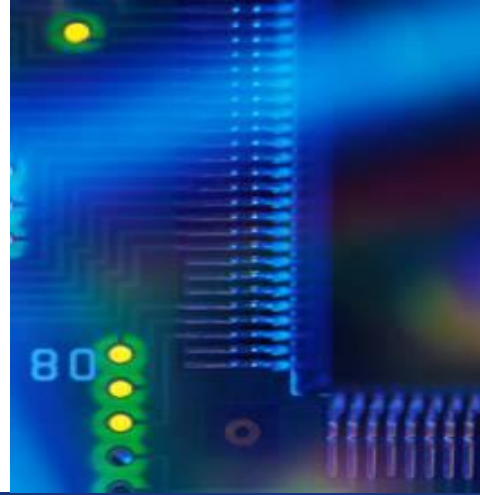
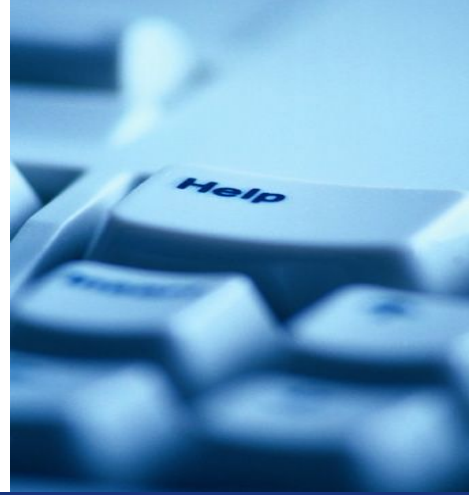




**UNIVERSITAS
BUDI LUHUR**



FAKULTAS TEKNOLOGI INFORMASI

Mobile Programming

[PG119/ 3 SKS]

Riskiana Wulan, MKom



CRUD Operation

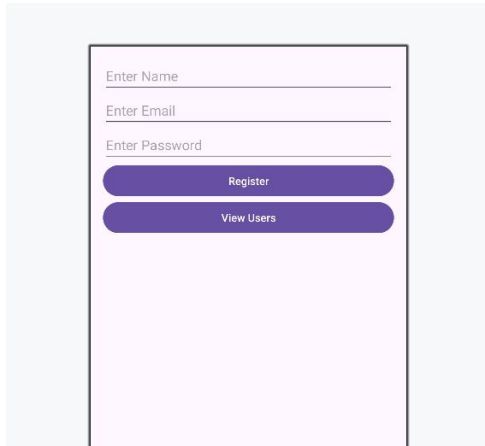
Agenda

1. **Create Database (AppDatabase)**
2. **DAO Interface (UserDao)**
3. **Entity (User)**
4. **Adapter (UserAdapter)**
5. **Activities for Listing, Editing, and Adding Users**
6. **Layouts for all screens**

1. Create Layouts

Main Layout

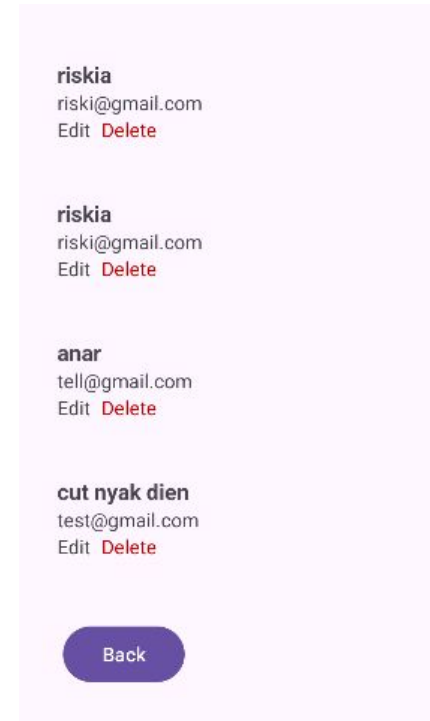
Jika register button diklik, maka akan menyimpan user baru ke database



The Main Layout form is a light purple rectangular box with a thin black border. It contains three input fields at the top, each with a placeholder text: 'Enter Name', 'Enter Email', and 'Enter Password'. Below these fields are two rounded rectangular buttons. The first button is purple with the text 'Register' in white. The second button is also purple with the text 'View Users' in white.

Jika View Users diklik, maka akan muncul layar view users

View Users Layout



The View Users Layout is a light purple rectangular box. It displays a list of users. Each user entry consists of a username, an email address, and two action links: 'Edit' and 'Delete'. The 'Delete' link is in red. At the bottom of the layout is a rounded rectangular button with the text 'Back' in white.

riskia	riski@gmail.com	Edit	Delete
riskia	riski@gmail.com	Edit	Delete
anar	tell@gmail.com	Edit	Delete
cut nyak dien	test@gmail.com	Edit	Delete

Back

Edit User layout

Saat link edit diklik, maka akan muncul layar edit user

EDIT USER

Name: cut nyak dien

test@gmail.com

.....

Save

Delete Link

Saat delete link diklik, maka akan muncul konfirmasi. Yes, maka akan menghapus user dari database

anar

tell@gmail.com

Edit Delete

cut nyak dien

cut@gmail.com

Edit Delete

Delete User

Are you sure you want to delete cut nyak dien?

NO

YES

activity_view_users.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout android:layout_width="match_parent"
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_height="match_parent"
5      android:orientation="vertical"
6      android:padding="16dp">
7      <androidx.recyclerview.widget.RecyclerView
8          android:id="@+id/recyclerViewUsers"
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:layout_marginTop="16dp" />
12
13     <Button
14         android:layout_margin="20dp"
15         android:id="@+id/back_button"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:text="Back" />
19 </LinearLayout>

```

activity_main.xml

```

2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical"
6      android:padding="16dp">
7      <EditText
8          android:id="@+id/editTextName"
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:hint="Enter Name"
12         android:inputType="text" />
13
14     <EditText
15         android:id="@+id/editTextEmail"
16         android:layout_width="match_parent"
17         android:layout_height="wrap_content"
18         android:hint="Enter Email"
19         android:inputType="textEmailAddress" />
20
21     <EditText
22         android:id="@+id/editTextPassword"
23         android:layout_width="match_parent"
24         android:layout_height="wrap_content"
25         android:hint="Enter Password"
26         android:inputType="textPassword" />
27
28     <Button
29         android:id="@+id/buttonRegister"
30         android:layout_width="match_parent"
31         android:layout_height="wrap_content"
32         android:text="Register" />
33
34     <Button
35         android:id="@+id/btnViewUsers"
36         android:layout_width="match_parent"
37         android:layout_height="wrap_content"
38         android:text="View Users" />
39 </LinearLayout>

```

item_user.xml

```

2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
25
26 <LinearLayout
27     android:layout_width="match_parent"
28     android:layout_height="wrap_content"
29     android:orientation="horizontal">
30
31 <!-- Edit Link -->
32 <TextView
33     android:id="@+id/link_edit"
34     android:layout_width="wrap_content"
35     android:layout_height="wrap_content"
36     android:text="Edit"
37     android:textColor="@color/cardview_dark_background"
38     android:textSize="14sp"
39     android:layout_gravity="center_vertical"
40     android:layout_marginEnd="8dp"
41     android:clickable="true"
42     android:focusable="true"/>
43
44 <!-- Delete Link -->
45 <TextView
46     android:id="@+id/link_delete"
47     android:layout_width="wrap_content"
48     android:layout_height="wrap_content"
49     android:text="Delete"
50     android:textColor="@android:color/holo_red_dark"
51     android:textSize="14sp"
52     android:layout_gravity="center_vertical"
53     android:clickable="true"
54     android:focusable="true"/>
55 </LinearLayout>
56

```

activity_edit_user.xml

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
<LinearLayout>
</LinearLayout>

<EditText
    android:id="@+id/editTextEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Email"
    android:inputType="textEmailAddress"/>

<EditText
    android:id="@+id/editTextPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Password"
    android:inputType="textPassword"/>

<Button
    android:id="@+id/buttonSave"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Save" />

<Button
    android:id="@+id/buttonCancel"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Cancel" />

</LinearLayout>

```




2. Create Database

Database Connection

Ensures that only one instance of AppDatabase is created during the app's lifecycle.

```

@Database(entities = {User.class}, version = 1) 7 usages 1 inheritor
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao(); 1 usage 1 implementation

    private static AppDatabase instance; 3 usages

    public static synchronized AppDatabase getInstance(Context context) { no usages
        if (instance == null) {
            instance = Room.databaseBuilder(context.getApplicationContext(),
                AppDatabase.class, name: "app-database")
                .build();
        }
        return instance;
    }
}
  
```

This is a **singleton pattern** for creating the AppDatabase instance.

- **Ensure a single instance of the database:** Room databases are heavy objects that should be instantiated only once. Creating multiple instances can lead to performance issues and data inconsistencies.
- **Thread safety:** The synchronized keyword ensures that only one thread at a time can access this method, preventing race conditions when multiple threads try to initialize the database.
- **Global access:** The getInstance method provides a globally accessible entry point to the AppDatabase.

Steps in AppDatabase Creation:

1. **Annotation @Database:**
 - Specifies the entities (tables) and the version of the database.
 - User.class is your database table.
2. **Singleton Pattern:**
 - Ensures that only one instance of AppDatabase is created during the app's lifecycle.
3. **Room.databaseBuilder:**
 - Builds the database with the specified name (user-database).
 - Optionally includes features like `fallbackToDestructiveMigration()` to reset the database if the schema changes.



3. Create Entity

Entity represent of users table in the database

```
@Entity(tableName = "users") 47 usages
public class User {
    @PrimaryKey(autoGenerate = true)
    public int id;

    @ColumnInfo(name = "name")
    public String name;

    @ColumnInfo(name = "email") 10 usages
    public String email;

    @ColumnInfo(name = "password") 9 usages
    public String password;

    @Ignore 3 usages
    public User(String name, String email, String password){
        this.name= name;
        this.email = email;
        this.password = password;
    }

    public User(int id, String name, String email, String password) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.password = password;
    }
}
```

//getter and setters



4. Create Data Access Object (DAO)

```
@Dao 9 usages 1 implementation
public interface UserDao {
    @Insert 1 usage 1 implementation
    void insertUser(User user);

    @Query("SELECT * FROM users") 1 usage 1 implementation
    LiveData<List<User>> getAllUsers();

    @Delete 1 usage 1 implementation
    void deleteUser(User user);

    @Update 1 usage 1 implementation
    void updateUser(User user);

    // Rename usages
    @Query("SELECT * FROM users WHERE name = :name LIMIT 1") 1
    User getUserByName(String name);
}
```

LiveData is an observable data holder provided by the Android Architecture Components.

It is lifecycle-aware, meaning it respects the lifecycle of the Android components (like Activity or Fragment) it is observing.

When the data in the LiveData changes, the UI that observes it automatically updates.

```
@Query("SELECT * FROM users")
LiveData<List<User>> getAllUsers();
```

- This method returns a LiveData object containing a list of users.
- The UI observes this LiveData. Whenever the list of users in the database changes, the observer (like a ViewModel or Activity) gets notified, and the UI gets updated automatically.



5. Create Adapter Class

UserAdapter

```

1 public class UserAdapter extends RecyclerView.Adapter<UserAdapter.UserViewHolder> { 7 usages
2     private List<User> users; 5 usages
3     private final OnUserActionListener listener; 3 usages
4
5     public UserAdapter(List<User> users, OnUserActionListener listener) { 1 usage
6         this.users = users; // Initialize with an empty or populated list
7         this.listener = listener;
8     }
9
10    public void setUsers(List<User> users) { 1 usage
11        this.users = users;
12        notifyDataSetChanged();
13    }
14
15    @NonNull
16    @Override
17    public UserAdapter.UserViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
18        View view = LayoutInflater.from(parent.getContext())
19            .inflate(R.layout.item_user, parent, attachToRoot: false);
20
21        return new UserViewHolder(view);
22    }
23
24    @Override
25    public void onBindViewHolder(@NonNull UserAdapter.UserViewHolder holder, int position) {
26        User user = users.get(position);
27        holder.bind(user);
28        // Set click listeners for Edit and Delete links
29        holder.linkEdit.setOnClickListener(v -> listener.onEdit(user));
30        holder.linkDelete.setOnClickListener(v -> listener.onDelete(user));
31    }
32 }
  
```

The `notifyDataSetChanged()` method is part of the `RecyclerView.Adapter` class.

Purpose:

- It tells the `RecyclerView` to **refresh its entire data set and rebind all items** in the adapter to the UI.
- Use this when the dataset backing the adapter (e.g., `List<User>` in `UserAdapter`) has been updated and needs to be reflected in the `RecyclerView`.

```

@Override
public int getItemCount() {
    return users != null ? users.size() : 0;
}

// Listener interface for handling "Edit" and "Delete" actions
public interface OnUserActionListener { 3 usages 1 implementation
    void onEdit(User user); 1 usage 1 implementation

    void onDelete(User user); 1 usage 1 implementation
}

public static class UserViewHolder extends RecyclerView.ViewHolder { 4 usages
    TextView textViewName, textViewEmail, linkEdit, linkDelete; 2 usages

    public UserViewHolder(@NonNull View itemView) { 1 usage
        super(itemView);
        textViewName = itemView.findViewById(R.id.textViewName);
        textViewEmail = itemView.findViewById(R.id.textViewEmail);
        linkEdit = itemView.findViewById(R.id.link_edit);
        linkDelete = itemView.findViewById(R.id.link_delete);
    }

    public void bind(User user) { 1 usage
        textViewName.setText(user.name);
        textViewEmail.setText(user.email);
    }
}

```

```
public void setUsers(List<User> users) {  
    this.users = users;  
    notifyDataSetChanged();  
}
```

- Whenever the user list (users) is updated (e.g., new data is fetched from the database), `notifyDataSetChanged()` ensures that the `RecyclerView` reflects the updated list.

Why is this important?

Without calling `notifyDataSetChanged()`, the `RecyclerView` would not know that its data has changed, and the UI would remain stale.



6. Create Activity Class

MainActivity

```
public class MainActivity extends AppCompatActivity {
    private EditText editTextName, editTextEmail, editTextPassword; 2 usages
    private Button buttonRegister, buttonViews; 2 usages
    private AppDatabase appDatabase; 2 usages

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        appDatabase = AppDatabase.getInstance(getApplicationContext());

        findViews();
        handleRegisterButton();
        handleViewUsersButton();
    }

    private void handleViewUsersButton() { 1 usage
        buttonViews.setOnClickListener(view -> {
            Intent intent = new Intent( packageContext: MainActivity.this, ViewUsersActivity.class);
            startActivity(intent);
        });
    }
}
```

```
private void handleRegisterButton() { 1 usage
    buttonRegister.setOnClickListener(view -> {
        String name = editTextName.getText().toString().trim();
        String email = editTextEmail.getText().toString().trim();
        String password = editTextPassword.getText().toString().trim();

        if (!name.isEmpty() && !email.isEmpty() && !password.isEmpty()) {
            User user = new User(name, email, password);
            new Thread() -> appDatabase.userDao().insertUser(user).start();
            Toast.makeText( context: MainActivity.this, text: "User registered successfully!", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText( context: MainActivity.this, text: "Please fill all fields!", Toast.LENGTH_SHORT).show();
        }
    });
}

private void findViews() { 1 usage
    editTextName = findViewById(R.id.editTextName);
    editTextEmail = findViewById(R.id.editTextEmail);
    editTextPassword = findViewById(R.id.editTextPassword);
    buttonRegister = findViewById(R.id.buttonRegister);
    buttonViews = findViewById(R.id.btnViewUsers);
}
}
```

Terdapat handle register button dan view users button

ViewUsersActivity

```
public class ViewUsersActivity extends AppCompatActivity implements UserAdapter.OnUserActionListener {
    private RecyclerView recyclerView; 4 usages
    private UserAdapter userAdapter; 4 usages
    private UserDao userDao; 3 usages
    private AppDatabase appDatabase; 2 usages
    private Button backButton; 2 usages

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_users);

        recyclerView = findViewById(R.id.recyclerViewUsers);
        backButton = findViewById(R.id.back_button);
        recyclerView.setLayoutManager(new LinearLayoutManager(context: this));

        // Initialize Adapter with an empty list
        userAdapter = new UserAdapter(new ArrayList<>(), listener: this);
        recyclerView.setAdapter(userAdapter);
        recyclerView.setAdapter(userAdapter); // Attach the adapter to RecyclerView

        // Initialize Database
        appDatabase = AppDatabase.getInstance(getApplicationContext());
        userDao = appDatabase.userDao();

        // Observe LiveData and update the adapter with the data from the database
        userDao.getAllUsers().observe(owner: this, users -> {
            if (users != null) {
                userAdapter.setUsers(users); // Update adapter's data with new list of users
            }
        });

        handleBackButton();
    }
}
```

```
// Handle "Edit" action
@Override 1 usage
public void onEdit(User user) {
    // Launch EditUserActivity with the user name
    Intent intent = new Intent(packageContext: ViewUsersActivity.this, EditUserActivity.class);
    intent.putExtra(name: "USER_NAME", user.getName());
    intent.putExtra(name: "USER_ID", user.getId());
    startActivity(intent);
}

// Handle "Delete" action
@Override 1 usage
public void onDelete(User user) {
    // Show a confirmation dialog before deleting
}
}
```

EditUserActivity

```
public class EditUserActivity extends AppCompatActivity {
    private EditText editEmail, editPassword; 3 usages
    private TextView textViewUsername; 2 usages
    private Button saveButton, cancelButton; 2 usages
    private UserDao userDao; 3 usages
    private String username; 5 usages
    private Integer userid; 2 usages
    private AppDatabase appDatabase; 2 usages

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_edit_user);

        initializeViews();

        // Initialize DAO
        appDatabase = AppDatabase.getInstance(getApplicationContext());
        userDao = appDatabase.userDao();

        // Get user ID passed from ViewUsersActivity
        userid = getIntent().getIntExtra( name: "USER_ID", defaultValue: -1);
        username = getIntent().getStringExtra( name: "USER_NAME");
        if (username != null) {
            textViewUsername.setText(username);
            // Fetch user data and populate the form
            new Thread() -> {
                User user = userDao.getUserByName(username);
                runOnUiThread() -> {
                    editPassword.setText(user.getPassword());
                    editEmail.setText(user.getEmail());
                };
            }.start();
        }
    }
}
```

```
56 // Save changes
57 saveButton.setOnClickListener(v -> {
58     String password = editPassword.getText().toString();
59     String email = editEmail.getText().toString();
60
61     if (!password.isEmpty() && !email.isEmpty()) {
62         User updatedUser = new User(userid, username, email, password);
63         Log.d( tag: "updatedUser", updatedUser.toString());
64         new Thread() -> {
65             userDao.updateUser(updatedUser); // Update the user in the background
66             runOnUiThread() -> {
67                 Toast.makeText( context: EditUserActivity.this, text: "User updated", Toast.LENGTH_SHORT).show();
68                 finish(); // Close the activity
69             };
70         }.start();
71     }
72 });
73 cancelButton.setOnClickListener(v -> {
74     Intent intent = new Intent( packageContext: EditUserActivity.this, ViewUsersActivity.class);
75     startActivity(intent);
76 });
77 }
78
79 private void initializeViews() { 1 usage
80     textViewUsername = findViewById(R.id.textViewName);
81     editPassword = findViewById(R.id.editTextPassword);
82     editEmail = findViewById(R.id.editTextEmail);
83     saveButton = findViewById(R.id.buttonSave);
84     cancelButton = findViewById(R.id.buttonCancel);
85 }
86 }
```

- The `runOnUiThread()` method ensures that the code inside its `Runnable` is executed **on the main thread**.
- You typically use it when performing operations in background threads (e.g., database or network operations) and need to reflect results on the UI.

Device Explorer

- View -> tool windows-> Device Explorer
- The apps' data directory :

`/data/data/<your.application.package.name>/databases/`

Your database will be found in there.

Device Explorer				
Pixel 6a API 35 Android 15.0 ("VanillaIceCream")				
Files Processes				
				
Name	Permissions	Date	Size	
▼ /	drwxr-xr-x	2009-01-01 07:00	679 B	
> acct	drwxr-xr-x	2009-01-01 07:00	27 B	
> apex	drwxr-xr-x	2024-12-10 07:22	1.6 KB	
> bin	lrw-r--r--	2009-01-01 07:00	11 B	
> bootstrap-apex	drwxr-xr-x	2024-12-10 07:22	220 B	
> cache	drwxrwx---	2009-01-01 07:00	27 B	
> config	drwxr-xr-x	2024-12-10 07:22	0 B	
> d	lrw-r--r--	2009-01-01 07:00	17 B	
▼ data	drwxrwx--x	2024-12-10 07:22	4 KB	
> app	drwxrwx--x	2024-12-10 07:22	4 KB	
▼ data	drwxrwx--x	2024-12-10 07:22	4 KB	
▼ com.ubl.mysimpleapp	drwxrwx--x	2024-12-10 07:22	4 KB	
> .agent-logs	drwx-----	2024-12-13 08:46	4 KB	
> cache	drwxrws--x	2024-12-13 08:44	4 KB	
> code_cache	drwxrws--x	2024-12-13 08:46	4 KB	
▼ databases	drwxrwx--x	2024-12-13 08:44	4 KB	
≡ app-database	-rw-rw----	2024-12-13 08:44	4 KB	
≡ app-database-shm	-rw-----	2024-12-13 08:46	32 KB	
≡ app-database-wal	-rw-----	2024-12-13 08:46	52.3 KB	