

Semestrálna práca z premetu *Vývoj aplikácií pre mobilné zariadenie*

OWD

**vypracoval:** Jakub Pastorek

**študijná** **skupina:** 5ZYI21

**cvičiaci:** doc. Ing. Patrik Hrkút, PhD.

**termín cvičenia:** streda bloky 1-2 v Žiline dňa 11.6.2024

1. Popis a analýza riešeného problému
   1. Špecifikácia zadania

**Názov:** Owd

**Platforma:** Android

**Popis:**

Owd je nová aplikácia na delenie výdavkov medzi skupinami ľudí. Ponúka intuitívne rozhranie a pokročilé funkcie pre jednoduché sledovanie výdavkov, delenie medzi účastníkov.

**Funkcie:**

* **Základné funkcie:** Sledovanie výdavkov, delenie výdavkov, tvorba skupín.
* **Pokročilé funkcie:** Offline režim, reporty a štatistiky

**Výhody:**

* **Jednoduché a intuitívne rozhranie:** Aplikácia je navrhnutá tak, aby bola ľahko použiteľná aj pre neskúsených užívateľov.
* **Pokročilé funkcie:** Spolu ponúka širokú škálu funkcií, ktoré uspokoja aj náročných užívateľov.
* **Offline režim:** Zadávajte výdavky aj bez internetového pripojenia.
* **Bezplatná verzia:** Spolu bude dostupná v bezplatnej verzii s plnou funkcionalitou.

**Cieľová skupina:**

* Skupiny kamarátov
* Spolubývajúci
* Rodiny
* Cestujúci
* Ktokoľvek, kto chce sledovať a deliť výdavky s ostatnými

**Záver:**

Owd je nová a inovatívna aplikácia na delenie výdavkov, ktorá ponúka širokú škálu funkcií a intuitívne rozhranie. Aplikácia je ideálna pre tých, ktorí chcú jednoducho sledovať a deliť výdavky s ostatnými.

* 1. Prehľad dostupných aplikácií podobného zamerania a rozdiely oproti mojej
* **Splitwise**

1. **Používateľské rozhranie a skúsenosti:**

Splitwise: Známa pre svoje jednoduché rozhranie a schopnosť sledovať účty a dlhy. Podporuje viaceré meny a ponúka funkcie ako emailové pripomienky pre platby.

Moja aplikácia: Zameriava sa na detailný prehľad každého výdavku vrátane grafického zobrazenia zostatkov a dlhov, čo môže poskytnúť viac vizuálne a intuitívne pochopenie skupinových financií.

1. **Funkcie:**

*Splitwise*: Ponúka Pro verziu s pokročilými funkciami ako ukladanie fotografií vo vysokom rozlíšení, vyhľadávanie v histórii výdavkov a bez reklám. Vyžaduje registráciu a pravidelné používanie emailu pre pripomienky.

*OWD*: Zahŕňa grafický prehľad zostatkov a integrované rozbaľovacie menu na kartách výdavkov pre akcie ako odstránenie, čo zjednodušuje správu výdavkov priamo v aplikácii.

* **Tricount**

1. **Používateľské rozhranie a skúsenosti:**

*Tricount*: Umožňuje všetkým členom skupiny pridávať výdavky a poskytuje jasný prehľad o tom, kto komu dlží. Neposkytuje však podrobné štatistiky ani konvertor mien v bezplatnej verzii.

*OWD*: Poskytuje prehľad všetkých výdavkov a detailný graf zostatkov, čo môže byť viac pútavé a informatívne pre používateľov.

1. **Funkcie:**

*Tricount*: Zahŕňa funkcie ako online zálohy a integráciu s PayPal pre platby, ale kategorizácia výdavkov je obmedzená a má pomerne vysoké náklady na používanie bez reklám.

*OWD*: Zameriava sa na jednoduchosť používania s integrovanými možnosťami menu pre správu výdavkov a nevyžaduje prémiové platby na odstránenie reklám, čo môže byť finančne výhodnejšie pre používateľov.

1. **Funkčnosť:**

*Tricount*: Vyžaduje registráciu a môže mať obmedzenú funkčnosť offline.

*OWD*: Poskytuje robustné možnosti offline a bezproblémové skúsenosti pre správu výdavkov a zostatkov bez potreby neustáleho pripojenia na internet.

**Zhrnutie**

Zatiaľ čo Splitwise a Tricount sú etablované na trhu so širokou škálou funkcií a používateľskými základňami, moja aplikácia vyniká dôrazom na vizuálne zobrazenie údajov prostredníctvom grafov a prehľadným používateľským rozhraním. Okrem toho ponúka základné funkcie bez nutnosti prémiových predplatných, čo môže poskytovať prístupnejšie a používateľsky príjemnejšie skúsenosti pre správu skupinových výdavkov.

A cell phone with a screen showing numbers and numbers

Description automatically generatedA cell phone with a screen on

Description automatically generatedSplitwise Tricount

1. Návrh riešenia problému
   1. Diagram prípadov použitia

OWD umožňuje používateľom spravovať výdavky v skupine. Používatelia môžu vytvárať skupiny, pridávať členov, pridávať výdavky a zobrazovať zostatky medzi členmi skupiny.

* **Prípady použitia**
  + **Vytváranie skupiny**
    - Používateľ klepne na tlačidlo "Pridať skupinu" na hlavnej obrazovke.
    - Zobrazí sa formulár na vytvorenie skupiny.
    - Používateľ zadá názov a popis skupiny (voliteľné).
    - Používateľ pridá členov skupiny.
    - Používateľ klepne na tlačidlo "Uložiť" na vytvorenie skupiny.
  + **Pridávanie výdavkov**
    - Používateľ otvorí detail skupiny.
    - Používateľ klepne na plávajúce tlačidlo akcie "Pridať výdavok".
    - Zobrazí sa formulár na pridanie výdavku.
    - Používateľ zadá názov výdavku, sumu a osobu, ktorá platila (voliteľné).
    - Používateľ klepne na tlačidlo "Uložiť" na pridanie výdavku.
  + **Zobrazenie zostatkov**
    - Používateľ otvorí detail skupiny.
    - Používateľ prepne na zobrazenie zostatkov.
    - Aplikácia zobrazí graf znázorňujúci, koľko peňazí každý člen skupiny dlží alebo má preplatené.
  + **Zobrazenie zoznamu výdavkov**
    - Používateľ otvorí detail skupiny.
    - Používateľ prepne na zobrazenie zoznamu výdavkov.
    - Aplikácia zobrazí zoznam všetkých výdavkov skupiny, vrátane názvu, sumy a osoby, ktorá platila.
  + **Mazanie výdavkov**
    - Používateľ otvorí detail skupiny.
    - Používateľ klepne na výdavok v zozname.
    - Používateľ klepne na tlačidlo "Viac" (dropdown menu).
    - Používateľ vyberie položku "Delete epxpense".
  1. A group of white rectangular objects with black text

     Description automatically generated with medium confidenceUML diagram tried

1. Popis implementácie
   1. Navigácia

Navigácia v aplikácii je implementovaná pomocou knižnice Jetpack Navigation, ktorá poskytuje jednoduchý a deklaratívny spôsob navigácie medzi rôznymi obrazovkami (destináciami) v aplikácii.

**Navigačný graf**

Navigačný graf definuje všetky destinácie (obrazovky) a akcie medzi nimi. V tomto prípade sa navigačný graf definuje pomocou súboru OWDNavGraph.kt. Každá destinácia má svoj vlastný cieľový kód (route), ktorý slúži na navigáciu na danú destináciu.

**OwdNavHost**

A computer screen shot of a program

Description automatically generatedTento komponent spracováva navigáciu v aplikácii pomocou Jetpack Navigation. Je definovaný v súbore OWDNavGraph.kt. Vnútri tohoto komponentu je navrhnutý navigačný graf pomocou komponenty NavHost, ktorá riadi navigáciu medzi jednotlivými obrazovkami. Každá destinácia (obrazovka) je definovaná pomocou metódy composable, kde je určené, čo sa má zobraziť pri navigácii na danú destináciu.

Definícia časti navigačného grafu

* 1. ViewModel

Pre každú obrazovku existuje odpovedajúci ViewModel, ktorý zabezpečuje správu údajov a logiky súvisiacej s danou obrazovkou.

Implementácia ViewModelov pre aplikáciu prebehla nasledovne:

1. **Definícia ViewModelov**: Pre každú obrazovku aplikácie bola vytvorená samostatná trieda ViewModel, ktorá zodpovedala za správu údajov a logiky pre danú obrazovku.
2. **Závislosti a repozitáre**: Každý ViewModel mal prístup k príslušným repozitárom, ktoré zabezpečovali prístup k údajom z lokálnej databázy. Tieto repozitáre boli poskytnuté cez konštruktor ViewModelu.
3. **ViewModelProvider.Factory**: Na vytvorenie inštancií ViewModelov bol použitý ViewModelProvider.Factory, ktorý poskytoval správne závislosti v rámci jednotlivých ViewModelov.
4. **Inicializačné bloky**: V inicializačných blokoch ViewModelProvider.Factory boli definované potrebné závislosti a inicializované repozitáre pre každý ViewModel.
5. A screenshot of a computer program

   Description automatically generated**Správa údajov**: V rámci jednotlivých ViewModelov boli implementované metódy na získavanie a aktualizáciu údajov z repozitárov a ich spracovanie pre použitie v komponentoch užívateľského rozhrania.

GroupsViewModel – príklad ViewModel implementácie

* 1. Room

Implementácia knižnice Room pre databázu v aplikácii prebehla nasledovne:

1. **Definícia entít**: Boli definované triedy entít, ktoré predstavovali jednotlivé tabuľky v databáze. Tieto triedy obsahovali anotácie definujúce názvy tabuliek, stĺpcov a ich vzájomné vzťahy.
2. **Vytvorenie databázy**: Bola vytvorená abstraktná trieda, ktorá rozšírila RoomDatabase. V tejto triede boli definované abstraktné metódy pre prístup k jednotlivým DAO (Data Access Object) triedam.
3. **DAO rozhrania**: Pre každú entitu bola vytvorená príslušná DAO trieda. Tieto triedy obsahovali metódy pre vykonávanie operácií CRUD (Create, Read, Update, Delete) nad príslušnými tabuľkami.
4. **Repozitáre**: Boli vytvorené repozitárové triedy, ktoré poskytovali prístup k databáze a vykonávali operácie nad entitami pomocou príslušných DAO tried.
5. **Inicializácia databázy**: Pri spustení aplikácie bola inicializovaná inštancia databázy pomocou metódy Room.databaseBuilder().
6. **Spracovanie dát**: Dáta boli spracované pomocou metód definovaných v DAO triedach. Načítavanie, vkladanie, aktualizovanie a mazanie údajov bolo vykonávané pomocou týchto metód.
7. **Korutiny a flow**: Na asynchrónne operácie s databázou sa používali korutiny a Flow, ktoré umožňovali efektívne spravovať operácie.

A screen shot of a computer program

Description automatically generatedCelkovým cieľom implementácie knižnice Room bolo zabezpečiť efektívne ukladanie, správu a získavanie dát v aplikácii a zabezpečiť spoľahlivú komunikáciu s lokálnou databázou.

Room database – príklad implementácie Room databázy

A screenshot of a computer program

Description automatically generated

Implementácia DAO triedy

A screen shot of a computer program

Description automatically generated

Entita (tabuľka) Group

* 1. Repozitáre

Repozitáre sú zodpovedné za interakciu s databázou a poskytovanie údajov ViewModelom. Každý ViewModel má prístup k odpovedajúcemu repozitáru, ktorý zabezpečuje získavanie a aktualizáciu údajov.

Repozitáre v aplikácii boli implementované nasledovne:

1. **Rozhrania repozitárov**: Pre každú entitu bola definovaná rozhraná trieda repozitára, ktorá obsahovala metódy pre prístup k dátam tejto entity. Tieto metódy zahŕňali operácie CRUD a iné príslušné operácie spojené s danou entitou.
2. **Implementácia repozitárov**: Na základe rozhraní repozitárov boli implementované triedy repozitárov, ktoré poskytovali konkrétnu implementáciu jednotlivých metód. Tieto triedy obsahovali inštancie príslušných DAO tried, pomocou ktorých bolo možné vykonávať operácie nad databázou.
3. **Korutiny a flow**: Na realizáciu asynchrónnych operácií boli použité korutiny a Flow, ktoré umožňovali efektívne pracovať s dátami a vykonávať operácie na vedľajších vláknach.
4. **Dependency Injection**: Repozitáre boli vložené do závislostí príslušných ViewModelov a iných častí aplikácie pomocou Dependency Injection. To umožnilo jednoduchý a čistý spôsob prístupu k dátam a ich manipulácie z rôznych častí aplikácie.

Implementácia repozitárov bola zameraná na oddelenie business logiky a prístupu k dátam od ostatných častí aplikácie, čo zlepšilo modularitu a prehľadnosť kódu.

A screen shot of a computer program

Description automatically generatedImplementácia repozitára pre Offline repozitár

A screenshot of a computer program

Description automatically generated

Implementácia pre interface repozitár

* 1. SQLite

Údaje sú uložené v SQLite databáze pomocou knižnice Room Persistence. Každá entita (napríklad skupina, osoba, výdavok) má svoju odpovedajúcu triedu, ktorá je mapovaná na tabuľku v databáze.

SQLite databáza v aplikácii bola implementovaná nasledujúcimi krokmi:

1. **Definícia entít a ich vzťahy**: Boli definované entity v databázovom modeli pomocou anotácií @Entity a ich vzťahy pomocou anotácií @ForeignKey a @PrimaryKey. Každá entita zodpovedala tabuľke v SQLite databáze, pričom vzťahy medzi entitami boli reprezentované pomocou cudzích kľúčov.
2. **Definícia prístupových tried DAO**: Pre každú entitu bola vytvorená prístupová trieda DAO (Data Access Object), ktorá obsahovala metódy pre vykonávanie operácií nad príslušnou entitou v databáze. Tieto metódy boli anotované pomocou @Query, @Insert, @Update a @Delete, ktoré zabezpečovali prístup k dátam v SQLite databáze.
3. **Vytvorenie triedy databázy**: Bola vytvorená trieda, ktorá rozširuje RoomDatabase. Táto trieda obsahovala abstraktné metódy, ktoré predstavovali prístupové body k jednotlivým DAO triedam. Tiež bola použitá anotácia @Database na definovanie databázového modelu a jej verzie.
4. **Inicializácia databázy pomocou RoomDatabase**: Bola vytvorená inštancia triedy databázy pomocou Room.databaseBuilder a nastavená správna konfigurácia, ako napríklad názov databázy a verzia. Bolo taktiež zabezpečené, aby bola táto inštancia databázy jedináčikom pomocou vzoru Singleton.
   1. ViewModelProvider

Na vytvorenie inštancie ViewModelu v komponente Compose sa používa viewModel funkcia, ktorá je poskytovaná knižnicou ViewModel Compose. Inštancia ViewModelu je vytvorená pomocou AppViewModelProvider.Factory, ktorý používa inicializačné bloky na poskytnutie správnych závislostí.

ViewModelProvider v aplikácii bol implementovaný nasledujúcimi krokmi:

1. **Vytvorenie ViewModel tried**: Boli vytvorené triedy ViewModel pre jednotlivé obrazovky alebo komponenty aplikácie, ako napríklad GroupsViewModel, AddGroupViewModel a GroupDetailsViewModel. Každá z týchto tried rozširuje ViewModel z Android Architecture Components.
2. **Definícia ViewModelProvider.Factory**: Bol vytvorený vlastný ViewModelProvider.Factory. Tento objekt bol zodpovedný za vytvorenie inštancií ViewModel tried s potrebnými závislosťami. Každá trieda ViewModelProvider.Factory bola definovaná s rôznymi inicializátormi pre vytvorenie rôznych ViewModel tried.
3. **Vloženie závislostí do ViewModel tried**: Pri vytváraní inštancií ViewModel tried pomocou ViewModelProvider.Factory boli injektované závislosti ako repozitáre, databáza alebo služby potrebné pre fungovanie ViewModelov. Toto vkladanie bolo zabezpečené vo ViewModelProvider.Factory.
4. **Využitie ViewModel v UI komponentoch**: Po vytvorení inštancií ViewModel tried pomocou ViewModelProvider.Factory boli tieto triedy použité v užívateľskom rozhraní, napríklad pre získanie dát z repozitárov a ich zobrazenie v užívateľskom rozhraní.

A screenshot of a computer program

Description automatically generatedImplementácia ViewModelProvider

* 1. YCharts

YCharts je knižnica určená na tvorbu interaktívnych grafov a tabuliek, ktoré umožňujú vizualizáciu dát. Je navrhnutá tak, aby poskytovala užívateľsky prívetivé prostredie a flexibilitu pri vytváraní rôznorodých vizuálnych prezentácií dát.

Implementácia YCharts knižnice pre vytvorenie grafu prebiehala nasledovne:

1. **Použitie YCharts knižnice**: Najprv bola pridaná YCharts knižnica ako závislosť do projektu. To bolo dosiahnuté pridaním potrebných závislostí v súbore build.gradle.kts projektu.
2. **Nastavenie dát pre graf**: Dáta pre graf boli poskytnuté z ViewModel alebo iných častí aplikácie, ktoré obsahovali požadované dáta vo formáte akceptovateľnom pre YCharts knižnicu. Tieto dáta mohli pochádzať z repozitárov, siete alebo iných zdrojov.
3. **Definícia vlastností grafu**: Boli nastavené rôzne vlastnosti grafu podľa potrieb aplikácie. Tieto vlastnosti mohli zahŕňať farby, štýly, rozsahy osí, zobrazenie popisov, interaktivitu a ďalšie vlastnosti grafu.
4. **Inicializácia grafu v užívateľskom rozhraní**: V užívateľskom rozhraní, kde mal byť zobrazený graf, bol pridaný komponent YCharts graf. Tento komponent bol inicializovaný s potrebnými parametrami, ako sú dáta pre graf, typ grafu, formátovanie a iné vlastnosti.
5. A screen shot of a computer program

   Description automatically generated**Aktualizácia grafu**: Graf bol aktualizovaný pri zmene dát alebo iných podmienok, ktoré mali vplyv na jeho zobrazenie. Táto aktualizácia mohla byť spúšťaná pri zmene dát v ViewModel alebo na základe udalostí v užívateľskom rozhraní.

Vytvorenie grafu pomocou YCharts z dát

1. Zoznam použitých zdrojov

* Codelaby z cvičení
* Prednášky
* Android developer dokumentácia
* <https://github.com/codeandtheory/YCharts> - príklady použitia YCharts library