ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Semestrálna Práca

Mobilná aplikácia pre školský informačný systém

Marek Tvrdošinský

Obsah

1 ÚVOD

V tejto dokumentácií poskytnem prehľad o mobilnej aplikácie, ktorá je navrhnutá tak aby fungovala s informačným systémom pre malé školské inštitúcie, ktorý bol vytvorené ako súčasť mojej bakalárskej práce.

2 POPIS PROBLÉMU

V rámci bakalárskej práce bol vyvinutý webový informačný systém pre správu malých školských inštitúcií. Aplikácia, ktorá je predmetom tejto semestrálnej práce, rozširuje dostupnosť a funkčnosť tohto systému o mobilnú platformu. Cieľom aplikácie je zjednodušiť prístup pre rodičov a žiakov k dôležitým informáciám ako sú rozvrhy, známky a dochádzka, bez nutnosti otvárať webový prehliadač.

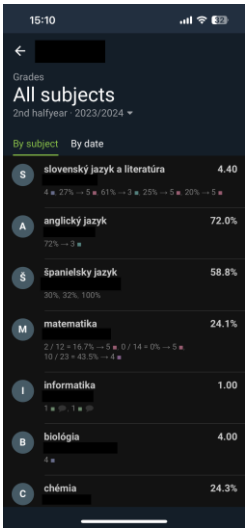
3 EXISTUJÚCE RIEŠENIA

Ako aj moja bakalárska práca tak aj táto aplikácia je inšpirované systémom EduPage.

3.1 EduPage

EduPage je školský informačný systém, ktorý ponúka množstvo nástrojov na zjednodušenie každodennej práce v škole. Umožňuje učiteľom, rodičom a žiakom ľahko sledovať, ako si žiaci vedú v škole. Pre túto aplikáciu som vybral iba potrebné funkcie daného systému, ktorými následne sa inšpirujem pri tvorbe.

Vybrané funkcie sú:

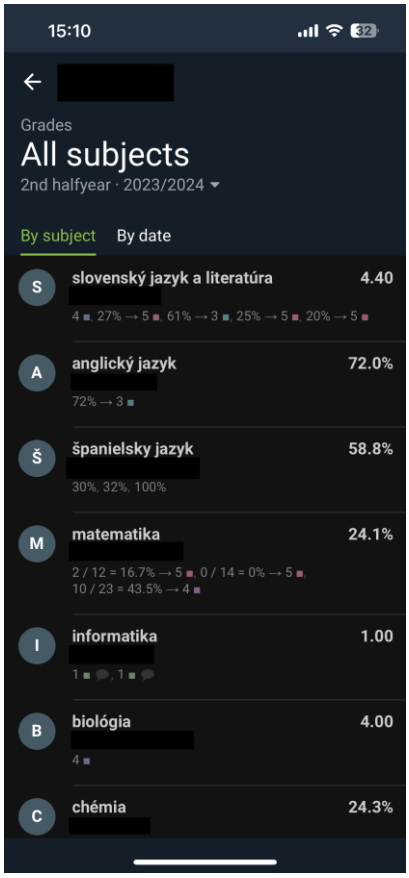


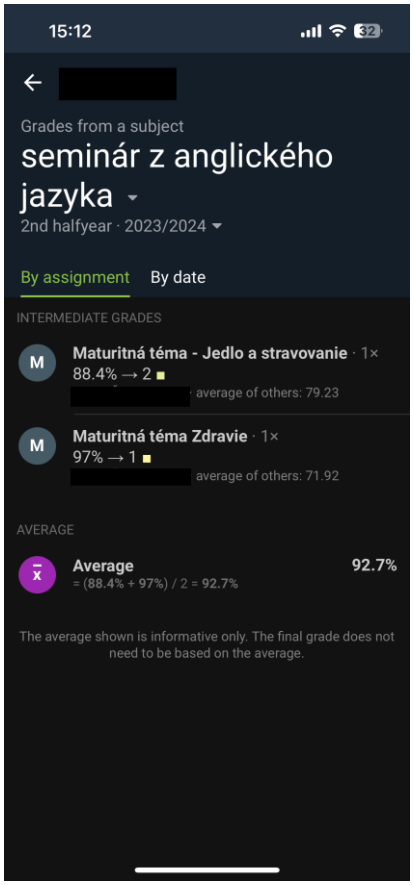
3.1.1 Zobrazenie známok žiakov

Aplikácie podporuje zobrazenie známok pre jednotlivých žiakov, poprípade pre rodičov pre ich detí.

Známky v rámci predmetu môžu byť rozdelené do kategórií, ako napríklad skúšanie, testy alebo prezentácie, pričom každá kategória má priradenú určitú váhu, ktorá ovplyvňuje výsledné hodnotenie. Známky nemusia byť len celočíselné (od 1 do 5), ale môžu byť aj v bodoch, percentách alebo ako slovné hodnotenia, čo je obzvlášť vhodné pre mladšie ročníky.

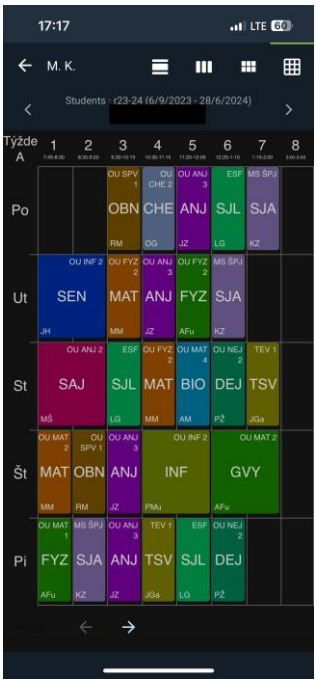
Z pohľadu žiaka alebo rodiča je možné vidieť všetky známky, ktoré boli žiakovi pridelené vo všetkých predmetoch. Systém navyše umožňuje "podpísanie" známok, čo indikuje potvrdenie ich videnia, podobne ako v tradičnej žiackej knižke.





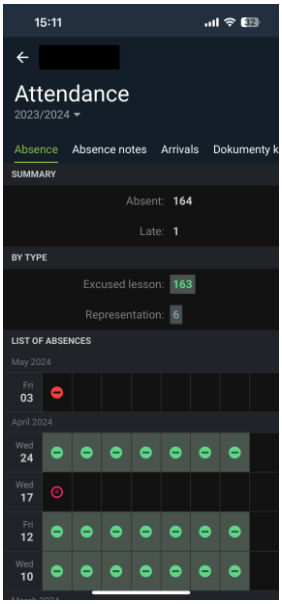
3.1.2 Rozvrh

Systém podporuje zobrazenie rozvrhu pre jednotlivého žiaka v danom školskom období. Okrem zobrazeniu celého rozvrhu podporuje zobrazenie rozvrhu pre aktuálny vyučovací deň.



3.1.3 Dochádzka

Pre dochádzku dovoľuje rodičom ale aj jednotlivým žiakom, dovoľuje zobrazovať jednotlivé absencie žiaka rozdelené na jednotlivé dni a vyučovacie bloky v rozvrhu

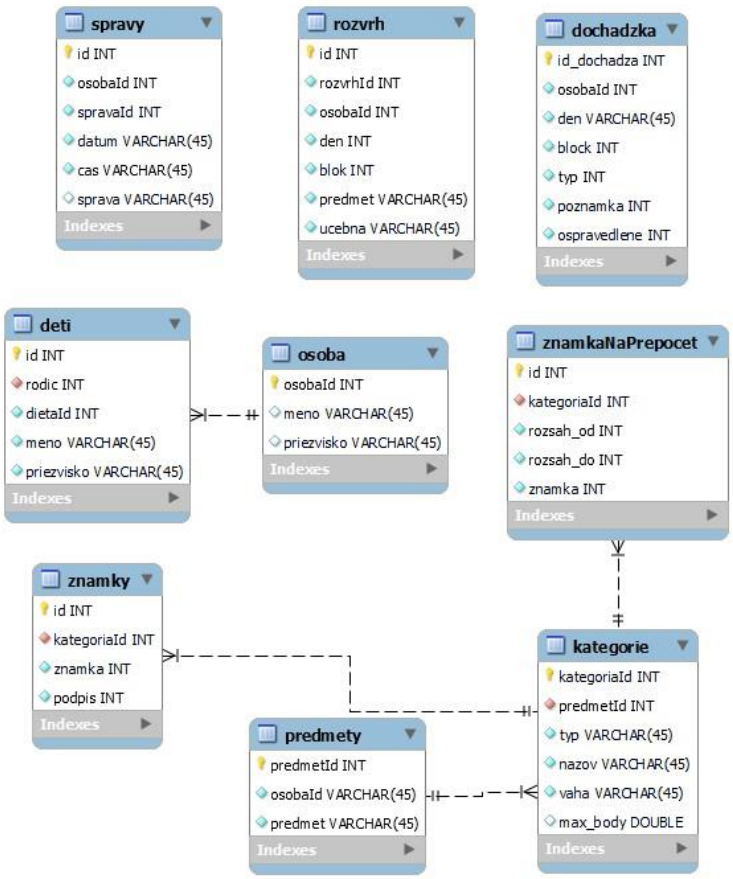


4 NÁVRH APLIKÁCIE

4.1 Databáza

Základom každého informačného systému je jeho databáza. V prípade môjho školského informačného systému predstavuje databáza kľúčový komponent, ktorý umožňuje efektívne uchovávanie a spracovávanie dát nevyhnutných pre jeho správne fungovanie.

V tejto sekcií podrobne predstavím architektúru mojej databázy. Popíšem jednotlivé tabuľky, ich účel a ich vzájomné prepojenia s ostatnými tabuľkami, aby som poskytol komplexný pohľad na organizáciu dát.



4.1.1 Tabuľka o užívateľoch

Na tieto účely slúžia tabuľky 'osoba' a 'deti'. V tabuľke 'osoba' je vždy zaznamenaný prihlásený užívateľ. Pokiaľ prihlásený užívateľ je rodič tak jeho deti sú zaznamenané v tabuľke 'deti'. Pokiaľ je po prihlásení na načítaní údajov zo servera tabuľka deti prázdna, tak sa v celom systéme bude používať 'osobaId' z tejto tabuľky, inak sa bude využívať 'osobaId' z prvého možného záznamu v tabuľke 'deti'

4.1.2 Tabuľka o známkach žiaka

Po úspešnom prihlásení a určeniu o akého užívateľa ide, sa následne získajú informácie zo servera. Tabuľky sú rozdelené tak aby reprezentovali jednotlivé predmety žiaka a jeho známky. Kde každý predmet má svoje kategórie pre známky a každá známka je v nejakej kategórií. Kategórie nemusia uchovávať známky ktoré sú celočíselné (od 1 do 5), ale môžu byť aj v bodoch, percentách. Na tieto ostatné typy slúži tabuľka 'znamkaNaPrepocet ' ktorá uchováva pre danú kategóriu rozsahy podľa ktorých sa dá zistiť celočíselná známka.

4.1.3 Ostatné tabuľky

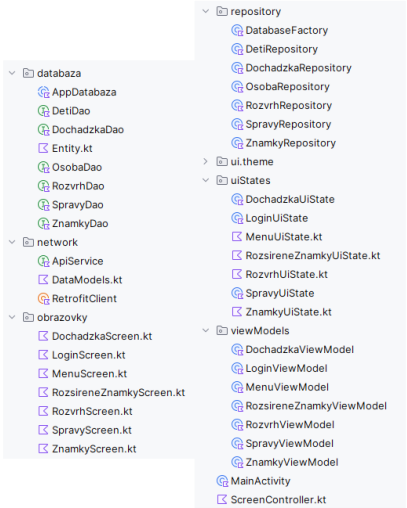
Tieto tabuľky obsahujú zvyšné potrebné záznamy ktoré aplikácia môže zobraziť. Nachádza sa tu tabuľka 'spravy ' ktorá slúži na archivovanie informácií o zmenách v známkach alebo v dochádzke žiaka.

Následne tabuľka 'rozvrh' ktorá zaznamenáva všetky plánované vyučovacie hodiny pre jednotlivého žiaka rozdelené do jednotlivých učebných blokov podľa dňa v týždni.

A na záver tabuľka 'dochadzka ' umožňuje monitorovanie účasti žiakov na jednotlivých vyučovacích hodinách. Záznamy v nej sú uložené podľa jednotlivých dní a blokov v danom vyučovacom dni. Zaznamenáva rôzne druhy absencie ale aj to že či jednotlivé absencie boli ospravedlnené alebo neospravedlnené.

4.2 Aplikácia

Celková architektúra aplikácie je rozdelená do viacerých logických balíkov (packages), ktoré obsahujú súbory databázy, sieťové komponenty, obrazovky, repozitáre a viewModely.

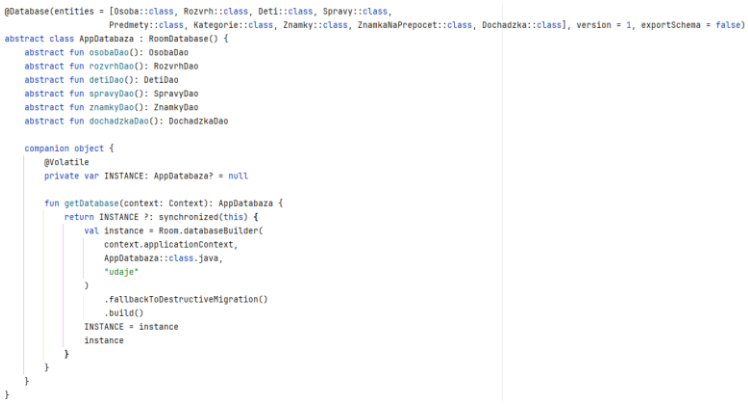


4.2.1 Package Databaza

V tomto balíku sa nachádzajú všetky súbory potrebné na vytvorenie a komunikáciu s databázou o ktorej som bližšie písal vo vyššej uvedenej sekcií.

4.2.1.1 Trieda AppDatabaza

Trieda 'AppDatabaza' je srdcom databázovej vrstvy. Definuje entity a verziu databázy. Zároveň poskytuje metódy na prístup k DAOs (Data Access Objects), ktoré sú zodpovedné za operácie s dátami, ako je ich vkladanie, aktualizácia, mazanie a vyhľadávanie.



```
@Database(entities = [Osoba::class, Rozvrh::class, Deti::class, Spravy::class,
                      Predmety::class, Kategorie::class, Znamky::class, ZnamkaNaPrepocet::class, Dochadzka::class], version = 1, exportSchema = false)
abstract class AppDatabaza : RoomDatabase() {
    abstract fun osobaDao(): OsobaDao
    abstract fun rozvrhDao(): RozvrhDao
    abstract fun detiDao(): DetiDao
    abstract fun spravyDao(): SpravyDao
    abstract fun znamkyDao(): ZnamkyDao
    abstract fun dochadzkaDao(): DochadzkaDao

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

        fun getDatabase(context: Context): AppDatabaza {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabaza::class.java,
                    "udaje"
                )
                    .fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

4.2.1.2 DAO rozhrania

Každá entity má priradené DAO rozhranie, ktoré definuje špecifické operácie možné vykonávať s daným typom entity

```
@Dao
interface OsobaDao {
    @Insert
    suspend fun insertOsoba(osoba: Osoba) : Long

    @Delete
    suspend fun deleteOsoba(osoba: Osoba)

    @Query("SELECT * FROM osoba")
    suspend fun getAllOsoby(): List<Osoba>

    @Query("SELECT * FROM osoba ORDER BY osobaId DESC LIMIT 1")
    suspend fun getLatestUser(): Osoba?
}
```

4.2.2 Package Network

Tento balík obsahuje všetky potrebné komponenty pre sieťovú komunikáciu aplikácie s webovým serverom.

4.2.2.1 RetrofitClient

Je konfiguračný súbor pre Retrofit knižnicu, ktorá je využívaná na správu HTTP požiadaviek. Udržuje základnú URL adresu a inicializuje Retrofit s príslušným konvertorom na prijatie JSON odpovedí zo servera. Pre správnu komunikáciu aplikácie so serverom musí byť užívateľ pripojený k VPN serveru na ktorom funguje server. Z dôvodu lebo server sa nenachádza na verejnej adrese

```
object RetrofitClient {
    private const val BASE_URL = "http://192.168.191.76:8888/"

    private val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl(BASE_URL) Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val apiService: ApiService by lazy {
        retrofit.create(ApiService::class.java)
    }
}
```

4.2.2.2 DataModels

Obsahuje všetky dátové modely ktoré predstavujú štruktúru dát, s ktorými aplikácia pracuje na správnu komunikáciu a mapovanie odpovedí zo servera

```
data class KategorieAllid(val list: List<kategorieZnamky>)

data class PoziadavkyNaPodpisZnamok(val kategorieAllid: List<Int>, val userId: Int)

data class Dochadzka(val list: List<dochadzkaDen>)

data class kategorieZnamky(
    val kategoria_id: Int,
    val znamka: Int,
    val znamka_od: Int,
    val znamka_do: Int
)

data class RozvrhTuple(
    val rozvrhId: Int,
    val osobaId: Int,
    val den: Int,
    val blok: Int,
    val predmet: String,
    val trieda: String
)
```

4.2.2.3 ApiService

Tento súbor definuje všetky API volania, ktoré mobilná aplikácia využíva na komunikáciu so serverom. Funkcie pracujú so špecifickými dátovými modelmi a umožňujú asynchrónne vykonávanie sieťových požiadaviek.

```
interface ApiService {
    @POST("Mobilelogin")
    suspend fun loginUser(@Body loginData: LoginData): LoginResponse

    @POST("MobileGetMeno")
    suspend fun getUserName(@Body userId: UserId): UserName
```

4.2.3 Package Repository

V tomto balíku sa nachádzajú súbory, ktoré slúžia ako úložisko dát pre aplikáciu. Tieto súbory obsahujú metódy pre prácu s databázou, vytváranie požiadaviek, ich spracovanie a správu dát pre rôzne časti aplikácie.

4.2.3.1 DatabaseFactory

Táto trieda zabezpečuje, že každý ViewModel má prístup k potrebným repozitárom.

```
class DatabaseFactory(
    private val osobaRepository: OsobaRepository,
    private val rozvrhRepository: RozvrhRepository,
    private val detiRepository: DetiRepository,
    private val spravyRepository: SpravyRepository,
    private val znamkyRepository: ZnamkyRepository,
    private val dochadzkaRepository: DochadzkaRepository
) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return when {
            modelClass.isAssignableFrom(LoginViewModel::class.java) -> {
                LoginViewModel(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository) as T
            }
            modelClass.isAssignableFrom(MenuViewModel::class.java) -> {
                MenuViewModel(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository) as T
            }
            modelClass.isAssignableFrom(SpravyViewModel::class.java) -> {
                SpravyViewModel(spravyRepository) as T
            }
            modelClass.isAssignableFrom(ZnamkyViewModel::class.java) -> {
                val menuViewModel = MenuViewModel(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository)
                ZnamkyViewModel(znamkyRepository, menuViewModel) as T
            }
            modelClass.isAssignableFrom(RozsireneZnamkyViewModel::class.java) -> {
                val menuViewModel = MenuViewModel(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository)
                RozsireneZnamkyViewModel(znamkyRepository, menuViewModel) as T
            }
            modelClass.isAssignableFrom(RozvrhViewModel::class.java) -> {
                RozvrhViewModel(rozvrhRepository) as T
            }
            modelClass.isAssignableFrom(DochadzkaViewModel::class.java) -> {
                DochadzkaViewModel(dochadzkaRepository) as T
            }
            else -> throw IllegalArgumentException("Unknown ViewModel class")
        }
    }
}
```

4.2.3.2 Repository

Každý repozitár obsahuje potrebné operácie súvisiace na správu svojej tabuľky v databáze, ako sú napríklad operácie vkladania, mazania, a získavania záznamov.

```
class DochadzkaRepository(private val dochadzkaDao: DochadzkaDao) {

    suspend fun insertdochadzka(dochadzka: Dochadzka) {
        dochadzkaDao.insertDochadzka(dochadzka)
    }

    suspend fun getDochadzka(id: Int): List <Dochadzka> {
        return dochadzkaDao.getDochadzka(id)
    }

    suspend fun deleteAllDochadzka(id: Int) {
        dochadzkaDao.deleteAllDochadzka(id)
    }
}
```

4.2.4 Package UiStates

V tejto sekcii sa nachádzajú súbory, ktoré sú zodpovedné za správu stavu používateľského rozhrania v aplikácii. Tieto súbory sú dôležité pre správnu funkčnosť aplikácie, pretože uchovávajú informácie o aktuálnom stave používateľských rozhraní.

```
data class DochadzkaUiState (
    val selectUser: Int,
    val dochadzka: List<dochadzkaDen>,
    val celkoveChybanie: Int,
    val pocetOspravedlnenych: Int,
    val pocetNeospravedlnenych: Int
)
```

.

4.2.5 Package ViewModels

Tento balík obsahuje súbory tried, ktoré sú kľúčovou súčasťou architektúry aplikácie a zabezpečujú oddelenie aplikačnej logiky od užívateľského rozhrania. Každý jednotlivý súbor obsahuje operácie nevyhnutné na prácu s dátami do alebo z databázy a ich následnú prípravu na zobrazenie.

4.2.5.1 MenuViewModel

Pre príklad uvediem túto triedu, ktorá zabezpečuje načítanie a údajov do databázy po úspešnom prihlásení alebo pri spustení aplikácie na aktualizáciu dát. O túto úlohu sa stará funkcia 'LoadData'. Ktorá si vyžiada potrebné dáta od servera a následne ich porovná s dátami uložených v databáze a po prípadne nezhodách s novými dátami aktualizuje lokálnu databázu o nové dáta. Na záver ešte zabezpečí aby nové dáta boli aj uložené v UiState pomocou ktoré sa zobrazujú informácie na obrazovke.

```
suspend fun LoadData(context: Context) {
    if (_uiState.value.reload) {
        resetUiState()
        _uiState.update { it.copy(reload = false) }
        val (existRozvrh, existSpravy) = DatabaseRequest()
        val deti = detiRepository.getAllDeti()
        if (deti.isNotEmpty()) {
            _uiState.update { it.copy(selectUser = deti.first().dietaId, zoznamDeti = deti) }
            setBlokovRozvrhu()
            deti.forEach { item ->
                val (rozvrh, spravy, znamky, dochadzka) = ServerRequest(item.dietaId, context)
                if (rozvrh.isNotEmpty()) {...}
                if (spravy.isNotEmpty()) {...}
                if (dochadzka.isNotEmpty()) {...}
                if (znamky.isNotEmpty()) {...}
            }
        } else {
            val osoba = osobaRepository.jePrihlaseny()
            if (osoba != null) {
                _uiState.update { it.copy(reload = false, selectUser = osoba.osobaId) }
                setBlokovRozvrhu()
                val (rozvrh, spravy, znamky, dochadzka) = ServerRequest(osoba.osobaId, context)
                if (rozvrh.isNotEmpty()) {...}
                if (spravy.isNotEmpty()) {...}
                if (dochadzka.isNotEmpty()) {...}
                if (znamky.isNotEmpty()) {...}
            }
        }
        setBlokovRozvrhu()
    }
}
```

4.2.6 Package Obrazovky

V tomto balíku sa nachádzajú všetky súbory definujúce jednotlivé užívateľské rozhrania aplikácie. Každý súbor reprezentuje jednu obrazovku v aplikácii a obsahuje logiku pre jej vykreslenie a interakciu s užívateľom.

4.2.6.1 MenuScreen

Pre príklad uvediem 'MenuScreen'. Táto obrazovka reprezentuje centrálnu obrazovku aplikácie poskytujúca prístup k rôznym častiam systému ako sú správy, rozvrhy, známky a dochádzka. Okrem hlavného vykresľovanie kódu obsahuje aj funkcie ktoré podporujú zobrazenie užívateľského rozhrania, čím sa s kódu stáva prehľadnejší a jednoduchší na pochope nie jeho častí.

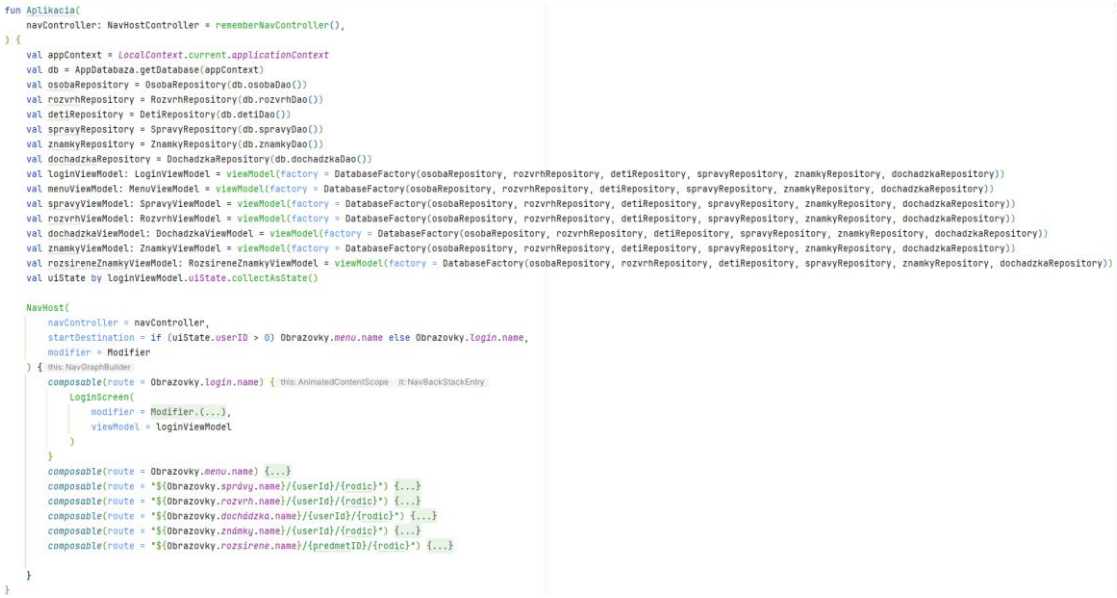
```
@Composable
fun MenuScreen(
    modifier: Modifier = Modifier,
    navController: NavController,
    loginViewModel: LoginViewModel,
    menuViewModel: MenuViewModel
) {
    val uiStatelogin by loginViewModel.uiState.collectAsState()
    val uiStatemenu by menuViewModel.uiState.collectAsState()
    val context = LocalContext.current
    LaunchedEffect(uiStatelogin.userID) {...}

    val configuration = LocalConfiguration.current
    val isPortrait = configuration.orientation == Configuration.ORIENTATION_PORTRAIT
    val gradientColors = listOf(Color( color: 0xFF8ECEC0), Color( color: 0xFFF1F1F1))

    Column(modifier = modifier.(...)) { this: ColumnScope
        Box(
            modifier = Modifier.(...)
        ) { this: BoxScope
            Column {...}
        }
        Column(
            modifier = Modifier.(...),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            val rodic = if (uiStatemenu.zoznamDeti.size > 0) 1 else 0
            if (isPortrait) {
                Row (modifier = Modifier.padding(bottom = 4.dp, top = 20.dp)) {...}
                Row (modifier = Modifier.padding(top = 4.dp)) {...}
            } else {
                Row (modifier = Modifier.padding(top = 30.dp))  {...}
            }
        }
    }
}
```

```
@Composable
fun NavigacneTlacidla(text: String, image: Int, navController: NavController, selectUser: Int, rodic: Int) {
    Button(
        modifier = Modifier
            .size(width = 200.dp, height = 100.dp)
            .padding(1.dp),
        shape = RectangleShape,
        colors = ButtonDefaults.buttonColors(containerColor = Color.White.copy(alpha = 0.1F), contentColor = Color.Black),
        onClick = {
            navController.navigate( route: "${Obrazovky.valueOf(text.toLowerCase()).name}/${selectUser}/${rodic}")
        }
    ) { this: RowScope
        Image(painter = painterResource(id = image), contentDescription = null, modifier = Modifier.width(50.dp))
        Spacer(Modifier.width(5.dp))
        Text(text = text)
    }
}
```

4.2.7 ScreenController

Je to základný stavebný kameň celej architektúry aplikácie. Pomocou tohto súboru aplikácia si pripraví všetky požadované viewmodely ale aj obrazovky.

Obsahuje 'NavHostController' a jeho pomocou je realizovaná navigácia medzi obrazovkami s možnosťou vrátenia sa a prenosu parametrov potrebných a správne zobrazenie informácií v danej obrazovke. Každá obrazovka má definované svoje trasy (routes), ktoré sú pevne stanovené v enume 'Obrazovky'.



```
fun Aplikacia(
    navController: NavHostController = rememberNavController(),
) {
    val appContext = LocalContext.current.applicationContext
    val db = AppDatabaza.getDatabase(appContext)
    val osobaRepository = OsobaRepository(db.osobaDao())
    val rozvrhRepository = RozvrhRepository(db.rozvrhDao())
    val detiRepository = DetiRepository(db.detiDao())
    val spravyRepository = SpravyRepository(db.spravyDao())
    val znamkyRepository = ZnamkyRepository(db.znamkyDao())
    val dochadzkaRepository = DochadzkaRepository(db.dochadzkaDao())
    val loginViewModel: LoginViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val menuViewModel: MenuViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val spravyViewModel: SpravyViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val rozvrhViewModel: RozvrhViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val dochadzkaViewModel: DochadzkaViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val znamkyViewModel: ZnamkyViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val rozsirenеZnamkyViewModel: RozsirenеZnamkyViewModel = viewModel(factory = DatabaseFactory(osobaRepository, rozvrhRepository, detiRepository, spravyRepository, znamkyRepository, dochadzkaRepository))
    val uiState by loginViewModel.uiState.collectAsState()

    NavHost(
        navController = navController,
        startDestination = if (uiState.userID > 0) Obrazovky.menu.name else Obrazovky.login.name,
        modifier = Modifier
    ) { this: NavGraphBuilder
        composable(route = Obrazovky.login.name) { this: AnimatedContentScope  it: NavBackStackEntry
            LoginScreen(
                modifier = Modifier.(...),
                viewModel = loginViewModel
            )
        }
        composable(route = Obrazovky.menu.name) {...}
        composable(route = "${Obrazovky.správy.name}/{userId}/{rodic}") {...}
        composable(route = "${Obrazovky.rozvrh.name}/{userId}/{rodic}") {...}
        composable(route = "${Obrazovky.dochádzka.name}/{userId}/{rodic}") {...}
        composable(route = "${Obrazovky.známky.name}/{userId}/{rodic}") {...}
        composable(route = "${Obrazovky.rozsírene.name}/{predmetID}/{rodic}") {...}

    }
}
```