



ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia  
a informatiky

# **Pizza Pro**

## semestrálna práca

Vypracoval: **Erik Mešina**

Študijná skupina: **5ZYR34**

Predmet: **Vývoj aplikácií pre mobilné zariadenia**

Cvičiaci: **doc. Ing. Patrik Hrkút, PhD.**

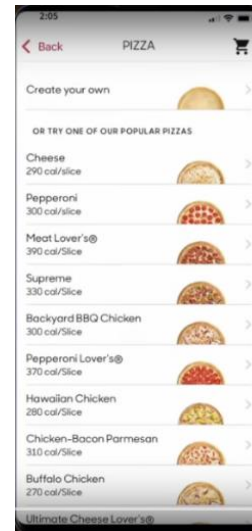
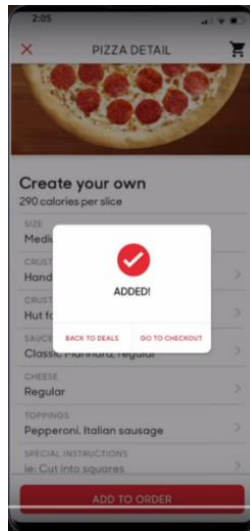
## Obsah

1. [Prehľad aplikácií s podobným zameraním](#)
2. [Analýza navrhovanej aplikácie](#)
3. [Návrh architektúry aplikácie](#)
4. [Ukážka návrhu obrazoviek aplikácie](#)
5. [Fragmenty](#)
6. [Navigačný graf](#)
7. [Adaptéry](#)
8. [Nastavenia - menu](#)
9. [Room databáza](#)
10. [Použité zdroje](#)

## Prehľad aplikácií s podobným zameraním

### Pizza Hut

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 Pizza Brasil

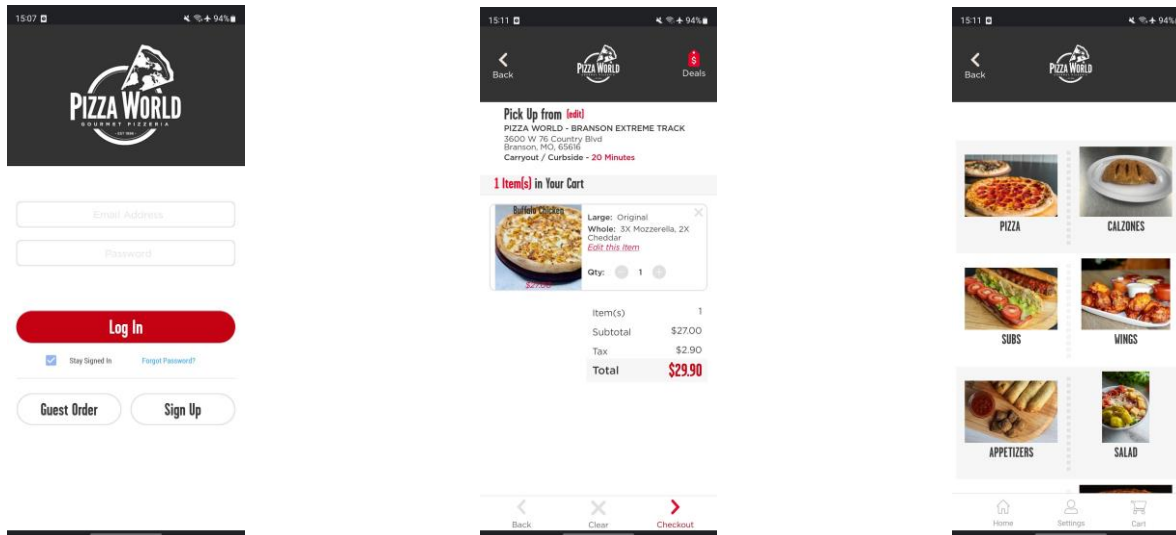
Domino's, jeden z najväčších reťazcov pizze na svete, sprístupňuje väčšinu miest, v ktorých je prítomná služba doručovania. Aplikácia poskytuje ponuku siete pobočiek Domino's a umožňuje priame platby prostredníctvom aplikácie a sledovanie doručovania v reálnom čase.



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

## Pizza World

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>

### Analýza navrhovanej aplikácie

Aplikácia pre objednávanie pizze je praktickým a efektívnym spôsobom, ako si zákazníci môžu objednať svoju obľúbenú pizzu z pohodlia svojho domova.

**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, online bankovníctvom alebo platobnými bránami.

**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.

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 odporúčenie aplikácie.

## Návrh architektúry aplikácie

**Prehľad menu:** Zákazník by mal byť schopný prehliadať kompletne menu reštaurácie, vrátane cien, popisu a obrázkov. Pizza Pro 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 má k dispozícii rôzne možnosti pre vytváranie objednávky, ako napríklad voľba veľkosti a zmena počtu kusov.

**Platba online formou:** Aplikácia by mala umožniť zákazníkovi jednoducho a bezpečne zaplatiť za svoju objednávku. Je tu výber z rôznych možností platby, ako napríklad kreditná karta, PayPal, alebo Apple Pay.

**Nastavenia účtu:** Zákazníci by mali mať možnosť vytvoriť si svoj vlastný účet, kde by mohli spravovať svoje objednávky a získať výhody.

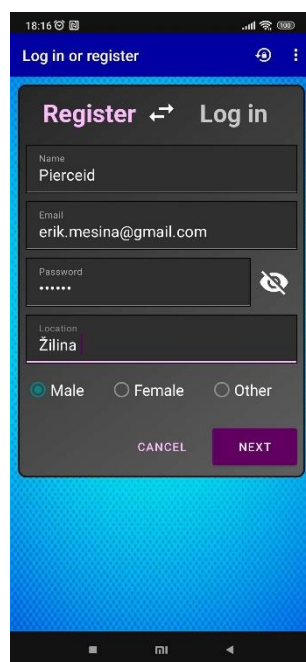
**Recenzie a hodnotenia:** Zákazníci by mali mať možnosť zanechať recenziu a hodnotenia na jednotlivé jedlá, aby mohli pomôcť ostatným zákazníkom pri výbere.

## Ukážka návrhu obrazoviek aplikácie

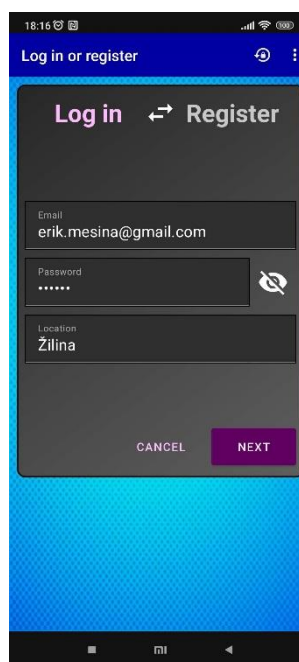
Úvodná obrazovka



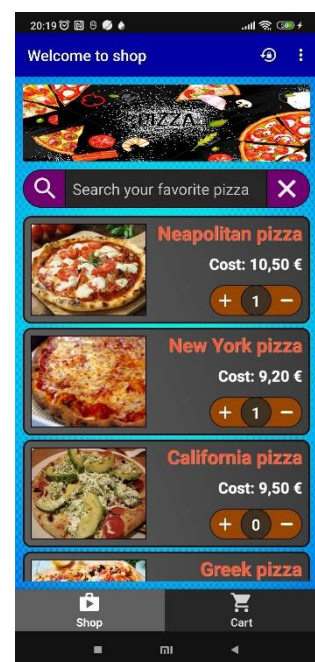
Registrácia



Prihlásenie

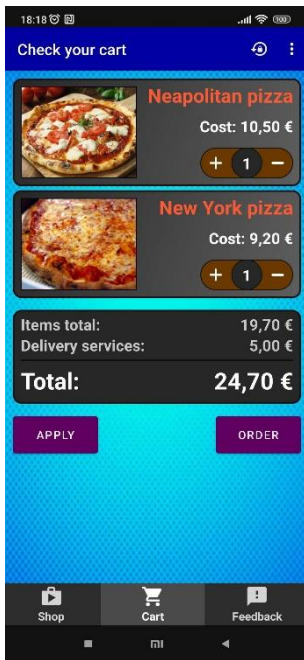


Hlavné menu

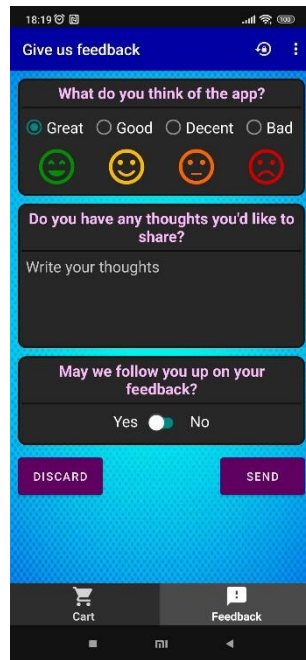




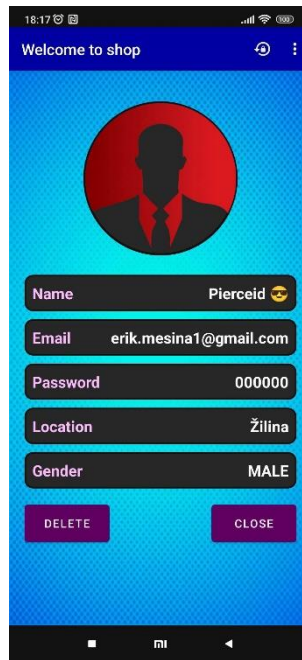
## Košík zákazníka



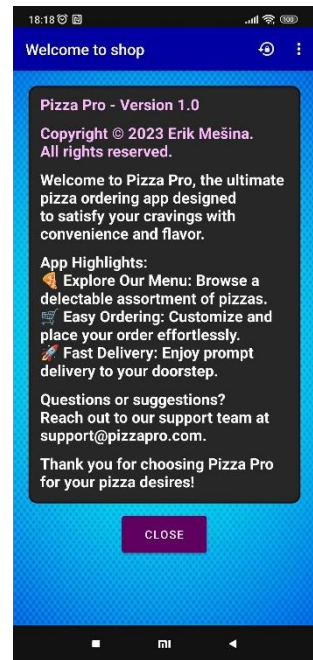
## Spätaná väzba



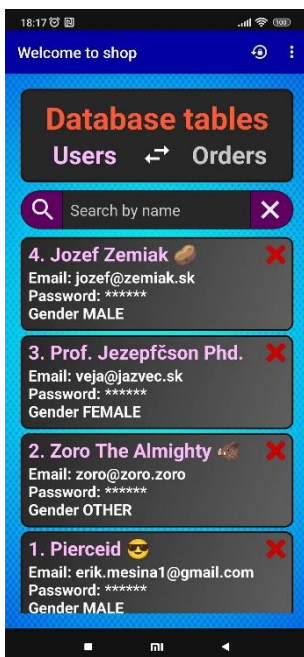
## Profil



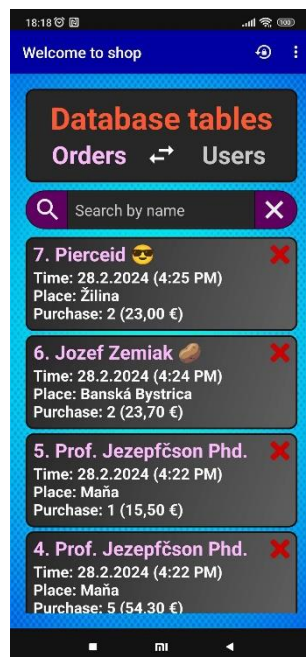
## O aplikácii



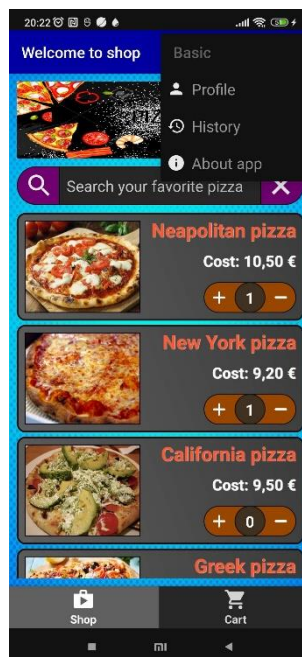
## História používateľov



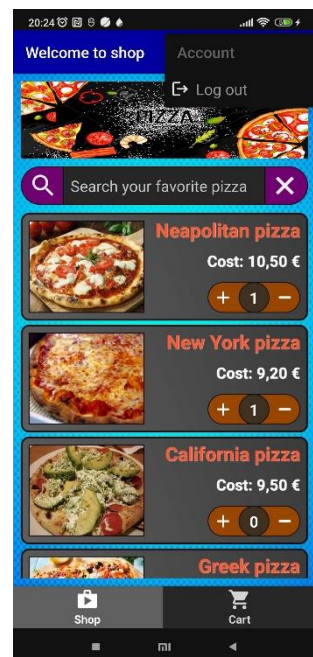
## História objednávok



## Nastavenia základné



## Nastavenia účtu



## Fragmenty

### IntroFragment

Úvodný fragment, obsahuje len tlačidlo, ktoré používateľ presunie na nasledujúci fragment.

### AccountFragment

Fragment, kde si používateľ môže vytvoriť účet. Musí zadať svoje základné informácie, ako sú meno, email, heslo, aktuálnu polohu a pohlavie. Pri nevyplnení niektorej z informácií ho na to systém upozorní. Po úspešnom vyplnení všetkých potrebných informácií môže používateľ pokračovať do hlavného menu, čiže obchodu.

### ShopFragment

V tejto časti si zákazník môže vyhľadať a vybrať konkrétne produkty, ako aj si navoliť ich ľubovoľné množstvá. Ak je zákazník spokojný s výberom, môže pokračovať do košíkovej časti.

### CartFragment

V tejto časti si používateľ môže skontrolovať výber z predošlého fragmentu, poprípade upraviť množstvá. Ak je zákazník spokojný s výberom môže si dané produkty objednať. Objednávka je vizuálne potvrdená vyskakujúcim oknom. Ak by mal zákazník záujem, môže pokračovať do poslednej hlavnej časti, v ktorej môže podať spätnú väzbu.

### FeedbackFragment

Fragment, kde používateľ môže vyjadriť svoju spokojnosť a podať nápady na zlepšenie.

### DetailFragment

Fragment, v ktorom sú bližšie popísané jednotlivé produkty (názov, hodnotenie, čas čakania, množstvo kalórií a stručný popis). Dostupný po kliknutí na názov alebo obrázok produktu.

### ProfileFragment

Fragment, v ktorom sú uvedené informácie o používateľovi zadané pri vytváraní konta. Dostupný po kliknutí na „Profile“ v nastaveniach (vpravo hore).

### HistoryFragment

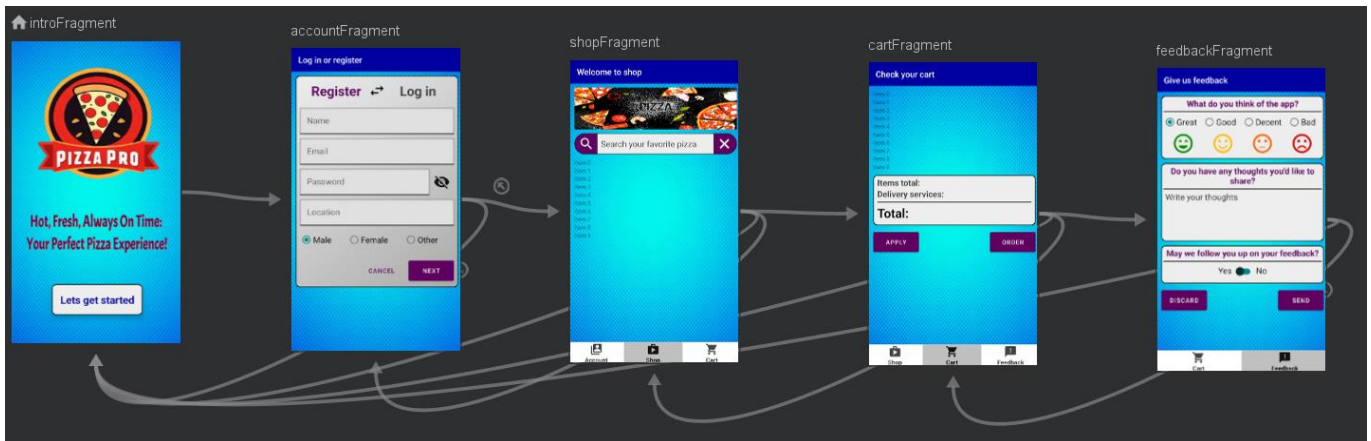
Fragment, v ktorom sú uvedené databázy používateľov a objednávok. Používateľ má možnosť si vyhľadať konkrétnych používateľov a objednávky pomocou zadania mena do vyhľadávacej kolonky. Taktiež má možnosť vymazať účty, stornovať objednávky alebo vymazať celé jednotlivé databázy. Dostupný po kliknutí na „History“ v nastaveniach (vpravo hore).

### AboutAppFragment

Fragment, v ktorom sú uvedené informácie o aplikácii (verzia, autorské práva, popis aplikácie,...). Dostupný po kliknutí na „About app“ v nastaveniach (vpravo hore)

## Navigačný graf

V aplikácii je využitý 1 navigačný graf, ktorý spravuje navigáciu pre všetky hlavné fragmenty (intro, account, shop, cart, feedback).



## Navigácia pre doplňujúce fragmenty

Pre navigáciu k doplňujúcim fragmentom, ako sú detail produktu a profil zákazníka, sa využíva nižšie uvedená statická metóda v triede Util. Taktiež je tu ošetrená duplikácia fragmentov pri opätovnom kliknutí na tú istú možnosť (napr. viac-krát za sebou kliknem na profil).

```
// navigates to an additional fragment
fun navigateToFragment(
    fragmentManager: FragmentManager, fragment: Fragment, bundle: Bundle? = null
) {
    val size = fragmentManager.fragments.size
    if (size > 1) {
        val currentTag = fragmentManager.fragments[size - 1].tag
        val previousTags = fragmentManager.fragments.subList(size - 2, size).map { it.tag }

        if (previousTags.contains(currentTag)) {
            fragmentManager.popBackStack()
        }
    }
    fragment.arguments = bundle
    fragmentManager.beginTransaction().replace(R.id.fragmentContainer, fragment)
        .addToBackStack(null).commit()
}
```

## Odhlásenie

Používateľ má možnosť kedykoľvek sa odhlásiť, čiže presunúť sa na úvodné okno. Dostupné po kliknutí na „Log out“ v nastaveniach (vpravo hore).



## Adaptéry

Adaptér slúži ako most medzi zdrojom údajov a RecyclerView. Je zodpovedný za vytváranie potrebných zobrazení pre každú položku a prepojenie údajov s týmito zobrazeniami.

### MyAdapter

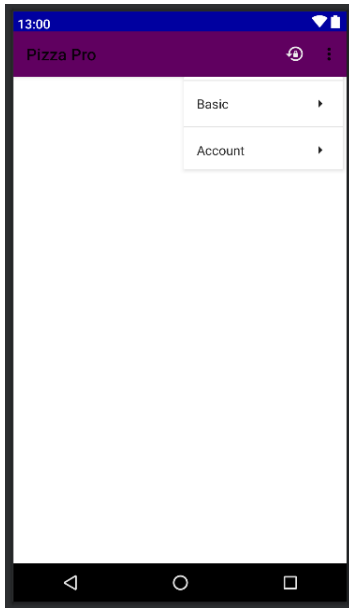
- 2 listenery (pre plusButton a MinusButton) upravujúce množstvo produktu (pizza)
- 2 listenery (pre obrázok a názov pizze) na navigáciu do Detail fragmentu (pizza)
- 1 listener (pre xButton) na odstránenie produktu (user/order)
- funkcie na filtrovanie produktov a inicializáciu zoznamu produktov

```
21 class MyAdapter<T>(private val myContext: MyContext) :  
22     RecyclerView.Adapter<MyAdapter<T>.HistoryViewHolder>() {  
23  
24     private var items: MutableList<T> = mutableListOf()  
25  
26     inner class HistoryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
27         val image: ImageView? = itemView.findViewById(R.id.iv_pizza)  
28         val name: TextView? = itemView.findViewById(R.id.tv_name)  
29         val count: TextView? = itemView.findViewById(R.id.tv_itemCount)  
30         val cost: TextView? = itemView.findViewById(R.id.tv_cost)  
31         val header: TextView? = itemView.findViewById(R.id.tv_header)  
32         val body: TextView? = itemView.findViewById(R.id.tv_body)  
33  
34         private val xButton: ImageView? = itemView.findViewById(R.id.btn_x)  
35         private val plusButton: ImageView? = itemView.findViewById(R.id.iv_plus)  
36         private val minusButton: ImageView? = itemView.findViewById(R.id.iv_minus)  
37  
38         init {  
39             when (myContext.type) {  
40                 "pizzas" -> {  
41                     plusButton?.setOnClickListener {  
42                         val pizza = items[adapterPosition] as Pizza  
43                         if (adapterPosition != RecyclerView.NO_POSITION && pizza.count < 10) {  
44                             pizza.count++  
45                             notifyItemChanged(adapterPosition)  
46                         }  
47                     }  
48                 }  
49             }  
50         }  
51     }  
52  
53     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HistoryViewHolder {  
54         // creates the view holder  
55         override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): HistoryViewHolder {  
56             val layout = if (myContext.type == "pizzas") R.layout.item_pizza else R.layout.item_history  
57             return HistoryViewHolder(  
58                 LayoutInflater.from(parent.context).inflate(layout, parent, false)  
59             )  
60         }  
61     }  
62  
63     override fun onBindViewHolder(holder: HistoryViewHolder, position: Int) {  
64         when (myContext.type) {  
65             "pizzas" -> {  
66                 val pizza = items[position] as Pizza  
67                 holder.image?.setImageResource(pizza.imageSource)  
68                 holder.name?.text = pizza.name  
69                 holder.count?.text = pizza.count.toString()  
70                 holder.cost?.text =  
71                     String.format("Cost: $s", NumberFormat.getCurrencyInstance().format(pizza.cost))  
72             }  
73             "users" -> {  
74                 val user = items[position] as User  
75                 holder.header?.text = String.format("Id: $s", user.id, user.name)  
76                 holder.body?.text = String.format(  
77                     "Email: $s\nPassword: $s\nGender: $s",  
78                     user.email, user.password.map { '*' }.joinToString(""), user.gender  
79                 )  
80             }  
81             "orders" -> {  
82                 val order = items[position] as Order  
83                 holder.header?.text = String.format("Id: $s", order.id, order.name)  
84                 holder.body?.text = String.format(  
85                     "Time: $s\nPlace: $s\nPurchase: $d ($s)",  
86                     order.time, order.place, order.items, order.cost  
87                 )  
88             }  
89         }  
90     }  
91  
92     override fun getItemCount(): Int {  
93         return items.size  
94     }  
95  
96     fun updateItems(items: List<T>) {  
97         this.items = items.toMutableList()  
98     }  
99  
100     fun notifyItemChanged(position: Int) {  
101         notifyItemChanged(position)
```

```
102     }  
103  
104     override fun onBindViewHolder(holder: HistoryViewHolder, position: Int) {  
105         when (myContext.type) {  
106             "pizzas" -> {  
107                 val pizza = items[position] as Pizza  
108                 holder.image?.setImageResource(pizza.imageSource)  
109                 holder.name?.text = pizza.name  
110                 holder.count?.text = pizza.count.toString()  
111                 holder.cost?.text =  
112                     String.format("Cost: $s", NumberFormat.getCurrencyInstance().format(pizza.cost))  
113             }  
114             "users" -> {  
115                 val user = items[position] as User  
116                 holder.header?.text = String.format("Id: $s", user.id, user.name)  
117                 holder.body?.text = String.format(  
118                     "Email: $s\nPassword: $s\nGender: $s",  
119                     user.email, user.password.map { '*' }.joinToString(""), user.gender  
120                 )  
121             }  
122             "orders" -> {  
123                 val order = items[position] as Order  
124                 holder.header?.text = String.format("Id: $s", order.id, order.name)  
125                 holder.body?.text = String.format(  
126                     "Time: $s\nPlace: $s\nPurchase: $d ($s)",  
127                     order.time, order.place, order.items, order.cost  
128                 )  
129             }  
130         }  
131     }  
132  
133     override fun getItemCount(): Int {  
134         return items.size  
135     }  
136  
137     fun updateItems(items: List<T>) {  
138         this.items = items.toMutableList()  
139     }  
140  
141     fun notifyItemChanged(position: Int) {  
142         notifyItemChanged(position)
```

## Nastavenia – menu

Slúži na poskytnutie používateľom spôsobu, ako prechádzať medzi rôznymi časťami aplikácie. Menu tejto aplikácie obsahuje 2 časti: základnú a účtovú. Základná časť obsahuje prepojenie na profil používateľa, históriu objednávok a informácie o aplikácii, zatiaľ čo účtová ponúka možnosť odhlásenia. Súčasťou menu je aj priamo viditeľná možnosť zamknúť/odmknúť rotáciu obrazovky.



## 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.

```
1 package com.example.pizza_pro.database
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.PrimaryKey
6 import com.example.pizza_pro.options.Gender
7
8 @Entity(tableName = "user_table")
9 data class User(
10     @PrimaryKey(autoGenerate = true) var id: Long = 0L,
11     @ColumnInfo(name = "name") var name: String = "",
12     @ColumnInfo(name = "email") var email: String = "",
13     @ColumnInfo(name = "password") var password: String = "",
14     @ColumnInfo(name = "location") var location: String = "",
15     @ColumnInfo(name = "gender") var gender: Gender = Gender.OTHER
16 )
17
18 @Entity(tableName = "order_table")
19 data class Order(
20     @PrimaryKey(autoGenerate = true) var id: Long = 0L,
21     @ColumnInfo(name = "name") var name: String = "",
22     @ColumnInfo(name = "time") var time: String = "",
23     @ColumnInfo(name = "place") var place: String = "",
24     @ColumnInfo(name = "items") var items: Int = 0,
25     @ColumnInfo(name = "cost") var cost: String = ""
26 )
```

## MyDatabase

Abstraktná trieda, ktorá deklaruje tabuľky v databáze. Tabuľky obsahujú stĺpce, ktoré reprezentujú jednotlivé údaje z entít User a Order.

```
1 package com.example.pizza_pro.database
2
3 import android.content.Context
4 import androidx.room.Database
5 import androidx.room.Room
6 import androidx.room.RoomDatabase
7
8 @Database(entities = [User::class, Order::class], version = 3, exportSchema = false)
9 abstract class MyDatabase : RoomDatabase() {
10
11     abstract val dao: MyDao
12
13     companion object {
14
15         @Volatile
16         private var INSTANCE: MyDatabase? = null
17
18         fun getDatabase(context: Context): MyDatabase {
19             synchronized(this) {
20                 var instance = INSTANCE
21                 if (instance == null) {
22                     instance = Room.databaseBuilder(
23                         context.applicationContext,
24                         MyDatabase::class.java,
25                         "my_database"
26                     ).fallbackToDestructiveMigration().build()
27                     INSTANCE = instance
28                 }
29                 return instance
30             }
31         }
32     }
33 }
```

## MyDao

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

```
6 @Dao
7 interface MyDao {
8     @Upsert
9     suspend fun upsertUser(user: User)
10
11     @Delete
12     suspend fun deleteUser(user: User)
13
14     @Upsert
15     suspend fun upsertOrder(order: Order)
16
17     @Delete
18     suspend fun deleteOrder(order: Order)
19
20     @Query("DELETE FROM user_table")
21     suspend fun cleanAllUsers()
22
23     @Query("DELETE FROM order_table")
24     suspend fun cleanAllOrders()
25
26     @Query("SELECT * FROM user_table WHERE name = :name OR email = :email LIMIT 1")
27     fun getUser(name: String, email: String): User?
28
29     @Query("SELECT * FROM user_table WHERE name LIKE '%' || :regex || '%' ORDER BY id DESC")
30     fun getFilteredUsers(regex: String): LiveData<MutableList<User>>
31
32     @Query("SELECT * FROM order_table WHERE name LIKE '%' || :regex || '%' ORDER BY id DESC")
33     fun getFilteredOrders(regex: String): LiveData<MutableList<Order>>
34
35     @Query("SELECT * FROM user_table ORDER BY id DESC")
36     fun getAllUsers(): LiveData<MutableList<User>>
37
38     @Query("SELECT * FROM order_table ORDER BY id DESC")
39     fun getAllOrders(): LiveData<MutableList<Order>>
40 }
```

## MyRepository

Účelom tejto triedy je abstrahovať zdroj údajov, ako je databáza alebo lokálne úložisko, od zvyšku aplikácie. Poskytuje čisté a konzistentné API na prístup a správu údajov, čo uľahčuje údržbu a testovanie aplikácie.

```
1 package com.example.pizza_pro.database
2
3 import androidx.lifecycle.LiveData
4
5 class MyRepository(private val dao: MyDao) {
6
7     var allUsers: LiveData<MutableList<User>> = dao.getAllUsers()
8     var allOrders: LiveData<MutableList<Order>> = dao.getAllOrders()
9
10    suspend fun addUser(user: User) = dao.upsertUser(user)
11
12    suspend fun addOrder(order: Order) = dao.upsertOrder(order)
13
14    suspend fun removeUser(user: User) = dao.deleteUser(user)
15
16    suspend fun removeOrder(order: Order) = dao.deleteOrder(order)
17
18    suspend fun clearAllUsers() = dao.clearAllUsers()
19
20    suspend fun clearAllOrders() = dao.clearAllOrders()
21
22    fun getUser(name: String, email: String) = dao.getUser(name, email)
23
24    fun getFilteredOrders(regex: String) = dao.getFilteredOrders(regex)
25
26    fun getFilteredUsers(regex: String) = dao.getFilteredUsers(regex)
27 }
```

## MyViewModel

Trieda, ktorá slúži na komunikáciu medzi repozitárom a používateľským rozhraním.

```
10 class MyViewModel(application: Application) : AndroidViewModel(application) {
11
12     private val repository: MyRepository
13     var users: LiveData<MutableList<User>>
14     var orders: LiveData<MutableList<Order>>
15     var user: User?
16
17     init {
18         val dao = MyDatabase.getDatabase(application).dao
19         repository = MyRepository(dao)
20         orders = repository.allOrders
21         users = repository.allUsers
22         user = null
23     }
24
25     fun addUser(user: User) {
26         viewModelScope.launch(Dispatchers.IO) {
27             repository.addUser(user)
28         }
29     }
30
31     fun addOrder(order: Order) {
32         viewModelScope.launch(Dispatchers.IO) {
33             repository.addOrder(order)
34         }
35     }
36
37     fun removeUser(user: User) {
38         viewModelScope.launch(Dispatchers.IO) {
39             repository.removeUser(user)
40         }
41     }
42
43     fun removeOrder(order: Order) {
44         viewModelScope.launch(Dispatchers.IO) {
45             repository.removeOrder(order)
46         }
47     }
48
49     fun clearAllUsers() {
50         viewModelScope.launch(Dispatchers.IO) {
51             repository.clearAllUsers()
52         }
53     }
54
55     fun clearAllOrders() {
56         viewModelScope.launch(Dispatchers.IO) {
57             repository.clearAllOrders()
58         }
59     }
60
61     fun getUser(name: String, email: String) {
62         viewModelScope.launch(Dispatchers.IO) {
63             user = repository.getUser(name, email)
64         }
65     }
66
67     fun getFilteredUsers(regex: String) {
68         users = repository.getFilteredUsers(regex)
69     }
70
71     fun getFilteredOrders(regex: String) {
72         orders = repository.getFilteredOrders(regex)
73     }
74 }
```

## Použité zdroje

- <https://developer.android.com/teach#for-instructors-teaching-a-course>
- <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.fileformat.info/info/unicode/char/search.htm>
- <https://www.javatpoint.com/kotlin-android-alertdialog>
- <https://github.com/DanielMartinus/Konfetti>