TUTilityの構造

- model
- provider
- router
- page
- widget

model

- データの形式(構造体，クラス)がまとめられている
  - Subject
  - Timetable
  - ...
- JSON(構造化したデータをテキストでやりとりするための形式)に変換するための関数を自動生成している(freezedパッケージ)

provider

- アプリ全体で共有したいデータ(状態)がまとめられている
  - 例えば，Timetableクラスはアプリのあらゆる場所から読み書きしたい
- riverpodというライブラリを使用している

router

- アプリ内のページ(画面)をどのようにルーティングするか書いてある
  - 時間割，食堂，リンク，その他
  - 時間割取得ページは独立させている

page

- routerによって切り替えられる，各画面のレイアウト
- 後述するWidgetを配置している

widget

- FlutterにおけるWidgetをまとめている
  - Dialog
  - Timetableの各種Tile
  - ...

アプリ起動の流れ

1. Flutterのエンジンを初期化
2. 端末画面の向きを縦に固定(iPadには適用されない)
3. 各種Providerの準備
4. Shared Preferences(アプリ固有の設定を保持する機構)の準備
5. AppRouterがNavigationPageとTimetablePageにルーティング
6. TimetablePageにTimetableNotifierProviderから時間割データが供給されて描画される

Dart

- JavaScript, Java, C#みたいな見た目の言語
- Google製
- 主にFlutterで使われるが，汎用プログラミング言語なので一般的なプログラム(コマンドラインで動作するアプリ等)も作成可能
- Dart VM上で動かすこともできるし，機械語にコンパイルすることも可能

Flutter

- 主にスマートフォン向けアプリを作るためのフレームワーク
- 一つのコードで複数のOS向けにビルドできる
- Google製
- Dartで記述する(OS固有の機能を呼び出したい時はSwiftやKotlin, Javaを触る必要がある)

Flutterでクロスプラットフォームを実現する仕組み

- Framework: アプリ開発者が触る層，Dart
- Engine: Dartのランタイム実行，EmbedderとFrameworkを繋げる，C/C++
- Embedder: 描画する画面の準備，各種スレッドのセットアップなど，OS依存