ŽILINSKÁ UNIVERZITA
V ŽILINE

Pizza Pro

semestrálna práca

Fakulta: Fakulta riadenia a informatiky

Program: Informatika a riadenie

Predmet: VAMZ

Zadanie a špecifikácia: Vytvorenie mobilnej aplikácie

Vypracoval: Erik Mešina

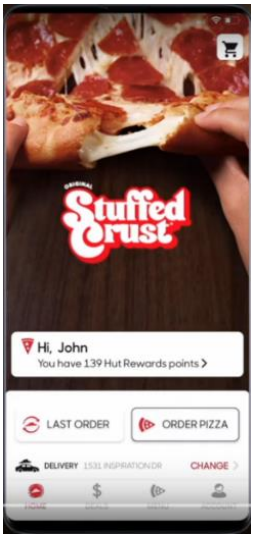
Skupina: 5ZYR34

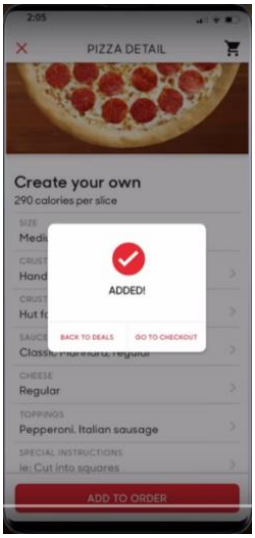
Obsah

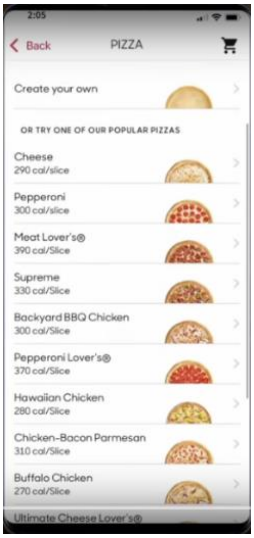
Prehľad aplikácií s podobným zameraním

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

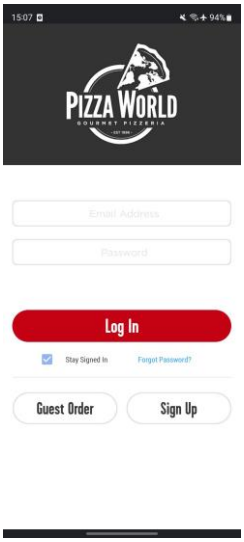
Domino's Pizza Brasil

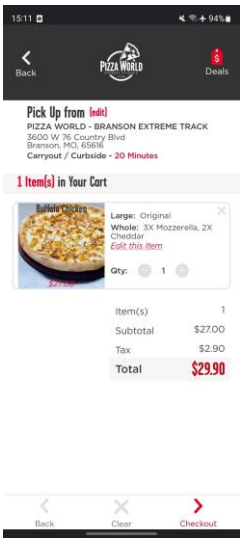
Domino's, jeden z najväčších reťazcov pizze na svete, sprístupňuje väčšinu miest, v ktorých je prítomná služba doručovania. Aplikácia poskytuje ponuku siete pobočiek Domino’s a umožňuje priame platby prostredníctvom aplikácie a sledovanie doručovania v reálnom čase.

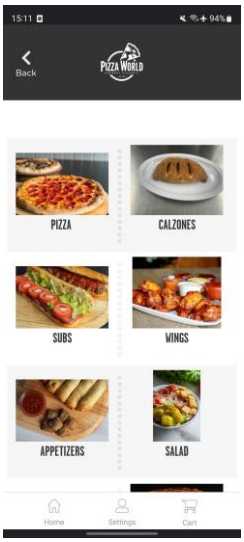
Zdroj: https://apkpure.com/domino-s-pizza-brasil/br.com.dominos.mobile

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 odporučenie aplikácie.

Návrh architektúry aplikácie

Prehľad menu: Zákazník by mal byť schopný prehliadať kompletné 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 zákazníkom 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 získavať výhody.

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

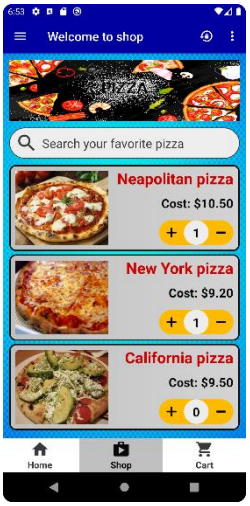
Ukážka návrhu obrazoviek aplikácie

Úvod

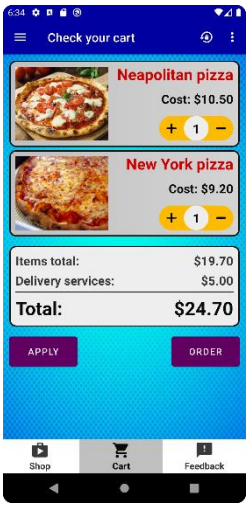
Registrácia

Prihlásenie

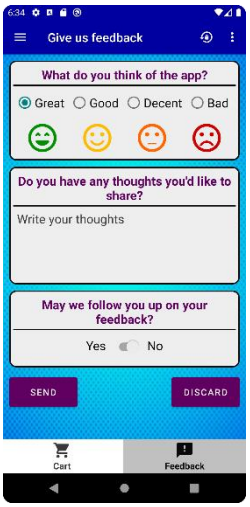
Hlavné menu



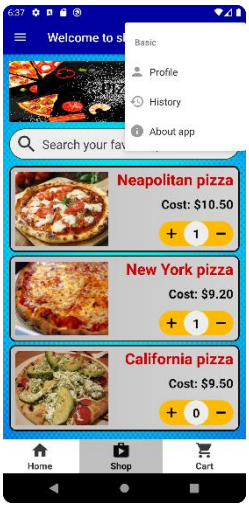
Košík zákazníka



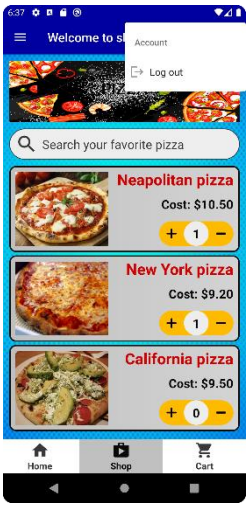
Spätná väzba



Nastavenia 1



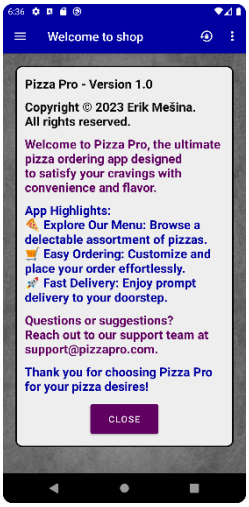
Nastavenia 2



Profil

História

O aplikácií



Fragmenty

IntroFragment

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

AccountFragment

Fragment, kde si používateľ môže vytvoriť účet. Musí zadať svoje základné informácie, ako sú meno, email, heslo, pohlavie a lokalizáciu. 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 produktu.

ProfileFragment

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

HistoryFragment

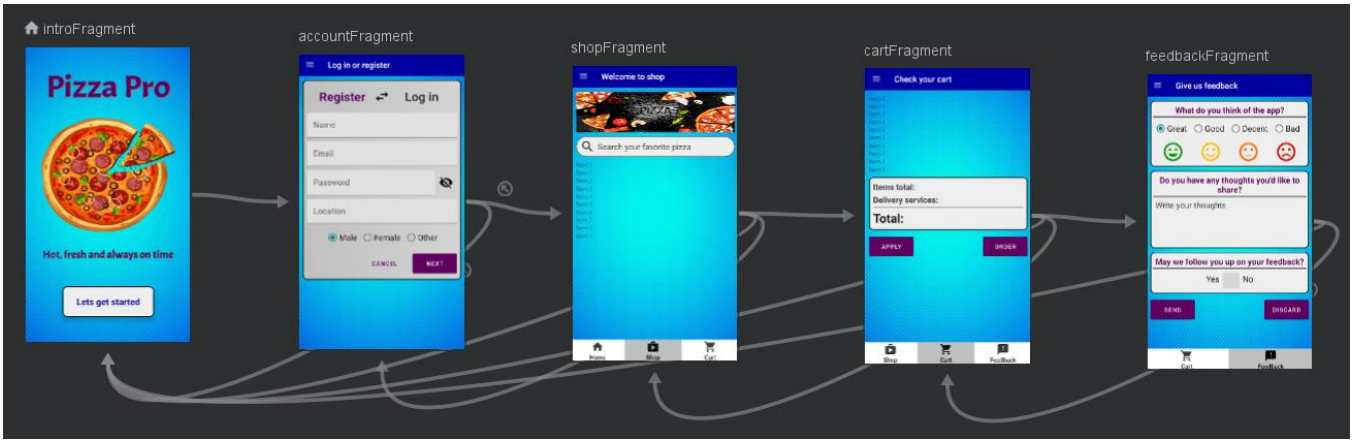
Fragment, v ktorom sú uvedené úspešné objednávky. Používateľ má možnosť si vyhľadať konkrétne objednávky pomocou zadania mena. Taktiež má možnosť vymazať databázu. Dostupný po kliknutí na „History“ v nastaveniach (vpravo hore).

AboutAppFragment

Fragment, v ktorom sú uvedené informácie o aplikácií (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, profil zákazníka, história objednávok a informácie o aplikácii, 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 a 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( name: 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).

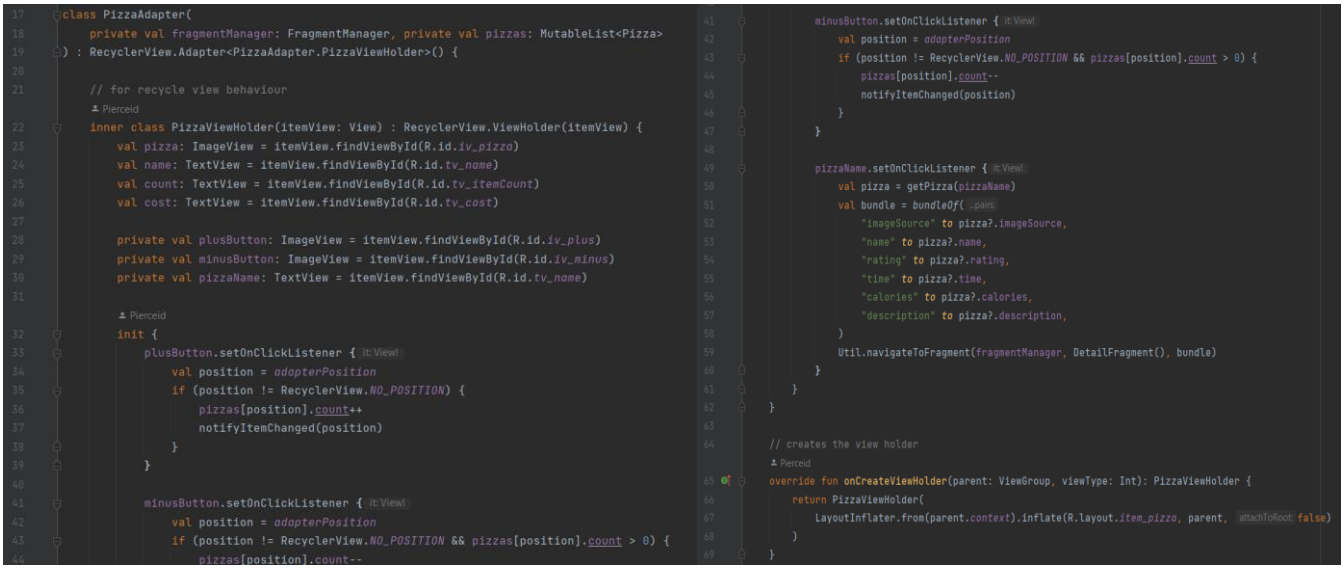
Adaptéry

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

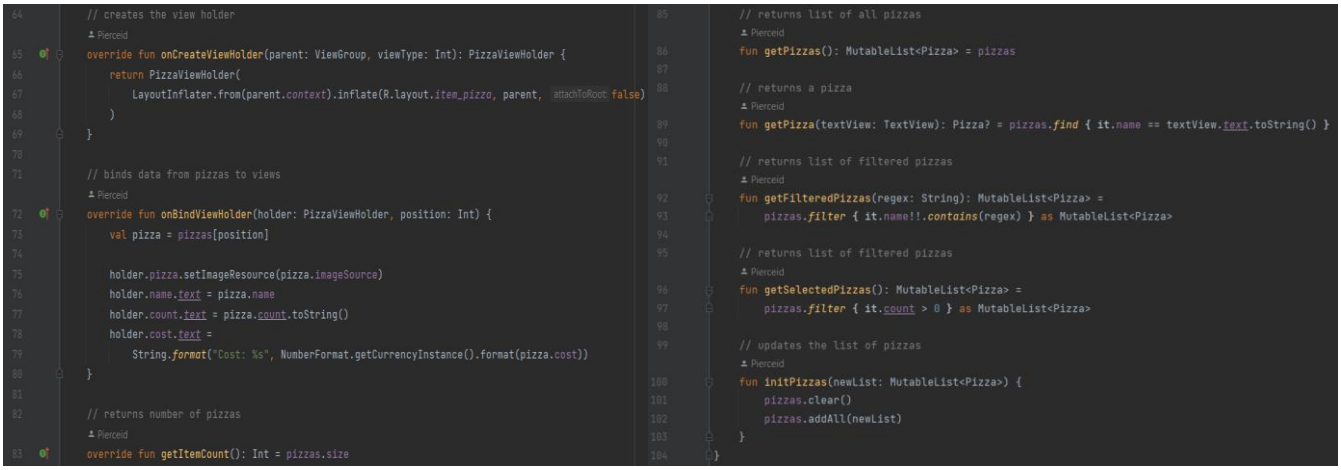
PizzaAdapter

Pizza adaptér je rozšírený o niekoľko funkcií:

- 2 listenery (pre plusButton a MinusButton) upravujúce množstvo produktu
- 1 listener pre navigáciu do Detail fragmentu
- funkcie na filtrovanie produktov a inicializáciu zoznamu produktov



```
class PizzaAdapter(
    private val fragmentManager: FragmentManager, private val pizzas: MutableList<Pizza>
) : RecyclerView.Adapter<PizzaAdapter.PizzaViewHolder>() {

    // for recycle view behaviour
    Pierceid
    inner class PizzaViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val pizza: ImageView = itemView.findViewById(R.id.iv_pizza)
        val name: TextView = itemView.findViewById(R.id.tv_name)
        val count: TextView = itemView.findViewById(R.id.tv_itemCount)
        val cost: TextView = itemView.findViewById(R.id.tv_cost)

        private val plusButton: ImageView = itemView.findViewById(R.id.iv_plus)
        private val minusButton: ImageView = itemView.findViewById(R.id.iv_minus)
        private val pizzaName: TextView = itemView.findViewById(R.id.tv_name)

        Pierceid
        init {
            plusButton.setOnClickListener { it:View!
                val position = adapterPosition
                if (position != RecyclerView.NO_POSITION) {
                    pizzas[position].count++
                    notifyItemChanged(position)
                }
            }

            minusButton.setOnClickListener { it:View!
                val position = adapterPosition
                if (position != RecyclerView.NO_POSITION && pizzas[position].count > 0) {
                    pizzas[position].count--
                    notifyItemChanged(position)
                }
            }

            pizzaName.setOnClickListener { it:View!
                val pizza = getPizza(pizzaName)
                val bundle = bundleOf( ...pairs:
                    "imageSource" to pizza?.imageSource,
                    "name" to pizza?.name,
                    "rating" to pizza?.rating,
                    "time" to pizza?.time,
                    "calories" to pizza?.calories,
                    "description" to pizza?.description,
                )
                Util.navigateToFragment(fragmentManager, DetailFragment(), bundle)
            }
        }
    }

    // creates the view holder
    Pierceid
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PizzaViewHolder {
        return PizzaViewHolder(
            LayoutInflater.from(parent.context).inflate(R.layout.item_pizza, parent, attachToRoot: false)
        )
    }
```



```
    // creates the view holder
    Pierceid
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PizzaViewHolder {
        return PizzaViewHolder(
            LayoutInflater.from(parent.context).inflate(R.layout.item_pizza, parent, attachToRoot: false)
        )
    }

    // binds data from pizzas to views
    Pierceid
    override fun onBindViewHolder(holder: PizzaViewHolder, position: Int) {
        val pizza = pizzas[position]

        holder.pizza.setImageResource(pizza.imageSource)
        holder.name.text = pizza.name
        holder.count.text = pizza.count.toString()
        holder.cost.text =
            String.format("Cost: %s", NumberFormat.getCurrencyInstance().format(pizza.cost))
    }

    // returns number of pizzas
    Pierceid
    override fun getItemCount(): Int = pizzas.size

    // returns list of all pizzas
    Pierceid
    fun getPizzas(): MutableList<Pizza> = pizzas

    // returns a pizza
    Pierceid
    fun getPizza(textView: TextView): Pizza? = pizzas.find { it.name == textView.text.toString() }

    // returns list of filtered pizzas
    Pierceid
    fun getFilteredPizzas(regex: String): MutableList<Pizza> =
        pizzas.filter { it.name!!.contains(regex) } as MutableList<Pizza>

    // returns list of filtered pizzas
    Pierceid
    fun getSelectedPizzas(): MutableList<Pizza> =
        pizzas.filter { it.count > 0 } as MutableList<Pizza>

    // updates the list of pizzas
    Pierceid
    fun initPizzas(newList: MutableList<Pizza>) {
        pizzas.clear()
        pizzas.addAll(newList)
    }
}
```

OrderAdapter

Objednávkový adaptér je taktiež rozšírený o:

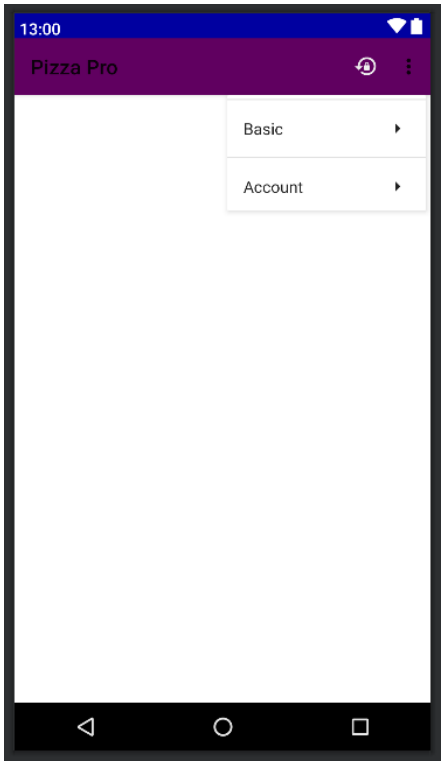
- funkciu na inicializáciu zoznamu produktov

```
class OrderAdapter : RecyclerView.Adapter<OrderAdapter.OrderViewHolder>(){

    private val orders: MutableList<Order> = mutableListOf()

    Pierceid
    inner class OrderViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val header: TextView = itemView.findViewById(R.id.tv_header)
        val body: TextView = itemView.findViewById(R.id.tv_body)
    }

    // creates the view holder
    Pierceid
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): OrderViewHolder {
        return OrderViewHolder(
            LayoutInflater.from(parent.context).inflate(R.layout.item_order, parent, attachToRoot: false)
        )
    }
```

```
    // binds data from orders to views
    Pierceid *
    override fun onBindViewHolder(holder: OrderViewHolder, position: Int) {
        val order = orders[position]

        holder.header.text = String.format("%d. %s", order.id, order.userInfo.name)
        holder.body.text = String.format(
            "Email: %s\nTime: %s\nPlace: %s\nItems: %d\nCost: %s",
            order.userInfo.email,
            order.time,
            order.place,
            order.items,
            order.cost
        )
    }

    // returns number of orders
    Pierceid
    override fun getItemCount(): Int = orders.size

    // updates the list of orders
    Pierceid
    fun initOrders(newList: MutableList<Order>) {
        val removedItems = orders.size
        val insertedItems = newList.size
        orders.clear()
        orders.addAll(newList)

        if (removedItems > 0) notifyItemRangeRemoved( positionStart: 0, removedItems)
        if (insertedItems > 0) notifyItemRangeInserted( positionStart: 0, insertedItems)
    }
```

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 aplikácií, zatiaľ čo účtová ponúka možnosť odhlásenia. Súčasťou menu je aj priamo viditeľná možnosť zamknúť/odomknúť rotáciu obrazovky.



Room Databáza

Room databáza pre objednávky je komponent v Android aplikácii, ktorý umožňuje efektívne uchovávanie a spravovanie údajov o objednávkach. Táto databáza je navrhnutá tak, aby ukladala nasledujúce informácie:

- **ID Objednávky**: Unikátne číslo, ktoré jednoznačne identifikuje každú objednávku.
- **Informácie O Zákazníkovi**: Ide o súhrn základných informácií zákazníka, ako sú jeho meno, emailová adresa pre komunikáciu a notifikácie, heslo na zabezpečenie účtu a pohlavie.
- **Čas Objednávky**: Čas uskutočnenia objednávky zákazníkom.
- **Miesto Donášky**: Miesto donášky objednávky vybrané zákazníkom.
- **Počet Objednaných Kusov**: Množstvo tovaru alebo produktov, ktoré boli objednané v rámci tejto objednávky.
- **Celková Cena za Nákup**: Suma peňazí, ktoré zákazník zaplatí za objednaný tovar.

Tieto údaje sú definované v entite a Room Databáza poskytuje prostredie na efektívne ukladanie, získavanie a aktualizovanie týchto dát v rámci Android aplikácie. Room Databáza taktiež zabezpečuje, že dáta sú konzistentné a spoľahlivé a že s nimi je možné pracovať na hlavnom vlákne alebo asynchrónne podľa potreby.

Order

Dátová trieda, ktorá uchováva informácie o objednávke (id, informácie o zákazníkovi, čas objednania a miesto doručenia, počet produktov a cenu nákupu). Id objednávky je špeciálny identifikátor, ktorý sa automaticky generuje a podľa ktorého vieme presne určiť zvyšné informácie o objednávke.

```kotlin
package com.example.pizza_pro.database

import ...

Pierceid *
@Entity(tableName = "pizza_order_table")
data class Order(
    @PrimaryKey(autoGenerate = true) var id: Long = 0L,

    @Embedded var userInfo: UserInfo = UserInfo(),

    @ColumnInfo(name = "time") var time: String = "",

    @ColumnInfo(name = "place") var place: String = "",

    @ColumnInfo(name = "items") var items: Int = 0,

    @ColumnInfo(name = "cost") var cost: String = ""
)

new *
data class UserInfo(
    var name: String = "",
    var email: String = "",
    var password: String = "",
    var gender: Gender = Gender.MALE
)
```

OrderDatabase

Abstraktná trieda, ktorá deklaruje tabuľku v databáze. Tabuľka obsahuje stĺpce, ktoré reprezentujú jednotlivé údaje z dátovej triedy Order.

```kotlin
package com.example.pizza_pro.database

import ...

Pierceid *
@Database(entities = [Order::class], version = 1, exportSchema = false)
abstract class OrderDatabase : RoomDatabase() {

    abstract val dao: OrderDao

    Pierceid
    companion object {

        @Volatile
        private var INSTANCE: OrderDatabase? = null

        Pierceid
        fun getDatabase(context: Context): OrderDatabase {
            synchronized( lock: this) {
                var instance = INSTANCE
                if (instance == null) {
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        OrderDatabase::class.java,
                        name: "pizza_order_history_database"
                    ).fallbackToDestructiveMigration().build()
                    INSTANCE = instance
                }
                return instance
            }
        }
    }
```

OrderDao

Rozhranie, ktorý obsahuje SQL príkazy pre prístup a manipuláciu s údajmi v databáze. Obsahuje metódy na vkladanie, aktualizovanie a odstránenie objednávky, odstránenie všetkých objednávok v databáze, filtrovanie objednávok podľa mena alebo emailu zákazníka a metódu na prístup k všetkým objednávkam v databáze.

```kotlin
package com.example.pizza_pro.database

import ...

Pierceid *
@Dao
interface OrderDao {
    Pierceid
    @Upsert
    suspend fun upsertOrder(order: Order)

    Pierceid
    @Delete
    suspend fun deleteOrder(order: Order)

    Pierceid *
    @Query("DELETE FROM pizza_order_table")
    suspend fun clearAllOrders()

    new *
    @Query("SELECT * FROM pizza_order_table WHERE email = :email LIMIT 1")
    suspend fun getOrder(email: String): Order?

    Pierceid
    @Query("SELECT * FROM pizza_order_table WHERE name LIKE '%' || :regex || '%' ORDER BY id DESC")
    fun getFilteredOrders(regex: String): LiveData<MutableList<Order>>

    Pierceid
    @Query("SELECT * FROM pizza_order_table ORDER BY id DESC")
    fun getAllOrders(): LiveData<MutableList<Order>>
}
```

Repository

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.

```kotlin
package com.example.pizza_pro.database

import androidx.lifecycle.LiveData

Pierceid *
class OrderRepository(private val dao: OrderDao) {

    var allOrders: LiveData<MutableList<Order>> = dao.getAllOrders()

    Pierceid
    suspend fun addOrder(order: Order) = dao.upsertOrder(order)

    Pierceid
    suspend fun clearAllOrders() = dao.clearAllOrders()

    new *
    suspend fun getOrder(email: String) = dao.getOrder(email)

    Pierceid
    fun getFilteredOrders(regex: String) = dao.getFilteredOrders(regex)
}
```

OrderViewModel

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

```kotlin
package com.example.pizza_pro.database

import ...

Pierceid
class OrderViewModel(application: Application) : AndroidViewModel(application) {

    private val repository: OrderRepository
    var orders: LiveData<MutableList<Order>>
    var userOrder: Order?

    Pierceid
    init {
        val dao = OrderDatabase.getDatabase(application).dao
        repository = OrderRepository(dao)
        orders = repository.allOrders
        userOrder = null
    }

    Pierceid
    fun addOrder(order: Order) {
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
            repository.addOrder(order)
        }
    }

    Pierceid
    fun clearAllOrders() {
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
            repository.clearAllOrders()
        }
    }

    new *
    fun getUserOrder(email: String) {
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
            userOrder = if (email.isNotEmpty()) repository.getOrder(email) else null
        }
    }

    Pierceid
    fun filterOrders(regex: String) {
        orders = if (regex.isNotEmpty()) repository.getFilteredOrders(regex)
        else repository.allOrders
    }
}
```

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