Manage Passwords (MaPass)

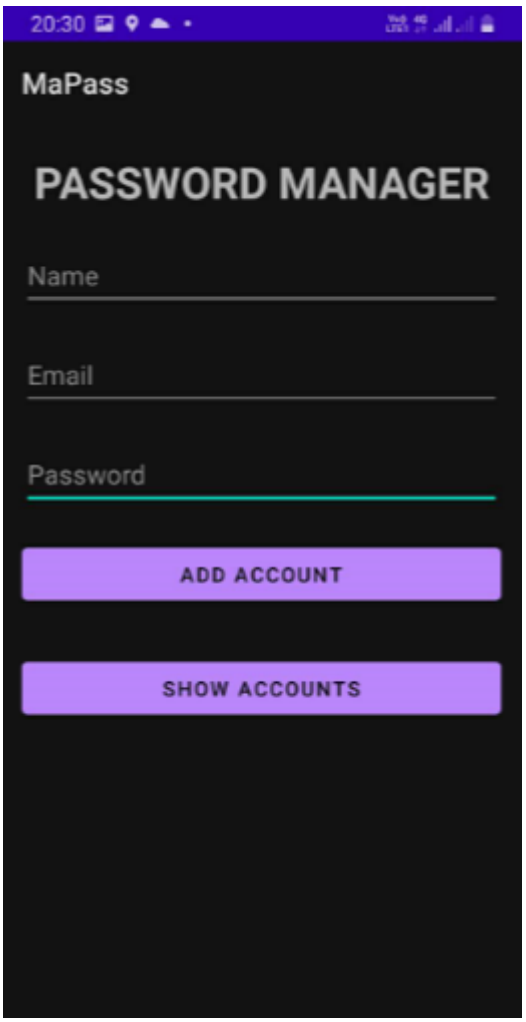
Μια εφαρμογή διαχείρισης κωδικών πρόσβασης που παρέχει ασφάλεια και ευκολία. Με αυτή την εφαρμογή, μπορείτε να αποθηκεύσετε και να οργανώσετε τους κωδικούς πρόσβασης σας για διάφορες υπηρεσίες. Παρέχει κρυπτογράφηση για την ασφάλεια των δεδομένων σας.

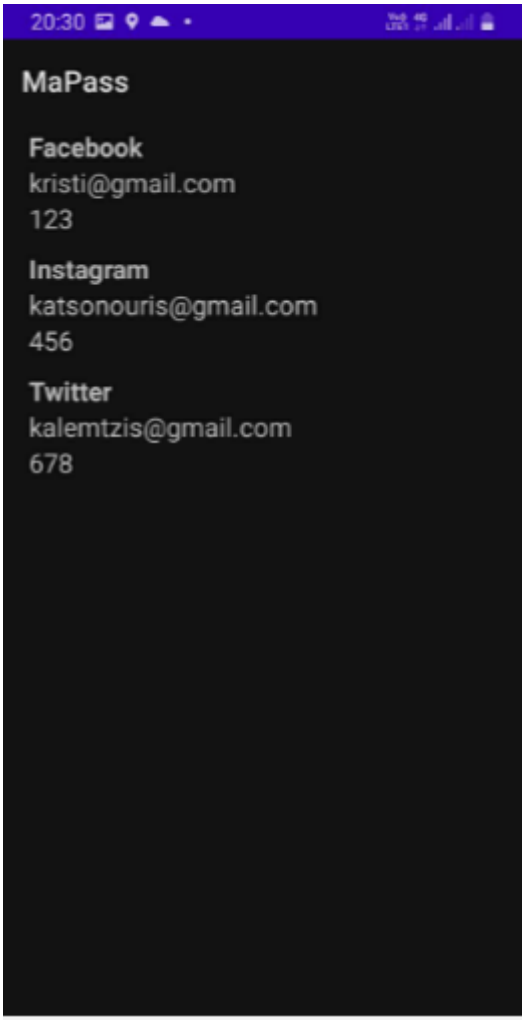
KRISTI CAMI 3882
ΚΑΤΣΟΝΟΥΡΗΣ ΑΝΤΡΕΑΣ 4054
ΚΑΛΕΜΤΖΗΣ ΒΑΣΙΛΗΣ 3876

Η εφαρμογή διαχείρισης κωδικών πρόσβασης αποτελεί ένα πολύτιμο εργαλείο για την απλούστευση και ασφάλεια της καθημερινής σας διαδικασίας σύνδεσης σε διάφορες πλατφόρμες και υπηρεσίες. Σε μια εποχή όπου ο αριθμός των προσωπικών λογαριασμών μας αυξάνεται συνεχώς, η απομνημόνευση των κωδικών πρόσβασης γίνεται όλο και πιο περίπλοκος και επιρρεπής σε λάθη.

Η εφαρμογή διαχείρισης κωδικών πρόσβασης αποτελεί μια κεντρική και ασφαλή αποθήκη για όλους τους κωδικούς πρόσβασης που χρησιμοποιούμε. Μπορείτε να αποθηκεύσετε τους κωδικούς σας για ιστοσελίδες, εφαρμογές, email, κοινωνικά δίκτυα και άλλες υπηρεσίες με ευκολία και ασφάλεια. Αυτό εξαλείφει την ανάγκη να θυμάστε πολλούς κωδικούς ή να τους γράφετε σε ανασφαλή μέρη.

Επιπλέον, η εφαρμογή προσφέρει προηγμένες δυνατότητες ασφάλειας. Οι κωδικοί πρόσβασης αποθηκεύονται με κρυπτογράφηση, εξασφαλίζοντας την απορρητότητα των πληροφοριών σας. Με τη βοήθεια αυτής της εφαρμογής, έχετε τη δυνατότητα να επικεντρωθείτε στην ασφάλεια και την ευκολία. Δεν χρειάζεται πλέον να ανησυχείτε για την απώλεια κωδικών, καθώς μπορείτε να αποκτήσετε πρόσβαση σε αυτούς με έναν ασφαλή τρόπο και με ελάχιστη προσπάθεια. Απολαύστε την αίσθηση της ευκολίας και της ασφάλειας καθώς διαχειρίζεστε τους κωδικούς πρόσβασής σας με αυτήν την εξαιρετική εφαρμογή.





Για την υλοποίηση της εφαρμογής δημιουργήθηκαν οι παρακάτω κλάσεις στο empty activity του android studio (Hello World):

- Accounts
- EncryptionUtils
- FetchDataActivity
- LanguageActivity
- MainActivity
- MyAdapter
- MyDatabase
- MyInterface
- UpdataActivity

Οι κλάσης αυτές βρίσκονται στο path (app/java/com/example/mapass)

Ανάλυση κλάσεων

Accounts

Η κλάση Accounts αντιπροσωπεύει έναν λογαριασμό χρήστη

Με το @Entity αναφερόμαστε στο ότι η κλάση αυτή θα αντιστοιχηθεί εσωτερικά με έναν πίνακα σε μια βάση δεδομένων με το όνομα **tableName = "accounts"**.

```
KristiCami
@Entity(tableName = "accounts")
```
/

Ο πινάκας αυτός περιέχει τέσσερα χαρακτηριστικά (**id**, sName, email, password), με το @PrimaryKey του να δημιουργείται αυτόματα μέσω της παραμέτρου autoGenerate.

```
4 usages
@PrimaryKey(autoGenerate = true)
private int id;
```

Στην κλάση επίσης θα βρούμε κατασκευαστή και της δίφορες μεθόδους **setters, getters** που είναι απαραίτητες για σύνδεση του κώδικα με άλλα τμήματα.

FetchDataActivity

Ο κώδικας παρουσιάζει μια απλή λειτουργικότητα για την εμφάνιση και προβολή δεδομένων από μια βάση δεδομένων.

Εισαγωγή απαραίτητων βιβλιοθηκών και κλάσεων για την ανάπτυξη της εφαρμογής.

Ορισμός των απαιτούμενων μεταβλητών για την εφαρμογή.

Υλοποίηση της μεθόδου **onCreate()**, η οποία εκτελείται κατά την έναρξη της δραστηριότητας.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityFetchDataActivityBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    myDb = Room.databaseBuilder( context: this,MyDatabase.class, name: "accounts").allowMainThreadQueries().build();
    myInt=myDb.myInterface();

    accountsList=myInt.getAllAccounts();
    Toast.makeText( context: this, text: "You have "+accountsList.size() + " accounts",Toast.LENGTH_SHORT ).show();
    myAdapter=new MyAdapter( context: this,accountsList);
    binding.dataRecycler.setLayoutManager(new LinearLayoutManager( context: this));
    binding.dataRecycler.setAdapter(myAdapter);

}
```

Επισύναψη του layout αρχείου **(ActivityFetchDataActivityBinding)** στη δραστηριότητα.

Δημιουργία μιας βάσης δεδομένων **(myDb)** με τη χρήση του **Room**, η οποία επιτρέπει την αποθήκευση και ανάκτηση δεδομένων.

Ανάκτηση ενός διασυνδεδεμένου αντικειμένου **myInt** της βάσης δεδομένων.

Λήψη μιας λίστας (**accountsList**) που περιέχει αντικείμενα τύπου **Accounts** από τη βάση δεδομένων με τη χρήση του **myInt**.

```
4 usages
ActivityFetchDataActivityBinding binding;
2 usages
MyDatabase myDb;
2 usages
MyInterface myInt;
3 usages
List<Accounts> accountsList;
2 usages
MyAdapter myAdapter;
```

Εμφάνιση μιας ειδοποίησης (**Toas**t) που ενημερώνει τον χρήστη για τον αριθμό των λογαριασμών που υπάρχουν στη λίστα **accountsList**.

```
Toast.makeText( context: this, text: "You have "+accountsList.size() + " accounts",Toast.LENGTH_SHORT ).show();
```

Δημιουργία ενός αντικειμένου **myAdapter** τύπου **MyAdapter** για την προβολή των δεδομένων από τη λίστα **accountsList**.

Ορισμός της διάταξης (**LinearLayoutManager**) του αναδιπλούμενου (**RecyclerView**) στοιχείου **dataRecycler** που χρησιμοποιείται για την εμφάνιση των δεδομένων στην οθόνη.

Ορισμός του προσαρμογέα (**myAdapter**) για το **dataRecycler**, προκειμένου να εμφανίζονται τα δεδομένα της λίστας στην οθόνη.

MainActivity

Η κλάση αυτη είναι υπεύθυνη για την αρχική εμφάνιση της οθόνης και τη διαχείριση των συμβάντων.

Ορισμένες σημαντικές λειτουργίες και μεταβλητές που χρησιμοποιούνται στην κλάση **MainActivity** είναι οι εξής:

ActivityMainBinding binding: Μια μεταβλητή που αναφέρεται στον αρχείο δέσμευσης (binding) που δημιουργείται αυτόματα από το Android Studio, το οποίο συνδέει τα στοιχεία της διεπαφής χρήστη στο αρχείο XML με την κλάση Java.

MyDatabase myDb: Ένα αντικείμενο της κλάσης **MyDatabase,** που αντιπροσωπεύει τη βάση δεδομένων SQLite. Η βάση δεδομένων δημιουργείται με τη βοήθεια της βιβλιοθήκης Room και χρησιμοποιείται για την αποθήκευση των λογαριασμών.

MyInterface myInt: Ένα αντικείμενο της διεπαφής **MyInterface**, που παρέχει μεθόδους για την αλληλεπίδραση με τη βάση δεδομένων.

```
7 usages
ActivityMainBinding binding;
2 usages
MyDatabase myDb;
2 usages
MyInterface myInt;
```

*Στη μέθοδο **onCreate**, που καλείται κατά την εκκίνηση της δραστηριότητας, εκτελούνται οι εξής ενέργειες:*

Αρχικοποίηση της μεταβλητής **binding** με τη χρήση της μεθόδου **ActivityMainBinding.inflate(getLayoutInflater())**. Αυτό δημιουργεί ένα αντικείμενο δέσμευσης (binding object) που συνδέει τα στοιχεία της διεπαφής χρήστη με την κλάση Java.

Ορισμός της διεπαφής της δραστηριότητας με τη χρήση της μεθόδου **setContentView(binding.getRoot())**. Αυτό καθορίζει ποιο αρχείο διάταξης (layout file) θα χρησιμοποιηθεί για την εμφάνιση της οθόνης της δραστηριότητας.

Δημιουργία της βάσης δεδομένων SQLite με τη βοήθεια της μεθόδου **databaseBuilder**. Η βάση δεδομένων ονομάζεται "accounts" και η παράμετρος **allowMainThreadQueries()** επιτρέπει τις επερωτήσεις στη βάση δεδομένων να εκτελούνται στον κύριο νήμα εκτέλεσης (UI thread). Η παράμετρος **fallbackToDestructiveMigration()** χρησιμοποιείται για την αυτόματη διαγραφή και επαναδημιουργία της βάσης δεδομένων σε περίπτωση αλλαγών στην έκδοση της βάσης.

Ανάθεση του αντικειμένου διεπαφής της βάσης δεδομένων στη μεταβλητή **myInt** με την εκχώρηση **myDb.myInterface()**.

```
super.onCreate(savedInstanceState);
binding = ActivityMainBinding.inflate(getLayoutInflater());
setContentView(binding.getRoot());

myDb= Room.databaseBuilder( context: this,MyDatabase.class, name: "accounts").allowMainThreadQueries().fallbackToDestructiveMigration().build();
myInt=myDb.myInterface();

KristiCami
```

Ορισμός της ενέργειας κλικ για το κουμπί **addAcc**. Όταν γίνει κλικ στο κουμπί, ανακτώνται οι τιμές των πεδίων **socialMediaName, mobileNumber** και **password**. Στη συνέχεια, δημιουργείται ένα αντικείμενο **Accounts** με τις τιμές αυτές και εισάγεται στη βάση δεδομένων μέσω της μεθόδου **insert** της διεπαφής **myInt**. Τέλος, εμφανίζεται ένα μήνυμα επιτυχίας με τη χρήση της μεθόδου **makeText** και **show** της κλάσης **Toast**.

```
KristiCami
binding.addAcc.setOnClickListener(new View.OnClickListener() {
    KristiCami
    @Override
    public void onClick(View view) {
        String name = binding.socialMediaName.getText().toString();
        String email = binding.mobileNumber.getText().toString();
        String password = binding.password.getText().toString();

        Accounts accounts = new Accounts( id: 0,name,email,password);
        myInt.insert(accounts);
        Toast.makeText(getApplicationContext(),  text: "Account added successfully", Toast.LENGTH_SHORT).show();
    }
});
```

Ορισμός της ενέργειας κλικ για το κουμπί showAccBtn. Όταν γίνει κλικ στο κουμπί, ξεκινά μια νέα δραστηριότητα με τη χρήση της μεθόδου **startActivity**. Η νέα δραστηριότητα είναι η **FetchDataActivity**.

```
KristiCami
binding.showAccBtn.setOnClickListener(new View.OnClickListener() {
    KristiCami
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: MainActivity.this, FetchDataActivity.class));
    }
});
```

Η κλάση **MainActivity** αντιπροσωπεύει την αρχική οθόνη της εφαρμογής και περιλαμβάνει τη λογική για την αποθήκευση νέων λογαριασμών και την προβολή των υπάρχοντων λογαριασμών.

MyAdapter

Η κλάση **MyAdapter** αντιπροσωπεύει έναν προσαρμογέα (adapter) για την εμφάνιση δεδομένων σε ένα RecyclerView. Ο προσαρμογέας αυτός συνδέει τα δεδομένα της λίστας λογαριασμών (accountsList) με τα στοιχεία γραφικής διεπαφής που εμφανίζονται στα αντικείμενα RecyclerView.

Οι σημαντικές λειτουργίες και μεταβλητές που χρησιμοποιούνται στην κλάση MyAdapter είναι οι εξής:

Context context: Το περιβάλλον του προσαρμογέα, το οποίο χρησιμοποιείται για τη δημιουργία νέων δραστηριοτήτων και άλλων ενεργειών στην εφαρμογή.

List<Accounts> accountsList: Η λίστα που περιέχει τα αντικείμενα **Accounts** που θα εμφανιστούν στο RecyclerView.

```
3 usages
private Context context;
3 usages
private List<Accounts> accountsList;
```

Ο κατασκευαστής **MyAdapter(Context context, List<Accounts> accountsList)**: Δέχεται το περιβάλλον του προσαρμογέα και τη λίστα των λογαριασμών και τις αποθηκεύει στις αντίστοιχες μεταβλητές.

```
1 usage  KristiCami
public MyAdapter(Context context, List<Accounts> accountsList) {
    this.accountsList = accountsList;
    this.context = context;
}
```

Η μέθοδος **onCreateViewHolder()**: Δημιουργεί ένα αντικείμενο **MyViewHolder** που αναπαριστά την προβολή για κάθε στοιχείο της λίστας. Χρησιμοποιεί τον φουσκωτό (inflater) για να φορτώσει την διάταξη από το αρχείο XML single_row και το επιστρέφει.

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

Η μέθοδος **onBindViewHolder()**: Δεσμεύει τα δεδομένα του λογαριασμού στις αντίστοιχες προβολές (TextViews) του MyViewHolder. Επίσης, ορίζει την ενέργεια κλικ για κάθε στοιχείο της λίστας, ώστε όταν γίνει κλικ σε ένα αντικείμενο, να ξεκινά μια νέα δραστηριότητα **UpdateActivity** και να περνά τα απαραίτητα δεδομένα μέσω του Intent.

```
KristiCami
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {

    Accounts accounts = accountsList.get(position);
    holder.name.setText(accounts.getsName());
    holder.email.setText(accounts.getEmail());
    holder.password.setText(accounts.getPassword());

    KristiCami
    holder.itemView.setOnClickListener(new View.OnClickListener() {
        KristiCami
        @Override
        public void onClick(View v) {
            Intent intent=new Intent(context,UpdateActivity.class);
            intent.putExtra( name: "name",accounts.getsName());
            intent.putExtra( name: "id",accounts.getId());

            intent.putExtra( name: "email",accounts.getEmail());
            intent.putExtra( name: "name",accounts.getPassword());
            context.startActivity(intent);
        }
    });
}
```

Η μέθοδος **getItemCount()**: Επιστρέφει τον αριθμό των στοιχείων στη λίστα.

```
KristiCami
@Override
public int getItemCount() { return accountsList.size(); }
```

Η κλάση **MyViewHolder**: Είναι ένας εσωτερικός ταξινομημένος κλάσης που κρατά τις αναφορές προβολής (TextViews) για κάθε στοιχείο της λίστας. Οι αναφορές αυτές αρχικοποιούνται στον κατασκευαστή της κλάσης.

```
4 usages  KristiCami
public class MyViewHolder extends RecyclerView.ViewHolder{
    2 usages
    TextView name,email,password;
    1 usage  KristiCami
    public MyViewHolder(@NonNull View itemView) {
        super(itemView);
        name = itemView.findViewById(R.id.sName_one);
        email=itemView.findViewById(R.id.email_one);
        password = itemView.findViewById(R.id.pass_one);

    }
}
```

MyDatabase

Η κλάση **MyDatabase** αντιπροσωπεύει μια βάση δεδομένων στο πλαίσιο της αρχιτεκτονικής Room στο Android Studio. Η κλάση αυτή επεκτείνει την **RoomDatabase** και ορίζει τη δομή της βάσης δεδομένων καθώς και τις σχέσεις μεταξύ των πινάκων.

*Οι σημαντικές λειτουργίες και μεταβλητές που χρησιμοποιούνται στην κλάση **MyDatabase** είναι οι εξής*:

Η ετικέτα @Database: Αυτή η ετικέτα χρησιμοποιείται για να δηλώσει ότι η κλάση αναπαριστά μια βάση δεδομένων Room. Παίρνει ορισμένες παραμέτρους, όπως ο πίνακας **entities**, που περιγράφει τις κλάσεις οντοτήτων που σχετίζονται με τη βάση δεδομένων, και η **version**, που καθορίζει τον αριθμό έκδοσης της βάσης δεδομένων.

Η μέθοδος **myInterface()**: Αυτή η αφηρημένη μέθοδος παρέχει ένα αντικείμενο MyInterface που αναπαριστά τη διεπαφή πρόσβασης στη βάση δεδομένων.

```
import androidx.room.Database;
import androidx.room.RoomDatabase;

import com.example.mapass.MyInterface;
7 usages  1 inheritor  KristiCami
@Database(entities = {Accounts.class},version = 4)
public abstract class MyDatabase extends RoomDatabase {
    3 usages  1 implementation  KristiCami
    public abstract MyInterface myInterface();
}
```

Η κλάση **MyDatabase** είναι αφαιρετική (abstract), πράγμα που σημαίνει ότι δεν μπορεί να δημιουργηθεί απευθείας ένα αντικείμενο από αυτήν. Αντ' αυτού, χρησιμοποιείται για τη δημιουργία μιας υποκλάσης που υλοποιεί τη λειτουργικότητα της βάσης δεδομένων Room και παρέχει την απαραίτητη μέθοδο **myInterface()** για την πρόσβαση στη βάση δεδομένων.

MyInterface

Η διεπαφή **MyInterface** αντιπροσωπεύει ένα Data Access Object στο πλαίσιο της βιβλιοθήκης Room. Η διεπαφή αυτή περιέχει μεθόδους που χρησιμοποιούνται για την πρόσβαση και τη διαχείριση των δεδομένων στη βάση δεδομένων.

*Οι σημαντικές μεθόδοι που ορίζονται στην διεπαφή **MyInterface** είναι οι εξής:*

void insert(Accounts entityClass): Αυτή η μέθοδος χρησιμοποιείται για την εισαγωγή (insert) ενός αντικειμένου Accounts στη βάση δεδομένων.

void update(Accounts entityClass): Αυτή η μέθοδος χρησιμοποιείται για την ενημέρωση (update) ενός αντικειμένου Accounts στη βάση δεδομένων.

void deleteAcc(int id): Αυτή η μέθοδος χρησιμοποιείται για τη διαγραφή (delete) ενός αντικειμένου Accounts από τη βάση δεδομένων, με βάση το πεδίο id.

List<Accounts> getAllAccounts(): Αυτή η μέθοδος χρησιμοποιείται για να ανακτήσει (query) όλα τα αντικείμενα Accounts από τη βάση δεδομένων.

```
9 usages  1 implementation  KristiCami
@Dao
public interface MyInterface {
    1 usage  1 implementation  KristiCami
    @Insert
    void insert(Accounts entityClass);
    1 implementation  KristiCami
    @Update
    void update(Accounts entityClass);
    1 usage  1 implementation  KristiCami
    @Query("DELETE from accounts where id=:id")
    void deleteAcc(int id);
    1 usage  1 implementation  KristiCami
    @Query("SELECT * From accounts")
    List<Accounts> getAllAccounts();
}
```

Η διεπαφή **MyInterface** αντιστοιχεί στη λειτουργικότητα των ερωτημάτων (queries) που εκτελούνται στη βάση δεδομένων Room, όπως η εισαγωγή, η ενημέρωση, η διαγραφή και η ανάκτηση δεδομένων.

UpdateActivity

Η κλάση **UpdateActivity** χρησιμοποιείται για την ενημέρωση και διαγραφή ενός λογαριασμού στη βάση δεδομένων**.**

Οι σημαντικές μεταβλητές και λειτουργίες που χρησιμοποιούνται στην κλάση UpdateActivity είναι οι εξής:

Η μεταβλητή **binding**: Αντικείμενο της κλάσης **ActivityUpdateBinding** που παρέχει πρόσβαση στα στοιχεία διεπαφής χρήστη που έχουν καθοριστεί στο αρχείο XML της δραστηριότητας.

Οι μεταβλητές **myDb** και **myInt**: Αντικείμενα της κλάσης **MyDatabase** και **MyInterface** αντίστοιχα, που χρησιμοποιούνται για την πρόσβαση στη βάση δεδομένων.

Οι μεταβλητές **name, email, password** και **id**: Λαμβάνουν τις τιμές των παραμέτρων που περνιούνται από την προηγούμενη δραστηριότητα μέσω του αντικειμένου **Intent**.

```
myDb= Room.databaseBuilder( context: this,MyDatabase.class, name: "accounts").allowMainThreadQueries().build();
myInt=myDb.myInterface();

String name = getIntent().getStringExtra( name: "name");
String email = getIntent().getStringExtra( name: "email");
String password = getIntent().getStringExtra( name: "password");
int id = getIntent().getIntExtra( name: "id", defaultValue: -1);

binding.socialMediaName.setText(name);
binding.mobileNumber.setText(email);
binding.password.setText(password);
```

Η μέθοδος **onCreate()**: Αυτή η μέθοδος καλείται κατά την εκκίνηση της δραστηριότητας και εκτελεί τον κώδικα για την αρχικοποίηση των στοιχείων διεπαφής και τη δημιουργία των ακροατών γεγονότων για τα κουμπιά.

Ο ακροατής γεγονότος για το κουμπί **updateAcc**: Αυτός ο ακροατής αντιδρά στο κλικ του κουμπιού "Ενημέρωση" και εκτελεί τη λειτουργία ενημέρωσης του λογαριασμού στη βάση δεδομένων. Αντικείμενο **Accounts** δημιουργείται με τις νέες τιμές και η μέθοδος **update()** του αντικειμένου **myInt** καλείται για να εκτελέσει την ενημέρωση. Επίσης, εμφανίζεται ένα αναδυόμενο μήνυμα με την ένδειξη ότι ο λογαριασμός ενημερώθηκε επιτυχώς και η δραστηριότητα τερματίζει.

```
KristiCami
binding.updateAcc.setOnClickListener(new View.OnClickListener() {
    KristiCami
    @Override
    public void onClick(View v) {
        String name = binding.socialMediaName.getText().toString();
        String email = binding.mobileNumber.getText().toString();
        String pass = binding.password.getText().toString();

        Accounts accounts = new Accounts(id,name,email,pass);
        myInt.update(accounts);
        Toast.makeText(getApplicationContext(), text: "Account updated successfully", Toast.LENGTH_SHORT).show();
        finish();
    }
});
```

Ο ακροατής γεγονότος για το κουμπί **delAcc**: Αυτός ο ακροατής αντιδρά στο κλικ του κουμπιού "Διαγραφή" και εκτελεί τη λειτουργία διαγραφής του λογαριασμού από τη βάση δεδομένων. Η μέθοδος **deleteAcc()** του αντικειμένου **myInt** καλείται για να εκτελέσει τη διαγραφή. Εμφανίζεται ένα αναδυόμενο μήνυμα με την ένδειξη ότι ο λογαριασμός διαγράφηκε επιτυχώς και η δραστηριότητα τερματίζει.

```
KristiCami
binding.delAcc.setOnClickListener(new View.OnClickListener() {
    KristiCami
    @Override
    public void onClick(View v) {
        myInt.deleteAcc(id);
        Toast.makeText(getApplicationContext(), text: "Account deleted successfully", Toast.LENGTH_SHORT).show();
        finish();
    }
});
```

EncryptionUtils

Η κλάση **EncryptionUtils** παρέχει μεθόδους για την κρυπτογράφηση κωδικών χρησιμοποιώντας τον αλγόριθμο PBEWithMD5AndDES.

*Οι σημαντικές μεταβλητές και μέθοδοι που χρησιμοποιούνται στην κλάση **EncryptionUtils** είναι οι εξής:*

Οι σταθερές μεταβλητές **ENCRYPTION_ALGORITHM, ENCRYPTION_PASSWORD** και **SALT**: Ορίζουν τον αλγόριθμο κρυπτογράφησης, τον κωδικό πρόσβασης για την κρυπτογράφηση και το αλάτι που χρησιμοποιείται για την αύξηση της ασφάλειας του αλγορίθμου.

```
4 usages
private static final String ENCRYPTION_ALGORITHM = "PBEWithMD5AndDES";
2 usages
private static final String ENCRYPTION_PASSWORD = "j^D0bqKzV@y#4&8";
4 usages
private static final byte[] SALT = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

Η μέθοδος **encrypt(String password)**: Χρησιμοποιεί τον αλγόριθμο PBEWithMD5AndDES για να κρυπτογραφήσει έναν κωδικό πρόσβασης. Δημιουργεί έναν κρυπτογραφητή **Cipher** και τον παραμετροποιεί για λειτουργία κρυπτογράφησης με τον μυστικό κλειδί και το αλάτι. Ο κωδικός πρόσβασης μετατρέπεται σε byte array και κρυπτογραφείται. Τέλος, επιστρέφει το κρυπτογραφημένο αποτέλεσμα σε μορφή Base64.

```
2 usages  Andreascy01
@RequiresApi(api = Build.VERSION_CODES.O)
public static String encrypt(String password) {
    try {
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(ENCRYPTION_ALGORITHM);
        KeySpec keySpec = new PBEKeySpec(ENCRYPTION_PASSWORD.toCharArray(), SALT,  iterationCount: 65536,  keyLength: 256);
        SecretKey secretKey = keyFactory.generateSecret(keySpec);

        Cipher cipher = Cipher.getInstance(ENCRYPTION_ALGORITHM);
        AlgorithmParameterSpec paramSpec = new PBEParameterSpec(SALT,  iterationCount: 100);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, paramSpec);

        byte[] encryptedBytes = cipher.doFinal(password.getBytes( charsetName: "UTF-8"));
        return Base64.getEncoder().encodeToString(encryptedBytes);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Η μέθοδος **decrypt(String encryptedPassword)**: Χρησιμοποιεί τον αλγόριθμο PBEWithMD5AndDES για να αποκρυπτογραφήσει έναν κρυπτογραφημένο κωδικό πρόσβασης. Δημιουργεί έναν κρυπτογραφητή **Cipher** και τον παραμετροποιεί για λειτουργία αποκρυπτογράφησης με τον μυστικό κλειδί και το αλάτι. Ο κρυπτογραφημένος κωδικός πρόσβασης αποκωδικοποιείται από Base64 και αποκρυπτογραφείται. Τέλος, επιστρέφει τον αποκρυπτογραφημένο κωδικό πρόσβασης σε μορφή String.

```
4 usages  Andreascy01
@RequiresApi(api = Build.VERSION_CODES.O)
public static String decrypt(String encryptedPassword) {
    try {
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(ENCRYPTION_ALGORITHM);
        KeySpec keySpec = new PBEKeySpec(ENCRYPTION_PASSWORD.toCharArray(), SALT,  iterationCount: 65536,  keyLength: 256);
        SecretKey secretKey = keyFactory.generateSecret(keySpec);

        Cipher cipher = Cipher.getInstance(ENCRYPTION_ALGORITHM);
        AlgorithmParameterSpec paramSpec = new PBEParameterSpec(SALT,  iterationCount: 100);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, paramSpec);

        byte[] decodedBytes = Base64.getDecoder().decode(encryptedPassword);
        byte[] decryptedBytes = cipher.doFinal(decodedBytes);
        return new String(decryptedBytes,  charsetName: "UTF-8");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Η κλάση **EncryptionUtils** είναι χρήσιμη για την ασφαλή αποθήκευση και μεταφορά κρίσιμων πληροφοριών, όπως κωδικοί πρόσβασης, μέσω απλού κειμένου σε μια εφαρμογή Android.

LanguageManager

Η κλάση **LanguageManager** παρέχει μια μέθοδο για την ενημέρωση των ρυθμίσεων γλώσσας σε μια εφαρμογή Android.

*Οι σημαντικές μεταβλητές και μέθοδοι που χρησιμοποιούνται στην κλάση **LanguageManager** είναι οι εξής:*

Η μεταβλητή **ct**: Αναπαριστά το περιβάλλον Context στο οποίο εφαρμόζονται οι αλλαγές γλώσσας.

Ο κατασκευαστής **LanguageManager(Context context)**: Δέχεται ένα αντικείμενο **Context** κατά τη δημιουργία του **LanguageManager** για την αρχικοποίηση της μεταβλητής **ct**.

```
// Retrieving the stored language preference
4 usages
private Context ct;

Andreascy01
public LanguageManager(Context context) { this.ct = context; }
```

Η μέθοδος **updateResource(String code)**: Λαμβάνει έναν κωδικό γλώσσας ως είσοδο και ενημερώνει τις ρυθμίσεις γλώσσας της εφαρμογής. Αρχικά, δημιουργεί ένα νέο αντικείμενο **Locale** με βάση τον κωδικό γλώσσας. Στη συνέχεια, αλλάζει την προεπιλεγμένη τοπική γλώσσα στο αντικείμενο **Configuration** και ενημερώνει το περιβάλλον **Context** με τη νέα τοπική γλώσσα. Τέλος, γίνεται ενημέρωση των πόρων της εφαρμογής με τις νέες ρυθμίσεις γλώσσας.

```
Andreascy01
public void updateResource(String code)
{
    Locale locale = new Locale(code);
    Locale.setDefault(locale);
    Resources resources = ct.getResources();
    Configuration configuration = resources.getConfiguration();
    configuration.setLocale(locale);

    ct = ct.createConfigurationContext(configuration);
    resources.updateConfiguration(configuration, resources.getDisplayMetrics());
}
```

Με τη χρήση αυτής της κλάσης, οι πόροι της εφαρμογής, όπως κείμενα και συμβολισμοί, μπορούν να ενημερώνονται δυναμικά για να αντιστοιχούν στην επιλεγμένη γλώσσα.

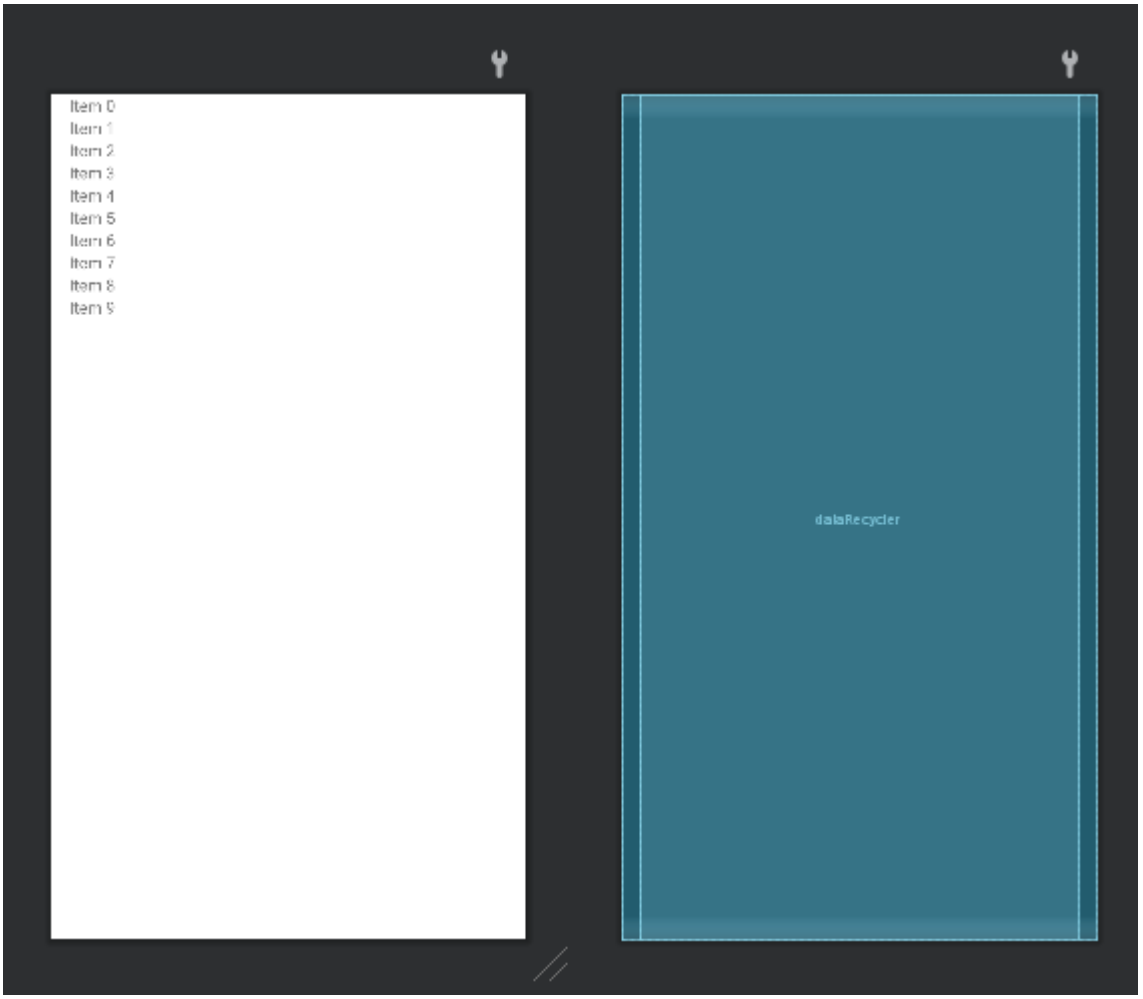
Χρειάστηκε έπεισες να δημιουργηθούν και τέσσερα xml αρχεία:

- activity_fetch_data_activity.xml
- activity_main.xml
- activity_update.xml
- single_row.xml
- dialog_image.xml

Τα αρχεία αυτά βρίσκονται στο path (app/res/layout)

Ανάλυση xml αρχείων

activity_fetch_data_activity.xml



Οι σημαντικές λειτουργίες και ιδιότητες που χρησιμοποιούνται στον κώδικα XML είναι οι εξής:

Το αρχικό tag **LinearLayout**: Ορίζει ένα γραμμικό layout που τοποθετεί τα στοιχεία του παιδιού του κατακόρυφα.

Τα επιπλέον attributes **xmlns:android, xmlns:app** και **xmlns:tools**: Ορίζουν τους ορθογώνιους χώρους ονομάτων για τις βιβλιοθήκες Android, όπως τα στοιχεία UI (android), χαρακτηριστικά προσαρμογής (app) και εργαλεία ανάπτυξης (tools).

Το attribute **tools:context**: Ορίζει την κλάση που θα χρησιμοποιηθεί ως περιβάλλον για τον στατικό ελεγκτή προτύπου (template controller), σε αυτήν την περίπτωση την **MainActivity.**

```
tools:context=".MainActivity"
```

Το attribute **android:orientation**: Ορίζει τον προσανατολισμό του γραμμικού layout, σε αυτήν την περίπτωση κατακόρυφος.

Τα attributes **android:paddingLeft** και **android:paddingRight**: Ορίζουν την εσοχή αριστερά και δεξιά του γραμμικού layout, για προσθήκη αρθρώσεων.

```
android:paddingLeft="16dp"
android:paddingRight="16dp">
```

Το στοιχείο **<androidx.recyclerview.widget.RecyclerView>:** Ορίζει ένα RecyclerView, που είναι ένα περιοχής κύλισης που χρησιμοποιείται για την εμφάνιση μιας λίστας στοιχείων. Ορίζεται με το αναγνωριστικό αναγνώρισης **@+id/dataRecycler** και έχει ρυθμιστεί να καταλαμβάνει τον ίδιο χώρο με το γονικό γραμμικό layout.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/dataRecycler"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

activity_main.xml

Οι σημαντικές λειτουργίες και ιδιότητες που χρησιμοποιούνται στον κώδικα XML είναι οι εξής:

Το αρχικό tag **ScrollView**: Ορίζει ένα ScrollView που επιτρέπει την κύλιση του περιεχομένου εάν υπερβαίνει το περιορισμένο χώρο που είναι ορατό.

Τα επιπλέον attributes **xmlns:android, xmlns:app** και **xmlns:tools**: Ορίζουν τους ορθογώνιους χώρους ονομάτων για τις βιβλιοθήκες Android, όπως τα στοιχεία UI (android), χαρακτηριστικά προσαρμογής (app) και εργαλεία ανάπτυξης (tools).

Το στοιχείο **<LinearLayout>**: Ορίζει ένα γραμμικό layout που τοποθετεί τα στοιχεία του παιδιού του κατακόρυφα.

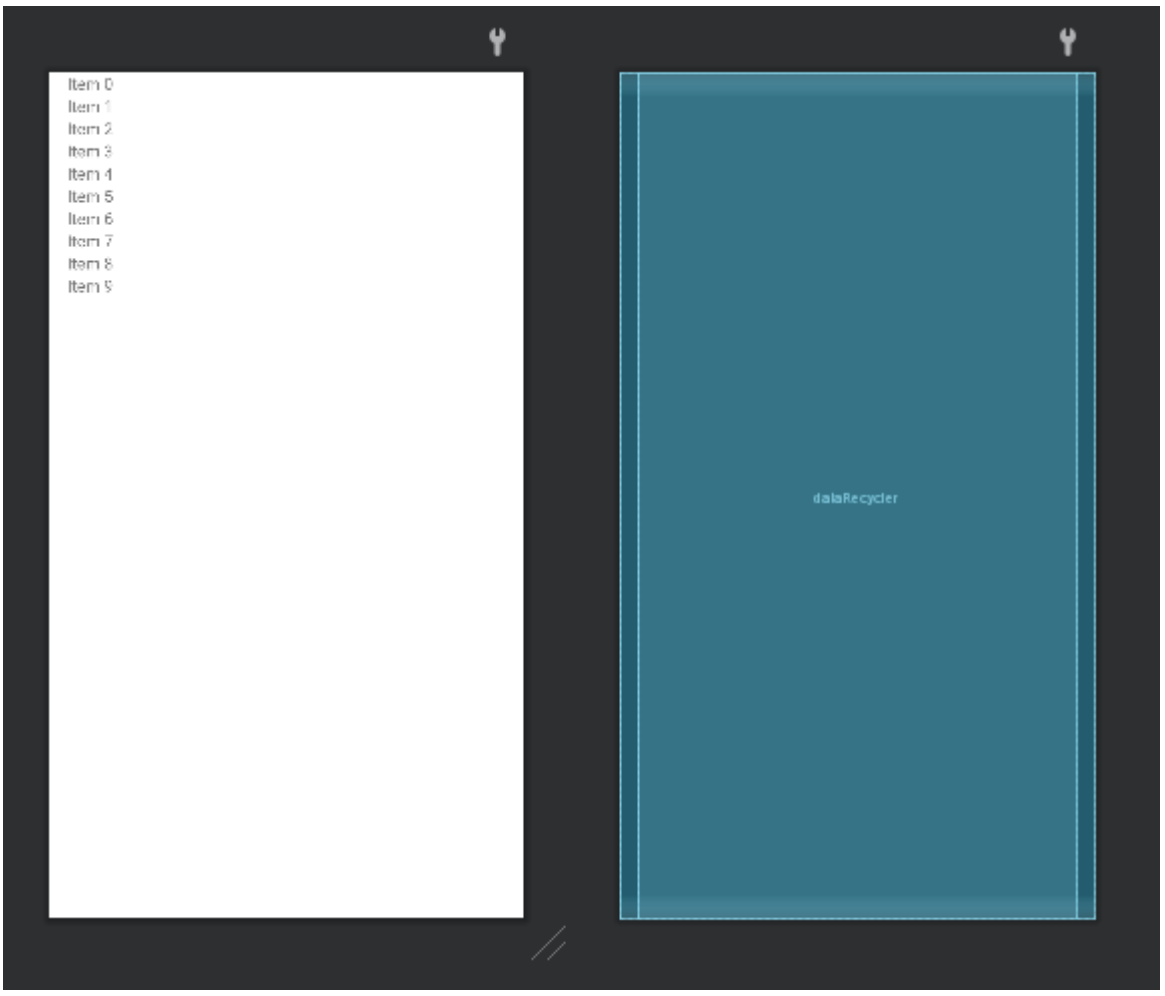
```
tools:context=".MainActivity"
android:orientation="vertical"
android:paddingLeft="16dp"
android:paddingRight="16dp"
android:layout_height="match_parent"
android:layout_width="match_parent">
```

Το attribute **tools:context:** Ορίζει την κλάση που θα χρησιμοποιηθεί ως περιβάλλον για τον στατικό ελεγκτή προτύπου (template controller), σε αυτήν την περίπτωση την **MainActivity.**

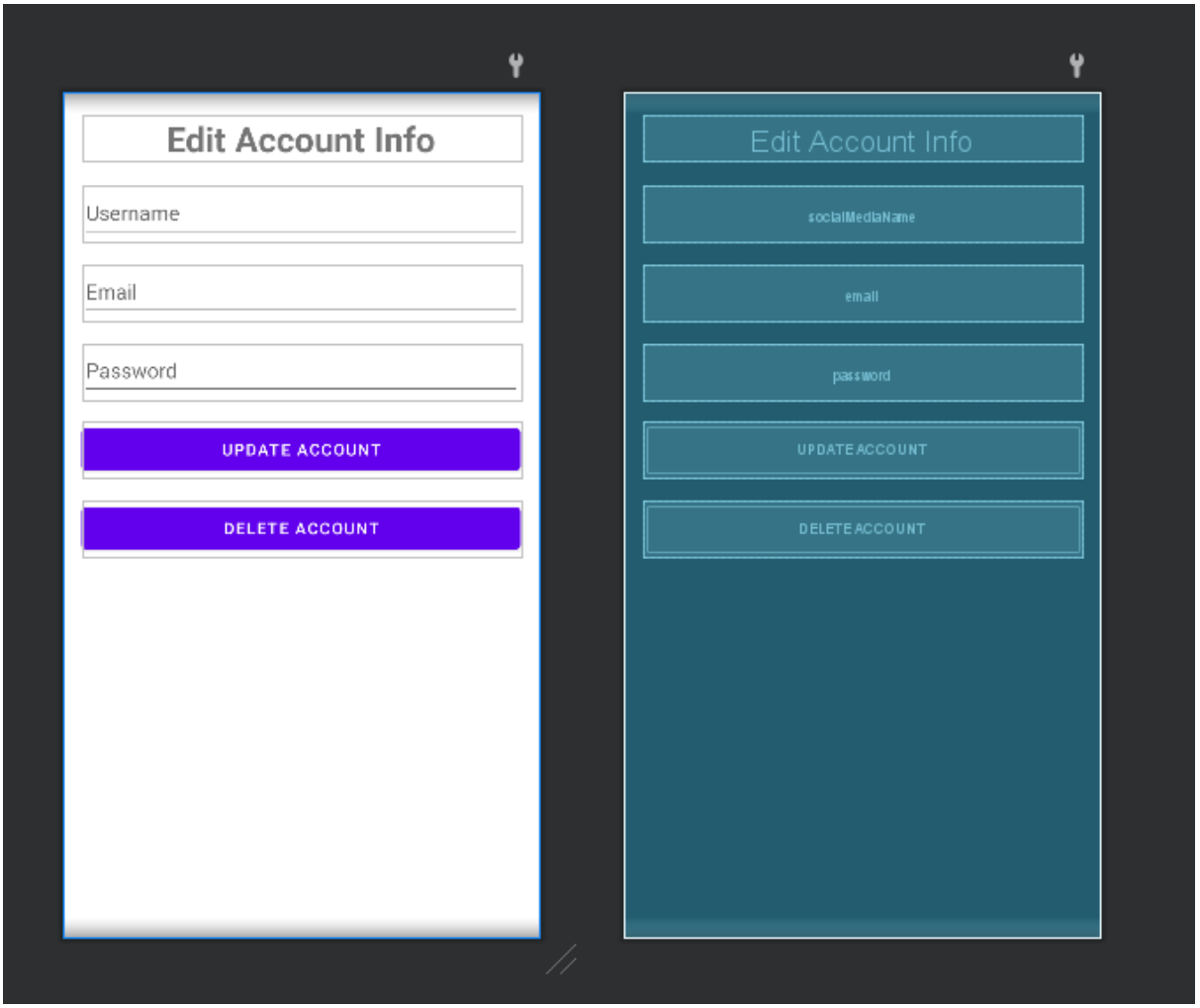
Τα attributes **android:layout_height** και β: Ορίζουν το ύψος και το πλάτος του γραμμικού layout ως "match_parent", που σημαίνει ότι θα καταλάβει όλο το διαθέσιμο χώρο.

Τα στοιχεία **<TextView>**, **<EditText>** και **<Button>**: Ορίζουν αντίστοιχα ένα κείμενο, ένα πεδίο κειμένου και ένα κουμπί.

Τα attributes που χρησιμοποιούνται στα παραπάνω στοιχεία: Ορίζουν διάφορες ιδιότητες για κάθε στοιχείο, όπως το μέγεθος, η ευθυγράμμιση και το περιεχόμενο τους.



activity_update.xml



Ας περιγράψουμε τις κύριες λειτουργίες και ιδιότητες που χρησιμοποιούνται σε αυτόν τον κώδικα:

Τα attributes **xmlns:android, xmlns:app** και **xmlns:tools**: Ορίζουν τους ορθογώνιους χώρους ονομάτων για τις βιβλιοθήκες Android, όπως τα στοιχεία UI (android), χαρακτηριστικά προσαρμογής (app) και εργαλεία ανάπτυξης (tools).

Το στοιχείο **<LinearLayout>:** Ορίζει ένα γραμμικό layout που τοποθετεί τα στοιχεία του παιδιού του κατακόρυφα.

Το attribute **tools:context**: Ορίζει την κλάση που θα χρησιμοποιηθεί ως περιβάλλον για τον στατικό ελεγκτή προτύπου (template controller), σε αυτήν την περίπτωση την **MainActivity**.

Τα attributes **android:layout_height** και **android:layout_width**: Ορίζουν το ύψος και το πλάτος του γραμμικού layout ως "match_parent", που σημαίνει ότι θα καταλάβει όλο το διαθέσιμο χώρο.

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Τα στοιχεία **<TextView>**, **<EditText>** και **<Button>**: Ορίζουν αντίστοιχα ένα κείμενο, ένα πεδίο εισαγωγής κειμένου και ένα κουμπί.

```
<Button
    android:layout_marginTop="20dp"
    android:id="@+id/update_acc"
    android:layout_gravity="center_horizontal"
    android:text="Update Account"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Τα attributes που χρησιμοποιούνται στα παραπάνω στοιχεία: Ορίζουν διάφορες ιδιότητες για κάθε στοιχείο, όπως το μέγεθος, η ευθυγράμμιση, ο τύπος γραμματοσειράς και το περιεχόμενο τους.

Ο παραπάνω κώδικας περιγράφει ένα layout που περιέχει έναν τίτλο, τρία πεδία εισαγωγής κειμένου και δύο κουμπιά.

single_row.xml

Ας περιγράψουμε τις κύριες λειτουργίες και ιδιότητες που χρησιμοποιούνται σε αυτόν τον κώδικα:

Το attribute **xmlns:android:** Ορίζει τον ορθογώνιο χώρο ονομάτων για τα στοιχεία UI της βιβλιοθήκης Android.

Το στοιχείο **<LinearLayout>:** Ορίζει ένα γραμμικό layout που τοποθετεί τα στοιχεία του παιδιού του κατακόρυφα.

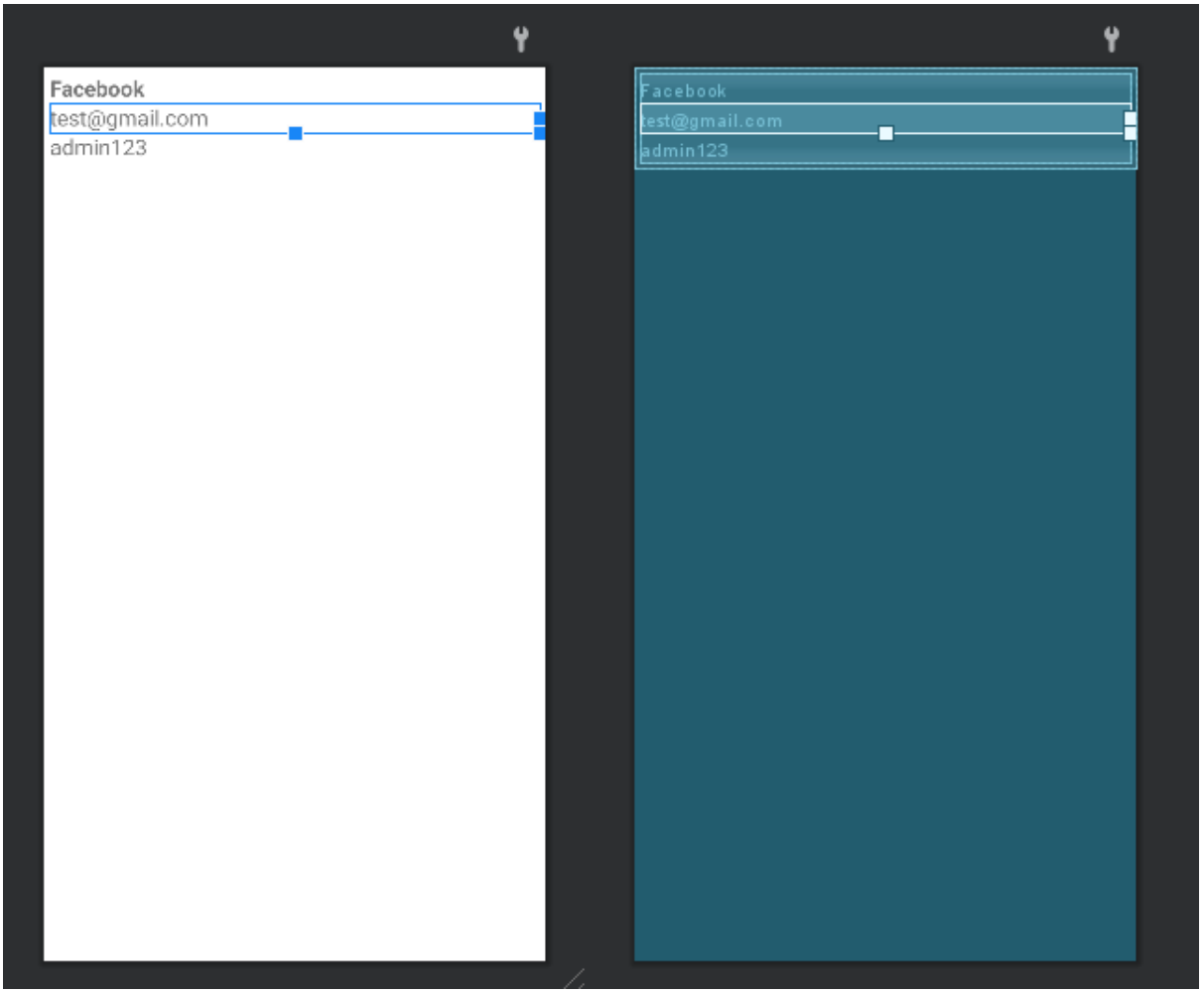
```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="5dp">
```

Τα attributes **android:layout_height** και **android:layout_width:** Ορίζουν το ύψος του γραμμικού layout ως "wrap_content", που σημαίνει ότι θα προσαρμοστεί στο περιεχόμενό του, και το πλάτος του ως "match_parent", που σημαίνει ότι θα καταλάβει τον διαθέσιμο χώρο στον γονέα του.

Τα στοιχεία **<TextView>**: Ορίζουν ένα κείμενο με διάφορες ιδιότητες, όπως το μέγεθος γραμματοσειράς, ο στυλ και το περιεχόμενο τους.

```
<TextView
    android:id="@+id/sName_one"
    android:textSize="18sp"
    android:textStyle="bold"
    android:text="Facebook"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Ο παραπάνω κώδικας περιγράφει ένα layout που περιέχει τρία TextView στοιχεία. Κάθε TextView περιέχει πληροφορίες για έναν λογαριασμό κοινωνικής δικτύωσης, όπως το όνομα του λογαριασμού, το email και ο κωδικός πρόσβασης.



dialog_image.xml

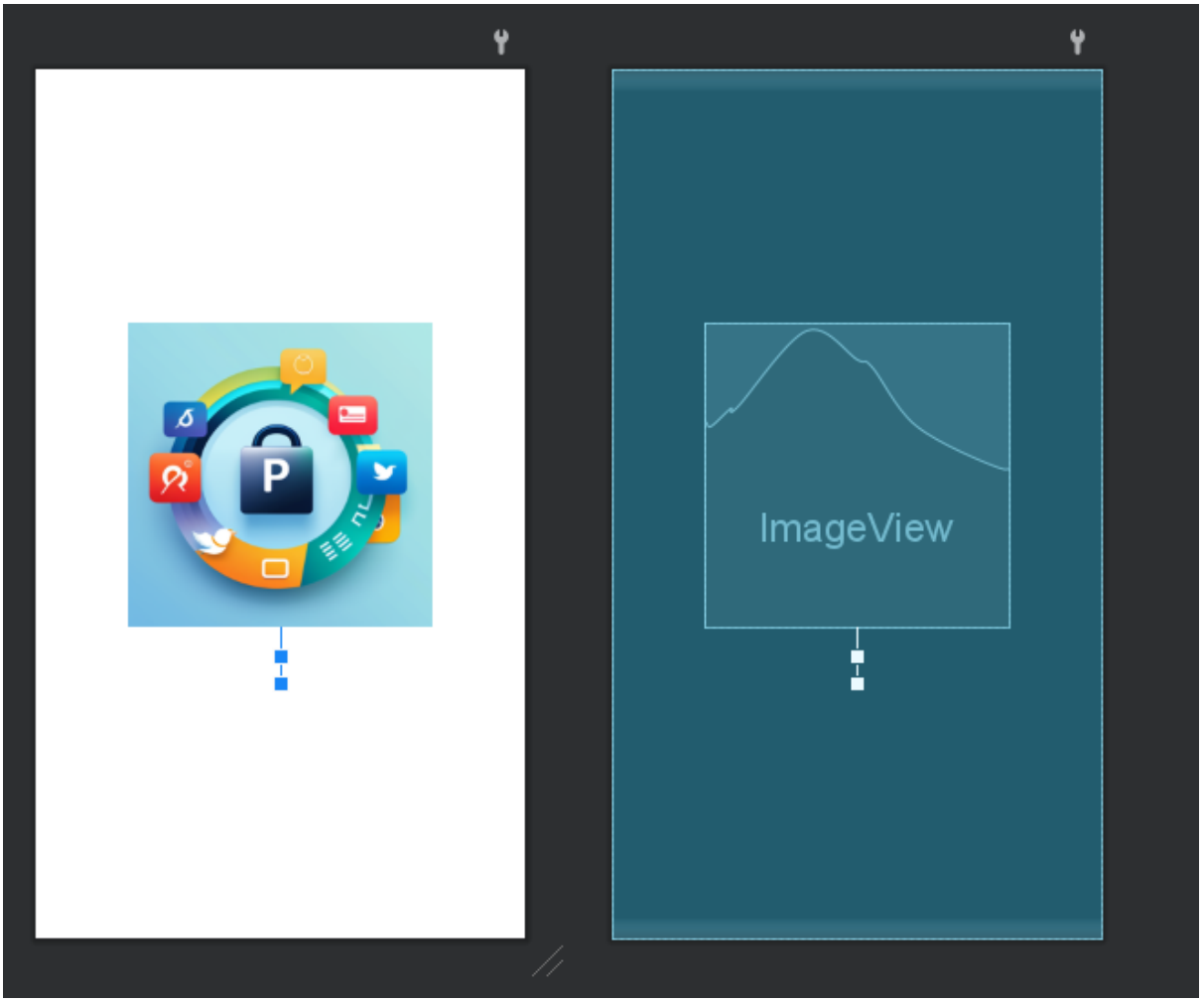
Ας περιγράψουμε τις κύριες λειτουργίες και ιδιότητες που χρησιμοποιούνται σε αυτόν τον κώδικα:

Το attribute **android:orientation**: Ορίζει τον προσανατολισμό του γραμμικού layout. Στην περίπτωση αυτή, ορίζεται ως "vertical", που σημαίνει ότι τα στοιχεία τοποθετούνται κατακόρυφα.

Το στοιχείο **<ImageView>**: Προβάλλει μια εικόνα μέσα στον διάλογο. Το **android:src** ορίζει την πηγή της εικόνας που θα προβληθεί. Στην περίπτωση αυτή, χρησιμοποιείται ένα αρχείο εικόνας με το όνομα "applogo".

```
<ImageView
    android:id="@+id/dialogImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/applogo"/>
```

Το στοιχείο **<TextView>**: Εμφανίζει ένα κείμενο μέσα στον διάλογο. Το **android:text** ορίζει το κείμενο που θα εμφανιστεί. Το **android:textColor** ορίζει το χρώμα του κειμένου. Το **android:textSize** ορίζει το μέγεθος του κειμένου. Το **android:textStyle** ορίζει το στυλ του κειμένου. Στην περίπτωση αυτή, το κείμενο θα προσαρμοστεί στο κέντρο του γονέα του.



Ως βάση δεδομένων χρησιμοποιήθηκαν υπηρεσίες της **Room** που αποθηκεύουν τα δεδομένα τοπικά στην μνήμη του κινητού. Συγκεκριμένα για να το πετύχουμε αυτό χρειάστηκε να τροποποιήσουμε το αρχείο build.gradle(app) και να προσθέσουμε τον παρακάτω κώδικα στο dependencies(Save data in a local database using Room | Android Developers)

```
//room
def room_version = "2.5.1"
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"
```

Για δική μας διευκολυνση προσθέσαμε και το παρακάτω για πιο αμεση προσβαση σε κλασεις και μεθοδους

```
buildFeatures { viewBinding true }
```