouze jeden vstupní bod. Nicméně aplikace jako např email nebo sociální a chatovací aplikace mají obvykle navíc aktivitu která je registrovaná v systému jako možnost sdílení a systém ji v případě výběru uživatelem v menu sdílení volá.

Dříve se pro přechody mezi layouty aplikace používaly právě aktivity, později ale byly do frameworku přidané tzv. fragmenty, které neobsluhuje přímo systém, ale dotyčný framework, což je daleko rychlejší a efektivnější. Jak již tedy bylo výše zmíněno, layout aplikace je záležitost především fragmentů, což je prvek svou funkcionalitou příbuzný aktivitě a je do ní vnořen. Aktivita pak řídí přechod mezi fragmenty, jejich vykreslování apod. Také je možné prostor aktivity rozdělit na části a najednou tak může zobrazovat různé fragmenty, které se mohou dokonce dál vnořovat. Opět uvedu příklad na mé aplikaci. Hlavní vyhledávací obrazovka s recepty je jedním z fragmentů. Obsahuje vyhledávací pole a pod ním seznam s výsledky. Tento seznam výsledků je právě onen vnořený fragment, o kterém jsem se zmiňoval výše.

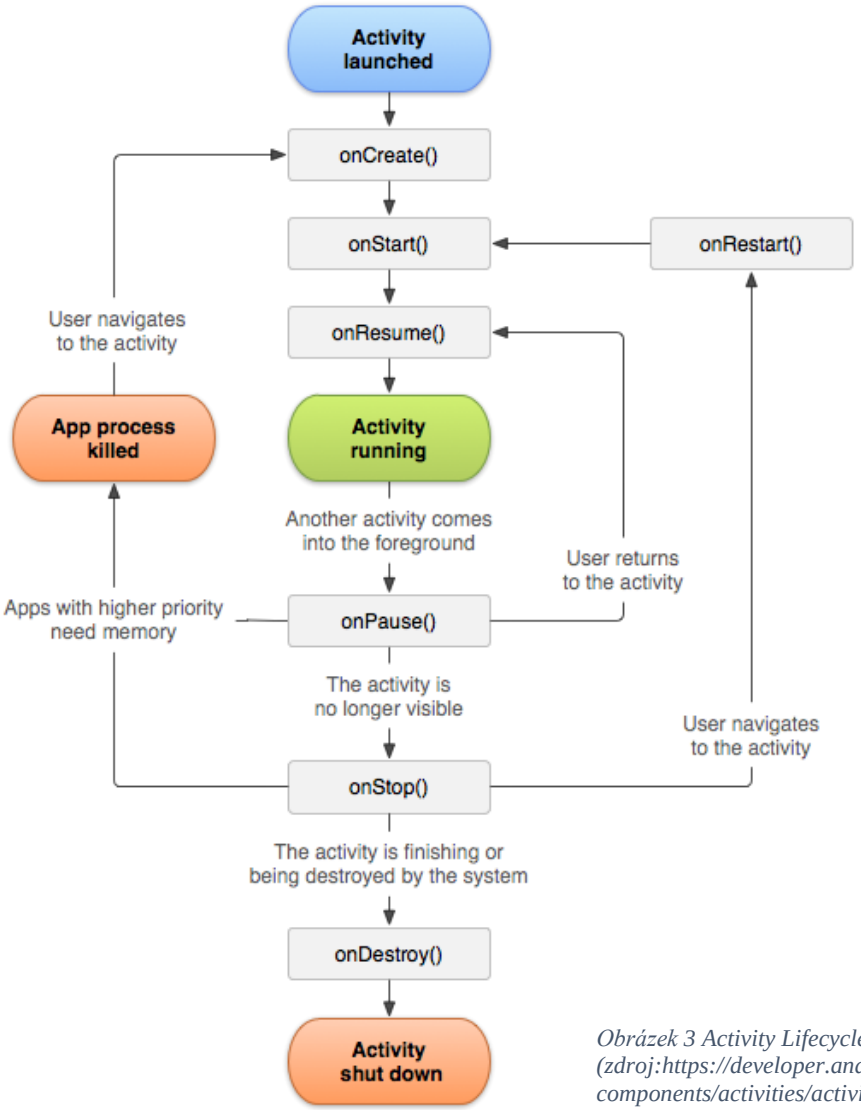


Obrázek 2 Fragment lifecycle (zdroj: https://developer.android.com/guide/fragments/lifecycle)

2.1.3 Lifecycle fragmentů a aktivit

Aplikace musí přirozeně umět reagovat, když ji uživatel opustí, zavře, nebo jen zamkne obrazovku. Následně je pak užitečné, když aplikace dostane informaci, že se k ní uživatel vrátil a je jedy schopna včas uložit, či přerušit náročné operace, aby nezpomalovala zařízení. Toto je v Androidu řešeno pomocí příslušných překrytých metod, které jsou za daných okolností volány. V mé práci používám hlavně metody na počátku tohoto řetězce. Jsou totiž nezbytné pro správné načtení dat a připravení uživatelského rozhraní. Ve chvíli, kdy je již připraven a načten XML layout, je zavolána metoda onViewCreated (…). V těle této metody je pak

možné přistupovat k určitým prvkům layoutu a data tak mohou být vložena a zobrazena. Pro ilustraci přikládám schéma lifecycle aktivit (obt. X) a fragmentů (obr. X).



Obrázek 3 Activity Lifecycle (zdroj:https://developer.android.com/guide/components/activities/activity-lifecycle)

Při přecházení mezi fragmenty potažmo aktivitami je vytvořen objekt `android.os.Bundle`, do kterého se ještě před zahájením přechodu uloží informace, které si pak otvíraný fragment/aktivita může zpracovat. Toho je využito například při otevření receptu z vyhledávání, kdy se předá fragmentu id právě otevíraného receptu a otevíraný fragment si data načte z SQLite databáze systému viz následující kapitola o SQLite. Vlastních způsobů jak je možné informace mezi články aplikace sdílet je možné vytvořit více, nicméně tento způsob je součástí framework a je doporučeno ho používat.

2.1.3 SQLite

Velmi užitečnou funkcí celého Android API je přítomnost vestavěné SQLite databáze. Není ji potřeba nijak instalovat, je součástí systému. Stačí si vytvořit vlastní třídu, která k datům bude přistupovat a následné ukládání na disk pak řeší systém. Každá aplikace má přístup jen do vlastní databáze, což také zajišťuje systém a je při odinstalaci aplikace automaticky odstraněna. Stejně tak tomu je, pokud ukládáme data na disk a rozhodneme se je ukládat do privátní složky

aplikace. Opět je systém garantem toho, že by se k datům neměla dostat jiná aplikace a to ani uživatel sám pomocí správce souborů. Variantu, že by uživatel získal oprávnění modifikovat systém `root` a zařízení si tzv. „rootnul“ nebo také „flashnul“ [4], neuvažuji.

2.1.4 HttpURLConnection

Pro vytváření http requestů nepoužívám žádnou externí knihovnu a vše dělám v `java.net.HttpURLConnection`. Přesto, že je toto řešení doporučováno v oficiální dokumentaci Google Developer [5] a ve finále funguje, mi některé věci nevyhovovaly a lehce mi komplikovaly práci. Například je nevyhnutelné pracovat s `input` a `output streamy` i u malých requestů, kde bych viděl jako optimální způsob předat jako parametr body rovnou `String`. Musí se také dodržet určité pořadí při vytváření streamů a do body metody get requestu nelze vkládat data, proto když potřebuji s requestem odeslat identifikační klíč, používám post request, což odporuje myšlence http. Toto mě opravdu překvapilo, protože http protokol body v get requestu umožňuje a při mých dřívějších zkušenostech s klientským JavaScriptem jsem s tím nikdy neměl problém. Bohužel až potom, co jsem toto měl hotové, mi došlo, že je možné klíč odesílat v hlavičce requestu. Nicméně objevil jsem to opravdu krátce před termínem odevzdání práce a na přepsání kódu už mi nezbyl čas.

Z objektivního pohledu je však použití `java.net.HttpURLConnection` o mnoho jednodušší než pracovat přímo se Sockety, kde by se ještě muselo řešit, zda je spojení https (SSL) nebo pouze http. Tady stačí poskytnout URL (začínající „http://...“ nebo „https://...“) a protokol se nastaví sám. Ještě by asi bylo dobré zmínit, že aplikace je testována jak pro server běžící s i bez SSL. Jen je nutné v případě použití pouze http (bez SSL) v AndroidManifest.xml povolit nezabezpečenou komunikaci, kterou systém jinak detekuje a generuje výjimky. Použití nezabezpečeného http je však velmi nešikovné, protože klientské klíče jsou přes internet posílány v textovém nešifrovaném formátu a to je pro běžné použití nepřípustné. Co se týká následného generování nebo čtení dat ve formátu JSON poslaných ze serveru, používám opět třídu, která je již součástí, a to konkrétně `org.json.JSONObject` [6].

Závěrem k technologiím zvoleným pro klientskou aplikaci musím konstatovat, že práce se sítí a veškeré parsování dat do a z JSON formátu je v jazyce Java nesrovnatelně náročnější, než v JavaScriptu.

2.2 Serverová část

Jak již bylo zmíněno v úvodu, uživatel má v aplikaci možnost recept sdílet. K zajištění této funkcionality je zapotřebí server, na který se recept odešle, uloží se do databáze a vrátí odkaz, který je možno sdílet. Po otevření odkazu v internetovém prohlížeči se pak načte webová stránka se sdíleným receptem. Pro její tvorbu jsem využil HTML, CSS, JavaScript a Lightbox [7] pro možnost zvětšení obrázku po kliknutí. Jelikož mám již z dřívějška elementární znalosti Node.js a balíčku Express pro http komunikaci, využil jsem pro serverovou část tyto technologie. A jako databázový server zvolil MongoDB.

2.2.1 MongoDB

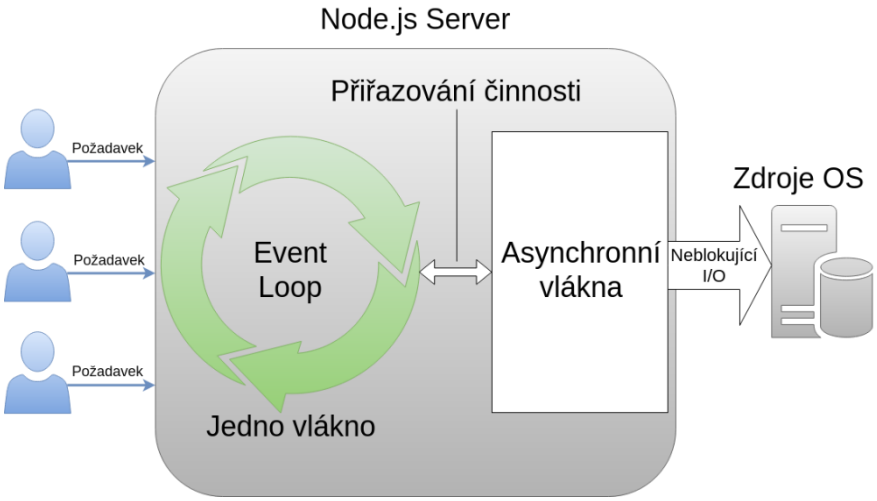
Ukládání dat na serveru je řešeno pomocí MongoDB, což je dokumentová databáze klasifikována jako NoSQL databáze, která ukládá data ve formátu binárního JSON souboru tzv. BSON. Na rozdíl od SQL relačních databází nemá předem danou vnitřní strukturu dat, tím pádem data nijak nevaliduje a uloží data tak, jak jsou poskytnuta. Proto je potřeba vyřešit jejich

validaci ještě před ukládáním. Validace je užitečná, neb zabrání uložení invalidních dat a tak nedojde k dalším chybám při následné práci s nimi. V neposlední řadě pak usnadňuje hledání chyb během vývoje. Zpětně hodnotím použití MongoDB za velmi dobré rozhodnutí, neboť ukládání dat ve formátu JSON je pohodlné, obzvlášť když se v tom samém formátu data přijímají pomocí Expressu přes http (Hypertext Transfer Protocol).

2.2.2 Node.js

Zde bych se rád chvíli věnoval prostředí Node.js a rád bych vysvětlil důvody, které mě vedly k tomu se ho začít učit a později v této práci použít.

Běhové prostředí Node.js je technologie pro psaní serverových částí aplikací (backend) s využitím jazyka JavaScript, který se jinak využívá pro programování dynamických částí webů a webových aplikací spouštěných v internetovém prohlížeči. Node.js byl v roce 2009 představen Ryanem Dahlem, jako open-source, multiplatformní nástroj pro vývoj vysoce škálovatelných serverových částí. Jádrem technologie jsou V8 JS engine [7], který pohání také internetový prohlížeč Google Chrome a moduly, rozšiřující původní nástroj o další funkcionalitu (podobně jako u JS knihovny) [8]. Node.js je na rozdíl od PHP velmi dobře škálovatelný a zvládá vykonávat velké množství požadavků najednou. Požadavky jsou totiž přidávány do smyčky událostí, ze které jsou pak postupně vyzvedávány ke zpracování. Jsou zpracovávány asynchronně, a tak požadavky, jejichž zpracování je náročnější neblokují ostatní, které mohou být dokončeny dříve.



Obrázek 4 Smyčka událostí (Event Loop)

(zdroj: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs)

Jeden z důvodů, které mě vedly k výběru tohoto řešení byl fakt, že pro mě JavaScript není novinkou, a navíc se pro použití s MongoDB velmi dobře hodí. Psaní backendu v Javě jsem zavrhl, neboť s tím nemám žádné zkušenosti a kód by byl pravděpodobně daleko delší a vývoj zdlouhavější.

3. Použité nástroje

3.1 Android Studio

Android studio je vývojové prostředí vydané společností Google roku 2013 a jeho poslední verze, kterou jsem použil byla z 16.3.2022. Je tedy stále aktuálně vyvíjené a disponuje množstvím opravdu pokročilých funkcí, z nichž některé jsem během práce ani nestihl vyzkoušet. Kromě předpokládané funkcionality, kterou od podobného nástroje pravděpodobně očekáváme, jakou je např. layout editor a standartní funkce IDE (kontrola syntaxe, debugger apod.), obsahuje třeba také Motion Editor, který umožnuje připravit vlastní animace a generovat je do xml souboru. Dalšími funkcemi, kterými Android Studio disponuje Emulator a Device Manager nebo třeba Multiple device run, který umožní spustit aplikaci jedním kliknutím na více virtuálních i fyzických zařízeních. Android Studio je od samého počátku postaveno na IntelliJ IDEA od firmy JetBrains. Drtivá většina funkcí IDE pochází právě z IntelliJ, ale Google funkcionalitu velmi rapidně rozšířil o své nástroje pro tvorbu layoutu a dalších věcí specifických pro android aplikace. Android Studio je na rozdíl od IntelliJ k dispozici ke stažení a používání zdarma a nemá žádné placené veze.

3.2 Server a API

Pro psaní JavaScriptu, HTML a CSS jsem použil Visual Studio Code [9] s příslušnými pluginy pro kontrolu syntaxe a debugger pro Node.js. Pro testování serverového API jsem použil software Postman [10], který umožňuje editování a odesílání různých druhů requestů.

4. Uživatelské rozhraní (GUI)

Uživatelské rozhraní aplikace se skládá ze tří layoutů (fragmentů) a postranního menu, kde se mezi nimi dá přecházet. Výchozí stránkou, která se otevře po startu aplikace je vyhledávání receptů.

4.1 Recepty (vyhledávání)

Uživatel může recepty procházet pomocí rolování směrem dolů a zobrazené výsledky si může vyfiltrovat pomocí zadané fráze do vyhledávacího pole. Vyhledávací frázi není nutné nijak potvrzovat, protože na vyhledávací políčko je nastaven listener, který výsledky aktualizuje při každé změněně. Celý proces pak probíhá ve vlákně, aby nedocházelo k zaseknutí UI a aplikace pracovala plynule. Vyhledávací vlákno běží vždy jen jedno a to i v případě, že se výsledky nestihly od poslední změny textu aktualizovat. Pokud dojde k takové situaci, například když uživatel zadává frázi rychle a uložených receptů je velké množství, starší vlákno je vždy ukončeno a nahrazeno vláknem novým.

Další optimalizací, která by měla umožnit práci s větším množstvím dat, je třída ArrayListRestorable, což je potomek třídy ArrayList a je místo tohoto předka používána k uchovávání právě zobrazovaných dat ve výsledcích hledání. Při filtrování výsledků hledání jsou z tohoto Listu odebírány recepty, které nevyhovují hledané frázi, pokud ale frázi začneme zkracovat, výsledky je potřeba opět přidávat. Proto si potomek uchovává všechny odebrané

prvky, aby mohly být zase znovu obnoveny bez nutnosti opětovného načítání z databáze, což je proces zdlouhavý. Dalším důvodem, který mě vedl k této optimalizaci, je způsob práce Garbage collectoru, který by odebrané výsledky nestíhal smazat z paměti, a tak by během vyhledávání rapidně rostla spotřeba paměti. Pro filtrování výsledků jsem také do třídy přidal jednoduchý algoritmus, který filtruje podle názvu receptu a jeho ingrediencí. Toto je funkcionalita, kterou jiné receptové portály nemají a já ji považuji za praktickou.

4.2 Nový recept

Pokud si chce uživatel přidat recept do aplikace, vybere v bočním menu odpovídající položku a je přesměrován na příslušný fragment. Tam může recept upravit, ikonou plus přidávat další ingredience, a nakonec potvrdí vpravo nahoře ✓ HOTOVO a recept uloží. Součástí horní nabídky jsou ale i další tlačítka. Nalevo je ikona odpadkového koše. Po kliknutí je vytvořeno dialogové okno, které požádá uživatele o souhlas, zda může recept odstranit. Vpravo nahoře se pak nachází možnost sdílení, která se dle rozměrů displeje přesune do rozbalovacího menu. Pokud uživatel vybere tuto položku, recept se odešle na server a vrátí odkaz, pod kterým je možné ho dál šířit.

4.3 Nastavení

Pro účely nastavení aplikace a provádění zálohy dat na server je v menu možnost přejít na příslušný fragment, kde je nejprve uživatel seznámen s principem zálohování:

„Aby bylo možné v budoucnu vaše data obnovit na jakémkoli jiném zařízení, je potřeba provést jejich zálohu. Zkopírujte klíč níže a uložte si ho na bezpečné místo, nebo zadejte svůj vlastní. Berte v potaz, že kdokoli, kdo odhadne, nebo jinak zjistí váš klíč bude mít přístup k vaší záloze.“

Potom co si uživatel zkopíruje (vygeneruje, pokud není žádný uložený) identifikační klíč, je mu umožněno provádět zálohování a obnovu. Přirozeně je nutné nejdříve data zálohovat a až pak je možné spustit obnovu. K identifikaci uživatele pak slouží již výše zmíněný klíč.

Během zálohování se postupně odešlou všechny recepty na server, který je pod poskytnutým klíčem uloží. Stará záloha je tak v databázi serveru nahrazena novou. Při obráceném procesu, tedy při obnově ze zálohy, se nejprve stáhne seznam id všech receptů, které jsou na serveru pod daným klíčem uložené. A až potom se recepty začnou jeden po druhém do aplikace stahovat. Po stáhnutí každého z receptů je vyhodnoceno, zda se v lokální SQLite databázi recept pod stejným id již nachází, pokud ano, je přepsán tím ze zálohy. Pokud se v databázi nenachází dotyčné id, je recept vložen. Recepty, které byly vloženy během doby od provedení zálohy a nejsou tedy součástí stahovaných dat, nebudou ovlivněny probíhajícími operacemi a nedojde tak ke ztrátě dat. O výsledku akce je pak uživatel informován prostřednictvím Toast upozornění, stejně jako při sdílení. Celý tento proces se dá využít nejen k zálohování dat pro případ ztráty zařízení nebo poškození. Ideální využití této funkcionality nalezneme i pokud chceme data přenést mezi cizími zařízeními, například chceme jinému uživateli předat všechny recepty, které máme v aplikaci. Jednoduše vložíme nový klíč, nebo ho necháme generovat automaticky, data přes něj nahrajeme a v druhém zařízení pak pomocí stejného klíče data stáhneme. V tomto případě pak ale musíme počítat s tím, že data, jejichž lokální id je shodné s id stahovaných receptů z jiného zařízení, budou přepsána, protože SQLite databáze na obou zařízeních bude indexovat pravděpodobně stejnými indexy. Proto bych doporučil toto používat, jen pokud na cílovém zařízení žádné recepty uložené nejsou.

Závěr

S vývojem této aplikace by se dalo ještě dále pokračovat a přidávat nové funkce, či vylepšovat ty stávající. Napadá mě například přihlašování uživatelů přes email, či dokonalejší práce s obrázky, kterou bych v současnou chvíli považoval spíš za bonus, nad rámec zadání projektu. Původně jsem chtěl obrázky zálohovat stejně jako ostatní data receptu, ale kvůli určitým problémům, které se vyskytly, jsem od toho raději ustoupil. Pracoval jsem s opravdu s širokým spektrem technologií: dvěma druhy databáze, dvěma programovacími jazyky a protokolem http. Výsledek práce naplňuje moje představy a očekávání, s výsledkem jsem spokojen. O tom že se mi podaří naprogramovat například zálohování jsem z počátku měl značné pohybnosti, a tak mé očekávání byla částečně i překonána.

Seznam obrázků

Obrázek 1 Procentuální zastopupení uživatlů systému Android podle úrovně Anroid API 3
Obrázek 2 Fragment lifecycle 4
Obrázek 3 Activity Lifecycle 5
Obrázek 4 Smyčka událostí (Event Loop) 7

Reference

[1] Google Developers, „Android API reference,“ [Online]. Available: https://developer.android.com/reference.

[2] Google Developers, „AndroidX Overview,“ [Online]. Available: https://developer.android.com/jetpack/androidx?gclsrc=ds&gclsrc=ds.

[3] Google Developers, „SDK Platform release notes,“ [Online]. Available: https://developer.android.com/studio/releases/platforms.

[4] Wikipedie, „Rootování Androidu,“ [Online]. Available: https://cs.wikipedia.org/wiki/Rootov%C3%A1n%C3%AD_Androidu.

[5] Google Developer, „HttpURLConnection,“ [Online]. Available: https://developer.android.com/reference/java/net/HttpURLConnection.

[6] Google Developer, „JSONObject,“ [Online]. Available: https://developer.android.com/reference/org/json/JSONObject.

[7] Google, Chrominum comunity, „About V8,“ [Online]. Available: https://v8.dev/docs.

[8] B. Koďousková, „PROČ K VÝVOJI WEBOVÝCH APLIKACÍ POUŽÍT TECHNOLOGII NODEJS?,“ 2021. [Online]. Available: https://www.rascasone.com/cs/blog/node-js-architektura-moduly-npm.

[9] Microsoft, „Visual Studio Code,“ [Online]. Available: https://code.visualstudio.com/.

[10] Postman Inc., „Postman,“ [Online]. Available: https://www.postman.com/downloads/.

[11] Google Developers, [Online]. Available: https://developer.android.com/guide/topics/ui/settings/components-and-attributes.

[12] Github comunity, „LIGHTBOX,“ [Online]. Available: https://lokeshdhakar.com/projects/lightbox2/.

Další použité materiály

Ikona aplikace

https://www.iconfinder.com/icons/7150651/fork_knife_kitchen_food_cooking_restaurant_cook_icon

Tutoriály

https://developer.android.com/guide/navigation

https://developer.android.com/training/basics/firstapp

https://developer.android.com/guide/navigation/navigation-principles

https://developer.android.com/guide/fragments

https://stackoverflow.com/questions/886955/how-do-i-break-out-of-nested-loops-in-java

https://www.java67.com/2018/12/how-to-remove-objects-or-elements-while-iterating-Arraylist-java.html