ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Zadanie semestrálnej práce

Companion Aplikácia

Vypracoval: **Radovan Findra**
Študijná skupina: 5ZYS22
Predmet: vývoj aplikácií pre mobilné zariadenia
Cvičiaci: doc. Ing. Patrik Hrkút, PhD.

Obsah

1. Úvod

Companion aplikácie sú sprievodné aplikácie ktoré často využívajú ineligentné hodinky a podobné zariadenia. Cieľom je posielať informácie o notifikáciách, aktuálne prehrávanej hudbe, stave batérie, záznamoch v kalendári...

K vytvoreniu takejto aplikácie ma motivoval môj voľnočasový projekt ktorý sa sústredí na implementovanie moderných funkcií do starších automobilov bez rušivého zásahu do pôvodného designu interiéru.

Po preskúmaní podobných aplikácií som nenašiel žiadnu ktorá by spĺňala požiadavky alebo by bola dostatočne prispôsobiteľná na moje účely.

2. Popis riešeného problému

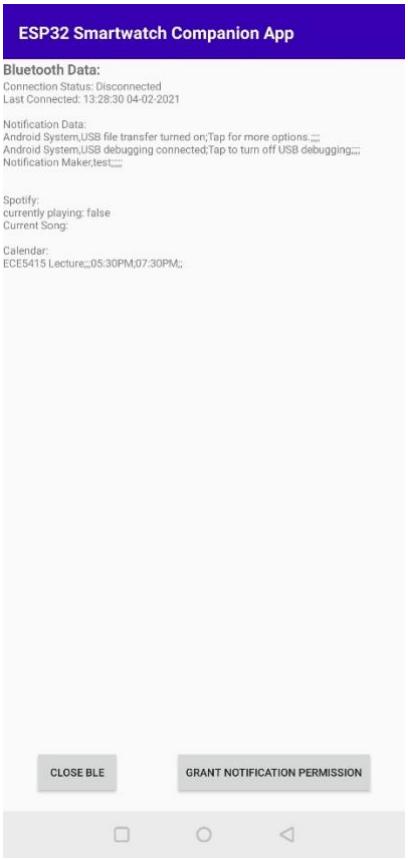
Mojím cieľom bolo naprogramovať aplikáciu ktorá dokáže komunikovať s viacerými zariadeniami súčasne a prispôsobiť posielané informácie na konkrétne zariadenia.

Porovnanie podobných aplikácií

Podobné aplikácie som na Google play nenašiel. Našiel som aplikácie ktoré dokážu ovládať GPIO piny ale žiadnu čo by fungovala ako companion aplikácia.

a. Android Companion App For BLE Devices

Táto aplikácia je companion aplikácia pre Esp32 hodinky čo je zaujímavý open source projekt. Táto aplikácia je naprogramované v Java a v mojom prípade som natrafil na kompatibilné problémy.



Výhody:

- Jednoduché použitie
- Dostatočná funkcionalita

Nevýhody:

- Aplikácia je napísaná v Jave
- Užívateľské rozhranie
- Chýba možnosť úpravy posielaných správ
- Posielanie správ len na jedno zariadenie

Moja aplikácia dokáže odosielať správy na viac zariadení, dokáže prispôsobiť aké informácie budú odoslané a súčasťou mojej aplikácie je widget na správu aktívnosti zariadení

b. Arduino ESP Bluetooth – Dabble

Toto je zaujímavá aplikácia ktorá podporuje množstvo interakcií ako napríklad odosielanie informácií o hodnotách senzorov mobilného telefónu ako akcelerometer alebo gyroskop.

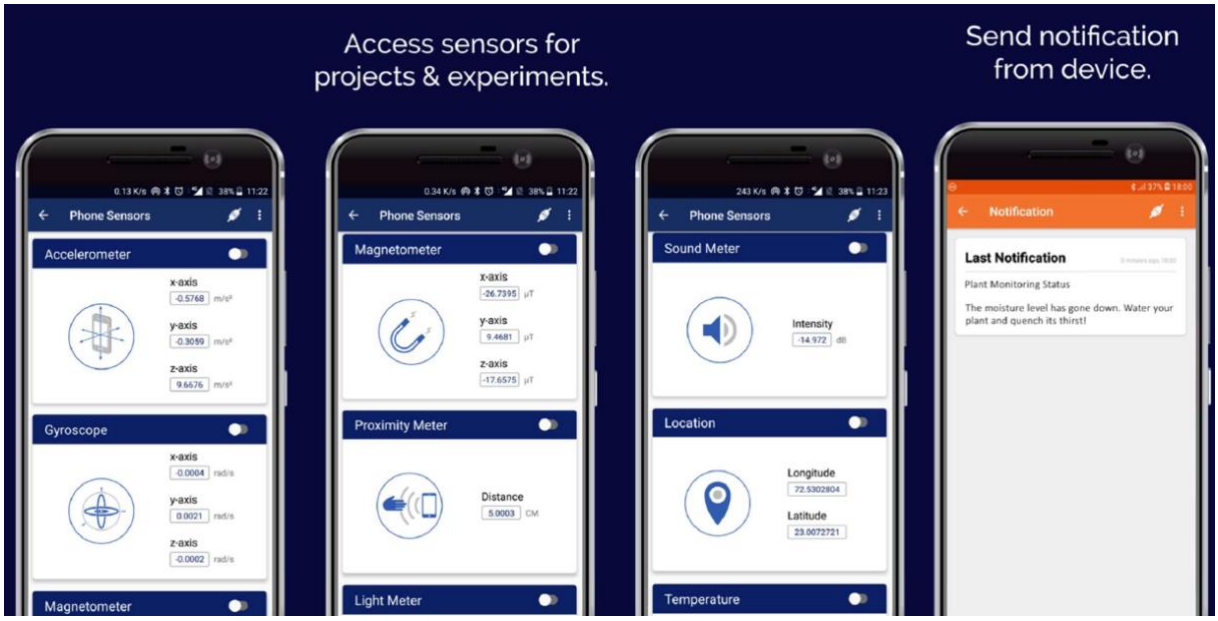
Výhody:

- Jednoduché použitie
- Veľké množstvo interakcií zo zariadením
- Možnosť posielať notifikácie

Nevýhody:

- Chýba možnosť úpravy posielaných správ
- Posielanie správ len na jednu dosku

Táto aplikácia slúži ako jednoduchý ovládač cez ktorý môžeme vyžívať veľké množstvo senzorov ktorými väčšina mobilných telefónov disponuje. Nie je určená ako Companion Aplikácia ale ako aplikácia pre učenie sa ideálne pre študentov a učiteľov.



3. Návrh riešenia problému

Analýza problému:

Získať MAC adresu zariadenia – Bluetooth discovery,

Odoslanie správy na MAC adresu - OutputStream,

Získanie informácií o notifikáciách a stave telefónu – NotificationListener,

Filtrovanie notifikácii – Definovanie aplikácií v Event,

Definovanie príležitostí na posielanie správ - Event,

Štruktúra správ,

Užívateľské rozhranie - Jetpack Compose

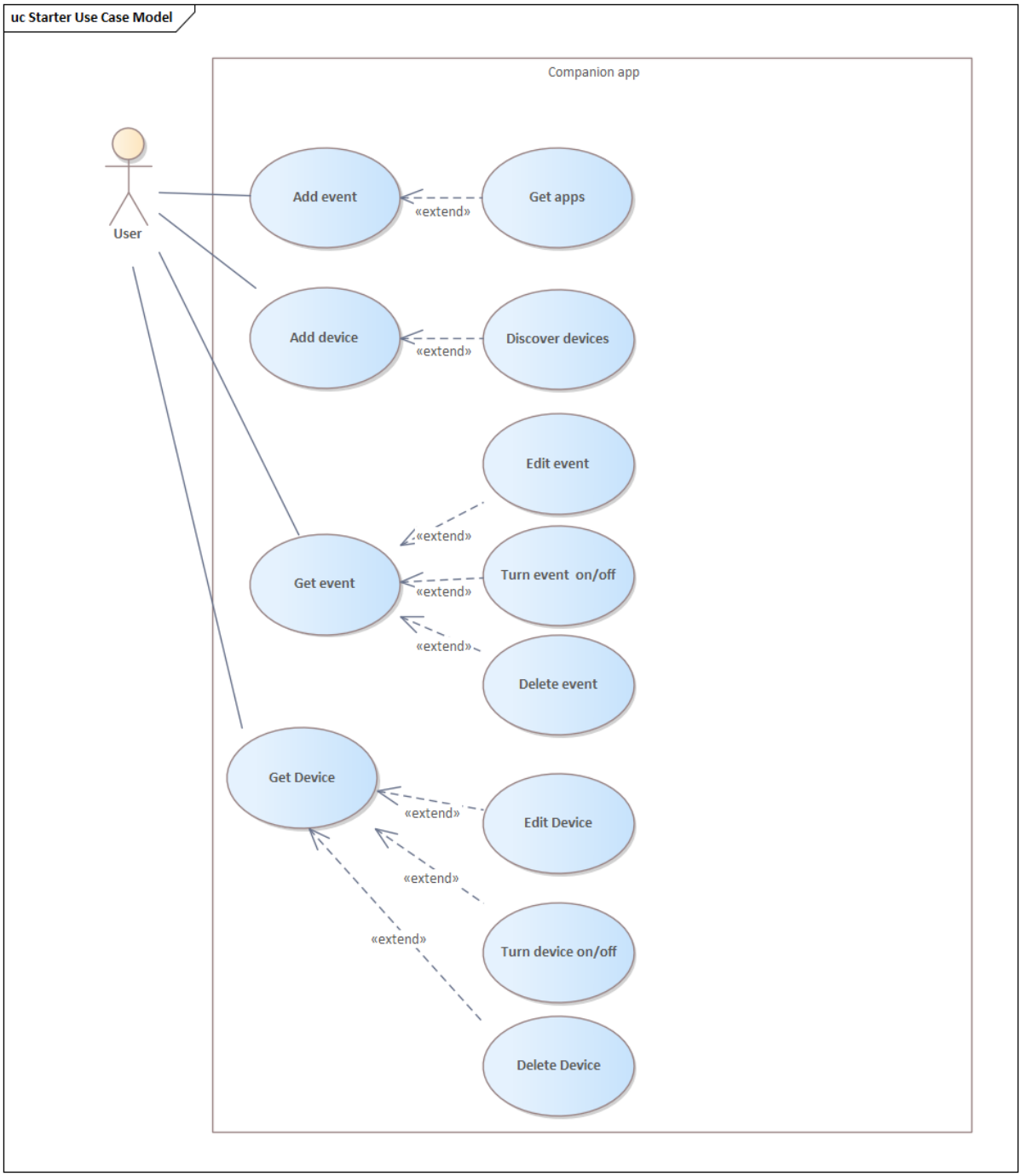
Návrh Aplikácie

Najskôr si užívateľ definuje príležitosti pri ktorých sa budú dáta odosielať. Tu si užívateľ môže definovať aplikácie s ktorých chce dostávať upozornenia. Ďalej užívateľ si musí definovať zariadenie na ktoré chce odosielať. V zariadení si priradí príležitosti ktoré chce pri ktorých chce odosielať správu na zariadenie. Toto všetko je možné upravovať počas behu aplikácie a zapínanie zariadení aj cez widget.

Zariadenia a Eventy je možné upravovať počas behu aplikácie. Keďže používam dve databázy pri úprave jednej sa skontroluje či je potrebné upraviť aj druhú (napr. pri vymazaní Eventu sa event musí vymazať aj so zariadenia).

Aplikácia si zapnutí zistí aktuálne notifikácie a potom svoj zoznam upravuje pri zmene notifikácii (Pridanie alebo vymazanie). Taktiež sa spustí periodické odosielanie správ každú minútu. Samotné odosielanie sa vykoná len v prípade že zapne odosielanie na dané zariadenie a len z eventu ktorý je zapnutý.

Aplikácia nemôže správne fungovať bez potrebných povolení. Preto je užívateľ na začiatku vyzvaný aby udelil povolenia a prístup ku notifikáciám. Ak tak nevykoná nebude možné vstúpiť do aplikácie.



Use Case Diagram

4. Popis implementácie

V mojej Aplikácií využívam:

AndroidX komponenty:

- Room Databázu,
- LifeCycles,
- ViewModel,
- Navigation,
- WorkManager

Externé frameworky / knižnice:

- Jetpack Glance framework na tvorbu widgetu

Komponenti:

- Service,
- Broadcast Receiver,

Sieťová komunikácia

- Bluetooth

Ukážky kódov

```
Radovan Findra
class BluetoothService(
    private val context: Context,
) {

    private val bluetoothAdapter: BluetoothAdapter? = BluetoothAdapter.getDefaultAdapter()
    private val bluetoothDevices = mutableListOf<BluetoothDevice>()
    private val bluetoothSockets = mutableListOf<BluetoothSocket>()
    private val outputStreams = mutableListOf<OutputStream>()
    private val connectionStatus = mutableMapOf<String, Boolean>()
    private val MY_UUID = UUID.fromString( name: "00001101-0000-1000-8000-00805F9B34FB")
```

Použite Bluetooth

```
Radovan Findra
@Database(entities = [DeviceItem::class], version = 1, exportSchema = false)
@TypeConverters(Converters::class)
abstract class DeviceItemsDatabase : RoomDatabase() {

    Radovan Findra
    abstract fun deviceItemDao(): DeviceItemDao

    Radovan Findra
    companion object {
        @Volatile
        private var Instance: DeviceItemsDatabase? = null

        Radovan Findra
        fun getDatabase(context: Context): DeviceItemsDatabase {
            return Instance ?: synchronized( lock: this) {
                Room.databaseBuilder(context, DeviceItemsDatabase::class.java, name: "deviceItems_database") Builder<DeviceItemsDatabase>
                    .fallbackToDestructiveMigration()
                    .build() DeviceItemsDatabase
                    .also { Instance = it }
            }
        }
    }
}
```

Použite Room databáza

```
        <receiver android:name=".ui.widget.CompanionWidgedReciever"
            android:exported="true">
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data
                android:name="android.appwidget.provider"
                android:resource="@xml/my_app_widget_info" />
        </receiver>

        <service
            android:name=".Notification.NotificationListener"
            android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE"
            android:exported="false">
            <intent-filter>
                <action android:name="android.service.notification.NotificationListenerService" />
            </intent-filter>
        </service>
    </application>
```

Použitie Service, Broadcast Reciever

```
Radovan Findra *
class CompanionWidget : GlanceAppWidget() {

    Radovan Findra *
    override suspend fun provideGlance(context: Context, id: GlanceId) {
        provideContent {
            WidgetBody(
                context = context,
                onDeviceSwitch = { deviceId -> changeEnableState(context, deviceId) }
            )
        }
    }

    Radovan Findra
    private fun changeEnableState(context: Context, deviceId: Int) {
        val workRequest = OneTimeWorkRequestBuilder<ChangeDeviceStateWorker>()
            .setInputData(workDataOf( ...pairs: "device_id" to deviceId))
            .build()
        WorkManager.getInstance(context).enqueue(workRequest)
    }
}
```

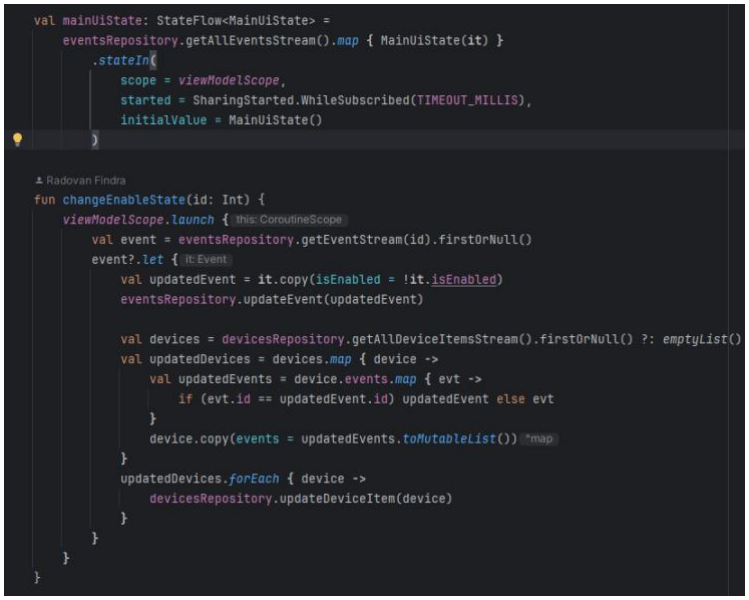
Použitie Glance

```
Radovan Findra
class EventEditViewModel(
    savedStateHandle: SavedStateHandle,
    private val deviceItemsRepository: DeviceItemsRepository,
    private val eventsRepository: EventsRepository,
    installedAppsGetter: InstalledAppsGetter
) : ViewModel() {

    private val eventId: Int = checkNotNull(savedStateHandle[EventEditScreenDestination.eventIdArg])

    private val _eventEditUiState = MutableStateFlow(EventEditUiState())
    val eventEditUiState: StateFlow<EventEditUiState> = _eventEditUiState.asStateFlow()

    Radovan Findra
    init {
        viewModelScope.launch { this: CoroutineScope
            val eventDetails = eventsRepository.getEventStream(eventId) Flow<Event?>
                .filterNotNull() Flow<Event>
                .first() Event
                .toEventDetails()
            val apps = installedAppsGetter.getInstalledApps()
            _eventEditUiState.value = EventEditUiState(eventDetails = eventDetails, apps = apps)
        }
    }
```

Použitie ViewModel

```
Radovan Findra *
@Composable
fun CompanionNavHost(
    navController: NavHostController,
    modifier: Modifier = Modifier
) {
    NavHost(
        navController,
        startDestination = MainEventScreenDestination.route,
        modifier = modifier,
        enterTransition = { fadeIn(animationSpec = tween( durationMillis: 300)) }
    ) { this: NavGraphBuilder
        composable(MainEventScreenDestination.route) { this: AnimatedContentScope  it: NavBackStackEntry
            MainEventScreen(
                navController = navController,
                navigateToEventAdd = { navController.navigate(EventAddDestination.route) },
                navigateToEventEdit = { navController.navigate( route: "${EventEditScreenDestination.route}/$it") }
            )
        }
        composable(EventAddDestination.route) { this: AnimatedContentScope  it: NavBackStackEntry
```

Použitie Navigation



```
val mainUiState: StateFlow<MainUiState> =
    eventsRepository.getAllEventsStream().map { MainUiState(it) }
        .stateIn(
            scope = viewModelScope,
            started = SharingStarted.WhileSubscribed(TIMEOUT_MILLIS),
            initialValue = MainUiState()
        )

Radovan Findra
fun changeEnableState(id: Int) {
    viewModelScope.launch { this: CoroutineScope
        val event = eventsRepository.getEventStream(id).firstOrNull()
        event?.let { it: Event
            val updatedEvent = it.copy(isEnabled = !it.isEnabled)
            eventsRepository.updateEvent(updatedEvent)

            val devices = devicesRepository.getAllDeviceItemsStream().firstOrNull() ?: emptyList()
            val updatedDevices = devices.map { device ->
                val updatedEvents = device.events.map { evt ->
                    if (evt.id == updatedEvent.id) updatedEvent else evt
                }
                device.copy(events = updatedEvents.toMutableList()) *map
            }
            updatedDevices.forEach { device ->
                devicesRepository.updateDeviceItem(device)
            }
        }
    }
}
```

Použite LifeCycles

```
Radovan Findra
class ChangeDeviceStateWorker(
    context: Context,
    workerParams: WorkerParameters
) : CoroutineWorker(context, workerParams) {

    Radovan Findra
    override suspend fun doWork(): Result {
        val appContainer = AppDataContainer(applicationContext)
        val devicesRepository = appContainer.deviceItemsRepository

        val deviceId = inputData.getInt( key: "device_id",  defaultValue: -1)
        if (deviceId == -1) return Result.failure()

        val deviceItem = devicesRepository.getDeviceItemStream(deviceId).firstOrNull() ?: return Result.failure()
        val updatedDeviceItem = deviceItem.copy(isEnabled = !deviceItem.isEnabled)
        devicesRepository.updateDeviceItem(updatedDeviceItem)

        // Optionally, refresh the stored data
        val deviceItems = devicesRepository.getAllDeviceItemsStream().firstOrNull() ?: emptyList()
        saveDeviceItemsToPreferences(deviceItems)

        return Result.success()
    }
```

Použite WorkerManager

5. Záver

Počes implementácie mojej aplikácie som sa naučil pracovať v programovacom jazyku Kotiln a Jetpack Compose. Taktiež aj s základnými AndroidX komponentami.

Moja aplikácia spĺňa moje očakávania a ciele a využíva komponenty ktoré som. Sa naučil na cvičeniach.

V budúcnosti budem moju aplikáciu aktualizovať a funkčnosť ktorú som nestihol naprogramovať. Ako sú: Presety, a reciever správ s Bluetooth zariadení.

Moja aplikácia poskytuje základ pre open source Companion aplikácie. Taktiež pre môj osobný projekt ktorý som spomínal v úvode.

Zoznam použitých zdrojov

https://stackoverflow.com/

https://www.geeksforgeeks.org/

https://www.youtube.com/@PhilippLackner