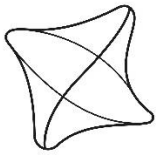


ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Fakulta riadenia a informatiky

Dokumentácia
mobilnej aplikácie WareFlow

Semestrálna práca

Mário Stachera

Študijný program: Informatika a riadenie
Cvičiaci: doc. Ing. Patrik Hrkút, PhD.
Akademický rok: 2023/2024

Obsah

ZOZNAM OBRÁZKOV

1. POPIS A ANALÝZA PROBLÉMU

Cieľom tejto semestrálnej práce je vytvorenie mobilnej aplikácie na predmet Vývoj aplikácií pre mobilné zariadenia.

1.1 ZADANIE

Témou aplikácie je spravovanie skladových zásob, tvorba objednávok a užívateľský príjemný spôsob primania objednávok. Bude sa teda jednať o jednoduchý informačný systém pre riadenie skladových zásob a objednávok.

Aplikácia bude vytvorená v jazyku **Kotlin** a bude využité vývojové prostredie **Android studio**. Na tvorbu dizajnu bol využívaný online nástroj **Figma**, ktorý podporuje rôzne pluginy, ktoré môžu pomôcť aj s implementáciou dizajnu priamo do aplikácie pomocou vygenerovania kódu.

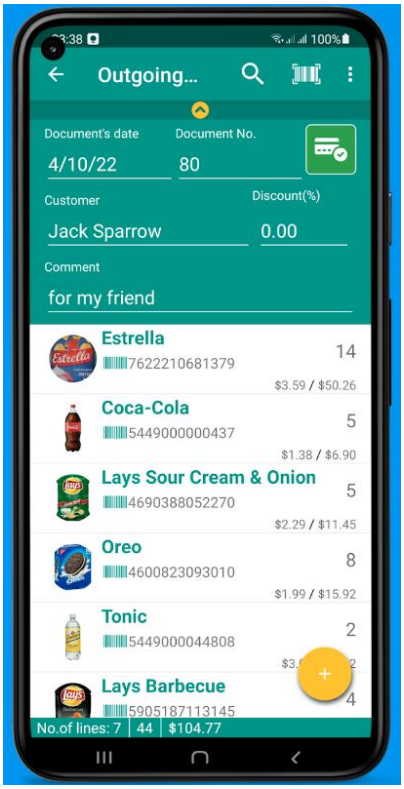
1.2 PODOBNÉ APLIKÁCIE

V tejto kapitole je zhrnutý prehľad podobných aplikácií, ktoré sa už nachádzajú na trhu a jednoduchý popis ich funkcionalít spolu s ich náhľadom.

1.2.1 STOCK AND INVENTORY SIMPLE

Táto aplikácie podporuje evidenciu skladových zásob. Je cielená na menšie obchody a umožňuje importovanie alebo exportovanie súborov vo formáte programu Excel. Aplikáciu je možné využívať aj na správu predaja a nákupu a taktiež podporuje skenovanie čiarových kódov. [1]

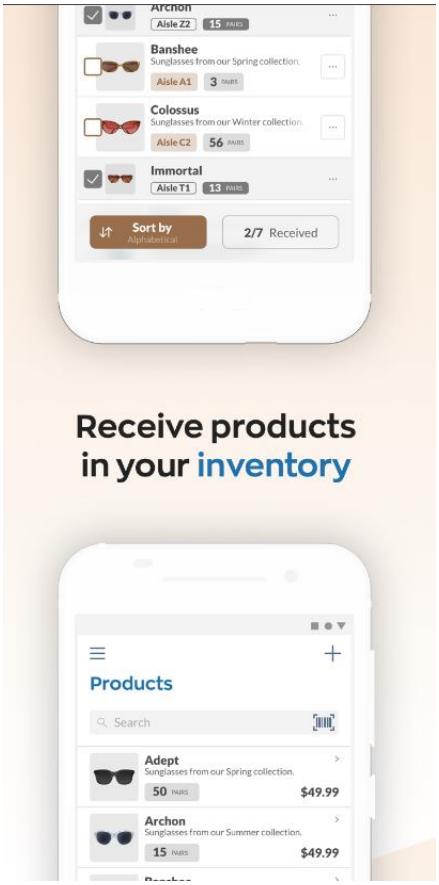
Rozdielom oproti vyvíjanej aplikácií je zobrazenie náhľadových obrázkov produktov a skenovanie čiarových kódov.



Obrázok 1 - zobrazenie aplikácie Stock and Inventory Simple

1.2.2 INFLOW CLOUD

Aplikácia je využívaná na správu objednávok a zásob. Podporuje funkcionality generovania nových objednávok, monitorovanie skladových zásob a taktiež skenovanie čiarových kód produktov. [2]

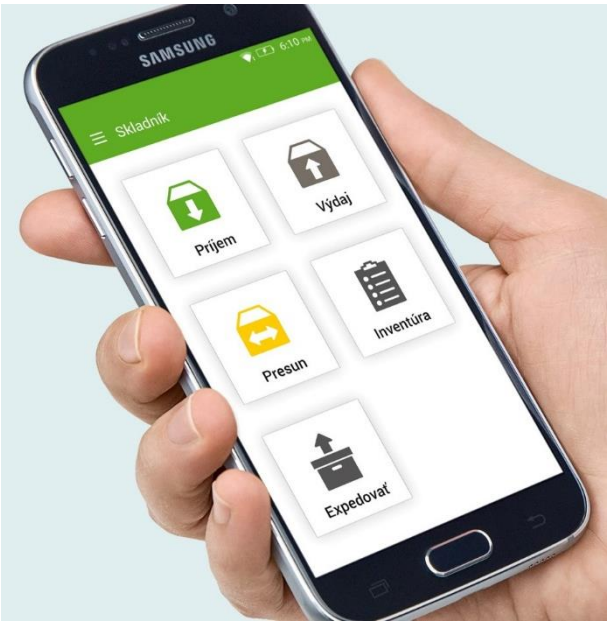


Obrázok 2 - zobrazenie aplikácie inFlow Cloud

1.2.3 SOFTIP MOBILNÝ SKLADNÍK

Táto mobilná aplikácie umožňuje príjem tovaru na sklad a následný výdaj zo skladu, vykonávanie inventúry a úprav. Cieľom aplikácie je vytvoriť prehľad o zásobách, minimalizáciu chýb, zrýchlenie inventúry a prehľad o produktivite. [3]

Aplikácia SOFTIP Mobilný skladník sa odlišuje možnosťou vykonávania inventúry a úprav skladových zásob.



Obrázok 3 - zobrazenie aplikácie SOFTIP Mobilný skladník

2. NÁVRH RIEŠENIA

V tejto kapitole je znázornený UML diagram prípadov použitia, ktorý opisuje používanie aplikácie z pohľadu používateľa. Na modelovanie diagramu bol využitý nástroj **Enterprise architect**.

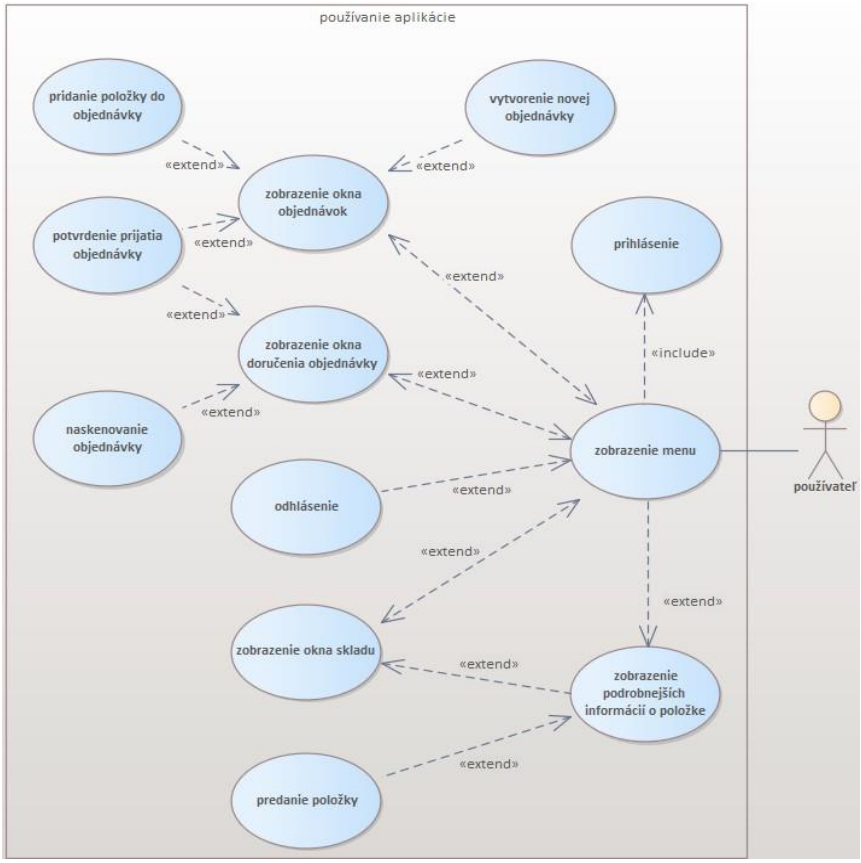
Aplikácia bude komunikovať s lokálnou databázou na ukladanie údajov, ktoré bude využívať alebo vytvárať samotná aplikácia. Na vytvorenie modelu bol použitý **nástroj Erwin data modeller**.

2.1 USE CASE DIAGRAM

Používateľ si môže zobraziť hlavné menu, ktoré je kľúčovým prvkom ovládania a navigácie v aplikácií. Na zobrazenie menu je potrebné úspešné prihlásenie. Z hlavného menu sa používateľ môže dostať do troch hlavných zobrazení. Prvým zobrazením sú doručené objednávky, ktoré poskytujú funkcie potvrdenia prijatia objednávky alebo tlačidlo na naskenovanie QR/čiarového kódu z balenia fyzickej objednávky.

Ďalším zobrazením sú objednávky. Toto zobrazenie umožňuje pridanie novej položky do novej objednávky a následne finálne vytvorenie a potvrdenie objednávky ako aj zobrazenie detailu objednávky.

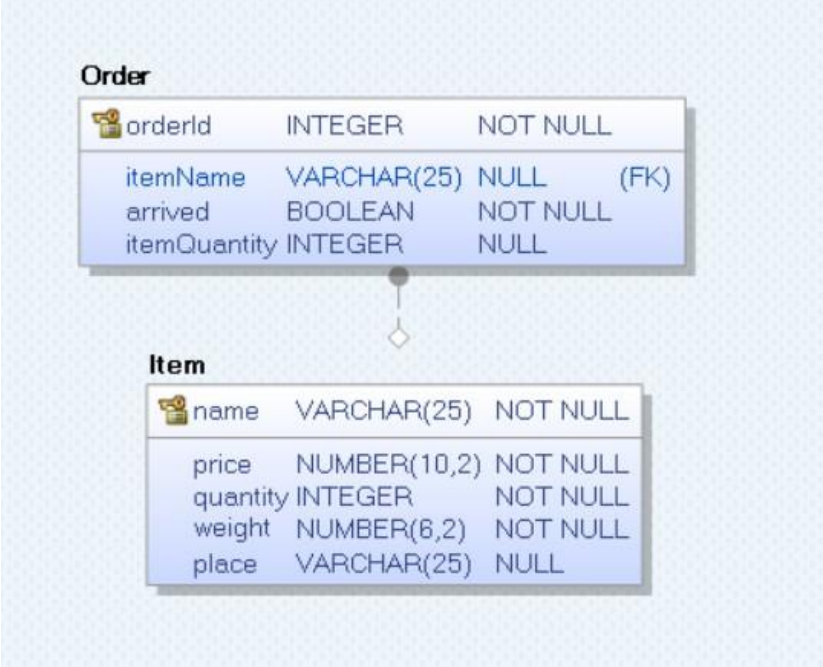
V zobrazení skladu sa nachádzajú údaje o všetkých položkách v sklade. Možné je zobrazenie aj podrobnejších informácií. Existuje možnosť predania položky po jednotlivých kusoch. Taktiež ako vyplýva z diagramu, z každého hlavného zobrazenia sa je možné vrátiť na menu aplikácie pomocou tlačidla.



Obrázok 4 - use case diagram

2.2 DÁTOVÝ MODEL APLIKÁCIE

Dátový model obsahuje tri entity. Entita **Order** predstavuje tabuľku pre ukladanie dát o objednávkach. V každej objednávke sa nachádzajú položky jedného typu, ktoré sú uložené v tabuľke **Item**. Pre jednoduchosť aplikácie bol dátový model navrhnutý tak, aby každá objednávka mohla obsahovať len položku jedného typu v ľubovoľnom počte.



Obrázok 5 - dátový model aplikácie

2.3 GRAFICKÁ PREDLOHA APLIKÁCIE

Grafická predloha aplikácie bola vytvorená pre potreby **checkpointu** v online prostredí **Figma** a aplikácia bola vyvíjaná aby sa čo najviac podobala na pôvodný návrh. Pri procese tvorby grafického rozhrania boli využívané prvky práve z tohto návrhu.

Obrázok 6 - pôvodný grafický návrh aplikácie

3 IMPLEMENTÁCIA

V tejto kapitole je podrobne opísaný proces tvorby aplikácie a rozobraté sú aj využité technológie a postupy dôležité pre hodnotenie tejto práce.

3.1 TVORBA UI

Na tvorbu UI aplikácie bol využívaný **Jetpack Composed**. Jedná sa o moderný nástrojový balík pre tvorbu natívnych používateľských rozhraní.

3.1.1 POUŽITIE JETPACK COMPOSED

Táto funkcia je kompozitná, označená anotáciou **@Composable**, čo znamená, že sa používa na vytvorenie komponentu používateľského rozhrania - tlačidla. Ako parameter sa tu nachádza **onStorage**, ktorý je **lambda** funkcia, ktorá je volaná po stlačení tlačidla. Taktiež sa tu využíva modifikátor na nastavenie veľkosti samotného tlačidla. Vizuál tlačidla tvorí obrázok ktorý je načítaný zo zdrojových súborov.

```
/**
 * funkcia tlačidla na prechod do sekcie StorageContent
 */
 Mario
@Composable
fun ButtonStorage(
    onStorage: () -> Unit
) {
    IconButton(
        onClick = { onStorage() },
        modifier = Modifier
            .size(180.dp)
    ) {
        Image(
            painter = painterResource(R.drawable.storage),
            contentDescription = (stringResource("icon")),
            modifier = Modifier.size(140.dp)
        )
    }
}
```

Obrázok 7 - ukážka funkcie pre tlačidlo

3.2 OTÁČANIE OBRAZOVKY – POUŽITIE SENZORA

Aplikácia využíva na orientáciu obrazovky senzor zariadenia, teda aplikácia sa otáča podľa polohy zariadenia. Pre uchovávanie hodnôt textových polí bolo nutné zabezpečiť ich uchovávanie aj po zmene orientácie. Pre ich zapamätanie na prihlasovacej obrazovke, sa používa funkcia **rememberSaveable**, ktorá slúži na uchovanie stavu komponentu tak, aby bol zachovaný aj po zmene orientácie obrazovky alebo iných zmenách.

```
Mario *
@Composable
fun LoginContent(onLoginSuccess: () -> Unit) {
    val context = LocalContext.current
    // rememberSaveable - sluzi na zapamätanie obsahu aj pri zmene orientácie aplikácie
    var LoginText by rememberSaveable { mutableStateOf("Login") }
    var PasswdText by rememberSaveable { mutableStateOf("Password") }
```

Obrázok 8 - riešenie problematiky zmeny orientácie

Pre lepšie a vizuálne krajšie rozloženie grafických komponentov na obrazovke, pri zmene orientácie, bol využívaný **list** v ktorom sa nachádzali hodnoty pre nastavovanie medzier medzi komponentami.

```
// list medzier na základe orientácie aplikácie
val spacerList = remember {
    if (orientation == Configuration.ORIENTATION_PORTRAIT) {
        listOf(155.dp, 50.dp, 16.dp, 90.dp) ^remember // portrait orientation
    } else {
        listOf(50.dp, 25.dp, 20.dp, 240.dp) ^remember // landscape orientation
    }
}
```

Obrázok 9 - riešenie problematiky zmeny orientácie 2

3.3 APLIKÁCIA FOTOAPARÁTU

Na obrazovke pre doručenie objednávky sa nachádza tlačidlo na skenovanie, momentálne je táto funkcionalita nedokončená avšak po kliknutí na tlačidlo sa otvorí fotoaparát, ktorý umožňuje zaznamenanie fotografie.

```
// premenna na spracovanie vysledkov z kamery
val cameraLauncher =
    rememberLauncherForActivityResult(
        ActivityResultContracts.StartActivityForResult()
    ) { /* spracovanie vysledku */ }
```

Obrázok 10 - aplikácia fotoaparátu

Po kliknutí sa v **lambda** bloku vytvorí akcia na zachytenie obrazu a následne sa spustí fotoaparát zariadenia.

```
@Composable
fun ScanButton(
    // parameter slúži na spustenie aktivity fotoaparátu
    cameraLauncher: ActivityResultLauncher<Intent>
) {
    IconButton(
        onClick = {
            // vytvorenie akcie
            val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // otvorenie fotoaplikacie zaznamenanie obrazku
            cameraLauncher.launch(takePictureIntent)
        },
```

Obrázok 11 - aplikácia fotoaparátu 2

3.5 VIEWMODELS

Viewmodels slúžia v tejto aplikácií na prácu s databázou, tvoria spojenie medzi grafickými prvkami jednotlivých obrazoviek a údajmi uloženými v databáze. Na obrázku nižšie je funkcia, ktorá označí objednávku za doručenú, teda jej atribút **arrived** nastaví na hodnotu **true**. Následne sa vykoná aj aktualizácia skladu pre pripočítanie doručených položiek z danej objednávky.

```
/**
 * funkcia na označenie objednávky za doručenú a aktualizovanie skladu
 */
Mario
fun markOrderAsDeliveredAndUpdateStorage(deliveredOrderId: Int) {
    viewModelScope.launch { this: CoroutineScope
        // najdenie objednavky na zaklade zadaného ID
        val deliveredOrder = orderRepository.getOrderStream(deliveredOrderId).firstOrNull()
        // oznacenie objednavky za dorucenu
        if (deliveredOrder != null) {
            orderRepository.updateOrder(deliveredOrder.copy(arrived = true))
        }
        // aktualizovanie skaldu
        if (deliveredOrder != null) {
            orderRepository.updateStorage(deliveredOrder)
        }
    }
}
```

Obrázok 12 - príklad funkcie viewmodelu

3.6 TVORBA DATABÁZY

Pre potreby vytvorenia lokálnej databázy bola zvolená knižnica **Room**, ktorá je navrhnutá tak, aby uľahčila prácu s lokálnou **SQLite** databázou v aplikáciách Android.

```
//Room
implementation("androidx.room:room-runtime:${rootProject.extra["room_version"]}")
ksp("androidx.room:room-compiler:${rootProject.extra["room_version"]}")
implementation("androidx.room:room-ktx:${rootProject.extra["room_version"]}")
```

Obrázok 13 - pridanie databázy Room do projektu

Následne boli vytvorené tabuľky objednávok a položiek podľa vyššie uvedeného návrhu dátového modelu. Následne bol vytvorený pre každú tabuľku **Data Access Object** ktorý poskytuje prístup k dátam v databáze.

```
@Transaction
suspend fun upsert2(item: Item, newQuantity: Int) {
    val existingItem = getItem(item.name).firstOrNull()
    if (existingItem != null) {
        // uprav mnozstvo
        val newQuantity = existingItem.quantity + newQuantity
        updateQuantity(item.name, newQuantity, item.price, item.place, item.weight)
    } else {
        // ak neexistuje tak insert
        insertNew(item)
```

Obrázok 14 - funkcia aktualizovania počtu kusov produktov

3.7 NAVIGÁCIA MEDZI OBRAZOVKAMI

Pre navigáciu medzi obrazovkami bola využívaná inštancia triedy **NavHostController**, v tejto funkcii sú definované destinácie (obrazovky) spolu z ich cestami a akciami, ktoré sa vykonajú po kliknutiach na konkrétne grafické elementy.

```
composable(route = MainDestination.route) { this: AnimatedContentScope  it: NavBackStackEntry
    MainContent(
        onLogout = { navController.navigate(LoginDestination.route) },
        onNewOrder = { navController.navigate(OrderDestination.route) },
        onReceiveOrder = { navController.navigate(ReceiveOrderDestination.route) },
        onStorage = { navController.navigate(StorageDestination.route) }
    )
}
```

3.8 EXTERNÉ KNIŽNICE/FRAMEWORKY

Využitý bol aj **Material Design**, ktorý poskytuje implementáciu dizajnových prvkov/komponentov pre **Jetpack Compose**. Využívaný bol napríklad pri použití komponentu **Scaffold**. Jedná sa o komponent, ktorý slúži ako základná kostra pre tvorbu obrazoviek v aplikáciách Android.

```
Scaffold(
    floatingActionButton = {
        FloatingActionButton(
            onClick = { onNewOrder() },
            shape = MaterialTheme.shapes.medium,
            modifier = Modifier
                // nastavenie paddingu

                .padding(
                    end = WindowInsets.safeDrawing.asPaddingValues()
                        .calculateEndPadding(LocalLayoutDirection.current)
                )
        ) {
```

Obrázok 15 – využitie komponentu Scaffold

3.9 TESTY

Pre potreby overenia funkčnosti databázy boli vytvorené aj testy podľa vzoru, ktorý sme si ukazovali na cvičeniach. Jedná sa o jednoduché overenie základných operácií ako **INSERT**, **UPDATE**, **DELETE** a kontrola ich očakávaných výsledkov.

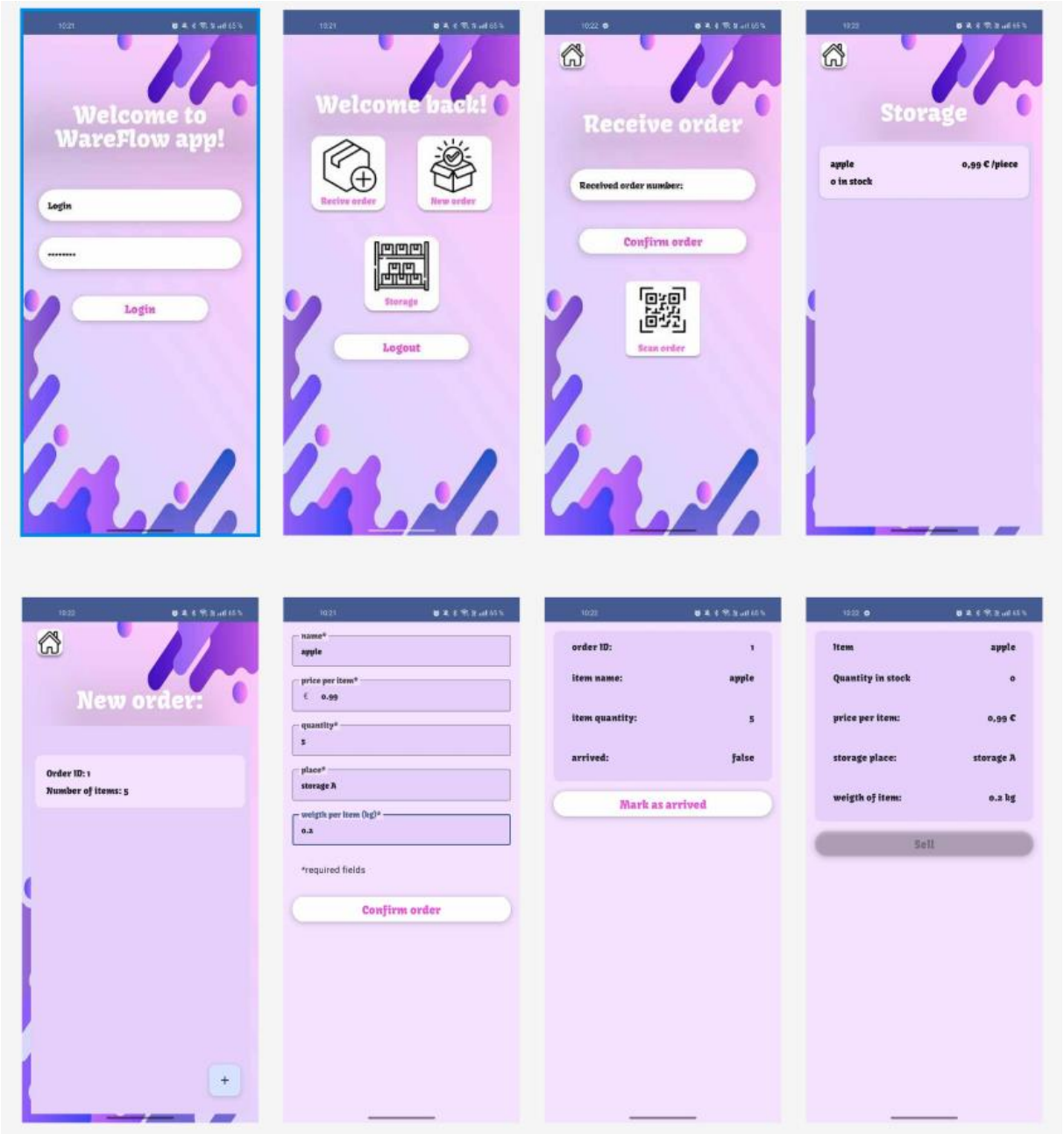
```
@Test
@Throws(Exception::class)
fun daoUpdateOrderes_updatesOrdersInDB() = runBlocking { this: CoroutineScope
    addTwoOrdersToDb()
    orderDao.update(Order( orderId: 1,  itemName: "Apples",  itemQuantity: 10,  arrived: true))
    orderDao.update(Order( orderId: 2,  itemName: "Apples",  itemQuantity: 20,  arrived: true))

    val allOrders = orderDao.getAllItems().first()
    Assert.assertEquals(allOrders[0], Order( orderId: 1,  itemName: "Apples",  itemQuantity: 10,  arrived: true))
    Assert.assertEquals(allOrders[1], Order( orderId: 2,  itemName: "Apples",  itemQuantity: 20,  arrived: true))
}
```

Obrázok 16 - príklad testu n overenie správneho fungovania príkazu UPDATE

4.0 FINÁLNY DIZAJN APLIKÁCIE

Snímky obrazoviek z finálnej verzie aplikácie, oproti návrhu sa mierne odlišuje v niektorých aspektoch, avšak aplikácia ako taká spĺňa vopred stanovený cieľ po funkčnej ale aj grafickej stránke.



Obrázok 17 - finálny dizajn aplikácie

ZDROJE

Pri tvorbe aplikácie boli využité niektoré časti kódu z projektu, ktorý je uvedený ako **4. zdroj**, tieto časti sú v zdrojovom kóde riadne označené v dokumentačných komentároch ako **„Prevzaté“** a v prípade ak prevzatý kód prešiel väčšími zmenami a úpravami tak je označený ako **„Upravený“**.

[1] Chester Software (Xaltos Technologies Ltd), „Google play,“ [Online]. Available: https://play.google.com/store/apps/details?id=com.stockmanagment.next.app&hl=sk&gl=US. [Cit. 01 04 2024].

[2] Archon Systems Inc., „Google play,“ [Online]. Available: https://play.google.com/store/apps/details?id=com.inflowinventory.mobile&hl=sk&gl=US. [Cit. 01 04 2024].

[3] SOFTIP, „softip.sk,“ [Online]. Available: https://www.softip.sk/sk/produkty/erp-systemy/nadstavby-rozsirenia/mobilny-skladnik/. [Cit. 2024 04 01].

[4] Google Codelabs, [Online]. Available: https://github.com/google-developer-training/basic-android-kotlin-compose-training-inventory-app.git