**VICTOR COSTA LIMA**

**3º INFORMÁTICA**

**Desenvolvimento de Sistemas**

Flutter Desktop

**ETEC PAULISTANO**

**2020**

**INTRODUÇÃO**

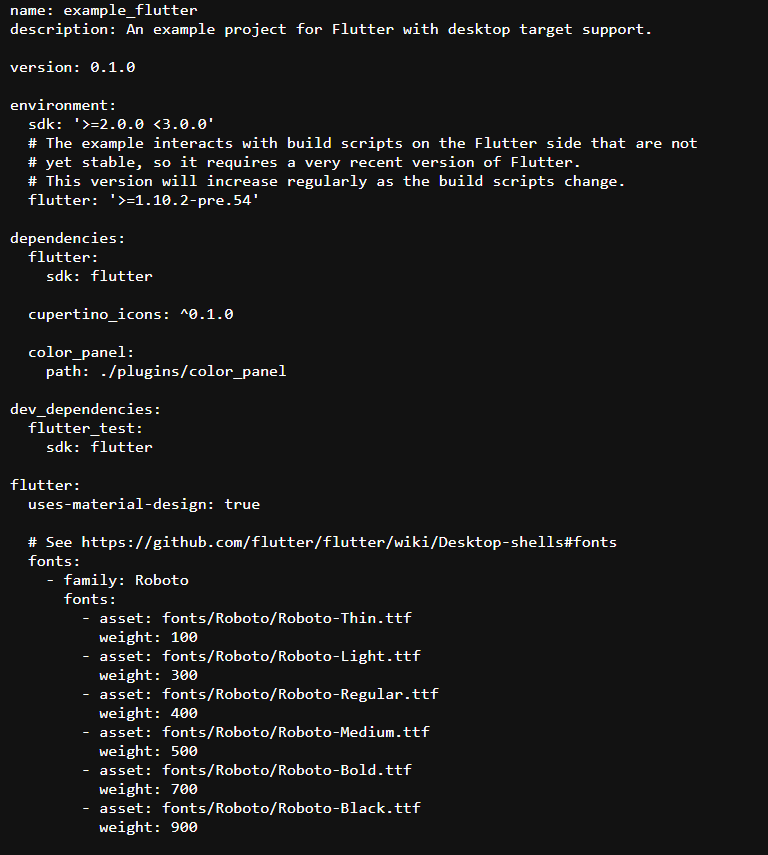
Com base no sucesso da linguagem Dart e do ecossistema Flutter, o suporte ao desktop target oferece novas oportunidades para a criação de ótimos aplicativos de desktop com bases de código claras, legíveis e binárias de alto desempenho.

**SETUP DO PROJETO**

Para criar um aplicativo macOS, basta executar $ flutter create e um projeto será criado. Para criar um aplicativo que também pode suportar Linux e Windows, um projeto inicial com suporte para todos os três ambientes é fornecido pela equipe do Flutter com o projeto de incorporação de desktop flutter. Isso pode ser clonado ou baixado para usar como ponto de partida para novos projetos.

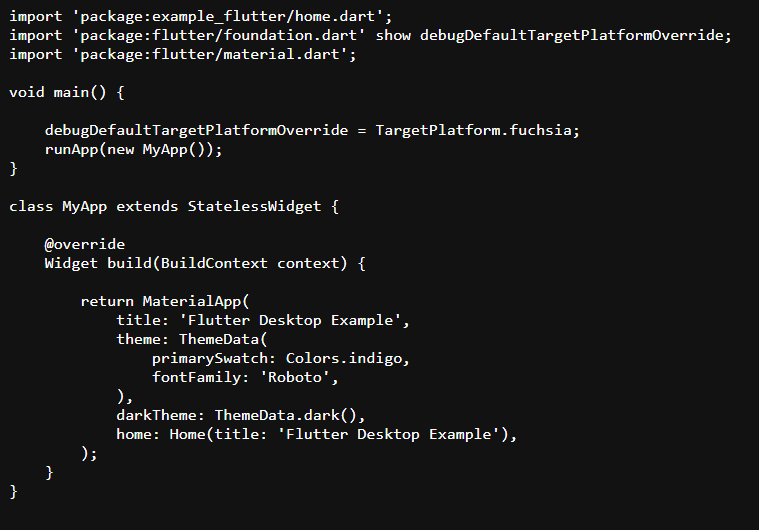
O projeto de exemplo foi criado clonando o repositório de incorporação do desktop flutter e copiando o conteúdo de example/ em um novo projeto.

Vamos dar uma olhada no arquivo de definição do projeto, pubspec.yaml:



**Ponto de Entrada da Aplicação**

O arquivo principal para este aplicativo é o lib / main.dart padrão:

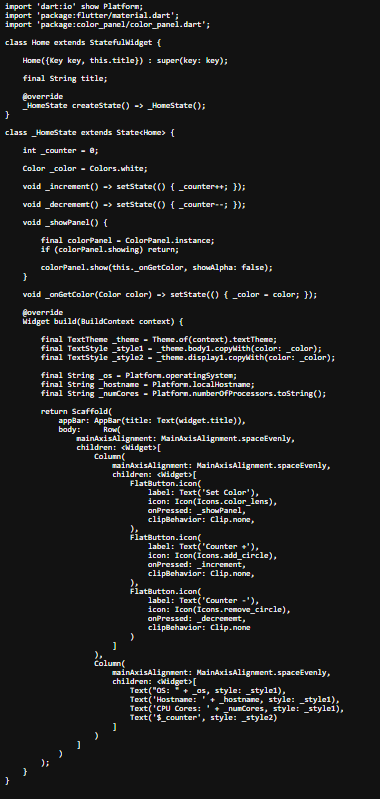


A função main () inicializa o aplicativo e define a plataforma de destino de depuração padrão como fúcsia, que é um sistema operacional desenvolvido no Google que inclui uma IU construída com Dart e Flutter. A classe MyApp é uma implementação básica de StatelessWidget que retorna o contêiner principal para o próprio aplicativo, com um título, tema e widget inicial padrão para qualquer aplicativo Flutter.

Observe a propriedade darkTheme e seu valor ThemeData.dark (), que fornece um tema escuro quando solicitado pelo sistema operacional, como no caso do macOS com tema escuro para todo o sistema habilitado. A seguir, daremos uma olhada no conteúdo da página de exemplo e o que ele faz.

**Home**

A maior parte do conteúdo do aplicativo para esta demonstração simples está localizada em lib / home.dart:

 A classe Home importa alguns pacotes (incluindo o plugin color\_panel) e implementa um StatefulWidget com backing State, para permitir que esta tela armazene e atualize o estado do aplicativo. O Dart e o Flutter fornecem ótimos recursos para gerenciar o estado usando uma variedade de ferramentas e padrões, o mais simples deles sendo um widget com estado como o mostrado neste arquivo.

As propriedades gerenciadas no estado deste widget incluem uma cor e um contador. Os métodos \_increment e \_decrement usam o método setState para incrementar ou decrementar o contador. O método \_showPanel solicita uma instância do ColorPanel do plug-in importado e, em seguida, chama show nele, que solicitará um seletor de cor nativo do sistema operacional. Quando uma cor é escolhida, setState é usado para atualizar o estado do widget com uma nova cor, que o colocará na fila para ser redesenhado pelo mecanismo Flutter.

O método de construção para este widget configura dados de tema e estilo e solicita o nome do SO, o nome do host local e o número de núcleos de processador da classe Platform. Eles podem ser inicializados diretamente nas propriedades designadas como finais, ou mesmo colocados em um arquivo externo e usados ​​como um serviço, mas para este aplicativo de demonstração, eles são variáveis ​​locais colocadas no método de construção do widget para simplificar.

O widget retornado pelo método build é um Scaffold, com um AppBar e um corpo composto por uma linha horizontal de duas colunas. A primeira coluna (esquerda) contém alguns botões que são mapeados para os métodos de incremento, decremento e seleção de cores descritos acima. A segunda coluna exibe as informações da máquina local coletadas anteriormente junto com o contador, na cor selecionada pelo usuário (ou padrão se nenhuma ainda não tiver sido selecionada).