Gymnázium, Praha 6, Arabská 14

Obor Programování

MATURITNÍ PRÁCE

Matěj Bittner

Asistent Oddílové činnosti

Duben 2024

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna užitá 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, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na doby časově neomezenou a bez omezení územního rozsahu.

V……...dne…… Matěj Bittner ……

Název práce: Asistent oddílové činnosti

Autor: Matěj Bittner

Anotace: Cílem projektu bylo navrhnout a naprogramovat mobilní aplikaci pro použití v kontextu činnosti dětských oddílů (charakterově podobných např. Skautu). Při práci jsem vycházel z vlastní zkušenosti s touto činností a osobních požadavků na takovou aplikaci.

Výsledná aplikace umožňuje vytvářet akce a vyplňovat k nim dodatečné informace, se kterými dále pracuje. Například umožňuje vybrat z členů organizace ty, kteří se akce účastní, a z uložených jídel ta, která se budou vařit, načež vygeneruje nákupní seznam.

Rozdíl oproti zadání je, že aplikace nepodporuje více organizací, důvodem, proč tomu tak je se zabývám v dokumentaci.

Zadání práce: Mobilní aplikace určená pro vedoucí oddílů na usnadnění organizace oddílových akcí. Jádrem aplikace bude vytváření akcí a vyplňování informací o nich, které program zobrazí v přehledném UI a bude s nimi i jinak pracovat. Například podle počtu účastníků a jídel vytvoří program nákupní seznam, který bude možné exportovat jako pdf, nebo přímo v aplikaci odškrtávat.

Aplikace bude strukturována do organizací, k akcím organizace budou mít přístup pouze uživatelé, kteří jsou součástí dané organizace.

Jelikož aplikace bude pracovat s osobními údaji dětí, bude nutné ošetřit bezpečnost.

Obsah

1. Úvod.....5
2. Použití aplikace.....6
    2.1 Administrace členů organizace.....6
    2.2 Administrace jídel a ingrediencí.....7
    2.3 Administrace akcí.....8
3. Použité technologie.....10
    3.1 Vývojové prostředí a programovací jazyk.....10
    3.2 Komponenty Struktury aplikace.....10
        3.2.1 Room.....10
        3.2.2 Aktivity a Fragmenty.....12
        3.2.3 Recycler View.....13
4. Databázová struktura.....15
5. Nedostatky projektu.....16
    5.1 Členění do organizací.....16
    5.2 Sdílená databáze.....16
    5.3 Bezpečnost dat.....16
    5.4 UI a exportování pdf.....16
6. Závěr.....17
7. Zdroje.....18

1. Úvod

Cílem tohoto projektu bylo navrhnout a naprogramovat mobilní aplikaci. Hlavní technologie, které jsem pro to zvolil jsou:

Android Studio – vývojové prostředí pro android

Kotlin – programovací jazyk doporučený pro android

SQLite – knihovna pro práci s SQL databází

Room – prostředník mezi SQLite databází a android aplikací

Aplikace splňuje většinu zadaných funkcí, takže se v této dokumentaci zabývám převážně úspěšnými částmi a architekturou. Přesto že jsem v určitých ohledech zadání nesplnil nelze podle mého názoru tento projekt považovat za neúspěšný.

Dokumentace se tedy člení na tři části: popis aplikace, popis architektury a nedostatky projektu.

2. Použití aplikace

Jak bylo již zmíněno, aplikace slouží k usnadnění organizace akcí dětských oddílů. Její zamýšlený uživatel je vedoucí v takovém oddílu.

Funkce aplikace jsem specificky implementoval podle potřeby organizace, kde sám působím, což mělo za vedlejší účinek, že se počet potenciálních uživatelů zmenšil na asi deset lidí (počet vedoucích v naší organizaci). Je stále možné že nějaká jiná organizace nalezne v mojí aplikaci nějakou funkci i pro své užití, ale osobně to nepovažuji za pravděpodobné. Nepomáhá ani to, že UI mojí aplikace není z nejhezčích, a že obecně prostě není dovedená k dokonalosti.

V následujících kapitolách popisuji jednotlivá užití aplikace.

2.1 Administrace členů organizace

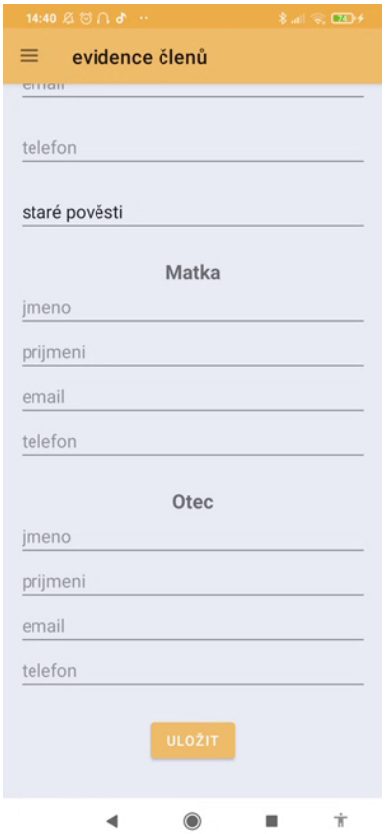
Umožňuje přidávat, mazat a upravovat členy organizace. Informace o členech se ukládají do lokální databáze.

Informace o každém členovy jsou: jméno, příjmení, přezdívka, funkce (dítě, vedoucí, atd.) , datum narození, jestli má ISIC, email, telefon, poznámka, jména a kontaktní údaje rodičů.

Ale ne všechny musí být vyplněné (například u dětí často nejsou kontaktní údaje, zato u vedoucích chybí rodiče).

Obr 1: seznam členů

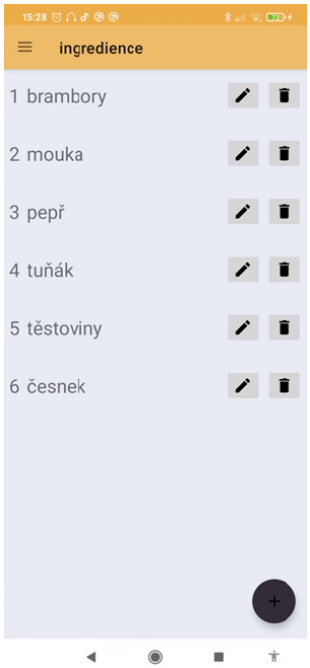
Obr 2: přidání/upravení člena 1



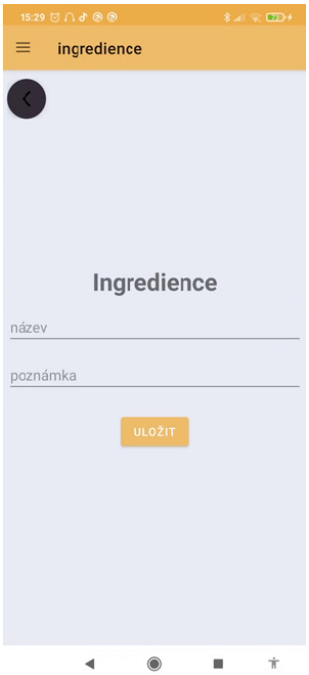
Obr 3: přidání/upravení člena 2

2.2 Administrace jídel a ingrediencí

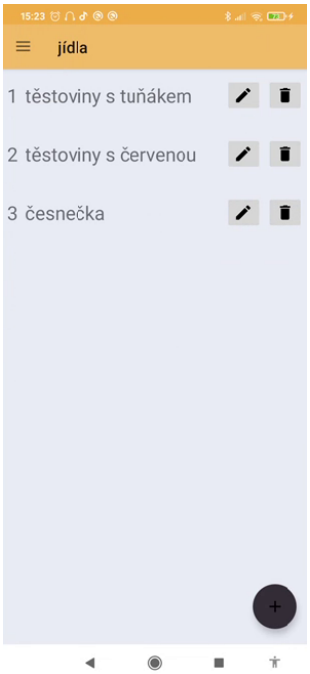
Umožňuje spravovat jídla a ingredience pro použití při organizaci akce. K jídlu se přidávají uložené ingredience a jejich množství na jednu porci. Tento přístup umožňuje uživateli kdykoliv efektivně přidat nové jídlo tak, aby při jeho přidání do akce bylo ihned možné vygenerovat nákupní seznam



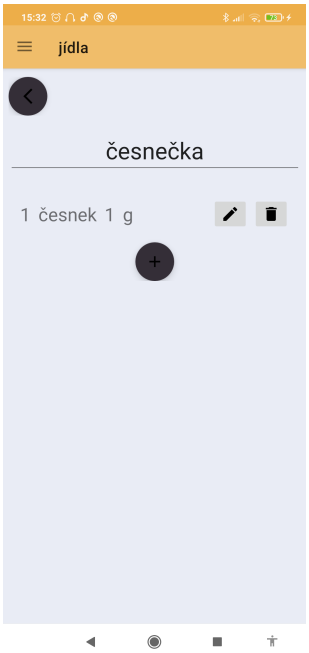
Obr 4: seznam ingrediencí



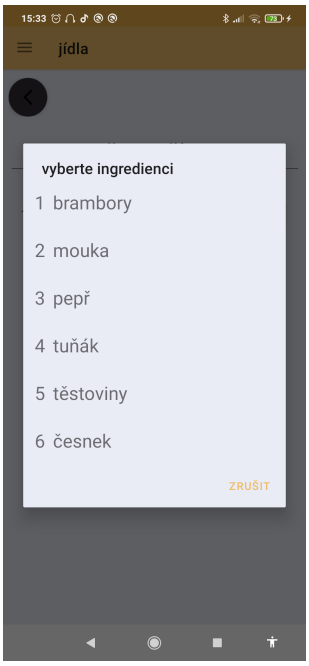
Obr 5: přidání ingredience



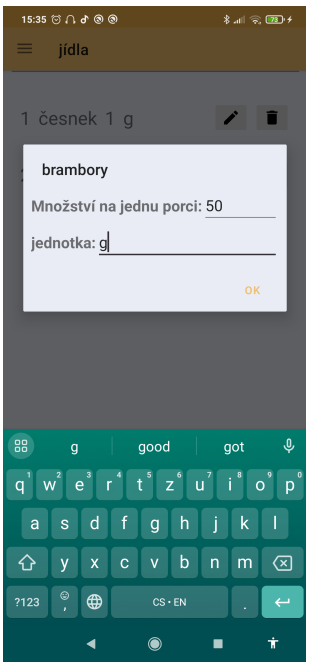
Obr 6: seznam jídel



Obr 7: úprava jídla



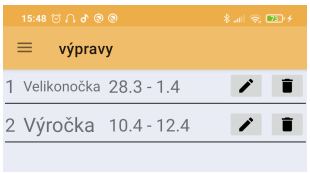
Obr 8: přidání ingredience



Obr 9: úprava množství na jednu porci

Obr 10: upravený recept

2.3 Administrace akcí

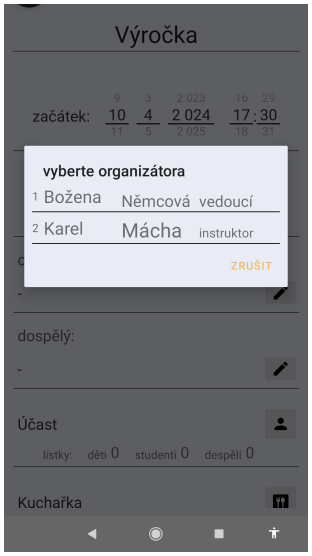


Obr 11: seznam akcí

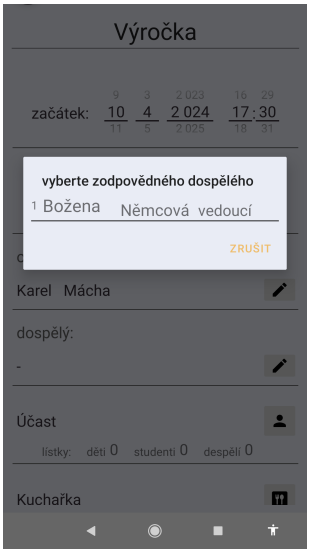
Umožňuje vytvářet akce a upravovat informace o nich.

Zde může uživatel vybrat, kromě času kdy se akce koná, například i kdo ji organizuje, a kdo je za ní zodpovědný. Organizovat může vedoucí, nebo instruktor (instruktor se podílí na vedení, ale není zletilý), ale zodpovědný může být jen vedoucí. Podle toho aplikace filtruje z evidence členů a předloží jen validní členy.

Obr 12: úprava akce

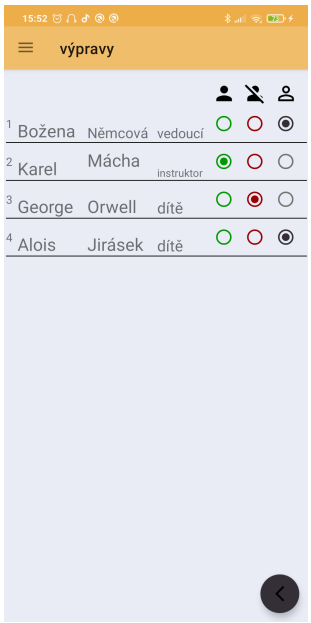


Obr 13: výběr organizátora

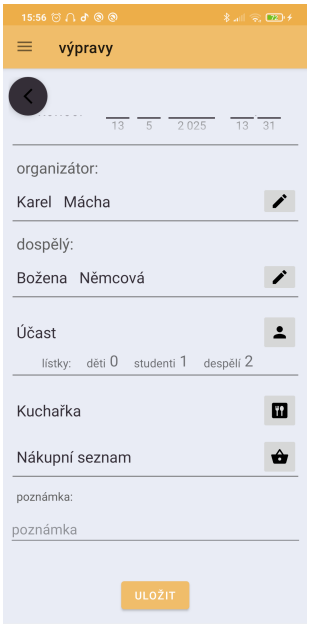


Obr 14: výběr zodpovědného

Dále může vyplňovat očekávanou účast. Podle ní se pak vygenerují počty lístků různých kategorií. Na obrázku č.16 jsou vygenerovány pro účast tří lidí.

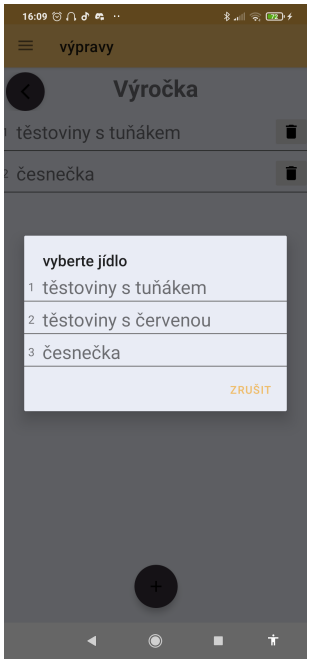


Obr 15: výběr účastníků

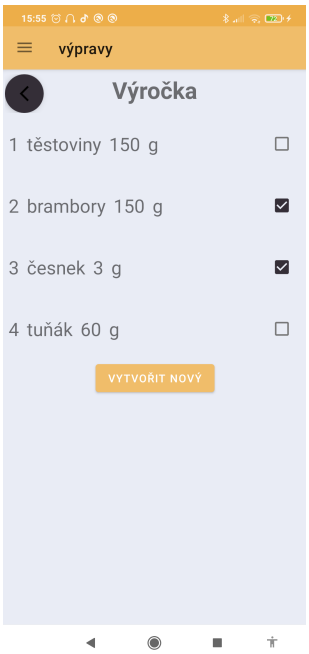


Obr 16: vygenerované počty lístků

Nakonec může ze seznamu jídel přidávat k akci jídla a podle nich, a počtu účastníků, vygenerovat nákupní seznam.



Obr 17: přidání jídla k akci



Obr 18: vygenerovaný nákupní seznam

Oba zaškrtávací seznamy (evidence a nákupní seznam) se ukládají v momentálním stavu do databáze, takže je uživatel může zavřít a posléze ve stejném stavu otevřít.

3. Použité technologie

3.1 Vývojové prostředí a programovací jazyk

Za vývojové prostředí jsem si zvolil Androi Studio, což nebylo těžké rozhodnutí, protože je to prominentní prostředí, co se týče vývoje aplikací pro android. Android Studio od Googlu je postavené na prostředí InteliJ IDEA od firmy JetBrains s úpravami pro vývoj aplikací pro android.

Volba programovacího jazyka byla o něco těžší. Pro vývoj aplikací v Android Studiu jsem měl na výběr mezi dvěma, Javou a Kotlinem. Rozhodl jsem se pro Kotlin, protože je Googlem od roku 2019 pro android preferovaný. A navíc, i když jsem s ním v minulosti nepracoval, je Javě poměrně podobný.

Kotlin je objektový programovací jazyk, který se primárně kompiluje pro Java Virtual Machine. Kotlin svým návrhem vychází z Javy a snaží se být lepší, ale je s Javou velmi kompatibilní, což umožňuje vývojářům postupný přechod.

Podrobnější rozdíly jsou přehledně popsané na oficiálních stránkách.

Comparison to Java | Kotlin. (n.d.). Kotlin Help. https://kotlinlang.org/docs/comparison-to-java.html

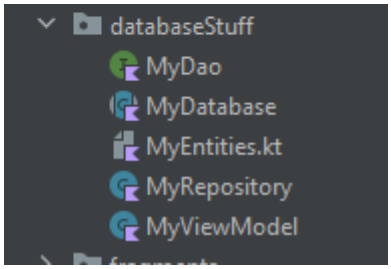
Jako nástroj pro kompilaci jsem využil Gradle.

3.2 Komponenty Struktury aplikace

3.2.1 Room

Aplikace pracuje s lokální databází, k čemuž jsem využil knihovnu Room. Room je abstraktní vrstva nad SQLite databází, která usnadňuje komunikaci s databází.

Všechna struktura potřebná ke komunikaci s databází je rozdělena do pěti souborů.



Obr 19: implementace Room databáze

V souboru ***MyEntities*** jsou popsány všechny tabulky lokální databáze. Jméno třídy s anotací @Entity je jméno korespondující tabulky, a každá proměnná třídy představuje jeden sloupec.

Soubor ***MyDao*** slouží pro přístup k datům (*DAO = Data Acces Object*). Pomocí anotací lze definovat některé funkce ,například pro smazání, nebo upravení řádku v tabulce, bez psaní SQL. Ale u složitějších požadavkú je třeba dotaz napsat ručně.

```
Matej Bittner
@Entity @Parcelize
data class Person(
    @PrimaryKey(autoGenerate = true)
    var id: Int = 0,
    var name: String,
    var surname: String,
    var status: Int,    //0 - dite, 1 - i
    var alias: String,
    var birthdate: Long,
    var fatherId: Int,
    var motherId: Int,
    var email: String,
    var phoneNumber: String,
    var note: String,
    var isic: Boolean
): Parcelable {
}

Matej Bittner
@Entity @Parcelize
data class Ingredient(
    @PrimaryKey(autoGenerate = true)
    var id: Int,
    var name: String,
    var note: String
): Parcelable{}
```

Obr 20: MyEntities

```
@Dao
interface MyDao {

    Matej Bittner
    @Upsert
    suspend fun upsertPerson(person: Person)

    Matej Bittner
    @Delete
    suspend fun deletePerson(person: Person)

    Matej Bittner
    @Upsert
    suspend fun upsertIngredient(ingredient: Ingredient)

    Matej Bittner
    @Delete
    suspend fun deleteIngredient(ingredient: Ingredient)
```

Obr 21: MyDao - automatické dotazy

```
Matej Bittner
@Query("SELECT * FROM EventShoppingLine WHERE eventId=:id")
fun getShoppingLinesForEventLive(id: Int): LiveData<List<EventShoppingLine>>

Matej Bittner
@Query("SELECT * FROM EventShoppingLine WHERE eventId=:id")
fun getShoppingLinesForEvent(id: Int): List<EventShoppingLine>

Matej Bittner
@Query("WITH at AS (SELECT * FROM EventAttendance WHERE eventId=:id) SELECT * FROM Person LEFT JOIN at ON Person.id=at.personId WHERE Person.status<4 ORDER BY Person.birthdate
fun getAttendanceOrderedByAgeLive(id: Int): LiveData<Map<Person, EventAttendance?>>

Matej Bittner
@Query("WITH at AS (SELECT * FROM EventAttendance WHERE eventId=:id) SELECT COUNT(*) FROM at JOIN person ON at.personId=Person.id WHERE Person.birthdate>:unix AND at.atends=1")
fun getChildrenCountForEvent(id: Int, unix: Long): Int  //unix je počet milisekund od epochy po datum před 18 lety
Matej Bittner
@Query("WITH at AS (SELECT * FROM EventAttendance WHERE eventId=:id) SELECT COUNT(*) FROM at JOIN person ON at.personId=Person.id WHERE Person.birthdate<:unix AND Person.isic=1
fun getStudentCountForEvent(id: Int, unix: Long): Int  //unix je počet milisekund od epochy po datum před 18 lety
```

Obr 22: MyDao - složitější dotazy

Soubory ***MyRepository*** a ***MyViewModel*** sprostředkovávají přístup k funkcím *MyDao* pro zbytek aplikace. Třída která potřebuje pracovat s databází vytvoří instanci třídy *MyViewModel* a volá její metody.

```
viewModel = ViewModelProvider( owner: this)[MyViewModel::class.java]
viewModel.getAllOrderedByName.observe(viewLifecycleOwner, Observer{person ->
    adapter.setData(person)
})
```

Obr 23: užití ViewModel pro načtení dat z databáze

A nakonec v souboru ***MyDatabase*** se databáze v anotaci definuje pomocí Entit a v těle třídy inicializuje.

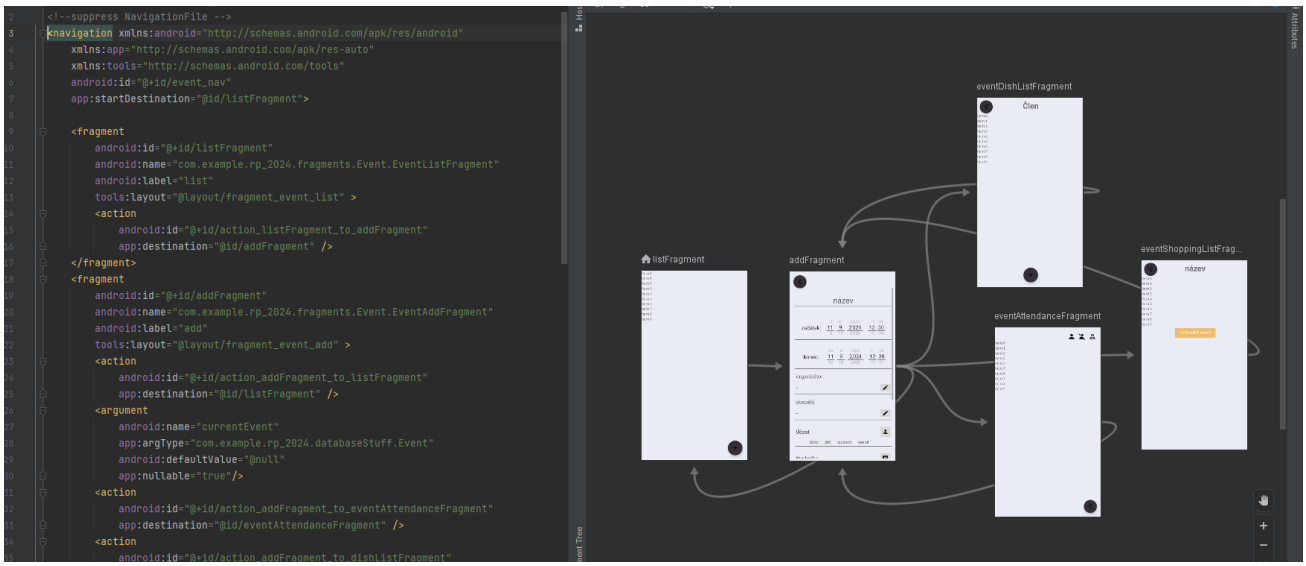
```
Matej Bittner *
@Database(
    entities = [Person::class, Ingredient::class, Dish::class,
        RecipeLine::class, Event::class, EventAttendance::class,
        EventDish::class, EventShoppingLine::class],
    version = 11,
    exportSchema = true,
)
abstract class MyDatabase: RoomDatabase() {
    Matej Bittner
    abstract fun dao(): MyDao
```

Obr 24: definování parametrů databáze

3.2.2 Aktivity a Fragmenty

Hlavními prvky struktury aplikace na androidu jsou aktivity, kde aktivita je zjednodušeně jedna obrazovka, která se může v aplikaci objevit. Jeden z přístupů k architektuře aplikace je definovat podobu aktivity v xml souboru a následně ve třídě aktivity přistupovat k prvkům tohoto souboru pomocí Id jeho prvků. Tento postup jsem zvolil i já. Jednou z jeho výhod je, že Android Studio má zabudovaný scene builder, který umožňuje v grafickém rozhraní rozmístit komponenty a následně xml soubor vygenerovat.

Kromě aktivit, jsou ve vývoji aplikací pro android hojně využívány fragmenty. Fragmenty ve zkratce umožňují existenci více obrazovek nad jednou aktivitou a snažší navigaci mezi nimi, pomocí navigačního grafu. Navigační graf je xml soubor s grafickým rozhraním podobným scene builderu, kde jsou definovány fragmenty grafu. Pro každý fragment mohou být definovány akce (tj. přesun z tohoto fragmentu na konkrétní jiný fragment v grafu) a argumenty (tj. informace, kterou fragment při navigování na něj očekává).



Obr 25: navigační graf

3.2.3 Recycler View

View je prvek rozložení stránky. Recycler View slouží k zobrazování většího množství dat ve formátu, který se opakuje. Například zobrazí seznam členů, jako tabulku, kde každý řádek reprezentuje jednoho člena a je kromě obsahu stejný jako ostatní. Tedy stačí vytvořit xml rozložení pro jeden řádek, a napsat třídu adapter, která pro každého člena vytvoří novou instanci tohoto rozložení a přidá jí za ostatní.

Postup je následující:

Vytvoříme rozložení *custom_row* třeba jako na obrázku č.26 .

Poté vytvoříme rozložení obsahující RecyclerView a k němu třídu typu Fragment.

Obr 26: custom_row layout

Vytvoříme *adapter, který se bude starat o zobrazení dat ze seznamu do rozložení.*

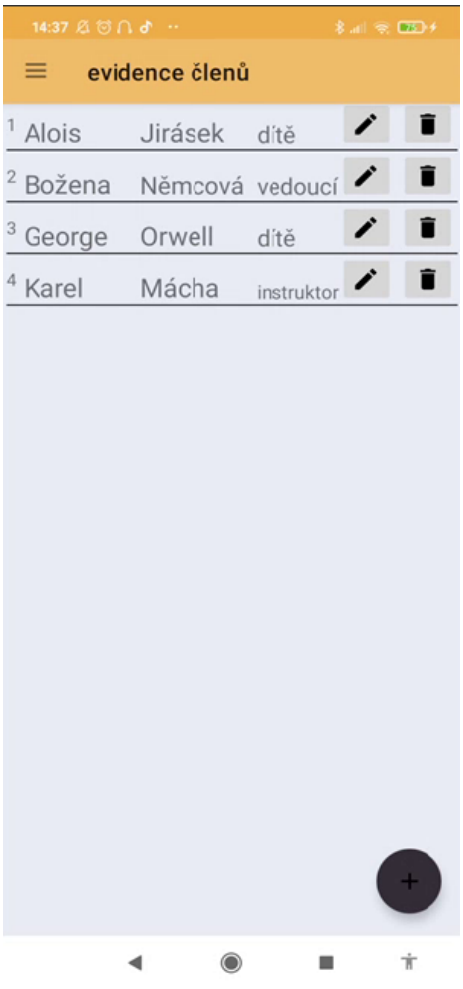
```
Matej Bittner *
class PersonListAdapter(private val viewModel : MyViewModel): RecyclerView.Adapter<PersonListAdapter.MyViewHolder>() {

    private var personList = emptyList<Person>()
    Matej Bittner
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val binding = PersonCustomRowBinding.inflate(LayoutInflater.from(parent.context), parent,  attachToParent: false)
        return MyViewHolder(binding)
    }

    Matej Bittner
    override fun getItemCount(): Int {
        return personList.size
    }

    Matej Bittner *
    @SuppressLint("SetTextI18n", "NotifyDataSetChanged")
    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {

        with(holder){ this: PersonListAdapter.MyViewHolder
            with(personList[position]) {...} //vyplní data z personList do person_custom_row
        }
    }

    Matej Bittner
    @SuppressLint("NotifyDataSetChanged")
    fun setData(person: List<Person>){
        this.personList = person
        notifyDataSetChanged()
    }
    Matej Bittner
    inner class MyViewHolder(val binding: PersonCustomRowBinding): RecyclerView.ViewHolder(binding.root)
```

Obr 27: adapter pro recycler view

A nakonec ve třídě typu fragment nastavíme našemu recycleru na parametr *adapter* instanci našeho adapteru a nastavíme obsah recycleru pomocí metody *setData()* adapteru. V tomto případě propojíme adapter s ViewModelem a databází, protože metoda *getAllOrderedByName* vrací objekt typu LiveData, který znovu zavolá *setData()* když detekuje v databázi změnu.

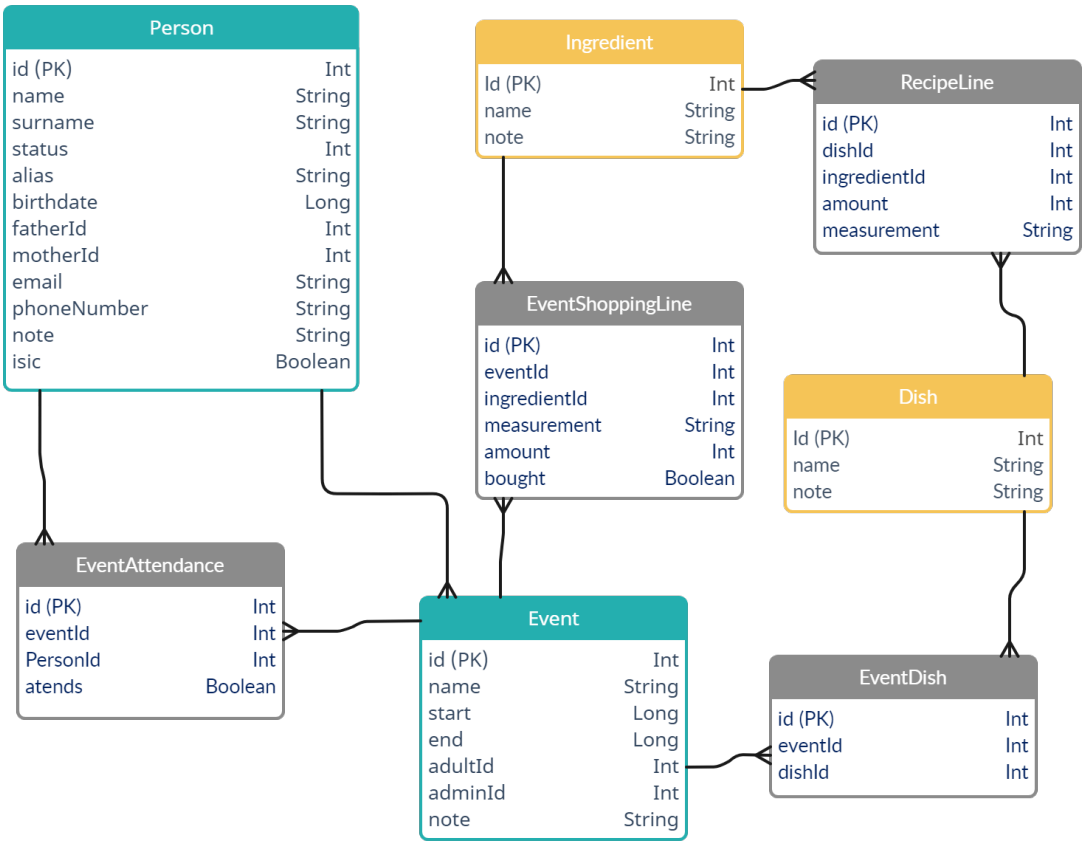
```
val adapter = PersonListAdapter(viewModel)
val recycler = binding.recyclerView
recycler.adapter = adapter
recycler.layoutManager = LinearLayoutManager(requireContext())

viewModel = ViewModelProvider( owner: this)[MyViewModel::class.java]
viewModel.getAllOrderedByName.observe(viewLifecycleOwner, Observer{person ->
    adapter.setData(person)
})
```

Obr 28: nastavení dat adapteru



Obr 29: výsledek implementace recycler view

4. Databázová struktura



Obrázek 30: RM diagram databázového schématu

Na diagramu je znázorněno databázové schéma mé aplikace. Tabulky by se dali rozdělit na skupinu předmětných (barevně) a na tabulky popisující vztahy mezi nimi (šedě). Například vztahy tabulek akcí a jídel jsou popsány v tabulce *EventDish*. Tedy za každé jídlo, které je plánováno na nějakou akci bude v této tabulce řádek s id dané akce a daného jídla. A podobnému účelu slouží i ostatní šedě znázorněné tabulky.

Kalendářní data se do všech tabulek ukládají ve formě počtu milisekund od epochy (Unix timestamp), což usnadňuje řazení dat, přesněji umožňuje to nechat řazení dat na starost SQLite už při získávání dat z databáze.

Rodiče osoby se ukládají do stejné tabulky jako osoba sama, pouze s hodnotou *status=4* a vyplněnými jen poly se jménem, příjmením a kontakty.

Vyplňování dat o nákupním seznamu k akci do tabulky *EventShoppingLine* je netriviální a tak ho v aplikaci řeším tak, že data pro každou akci vyplním jednorázově po stlačení tlačítka uživatelem. Potom se momentální nákupní seznam, i se zaškrtnutým políčkem jestli byla položka nakoupená, zachová, dokud uživatel sám opět nevygeneruje nový. Data se vyplňují podle tabulek *EventDish* a *RecipeLine*, problém je s tím že nelze efektivně reagovat na všechny možné změny v receptu jídel, v tom která jídla jsou plánovaná, a v účasti na akci v kombinaci se zachováním informace o tom, které řádky nákupního seznamu už byly nakoupeny. .

5. Nedostatky projektu

5.1 Členění do organizací

Ten kdo četl zadání tohoto projektu tuší oč se jedná. Původním plánem pro tuto aplikaci bylo aby ji mohlo využívat více členů z většího počtu organizací. Toho by se dalo dosáhnout sdílením databáze přes síť a oddělením databází různých organizací. Ale vyžadovalo by to i přihlašování uživatelů, i kdyby měla celá organizace jedno přihlašovací jméno a heslo, aby aplikace věděla pro kterou organizaci má data zobrazit.

Důvodem proč tato funkce v projektu chybí je, že jsem si špatně rozvrhnul čas a projekt dovedl do momentálního stavu na poslední chvíli. Nicméně abych se na to podíval z té lepší stránky, tak teď mám alespoň příležitost zjistit, jestli má přidávání dalších organizací nějaký smysl. Tedy pokud nebude mít o aplikaci nikdo jiný zájem, ušetřil jsem si „zbytečnou“ práci.

5.2 Sdílená databáze

V zadání se o tom nezmiňuji, ale od začátku byl můj plán, že se databáze bude synchronizovat s kopií někde na webu, ke které budou mít přístup všichni uživatelé. Tedy databáze na každém uživatelském zařízení se bude pravidelně aktualizovat a všichni budou mít relativně aktuáloní data. Toto je jediný nedostatek, který brání okamžitému praktickému použití aplikace v mé organizaci. Dokud si bude každý uživatel muset vést vlastní evidenci všeho, tak se nedá aplikace prakticky použít.

Přitom nejprimitivnější implementace by mohla být prostě nahrání souboru s databází na sdílené úložiště, případně jeho stažení.

5.3 Bezpečnost dat

Při zakončování práce na projektu jsem se pokusil o implementaci šifrování Room databáze, ale neúspěšně. Důvod neúspěchu mi není úplně jasný, ale nejspíš je to mým neporozuměním knihovně *sqlcipher*, kterou jsem se pokusil použít.

Také je nedobrý fakt že v momentálním stavu aplikace nevyžaduje přihlašovací údaje. Implementace přihlašování pro všechny členy organizace jedním heslem by bylo triviální, a rozšířením databáze o uživatele s přihlašovacími údaji by taky nebyl problém. Ale důvod je vždy stejný, špatné plánování způsobilo že jsem nestihl v termínu vše, co jsem zamýšlel.

5.4 UI a exportování pdf

Uživatelské rozhraní by potřebovalo aby někdo strávil čas jeho optimalizací, upravil barvy, odsazení atd. Důvod proč se tomu tak nestalo je stejný, jako pro dříve zmíněné nedostatky. Z důvodu špatného plánování mi nezbyl čas.

Ze stejného důvodu chybí exportování nákupního seznamu jako pdf

6. Závěr

Závěrem bych rád vyzdvihnul, že i přes nezpochybnitelné nedostatky, je výsledkem mého projektu poměrně funkční základ aplikace, kterému nechybí mnoho k tomu, aby se stal plně použitelným.

V budoucnosti se rozhodně zamyslím nad implementací některých nedostatků, jmenovitě sdílení dat přes síť by posunulo aplikaci mílovými kroky k dokončení, stejně tak zpřehlednění uživatelského rozhraní

7. Zdroje

- menu pro celou aplikaci pomocí základní aktivity, ze které ostatní aktivity dědí

  Coding Pursuit. (2021, November 2). *Navigation Drawer on Multiple Activities Using Base Activity - 08 - Water Delivery Android App* [Video]. YouTube. https://www.youtube.com/watch?v=pRieCkF1Yts [cit. 1.4.2004]

- skrytí klávesnice

  Close/hide the Android Soft Keyboard with Kotlin. (n.d.). Stack Overflow. https://stackoverflow.com/a/44500926/23886523 [cit. 1.4.2004]

- povrchní informace o *Material theme*

  How to change the status bar color in Android? (n.d.). Stack Overflow. https://stackoverflow.com/a/24997241/23886523 [cit. 1.4.2004]

- implementace Room databáze a navigačních prvků

  Stevdza-San. (2020, June 24). *ROOM Database - #1 Create Database Schema | Android Studio Tutorial* [Video and other videos in this series]. YouTube. https://www.youtube.com/watch?v=lwAvI3WDXBY [cit. 1.4.2004]

- užití *ViewBinding*

  Sunday, D. (2020, September 7). *How to Use View Binding in RecyclerView Adapter*. Medium. https://medium.com/swlh/how-to-use-view-binding-in-recyclerview-adapter-f818b96c678a [cit. 1.4.2004]

- number picker view

  GitHub - ShawnLin013/NumberPicker: :slot_machine: The android library that provides a simple and customizable NumberPicker. (n.d.). GitHub. https://github.com/ShawnLin013/NumberPicker [cit. 1.4.2004]

- převod z UnixTimestamp na datum

  How to parse Unix timestamp to date string in Kotlin. (n.d.). Stack Overflow. https://stackoverflow.com/a/51482416/23886523 [cit. 1.4.2004]

- Room export schema location

  Room - Schema export directory is not provided to the annotation processor so we cannot export the schema. (n.d.). Stack Overflow. https://stackoverflow.com/a/48674264/23886523 [cit. 1.4.2004]

- spinner view

  Spinner in Android with Example - GeeksforGeeks. (n.d.). GeeksforGeeks. https://www.geeksforgeeks.org/spinner-in-android-using-java-with-example/ [cit. 1.4.2004]

- kotlin coroutines – základy

  Coroutines basics | Kotlin. (n.d.). Kotlin Help. https://kotlinlang.org/docs/coroutines-basics.html#coroutines-are-light-weight [cit. 1.4.2004]

- výjimka s přístupem k databázi v hlavním vláknu

  Android Room - simple select query - Cannot access database on the main thread. (n.d.). Stack Overflow. https://stackoverflow.com/a/44857008/23886523 [cit. 1.4.2004]

- oprava pro chybu v AndroidStudiu s argumentem v navigačním grafu

  Android Navigation - Define Argument. (n.d.). Stack Overflow. https://stackoverflow.com/a/77986934/23886523 [cit. 1.4.2004]

- navigace mezi fragmenty s argumentem

  Android Navigation Component : Pass value (arguments) in fragments. (n.d.). Stack Overflow. https://stackoverflow.com/a/59452703/23886523 [cit. 1.4.2004]

- přístup k ViewModel z těla recycler adapteru

  How to acces shared viewModel in my recyclerAdapter. (n.d.). Stack Overflow. https://stackoverflow.com/a/70191406/23886523 [cit. 1.4.2004]

- implementace alert dialogu

  Chugh, A. (2022, August 3). *Android Alert Dialog using Kotlin*. DigitalOcean | Cloud Infrastructure for Developers. https://www.digitalocean.com/community/tutorials/android-alert-dialog-using-kotlin [cit. 1.4.2004]

- ukončení dialogu z těla adapteru pro danný dialog

  How to cancel an alertdialog? (n.d.). Stack Overflow. https://stackoverflow.com/a/56969239/23886523 [cit. 1.4.2004]

- SQLCipher – interface pro šifrování databáze (mnou implementováno neúspěšně)

  GitHub - sqlcipher/sqlcipher-android: SQLCipher for Android provides an interface to SQLCipher databases on the Android platform. (n.d.). GitHub. https://github.com/sqlcipher/sqlcipher-android [cit. 1.4.2004]

- Room – migrace

  Migrate your Room database | Android Developers. (n.d.). Android Developers. https://developer.android.com/training/data-storage/room/migrating-db-versions#export-schema [cit. 1.4.2004]

- oficiální dokumentace kotlinu

  kotlin-stdlib - Kotlin Programming Language. (n.d.). Kotlin. https://kotlinlang.org/api/latest/jvm/stdlib/ [cit. 1.4.2004]

- wikipedie – Android Studio

  Contributors to Wikimedia projects. (2013, May 16). *Android Studio - Wikipedia*. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Android_Studio [cit. 1.4.2004]

- wikipedie – Kotlin

  Contributors to Wikimedia projects. (2014, February 2). *Kotlin (programming language) - Wikipedia*. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Kotlin_(programming_language) [cit. 1.4.2004]

- webový nástroj na výrobu diagramů

  Creately. (n.d.). Creately. https://app.creately.com/d/rdyRxS88S7A/edit