かなりのコーディングを行ったので、次は学習しやすくするために、少しコードを整理し、コードをリファクタリングする良い機会です。コードに追加します。

今、このレッスンはあなたのための挑戦として設計されています。

このレッスンの目標は、既存のコードを簡素化し、再編成することです。

このプロセスでは、スタブ内のコードに精通し、コードをモジュール化する方法、ウィジェットを抽出する方法、および定数ファイルを操作する方法を確認します。レッスンの最後に、アプリのデザインに最後の仕上げを行います。コードをリファクタリングしたおかげで、この変更がどれほど簡単に行えるかがわかります。

まず最初に。本当に必要なくなったprintステートメントをすべて削除して、コードのすべての関連部分をグループ化します。

ようこそ画面で、下にスクロールすると、これらのウィジェットの2つがあります。これらのパディングウィジェットは、マテリアルを標高で囲んでいます。

そのため、少し影が落ちます。

また、関連する画面に移動するためのマテリアルボタンが含まれています。

現在、これらの2つのパディングウィジェットは、タイトルとして含まれているテキスト、押されたときの動作、およびそれらの色を除いて、ほとんど同じです。これらのウィジェットは、ウェルカム画面、ログイン画面、登録画面の3つの画面に存在します。

だから、ここに挑戦の一部です。

これらのパディングウィジェットを個別のステートレスウィジェットにリファクタリングできますか？それを使用する場合、色プロパティとonPressed関数、およびボタンが表示するテキストを単純に渡しますか？

課題の一部として、かなりの数のステップが関係しています。

そのため、コースリソースに記載されている手順をステップごとに説明しました。

そのため、ビデオを一時停止して、このチャレンジを完了してみてください。

大丈夫。

そこで、これが課題の1つに対する解決策です。

いつものように、ウィジェットをリファクタリングする最も簡単な方法は、Flutterアウトラインに移動して、抽出するすべてのサブウィジェットを含むトップレベルウィジェットを選択し、それを右クリックして抽出ウィジェットをクリックすることです。

今、私はRoundedButtonを呼び出して、リファクタリングをクリックします。そして今、私たちは別個のステートレスウィジェットとして丸いボタンを持っています。

次に、その抽出から作成されたコンストラクターを削除し、3つのプロパティを初期化する独自のコンストラクターを作成します。タイプの色のプロパティになりますが、私はそれをイギリスの方法またはアメリカの方法のどちらか好きな色で綴るだけで、文字列を作成しますボタンのタイトル。

そして最後に、onPressedになる関数があります。ボタンが押されたときに何が起こるかです。そして、私のコンストラクターで、これら3つすべてを作成します。

this.title、this.color、this.onPressedです。

これがボタンであることを考えると、onPressedが必要であるというのは理にかなっていますか？

したがって、thi.onPressedの前にrequiredという名前の注釈を追加して、丸いボタンを作成するときにonPressedを持たないようにします。

これで、プロパティでコンストラクターの処理が完了したので、このステートレスウィジェットのビルドメソッド内でコンストラクターを実際に使用できます。

そのため、色の代わりに、渡される色に変更します。ログインしたテキストウィジェットの代わりに、渡されるタイトルに変更しますに。

そして、onPressedについては、現在のコールバックをそこから単純に切り取ります。少し後で使用するので、代わりに渡される関数を使用するからです。

丸いボタンを作成したので、あとは必要な場所で使用するだけです。

Dart 2.0の時点で前に述べたように、新しいオブジェクトを作成するときに新しいキーワードやconstキーワードを使用する必要がなくなったので、それを削除できます。そして、現在の丸いボタンは、必要なonPressedプロパティがあるため、警告を発しています。

先に進み、すべてのプロパティを入力しましょう。

そのため、タイトルは以前と同じようにログインされます。

色は、colors.lightBlueAccentになり、最終的にonPressedは、以前ログインしていたコールバックになります。これは、ログイン画面をナビゲーションスタックにプッシュするナビゲーターです。

ここで、他のパディングウィジェットでもまったく同じことを行います。ここまで削除し、代わりに別の丸いボタンを使用します。

しかし、これはもちろん、代わりにレジスターと言い、ライトブルーのアクセントの代わりにブルーのアクセントの色を持ちます。

また、ログイン画面に移動する代わりにこのボタンを押すと、登録画面に移動するはずです。したがって、それをリファクタリングするために必要なことはこれだけです。

そして今、私たちの画面のビルド方法は大幅に簡素化され、ネストはずっと少なくなっています。そのため、実際にこのクラスを独自のファイルに移動できます。

コンポーネントと呼ぶ新しいフォルダまたは新しいディレクトリを作成します。

そして、コンポーネントと呼ばれるこのフォルダ内に、丸いボタンになる新しいDartファイルを作成します。そして今、前から作成したクラスを貼り付けて、マテリアルライブラリをインポートして、ステートレスウィジェットとは何かを知ることができます。

コンポーネントフォルダーをインポートできるようになり、ファイルは丸いボタンと呼ばれました。

これで、丸いボタンが認識され、このページのリファクタリングがかなりうまくなりました。

したがって、次にリファクタリングするのは、登録画面とログイン画面です。

ここにもボタンがあることに注意してください。

そのため、この登録ボタンを同じ丸いボタンに置​​き換えます。

コンポーネントフォルダーと丸いボタンファイルをインポートして、これらすべてを置き換えましょう。

青色のアクセントカラーがあり、onPressedには現在何もありません。また、登録のテキストもあります。

それで、それをregisterのタイトル、青いアクセントの色、現在完全に空のonPressedを持つ丸いボタンに置​​き換えましょう。

したがって、ここに空のコールバックを追加するだけで、後で少し処理します。

ログイン画面内に、これらのボタンの1つがここにあります。

そのため、これは空のonPressedとログインを示すタイトルが付いた水色のボタンです。

それを削除して、丸いボタンに置​​き換えましょう。

そしてもちろん、使用するためにはインポートする必要があります。

これでプロパティを追加して、タイトルをログインさせ、色をcolors.lightBlueAccent、onPressedを空のコールバックにすることができます。

これで完了です。リファクタリング。 OK。

そのため、今度はチャレンジのパート2が行われます。

ログイン画面と登録画面を見ると、これらのテキストフィールドがあることがわかります。

これらのテキストフィールドは、入力装飾によって大きくスタイル設定されています。そして、登録画面のテキストフィールドとログイン画面の両方で入力装飾が非常に長い理由は、実際にはかなり多くのものを指定しているためです。

これらの画面の両方に2つのテキストフィールドがあり、1つはユーザーがそのメールを入力し、もう1つはパスワードを入力します。

ただし、フィールドに配置するコンテンツにパディングを追加し、テキストフィールドにアウトラインスタイルの境界線を設定し、境界線のすべての辺の色を変更しています。

また、テキストフィールドがフォーカスされているときと有効になっているときでは、わずかに異なる外観を指定しています。

これは、画面が読み込まれたときにアプリが起動したときの外観であり、実際にアクティブ化されたとき、ユーザーが何かを入力しようとしているときの外観です。

したがって、境界にピクセルを1つ追加するだけで、少し太くなります。

私たちがしているのはそれだけです。

これで、通常、すべてのスタイルを定数ファイルに入れます。

したがって、ボタンテキスト、メッセージテキストフィールド、またはメッセージコンテナの装飾をスタイル設定するための定数を含む定数ファイルが実際にあります。

さて、この挑戦​​のためにあなたにしたいことは、入力装飾を抽出し、それを定数ファイルに入れることです。

それは私たちが他の装飾を持っている場所であり、これは私たちがregistration\_screenを簡素化することを可能にします

.dartおよびlog\_in\_screen.dartファイルはさらにさらに。

繰り返しになりますが、ソリューションを紹介する前に、ビデオを一時停止し、手順を追って説明することをお勧めします。

大丈夫。

ですから、いつものように、定数を作成するときは、繰り返されるスタイリングの一部を取るか、画面コードの外側に配置したいので、ここでこの入力装飾を切り取ります。それを定数ファイルに貼り付けます。

新しい定数を作成し、それをkTextFieldDecorationと呼び、コピーした入力装飾に等しくなるように設定し、コンマをセミコロンに変更します。

そしてhintTextについては、すべてのテキストフィールドにメールを入力するように言っても意味がありません。

したがって、ヒントテキストを指定しない場合は、そのhintTextを空白のままにするか、「値を入力する」だけにして、それがデフォルトのテキストになるようにします。

さて、登録画面に戻って、テキストフィールドの装飾をそのkTextFieldDecorationに設定すると、もちろんこれを行うには定数ファイルをインポートする必要があります。

そして今、私たちが持っているのは、そのデフォルト行を持つ沼地の標準テキストフィールドです。値を入力してください。

それでは、この問題をどのように解決しますか？

1つのプロパティだけでなく、基本的に同じ入力装飾ウィジェットを持つテキストフィールドをどのように作成できますか？

以前にテーマを調べたときに、Flutterに付属している特定のテーマ（ダークテーマなど）を使用できたことを思い出してください。次に、このメソッドcopyWithを使用して、「このテーマ全体をコピーし、この1つを変更します」と言います。

入力デコレーションでも同じことができます。テキストフィールドの装飾をそのまま使用する代わりに、kTextFieldDecoration.copyWithと言うことができます。この大量の変更から変更したいのは、単にhintTextです。

したがって、この上部のテキストフィールドでは、ヒントテキストは「メールを入力してください」であり、下部のテキストフィールドでは装飾はkTextFieldDecorationになり、copyWithを使用して「パスワードを入力してください」というhintTextを指定します。これで、丸みを帯びた境界線など、各テキストフィールドの外観を定義する同じ定数を使用できますが、実際には、ヒントテキストを変更する全体に1つの変更を指定しています。そして、ログインと登録ページ全体でそれを行うことができます。

したがって、これを定数ファイルから取得したものに置き換えます。

そして、これもここで置き換える必要のある最後のものです。

これで、実際にこれを削除して、テキストをすべて白にすることを覚えている暗いテーマを削除できます。

そのため、ログインまたは登録ページを表示するときに、実際にはhintTextは表示されません。

ただし、この暗いテーマを削除してアプリを実行すると、明るいテーマではテキストが黒で表示されますが、テキストフィールドのテキストはこの灰色で表示されます。両方のテキストフィールドが同じ装飾を使用している場合でも、装飾がない場合は、これがどのように見えるかがわかります。

それでは、それを切り取り、保存をクリックしてみましょう。

また、登録画面には、ここにあるカスタムフィールドではなく、沼地の標準的なテキストフィールドが表示されます。

しかし、先に進んで保存すると、それが素敵な丸いアウトラインテキストフィールドに変わり、クリックしたりタップしたりするときに境界線の幅を変更することでハイライトを見ることができますその上。私たちが解決した最後の1つは、アプリの最初にその暗いテーマを取り除き、ボタンが明るいテーマになったためです。

そのため、それらのテキストは黒です。

コードをリファクタリングしたため、すべてのボタンでこれを変更するために、ウェルカム画面、ログイン画面、そのコードが置かれている場所、これらのボタンのすべてをレンダリングするコードがどこにあるかを正確に知っていますrounded\_button.dartファイルの場合、この1か所でテキストの色を簡単に変更できます。

そのため、そのテキストウィジェット内に、テキストの色をcolors.whiteに変更するTextStyleになるスタイルプロパティを追加します。

ここで変更すると、そのボタンを使用するすべての場所に渡り、長いビルドメソッドや画面上のコードを介して検索することに煩わされる必要がなくなります。

したがって、リファクタリングのいくつかの直接的な利点があります。Flutterアプリの作成中およびコーディング中に、定期的にリファクタリングを行うことをお勧めします。蒸気を使い果たしたときに最後まですべてを残すよりもはるかに簡単になり、おそらくリファクタリングを行う時間がなく、最終的にはスパゲッティコードのようなものになります。

コードをリファクタリングし、登録画面とログイン画面を非常にシンプルで非常に簡単に見えるようになったので、Firebaseの実装と、ユーザーが送信するすべてのメッセージのバックエンドデータベースの追加を開始する準備ができましたFlutterアプリに作成されます。

したがって、そのすべてについて、次のレッスンで説明します。