Medicine Supply

Documentatie

Informatica, Anul II, Grupa 10LF283

- Puchianu Alexandra-Georgiana
- Zaharia Alexandra

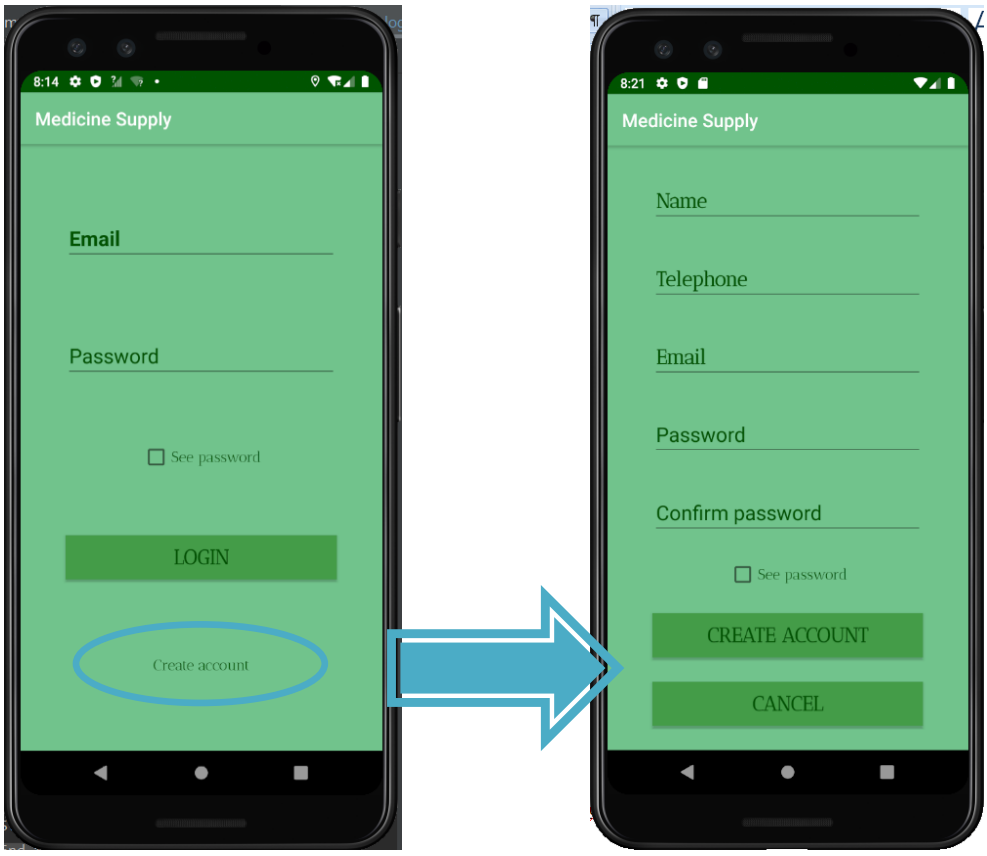
Universitatea Transilvania din Brasov
Facultatea de Matematica si Informatica

- Cuprins

- Prezentarea aplicatiei

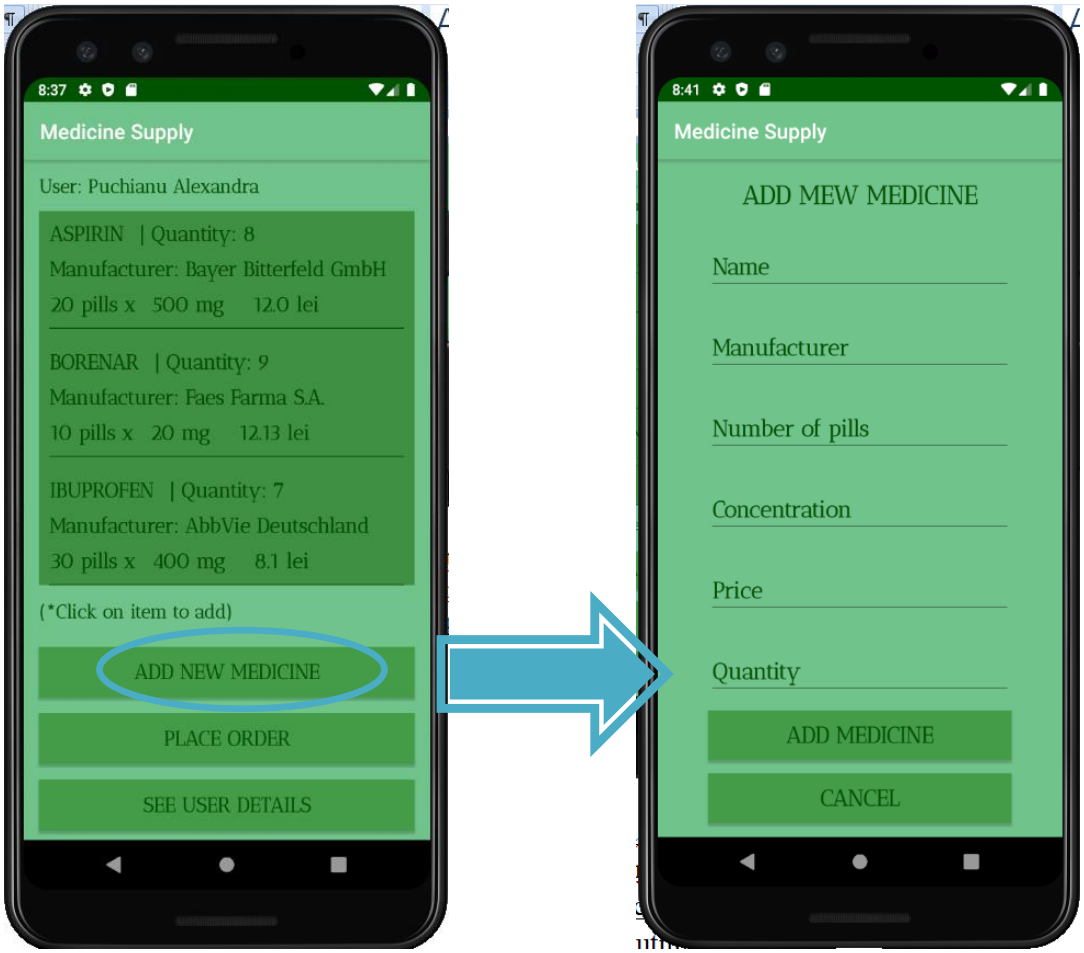
Aplicatia “*Medicine Supply*” se ocupa cu gestionarea stocului de medicamente al unei farmacii.

Aplicatia se deschide cu pagina de logare, unde utilizatorul trebuie sa completeze email-ul si parola pentru a intra in aplicatie apasand butonul “*Login*”, acesta poate de asemenea sa isi vada parola apasand pe “*See password*”. In cazul in care persoana nu are cont, aceasta isi poate crea unul, apansand pe butonul “*Create account*”.



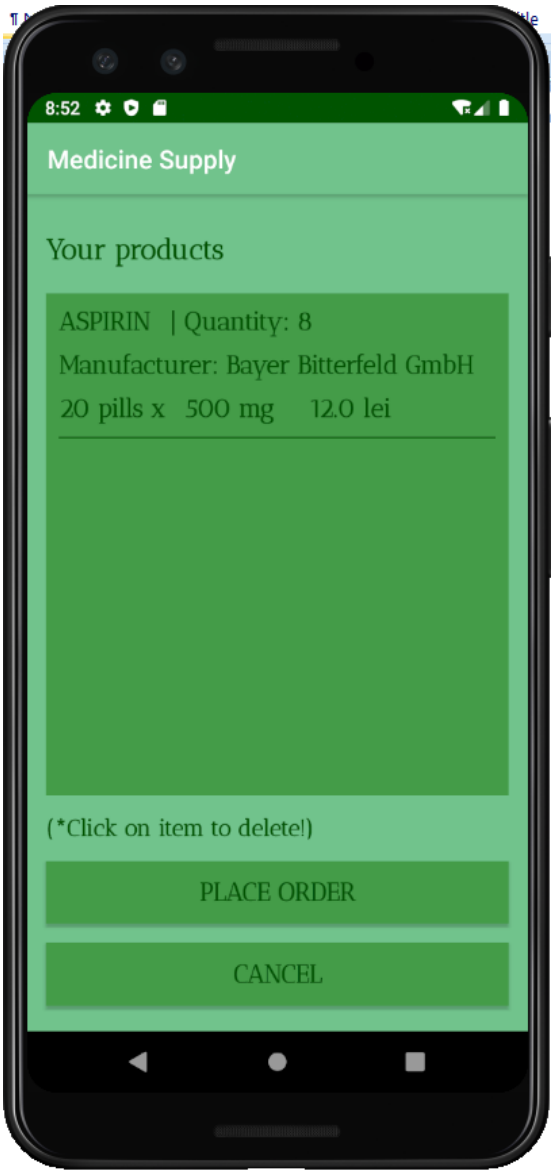
In pagina de “*Create account*” utilizatorul trebuie sa isi introduca numele, numarul de telefon, email-ul, parola si sa confirme parola. De asemenea butonul “*See password*” are aceeasi functionalitate ca si cel din pagina anterioara. Cand utilizatorul apasa pe butonul “*Create account*” contul va fi creat.

Dupa ce utilizatorul s-a logat cu success se deschide pagina cu stocul de medicamente prezent in farmacie. Acesta are optiunea de a adauga un medicament nou, a face o comanda sau a vedea detaliile contului sau.



Daca utilizatorul alege sa adauge un nou produs, atunci la apasarea butonului "*Add new medicine*" se va deschide o pagina unde el va putea introduce toate detaliile unui medicament (nume, producator, numar de comprimate, concentratie si pret), inclusiv cate bucati sa comande initial. Dupa ce sunt completate toate informatiile legate de medicament acesta este adaugat in baza de date prin apasarea butonului "*Add medicine*".

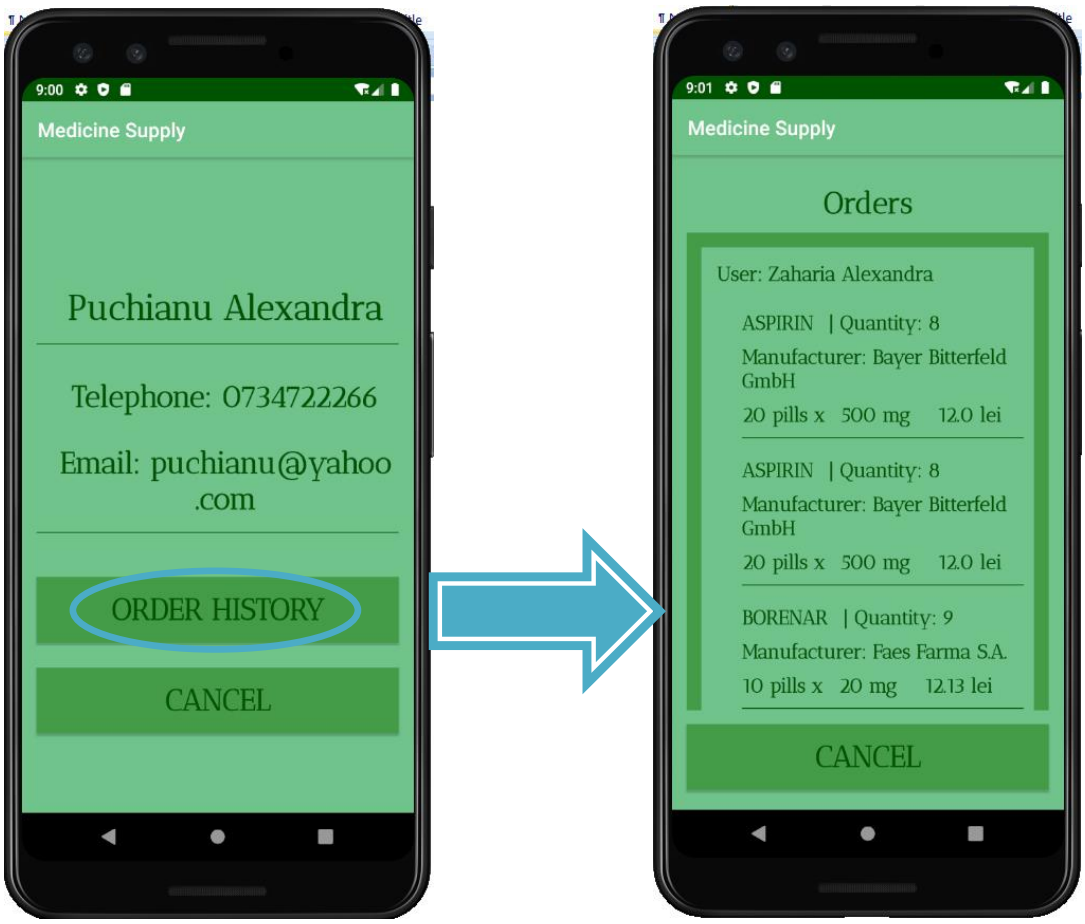
Daca utilizatorul doreste sa faca o comanda de produse, le poate adauga in cos apasand pe fiecare in parte, iar pentru a vedea produsele selectate va trebui sa apese butonul de "*Place order*". Dupa apasarea butonului se va deschide o fereastra unde este vizil cosul cu medicamentele selectate si optiunea de a plasa comanda. Daca unul dintre medicamnete nu mai este necesar, acesta poate fi sters din lista.



Daca utilizatorul apasa pe "*See user details*" din pagina cu stocul din farmacie, se va deschide o fereastra cu detaliile contului sau (nume, email si numar de telefon) si un buton cu textul "*Order history*".

Daca butonul "*Order history*" este apasat, va fi deschis istoricul comenzilor facute de toti utilizatorii din farmacia respectiva.

Navigarea intre ferestre se face prin intermediul butoanelor, iar butoanele "*Cancel*" au fost implementate astfel incat sa se intoarca la fragmentul anterior pentru a nu sari direct la activitatea anterioara. Acest lucru fost suprascris si pentru butonul "*Back*" in fiecare activitate.



- Utilitatea aplicatiei

Am conceput aceasta aplicatie pentru a usura munca farmacistilor si pentru ca ei sa poata rezolva anumite chestiuni administrative de pe telefon si a nu mai fi blocati din cauza unui numar mic de calculatoare.

Farmacistii pot face comenzi de medicamente si pot de asemenea vedea comenzile din trecut cu ajutorul acestei aplicatii.

- Arhitectura si tehnologie

Aplicatia a fost creata in Android Studio in limbajul Java, iar pentru realizarea sa am folosit baza de date Room.

Am structurat aplicatia in 3 activitati (`LoginActivity`, `InventoryActivity` si `ReportsActivity`), fiecare dintre ele avand mai multe fragmente.

- ✓ LoginActivity: `LoginFragment, NewAccountFragment`
- ✓ InventoryActivity: `InventoryFragment, NewMedicineFragment, OrderFragment`

- ✓ ReportsActivity: `UserFragment, OrderHistoryFragment`

Atunci cand un utilizator incearca sa isi creeze un cont, am implementat validari pentru fragmentele in care introduce propriul input, spre exemplu, validare pentru email si pentru numarul de telefon. Pentru validarea email-ului am inclus formatul sau (cu regex) si am luat in considerare si cazul in care exista deja un alt utilizator cu acelasi email.

```
private boolean validateEmail(String email) {
    final String regex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(email);
    return matcher.matches();
}

private boolean existingEmail(String email) {
    User user = null;
    AsyncTask<String, Void, User> userAsyncTask = new SelectUserByEmailAsync().execute(email);
    try {
        user = userAsyncTask.get();
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    if (user == null) {
        return false;
    } else {
        return true;
    }
}
```

```
private boolean validateTelephone(String telephone) {
    if (!Character.isDigit(telephone.charAt(0)) && telephone.charAt(0) != '+') {
        return false;
    }

    for (int index = 1; index < telephone.length(); index++) {
        if (!Character.isDigit(telephone.charAt(index))) {
            return false;
        }
    }

    return true;
}
```

Pentru retinerea datelor am folosit o baza de date Room si un fisier json.

Baza de date are ca entitati `User, Medicine, Order` si `MedicineOrder` (care face legatura intre o comanda si produsele sale). Fiecare entitate este creata cu ajutorul unei clase si contine o cheie primara, atribute, constructori cu si fara parametri si setteri si getteri pentru atribute.

```
@Database(entities = {User.class, Medicine.class, Order.class, MedicineOrder.class}, version = 2)
public abstract class AppDatabase extends RoomDatabase {

    public abstract UserDao userDao();
    public abstract MedicineDao medicineDao();
    public abstract OrderDao orderDao();
    public abstract MedicineOrderDao medicineOrderDao();
}
```

```
@Entity
public class User {

    @PrimaryKey(autoGenerate = true)
    private int id;
    @ColumnInfo(name = "name")
    private String name;
    @ColumnInfo(name = "email")
    private String email;
    @ColumnInfo(name = "password")
    private String password;
    @ColumnInfo(name = "telephone")
    private String telephone;

    public User(String name, String email, String password, String telephone) {
        this.name = name;
        this.email = email;
        this.password = password;
        this.telephone = telephone;
    }

    public User() {
        id = 0;
        name = "";
        email = "";
        password = "";
        telephone = "";
    }
```

Baza de date este accesata printr-un `Controller` si obiecte de tip `Dao` (Data Access Object). Am creat acest `Controller` pentru a putea avea acces la baza de date din orice activitate. Obiectele de tip `Dao` sunt implemetate in interfete care contin metode pentru a selecta,

sterge sau edita intrari din baza de date. `Controller`-ul acceseaza obiectele `Dao` atunci cand face operatii.

```java
public class AppController extends Application {

    private static AppController controller;
    private AppDatabase database;

    public static AppController getInstance() { return controller; }

    @Override
    public void onCreate() {
        super.onCreate();

        controller = this;
        database = Room.databaseBuilder(getApplicationContext(), AppDatabase.class, name: "app_database").build();
    }

    public AppDatabase getDatabaseInstance() { return database; }
}
```

```java
@Dao
public interface MedicineDao {
    @Insert
    void insertMedicine(Medicine medicine);

    @Query("UPDATE Medicine SET quantity=:quantity WHERE id=:id")
    void updateQuantity(int quantity, int id);

    @Query("SELECT * FROM Medicine WHERE id=:idMedicine")
    Medicine selectMedicine(int idMedicine);

    @Query("SELECT * FROM Medicine")
    List<Medicine> selectAllMedicines();
}
```

Pentru ca operatiile care au legatura cu baza de date sa nu se desfasoare pe thread-ul principal am folosit `AsyncTask`. O clasa care extinde `AsyncTask` trebuie neaparat sa contina functia `doInBackground`, unde se desfasoara activitatea bazei de date si functia `onPostExecute`, pentru ca in cazul in care `AsyncTask`-ul returneaza o valoare aceasta sa poata fi accesata.

```java
public class SelectMedicineAsync extends AsyncTask<Integer, Void, Medicine> {
    @Override
    protected Medicine doInBackground(Integer... integers) {
        Medicine medicine = AppController.getInstance().getDatabaseInstance().medicineDao().selectMedicine(integers[0]);
        return medicine;
    }

    @Override
    protected void onPostExecute(Medicine medicine) {
        super.onPostExecute(medicine);
    }
}
```

Am creat clase `RecyclerView.Adaptor` pentru a putea afisa liste de obiecte in `RecyclerView`. Scopul clasei `Adaptor` este de a adapta un obiect sau entitate intr-un layout creat pentru fiecare item din `RecyclerView`-ul respectiv.

```java
public class MedicineAdapter extends RecyclerView.Adapter<MedicineAdapter.ViewHolder> {

    private List<Medicine> medicines;

    public MedicineAdapter(List<Medicine> medicines) { this.medicines = medicines; }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.product_item, parent,  attachToRoot: false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        int quantity = medicines.get(position).getQuantity();
        int noPills = medicines.get(position).getNoPills();
        int concentration = medicines.get(position).getConcentration();
        double price = medicines.get(position).getPrice();

        holder.txtProductName.setText(medicines.get(position).getName());
        holder.txtQuantity.setText("| Quantity: " + Integer.toString(quantity));
        holder.txtManufacturer.setText("Manufacturer: " + medicines.get(position).getManufacturer());
        holder.txtPills.setText(Integer.toString(noPills) + " pills x ");
        holder.txtConcentration.setText(Integer.toString(concentration) + " mg    ");
        holder.txtPrice.setText(Double.toString(price) + " lei");
    }
```

si

```java
    @Override
    public int getItemCount() { return medicines.size(); }

    public class ViewHolder extends RecyclerView.ViewHolder {

        TextView txtProductName;
        TextView txtQuantity;
        TextView txtManufacturer;
        TextView txtPills;
        TextView txtConcentration;
        TextView txtPrice;

        public ViewHolder(@NonNull View itemView) {
            super(itemView);
            txtProductName = itemView.findViewById(R.id.txtProductName);
            txtQuantity = itemView.findViewById(R.id.txtQuantity);
            txtManufacturer = itemView.findViewById(R.id.txtManufacturer);
            txtPills = itemView.findViewById(R.id.txtPills);
            txtConcentration = itemView.findViewById(R.id.txtConcentration);
            txtPrice = itemView.findViewById(R.id.txtPrice);
        }
    }
}
```

In clasa `Adaptor` se gaseste si clasa `ViewHolder` care face legatura dintre obiect layout-ul creat pentru fiecare item din `RecyclerView`.