dream

IMPLEMENTATION

http://dream-implementation.com

info@dream-implementation.com

# ListView: Intro

- one of the most common elements in mobile development
- native Android and iOS implementations overly verbose and time consuming (RecyclerView, Adapter, ViewHolder, XML layout)
- Flutter offers simple and easy list implementation with 4 constructors:

1. **ListView**
2. **ListView.builder**
3. **ListView.separated**
4. **ListView.custom**

# ListView

- Takes a list of widgets and makes them scrollable
- Best for small, predetermined number of items
- children added as static list or through map function

```
ListView(
 children: <Widget>[
   ItemOne(),
   ItemTwo(),
   ItemThree(),
 ],
),
```

```
ListView(
 children: items
      .map((item) => listItem(item))
      .toList()
),
```

# ListView.builder()

- best for dynamic lists with undetermined number of items
- list items are constructed lazily (only on-screen items are created)
- exposes build function for list populating logic

```
ListView.builder(
 itemCount: itemCount,
 itemBuilder: (context, position) {
   return listItem(items[position]);
 },
),
```

# ListView.separated()

- enables separator between each list item
- separator can be any widget
- useful when inserting ads into lists

```
ListView.separated(
     itemBuilder: (context, position) {
       return ListItem(items[position]);
     },
     separatorBuilder: (context, position) {
       if (position % 10 == 0) {
           return AdItem()
       }
       return Divider();
),
```

# ListView.custom()

- enables lists with custom functionality
- fine control over children building process
- parameter required for this is a **SliverChildDelegate** which builds the items

1. SliverChildListDelegate - accepts a static list of children
2. SliverChildBuilderDelegate - provides a builder function

# ListView: other properties

- scrollDirection: Axis.vertical / Axis.horizontal
- reverse: true / false
- physics: NeverScrollableSP, BouncingSP (iOS), ClampingSP(Android)
- padding: affects the ListView, not individual list items
- controller: ScrollController()

# ScrollController

- controls the ListView's scroll with jumpTo() or animateTo() methods
- jump to index not supported, requires external plugins
- workaround with item size if items are of equal size

```
final double itemSize = 100.0;
final int index = 25
_controller.jumpTo(itemSize * index)
```

- enables listening to scroll events with addListener() method

```
_controller.addListener((){
  if (_controller.offset > 200){
    // doSomething
  }
});
```

# RefreshIndicator

- swipe-to-refresh functionality out of the box (onRefresh)
- user scrolling callback (notificationPredicate)

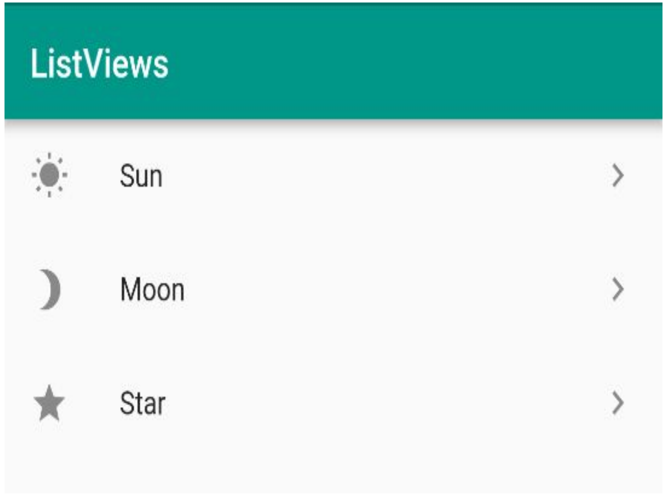
```
RefreshIndicator(
 onRefresh: () async => // refresh data,
 notificationPredicate: (ScrollNotification scrollInfo) {
   if (scrollInfo.metrics.pixels == scrollInfo.metrics.maxScrollExtent) {
          // bottom reached, load more items
   }
   If (scrollInfo.metrics.pixels == scrollInfo.metrics.minScrollExtent {
         // top reached, do something
   }
   return true;
 },
 child: ListView…
);
```

# ListTile

- convenience widget for populating a ListView

```
ListTile(
  leading: Icon(Icons.wb_sunny),
  title: Text('Sun'),
  trailing: Icon(Icons.keyboard_arrow_right),
  onTap: () => doSomething()
),
```

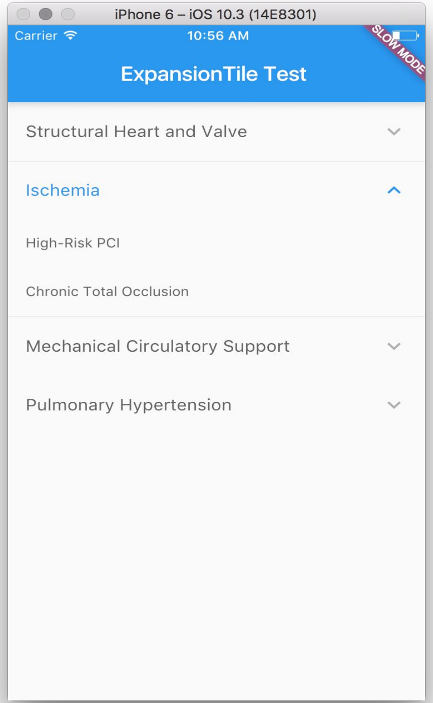
- other parameters: *subtitle, padding, enabled, dense,*



# ExpansionTile

- expandable list tile containing one or more widgets

```
ExpansionTile(
    title: Text(),
    children: <Widget>[
      Text(),
      Text(),
    ]
);
```



Structural Heart and Valve

Ischemia

High-Risk PCI

Chronic Total Occlusion

Mechanical Circulatory Support

Pulmonary Hypertension

## Handling list with BLoC

- Inside StreamBuilder put RefreshIndicator with onRefresh and notificationPredicate defined and ListView as a child

# Handling list with BLoC

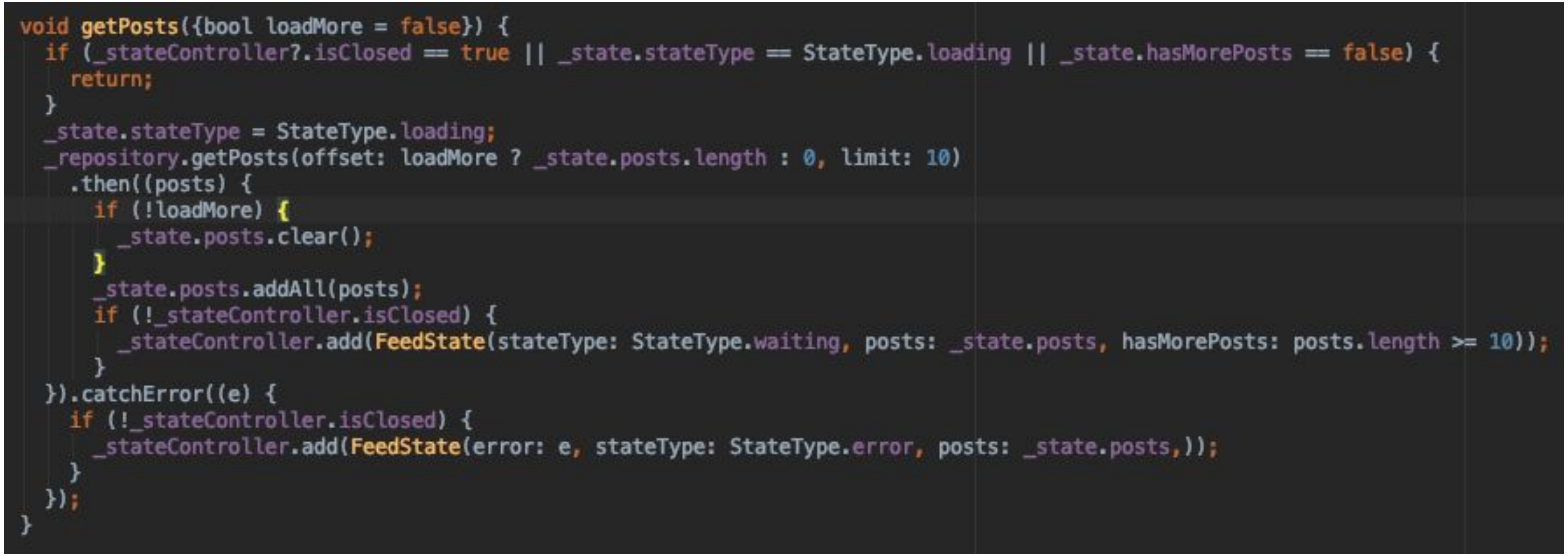
- Public method to fetch list data with loadMore parameter
- Fetch limited number of posts with offset
- Update stream controller with new data or error

# Handling list with BLoC

```
class FeedState extends ScreenState{
  FeedState({this.posts, StateType stateType = StateType.waiting,String
message, dynamic error, StackTrace stackTrace,this.hasMorePosts}) :
super(stateType: stateType, message: message, error: error, stackTrace:
stackTrace);


  List<Post> posts = [];
  bool hasMorePosts = true;
}
```

# Handling list with BLoC



```
void getPosts({bool loadMore = false}) {
  if (_stateController?.isClosed == true || _state.stateType == StateType.loading || _state.hasMorePosts == false) {
    return;
  }
  _state.stateType = StateType.loading;
  _repository.getPosts(offset: loadMore ? _state.posts.length : 0, limit: 10)
    .then((posts) {
      if (!loadMore) {
        _state.posts.clear();
      }
      _state.posts.addAll(posts);
      if (!_stateController.isClosed) {
        _stateController.add(FeedState(stateType: StateType.waiting, posts: _state.posts, hasMorePosts: posts.length >= 10));
      }
  }).catchError((e) {
    if (!_stateController.isClosed) {
      _stateController.add(FeedState(error: e, stateType: StateType.error, posts: _state.posts,));
    }
  });
}
```

# Thank you

Ivan Celija: ivan@dream-implementation.com

Goran Kovač: goran@dream-implementation.com