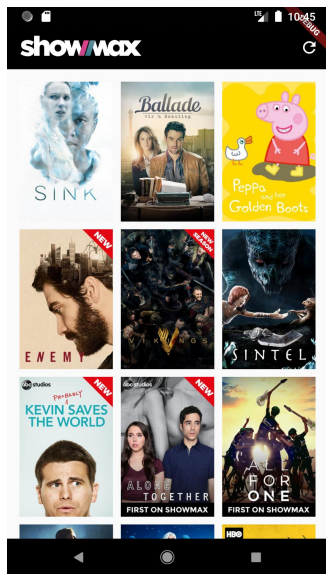


Video on Demand powered by Flutter

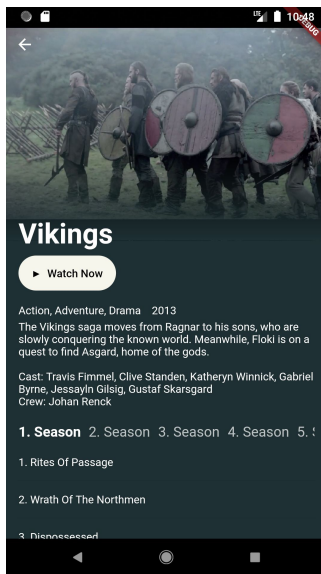
from zero to hero



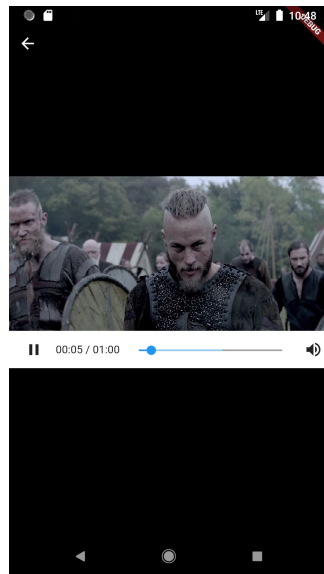
What we want to build



Grid



Detail



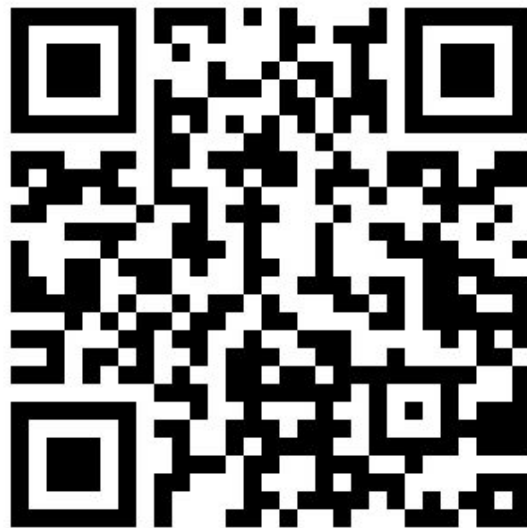
Player

Try it with us

Connect to Wi-Fi `guest.showmax.cz`

Clone this public repository:

<https://github.com/Showmax/flutter-workshop>



Get ready - you should have already done this ;-)

We gonna use Android Studio in this workshop as a reference

Complete 1-3 steps in <https://flutter.dev/docs/get-started/install>

Make sure all your necessary **flutter doctor** issues are resolved and you are able create new flutter project and run it either on iOS Simulator, Android emulator or real device be it Android / iOS

5 steps to victory

1. Build grid UI (stateless)
2. Build grid UI (stateful)
3. Load data from API
4. Build Detail and navigate to it
5. Build Player

Dart in a nutshell

What is it?

- Open-source Programming language 🧙
- It's a Google thing
- October 2011 announced
- Released 2013
- Current stable version 2.3.0

Important concepts

- Everything is an object
- It's strongly typed
- Supports generics
- Public/private concept with underscore
- Can do both JIT and AOT compiling
- It's easy to learn ??

Declaring variables/constant

	Dart	Kotlin	Swift
Implicitly	<code>var someInt = 5;</code>	<code>var someInt = 5</code>	<code>var someInt = 5</code>
Explicitly	<code>String name = 'Stan';</code>	<code>var name: String = "Stan"</code>	<code>var name: String = "Stan"</code>
Dynamic type	<code>dynamic name = 'Stan';</code>	<code>var name: dynamic = "Stan"</code>	Not available
Constants	<code>final name = 'Stan';</code> <code>const name = 'Stan';</code>	<code>val name = "Stan"</code>	<code>let name = "Stan"</code>

Nullability/Optionals

	Dart	Kotlin	Swift
Declaration	<pre>int id = null; id.abs(); // Exception id?.abs(); // Safe</pre>	<pre>var id: Int? = null id.inc(); //Compile error id!!.inc(); // Exception id?.inc(); // Safe</pre>	<pre>var id: Int? = nil id.signum(); //Compile error id!.signum(); // Exception id?.signum(); // Safe</pre>
Nil coalescing operator / Elvis operator	<pre>int id = null; int else = id ?? -1;</pre>	<pre>var id: Int? = null var else: Int = id ?: -1</pre>	<pre>var id: Int? = nil var else: Int = id ?? -1</pre>

Classes, protocols and inheritance

	Dart	Kotlin	Swift
Inheritance	<code>class Dog extends Mammal { }</code>	<code>class Dog: Mammal { }</code>	<code>class Dog: Mammal { }</code>
Data class / Struct	Not available	<code>data class A(var a:Int) { }</code>	<code>struct A { }</code>
Protocol / Interface	Not available	<code>interface Countable { fun count(): Int }</code>	<code>protocol Countable { func count() -> Int }</code>

Notable obstacles

- Braces everywhere
- Semicolons
- No shorthand argument names in closures
- No typealiases
- No interfaces
- Hello old C-style loop?

Step 1

Build our Grid UI

Intro to widgets

- In Flutter everything is Widget
- When unsure how to do something, then it's probably Widget
- Your screen/page to which you navigate is widget as well

```
Padding(  
  
  padding: EdgeInsets.all(16.0),  
  
  child: Text('Hello World!')  
  
)
```

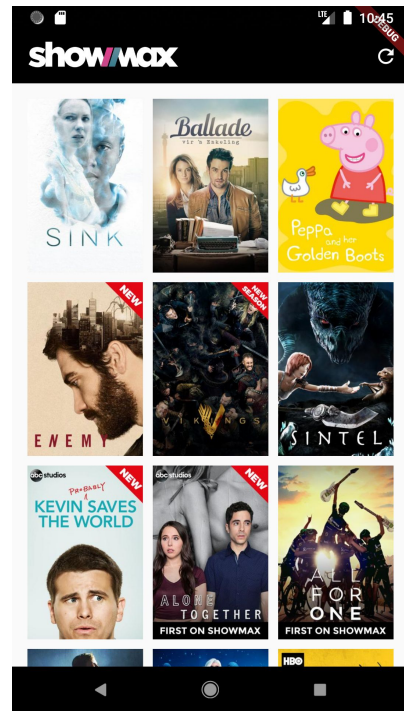
Stateless vs. Stateful widget

Your widgets will be commonly subclasses of:

- Stateful - manages state
- Stateless - doesn't have state

What we want to build in step1

Scaffold - top level widget of our first “screen”,
Provides useful appBar / body distinction how
build our UI



AppBar

showmax



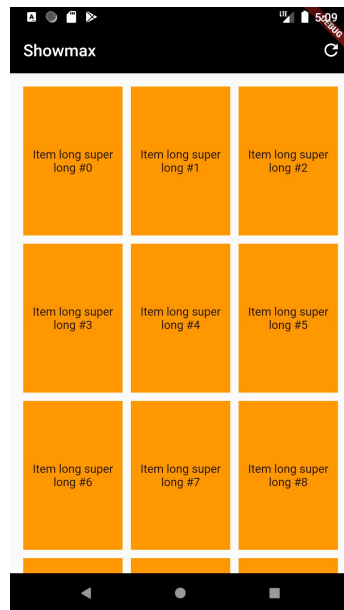
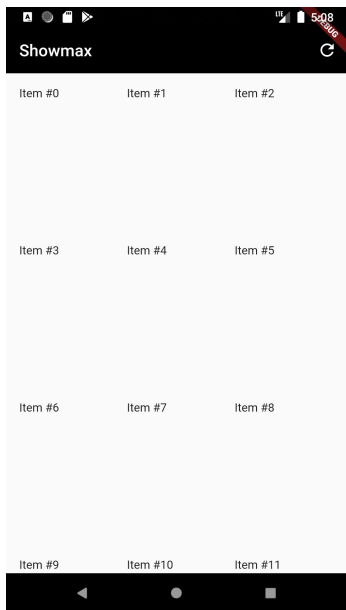
```
AppBar: new AppBar(  
  title: new Image.asset(  
    'assets/showmax_white_logo.png',  
    fit: BoxFit.cover,  
    height: 30.0),  
  elevation: 0,  
  backgroundColor: Colors.black,  
  textTheme: Theme.of(context).textTheme.apply(bodyColor: Colors.white),  
  actions: <Widget>[  
    new IconButton(  
      icon: const Icon(Icons.refresh),  
      onPressed: () {  
        // _viewModel.load();  
      })  
  ]  
)
```

TODO1: enable logo in appBar

1. Replace Text Widget in HomeScreen.dart with Image widget
2. Don't forget to uncomment in pubspec.yaml, otherwise it won't be available to flutter app (won't be packaged)

NOTE: pubspec.yaml - manages your dependencies

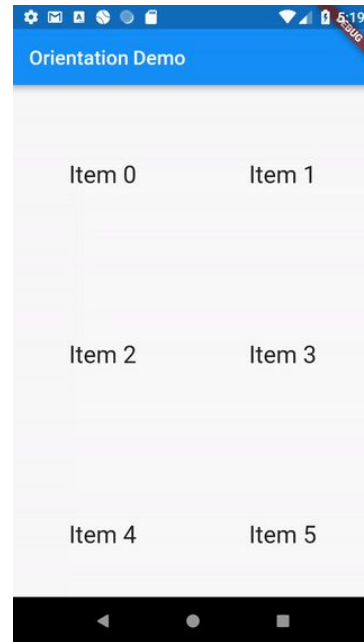
TODO2: generate GridView items



TODO3: make GridView react to orientation

- Remember: EVERYTHING is a Widget ;-)
- Wrap GridView.count in OrientationBuilder:

```
OrientationBuilder(builder: (context, orientation) {  
    return widgetAccordingToOrientation(orientation)  
})
```

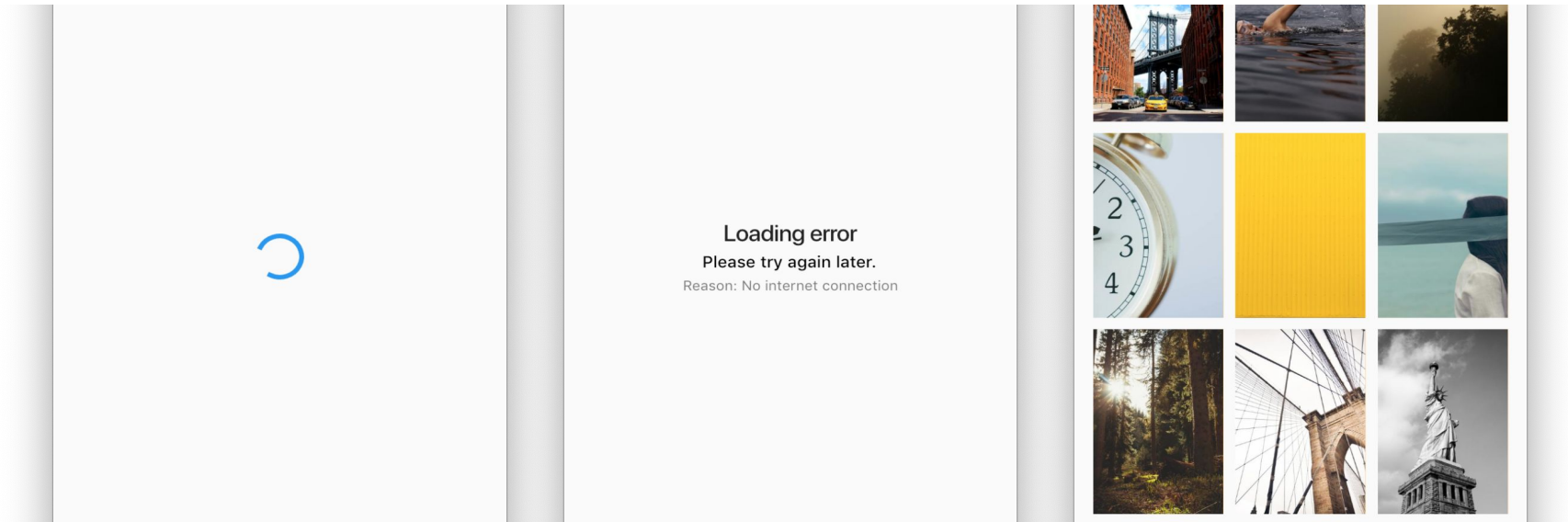


Step 2

Stateful home screen

Goal

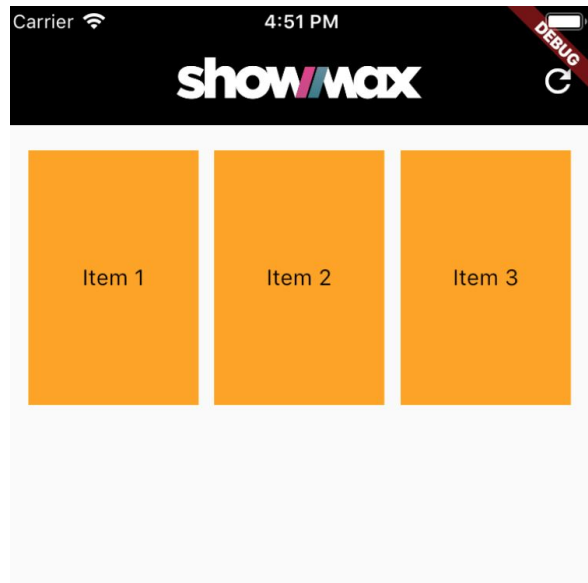
- Show loading, error and loaded state on home screen



TODO #1: Try simple loaded state

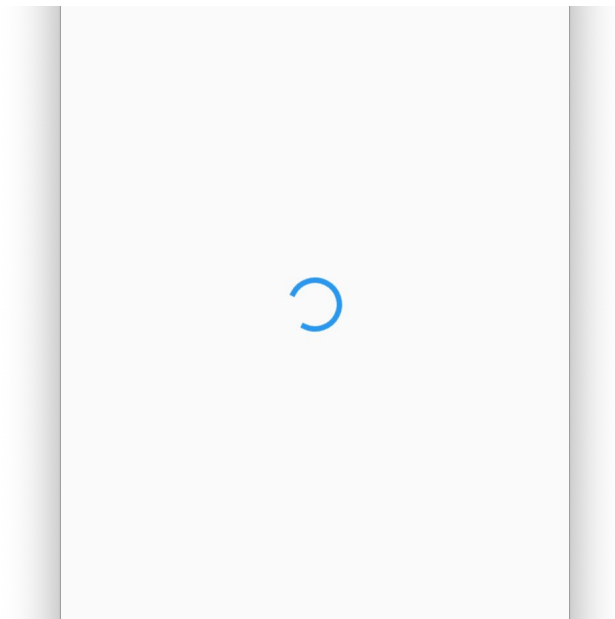
Open HomeScreen.dart:

```
Widget _body() {  
  return _loaded([  
    HomeItem(title: "Item 1", imageLink: "...", imageColor:  
    HomeItem(title: "Item 2", imageLink: "...", imageColor:  
    HomeItem(title: "Item 3", imageLink: "...", imageColor:  
  ]);  
}
```



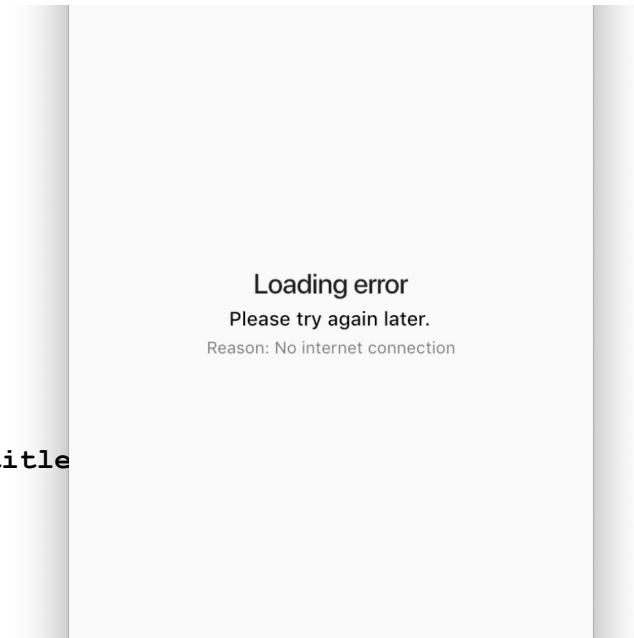
TODO #1: Implement loading state

```
Widget _body() {  
  return _waiting();  
}  
  
Widget _waiting() {  
  return Center(child: CircularProgressIndicator());  
}
```



TODO #1: Implement error state

```
Widget _body() {  
  return _error("No internet connection");  
}  
  
Widget _error(Object error) {  
  final textTheme = Theme.of(context).textTheme;  
  return Center(  
    child: Column(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: [  
        Text("Loading error", style: textTheme.title),  
        SizedBox(height: 5),  
        Text("Please try again later.", style: textTheme.subtitle),  
        SizedBox(height: 5),  
        Text("Reason: $error", style: textTheme.caption)  
      ],  
    ),  
  );  
}
```



Loading error
Please try again later.
Reason: No internet connection

TODO #2: View model specify state of screen

Output content for home screen

```
Stream<List<HomeItem>> items;
```

Stream produces sequence of events

- Data ... List<HomeItem>
- Error

Backed by StreamController (where you fill values/errors)

Handle user input: **void load()** by requesting content

TODO #3: HomeScreen is now Stateful

- Access to life cycle
 - load view model when widget initialized
- Keep instance during updates

```
class HomeScreenState extends State<HomeScreen> {  
  HomeViewModel _viewModel;  
  
  @override  
  void initState() {  
    super.initState();  
    _viewModel = HomeViewModel();  
    _viewModel.load();  
  }  
}
```

TODO #4: Listen on stream of items

Basic way

- Subscribe to stream
- Remember latest value
- Call setState to update UI

```
List<HomeItem> _data;

@Override
void initState() {
    super.initState();
    _viewModel = HomeViewModel();
    _viewModel.load();
    _viewModel.items.listen((data) {
        setState(() {
            _data = data;
        });
    });
}
```

TODO #4: Listen on stream of items

Update UI with
remembered value

Downside:

Need to remember
latest value somewhere

```
List<HomeItem> _data;  
  
Widget _body() {  
  print("Building body");  
  if (_data == null) {  
    return CircularProgressIndicator();  
  }  
  return _loaded(_data);  
}
```

TODO #4: Listen on stream of items

StreamBuilder widget

- Listens on given stream
- On each stream update calls builder to modify subwidgets
- ConnectionState

```
StreamBuilder<List<HomeItem>>(  
  stream: _viewModel.items,  
  builder: (  
    BuildContext context,  
    AsyncSnapshot<List<HomeItem>> snapshot) {  
    if (snapshot.hasError) {  
      return _error(snapshot.error);  
    }  
    switch (snapshot.connectionState) {  
      case ConnectionState.none:  
      case ConnectionState.waiting:  
        return _waiting();  
      case ConnectionState.active:  
      case ConnectionState.done:  
        return _loaded(snapshot.data);  
    }  
  },  
):
```

TODO #5: Try producing error in Stream

Open HomeViewModel.dart

Fill StreamController _items
with error

```
StreamController<List<HomeItem>> _items;
```

```
void load() {  
  _items.addError("Wrong JSON format");  
}
```



Loading error

Please try again later.

Reason: No internet connection

TODO #6: Add library for caching images

Open pubspec.yaml:

```
dependencies:
```

```
  flutter:
```

```
    sdk: flutter
```

```
http: ^0.12.0+2
```

```
cached_network_image: ^0.7.0
```

\$ flutter packages get



TODO #7: Add image to image widget

Open HomeScreen.dart

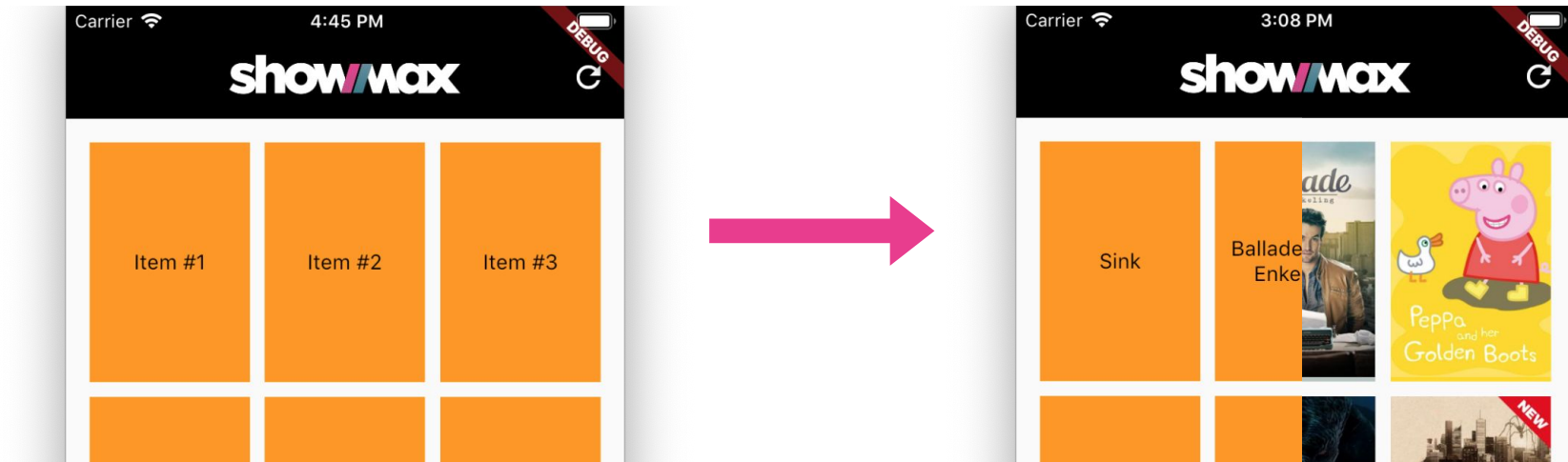
```
Widget _image(String path) {  
    return CachedNetworkImage(  
        fit: BoxFit.cover,  
        imageUrl: path,  
    );  
}
```

Step 3

Load movies from backend API

Goal

- Load movies from backend
- Show movie's title in home screen cell



Backend API

Load most popular movies

```
$ curl -X GET https://api.showmax.com/v115.0/ios/catalogue/grossing
```

```
{ ▼ 4 properties, 128 KB
  "count": 60,
  "items": [ ▼ 60 items, 128 KB
    { ▼ 5 properties, 2 KB
      "id": "7ef437f1-7d3f-4962-939a-e64078408fee",
      "images": [ ▶ 6 items, 1 KB { 7 properties, 307 bytes }, { 7 properties, 307 bytes }, { 7 properties, 291 bytes }, { 7 properties, 291 bytes }, { 7 properties, 291 bytes }, { 7 properties, 291 bytes } ],
      "metadata_direction": "ltr",
      "metadata_language": { ▼ 2 properties, 41 bytes
        "iso_639_3": "afr",
        "name": "Afrikaans"
      },
      "title": "Sink"
    },
    { ▼ 5 properties, 2 KB
      "id": "c0b04d06-e269-406a-9da2-413cfd1ac9f5",
      "images": [ ▶ 6 items, 1 KB { 7 properties, 291 bytes }, { 7 properties, 294 bytes }, { 7 properties, 307 bytes }, { 7 properties, 307 bytes }, { 7 properties, 307 bytes }, { 7 properties, 307 bytes } ],
```

TODO #1: Load JSON via HomeLoader

```
_viewModel = HomeViewModel (HomeLoader (API ())) ;
```

HomeLoader class

- Responsible for loading home items (popular movies)

API class

- Determines base path of endpoint (differs by platform)

Run app and see printed JSON in debug console.

TODO #2: Add library for parsing/validating JSON

Reduces boilerplate code

- when parsing JSON
- when validating properties (required, non null)

https://pub.dev/packages/json_annotation

```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^0.12.0+2  
  cached_network_image: ^0.7.0  
  json_annotation: ^2.0.0
```



```
$ flutter packages get
```

TODO #3: Parse JSON to Asset class

1. Open lib/api/entities/Asset.dart and implement properties

Id, Title, Images, Videos

2. Error when missing required field

```
@JsonKey(required: true, disallowNullValue: true)
```

3. Default value for empty fields

```
@JsonKey(defaultValue: [])
```

4. Generate boilerplate code

```
$ flutter packages pub run build_runner build
```

5. Include generated code to Asset.dart

```
part 'Asset.g.dart';
```

TODO #4: Fill stream in view model

Fill either error or movies data to show.

```
Future load() async {  
    try {  
        final popular = await _loader.loadPopular();  
        final homeItems = popular.map(_makeHomeItemFromAsset).toList();  
        _items.add(homeItems);  
    } catch (error) {  
        _items.addError(error);  
    }  
}
```


TODO #4: Map Asset to content showable in view

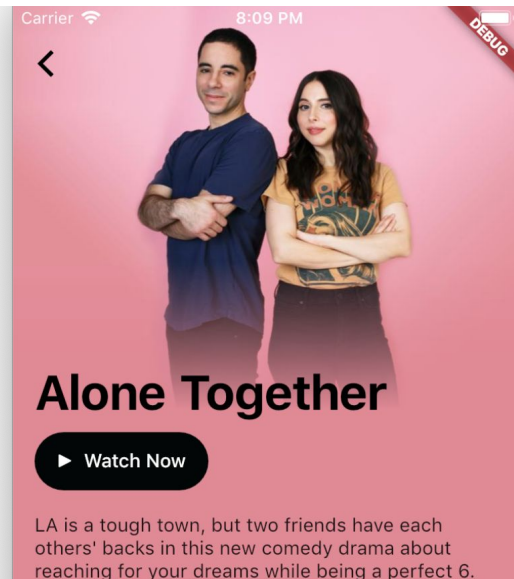
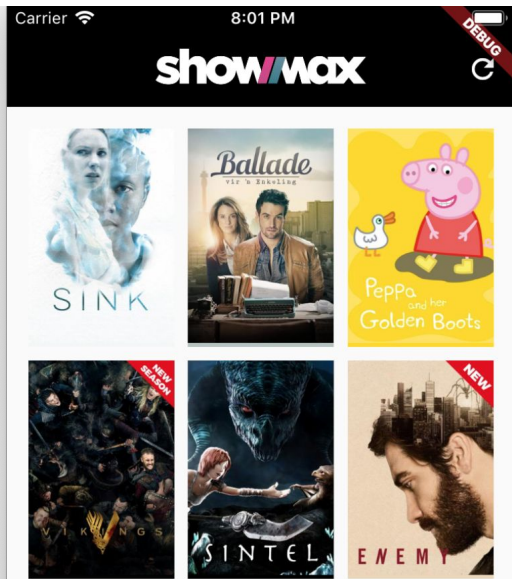
Find suitable image and title.

```
HomeItem _makeHomeItemFromAsset(Asset asset) {  
    final image = asset.images.firstWhere((i) => i.type == 'poster' && i.orientation == 'portrait');  
    if (image == null) {  
        throw "No image found for $asset";  
    }  
    final croppedLink = "${image.link}/300x";  
    return HomeItem(  
        title: asset.title,  
        imageLink: croppedLink,  
        imageColor: image.backgroundColor  
    );  
}
```

Step 4

Navigate to Detail screen

Goal: From Home to Detail



TODO #1: Recognize tap on home item

GestureDetector(

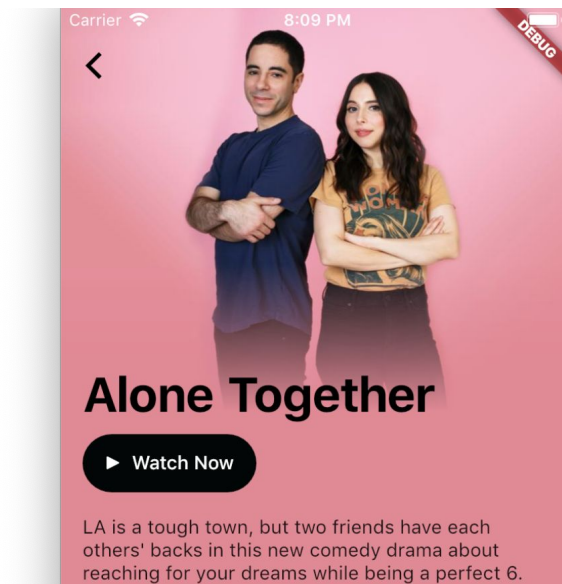
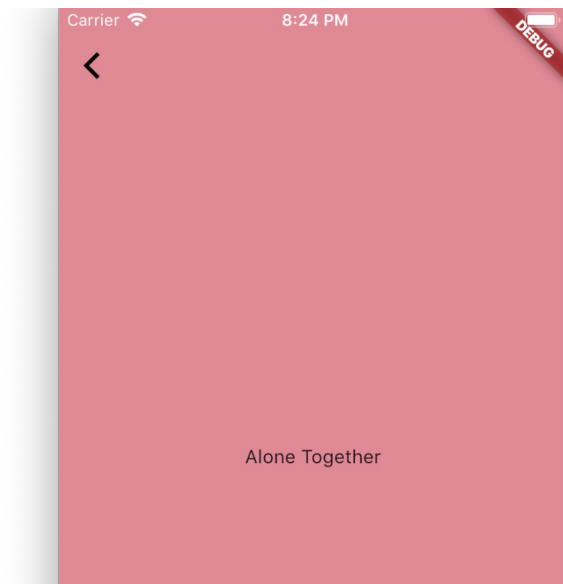
```
child: Container(  
  color: HexColor(item.imageColor),  
  child: Stack(  
    children: [  
      _title(item.title),  
      _image(item.imageLink),  
    ],  
  ),  
,  
,  
onTapUp: (tap) {  
  print("tapped on item");  
},  
)
```

TODO #2: Use Navigator

```
onTapUp: (tap) {  
    Navigator.push(context, MaterialPageRoute(  
        builder: (context) => DetailScreen(item.asset))  
    );  
}
```

TODO #3: Build detail screen

View model is ready. Now implement UI.



TODO #3: Build detail screen

```
Widget _loaded(BuildContext context) {  
  return Stack(children: [  
    ImageWithGradient(  
      link: _detail.imageLink,  
      height: _imageHeight,  
      gradientColor: _detail.backgroundColor  
    )  
  ] );  
}
```

TODO #3: Build detail screen

```
Widget _heading(BuildContext context) {  
  final watchButton = _watchButton(context);  
  return Container(  
    child: Container(  
      child: Column(  
        crossAxisAlignment: CrossAxisAlignment.start,  
        children: [  
          _title(),  
          if (watchButton != null) watchButton, // new dart v2.3 feature  
        ],  
      ),  
    ),  
  );  
}
```


Step 5

Build video player

Video Player ?

- <https://pub.dev/flutter> to the rescue
- Package: video_player
- Package: chewie - provides UI out of the box, uses video_player as its dependency

STEP5: Chewie modified version

- Add dependency to pubspec.yaml
- Normally we would go with vanilla version, but since video_player was detecting data source type based on extension in url (and our urls do not contain such a thing) we needed to use our custom modified version:

```
chewie:  
  git:  
    url: git://github.com/brozikcz/chewie  
    ref: version
```

Platform customization

- You don't want it, but sometimes you cannot avoid it

```
Platform.isAndroid
```

```
Platform.isIOS
```

```
Platform.isFuchsia  
Platform.isLinux  
Platform.isMacOS  
Platform.isWindows
```

How to use chewie

Chewie needs 2 things:

- Instantiate ChewieController, which needs
- Instantiate VideoPlayerController with url of video from video_player underlining library
- You create (1) put it into (2) and create chewie Widget
- Don't forget to dispose then in dispose() of Stateful Widget


Step 5 TODO: Keep instance of chewie running

Take advantage of StatefulWidget in PlayerScreen

Do not instantiate new chewie instance instead keep reference to already created one and supply it to build.

Unit testing

- viewmodel logic tests
- Happy / unhappy path tested
- Mocked dependencies of API loading using Mockito



```
test: ^1.6.1  
mockito: ^4.0.0
```

Flutter Cons/Pros

Cons

- No tvOS support in near future
- Not mature enough yet (video_player customization)
- Not using host system UI widgets (own render engine)
- Missing drag & drop UI builder
- vs React Native: not that much community libraries yet

Pros

- Hot reload, rapid development
- Fast 60FPS UI (because it's own render engine)
- Clear UI updates
- Documentation
- IDE support (VSCode, Android Studio)
- vs React Native: no javascript bridge

Thank You !

- [Flutter bootcamp on AppBrewery](#) (10 usd, discounted by Google)
- Comparisons
 - <https://harveynick.com/2018/05/21/an-ios-developers-opinions-of-flutter/>
 - <https://stxnext.com/blog/2019/02/28/react-native-vs-flutter-comparison/>
 - <https://nevercode.io/blog/flutter-vs-react-native-a-developers-perspective/>
 - <https://hackernoon.com/react-native-vs-flutter-which-is-preferred-for-you-bba108f808>
 - <https://hackr.io/blog/react-native-vs-flutter>
 - <https://medium.com/coding-with-flutter/dart-vs-swift-a-comparison-6491e945dc17>
- Optionals, nullability
 - <https://pub.dev/documentation/meta/latest/meta/required-constant.html>
 - [https://github.com/dart-lang/language/blob/master/working/0110-incremental-sound-nnbd/roadmap.m
d](https://github.com/dart-lang/language/blob/master/working/0110-incremental-sound-nnbd/roadmap.md)