ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Pizza Pro 2

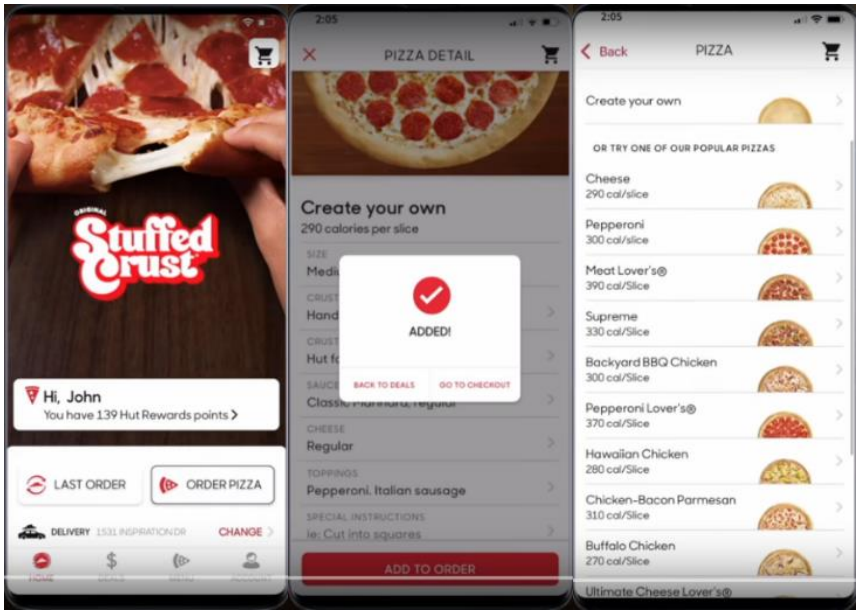
semestrálna práca

Vypracoval: **Erik Mešina**
Študijná skupina: **5ZYR34**
Predmet: **Vývoj aplikácií pre mobilné zariadenia**
Cvičiaci: **doc. Ing. Patrik Hrkút, PhD.**

Obsah

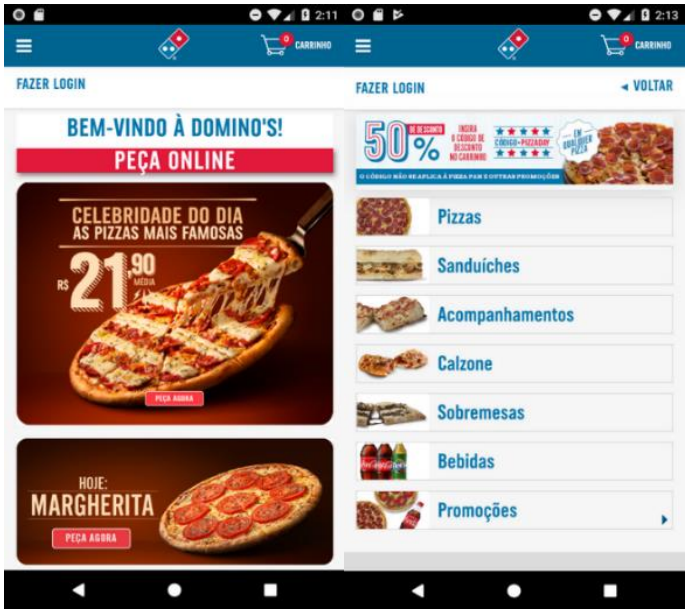
1. Prehľad aplikácií s podobným zameraním

Pizza Hut ponúka najjednoduchší spôsob objednávania svojej obľúbenej pizze, kuracích krídielok, dezertov a ďalších lahôdok. Obsahuje mnohé funkcie ako napríklad selekciu produktov, pridanie príloh a bezkontaktné objednávanie so zaručením rýchleho a bezpečného doručenia.



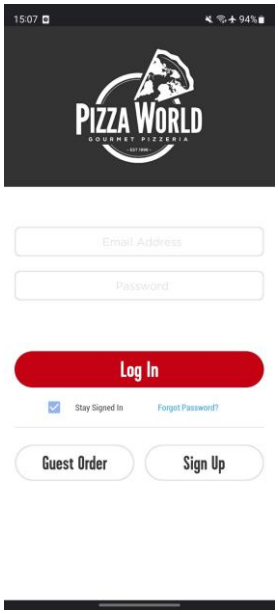
Zdroj: https://apkpure.com/pizza-hut-food-delivery-ta/com.yum.pizzahut

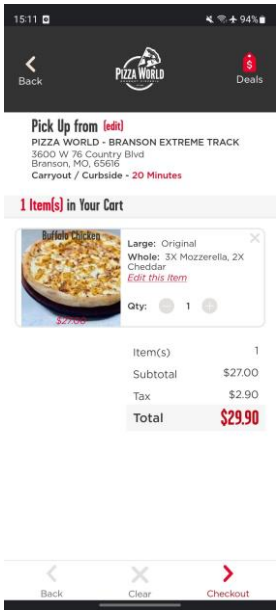
Domino's, jeden z najväčších reťazcov pizze na svete, sprístupňuje väčšinu miest, v ktorých je prítomná služba doručovania. Aplikácia poskytuje ponuku siete pobočiek Domino's a umožňuje priame platby prostredníctvom aplikácie a sledovanie doručovania v reálnom čase.



Zdroj: https://apkpure.com/domino-s-pizza-brasil/br.com.dominos.mobile

Pizza World je obchodným podnikom, ktorý spája vzrušujúci nový koncept gurmánskej pizze svetovej triedy s rýchlosťou, efektivitou a pohodlím tradičných reštaurácií na donášku pizze. Aplikácia taktiež ponúka možnosť zaregistrovania sa, dáva možnosť výberu z viacerých typov jedál ako aj sprístupnenie online platby.





Zdroj: https://apkpure.com/pizza-world/com.hungerrush.pizzaworldusa

2. Analýza navrhovanej aplikácie

Aplikácia pre objednávanie pizze je praktickým a efektívnym spôsobom, ako si zákazníci môžu objednať svoju obľúbenú pizzu z pohodlia svojho domova. Celkovo by mala byť aplikácia pre objednávanie pizze rýchla, spoľahlivá a jednoduchá na používanie. Ide hlavne o dosiahnutie spokojnosti zákazníkov, čo zabezpečí kladné recenzie, opätovné používanie a odporučenie aplikácie.

- **Z hľadiska funkcionality**

  Aplikácia by mala byť pre zákazníkov jednoduchá na používanie a ponúkať všetky potrebné funkcie, ako napríklad prehľad menu, objednávka, spätná väzba,...

- **Z hľadiska používateľského rozhrania**

  Rozhranie aplikácie by malo byť intuitívne, prehľadné a atraktívne pre zákazníkov, aby sa s ňou ľahko pracovalo a zákazníci sa v nej dokázali zorientovať.

- **Z hľadiska spôsobu platby**

  Aplikácia by mala ponúkať rôzne spôsoby platby, aby boli zákazníci pohodlní pri používaní aplikácie. Medzi bežné spôsoby patrí platba kartou, hotovosťou alebo online bankovníctvom.

- **Z hľadiska bezpečnosti**

  Aplikácia by mala mať zabezpečené dáta zákazníkov, aby sa zabránilo akejkoľvek neoprávnenej manipulácii s informáciami.

- **Hodnotenie a spätná väzba**

  Zákazníci by mali mať možnosť hodnotiť aplikáciu a jedlo, čo povedie k neustálej optimalizácii služieb a produktov.

3. Návrh architektúry aplikácie

- **Prehľad menu**

Zákazník by mal byť schopný prehliadať kompletné menu reštaurácie, vrátane cien, popisu a obrázkov. Pizza Pro 2 obsahuje široký sortiment produktov. Pomocou selekcie filtrov si môžete zvoliť presne tú, po ktorej túžite.

- **Objednávanie**

Zákazník by mal mať možnosť vytvoriť si vlastnú objednávku pomocou interaktívneho menu. Pizza Pro 2 má k dispozícii rôzne možnosti pre vytváranie objednávky, ako napríklad voľba miesta doručenia objednávky alebo zmena počtu kusov jednotlivých produktov.

- **Platba viacerými formami**

Aplikácia má za cieľ poskytnúť zákazníkom jednoduchý a bezpečný spôsob platby za ich objednávky. Umožňuje výber z rôznych možností platby vrátane kreditnej karty, hotovosti alebo kupónov, aby si každý mohol vybrať podľa svojich preferencií.

- **Nastavenia účtu**

Zákazníci by mali mať možnosť vytvoriť si svoj vlastný účet, kde by mohli spravovať svoje osobné údaje ako aj možnosť odhlásenia alebo odstránenia účtu.

- **História používateľov a objednávok**

Aplikácia by mala umožniť zákazníkom jednoducho a efektívne si prehliadať a manažovať svoje objednávky. V prípade nespokojnosti s objednávkou by ju mal byť zákazník schopný zrušiť. Taktiež by mal mať možnosť si prezrieť aktuálne aktívne kontá, tak isto s možnosťou manažmentu, ako predtým.

- **Recenzie a hodnotenia**

Zákazníci by mali mať možnosť zanechať recenzie a hodnotenia na jednotlivé jedlá, aby mohli pomôcť ostatným zákazníkom pri výbere.

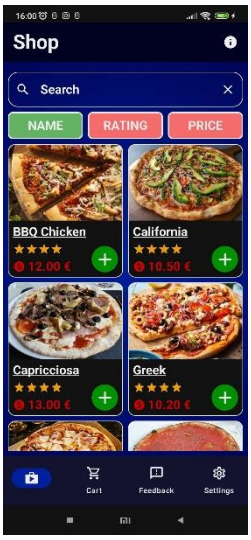
4. Ukážka návrhu obrazoviek aplikácie

Úvodná obrazovka

Registrácia

Prihlásenie

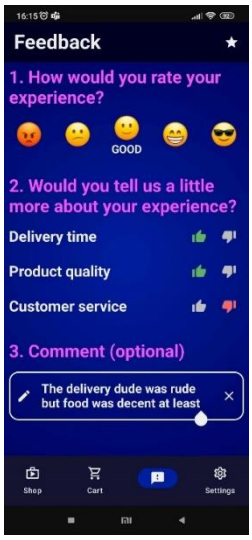
Hlavné menu



Košík zákazníka

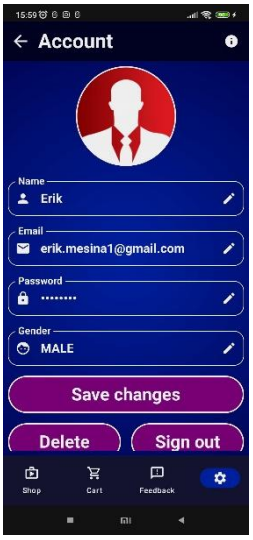
Detail produktu

Spätná väzba



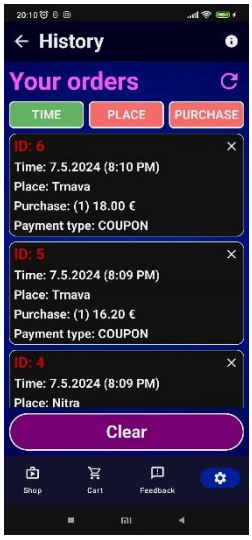
Nastavenia

Nastavenia účtu

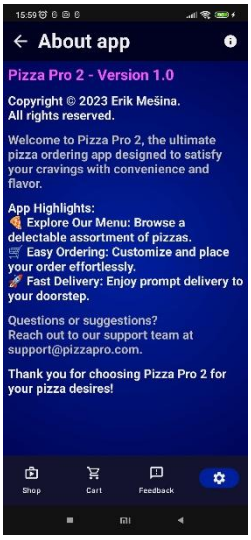


História 1

História 2



O aplikácií



5. Obrazovky

Takmer každej obrazovke prislúcha vlastný stav (niektoré majú zlúčený kvôli vysokej miere podobnosti), spracovateľa udalostí (pri kliknutí na tlačidlo, zmenu hodnôt atribútov,...) a model pohľadu (ViewModel - VM). V niektorých prípadoch bolo nutné pracovať s databázou, preto bol vytvorený poskytovateľ modelu pohľadu (MyViewModelProvider) pre dodatočnú inicializáciu repozitáru.

```
object MyViewModelProvider {

    val factory = viewModelFactory {
        initializer {
            AuthViewModel(myRepository = myApplication().myContainer.myRepository)
        }

        initializer {
            SharedViewModel(myRepository = myApplication().myContainer.myRepository)
        }

        initializer {
            AccountViewModel(myRepository = myApplication().myContainer.myRepository)
        }

        initializer {
            HistoryViewModel(myRepository = myApplication().myContainer.myRepository)
        }
    }

    private fun CreationExtras.myApplication(): MyApplication =
        (this[ViewModelProvider.AndroidViewModelFactory.APPLICATION_KEY] as MyApplication)
}
```

- **IntroScreen** - Úvodná obrazovka, obsahuje len tlačidlo, ktoré používateľa presunie na nasledujúcu obrazovku pre autentifikáciu používateľa.
- **SignUpScreen** - Obrazovka, kde si používateľ môže vytvoriť účet. Pri nevyplnení niektorej z informácií ho na to systém upozorní. Po úspešnom vyplnení všetkých potrebných informácií môže používateľ pokračovať do hlavného menu, čiže obchodu.
- **SignInScreen** - Obrazovka, kde sa používateľ môže prihlásiť do už existujúceho účtu. Pri nevyplnení niektorej z informácií ho na to systém upozorní. Po úspešnom vyplnení všetkých potrebných informácií môže používateľ pokračovať do hlavného menu, čiže obchodu.
- **ShopScreen** - V tejto časti si zákazník môže vyhľadať a vybrať konkrétne produkty, následne pridať do košíka. Ak je zákazník spokojný s výberom, môže pokračovať do košíkovej časti.
- **CartScreen** - V tejto časti si používateľ môže skontrolovať celú objednávku. Na úspešné vykonanie objednávky bude musieť zákazník zadať miesto doručenia. Objednávka je vizuálne potvrdená vyskakujúcim oknom.
- **FeedbackScreen** - Časť aplikácie, kde používateľ môže vyjadriť svoju spokojnosť a podať nápady na zlepšenie.
- **ProfileScreen** - Časť aplikácie, v ktorej sú uvedené informácie o používateľovi zadané pri vytváraní konta. Tu si môže používateľ zmeniť osobné údaje, ako aj odhlásiť sa alebo odstrániť svoj účet.
- **HistoryScreen** - V tejto časti sú uvedené databázy používateľov a objednávok. Používateľ má možnosť si vyhľadať konkrétnych používateľov pomocou zadania mena do vyhľadávania alebo usporiadať si svoje objednávky podľa rôznych kritérií ako napr. cena a meno. Taktiež má možnosť vymazať účty, stornovať objednávky alebo vymazať celé jednotlivé databázy.
- **AboutAppScreen** - Časť aplikácie, v ktorej sú uvedené informácie o aplikácií (verzia, autorské práva, popis aplikácie,...).

6. Navigácia

- **NavGraph**

Hlavný (koreňový) navigačný graf, ktorý pozostáva z menších navigačných grafov a komponentov tvoriace rôzne obrazovky. Navigácia medzi nimi je zabezpečená pomocou navigačného ovládača.

```
@Composable
fun NavGraph(navController: NavHostController) {
    NavHost(
        navController = navController,
        startDestination = GraphRoute.AuthGraph.name,
        route = GraphRoute.RootGraph.name
    ) {
        authNavGraph(navController = navController)
        composable(GraphRoute.HomeGraph.name) {
            HomeScreen()
        }
    }
}
```

- **AuthNavGraph**

Navigačný graf pozostávajúci z úvodných obrazoviek po otvorení aplikácie. Hlavnými obrazovkami sú obrazovka na prihlásenie a registráciu. V oboch prípadoch sa používateľ môže pokúsiť autentifikovať.

```
fun NavGraphBuilder.authNavGraph(navController: NavHostController) {
    navigation(
        startDestination = Screen.Intro.route,
        route = GraphRoute.AuthGraph.name
    ) {
        composable(route = Screen.Intro.route) {
            IntroScreen(navController = navController)
        }
        composable(route = Screen.SignUp.route) {
            SignUpScreen(navController = navController)
        }
        composable(route = Screen.SignIn.route) {
            SignInScreen(navController = navController)
        }
    }
}
```

- **HomeScreen**

Šablónová trieda pre obrazovky hlavnej časti aplikácie (domáce obrazovky), e-shopu. Každá z nich obsahuje hornú a dolnú lištu. Taktiež majú prístup k zdielanému VM a jeho operáciam.

```
@Composable
fun HomeScreen() {
    val sharedViewModel: SharedViewModel = viewModel(factory = MyViewModelProvider.factory)
    val sharedState by sharedViewModel.state.collectAsState()
    val onSharedEvent = sharedViewModel::onEvent
    val navController = rememberNavController()

    Scaffold(
        topBar = {
            TopBar(navController) {
                onSharedEvent(SharedEvent.DialogVisibilityChanged(true))
            }
        },
        bottomBar = {
            BottomBar(navController)
        },
        content = { innerPadding ->
            if (sharedState.isDialogVisible) {
                InfoDialog(
                    titleId = R.string.pizza_info,
                    textId = R.string.pizza_card_info,
                    onDismiss = {
                        onSharedEvent(SharedEvent.DialogVisibilityChanged(false))
                    },
                    dismissButton = R.string.cancel
                )
            }

            Box(modifier = Modifier.padding(innerPadding)) {
                BottomNavGraph(navController, sharedState, onSharedEvent)
            }
        }
    )
}
```

- **SettingsNavGraph**

Jednu z domácich obrazoviek tvorí navigačný graf pre rôzne funkcie poskytnuté v nastaveniach. Patria sem obrazovky pre manažovanie účtu, histórie a zobrazenie základných informácií o aplikácii.

```
fun NavGraphBuilder.settingsNavGraph(navController: NavHostController) {
    navigation(
        startDestination = Screen.Settings.route,
        route = GraphRoute.SettingsGraph.name
    ) {
        composable(route = Screen.Settings.route) {
            SettingsScreen(navController = navController)
        }
        composable(route = Screen.Account.route) {
            AccountScreen()
        }
        composable(route = Screen.History.route) {
            HistoryScreen()
        }
        composable(route = Screen.AboutApp.route) {
            AboutAppScreen()
        }
    }
}
```

- **BottomSheet**

Po kliknutí na obrázok niektorej pizze sa zobrazí dolný panel s jej detailným popisom. Je možné z neho vyčítať rôzne informácie ako hodnotenie, obsah kalórií a podrobný popis zloženia.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BottomSheet(pizza: Pizza, onDismiss: () -> Unit) {
    val viewModel = SheetViewModel(pizza = pizza)
    val state by viewModel.state.collectAsState()
```

7. Room Databáza

Room databáza pre používateľov a objednávky je komponent v Android aplikácii, ktorý umožňuje efektívne uchovávanie a spravovanie údajov. Poskytuje prostredie na efektívne ukladanie, získavanie a aktualizovanie dát v rámci Android aplikácie. Taktiež zabezpečuje, že dáta sú konzistentné, spoľahlivé a je s nimi možné pracovať na hlavnom vlákne alebo asynchrónne podľa potreby.

- **User a Order**

Entity, ktoré uchovávajú základné informácie o korešpondujúcich objektoch. Každý z nich taktiež obsahuje vlastný jedinečný identifikátor (id), vďaka ktorému sa dokážeme vyvarovať duplicitným záznamom. Objednávka navyše obsahuje UID (atribút user) pre určenie, ktorý používateľ zadal danú objednávku.

```kotlin
@Entity(tableName = "users")
data class User(
    @ColumnInfo(name = "name") val name: String,
    @ColumnInfo(name = "email") val email: String,
    @ColumnInfo(name = "password") val password: String,
    @ColumnInfo(name = "gender") val gender: Gender,
    @PrimaryKey(autoGenerate = true) val id: Int = 0
)
```

```kotlin
@Entity(tableName = "orders")
data class Order(
    @ColumnInfo(name = "user") val user: Int,
    @ColumnInfo(name = "time") val time: Long,
    @ColumnInfo(name = "place") val place: String,
    @ColumnInfo(name = "items") val items: Int,
    @ColumnInfo(name = "cost") val cost: Double,
    @ColumnInfo(name = "payment") val payment: String,
    @PrimaryKey(autoGenerate = true) val id: Int = 0
)
```

- **MyDatabase**

Abstraktná trieda, ktorá deklaruje tabuľky v databáze. Tabuľky obsahujú stĺpce, ktoré reprezentujú jednotlivé údaje z entít User a Order. Obsahuje jedinú metódu, ktorá slúži na vrátenie už existujúcej alebo, v prípade neexistencie databázy, vytvorenej novej inštancie databázy.

```kotlin
@Database(entities = [User::class, Order::class], version = 2, exportSchema = false)
abstract class MyDatabase : RoomDatabase() {

    abstract val myDao: MyDao

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

        fun getInstance(context: Context): MyDatabase {
            return INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context = context.applicationContext,
                    klass = MyDatabase::class.java,
                    name = "my_database"
                ).fallbackToDestructiveMigration().build().also { INSTANCE = it }
            }
        }
    }
}
```

- **MyDao**

Rozhranie, ktoré obsahuje SQL príkazy pre prístup a manipuláciu s údajmi v databáze. Obsahuje metódy na vkladanie, aktualizovanie a odstránenie (používateľa, objednávky), odstránenie všetkých používateľov a objednávok v databáze, filtrovanie používateľov a objednávok a metódy na prístup ku všetkým používateľom a objednávkam v databáze.

```
@Dao
interface MyDao {

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertOrder(order: Order)

    @Delete
    suspend fun deleteOrder(order: Order)

    @Transaction
    @Query("DELETE FROM users")
    suspend fun deleteAllUsers()

    @Transaction
    @Query("DELETE FROM orders WHERE user = :user")
    suspend fun deleteUsersOrders(user: Int)
```

```
    @Transaction
    @Query("SELECT * FROM users WHERE id = :id OR name = :name OR email = :email LIMIT 1")
    fun getUser(id: Int = -1, name: String = "", email: String = ""): Flow<User?>

    @Transaction
    @Query("SELECT * FROM users WHERE name LIKE '%' || :regex || '%' ORDER BY id ASC")
    fun getUsers(regex: String = ""): Flow<List<User>>

    @Transaction
    @Query("SELECT * FROM orders WHERE user = :user ORDER BY time DESC")
    fun getOrdersBasedOnTime(user: Int): Flow<List<Order>>

    @Transaction
    @Query("SELECT * FROM orders WHERE user = :user ORDER BY cost ASC")
    fun getOrdersBasedOnCost(user: Int): Flow<List<Order>>

    @Transaction
    @Query("SELECT * FROM orders WHERE user = :user ORDER BY place ASC")
    fun getOrdersBasedOnPlace(user: Int): Flow<List<Order>>
}
```

- **MyRepository**

Trieda zodpovedná za správu interakcie medzi aplikáciou a databázou. Obsahuje metódy na manipuláciu s dátami používateľov a objednávok v databáze. Pri inicializácii prijíma databázový prístupový objekt (myDao), ktorý poskytuje prístup k databáze.

Okrem metód na manipuláciu s databázou trieda MyRepository obsahuje aj dve premenné, ktoré sú dôležité pre funkcionalitu aplikácie. Premenná currentUser reprezentuje aktuálne prihláseného používateľa a slúži ako hlavný zdroj informácií o jeho údajoch. Premenná allUsers poskytuje zoznam všetkých používateľov v databáze a je využívaná vo viacerých častiach aplikácie, ako napríklad pri autorizácii používateľa alebo pri editovaní používateľského konta. Tieto premenné sú kľúčové pre efektívnu správu používateľských účtov a dát v aplikácii.

```
class MyRepository(private val myDao: MyDao) {
    var currentUser: Flow<User?> = flowOf(null)
    val allUsers: Flow<List<User>> = myDao.getUsers()

    suspend fun insertUser(user: User) = myDao.insertUser(user)

    suspend fun updateUser(user: User) = myDao.updateUser(user)

    suspend fun deleteUser(user: User) = myDao.deleteUser(user)

    suspend fun insertOrder(order: Order) = myDao.insertOrder(order)

    suspend fun deleteOrder(order: Order) = myDao.deleteOrder(order)

    suspend fun deleteAllUsers() = myDao.deleteAllUsers()

    suspend fun deleteUsersOrders(user: Int) = myDao.deleteUsersOrders(user)
```

```
    suspend fun deleteUsersOrders(user: Int) = myDao.deleteUsersOrders(user)

    fun setCurrentUser(id: Int = -1, name: String = "", email: String = "") {
        currentUser = myDao.getUser(id, name, email)
    }

    fun getUsers(regex: String = ""): Flow<List<User>> = myDao.getUsers(regex)

    fun getOrders(user: Int, orderSortType: OrderSortType): Flow<List<Order>> {
        return when (orderSortType) {
            OrderSortType.TIME -> myDao.getOrdersBasedOnTime(user)
            OrderSortType.PLACE -> myDao.getOrdersBasedOnPlace(user)
            OrderSortType.PURCHASE -> myDao.getOrdersBasedOnCost(user)
        }
    }
}
```

- **MyContainer**

Celkovo táto trieda slúži na poskytnutie jednoduchého prístupu k databázovej vrstve aplikácie tým, že vytvorí a inicializuje inštanciu triedy MyRepository s potrebnými závislosťami.

```
class MyContainer(private val context: Context) {

    val myRepository: MyRepository by lazy {
        MyRepository(MyDatabase.getInstance(context = context).myDao)
    }
}
```

- **MyApplication**

Celkovým účelom tejto triedy je zabezpečiť, že kontajner pre prístup k databáze je inicializovaný v celej aplikácii a je dostupný cez inštanciu triedy MyApplication. Tento prístup umožňuje jednoduchý a globálny prístup k databázovým operáciám z rôznych častí aplikácie.

```
class MyApplication: Application() {

    lateinit var myContainer: MyContainer

    override fun onCreate() {
        super.onCreate()
        myContainer = MyContainer(this)
    }
}
```

Použité zdroje

- https://developer.android.com/courses/android-basics-compose/course (MVP)
- https://www.youtube.com/@PhilippLackner (MVP)
- https://www.geeksforgeeks.org/room-database-with-kotlin-coroutines-in-android/
- https://www.ezcater.com/lunchrush/office/most-popular-types-of-pizza-around-country/
- https://www.tasteatlas.com/50-most-popular-pizzas-in-the-world
- https://www.tasteatlas.com/pizzas
- https://www.fileformat.info/info/unicode/char/search.htm
- https://www.javatpoint.com/kotlin-android-alertdialog