ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORATIKY

Semestrálna práca

PhotoWalk – Dokumentácia

Vypracoval: **Ivan Pastierik**
Študijná skupina: **5ZYI24**
Predmet: **Vývoj aplikácií pre mobilné zariadenia**

Popis a analýza riešeného problému

Popis aplikácie a zadania

Aplikácia PhotoWalk má za účel pomáhať ľuďom pri objavovaní nových miest formou zdieľania fotiek lokácií, na ktorých spravili fotku prostredníctvom tejto aplikácie. Používateľ si bude môcť pridávať priateľov a medzi nimi zdieľať fotky. Jeho priateľom sa následne fotky zobrazia na mape a budú si môcť k ním nastaviť trasu a oni sami budú môcť na danom mieste spraviť fotku. Týmto spôsobom chcem docieliť efekt, že ľudia budú navzájom navštevovať miesta, kde boli ich kamaráti. Ľudia majú takisto tendenciu chváliť sa, že kde všade boli a toto je celkom dobrý spôsob uchovávania informácií o lokáciách, ktoré navštívili. Aplikácia využíva databázu, v ktorej sú uložené informácie o účtoch, priateľoch a zároveň sú v nej uložené aj fotky.

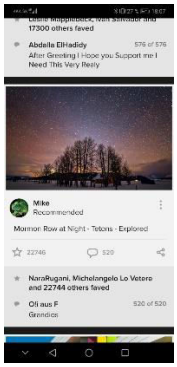
Prehľad aplikácií podobného zamerania

Na google play store som nenašiel žiadnu aplikáciu, ktorá by robila to čo moja aplikácia, preto som vybral štyri aplikácie, ktorými som sa inšpiroval a ktoré robia niektoré veci podobne.

Instagram

Instagram asi moc predstavovať nemusím, keďže je to v súčasnosti jedna z najpopulárnejších aplikácií na zdieľanie fotiek. Z tejto aplikácie som si zobral najmä to, že si môžu ľudia prezerať fotky ostatných ľudí a zároveň zdieľať svoje fotky. Vec, ktorú robím inak je tá, že ja nechcem, aby boli v mojej aplikácii nejaké komentáre, ale len čisté fotky bez textu, aby si ľudia sami zistili, že čo na tej fotke vlastne je a aby to v nich vzbudilo záujem sa tam ísť pozrieť.

Flickr



Flickr je aplikácia, ktorá je veľmi podobná aplikácií instagram, ale narozdiel od istagramu, kde mnoho ľudí fotí ich takzvané príbehy, tak flickr je skôr zameraný na zdieľanie fotiek prírody, alebo skôr na to umelecké fotenie. Z tejto aplikácie som si vzal nápad nejako presvedčiť ľudí, aby navštevovali krásne miesta v prírode a zároveň aby tieto miesta šírili medzi svojimi priateľmi.

Pokémon GO

Z hry pokémon Go, ktorá takisto obletela svet najmä svojou chytľavou hrateľnosťou, pričom ľudia očividne radi chodia po svete, zbierajú pokémonov a chvália sa svojím priateľom, že akého vzácneho pokémona chytili. Z tejto hry som sa inšpiroval spojiť aplikáciu s nejakým fyzickým pohybom a zároveň aj to, aby videli na mape značky s fotkami ostatných priateľov.

The Witcher: Monster Slayer

Hra the Witcher: Monster Slayer je hra na podobný štýl ako pokémon GO, akurát tam bolo pridaných viac príbehových prvkov oproti pokémon GO a zároveň je zasadená vo veľmi obľúbenom svete zaklínača. Túto hru som použil na doladenie detailov ohľadom spôsobu realizácie sledovania pohybu na mape a taktiež ma napadlo, že by bolo fajn si dať bod na mapu a naplánovať trasu.

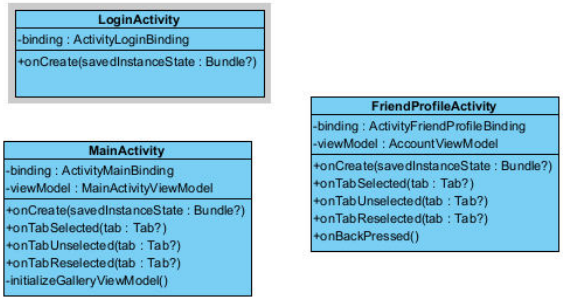
Návrh riešenia problému

Analýza navrhovanej aplikácie

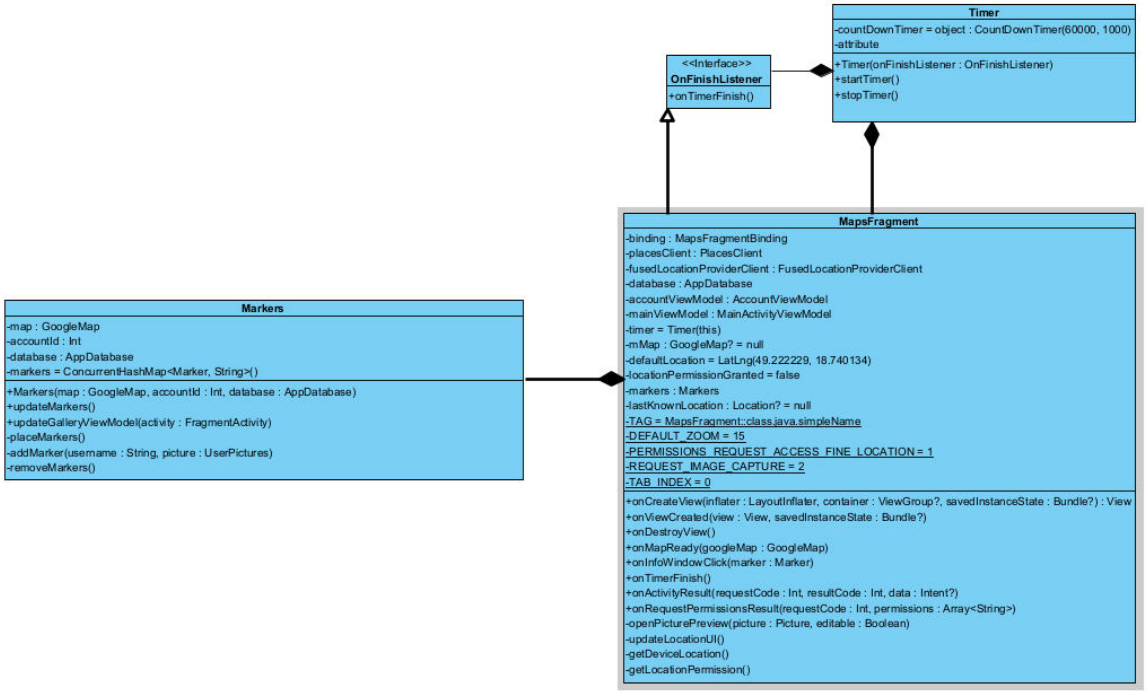
V aplikácií PhotoWalk bude potrebné mať vytvorený účet, čiže aby mohol používateľ aplikáciu používať, tak sa musí najskôr zaregistrovať. Po registrácií sa dostane na obrazovku s mapou, z ktorej bude môcť pristúpiť k nastaveniam účtu, zoznamu priateľov, galérií a zároveň bude môcť pristupovať k značkám priateľov na mape. K daným značkám bude môcť naplánovať trasu. Používateľ si bude môcť zmeniť osobné údaje svojho profilu, pridávať a odoberať priateľov a následne si prezerať aj ich galériu a ich osobné údaje. Cez galériu je možné prezerať fotky, ukázať fotku na mape a zároveň je možné aj fotku vymazať. Fotky je možné prehliadať aj cez mapu a aj cez tu mapu je možné ich mazať.

Návrh architektúry aplikácie(UML)

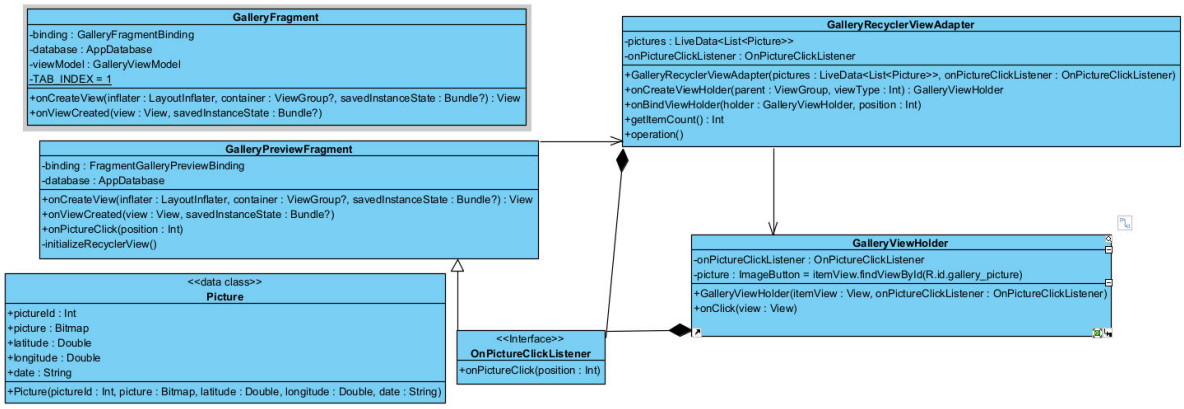
Aktivity



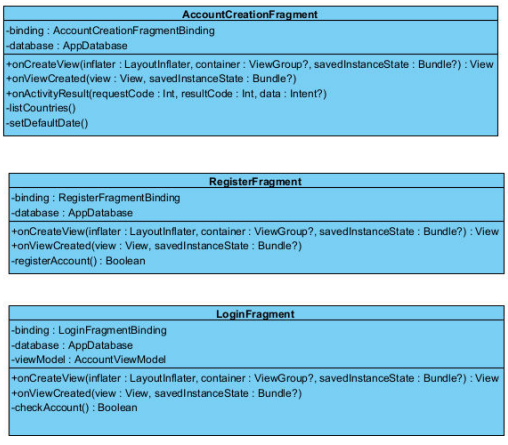
Triedy a fragmenty používané pri práci s mapou



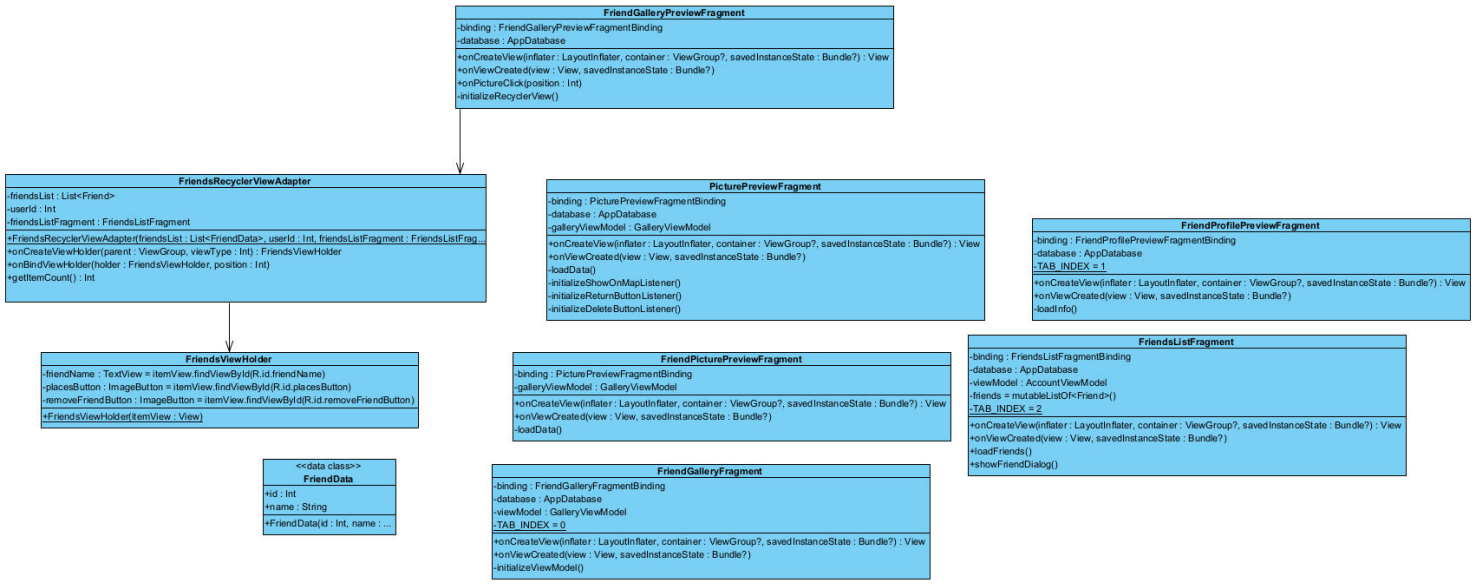
Triedy a fragmenty používané v galérií



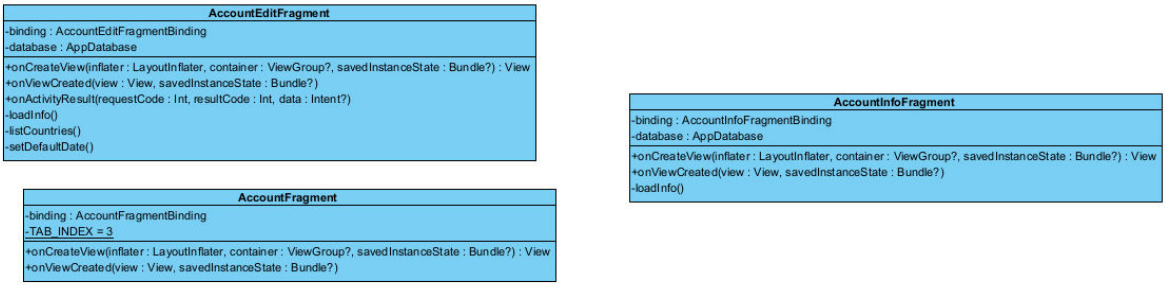
Triedy a fragmenty používané pri prihlasovaní a registrácii



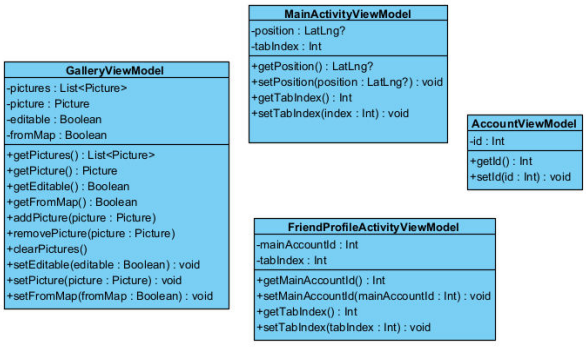
Triedy a fragmenty používané na prehliadanie profilu priateľa



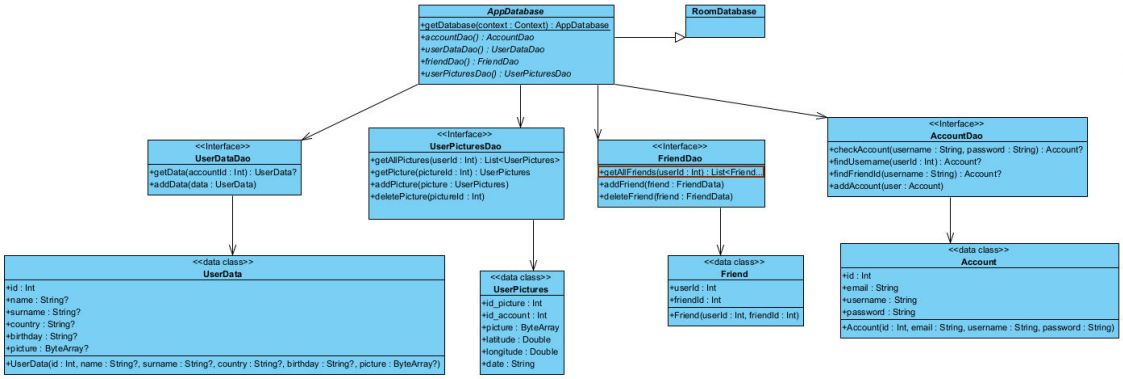
Triedy a fragmenty používané na prehliadanie a upravovanie profilu používateľa



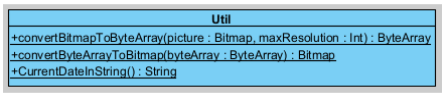
ViewModely



Databáza



Pomocná trieda



Popis implementácie

Popis aktivít a fragmentov

V aplikácií mám 3 aktivity, s tým že každá aktivita je zodpovedná za inú časť aplikácie. LoginActivity zodpovedá za časť prihlasovania sa do aplikácie, poprípade za registráciu, MainActivity zodpovedá za hlavnú činnosť aplikácie na svojom účte a FriendProfileActivity predstavuje účet priateľa používateľa.

- LoginActivity využíva hlavne fragmenty LoginFragment, RegisterFragment a AccountCreationFragment. LoginFragment sa používa pri prihlasovaní a RegisterFragment a AccountCreationFragment sa využívajú pri registrácií.
- MainActivity obsahuje 4 hlavné fragmenty, medzi ktorými sa prepína. Sú to MapsFragment, GalleryFragment, FriendsListFragment a AccountFragment. MapsFragment vykresľuje mapu a spolu s triedami Marker a Timer zabezpečuje jej funkcionalitu. Mapa je realizovaná pomocou Google Maps API(https://www.raywenderlich.com/230-introduction-to-google-maps-api-for-android-with-kotlin), ktorý je priamo integrovaný v android studio. GalleryFragment obsahuje NavHostFragment pre GalleryPreviewFragment a PicturePreviewFragment. GalleryPreviewFragment obsahuje RecyclerView, ktorý zobrazuje jednotlivé obrázky galérie a PicturePreviewFragment zobrazuje konkrétny obrázok z galérie. FriendsListFragment je zložený z RecyclerView, ktorý zobrazuje priateľov a umožňuje prezerať ich profil a aj umožňuje ich odstraňovať. AccountFragment obsahuje NavHostFragment pre AccountInfoFragment a AccountEditFragment, pričom AccountInfoFragment zobrazuje informácie o osobných údajoch a AccountEditFragment zabezpečuje možnosť editácie osobných údajov.
- FriendProfileActivity obsahuje 2 hlavné fragmenty FriendGalleryFragment a FriendProfilePreviewFragment. FriendGalleryFragment obsahuje NavHostFragment pre FriendGalleryPreviewFragment a FriendPicturePreviewFragment. Funkcionalita týchto fragmentov je podobná fragmentom GalleryPreviewFragment a PicturePreviewFragment. FriendAccountPreviewFragment zobrazuje profilové informácie o priateľovi.

Na navigáciu medzi fragmentami využívam Navigation.

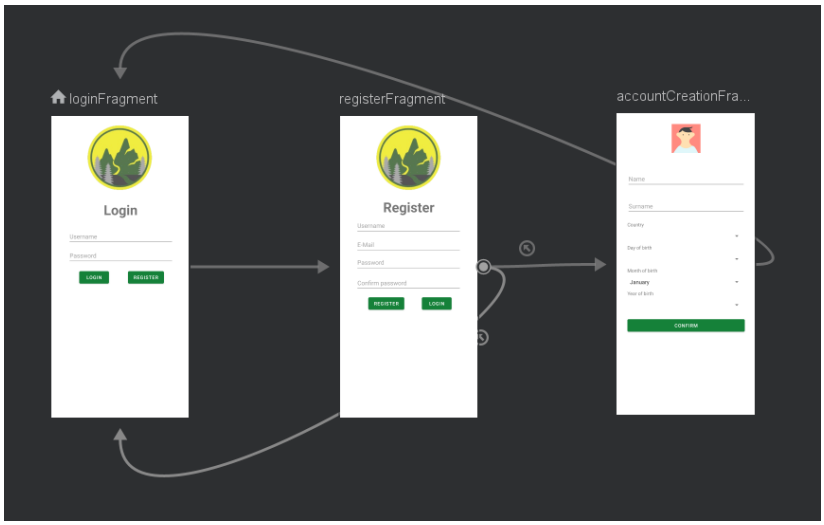
Predpokladom pre využitie Navigation je nastavenie NavHostFragmentu.

NavHostFragment v súbore activity_login.xml:

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/navHostFragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/navigation_login"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Dôležité je nastaviť atribút navGraph na xml súbor navigácie, ktorý chceme použiť a v tomto prípade aj defaultNavHost na true a nazov na androidx.navigation.fragment.NavHostFragment. Následne je možné vytvoriť akcie navigovania medzi fragmentami v súbore navigation_login.xml.

Príklad navigovania v súbore navigation_login.xml:



Každý jeden prechod, ktorý sme vytvorili má svoje špecifické id, pomocou ktorého sa odkazujeme na prechod. Navigovanie medzi fragmentami sa vykonáva pomocou metódy navigate triedy NavController, pričom jediným parametrom tejto metódy je id prechodu.

Príklad navigovania vo fragmente LoginFragment po stlačení tlačidla Register:

```
binding.RegisterButtonLogin.setOnClickListener {  it: View!
    it.findNavController().navigate(R.id.action_loginFragment_to_registerFragment)
}
```

V aktivitách aj fragmentoch využívam data binding, pričom má každá aktivita a každý fragment premennú binding, ktorá sa pri aktivitách inicializuje v onCreate a v prípade fragmentov v onCreateView. Na inicializáciu bindovacej premennej využívam triedu DataBindingUtil.

Príklad vytvorenia bindovacej premennej v aktivite:

```
binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_login)
```

Príklad vytvorenia bindovacej premennej vo fragmente:

```
binding = DataBindingUtil.inflate(inflater,
    R.layout.login_fragment,
    container,
     attachToParent: false
)
```

Predpokladom, aby data binding fungoval je mať layouty prevedené do data binding layoutov a povolený data binding v build.gradle:

```
buildFeatures {
    dataBinding true
}
```

View modely

Pre uchovávanie dát naprieč životným cyklom aplikácie používam view modely spolu s použitím databázy. Ako príklad by som uviedol použitie GalleryViewModelu, ktorý slúži na ukladanie obrázkov galérie a monitorovanie zmien v galérii ako je napríklad odstránenie obrázku.

Atribúty view modelu GalleryViewModel:

```
private var _pictures = mutableListOf<Picture>()
private var _picturesLiveData = MutableLiveData<List<Picture>>()
val pictures : LiveData<List<Picture>>
    get() = _picturesLiveData

private var _picture = MutableLiveData<Picture>()
val picture : LiveData<Picture>
    get() = _picture

private var _editable = MutableLiveData<Boolean>()
val editable : LiveData<Boolean>
    get() = _editable

private var _fromMap = MutableLiveData<Boolean>()
val fromMap : LiveData<Boolean>
    get() = _fromMap

init {
    _editable.value = false
    _picturesLiveData.value = _pictures
}
```

Atribúty view modelu využívajú triedu LiveData a MutableLiveData, pričom sa tieto triedy vyznačujú tým, že je možné sledovať ich zmeny naprieč životným cyklom fragmentu, alebo aktivity. Tento view model je viazaný na životný cyklus aktivity, ale sledovanie zmien je na úrovni fragmentu. Príklad registrácie premennej pictures na sledovanie vo fragmente GalleryPreviewFragment:

```
private fun initializeRecyclerView() {
    viewLifecycleOwner.lifecycleScope.launch {  this: CoroutineScope
        val galleryViewModel = ViewModelProvider(requireActivity())[GalleryViewModel::class.java]
        galleryViewModel.pictures.observe(viewLifecycleOwner) {  it: List<Picture>!
            binding.galleryRecyclerView.adapter = GalleryRecyclerViewAdapter(
                galleryViewModel.pictures,
                onPictureClickListener: this@GalleryPreviewFragment
            )
            binding.galleryRecyclerView.layoutManager = LinearLayoutManager(requireContext())
        }
    }
}
```

Databáza

Databáza v aplikácií je vytvorená pomocou komponentu Room, pričom databáza je vytvorená s myšlienkou uplatnenia princípu ORM. Pri tvorbe databázy využívam 3 hlavné prvky: Entity vo forme data class s anotáciou @Entity, ktorú poskytuje Room. Na prístup k entitám používam Dao, alebo data access object, čo je prakticky interface, ktorý používa anotáciu @Dao a ktorý poskytuje rôzne operácie, ktoré sa môžu robiť s entitami, alebo s databázou Jedná sa hlavne o Query príkazy na získavanie dát, mazanie dát z databázy, aktualizovanie dát v databáze a aj pridávanie dát do databázy. Posledným prvkom je samotná databáza, ktorú predstavuje abstraktná trieda s anotáciou @Database, ktorá má abstraktné metódy, ktoré vrátia daný Dao objekt. Konštrukcia databázy je riešená formou singletonu.

Príklad entity userData:

```
@Entity(tableName = "user_data", foreignKeys = [ForeignKey(entity = Account::class,
    parentColumns = ["id"],
    childColumns = ["id"])])
data class UserData(
    @PrimaryKey @ColumnInfo(name = "id") val id : Int,
    @ColumnInfo(name = "name") val name: String?,
    @ColumnInfo(name = "surname") val surname: String?,
    @ColumnInfo(name = "country") val country: String?,
    @ColumnInfo(name = "birthday") val birthday: String?,
    @ColumnInfo(name = "profile_picture", typeAffinity = ColumnInfo.BLOB) val picture: ByteArray?
)
```

Príklad Dao pre entitu userData:

```
@Dao
interface UserDataDao {

    /**
     * ziskanie pouzivatelskych dat
     *
     * @param accountId id pouzivatelskeho uctu
     * @return informacie o pouzivatelovi
     */
    @Query( value: "SELECT * FROM user_data WHERE id = :accountId")
    suspend fun getData(accountId : Int) : UserData?

    /**
     * pridanie pouzivatelskych dat
     *
     * @param data pouzivatelske data
     */
    @Insert(onConflict = REPLACE)
    suspend fun addData(data : UserData)
}
```

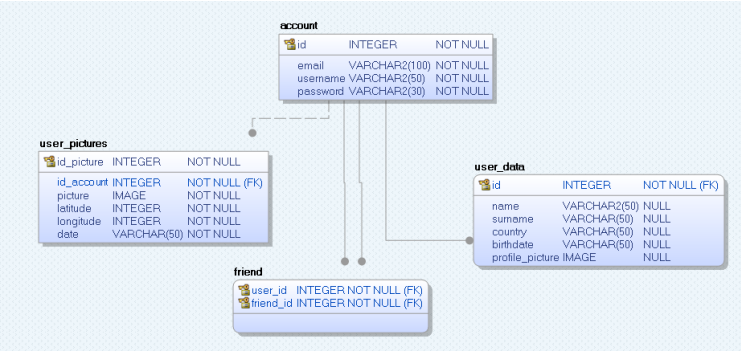
Príklad databázy:

```
@Database(entities = [Account::class, UserData::class, Friend::class, UserPictures::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    /**
     * vrati object pre pristup k entite account
     *
     * @return objekt pre pristup k datam entity account
     */
    abstract fun accountDao(): AccountDao

    /**
     * vrati object pre pristup k entite user_data
     *
     * @return objekt pre pristup k datam entity user_data
     */
    abstract fun userDataDao(): UserDataDao

    /**
     * vrati object pre pristup k entite friend
     *
     * @return objekt pre pristup k datam entity friend
     */
    abstract fun friendDao(): FriendDao

    /**
     * vrati object pre pristup k entite user_pictures
     *
     * @return objekt pre pristup k datam entity user_pictures
     */
    abstract fun userPicturesDao() : UserPicturesDao
```

Konštruktor databázy:

```
companion object {
    @Volatile
    private var INSTANCE: AppDatabase? = null

    /**
     * singleton triedy AppDatabase
     *
     * @param context kontext databazy
     * @return instancia triedy AppDatabase
     */
    fun getDatabase(context: Context): AppDatabase {
        val tempInstance = INSTANCE
        if (tempInstance != null) {
            return tempInstance
        }
        synchronized( lock: this) {
            val instance = Room.databaseBuilder(
                context.applicationContext,
                AppDatabase::class.java,
                 name: "app_database"
            ).build()
            INSTANCE = instance
            return instance
        }
    }
}
```

E-R model databázy, ktorú používam v aplikácií:



Databáza bude mať entity account, friend, user_data a user_pictures. Entita account slúži na uchovávanie informácií o používateľovi a jeho prihlasovacích údajoch. Entita user_pictures obsahuje obrázok ako BLOB a pozície o dátume odfotenia, zemepisnej dĺžke a šírke. Entita user_data obsahuje dodatočné informácie o používateľovi. Entita friend obsahuje zoznam priateľov používateľa.

RecyclerView s adaptérom

V aplikácií využívam 2 recycler viewy s adaptérmi, pričom jeden využívam na zobrazovanie obrázkov a druhý na zobrazovanie priateľov. Ako príklad ukážem recycler view na zobrazovanie obrázkov, ktorý sa využíva vo fragmentoch GalleryPreviewFragment a FriendGalleryPreviewFragment.

Pre fungovanie recyclerView je potrebné vytvoriť layout xml súbor, ktorý bude určovať, ako bude vyzerať jedna položka daného recyclerViewu, v mojom prípade sa jedná o súbor gallery_item.xml.

Následne môžeme pridať recyclerView do layoutu fragmentu, pričom ako listitem som nastavil ten môj layout súbor gallery_item.xml.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/gallery_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"
    tools:listitem="@layout/gallery_item" />
```

Keďže recyclerView nevie, ako pracovať s našim layoutom, tak mu to musíme povedať pomocou adaptéra, v mojom prípade GalleryRecyclerViewAdapter. Aby sme mohli spojazdniť adapter, tak potrebujeme v tele triedy adaptéra vytvoriť ešte jednu vnorenú triedu, ktorá bude predstavovať konrétny predmet v adaptéri. Táto trieda je holderom pre adapter.

V mojom prípade vyzerá holder takto:

```
class GalleryViewHolder(
    itemView: View,
    private val onPictureClickListener : OnPictureClickListener)
        : RecyclerView.ViewHolder(itemView), View.OnClickListener {
    val picture : ImageButton = itemView.findViewById(R.id.gallery_picture)

    init {
        picture.setOnClickListener(this)
    }
    override fun onClick(view: View) {
        onPictureClickListener.onPictureClick(adapterPosition)
    }
}
```

V tomto holderi využívam vlastný listener, ktorý implementujem vo fragmente GalleryPreviewFragment.

```
interface OnPictureClickListener {
    /**
     * metoda sa zavola po kliknuti na obrazok
     *
     * @param position index prvku holdera
     */
    fun onPictureClick(position: Int)
}
```

Po stlačení na obrázok sa zavolá metóda onPictureClick v GalleryPreviewFragmente, kde je doplnená logika, čo sa má udiať po kliknutí na obrázok.

```
override fun onPictureClick(position: Int) {
    val galleryViewModel = ViewModelProvider(requireActivity())[GalleryViewModel::class.java]
    galleryViewModel.setPicture(galleryViewModel.pictures.value!![position])
    view?.findNavController()?.navigate(R.id.action_galleryPreviewFragment_to_picturePreviewFragment)
}
```

GalleryRecyclerViewAdapter potom dedí z triedy RecyclerView.Adapter, pričom ako typový parameter je ten náš vytvorený holder. Náš adaptér implementuje z RecyclerView.Adapter tri metódy: onCreateViewHolder, onBindViewHolder a getItemCount. Metóda onCreateViewHolder nastavuje layout itemu recyclerViewu a zároveň vytvára samotný GalleryViewHolder. Metóda onBindViewHolder využíva atribút pictures na dodatočné nastavenie atribútov view holderu. V mojom prípade nastavujem obrázky. Metóda getItemCount vráti počet prvkov v holderi.

```
class GalleryRecyclerViewAdapter(private val pictures : LiveData<List<Picture>>,
                                 private val onPictureClickListener: OnPictureClickListener) :
    RecyclerView.Adapter<GalleryRecyclerViewAdapter.GalleryViewHolder>() {

    /**
     * metoda zabezpecuje vytvorenie holdera typu GalleryViewHolder
     *
     * @param parent kontext otca
     * @param viewType typ pohladu
     * @return holder typu GalleryViewHolder
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): GalleryViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(
            R.layout.gallery_item,
            parent,  attachToRoot: false)
        return GalleryViewHolder(itemView, onPictureClickListener)
    }

    /**
     * sluzi na inicializaciu prvkov holdera z atributu friendsList
     *
     * @param holder holder typu GalleryViewHolder
     * @param position index prvku holdera
     */
    override fun onBindViewHolder(holder: GalleryViewHolder, position: Int) {
        val currentItem = pictures.value!![position]

        holder.picture.setImageBitmap(currentItem.picture)

    }

    /**
     * vrati pocet prvkov v holderi
     *
     * @return pocet prvkov v holderi
     */
    override fun getItemCount(): Int {
        return pictures.value!!.size
    }
```

Pri vytváraní adaptéru sa ako prvý parameter konštruktora zvolí atribút pictures typu LiveData<List<Picture>> a ako druhý parameter konštruktora sa zvolí inštancia triedy, ktorá implementuje onPictureClickListener, čiže ištancia triedy GalleryPreviewFragment:

```
binding.galleryRecyclerView.adapter = GalleryRecyclerViewAdapter(
    galleryViewModel.pictures,
    onPictureClickListener: this@GalleryPreviewFragment
)
```

Implementácia mapy google vo fragmente MapsFragment

Mapa sa vytvorí pomocu SupportMapFragmentu ako je vidieť nižšie.

```
val mapFragment = childFragmentManager.findFragmentById(R.id.map) as SupportMapFragment?
mapFragment?.getMapAsync( callback: this)
```

Akonáhle bude mapa pripravená na fungovanie, tak sa zavolá metóda onMapReady, ktorej listener implementujem v triede MapsFragment.

```
override fun onMapReady(googleMap: GoogleMap) {
    mMap = googleMap
    googleMap.setOnInfoWindowClickListener(this)

    // Construct a PlacesClient
    getLocationPermission()



    markers = Markers(mMap!!, accountViewModel.id.value!!, database)
    viewLifecycleOwner.lifecycleScope.launch {  this: CoroutineScope
        markers.updateGalleryViewModel(requireActivity())
    }
    val galleryViewModel = ViewModelProvider(requireActivity())[GalleryViewModel::class.java]
    galleryViewModel.pictures.observe( owner: this) {  it: List<Picture>!
        lifecycleScope.launch {  this: CoroutineScope
            markers.updateMarkers()
        }
    }
    timer.startTimer()
}
```

Funkcia onMapReady vráti ako parameter inštanciu triedy GoogleMap a ako prvú vec registrujem seba ako listener onInfoWindowClick, čo je metóda, ktorá sa volá po kliknutí na štítok značky na mape, čo využívam na zobrazenie obrázku viazaného s daným markerom.

Následne sa zavolá funkcia getLocationPermission, ktorá zistí, či má aplikácia povolenie na prístup k polohe zariadenia a pokiaľ ho nemá, tak si ten prístup vyžiada. Potom vytvorím inštanciu triedy Markers, a aktualizujem všetky obrázky pre GalleryViewModel z databázy vrámci coroutine, čo predstavuje asynchronické spracovanie príkazov. Potom si získam GalleryViewModel a spustím observer na zmenu jeho atribútu pictures, ktorá vyvolá aktualizovanie značiek na mape a nakoniec spustím minútový častovač, ktorý zabezpečuje pravidelnú aktualizáciu mapy z databázy.

Metódu getLocationPermission som prevzal z oficiálnej dokumentácie od googlu pre google mapy a mierne som ju upravil pre svoje účely, pričom pokiaľ bolo pridelené oprávnenie tak sa zavolá metóda updateLocationUI a pokiaľ nie, tak sa požiada o oprávnenia.

```
private fun getLocationPermission() {
    /*
     * zdroj: https://developers.google.com/maps/documentation/android-sdk/location
     * Request location permission, so that we can get the location of the
     * device. The result of the permission request is handled by a callback,
     * onRequestPermissionsResult.
     */
    if (ContextCompat.checkSelfPermission(requireActivity().application,
            Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
        locationPermissionGranted = true
        updateLocationUI()
    } else {
        ActivityCompat.requestPermissions(requireActivity(), arrayOf(Manifest.permission.ACCESS_FINE
            PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION)
    }
}
```

Pokiaľ máme oprávnenia tak metóda udateLocationUI zabezpečí posunutie kamery na našu lokáciu a zavolá metódu getDeviceLocation

```
if (locationPermissionGranted) {
    mMap?.isMyLocationEnabled = true
    mMap?.uiSettings?.isMyLocationButtonEnabled = true
    mMap?.uiSettings?.isMapToolbarEnabled = true
    mMap?.uiSettings?.isRotateGesturesEnabled = true
    mMap?.uiSettings?.isZoomControlsEnabled = true
    if (mainViewModel.position.value != null) {
        mMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(
            mainViewModel.position.value!!,
            DEFAULT_ZOOM.toFloat()))
        mainViewModel.setPosition(null)
    } else {
        getDeviceLocation()
    }
    mainViewModel.position.observe( owner: this) {  it: LatLng?
        if (it != null) {
            mMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(
                it,
                DEFAULT_ZOOM.toFloat()))
            mainViewModel.setPosition(null)
        }
    }
```

V metóde getDeviceLocation zisťujem polohu zariadenia tak, že použijem fusedLocationClienta, ktorého atribút lastLocation obsahuje moju poslednú polohu a podľa toho, či je posledná poloha známa presuniem kameru buď na poslednú známu polohu, alebo na predvolené miesto.

```
val locationResult = fusedLocationProviderClient.lastLocation
locationResult.addOnCompleteListener(requireActivity()) { task ->
    if (task.isSuccessful) {
        // Set the map's camera position to the current location of the device.
        lastKnownLocation = task.result
        if (lastKnownLocation != null) {
            mMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(
                LatLng(lastKnownLocation!!.latitude,
                    lastKnownLocation!!.longitude), DEFAULT_ZOOM.toFloat()))
        } else {
            mMap?.moveCamera(CameraUpdateFactory
                .newLatLngZoom(defaultLocation, DEFAULT_ZOOM.toFloat()))
        }
```

Zoznam použitých zdrojov

- https://stackoverflow.com/questions/8232608/fit-image-into-imageview-keep-aspect-ratio-and-then-resize-imageview-to-image-d - konvertovanie bytmapy na ByteArray

```
fun convertBitmapToByteArray(picture: Bitmap, maxResolution: Int): ByteArray {
    // zdroj: https://stackoverflow.com/questions/8232608/fit-image-into-imageview-
    val stream = ByteArrayOutputStream()
    val ratio: Float = min(
        a: maxResolution.toFloat() / picture.width,
        b: maxResolution.toFloat() / picture.height
    )
    val width =
        (ratio * picture.width).roundToInt()
    val height =
        (ratio * picture.height).roundToInt()

    val resizedBitmap = Bitmap.createScaledBitmap(
        picture, width,
        height, filter: true
    )
    resizedBitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, stream)
    return stream.toByteArray()
}
```

- https://stackoverflow.com/questions/7620401/how-to-convert-image-file-data-in-a-byte-array-to-a-bitmap - konvertovanie ByteArray na Bitmap

```
fun convertByteArrayToBitmap(byteArray: ByteArray): Bitmap {
    // zdroj: https://stackoverflow.com/questions/7620401/how-to-convert-image-file-data-in-a-byte-array-to-a-bitmap
    return BitmapFactory.decodeByteArray(byteArray, offset: 0, byteArray.size)
}
```

- https://www.codegrepper.com/codeexamples/kotlin/get+formatted+current+date+and+time+kotlin – získanie aktuálneho času

```
fun CurrentDateInString() : String {
    // zdroj: https://www.codegrepper.com/code-examples/kotlin/get+formatted+current+date+and+time+kotlin
    val date = Calendar.getInstance().time
    return SimpleDateFormat( pattern: "dd.MM.yyyy':' HH:mm:ss z").format(date)
}
```

- https://stackoverflow.com/questions/62350236/how-to-use-an-arrayadapter-with-spinner-kotlin - použitie adaptéru pre spinner

```
binding.country.adapter = ArrayAdapter(requireActivity().application,
    R.layout.spinner_item,
    countries)
```

- https://stackoverflow.com/questions/9760341/retrieve-a-list-of-countries-from-the-android-os - získanie zoznamu všetkých krajín z operačného systému

```
val isoCountryCodes = Locale.getISOCountries()
val countries = mutableListOf<String>()
for (countryCode in isoCountryCodes) {
    countries.add(Locale( language: "", countryCode).displayCountry)
}
```