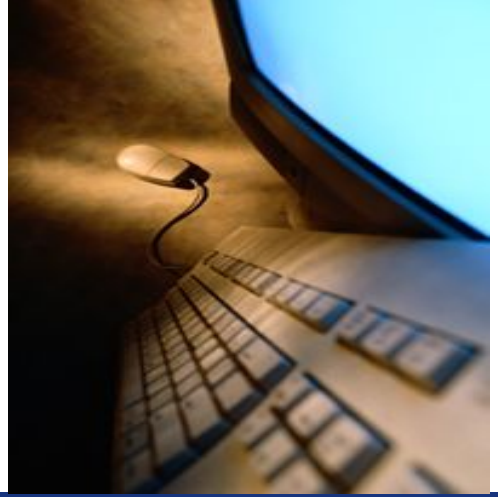
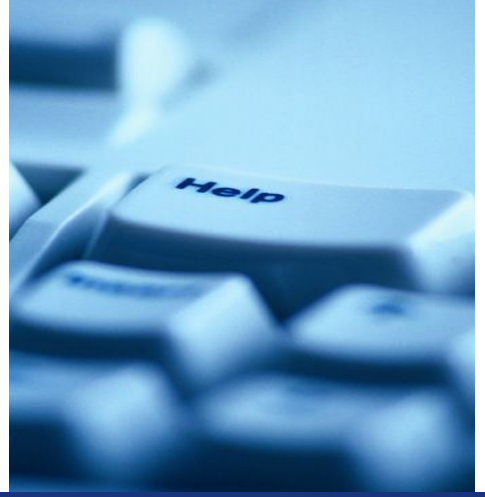
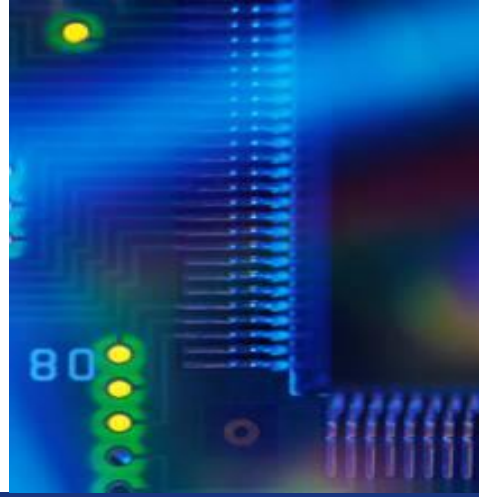
UNIVERSITAS
BUDI LUHUR



FAKULTAS TEKNOLOGI INFORMASI

Mobile Programming

[PG119/ 3 SKS]

Riskiana Wulan, MKom

CRUD Operation

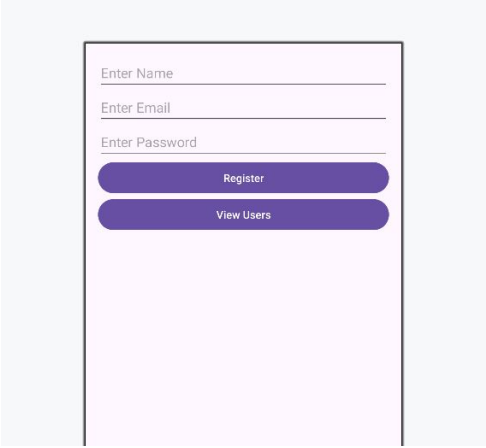
Agenda

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

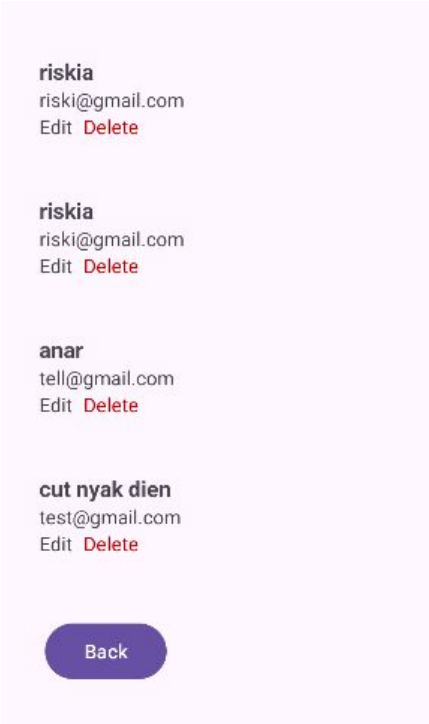
Main Layout

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



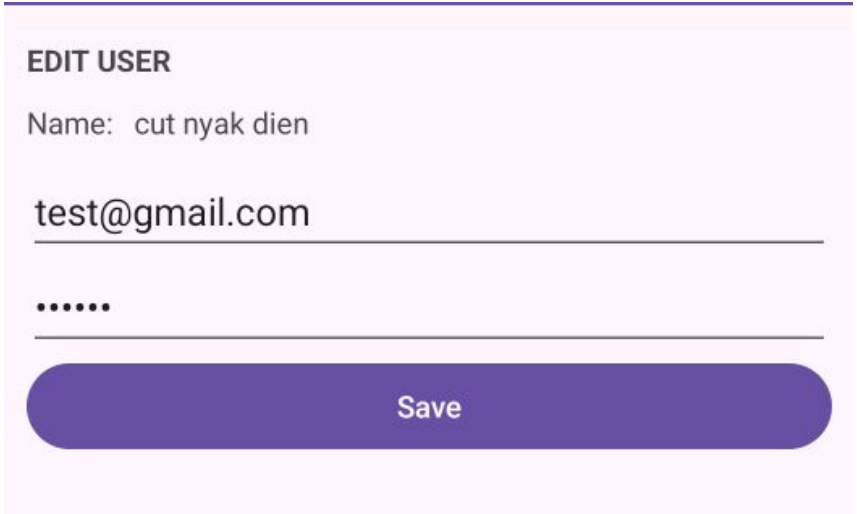
Jika View Users diklik, maka akan muncul layar view users

View Users Layout



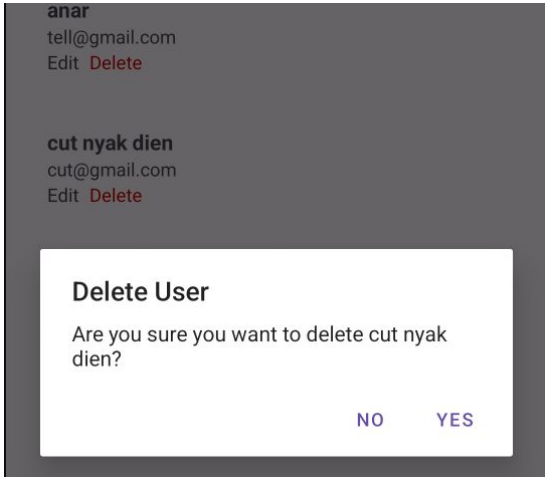
Edit User layout

Saat link edit diklik, maka akan muncul layar edit user



Delete Link

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



activity_view_users.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerViewUsers"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp" />

    <Button
        android:layout_margin="20dp"
        android:id="@+id/back_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Back"/>
</LinearLayout>
```

activity_main.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    <EditText
        android:id="@+id/editTextName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Name"
        android:inputType="text" />

    <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/buttonRegister"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Register" />

    <Button
        android:id="@+id/btnViewUsers"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="View Users" />
</LinearLayout>
```

item_user.xml

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <!-- Edit Link -->
        <TextView
            android:id="@+id/link_edit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Edit"
            android:textColor="@color/cardview_dark_background"
            android:textSize="14sp"
            android:layout_gravity="center_vertical"
            android:layout_marginEnd="8dp"
            android:clickable="true"
            android:focusable="true"/>

        <!-- Delete Link -->
        <TextView
            android:id="@+id/link_delete"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Delete"
            android:textColor="@android:color/holo_red_dark"
            android:textSize="14sp"
            android:layout_gravity="center_vertical"
            android:clickable="true"
            android:focusable="true"/>
    </LinearLayout>
```

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


```java
@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;
    }
```

Entity represent of users table in the database

//getter and setters

4. Create Data Access Object (DAO)


```java
@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

```java
public class UserAdapter extends RecyclerView.Adapter<UserAdapter.UserViewHolder> {  7 usages
    private List<User> users;  5 usages
    private final OnUserActionListener listener;  3 usages

    public UserAdapter(List<User> users, OnUserActionListener listener) {  1 usage
        this.users = users; // Initialize with an empty or populated list
        this.listener = listener;
    }

    public void setUsers(List<User> users) {  1 usage
        this.users = users;
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public UserAdapter.UserViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_user, parent,  attachToRoot: false);

        return new UserViewHolder(view);

    }

    @Override
    public void onBindViewHolder(@NonNull UserAdapter.UserViewHolder holder, int position) {
        User user = users.get(position);
        holder.bind(user);
        // Set click listeners for Edit and Delete links
        holder.linkEdit.setOnClickListener(v -> listener.onEdit(user));
        holder.linkDelete.setOnClickListener(v -> listener.onDelete(user));

    }
```

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.


```java
@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

```java
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);
        });
    }
```

```java
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

```java
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();
```

```java
    // 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

```java
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();
        }
```

```java
        // Save changes
        saveButton.setOnClickListener(v -> {
            String password = editPassword.getText().toString();
            String email = editEmail.getText().toString();

            if (!password.isEmpty() && !email.isEmpty()) {
                User updatedUser = new User(userid, username, email, password);
                Log.d( tag: "updatedUser", updatedUser.toString());
                new Thread(() -> {
                    userDao.updateUser(updatedUser);  // Update the user in the background
                    runOnUiThread(() -> {
                        Toast.makeText( context: EditUserActivity.this,  text: "User updated", Toast.LENGTH_SHORT).show();
                        finish();  // Close the activity
                    });
                }).start();
            }
        });
        cancelButton.setOnClickListener(v -> {
            Intent intent = new Intent( packageContext: EditUserActivity.this, ViewUsersActivity.class);
            startActivity(intent);
        });
    }

    private void initializeViews() {  1 usage
        textViewUsername = findViewById(R.id.textViewName);
        editPassword = findViewById(R.id.editTextPassword);
        editEmail = findViewById(R.id.editTextEmail);
        saveButton = findViewById(R.id.buttonSave);
        cancelButton= findViewById(R.id.buttonCancel);
    }
}
```


- 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.