

# Forms

# Forms

**We will build a form to save a new GIF to Persistence layer.**

# Forms

Forms have two separate concerns:

- The UI Layer
- The Data Layer

# UI Layer

- Typically, a list of views
- Usually scrollable
- Many different view kinds

# UI Layer

Allows the user to input things:

- Text (**UITextField**, **UITextView**, **UISearchController**)
- Touch interactions (**UISwitch**, **UIButton**)
- Redirections to other **UITableViewController**s

# UI Layer

UIKit provides a range of tools to choose from:

- **UIView**
- **UIStackView**
- **UITableView**
- **UICollectionView**

All of them have its pros and cons.

# Regular UIViews with constraints

## Pros

- Straightforward of what you see
- Totally customizable (i.e. not vertically stacked)

## Cons

- Not scalable:
  - *elements*  $\propto$  *constraints*
- Difficult to reason about (messy code)

# UIStackView

## Pros

- Straightforward to implement
- Easy to parametrize from the data layer (but not enforced)

## Cons

- Low performance on large forms
- Need to deal with the **UIScrollView**
- Only for iOS 9 and above



# UITableView (I)

## Pros

- Easy to scale
- Paves the way to parametrize with the data layer
- Form features built in (grouped table views, self-sizing cells)
- Performant for large forms

# UITableView (II)

## Cons

- Lots of boilerplate
- Only vertically stacked
- Not easily able to reach out specific UI elements

# UICollectionView

## Pros

- Highly customizable
- Easy to scale up or to parametrize from the data layer

## Cons

- Lots of boilerplate
- Form features **not** built in
- Not easily able to reach out specific UI elements

## TL;DR:

- For **reduced size** of forms or ones that will never grow (i.e. login), use UIView approach
- For **regular** forms, which can grow but are always stacked the same (i.e. personal data form)
- For **highly customized** forms *maybe* consider UICollectionView
- Almost never consider regular UIView approach.

# Things to Consider

- UI for different form states
  - not validated || validated-ok || error
- Keyboard manager
- How to show the error state:
  - pointing the errors to the related fields
  - showing generic error (i.e. **UIAlertController**)

# Data Layer

We've got to design a data structure that supports our needs. Things to consider:

- Input elements
- Data hierarchy (i.e. iOS Settings)
- Data validation
  - Sync: Mandatory fields / email / credit card
  - Async: Validation of State/City

# Data Layer

A `class`, instead of `struct` or `enum`:

- It should be a single representation of the data
- We don't want to mess with sync problems

```
struct Form { ... }
```

```
class FormViewController: UIViewController {  
    var form: Form {  
        return self.formValidator.form  
    }  
}
```

```
class FormValidator {  
    var form = Form()  
}
```



# Separation of Concerns

Separate Model and UI in the different Modules we created

# Separation of Concerns

Data and UI: separated in the different Modules we created

... so we can test them separately

# Separation of Concerns

Data and UI: separated in the different Modules we created

... so we can test them separately

... so we can reuse them (i.e. watchOS/tvOS)

# Separation of Concerns

Data and UI: separated in the different Modules we created

... so we can test them separately

... so we can reuse them (i.e. watchOS/tvOS)

... so we can optimize build times

# Demo