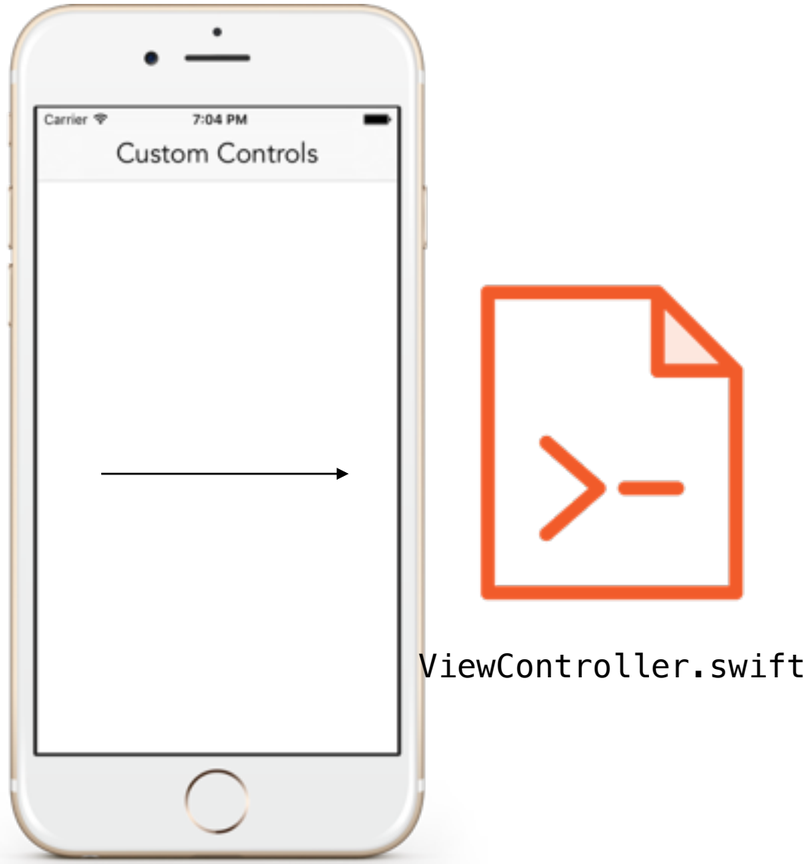
Communicating With View Controllers

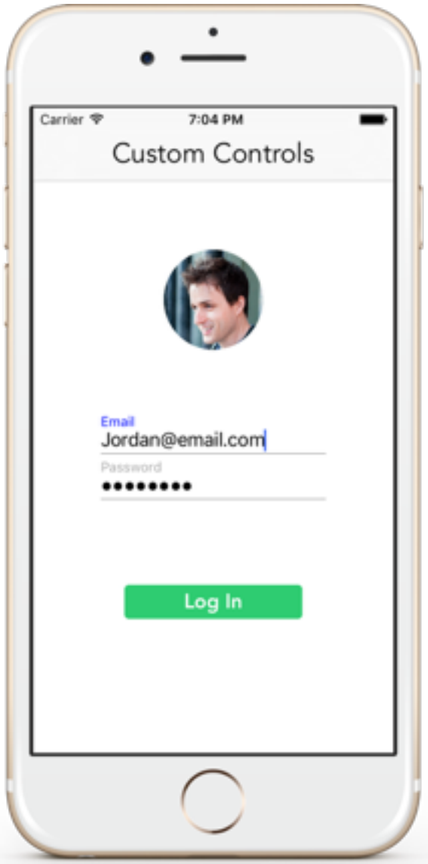
Jordan Morgan
IOS ENGINEER

@jordanmorgan10 www.dreaminginbinary.co

Implementing Communication



Implementing Communication



- Email Field

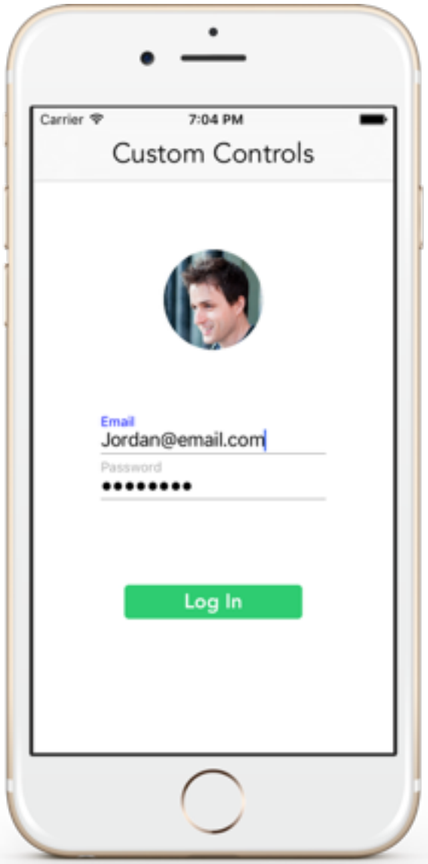
- Email sanitation is generally standard

- Password Field

- Passwords are more abstract

`ViewController.swift`

Implementing Communication



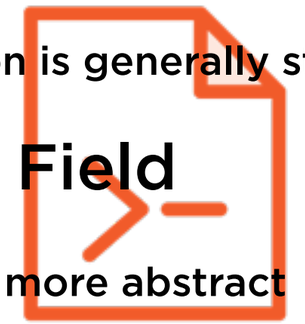
- Email Field

- Email sanitation is generally standard

- Or -

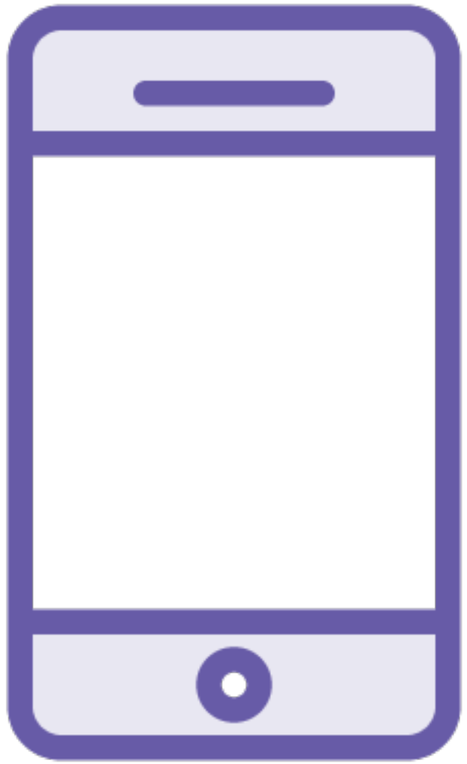
- Password Field

- Passwords are more abstract



`ViewController.swift`

Wiring Up Logic



ViewController.swift

- Repetition is key
- Hard to grasp first time
- Central ideas are discussed here

Ways to Communicate

Target-Action Pattern

Delegation

Closures (or blocks)

Key-Value Observing

Notifications

```
aControl.addTarget(aViewController, action: Selector(doSomething()),
forControlEvents: .TouchUpInside)
```

Target-Action Pattern

- **Identify a target and an action**
- **Action performed for an event**
- **Easy to use**

```
class AViewController : UIViewController, UITableViewDelegate
{
    self.aTableView.delegate = self

    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath)
    {
        //Do something when a tableview is tapped
    }
}
```

The Delegate Pattern

- **Delegates the work to whoever “conforms” to the protocol**
- **Much like a code contract, or an interface**
- **Be intentional, easy to overuse**

```
let aView = UIView()

//Animations are defined in the block here
UIView.animateWithDuration(1.0, animations: {
    aView.alpha = 0.0
})
```

Closures and Blocks

- **Much like callback functions**
- **Promotes loose coupling**
- **Watch out for retain cycle**
- **Nullability is important**

```
class AViewController: UIViewController
{
    override func viewDidLoad()
    {
        super.viewDidLoad()

        self.addObserver(self, forKeyPath: "SomethingHere", options: NSKeyValueObservingOptions.New, context: nil)
    }

    override func observeValueForKeyPath(keyPath: String?, ofObject object: AnyObject?, change: [String : AnyObject]?, context:
UnsafeMutablePointer<Void>)
    {
        if keyPath == "SomethingHere"
        {
            //Do intended work
        }
    }
}
```

Key-Value Observing

- **Commonly referred to as “KVO”**
- **Complete abstraction**
- **Posts notifications when the observed value changes**
- **A lot of pitfalls and hard to debug**

```
class AViewController: UIViewController
{
    override func viewDidLoad()
    {
        super.viewDidLoad()

        NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(doSomething), name: “MyNotification",
object: nil)
    }

    func doSomething()
    {
        //Notifcation psoted
    }
}
```

Notification Center

- **Singleton Object that posts notifications**
- **Can post just about anything**
- **Easy to set up**
- **...and easy to break things, and no compile time checks**

Which is Best?

Target-Action Pattern

Delegation

Closures (or blocks)

Key-Value Observing

Notifications

Demo

Create an animation for invalid input

Create a function taking in a closure

Demo

Call email validation logic from owner

Use a closure to perform on success