SEOlympic Application

~ Application for efficient work ~

Szoftver Rendszerek Tervezése

Informatika III.

Készítette:

- Benkő Edina
- Makkai Mátyás

Vezető tanár:

Szántó Zoltán

Sapientia, Marosvásárhely
2019

Tartalom

1. Bevezető

1.1 A dolgozat témája

Dolgozatunk témáját, amikor kigondoltuk, azt vettük figyelembe, hogy mi az, amire nekünk szükségünk volna és segítene a mindennapokban, mindennapi munkánkban.

Arra gondoltunk, hogyha lenne egy olyan applikáció, amely saját készítésű és azokat valósítja meg, amikre pontosan szükségünk van milyen jó volna. Ezért átgondoltuk, és összeírtunk egy listát a szükséges funkcionalitásokkal.

Az SEOlympic App saját vállalkozásunknak szolgál termékként, amelyben néhány alapvető funkció található meg egy helyen. Segít minket fontos adatokat eltárolni, emlékeztet a teendőinkre és egyéb ehhez hasonló funkcionalitásokkal áll a rendelkezésünkre.

1.2 A dolgozat célja

Az alkalmazásunk célja az, hogy egy olyan alapot szolgáljon számunkra, amelyre majd lehet még építeni. Kezdetekben néhány alap koncepcióval segítse a munkánkat, pont úgy, mint nagyobb vállalatoknál a saját rendszerek, amelyekre külön informatikusokat kérnek fel a megvalósításra.

Fontosnak tartom a célok között megemlíteni azt is, hogy ez az alkalmazás egy helyen több olyan funkcióval is rendelkezik, amelyeket eddig különböző ingyenes vagy fizetős forrásokból oldottunk meg. Ez azért fontos, mert ezekhez az adatokhoz egyszerre tudunk hozzáférni, rengeteg időt megspórolva a böngészőben több ablak külön megnyitásával és rákeresésével.

2. Követelmények

2.1 Felhasználói követelmények

Az alkalmazásunk megtervezése előtt, össze írtunk néhány olyan alap követelményt, amelyet mindenképp teljesítenie kell, hiszen ezek nélkül nehéz volna elképzelni a megfelelő működést.

Az elvárt követelmények a következők:

- Regisztráció:
    - nincs szükség külön regisztrációra.
    - a felhasználók a rendszerben vannak létrehozva.

- abban az esetben, ha egy új személyt szeretnénk hozzá adni az alkalmazásunkhoz, akkor azt nekünk kell létrehozni, mert nincs külön regisztrációs lehetőség.

- Bejelentkezés:
  - két külön felhasználóra van szükség.
  - különböző adatok eltárolása a felhasználóknak.
  - ha bejelentkezik csak a saját elmentett adatait tudja megjeleníteni.
  - email cím és jelszó alapján azonosítja és különíti el a felhasználókat.
- E-mail küldés:
  - minden felhasználónak lehetősége van e-mailt küldeni a saját címéről.
- Jegyzetek használata:
  - mindenki tud saját jegyzeteket létrehozni, hozzá adni.
  - ezen jegyzeteket tudja módosítani és törölni.
  - a jegyzeteket egy görgethető listában tudja megtekinteni.
  - ha a listában egyre rákattint, akkor egy külön ablak jelenik meg, ahol eldönti a továbbiakat.
- Emlékeztető:
  - betud állítani mindenki emlékeztetőt bizonyos teendőről, amely az alkalmazás lezárása után is értesítést küld a felhasználónak.
- Kliens adatok tárolása:
  - görgethető listában tudja megtekinteni az elmentett adatokat.
  - ha egyre rákattint, akkor van lehetősége törölni.
  - külön gombbal tud hozzá adni adatokat egy ív kitöltésével.

2.2 Rendszerkövetelmények

- Android operációs rendszerrel rendelkező eszköz.
- SQLite Adatbázis.
- Minimum SDK verzió: API level (15), Android 4.0.3 (IceCreamSandwich).
- Compile SDK verzió: API level (28), Android 9.0 (Pie).
- Internet csatlakozás.

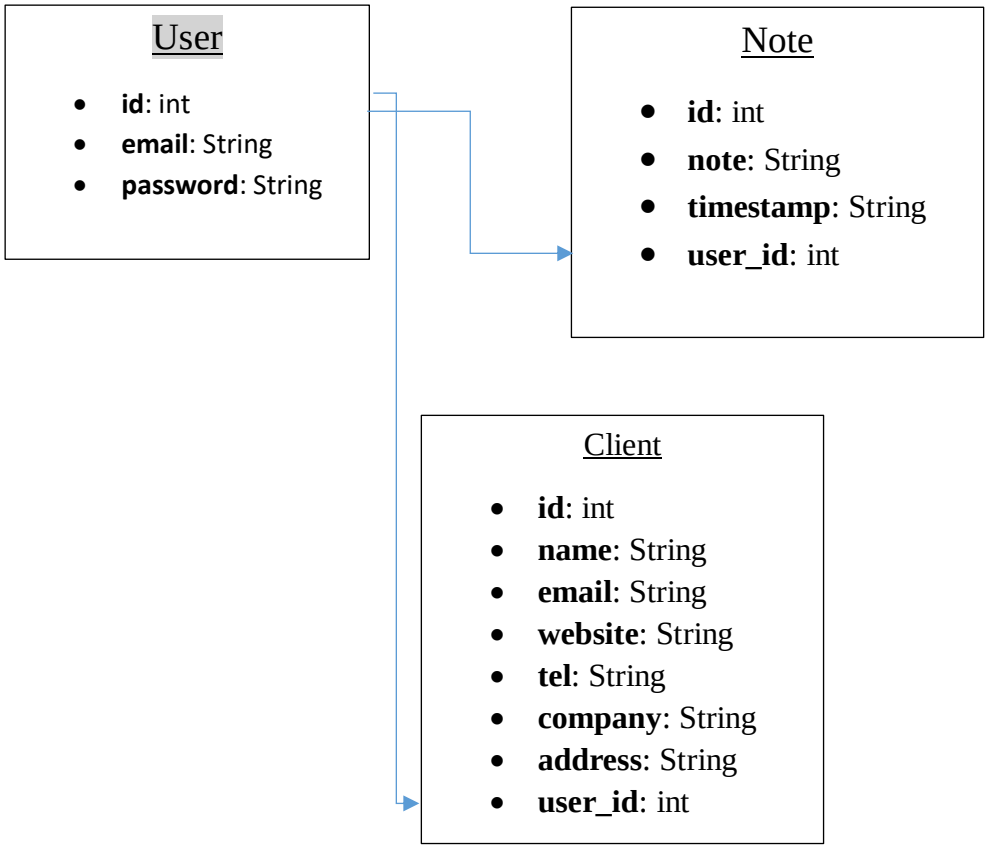
2.3 Alkalmazott rendszerkomponensek

- Adatbázis: SQLite.
- Android:
  - RecyclerView.

- LinearLayout.
- Activity.
- XML layout files
- stb.

3. Tervezés

3.1 Osztálydiagram



Három külön osztályt használtunk a funkcionalitások megvalósításához:

1. User osztály:
    - amelyben az id az egyedi kulcs.
    - ennek az osztálynak a felelőssége, hogy eltárolja és megkülönböztesse a felhasználókat id alapján.
2. Note osztály:
    - amelyben az id az egyedi kulcs, a user_id pedig külső kulcs, amely azt jelöli, hogy az adott Note melyik user-hez tartozik.
    - az osztály felelőssége az, hogy user_id szerint és id alapján megkülönbözteti a jegyzeteket.

3. Client osztály:
    - amelyben az id az egyedi kulcs, a user_id pedig külső kulcs, amely azt jelöli, hogy az adott Client melyik user-hez tartozik
    - az osztály felelőssége az, hogy user_id szerint és id alapján megkülönbözteti a klienseket.

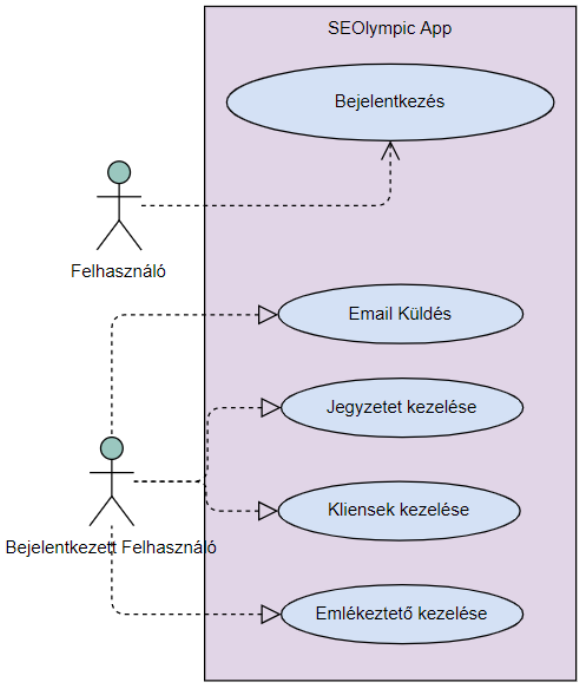
Az adatbázisban is ennek az elképzelésnek az alapján vannak létrehozva a táblák és a kapcsolatok:

```
//create table statements
private static final String CREATE_TABLE_USER = "CREATE TABLE "
        + TABLE_USER + "(" + USER_ID + " INTEGER PRIMARY KEY," + USER_EMAIL
        + " TEXT," + USER_PW + " TEXT" + ")";

private static final String CREATE_TABLE_NOTE = "CREATE TABLE "
        + TABLE_NOTE + "(" + NOTE_ID + " INTEGER PRIMARY KEY," + NOTE_NOTE
        + " TEXT," + NOTE_TIMESTAMP + " DATETIME," + USER_ID + " INTEGER" + ")";

private static final String CREATE_TABLE_CLIENTS = "CREATE TABLE "
        + TABLE_CLIENT + "(" + CLIENT_ID + " INTEGER PRIMARY KEY," + CLIENT_NAME
        + " TEXT," + CLIENT_EMAIL + " TEXT," + CLIENT_WEBSITE + " TEXT," + CLIENT_TEL
        + " TEXT," + CLIENT_COMPANY + " TEXT," + CLIENT_ADDRESS + " TEXT," + USER_ID + " INTEGER" + ")";
```

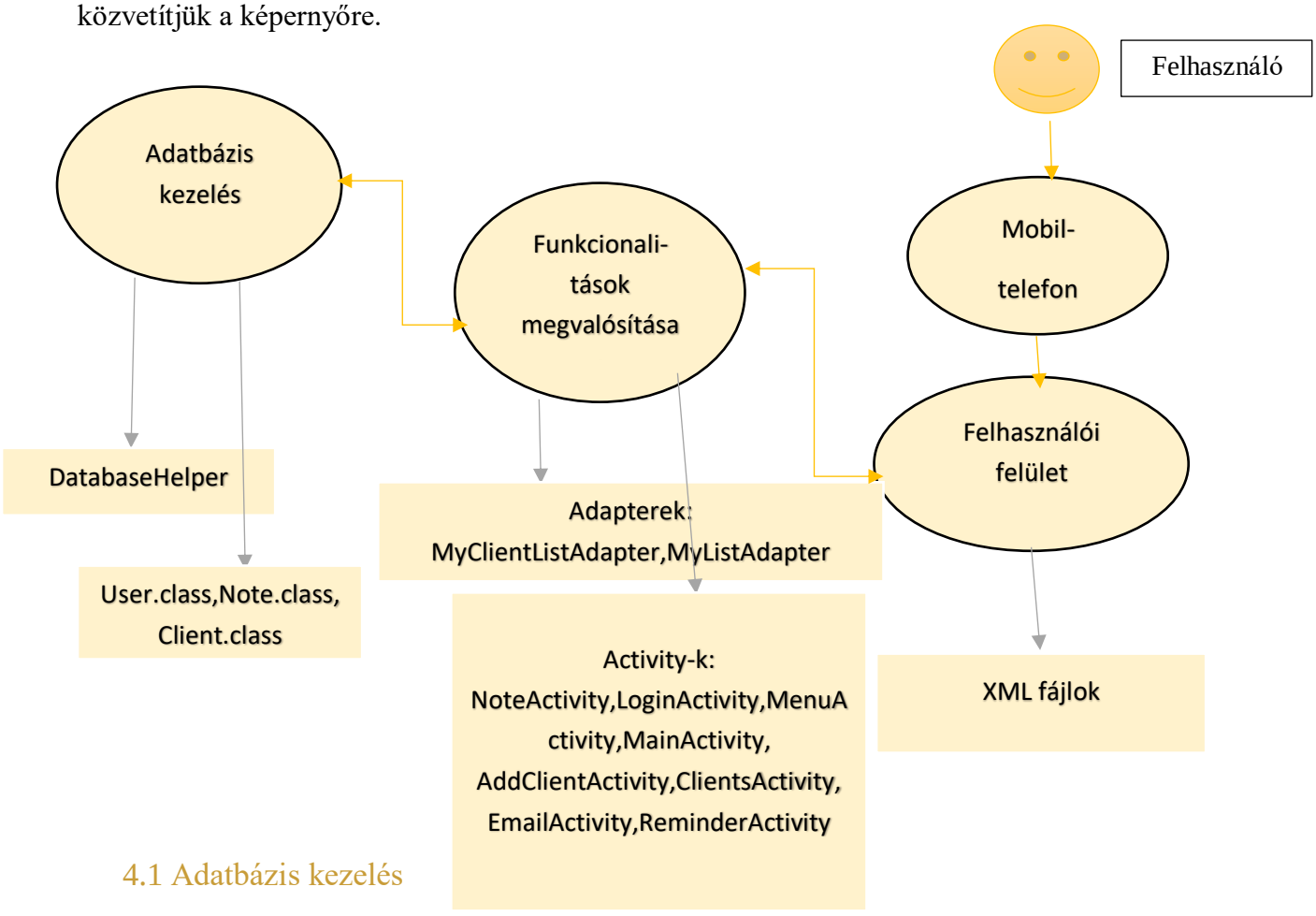
3.2 Use case diagram



4. Architektúra

Alkalmazásunk egy rétegelt architektúrával lett megtervezve, miszerint van benne egy adatbázis kezelő rész (DatabaseHelper), felhasználói felület (UI – User Interface), funkcionalitások megvalósítása (Activity-k, Adapterek, Osztályok).

Mindhárom réteg szoros kapcsolatban áll egymással, mivel mindegyik tevékenység hozzá járul a másik tevékenységéhez, és így tudnak egy közös végeredményt elérni. Például gondolok itt most a DatabaseHelper osztályban elkészített adatbázis kezelő részeket felhasználjuk a funkcionalitások megvalósításánál, mely eredményét a felhasználói felület keretén belül közvetítjük a képernyőre.



4.1 Adatbázis kezelés

Az adatbázis kezelés az SEOlympic alkalmazásban SQLite adatbázissal van megvalósítva. Ennél pontosabban egy osztályban vannak megírva az adatbázis műveletek, amelynek a neve DatabaseHelper.class.

Ebben az osztályban megvannak írva a következők:

- tábla létrehozása: User, Note, Client táblák.

- ide szükséges volt létrehozni egy User.class,Note.class és Client.class osztályokat, amelyeket segítenek az objektumok létrehozásában és megkönnyítik az elkövetkezendő adatbázis műveleteket is. Ez a három osztály modellként szolgál nekünk-

- CRUD műveletek.
- különböző lekérdezések, amelyek listáznak bizonyos adatokat.
- Kódrészletek:

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "datamanager";

    //table names
    private static final String TABLE_USER = "user";
    private static final String TABLE_NOTE = "note";
    private static final String TABLE_CLIENT = "client";
```

//egy adott note lekérése id szerint

```
public Note getANote(long note_id)
{
    SQLiteDatabase db = this.getReadableDatabase();
    String selectQuery = "SELECT  * FROM " + TABLE_NOTE + " WHERE "
            + NOTE_ID + " = " + note_id;

    Cursor c = db.rawQuery(selectQuery,  selectionArgs: null);
    if (c != null)
        c.moveToFirst();

    Note n = new Note();
    n.setUser_id(c.getInt(c.getColumnIndex(USER_ID)));
    n.setNote((c.getString(c.getColumnIndex(NOTE_NOTE))));
    n.setTimestamp(c.getString(c.getColumnIndex(NOTE_TIMESTAMP)));

    return n;
}
```

- hasonló lekérdezések: egy userhez tartozó összes kliens lekérése, egy userhez tartozó összes jegyzet lekérése, jegyzet/kliens lekérése id szerint, kliens/jegyzet törlése/módosítása, stb.

4.2 Funkcionalitások megvalósítása, kapcsolat:

A kódolás valós részét fedi, amely szoros kapcsolatban áll a felhasználói felülettel és az adatbázis kezeléssel is. Ez az, ami megteremti minden között a kapcsolatot. Például, feldolgozza, az adatbázis adatait a megfelelő módon majd azt továbbítja a felhasználó számára.

Ez oldja meg a bejelentkezést, és az adott felhasználóhoz tartozó adatok elkülönítését.

Megvalósítja kódban a listázást, például a használatban levő RecyclerView-hoz tartozó Adaptert, amelybe kilistázza az adatbázisból lekért adatokat és XML fájlokban van meg ennek a kinézete.

Példa:

Itt látható az összes felhasználóhoz tartozó jegyzet lekérése az adatbázisból egy listába.

List<Note> allNotes;

```
//we get all the Notes of the user
allNotes = db.getUserNotes(id);
for (Note note : allNotes) {
    Log.d( tag: "Notes", note.toString());
}
```

Az allNotes lista Note objektumokat tartalmaz.

A lekért listát tovább adjuk a RecyclerView adatperének, amely feldolgozza őket egyenként és kilistázza őket.

```
//RecyclerView setup
recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
recyclerView.addItemDecoration(new DividerItemDecoration( context: NoteActivity.this,DividerItemDecoration.VERTICAL));
adapter = new MyListAdapter(allNotes, context: NoteActivity.this,db);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
recyclerView.setAdapter(adapter);
adapter.notifyDataSetChanged();
```

A back-end részhez tartozó egyéb részletek:

- Bejelentkezés megvalósítása valós felhasználóval.
    - egy listába lekérjük az adatbázisban már előre létrehozott felhasználókat
    - gombnyomásra leellenőrizzük a bemenetet és azt, hogy talál-e a felhasználónév és a jelszó az adatbázisban levőkkel, ha minden rendben akkor beengedjük a felhasználókat, úgy hogy tovább küldjük az id-jukat.

```
//we get the users from the database
final List<User> users = db.getAllUsers();
button_login.setOnClickListener((view) → {
        EditText et_email = findViewById(R.id.et_email);
        EditText et_password = findViewById(R.id.et_password);
        String email = et_email.getText().toString();
        String password = et_password.getText().toString();
        //checking if the fields are empty
        if (email.isEmpty() || password.isEmpty())
        {
            et_email.setError("All the fileds are required");
            et_password.setError("All the fileds are required");
            Toast.makeText(getApplicationContext(), text: "All the fields are required", Toast.LENGTH_SHORT).show();

        }
        else {
            User userEdina = users.get(0);
            User userMatyas=users.get(1);
            //we check the login information
            if(email.matches(userEdina.getEmail()) && password.matches(userEdina.getPassword()))
            {
                Toast.makeText(getApplicationContext(), text: "Hello Edina!", Toast.LENGTH_SHORT).show();
                Log.d( tag: "Users", userEdina.toString());
                //passing the user id, and go to MenuActivity
                Intent intent = new Intent( packageContext: Login.this, Menu.class);
                intent.putExtra( name: "Id", value: 1);
                startActivity(intent);
            }
            if(email.matches(userMatyas.getEmail()) && password.matches(userMatyas.getPassword()))
            {
                Toast.makeText(getApplicationContext(), text: "Hello Matyas!", Toast.LENGTH_SHORT).show();
                Log.d( tag: "Users", userMatyas.toString());
                //passing the user id, and go to MenuActivity
                Intent intent = new Intent( packageContext: Login.this, Menu.class);
                intent.putExtra( name: "Id", value: 2);
                startActivity(intent);
            }
        }
});
```

- Client-ek lekérése, feldolgozása és listázása, hozzáadása
    - így adunk hozzá egy klienset az adatbázishoz, gombnyomásra lekérjük a bemenetet, létrehozunk egy Client objektumot a bekért adatokból, és hozzá rendelünk egy egyedi kulcsot a checkMaxId() függvény használatával, amely megnézi az eddigi legnagyobb kliens id-t, és így akkor a plusz egyedikre fogjuk ezt létrehozni, ennek a bemenő paramétere az adatbázisból már lekért kliens lista, az allClientA.

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
final  int id = extras.getInt( key: "id");
Log.d( tag: "id",  msg: id+"");
db = new DatabaseHelper(getApplicationContext());
final  List<Client> allClientA = db.getAllClients();
for(Client c : allClientA)
{
    Log.d( tag: "Actual client id's",  msg: c.getId()+"");
}
Log.d( tag: "Id for the next client", msg: checkMaxId(allClientA)+1+"");
 et_name = findViewById(R.id.et_name);
 et_email = findViewById(R.id.et_email);
 et_website = findViewById(R.id.et_website);
 et_tel = findViewById(R.id.et_tel);
 et_company =findViewById(R.id.et_company);
 et_address = findViewById(R.id.et_address);
Button bt_addClient = findViewById(R.id.bt_addClientForm);
bt_addClient.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //the completed field contents
        String name = et_name.getText().toString();
        String email = et_email.getText().toString();
        String website = et_website.getText().toString();
        String tel = et_tel.getText().toString();
        String company = et_company.getText().toString();
        String address = et_address.getText().toString();
        //adding client to the database
        Client client = new Client( id: checkMaxId(allClientA)+1,name,email,website,tel,company,address,id);
        Log.d( tag: "The new Client", client.toString());
        db.createClient(client);
        //go back with ok resultCode to the calling Activity
        Intent intent=new Intent();
        setResult( resultCode: 1,intent);
    }
});

hecks the maxId for the clients
```

- ha ráklikkelünk a kilistázott kliens objektumok közül egyre, akkor lehetőségünk van azt kitörölni, amelynek a megvalósítása a MyClientListAdapterben történik, egy AlertDialog segítségével.

```
//if we click on an item
holder.linearLayout.setOnClickListener((view) → {
        Log.d( tag: "Position", msg: position+"");
        final AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setTitle("Your Note");
        builder.setMessage("Do you want to delete this contact?");

        //we have the posibility to delete the requested contact item
        builder.setPositiveButton( text: "Yes", (dialogInterface, i) → {
                Client client = clientList.get(position);
                //delete from the database, notify the adapter and update the adapters list
                clientList.remove(position);
                notifyItemRemoved(position);
                db.deleteAClient(client.getId());
        });
        builder.setNeutralButton( text: "Cancel", listener: null);
        AlertDialog dialog = builder.create();
        dialog.show();

});
```

```
//recyclerview setup
recyclerView = (RecyclerView) findViewById(R.id.recyclerViewClients);
recyclerView.addItemDecoration(new DividerItemDecoration( context: ClientsActivity.this,DividerItemDecoration.VERTICAL));
adapter = new MyClientListAdapter(allClients, context: ClientsActivity.this,db);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
recyclerView.setAdapter(adapter);
adapter.notifyDataSetChanged();
```

- kliensek kilistázására RecyclerView-t használtunk, a fenti kódrészletben állítuk be és hívjuk meg az adaptert az allClients listára, amely tartalmazza az adott userhez tartozó összes klienset.

- Note-ok lekérése, feldolgozása és listázása

```
//RecyclerView setup
recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
recyclerView.addItemDecoration(new DividerItemDecoration( context: NoteActivity.this,DividerItemDecoration.VERTICAL));
adapter = new MyListAdapter(allNotes, context: NoteActivity.this,db);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
recyclerView.setAdapter(adapter);
adapter.notifyDataSetChanged();
```

- az összes userhez tartozó note kilistázására is recyclerView-t használtunk, amelyet az allNotes tömbre hívunk meg.

```
private void openDialog(final int id)
{
    //alertDialog view
    LayoutInflater inflater = LayoutInflater.from(NoteActivity.this);
    View subView = inflater.inflate(R.layout.alert_layout,  root: null);

    //get the alertDialog item
    final EditText et_note = subView.findViewById(R.id.update);

    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("Add Your Note");
    builder.setView(subView);
    AlertDialog alertDialog = builder.create();

    //adding a New Note to the database and refresh the adapter/recyclerView
    builder.setPositiveButton( text: "Add", (dialog, which) → {

            Log.d( tag: "Note from the EditText",et_note.getText().toString());
            Note note = new Note ( id: checkMaxId()+1,et_note.getText().toString(),dateFormat(),id);
            db.createNote(note);
            allNotes.add(note);
            adapter.notifyDataSetChanged();

    });
    builder.show();
}
```

- gombnyomásra hívódik meg ez a függvény abban az esetben, ha új jegyzetet szeretnénk hozzáadni az adatbázishoz. Felugrik egy új AlertDialog ablak, saját dizájnal, amely az alert_layout xml fájlban van megírva, bekérjük az adatokat, majd létrehozunk egy objektumot amelyet az adatbázishoz és az allNotes tömbhöz adjuk hozzá, és jelzünk az adapternek is, hogy frissítse a listát.

```
//alertDialog Modify Button functionality
builder.setPositiveButton( text: "Modify", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {

        //new alertDialog for the modified elements
        final View view = LayoutInflater.from(context).inflate(R.layout.alert_layout,  root: null);
        AlertDialog.Builder dialog = new AlertDialog.Builder(context);
        dialog.setView(view);
        dialog.setPositiveButton( text: "Done", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                EditText et_update = view.findViewById(R.id.update);
                Log.d( tag: "Update", et_update.getText().toString());
                Note note = noteList.get(position);
                Note noteUp = new Note(note.getId(),et_update.getText().toString(),note.getTimestamp(),note.getUser_id());

                //we update the  list, adapter, database
                noteList.set(position,noteUp);
                notifyItemChanged(position);
                db.updateNote(noteUp);
            }
        });
        AlertDialog d= dialog.create();
        d.show();
    }
});
```

```
//alertDialog Delete Button functionality
builder.setNeutralButton( text: "Delete", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        Note note = noteList.get(position);

        //remove from the database, and notify the adapter, and update the list
        noteList.remove(position);
        notifyItemRemoved(position);
        db.deleteOneNote(note.getId());
    }
});
```

- hogyha rákattintunk a listánkban levő egyik jegyzet elemre, akkor újra felugrik egy AlertDialog ablak, amelyben kitudjuk választani, hogy töröljük vagy módosítjuk az adott jegyzetet. Ezen funkcionalitások a MyListAdapterben vannak megvalósítva az onclick metódusban.

- Email küldése saját email címről

```
final EditText etTo;
final EditText etMessage;
final EditText etSubject;
Button bt_send;

etMessage = findViewById(R.id.et_message);
etTo = findViewById(R.id.et_to);
etSubject = findViewById(R.id.et_subject);
bt_send = findViewById(R.id.bt_send);

bt_send.setOnClickListener((view) -> {
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("mailto:" + etTo.getText().toString()));
        intent.putExtra(Intent.EXTRA_SUBJECT,etSubject.getText().toString());
        intent.putExtra(Intent.EXTRA_TEXT,etMessage.getText().toString());
        startActivity(intent);
});
```

- Emlékeztető beállítása

- Menü beállítása

```
//header Menu
public boolean onOptionsItemSelected(MenuItem item) {

   Intent mainIntent = getIntent();
    Bundle extras = mainIntent.getExtras();
    int id = extras.getInt( key: "Id");

    switch (item.getItemId()) {
        case R.id.note:
            Toast.makeText( context: this,  text: "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
             Intent intent = new Intent( packageContext: Menu.this, NoteActivity.class);
             intent.putExtra( name: "Id",id);
             startActivity(intent);
            return true;
        case R.id.clients:
        //    Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
            Intent intentContact = new Intent( packageContext: Menu.this, ClientsActivity.class);
            intentContact.putExtra( name: "Id", id);
            startActivity(intentContact);
            return true;
        case R.id.email:
            Toast.makeText( context: this,  text: "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
            Intent intentEmail = new Intent( packageContext: Menu.this, EmailActivity.class);
            startActivity(intentEmail);
            return true;
        case R.id.reminder:
            Toast.makeText( context: this,  text: "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
            Intent intentReminder = new Intent(  packageContext: Menu.this, ReminderActivity.class);
            startActivity((intentReminder));
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

4.3 Felhasználói felület

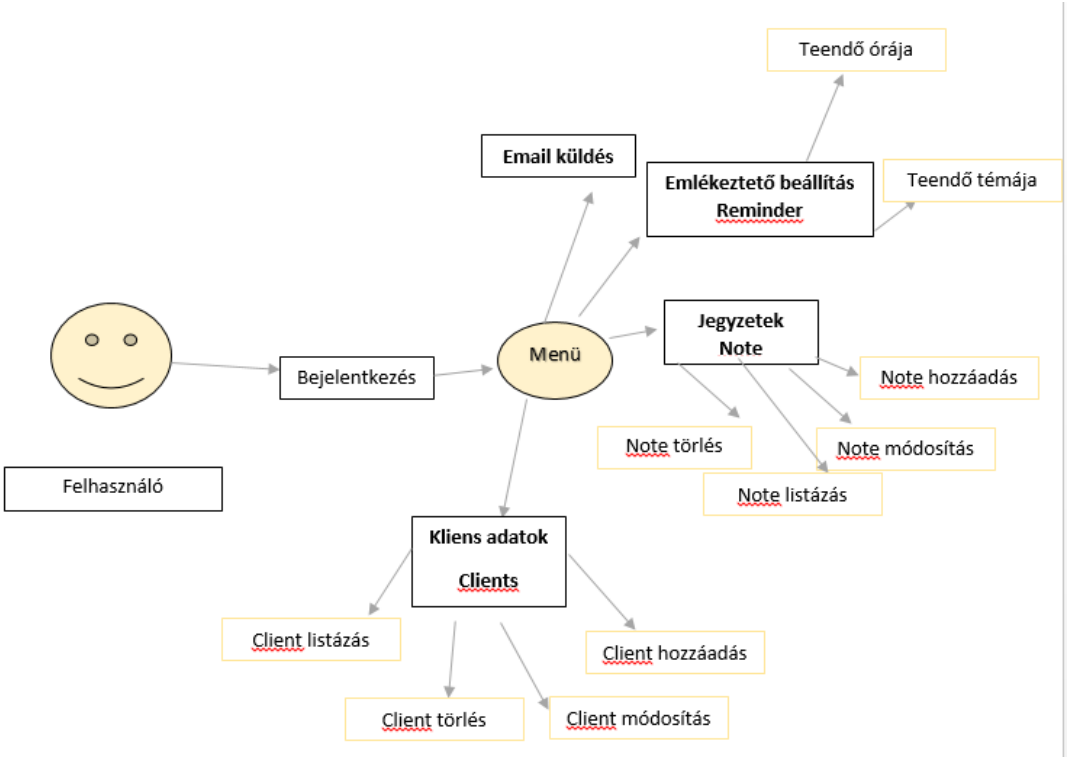
A felhasználói felület alatt értendő az alkalmazás külalakja, amit a felhasználó lát, azon tud navigálni, beírni adatokat és azokat megtekinteni.

Ezt is külön kódban valósítottuk meg, minden egyes képernyő-oldalhoz tartozik egy XML fájl, amelybe külön-külön minden egyes elemet egy-egy view határoz meg. Ebben megtalálhatóak gombok, kitöltendő üres mezők, szövegek és bizonyos form-ok amelyek kitöltésével adatok tud bevinni a felhasználó.

A görgethető RecyclerView-ban, amelyben listázandó elemek találhatóak, minden egyes lista elemre külön XML fájl van készítve, amelyben megvan egy elem felülete.

Itt vannak elkészítve egyéb külalaki fájlok is, mint például az email küldés felületnél bizonyos elemek, vagy a menü.

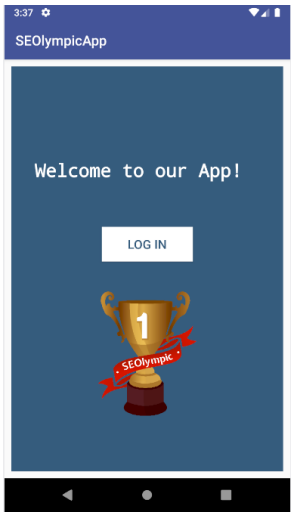
5. Az alkalmazás működési elve



Amint a fenti használati eset diagram is illusztrálja, van egy felhasználó, akinek van egy előre elkészített profilja, és azzal betud jelentkezni. (Regisztráció azért nincs, mivel ez egy belső rendszer, így csak azok tudják használni az alkalmazást, akinek van rá engedély adva és ezzel együtt felhasználói fiók is.) Bejelentkezés után a felhasználó eljut egy menühöz, ahol lehetősége van választani az alkalmazás 4 fontos funkciója közül, hogy mit szeretne csinálni: email-t küldeni, emlékeztetőt beállítani valamint a jegyzeteket és a klienseket szeretnék böngészni.

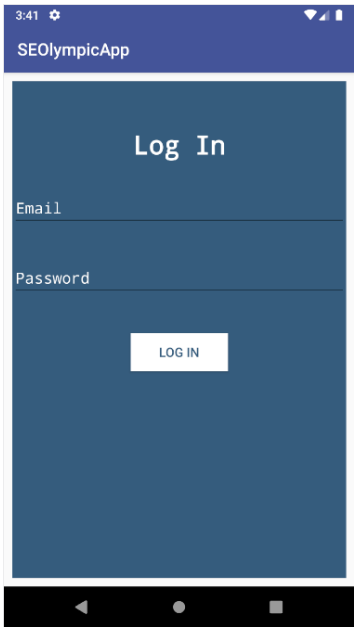
5.1 Főoldal

A főoldalon látható egy gomb, amellyel a Login részre tud eljutni, hogyha megnyomja.

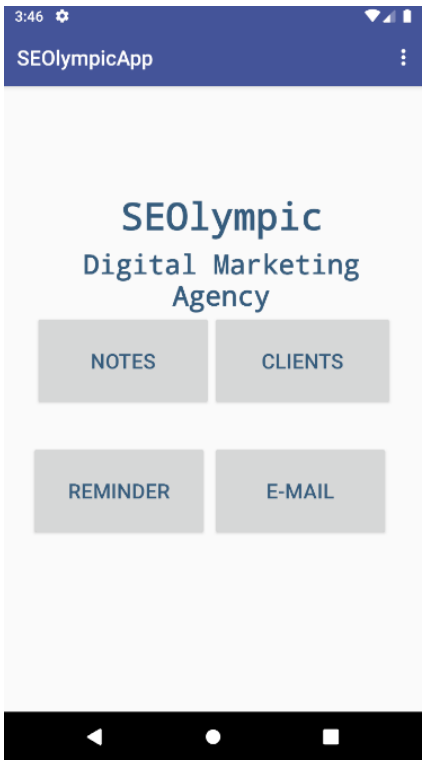


5.2 Login/Bejelentkezés

Amint átkerül ide, itt tudja beírni az érvényes email címet és a hozzá tartozó password-ot. Az alkalmazás a login gomb megnyomása után, csak akkor enged tovább, hogyha érvényesek a bejelentkezési adatok és nincsenek üres mezők, különben hibát fog kiadni.



5.3 Menü



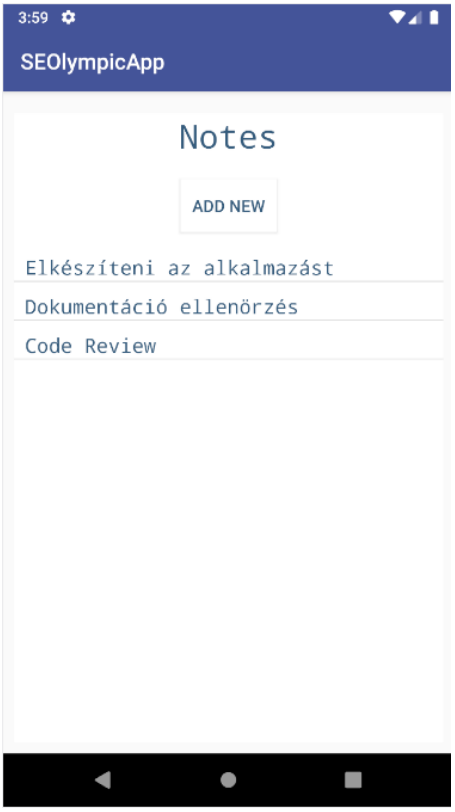
Két fajta menü lehetőség tárul az opciók kiválasztására a felhasználónak.

Az egyik a központban levő gombok egyikének a megnyomásával máris eltud jutni a kívánt funkció használatba vételéhez.

A másik viszont a fejléc végében elhelyezkedő három függőleges pontnak a megnyomásával egy legördülő lista fog megjelenni, amelyben pontosan azok az opciók is megjelennek majd, amelyek a gombokon is megtalálhatóak.

Innen 4 ágra lehet lebontani az alkalmazást.

5.4 Notes (Jegyzetek)

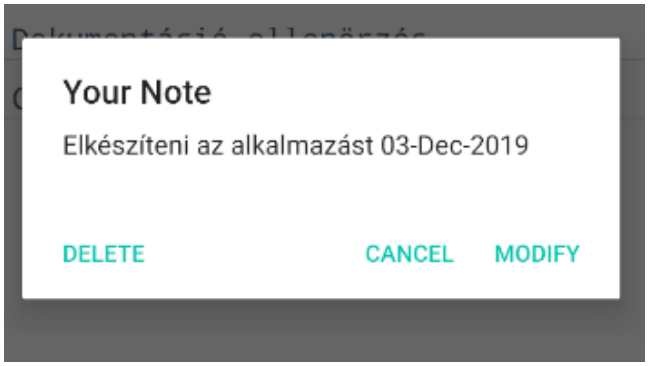


Ha a jegyzet funkciót választja, ahogy megnyitja rögtön megnyílik előtte az összes eddig hozzáadott jegyzet.

Ha az Add New gombra kattint, akkor új jegyzetet adhat hozzá egy kisebb felugró ablakban, abban a pillanatban ahogy hozzá adódott a jegyzet, frissül a lista és bekerül az új jegyzet is.

Ha bármelyik listabeli elemre kattint, akkor ott is felugrik egy ablakocska, ahol tudja módosítani az adott jegyzetet, tudja törölni, vagy vissza tud lépni.

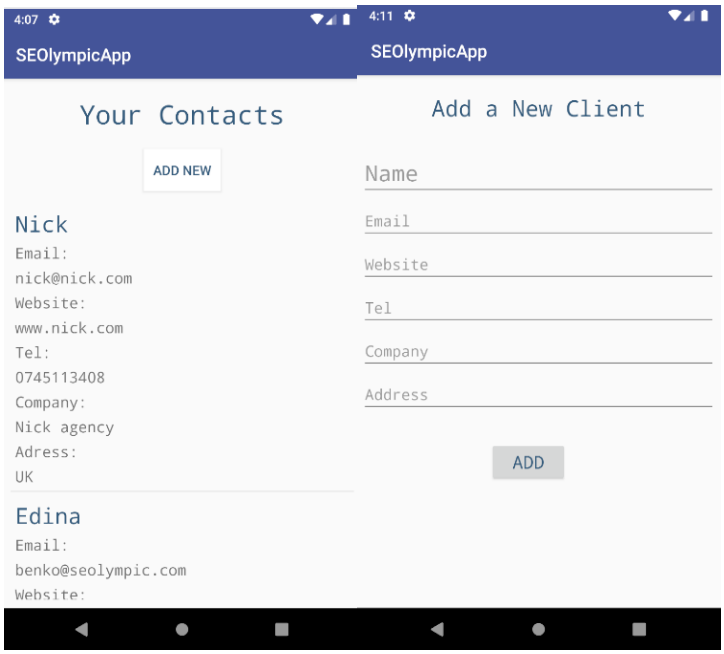
Törlés és módosítás után is egyből frissül a lista az aktuális jegyzetekkel.



5.5 Clients (Kliensek)

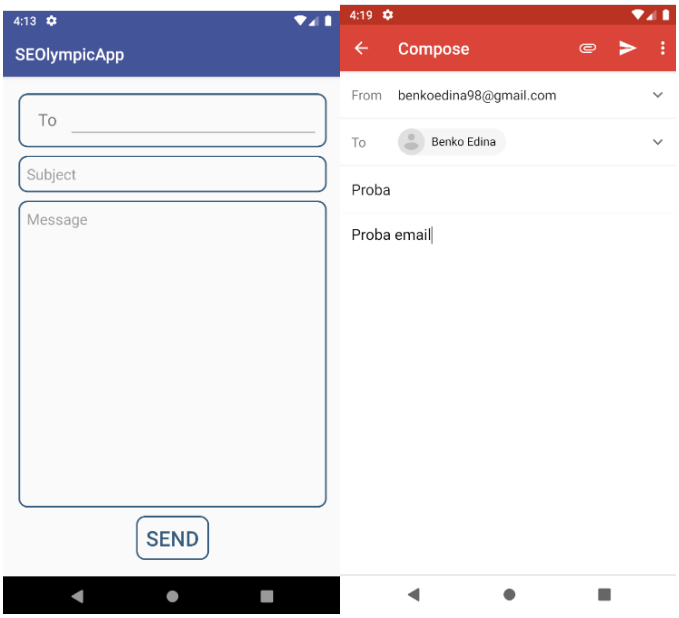
A kliensek menüpontot választva, hasonlóan, mint a jegyzeteknél egy listát fog látni a felhasználó, ahol megjelennek az eddig hozzá adott kliensek elérhetőségei. Azok közül bármelyikre kattintva kitudja azokat törölni, vagy vissza tud lépni.

Az Add New gombra kattintva pedig átkerülünk egy új ablakra, ahol egy Form fog megjelenni, amelynek a kitöltésével egy új klienset tudunk hozzáadni. Hozzáadás után ha visszalépünk, akkor a lista automatikusan frissül.



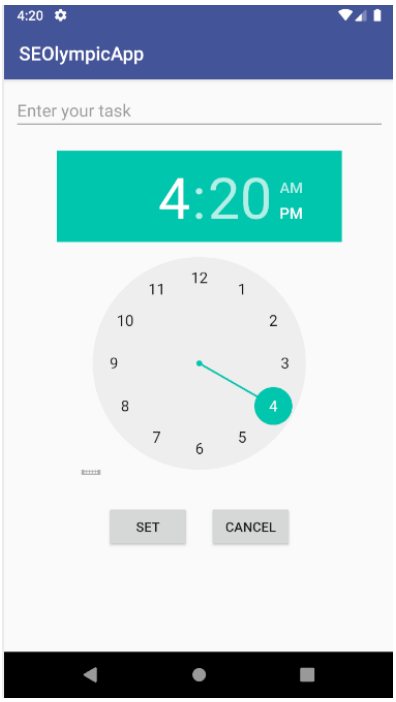
5.6 Email

Az Email küldést választva, egy olyan ablakra fog megérkezni, ahova betudja írni, hogy kinek akar email-t küldeni, mi annak a címe és a tartalma. Ha ez megvan és megnyomja a Send gombot, akkor ez átirányít egy olyan email felületre amilyen a mi email címünk, ha gmail akkor gmail-re, ha más akkor olyanra és automatikusan kitölti ugyanezeket a mezőket ott is, azokkal az adatokkal amelyeket előzőleg megadtunk és egy új send gyomb nyomással eltudjuk küldeni az üzenetünket.



5.7 Reminder (Emlékeztető)

Ha az emlékeztetőt választják, akkor betudnak állítani egy bizonyos fontos dolgot egy bizonyos órára, és akkor majd a telefon az alkalmazás lezárta után is értesítésben fog jelezni az elvégzendő dologról.



6. Továbbtervezési szempontok

Az alkalmazás eddigi szempontból is nagyon hasznosnak bizonyul, hiszen jó néhány alapvető funkciót tartalmaz egy helyen, amely nagyban elősegít minket.

Van néhány ötlet, amely még hasznos volna, ha meglenne valósítva az alkalmazásban:

- Ha lenne benne egy chat rendszer, amelyben az alkalmazást használók tudnának egymással kommunikálni.
- Ha lehetne dokumentumokat feltölteni rá, és azokat is megtudnánk nyitni és tudnánk törölni.
- Egy olyan rendszer lenne még, mint az email küldés, csak telefonálás szinten.
- Naptár alkalmazás, amelyhez nap szerint tudjunk hozzáadni teendőket.
- Regisztráció.

7. Összegzés

Végezetül nagyon örvendünk annak, hogy sikerült megvalósítani ezt az alkalmazást az eltervezett funkciókkal együtt.

Nagyon megvagyunk elégedve, de természetesen, hogyha még lesz rá lehetőségünk, akkor az előző pontban említett továbbszervezési szempontokat megszeretnénk valósítani, és hogyha úgy adódik, még azokhoz is csatoljunk egy-két funkciót.

Kellemes volt közösen egy csapatban dolgozni. Mindketten rengeteget tanultunk egymástól és eleve az android alkalmazás készítésének mivoltáról is. Megtanultunk megfelelően SQLite adatbázist használni, felelevenítettük a rég tanult adatbázis lekérdezések megírását, felelevenítettük az OOP rejtélyeit és a dokumentáció megírása alatt eltöltött idővel is csak újabb dolgokat tanultunk meg wordban.