Loading Indicator

This lesson will cover how to display a loading indicator and prevent user interaction with the views.

WE'LL COVER THE FOLLOWING

Flow overview

When the user clicks the login button, we will perform data validation flow. If the data is valid, proceed to:

1. Hide the login button to prevent a user from clicking the button
2. Disable the username and password input fields to prevent a user from changing the text

Layout update

To show the indeterminate loading indicator, we are going to use a `ProgressBar` view. Let's make this view appear instead of the *login* button. To do so, we need to add constraints relative to *login* button:

- `app:layout_constraintBottom_toBottomOf="@+id/loginButton"`
- `app:layout_constraintEnd_toEndOf="@+id/loginButton"`
- `app:layout_constraintStart_toStartOf="@+id/loginButton"`
- `app:layout_constraintTop_toTopOf="@+id/loginButton"`

The `ProgressBar` must be invisible by default, which can be done via the `visibility` attribute with the `invisible` parameter value.

```
<androidx.constraintlayout.widget.ConstraintLayout
```

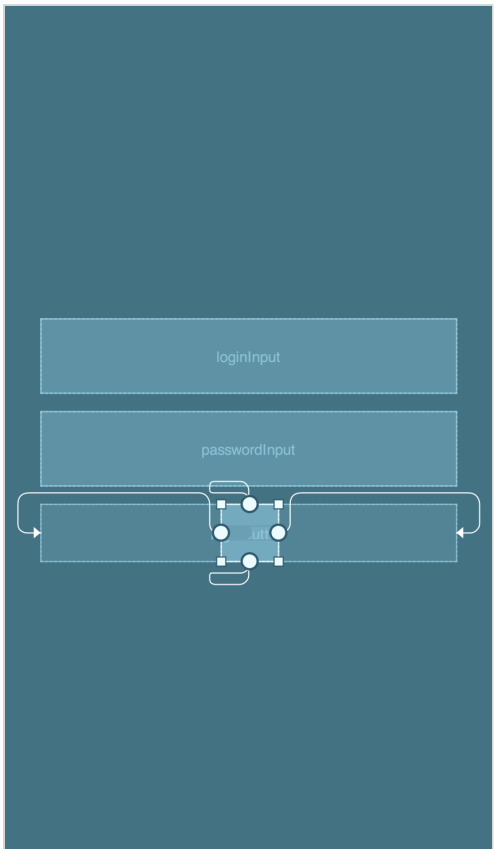
```
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    ...
    <com.google.android.material.button.MaterialButton
            android:id="@+id/loginButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="32dp"
            android:layout_marginTop="16dp"
            android:layout_marginEnd="32dp"
            android:text="Login"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textPasswordInput"/>

    <ProgressBar
            android:id="@+id/progressBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:visibility="invisible"
            app:layout_constraintBottom_toBottomOf="@+id/loginButton"
            app:layout_constraintEnd_toEndOf="@+id/loginButton"
            app:layout_constraintStart_toStartOf="@+id/loginButton"
            app:layout_constraintTop_toTopOf="@+id/loginButton" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_login.xml

As you can see in the preview below, the loading indicator is aligned with the login button.



Display loading indicator

Let's get back to our `LoginActivity` and bind the `ProgressBar` view from XML to Java objects via `findViewById` method.

```
public class LoginActivity extends AppCompatActivity {

    ...
    private ProgressBar progressBar;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        ...
        progressBar = findViewById(R.id.progressBar);
    }
}
```

LoginActivity

Next, let's add the final `else` block in the `onLoginClicked` method.

```
private void onLoginClicked() {
    String username = textUsernameLayout.getEditText().getText().toString();
    String password = textPasswordInput.getEditText().getText().toString();
    if (username.isEmpty()) {
        textUsernameLayout.setError("Username must not be empty");
    } else if (password.isEmpty()) {
        textPasswordInput.setError("Password must not be empty");
    } else if (!username.equals("admin") && !password.equals("admin")) {
        showErrorDialog();
    } else {
        performLogin();
    }
}
```

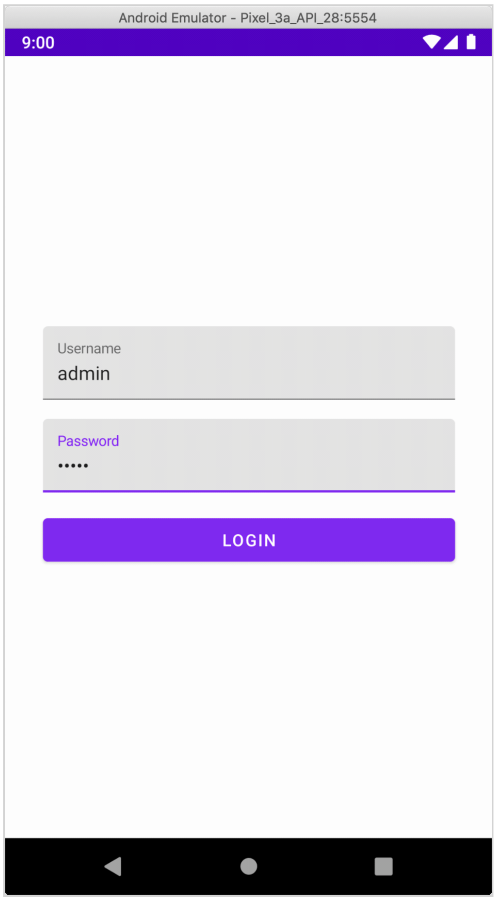
LoginActivity

Now, we are ready to create a `performLogin` method where we will hide the *login* button via the `setVisibility(View.INVISIBLE)` method and show the loading indicator via the `setVisibility(View.VISIBLE)` method.

```
private void performLogin() {
    loginButton.setVisibility(View.INVISIBLE);
    progressBar.setVisibility(View.VISIBLE);
}
```

LoginActivity

As you can see in the preview below, the *login* button becomes invisible when the loading indicator is visible.



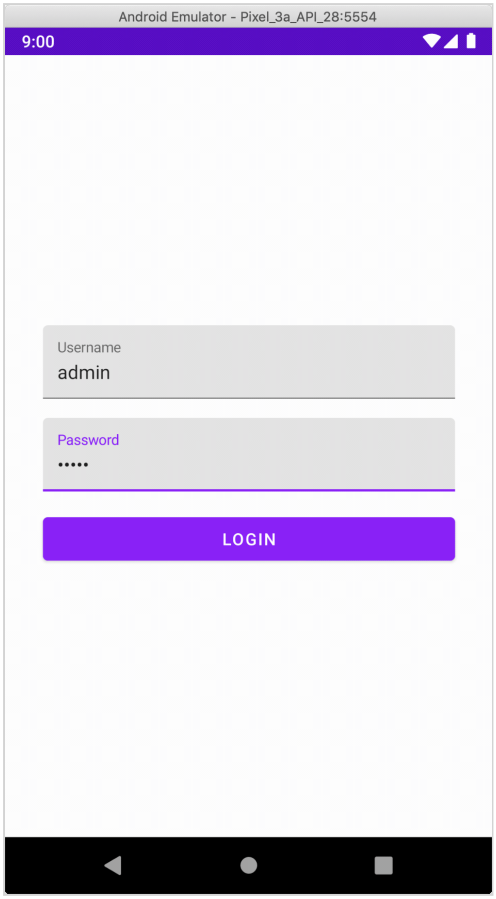
One last thing which we need to add to complete the loading indicator flow is to disable the *username* and *password* input fields to prevent a user from changing the text while loading is in progress.

This can be easily achieved via the `setEnabled(false)` method.

```
private void performLogin() {
    textUsernameLayout.setEnabled(false);
    textPasswordInput.setEnabled(false);
    loginButton.setVisibility(View.INVISIBLE);
    progressBar.setVisibility(View.VISIBLE);
}
```

LoginActivity

As you can see on the preview below, the *username* and *password* input fields are disabled when the loading indicator is visible.



Hit the *run* button to try it yourself.

```
package com.travelblog;

import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import com.google.android.material.textfield.TextInputLayout;

public class LoginActivity extends AppCompatActivity {

    private TextInputLayout textUsernameLayout;
    private TextInputLayout textPasswordInput;
    private ProgressBar progressBar;
    private Button loginButton;

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

        textUsernameLayout = findViewById(R.id.textUsernameLayout);
        textPasswordInput = findViewById(R.id.textPasswordInput);
```

```
    progressBar = findViewById(R.id.progressBar);
    loginButton = findViewById(R.id.loginButton);
    loginButton.setOnClickListener(v -> LoginActivity.this.onLoginClicked());

    textUsernameLayout
            .getEditText()
            .addTextChangedListener(createTextWatcher(textUsernameLayout));

    textPasswordInput
            .getEditText()
            .addTextChangedListener(createTextWatcher(textPasswordInput));
}

private void onLoginClicked() {
    String username = textUsernameLayout.getEditText().getText().toString();
    String password = textPasswordInput.getEditText().getText().toString();
    if (username.isEmpty()) {
        textUsernameLayout.setError("Username must not be empty");
    } else if (password.isEmpty()) {
        textPasswordInput.setError("Password must not be empty");
    } else if (!username.equals("admin") && !password.equals("admin")) {
        showErrorDialog();
    } else {
        performLogin();
    }
}

private void performLogin() {
    textUsernameLayout.setEnabled(false);
    textPasswordInput.setEnabled(false);
    loginButton.setVisibility(View.INVISIBLE);
    progressBar.setVisibility(View.VISIBLE);
}

private void showErrorDialog() {
    new AlertDialog.Builder(this)
            .setTitle("Login Failed")
            .setMessage("Username or password is not correct. Please try again.")
            .setPositiveButton("OK", (dialog, which) -> dialog.dismiss())
            .show();
}

private TextWatcher createTextWatcher(TextInputLayout textPasswordInput) {
    return new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s,
                                      int start, int count, int after) {
            // not needed
        }

        @Override
        public void onTextChanged(CharSequence s,
                                  int start, int before, int count) {
            textPasswordInput.setError(null);
        }

        @Override
        public void afterTextChanged(Editable s) {
            // not needed
        }
    };
}
```

```
}
```

In the next lesson, we will cover how to simulate a long-running operation and open a new screen.