Custom Styles

This lesson will cover how to apply custom styles.

WE'LL COVER THE FOLLOWING

- Recap
- Theme overwrite
- Individual styles

Recap

The login screen is almost ready. It's time to polish the user interface with some custom styles and colors.

Let's see what we need to do before applying custom styles. First, we need to make sure that our activity extends `AppCompatActivity`.

```
public class LoginActivity extends AppCompatActivity {
    ...
}
```

LoginActivity

Next, we need to use one of the material component themes inside the activity or application tags in the *AndroidManifest.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.travelblog">
    <application
            android:theme="@style/Theme.MaterialComponents.DayNight.NoActionBar"
            android:label="Travel Blog">
        ...
    </application>
</manifest>
```

AndroidManifest.xml

Now, all the user interface components are going to use default styles declared in the `Theme.MaterialComponents.DayNight.NoActionBar`. Let's try to extend this theme and overwrite default colors.

Theme overwrite

Create `styles.xml` file inside `app/src/main/res/values` folder. That's where all our styles are going to live. Declare a style with the name `AppTheme` and, as a parent, use `Theme.MaterialComponents.DayNight.NoActionBar`. Now, we can overwrite any item by declaring it inside `AppTheme` with the exact same name as in parent theme.

We are going to overwrite three colors:

- `colorPrimary` - This color is used to paint most of the main components. This is usually a neutral color, for our case we are going to use *dark blue 700* from the material design site.
- `colorSecondary` - This color is used to paint other parts of the main components. This is usually a bright color, for our case we are going to use *orange 500* from the material design site.
- `colorPrimaryDark` - This color is used to paint the system bar. This is usually a darker version of `colorPrimary`, for our case we are going to use *dark blue 800* from the material design site.

```
<resources>
    <style name="AppTheme" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <item name="colorPrimary">#344955</item>
        <item name="colorSecondary">#F9AA33</item>
        <item name="colorPrimaryDark">#232F34</item>
    </style>
</resources>
```

styles.xml

Defining colors directly in the styles file is not a good idea since it limits color reuse and promotes copy-pasting colors all over the place. The better approach would be to declare colors in one file and use ID references to use those colors.

Create `colors.xml` file inside `app/src/main/res/values` folder. That's where all

Create `colors.xml` file inside `app/src/main/res/values` folder. That's where all our colors are going to live. Declare the three colors mentioned above in this file.

```
<resources>
    <color name="blue700">#344955</color>
    <color name="blue800">#232F34</color>
    <color name="orange500">#F9AA33</color>
</resources>
```

colors.xml

Now that we have our colors declared, we can use them in `AppTheme` via ID references starting with @color.

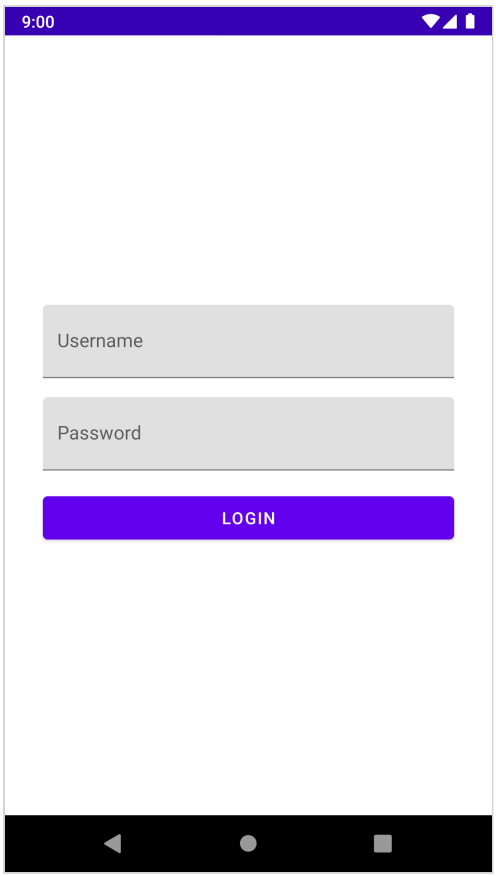
```
<resources>
    <style name="AppTheme" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <item name="colorPrimary">@color/blue700</item>
        <item name="colorSecondary">@color/orange500</item>
        <item name="colorPrimaryDark">@color/blue800</item>
    </style>
</resources>
```

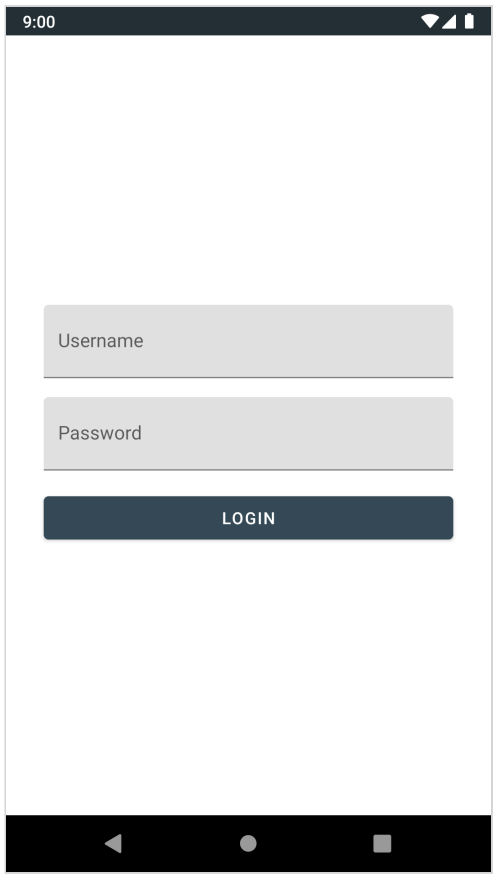
styles.xml

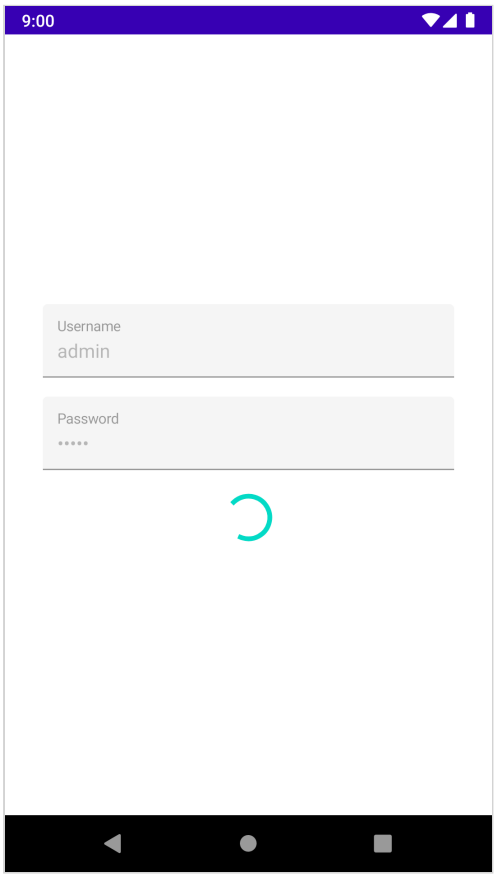
Finally, let's switch the application theme to the newly created in *AndroidManifest.xml* file.

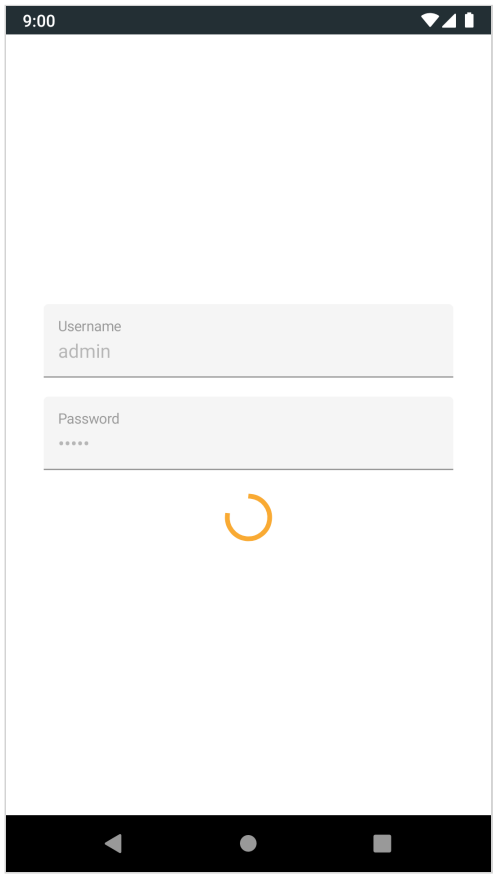
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.travelblog">
    <application
            android:theme="@style/AppTheme"
            android:label="Travel Blog">
        ...
    </application>
</manifest>
```

As you can see in the preview below, now instead of violet and green our components are painted in dark blue and orange.







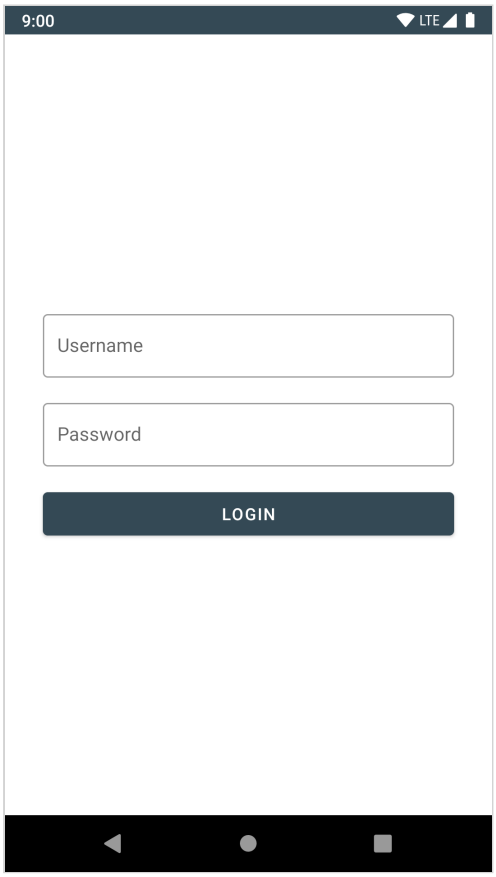


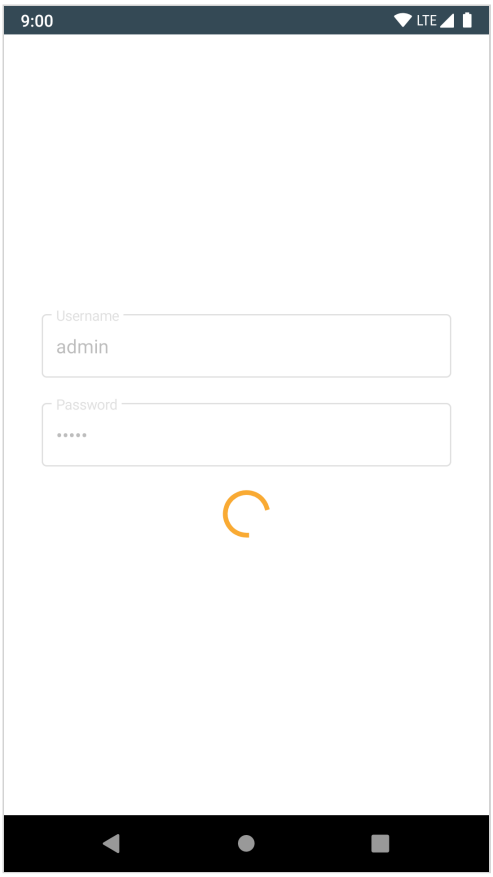
Individual styles

Very often we want to change the style of the individual component. To do so, the `style` attribute can be used. Let's use an alternative style for `TextInputLayout` defined in material components text input field guide.

```
<com.google.android.material.textfield.TextInputLayout
        android:id="@+id/textUsernameLayout"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        ... >
    <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/loginInput"
            ... />
</com.google.android.material.textfield.TextInputLayout>
```

activity_login.xml





Hit the *run* button to try it yourself.

```
package com.travelblog;

import android.content.Context;
import android.content.SharedPreferences;

public class BlogPreferences {

    private static final String KEY_LOGIN_STATE = "key_login_state";

    private SharedPreferences preferences;

    BlogPreferences(Context context) {
        preferences =
                context.getSharedPreferences("travel-blog", Context.MODE_PRIVATE);
    }
```

```
    public boolean isLoggedIn() {
        return preferences.getBoolean(KEY_LOGIN_STATE, false);
    }


    public void setLoggedIn(boolean loggedIn) {
        preferences.edit().putBoolean(KEY_LOGIN_STATE, loggedIn).apply();
    }
}
```