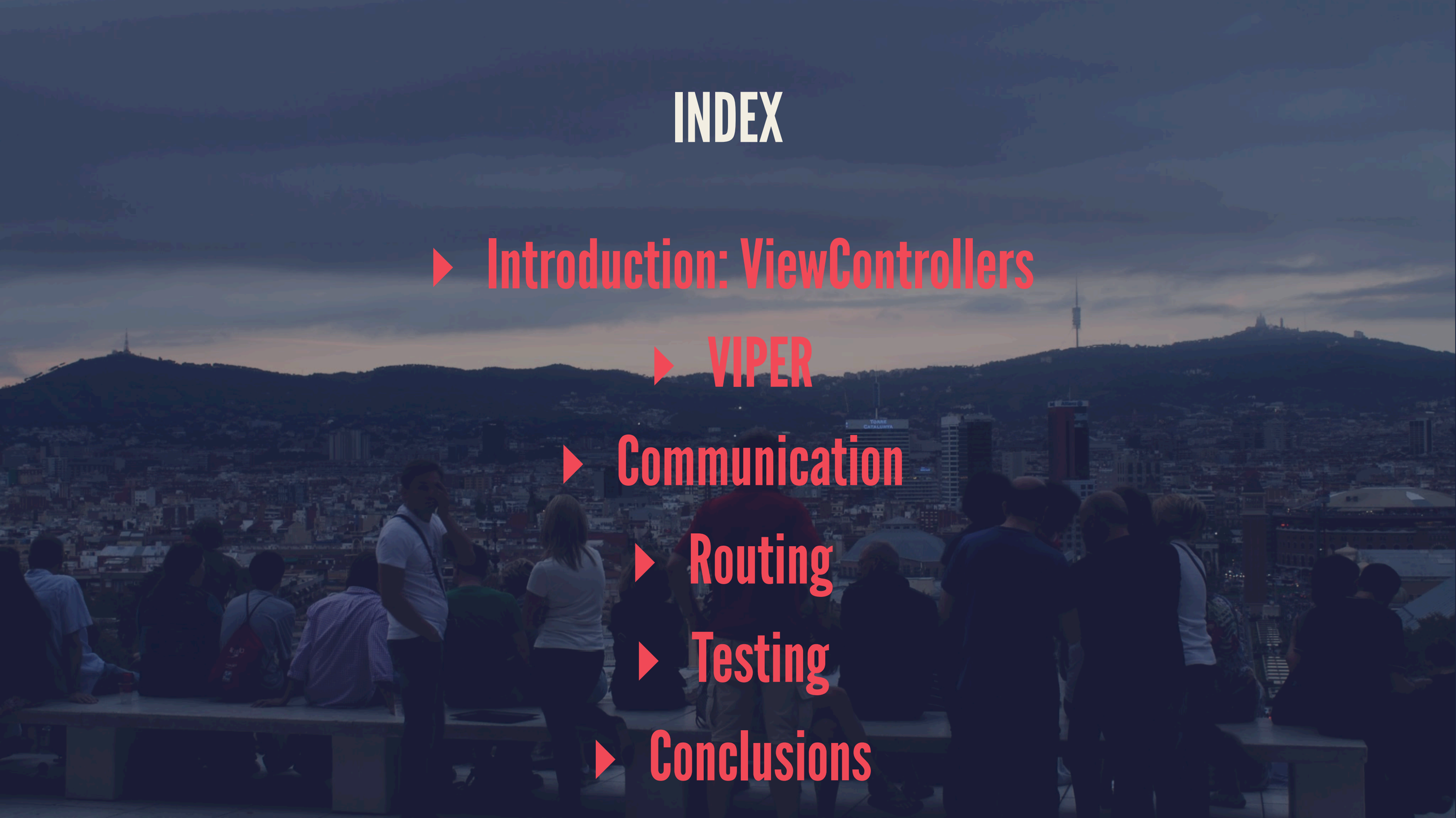
VIPER

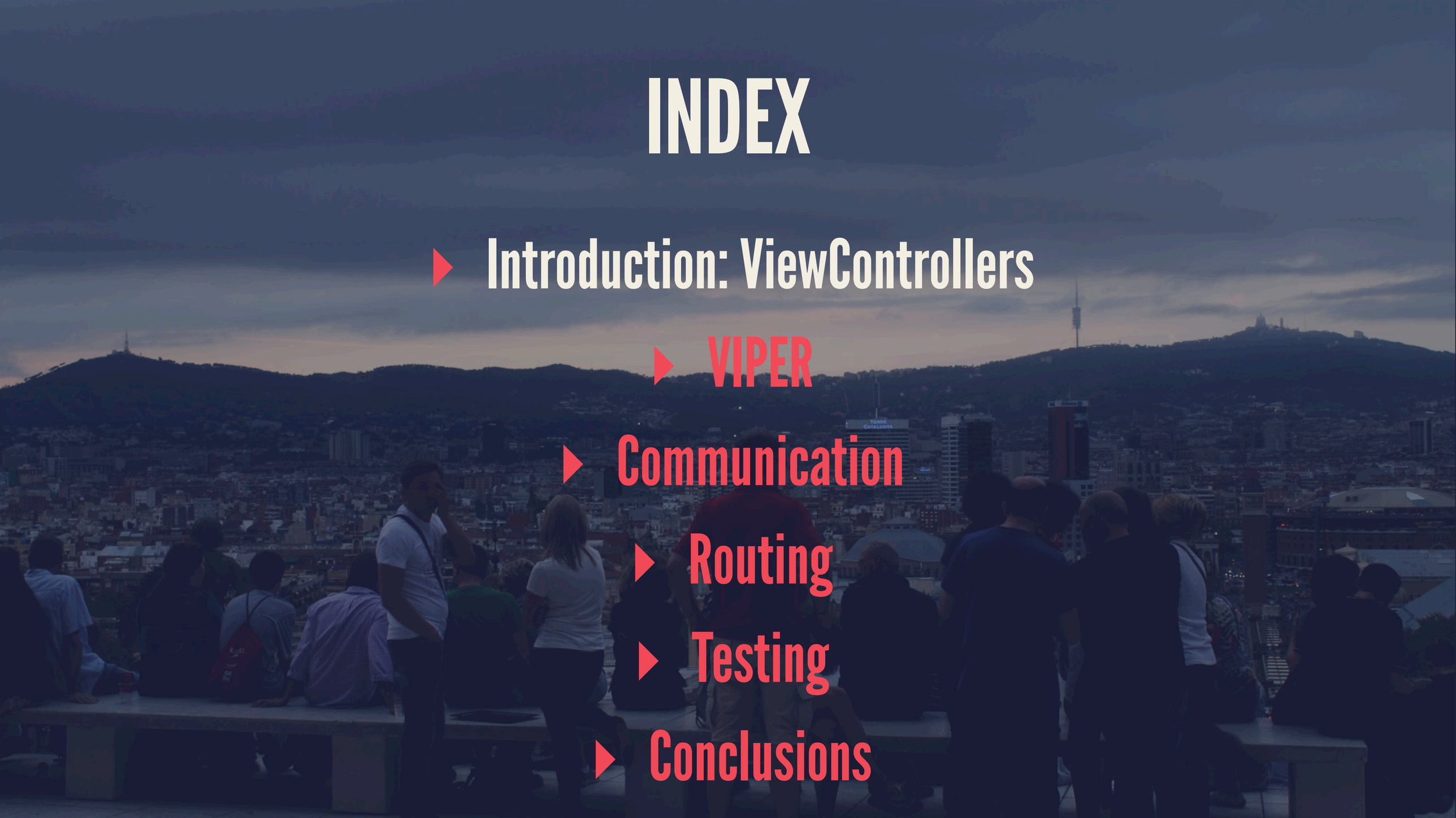
LOOKING FOR THE PERFECT ARCHITECTURE

@PEPIBUMUR



INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions



INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions

DOCUMENTATION, BOOKS, EXMAPLES, TUTORIALS

OBJC.IO, RAYWENDERLICH, IOS DEV WEEKLY, NSHIPSTER

PATTERNS

PROTOCOLS, DELEGATES, DATA SOURCES

KVO


OHMAGIF.COM
abc

Remember

THEY ARE EXAMPLES

CLEAN ARCHITECTURE

INDEPENDENT FROM FRAMEWORKS, UI, DATABASE AND EXTERNAL ENTITIES. EASY TO TEST

The Clean Architecture

VIEWCONTROLLER

A VIEW CONTROLLER COORDINATES ITS EFFORTS WITH MODEL OBJECTS AND OTHER CONTROLLER OBJECTS—INCLUDING OTHER VIEW CONTROLLERS

– Apple

~~ViewController~~Delegator

```
@interface TBThreadDetailViewController : RBViewController <UITableViewDataSource,
 UITableViewDelegate, RBViewControllerURLProtocol, NSFetchedResultsControllerDelegate, TBObjectDetailHeaderCellDelegate,
  TBUploadCellDelegate,TBWatchersViewDelegate>
```

Multi Responsibility

```
[self presentViewController:previewController animated:YES completion:nil]; //Navigation
[Task downloadNewObjectWithID:self.threadIdentifier.remoteIdentifier withSuccess:^{}]; // Networking
NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName:[Comment entityName]]; // Data persistence
[self.view setBackgroundColor:ss_Color_White]; // Formatting
```

Monster files

```
1325
1326 @end
```

- Impossible to test
- Components too coupled
- Hard to understand (and review) classes.

ARCHITECTURE

ARCHITECTURE

WHO WILL *persist* THE DATA?

AND THE *API* INTERACTION?

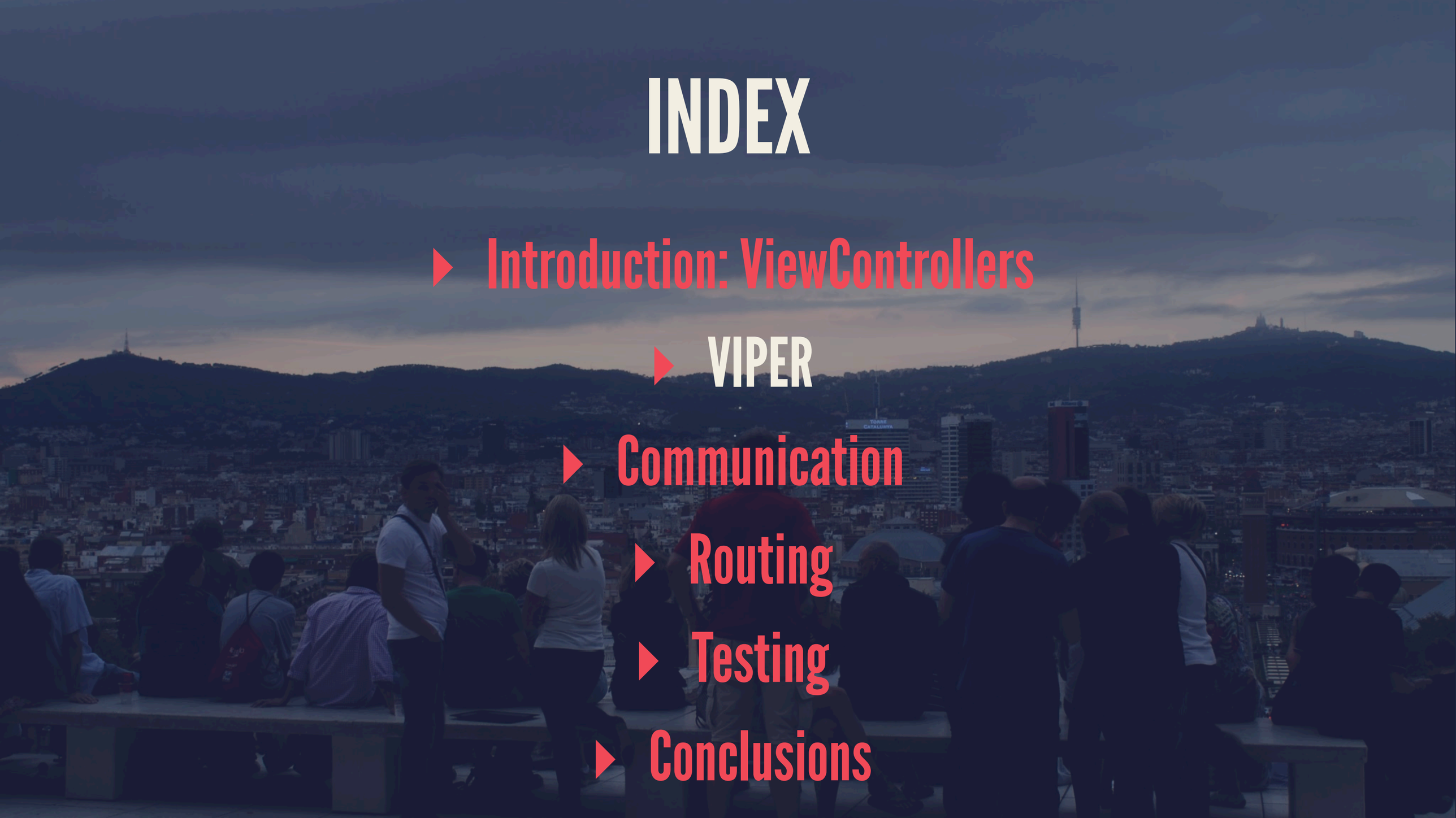
SHOULD THE VIEWCONTROLLER KNOW ABOUT HOW TO NAVIGATE?

ARCHITECTURE

VIPER

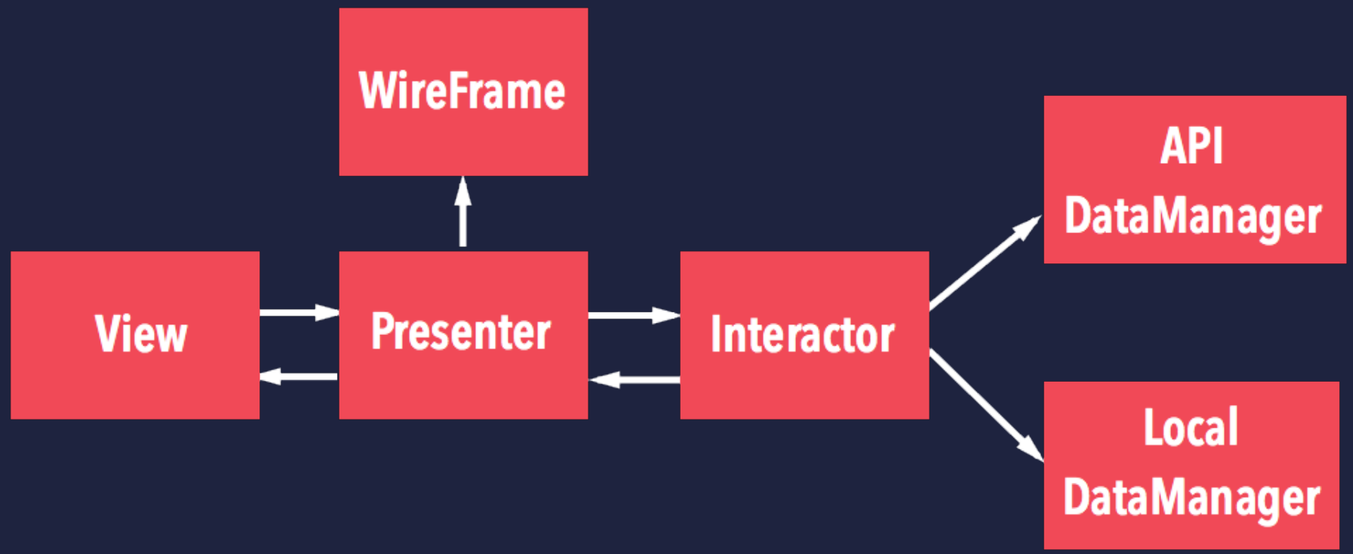
VIEW, INTERACTOR, PRESENTER, ENTITY, AND ROUTING

OBC.IO - VIPER



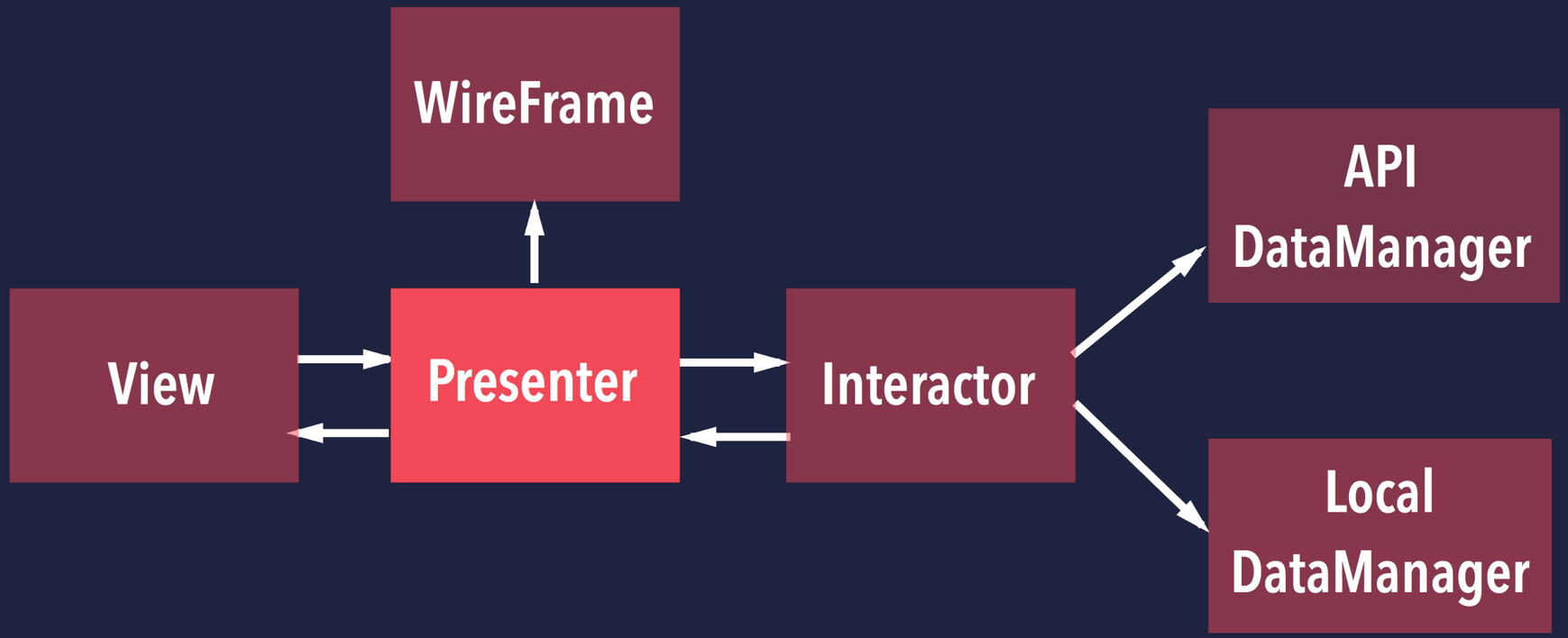
INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions


WireFrame
View
Presenter
Interactor
API
DataManager
Local
DataManager

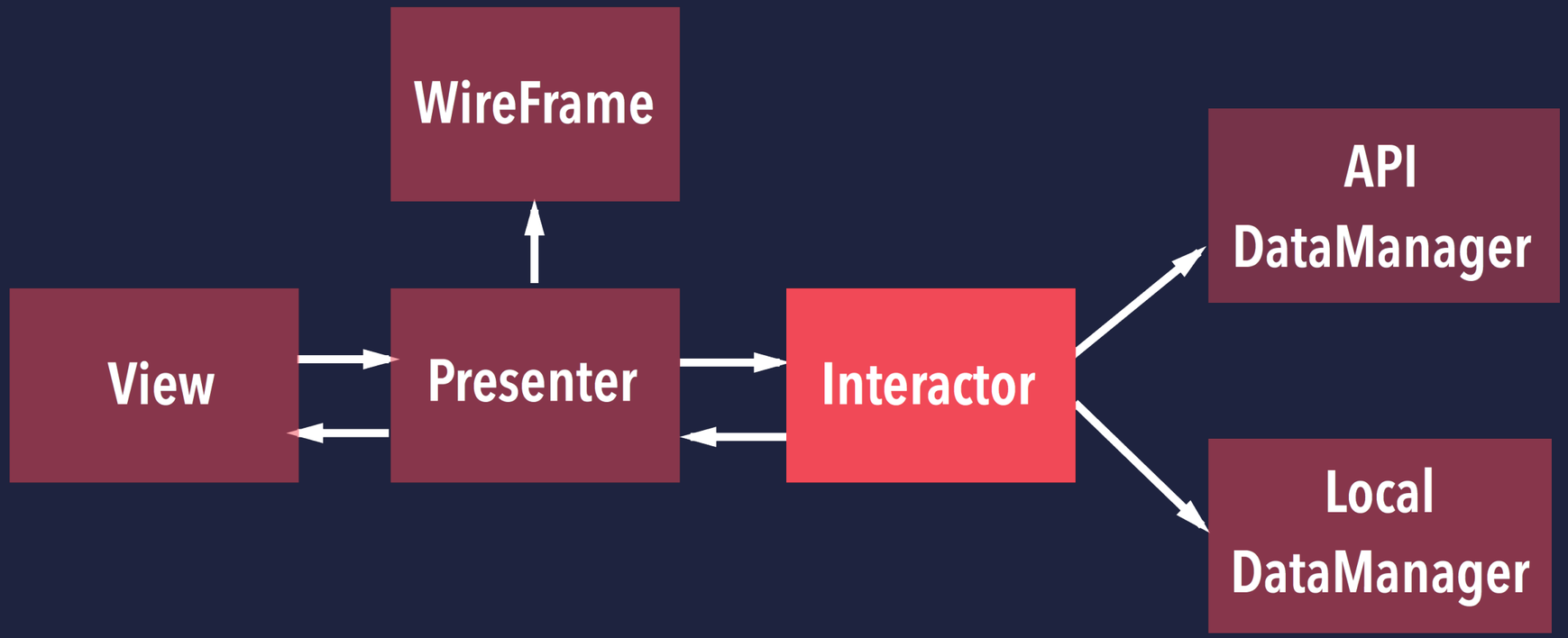
VIEW

- Shows the content that receives passively from the presenter
- Notifies the user's actions to the presenter
- The presenter doesn't know anything about UI


WireFrame
View
Presenter
Interactor
API
DataManager
Local
DataManager

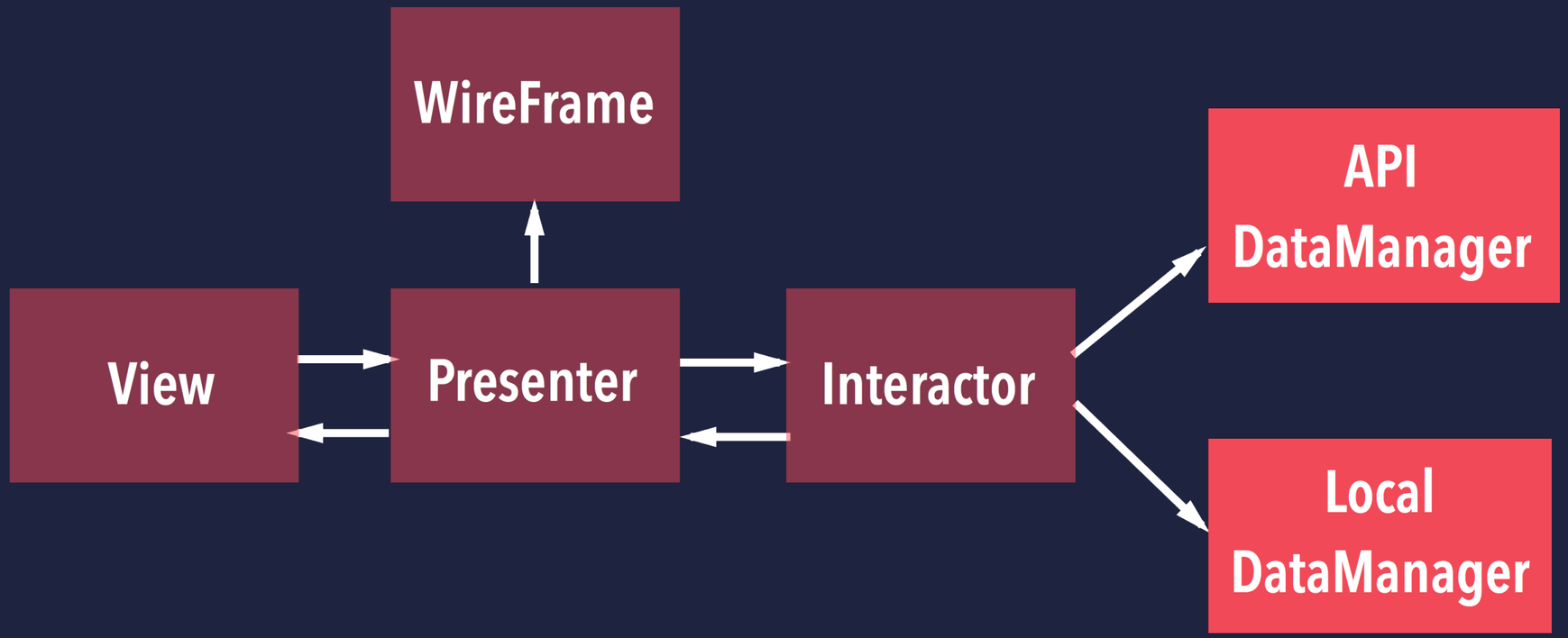
PRESENTER

- Includes the logic to format the view
- Gets the information from the interactor
- Receives actions from the view and traduces them into:
  - Navigation actions (wireframe)
  - Interactor requests


WireFrame
View
Presenter
Interactor
API
DataManager
Local
DataManager

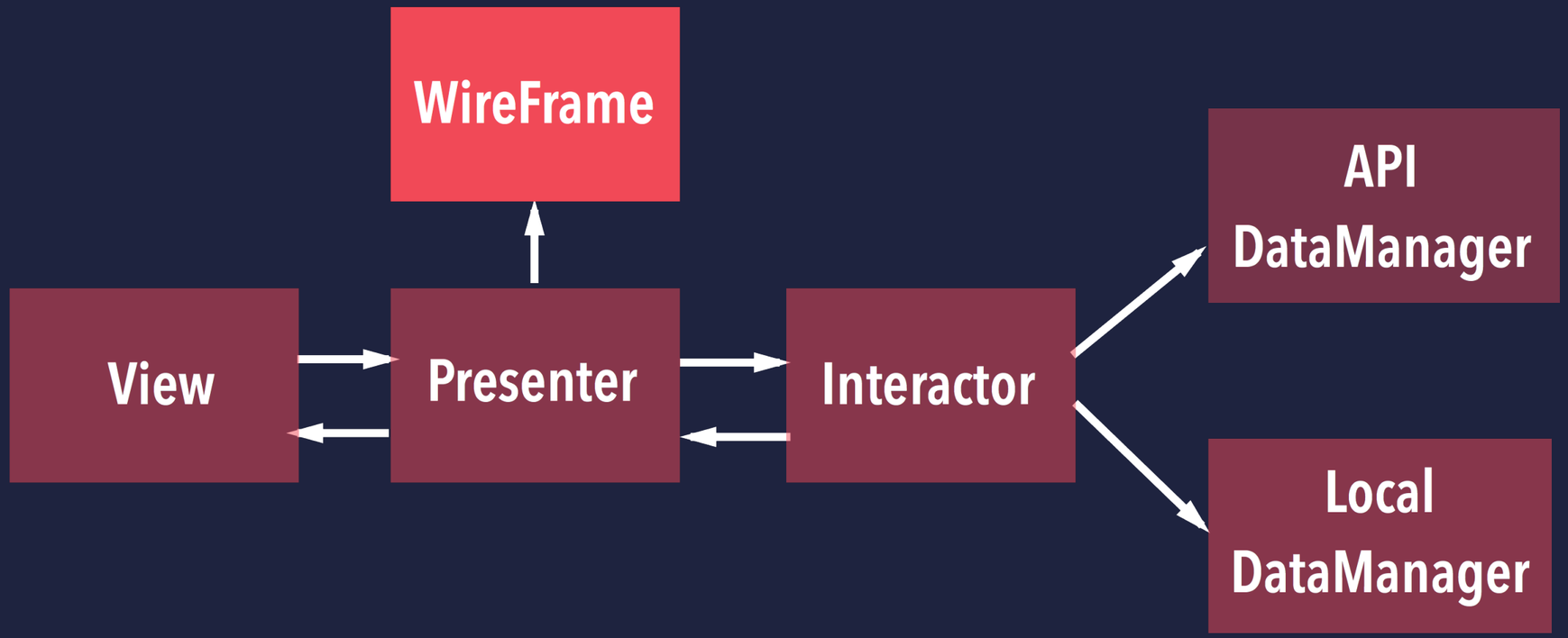
INTERACTOR

- Associated to an unique use case of the application
- Works with PONSO entities
- It doesn't know anything about UI
- Easy to develop using TDD


WireFrame
View
Presenter
Interactor
API
DataManager
Local
DataManager

DATAMANAGER

- Provider of entities for the Interactor
- Responsible of the persistence (Web and Local)
- The entities don't know about how to persist themselves


WireFrame
View
Presenter
Interactor
API
DataManager
Local
DataManager

WIREFRAME

- Responsible of the view creation and presentation
- It knows how to navigate
- Delegate of animations

PROTOCOLS

DEPENDENCY INVERSION PRINCIPLE (SOLID)

STRONG/WEAK

BE CAREFUL WITH RETAIL CYCLES

```
@interface TweetDetailViewController: UIViewController
@property (nonatomic, strong) id <TweetDetailPresenterInput> presenter;
@end

@interface TweetDetailPresenter: NSObject<TweetDetailPresenterInput>
@property (nonatomic, weak) id <TweetDetailViewInput> view;
@end

@implementation TweetDetailPresenter
- (void)sendTweet:(NSString*)tweet
{
  __weak typeof(self) welf = self;
  [self.view showLoader];
  [self.interactor sendTweetWithCompletion:^(NSError *task) {
    [welf.view hideLoader];
    if (!error) [welf.wireframe moveBack];
  }];
}
@end
```

ENTITIES

DON'T PASS NSMANAGEDOBJECTS!

USE PONSOS INSTEAD

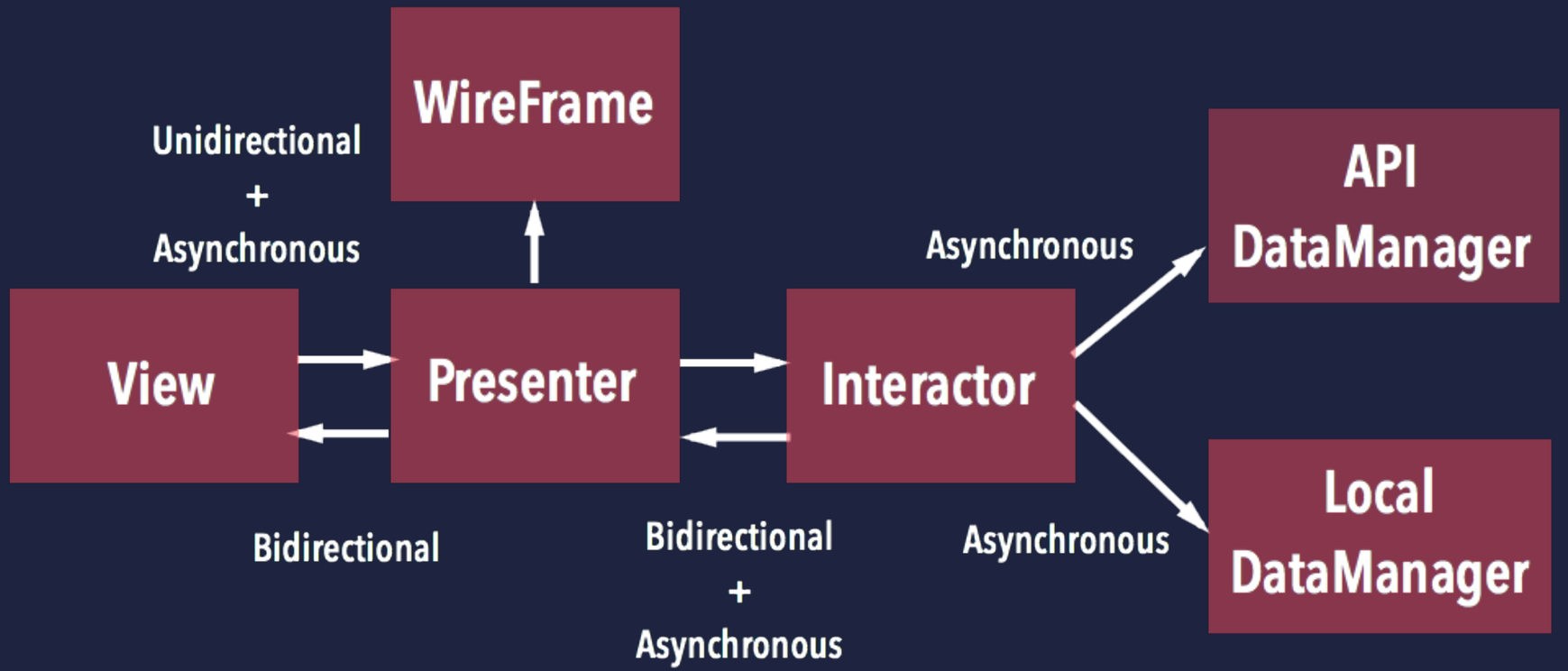
```
@interface TweetEntity: NSObject
@property (nonatomic, strong) NSSTring *body;
@property (nonatomic, strong) NSString *authorName;
@property (nonatomic, strong) NSDate *creationDate;
+ (TweetEntity*)tweetEntityFromTweet:(Tweet*)tweet;
@end
```


GIFs

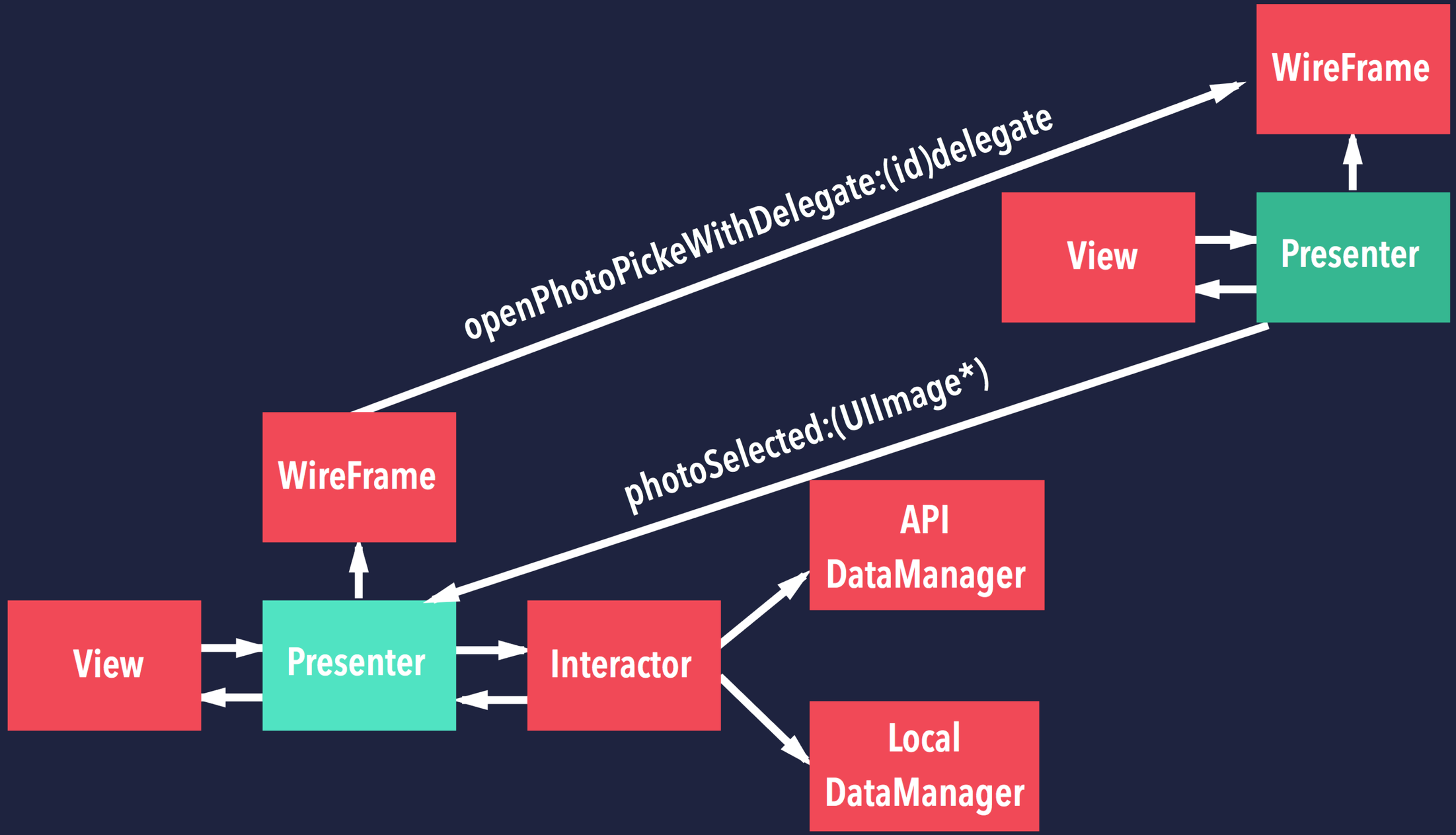
INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions

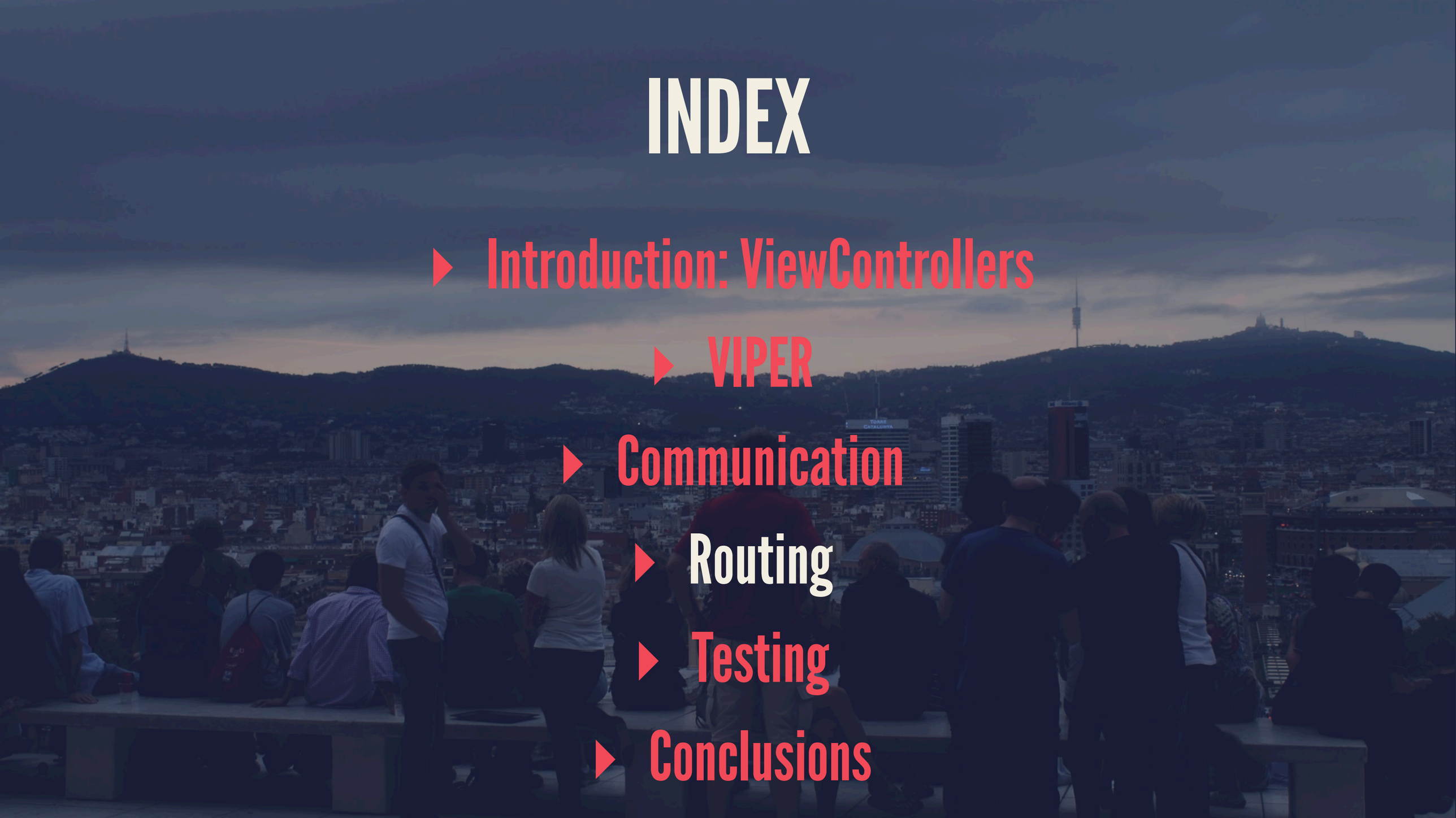
INNER COMMUNICATION


WireFrame
Unidirectional
+
Asynchronous
View
Presenter
Interactor
Asynchronous
API
DataManager
Local
DataManager
Asynchronous
Bidirectional
Bidirectional
+
Asynchronous

OUTER COMMUNICATION


WireFrame
openPhotoPickeWithDelegate:(id)delegate
View
Presenter
photoSelected:(UIImage*)
WireFrame
API
DataManager
View
Presenter
Interactor
Local
DataManager

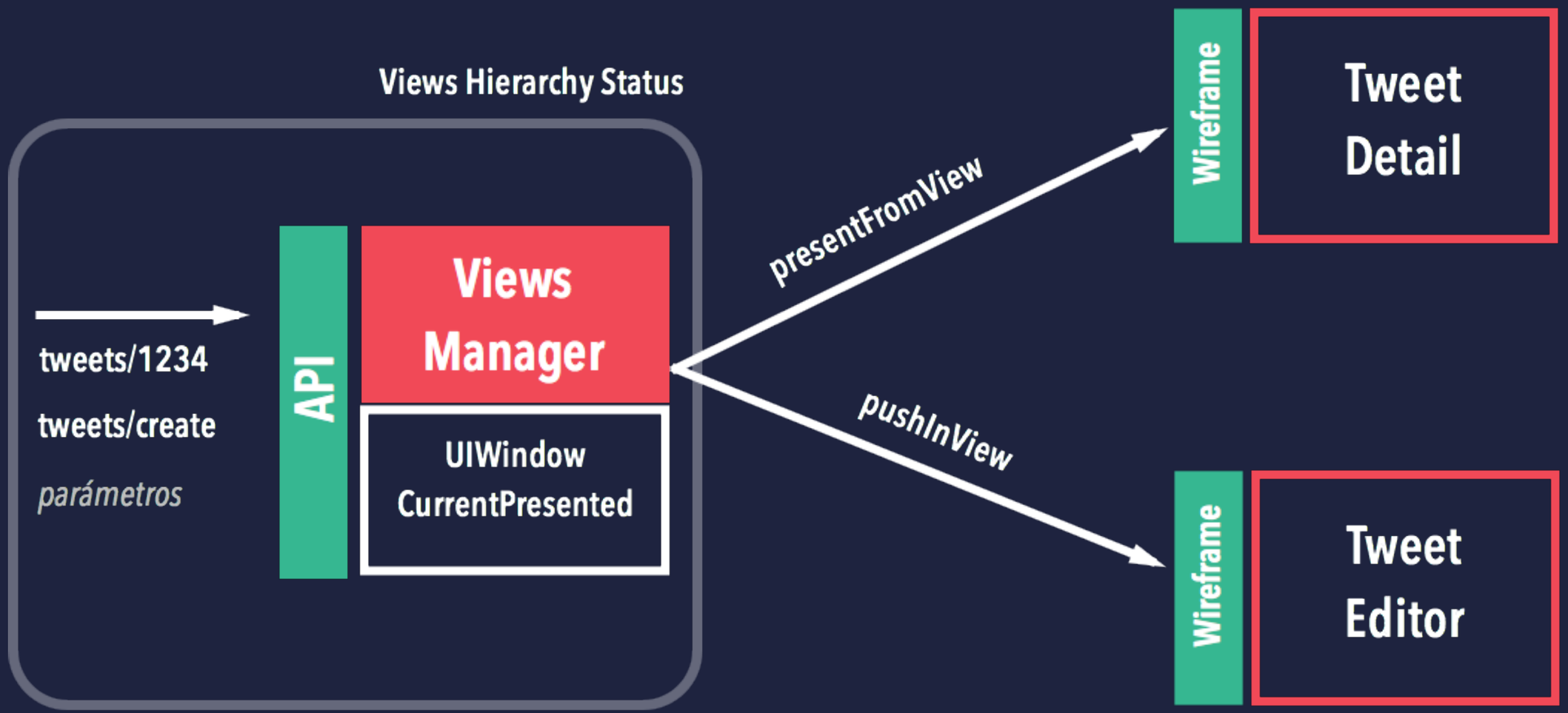
ACCESORY VIEWS



INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions

VIEWS MANAGER

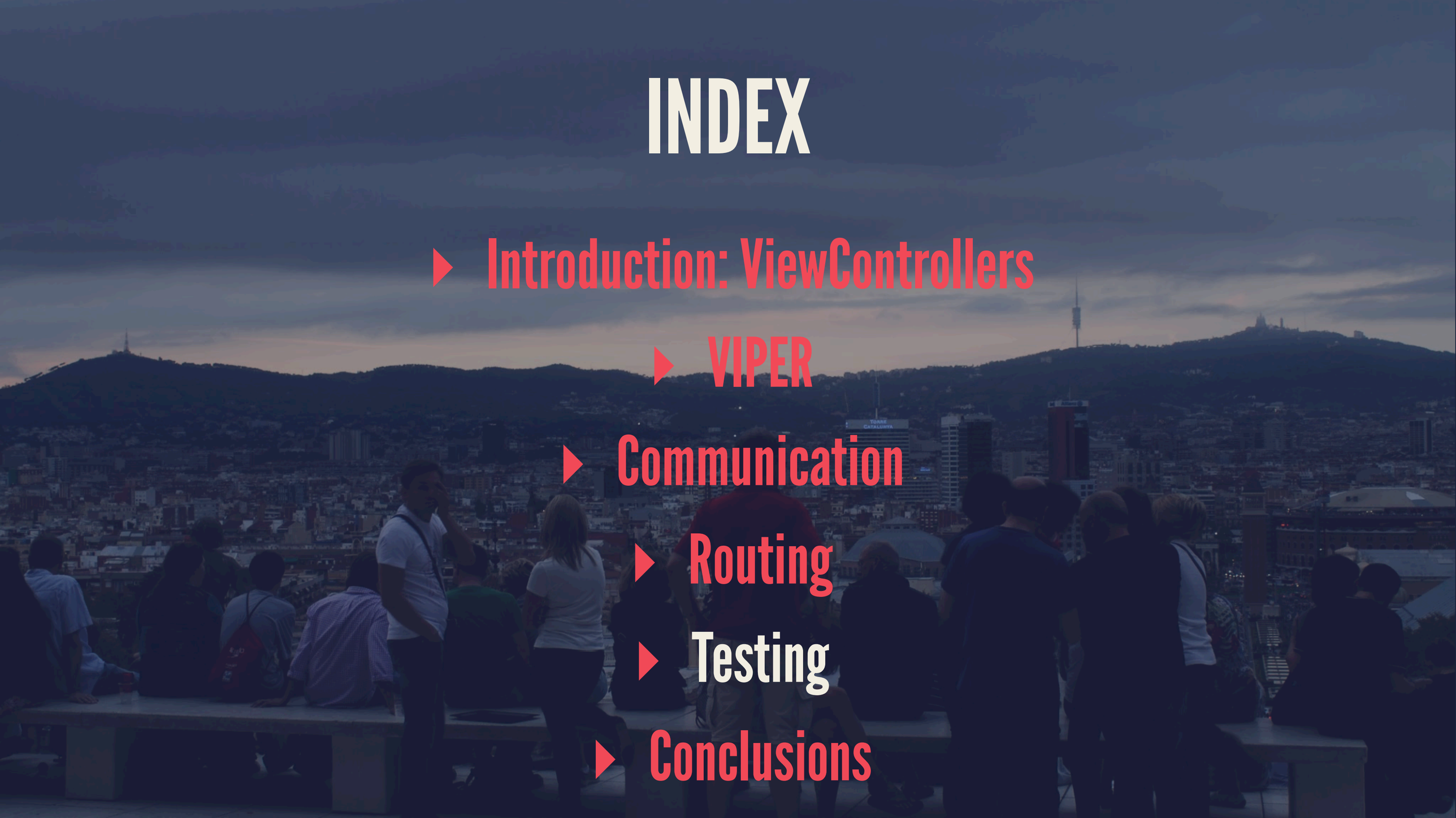

Views Hierarchy Status
tweets/1234
tweets/create
parámetros
API
Views Manager
UIWindow CurrentPresented
presentFromView
pushInView
Wireframe
Tweet Detail
Wireframe
Tweet Editor

API ROUTES

JLRoutes github.com/joeldev/JLRoutes

```
__weak typeof(self) welf = self;
[JLRoutes addRoute:@"/tweets/:tweetID" handler:^BOOL(NSDictionary *parameters) {
    NSString *tweetID = parameters[@"tweetID"];
    [TweetDetailWireframe presentInView:welf.currentPresentedView];
    return YES; // return YES to say we have handled the route
  }];
```


TESTING



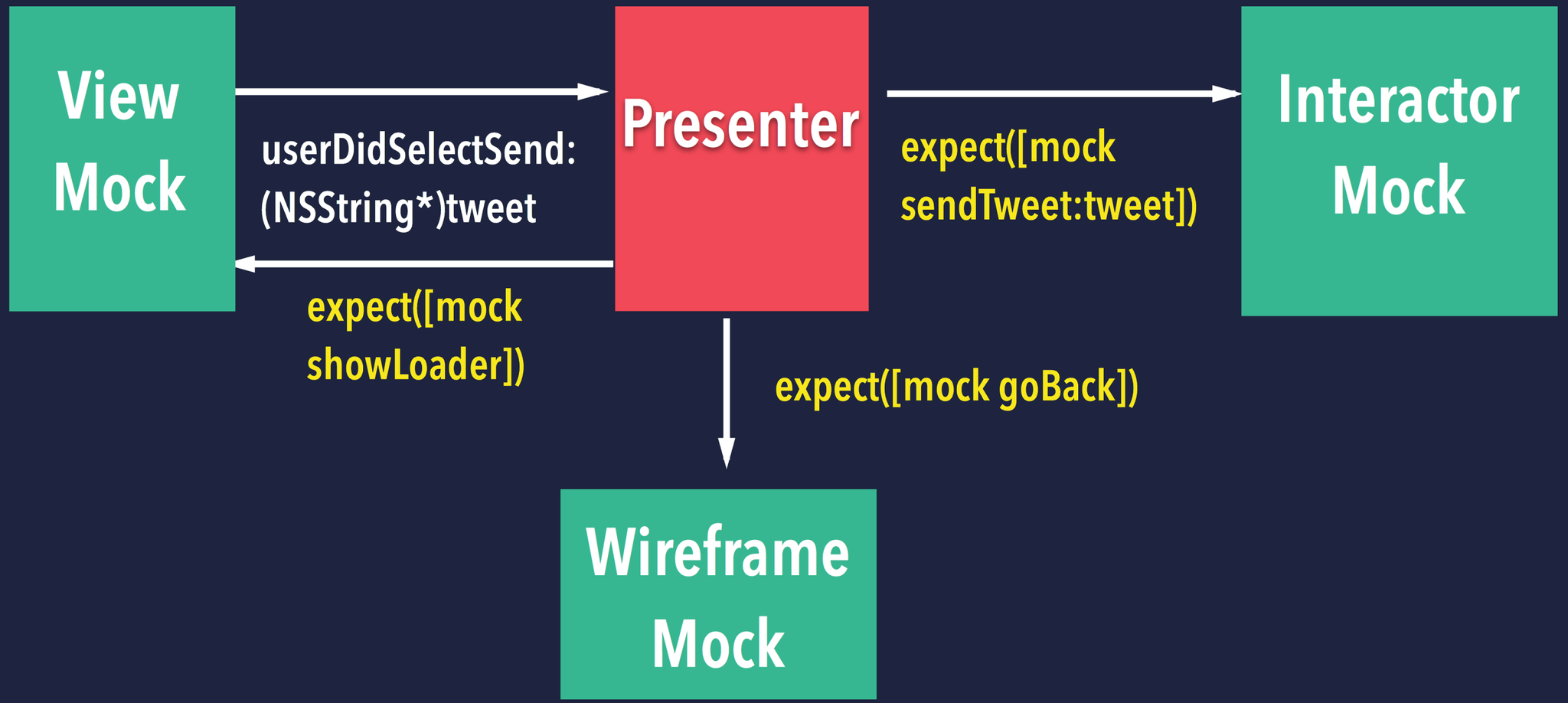
INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions

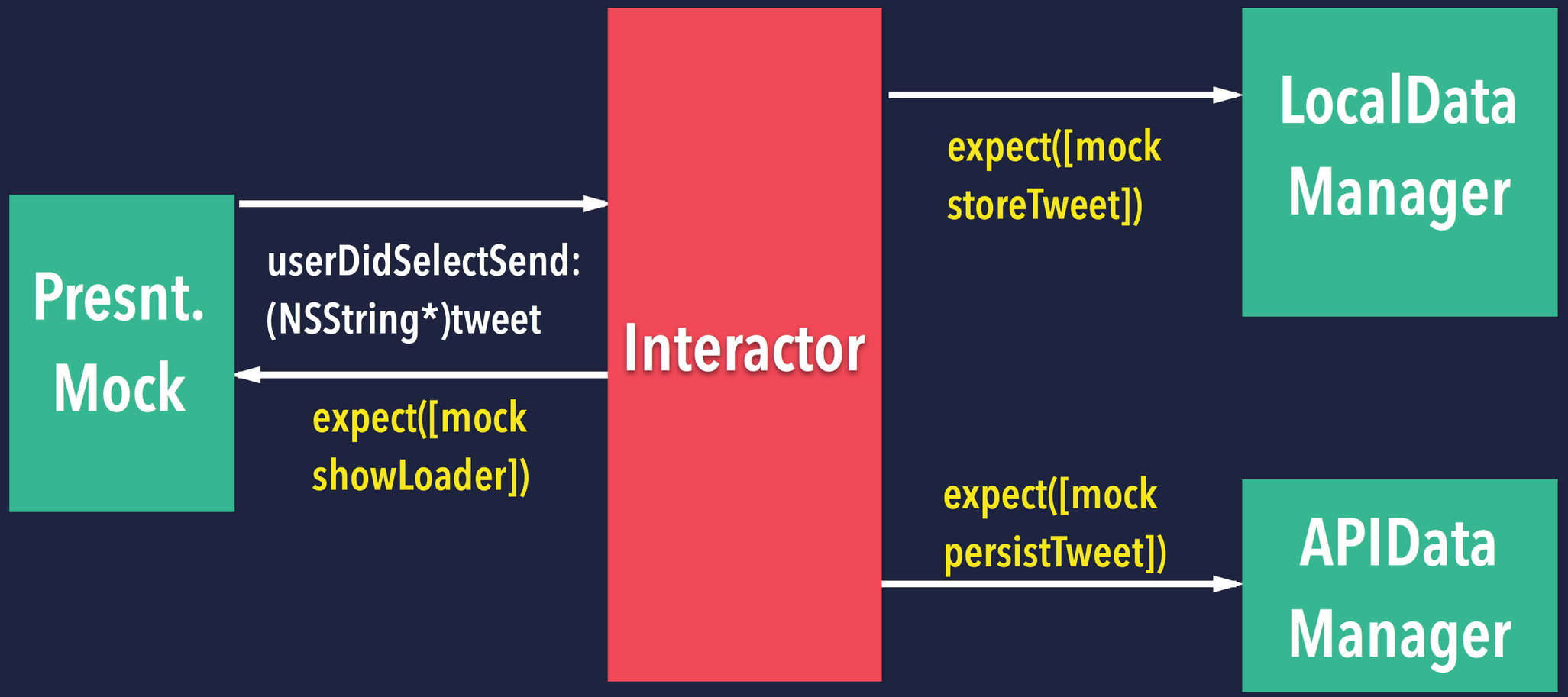
TESTING VIEW

Expected

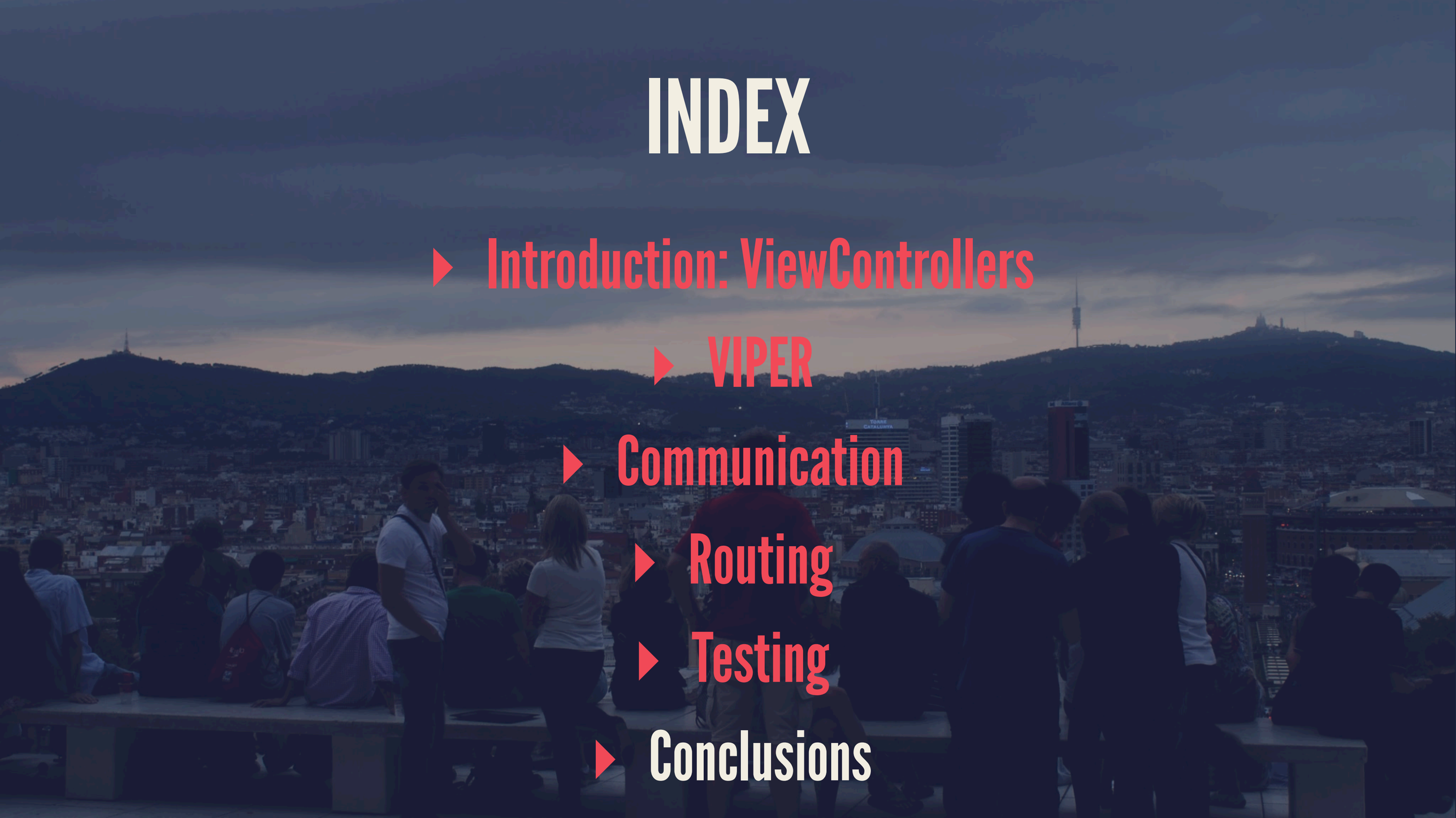
TESTING PRESENTER



TESTING INTERACTOR



EXAMPLE



INDEX

- Introduction: ViewControllers
- VIPER
- Communication
- Routing
- Testing
- Conclusions

CONCLUSIONS

- Heavy work but you and you'll team will thank it
- Keep in mind the responsibility of each component and don't mix them
- Refactor your components through iterations
- Decouple your code from the database models and data layers

RESOURCES

- VIPER Module Generator
- Objc.io post
- Mutual Mobile Engineering blog post
- Dobuts/Ideas/Suggestions pepibumur@gmail.com


Thank you.