Introduction to iOS Development

James Addyman
@jasarien

Spencer MacDonald
@objcolumnist

Antonio Martínez
@amartinezgar

!=

!=


A Web Page
http://
Web App Title
Home > Products > Xyz > Features
not selected
selected
disabled
One
Two
Three
ComboBox

How does mobile development differ to web development?

How does mobile development differ to web development?

- Limited memory

How does mobile development differ to web development?

- Limited memory
- Slower processor

How does mobile development differ to web development?

- Limited memory
- Slower processor
- Limited screen space

How does mobile development differ to web development?

- Limited memory
- Slower processor
- Limited screen space
- Must be aware of available connectivity (3G / WiFi)

How does mobile development differ to web development?

- Limited memory
- Slower processor
- Limited screen space
- Must be aware of available connectivity (3G / WiFi)
  - Be aware that data transfer can cost money

How does mobile development differ to web development?

- Limited memory
- Slower processor
- Limited screen space
- Must be aware of available connectivity (3G / WiFi)
  - Be aware that data transfer can cost money

How does mobile development differ to web development?

How does mobile development differ to web development?

- UI focussed on touch control and content

How does mobile development differ to web development?

- UI focussed on touch control and content
- Apps must be approved

How does mobile development differ to web development?

- UI focussed on touch control and content

- Apps must be approved

- Not every one will be running the latest version of your app (supporting multiple versions)

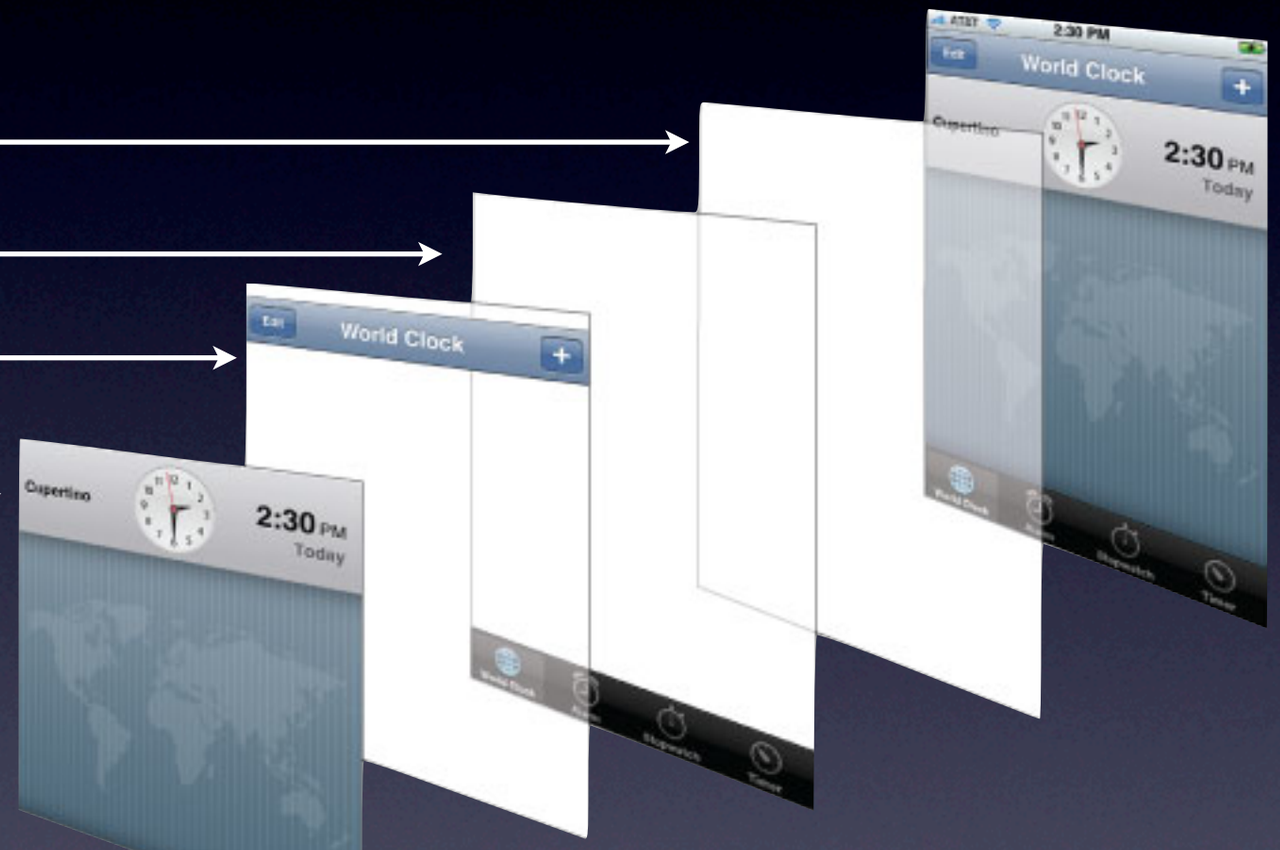
Structure of an iOS App

Window

Root view Controller

View

Subviews



Data Modelling Options

Data Modelling Options

- SQLite 3

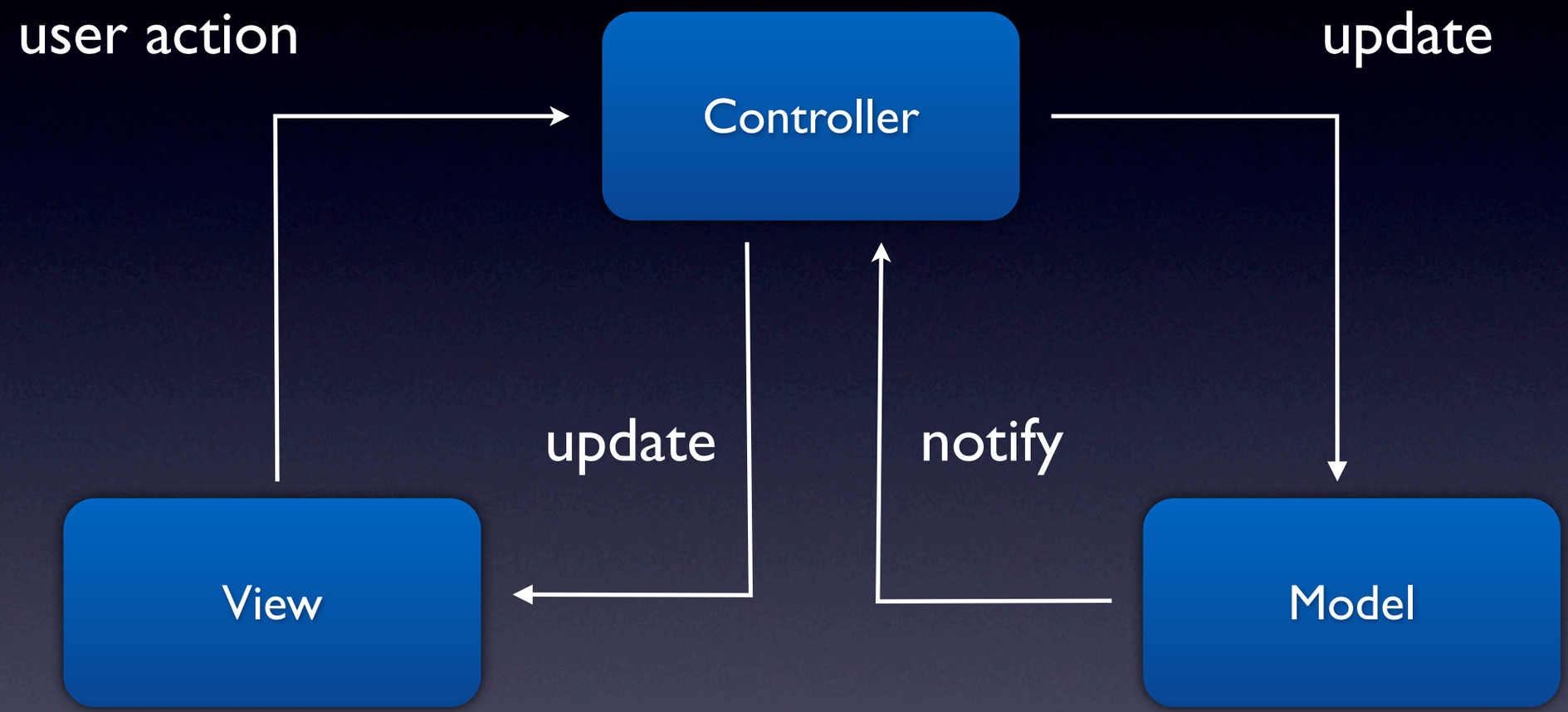
Data Modelling Options

- SQLite 3
- Core Data

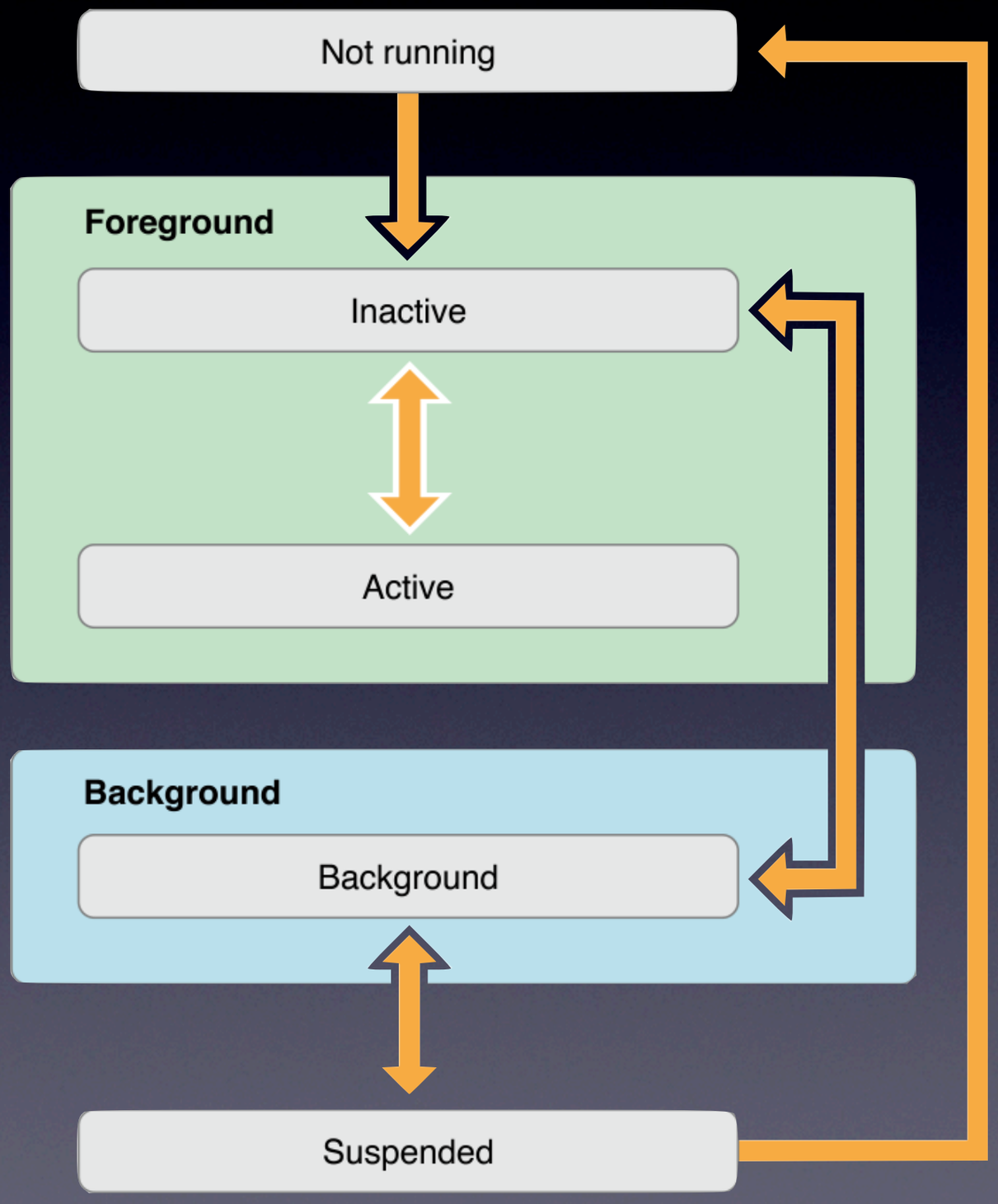
Data Modelling Options

- SQLite 3
- Core Data
- Object Serialisation

Model View Controller



App State and Multitasking



iOS Developer Program & App Review Process

- $99 / £59 per year
- Access to pre-release software
- Required for testing on physical devices
- Required for submitting apps to the app store
- Apps are reviewed before appearing on the store
    - Rejection is a risk, stick to the Human Interface Guidelines

Introduction to Objective-C

Introduction to Objective-C

Defining Classes

```
module MakersAcademy
  class Person < Object
  end
end
```

```
@interface MKADPerson : NSObject

@end


@implementation MKADPerson

@end
```

Introduction to Objective-C

Defining Classes

```
module MakersAcademy
  class Person < Object
  end
end
```

```
@interface MKADPerson : NSObject

@end


@implementation MKADPerson

@end
```

Introduction to Objective-C

Instantiating Objects

```
person = Person.new
```

```
MKADPerson *person = [[MKADPerson alloc] init];
```

Introduction to Objective-C

Accessor Methods

```
class Person < Object
  def first_name
    @first_name
  end

  def set_first_name(first_name)
    @first_name = first_name
  end
end
```

Introduction to Objective-C

Accessor Methods

```
@interface MKADPerson : NSObject {
      NSString *_firstName;
}

- (NSString *)firstName;
- (void)setFirstName:(NSString *)firstName;

@end

@implementation MKADPerson

- (NSString *)firstName {
   return _firstName;
}

- (void)setFirstName:(NSString *)firstName {
   _firstName = firstName;
}

- (void)dealloc {
   _firstName = nil;
}

@end
```

Introduction to Objective-C

Accessor Methods

```
MKADPerson *person = [[MKADPerson alloc] init];

[person setFirstName:@"Spencer"];
NSString *firstName = [person firstName];
```

Introduction to Objective-C

Read-Write Properties

```
class Person < Object
   attr_accessor :first_name
end
```

```
@interface MKADPerson : NSObject
  @property (strong) NSString *firstName;
@end

@implementation MKADPerson

- (void)dealloc {
  self.firstName = nil;
}

@end
```

Introduction to Objective-C

Using Properties

```
MKADPerson *person = [[MKADPerson alloc] init];

person.firstName = @"Spencer";
NSString *firstName = person.firstName;
```

Introduction to Objective-C

Read-Only Properties

```
class Person < Object
  attr_reader :first_name
end
```

```
@interface MKADPerson : NSObject
  @property (strong, readonly) NSString *firstName;
@end
```

Introduction to Objective-C

Initialisers

```
class Person < Object
  attr_accessor :first_name
  attr_accessor :last_name

  def initialize(first_name,last_name)
    self.first_name = first_name
    self.last_name = last_name
  end
end
```

```
MakersAcademy::Person.new("Spencer","MacDonald")
```

Introduction to Objective-C

Initialisers

```
@interface MKADPerson : NSObject

@property (strong) NSString *firstName;
@property (strong) NSString *lastName;

- (id)initWithFirstName:(NSString *)firstName lastName:(NSString *)lastName;

@end

@implementation MKADPerson

- (id)initWithFirstName:(NSString *)firstName lastName:(NSString *)lastName {
  if((self = [super init])) {
    self.firstName = firstName;
    self.lastName = lastName;
  }
  return self;
}

- (void)dealloc {
  self.fistName = nil;
  self.lastName = nil;
}

@end
```

Introduction to Objective-C

Testing Equality

```
if([person1 isEqual:person2])
{
   //Do Something
}
```

Introduction to Objective-C

Foundation Objects

- NSString
- NSArray
- NSDictionary
- NSNumber
- NSNull

Introduction to Objective-C

Strings

Creating and Initialising String Objects

```
NSString *firstName = @"Spencer";

NSString *firstName = [[NSString alloc] initWithString:@"Spencer"];

NSString *firstName = [NSString stringWithString:@"Spencer"];
```

Creating and Initialising Mutable String Objects

```
NSMutableString *firstName = [[NSMutableString alloc] initWithString:@"Spencer"];

NSMutableString *firstName = [NSMutableString stringWithString:@"Spencer"];
```

Introduction to Objective-C

Strings

```
firstName = "Spencer"
age = 26

text = "#{firstName} is #{age} years old."
```

```
NSString *firstName = @"Spencer";
NSUInteger age = 26;

NSString *text = [NSString stringWithFormat:@"%@ is %lu years old.", firstName,
age];
```

Introduction to Objective-C

Strings

```
firstName = "Spencer"
age = 26

text = "#{firstName} is #{age} years old."
```

```
NSString *firstName = @"Spencer";
NSUInteger age = 26;

NSString *text = [NSString stringWithFormat:@"%@ is %lu years old.", firstName,
age];
```

Introduction to Objective-C

Arrays

Immutable Array

```
NSArray *names = [NSArray arrayWithObjects:@"James",@"Spencer",nil];
NSArray *names = @[@"James",@"Spencer"];
NSString *first = names[0];
NSUInteger count = [names count];
```

Mutable Array

```
NSMutableArray *names = [NSMutableArray arrayWithObjects:@"James",@"Spencer",nil];
[names addObject:@"Antonio"];
[names removeObjectAtIndex:0];
[names insertObject:@"James" atIndex:0];
```

Fast Enumeration

```
for (NSString *name in names) {
   //Do Something
}
```

```
names.each do |name|
  # Do Something
end
```

Introduction to Objective-C

Dictionaries

Immutable Dictionary

```
NSDictionary *info = [NSDictionary
dictionaryWithObjectsAndKeys:@"Spencer",@"firstName",@"MacDonald",@"lastName",nil];
NSDictionary *info = @{ @"firstName" : @"Spencer", @"lastName" : @"MacDonald"};
NSString *firstName = info[@"firstName"];
```

Mutable Dictionary

```
NSMutableDictionary *info = [NSMutableDictionary
dictionaryWithObjectsAndKeys:@"Spencer",@"firstName",@"MacDonald",@"lastName",nil];

info[@"firstName"] = @"James";
info[@"lastName"] = @"Addyman";
```

Introduction to Objective-C

Numbers

```
NSUInteger x = 3;
NSUInteger y = 2;
```

```
NSNumber *x = [NSNumber numberWithUnsignedInt:3];
NSNumber *y = [NSNumber numberWithUnsignedInt:2];

NSUInteger result = [x unsignedIntValue] +  [y unsignedIntValue];
```

Introduction to Objective-C

Null

```
NSMutableDictionary *dict = [NSMutableDictionary dictionary];

if ([name length])
{
   [dict setObject:name forKey:@”name”];
}
else
{
   [dict setObject:[NSNull null] forKey:@”name”];
}
```

Introduction to Objective-C

Nil

```
NSString *firstName = nil;

NSUInteger length = [firstName length];
```

Memory Management

Manual Reference Counting

- Objective-C uses reference counting
- -retain increases reference count
- -release decreases reference count
- Objects are removed from memory when their reference count reaches 0
- Newly allocated objects have an owning reference (retain count of 1)
  - CARN Rule (Copy, Alloc, Retain, New)

Memory Management

Automatic Reference Counting (ARC)

- No need to call -retain or -release in your code
- Compiler intelligently inserts retains and releases when they're needed at compile time
- System still uses reference counting under the hood
- ARC is *not* garbage collection
- Not 100% memory-leak safe

Memory Management

Manual Reference Counting Example

```
MyObject *object = [[MyObject alloc] init]; // retain count 1

[array addObject:object]; // array retains object (2)

[object release]; // owner releases object (1)

[array removeObject:object]; // array releases object (0)
                             // object is deallocated
```

Memory Management

Manual Reference Counting Example

```
MyObject *object = [[MyObject alloc] init]; // retain count 1

[array addObject:object]; // array retains object (2)

[object release]; // owner releases object (1)

[array removeObject:object]; // array releases object (0)
                             // object is deallocated
```

Memory Management

Automatic Reference Counting Example

```
MyObject *object = [[MyObject alloc] init]; // retain count 1

[array addObject:object]; // array retains object (2)

// compiler will intelligently insert a release call here

[array removeObject:object]; // array releases object (0)
                             // object is deallocated
```

Starting the App

Hello World

- Create a new project
- Get acquainted with Xcode
- iOS Simulator
- Meet Interface Builder

Breif intro to Xcode

Brief into to Interface Builder

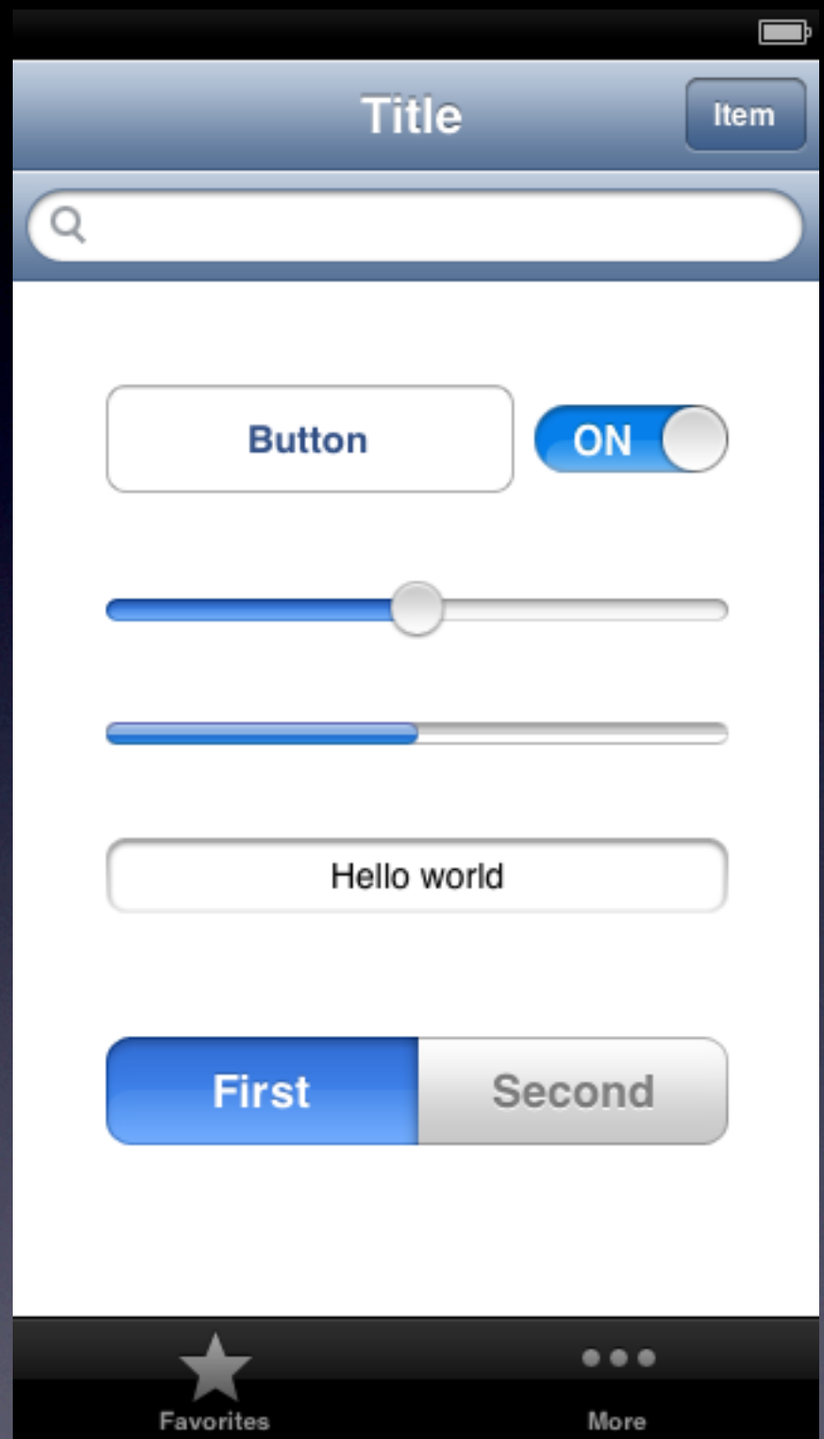
Putting a label on screen

Building their first app

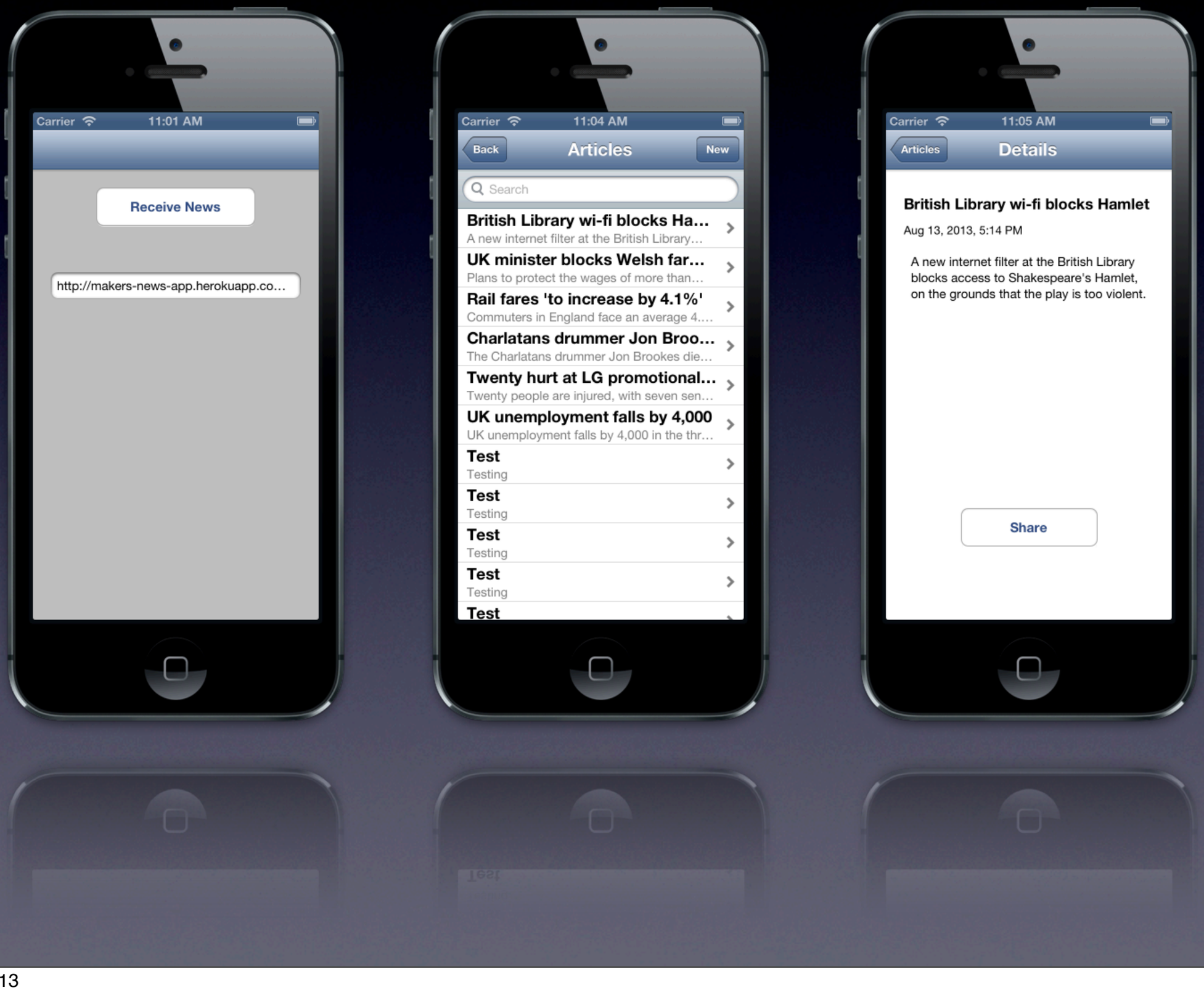
Demo

Designing the UI

- UIKit
- Views, buttons, sliders, switches, tables, text fields, labels, image views, scroll views, web views, maps.....
- UIKit favours a composition over inheritance approach
  - create new controls by combining views



Designing the UI



create main view

add button

add text field

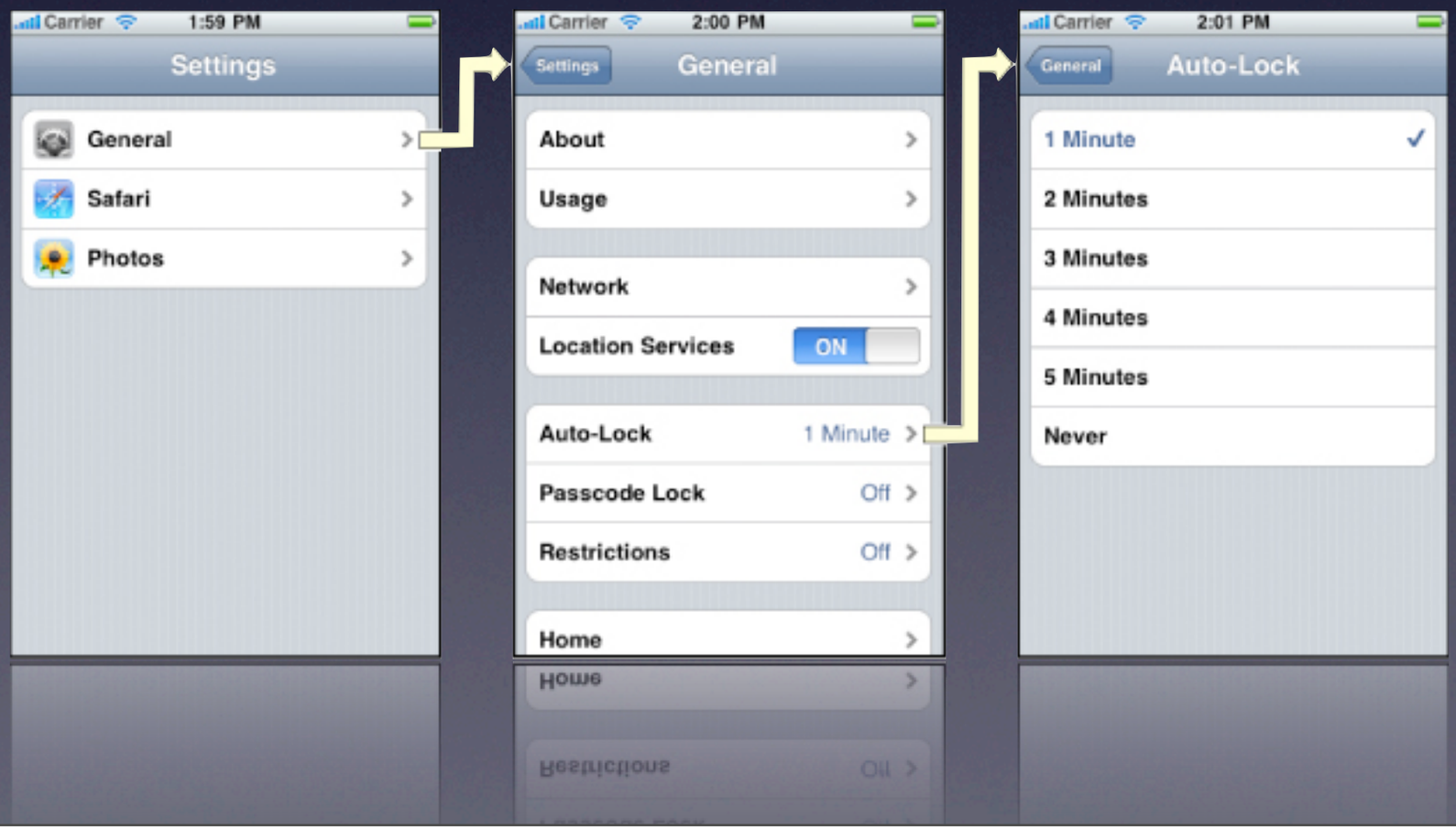
connect Outlets

emo

UINavigationController

Controlling your app's navigation

- Specialised view controller that manages navigation of hierarchical content
- Easy to implement and navigate



UINavigationController

Controlling your app's navigation

```
-(void)openNewView
{
  //Create a new instance of our view controller

  MKADArticleDetailsViewController *detailViewController
  = [[MKADArticleDetailsViewController alloc]
  initWithNibName:@"MKADArticleDetailsViewController"
  bundle:nil];

  //push it onto the navigation stack

  [self.navigationController
  pushViewController:detailViewController animated:YES];
}
```

UINavigationController

Controlling your app's navigation

```
-(void)openNewView
{
  //Create a new instance of our view controller

  MKADArticleDetailsViewController *detailViewController
  = [[MKADArticleDetailsViewController alloc]
  initWithNibName:@"MKADArticleDetailsViewController"
  bundle:nil];

  //push it onto the navigation stack

  [self.navigationController
  pushViewController:detailViewController animated:YES];
}
```

UINavigationController

Controlling your app's navigation

```
-(void)openNewView
{
  //Create a new instance of our view controller

  MKADArticleDetailsViewController *detailViewController
  = [[MKADArticleDetailsViewController alloc]
  initWithNibName:@"MKADArticleDetailsViewController"
  bundle:nil];

  //push it onto the navigation stack

  [self.navigationController
  pushViewController:detailViewController animated:YES];
}
```

UINavigationController

Controlling your app's navigation

- Navigating backwards usually handled for you by the navigation controller

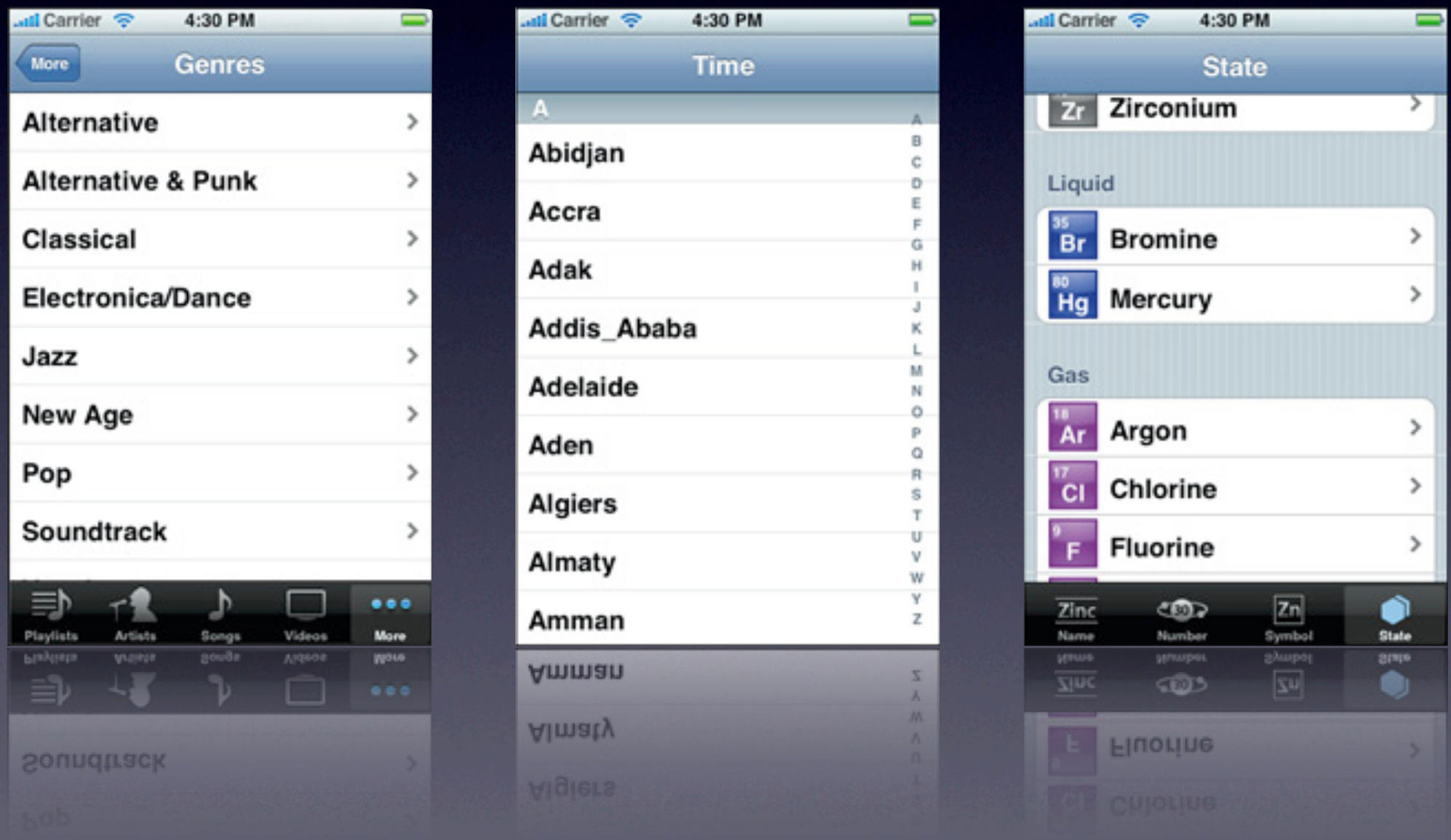
```
-(void)goBack
{
  self.navigationController
  popViewControllerAnimated:YES];
}
```

create navigation controller

Demo

UITableView

Representing data in a scrollable view with multiple rows



UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

Telling the table how many sections there are

```
- (NSInteger)numberOfSectionsInTableView:(UITableView
*)tableView{
  Return 1;
}
```

UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

Telling the table how many rows are in each section

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
  Return 15;
}
```

UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

Configure the cell for each row

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   static NSString *CellIdentifier = @"Cell";
   UITableViewCell *cell = [tableView
   dequeueReusableCellWithIdentifier:CellIdentifier];

   if (cell == nil) {
      cell = [[UITableViewCell alloc]
      initWithStyle:UITableViewCellStyleSubtitle
      reuseIdentifier:CellIdentifier];
    }

   cell.textLabel.text = [NSString stringWithFormat:@”%d”, indexPath.row];

   return cell;
}
```

UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

Configure the cell for each row

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   static NSString *CellIdentifier = @"Cell";
   UITableViewCell *cell = [tableView
   dequeueReusableCellWithIdentifier:CellIdentifier];

   if (cell == nil) {
      cell = [[UITableViewCell alloc]
      initWithStyle:UITableViewCellStyleSubtitle
      reuseIdentifier:CellIdentifier];
    }

   cell.textLabel.text = [NSString stringWithFormat:@"%d", indexPath.row];

   return cell;
}
```

UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

Configure the cell for each row

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   static NSString *CellIdentifier = @"Cell";
   UITableViewCell *cell = [tableView
   dequeueReusableCellWithIdentifier:CellIdentifier];

   if (cell == nil) {
      cell = [[UITableViewCell alloc]
      initWithStyle:UITableViewCellStyleSubtitle
      reuseIdentifier:CellIdentifier];
    }

   cell.textLabel.text = [NSString stringWithFormat:@"%d", indexPath.row];

   return cell;
}
```

UITableView

UITableViewDataSource - Providing the information UITableView needs to build the table

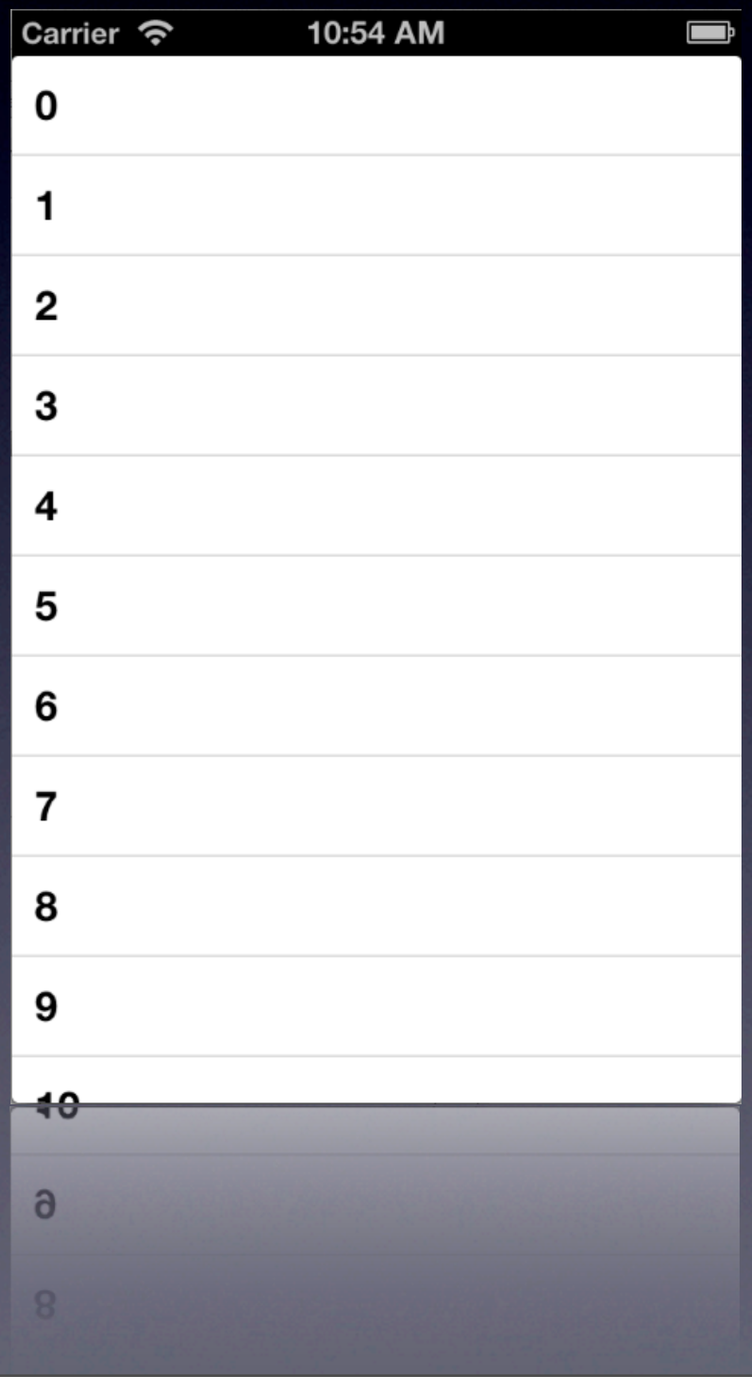
Configure the cell for each row

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   static NSString *CellIdentifier = @"Cell";
   UITableViewCell *cell = [tableView
   dequeueReusableCellWithIdentifier:CellIdentifier];

   if (cell == nil) {
      cell = [[UITableViewCell alloc]
      initWithStyle:UITableViewCellStyleSubtitle
      reuseIdentifier:CellIdentifier];
    }

   cell.textLabel.text = [NSString stringWithFormat:@"%d", indexPath.row];

   return cell;
}
```

UITableView

UITableViewDataSource - Providing the information
UITableView needs to build the table

The result



UITableView

Reusing Table View Cells

- Cells are loaded dynamically, on demand
- Only visible rows have a table view cell
- When a row is scrolled off screen, it's cell is reused for the new row that is scrolled on screen

UITableView

UITableViewDelegate - Implementing behaviour to manage selections, rearrange cells, configure headers, etc...

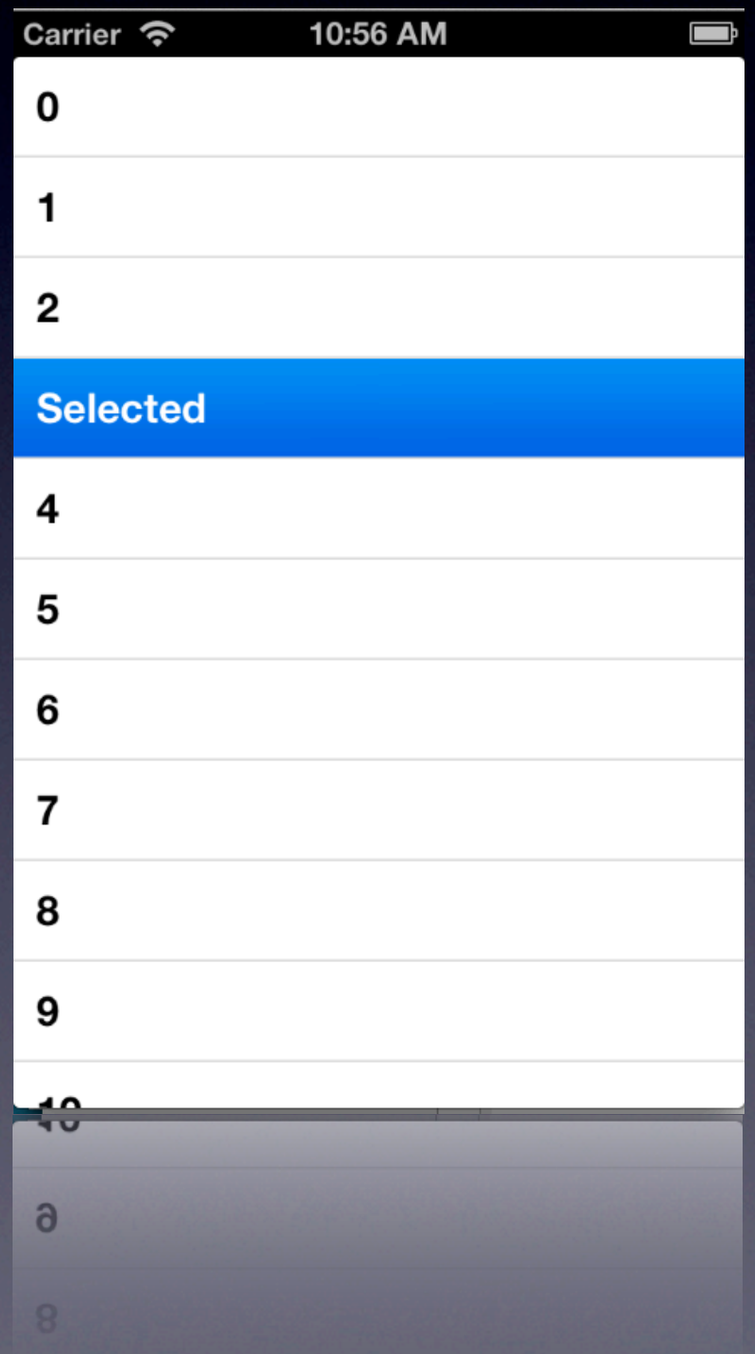
Selecting a row in the table

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
  UITableViewCell *cell = [tableView
  cellForRowAtIndexPath:indexPath];

  cell.textLabel.text = @"Selected";
}
```

UITableView

UITableViewDelegate - Implementing behaviour to manage selections, rearrange cells, configure headers, etc...

The result



UITableView

UITableViewDelegate - Implementing behaviour to manage selections, rearrange cells, configure headers, etc...

Configuring other parts of the table view

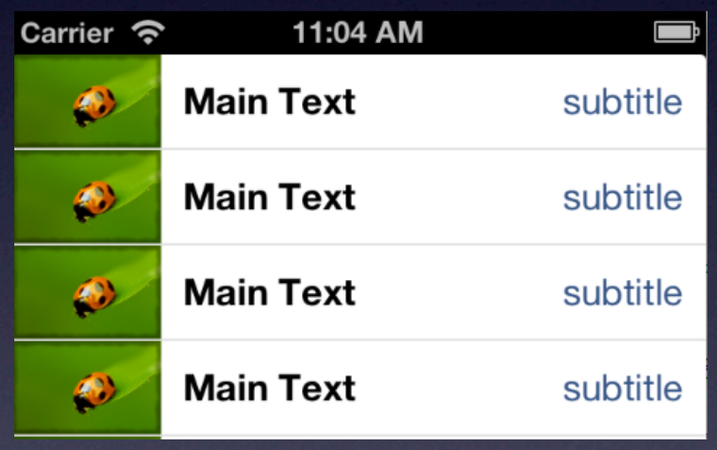
```
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath;

- (CGFloat)tableView:(UITableView *)tableView
heightForHeaderInSection:(NSInteger)section;

- (CGFloat)tableView:(UITableView *)tableView
heightForFooterInSection:(NSInteger)section;

- (UIView *)tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section;
```

UITableViewCell

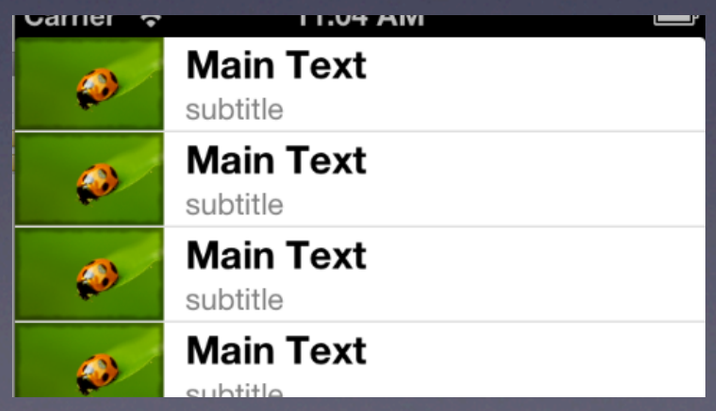
- A cell object is reusable
- Some built in cell styles ready for use

`UITableViewCellStyleDefault`

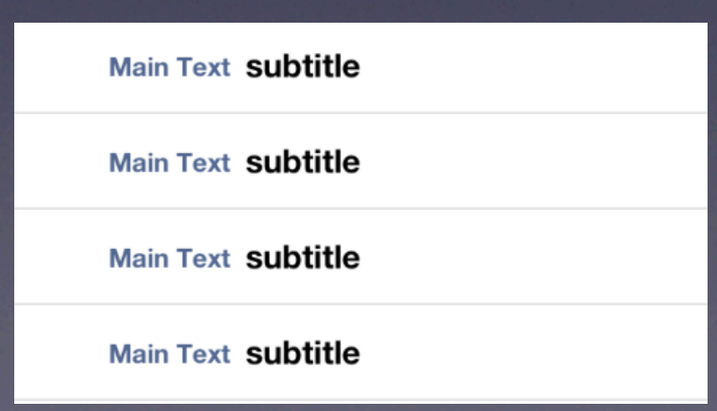
`UITableViewCellStyleValue1`



`UITableViewCellStyleSubtitle`

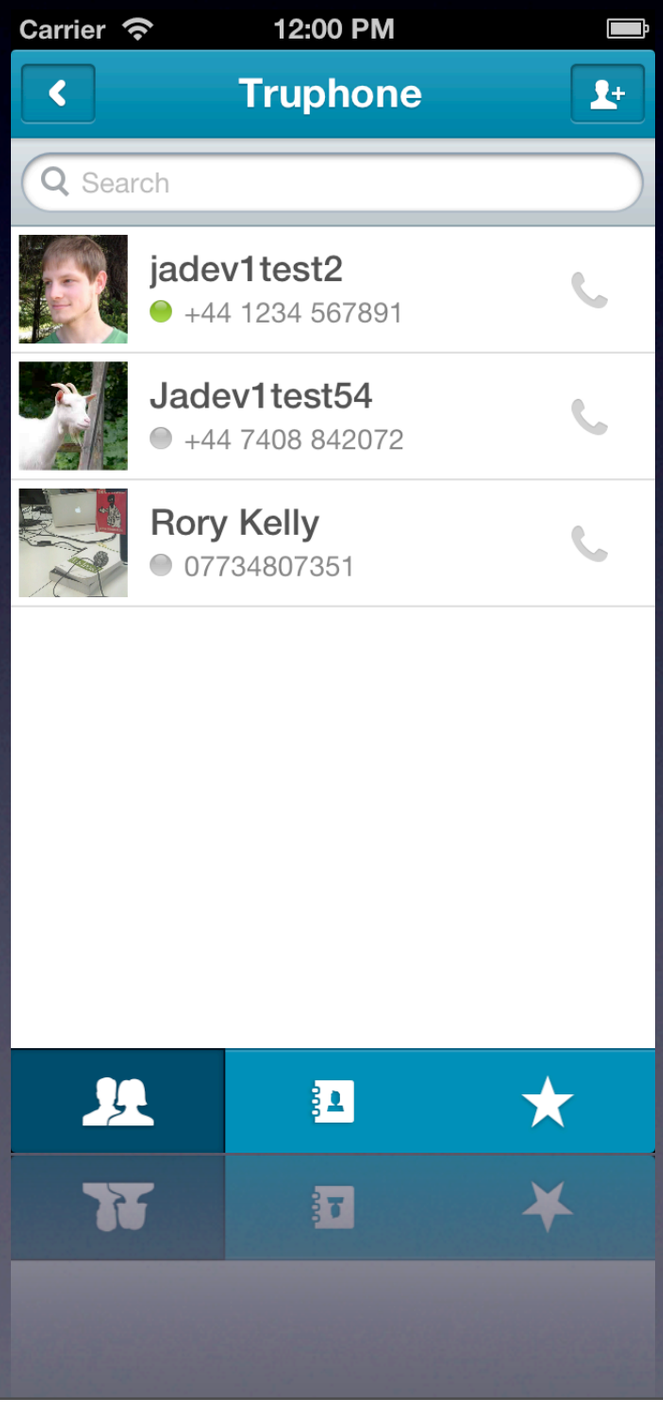


`UITableViewCellStyleValue2`



UITableViewCell

Custom Table Cells



Implement the Table view controller

Create detail view, push it on stack

Demo

Making an HTTP Request

- NSURLRequest
  - Encapsulates the request, its URL, method, headers and body
- NSURLConnection
  - Encapsulates the HTTP connection and returns the response to its delegate

Making an HTTP Request

```
NSURL *url = [NSURL URLWithString:@”http://makers-news-app.herokuapp.com/articles.json”];


NSURLRequest *request = [NSURLRequest requestWithURL:url];


[NSURLConnection connectionWithRequest:request delegate:self];
```

Making an HTTP Request

Downloading the response data

```
- (void)connection:(NSURLConnection *)connection
didReceiveResponse:(NSURLResponse *)response {
  self.dataReceived = [[NSMutableData alloc] init];
};



- (void)connection:(NSURLConnection *)connection
didReceiveData:(NSData *)data {
  [self.dataReceived appendData:data];
};
```

Making an HTTP Request

Handling Completion

```
- (void)connectionDidFinishLoading:(NSURLConnection
*)connection {
NSError *error = nil;
    NSArray *jsonReceivedArray = [NSJSONSerialization
JSONObjectWithData:self.dataReceived
           options:0
             error:&error];

    [self parseReceivedJSON:jsonReceivedArray];
}
```

Making an HTTP Request

Handling Failure

```
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError
*)error
{
   UIAlertView *alertView = [[UIAlertView alloc] init];
   [alertView setTitle:@”Error”];
   [alertView setMessage:error.localizedDescription];
   [alertView setDelegate:self];
   [alertView setCancelButtonTitle:@”OK”];

   [alertView show];
}
```

Making an HTTP Request

Parsing the response

```
- (void)parseReceivedJSON:(NSArray *)jsonReceivedArray
{
    //Parse it
    NSMutableArray *articlesMutableArray = [[NSMutableArray alloc] init];
    for (NSDictionary *dictionaryFromJSON in jsonReceivedArray)
    {
        Article *article = [[Article alloc] init];
        article.title = [dictionaryFromJSON objectForKey@”title”];
        // repeat for other properties
        [articlesMutableArray addObject:article];
    }

    self.articles = [NSArray arrayWithArray:articlesMutableArray];
    [self.tableView reloadData];
}
```

Making an HTTP Request

Parsing the response

```
- (void)parseReceivedJSON:(NSArray *)jsonReceivedArray
{
    //Parse it
    NSMutableArray *articlesMutableArray = [[NSMutableArray alloc] init];
    for (NSDictionary *dictionaryFromJSON in jsonReceivedArray)
    {
        Article *article = [[Article alloc] init];
        article.title = [dictionaryFromJSON objectForKey@”title”];
        // repeat for other properties
        [articlesMutableArray addObject:article];
    }

    self.articles = [NSArray arrayWithArray:articlesMutableArray];
    [self.tableView reloadData];
}
```

Making an HTTP Request

Parsing the response

```
- (void)parseReceivedJSON:(NSArray *)jsonReceivedArray
{
    //Parse it
    NSMutableArray *articlesMutableArray = [[NSMutableArray alloc] init];
    for (NSDictionary *dictionaryFromJSON in jsonReceivedArray)
    {
        Article *article = [[Article alloc] init];
        article.title = [dictionaryFromJSON objectForKey@”title”];
        // repeat for other properties
        [articlesMutableArray addObject:article];
    }

    self.articles = [NSArray arrayWithArray:articlesMutableArray];
    [self.tableView reloadData];
}
```

Making an HTTP Request

Parsing the response

```
- (void)parseReceivedJSON:(NSArray *)jsonReceivedArray
{
    //Parse it
    NSMutableArray *articlesMutableArray = [[NSMutableArray alloc] init];
    for (NSDictionary *dictionaryFromJSON in jsonReceivedArray)
    {
        Article *article = [[Article alloc] init];
        article.title = [dictionaryFromJSON objectForKey@”title”];
        // repeat for other properties
        [articlesMutableArray addObject:article];
    }

    self.articles = [NSArray arrayWithArray:articlesMutableArray];
    [self.tableView reloadData];
}
```

Making an HTTP Request

Parsing the response

```
- (void)parseReceivedJSON:(NSArray *)jsonReceivedArray
{
    //Parse it
    NSMutableArray *articlesMutableArray = [[NSMutableArray alloc] init];
    for (NSDictionary *dictionaryFromJSON in jsonReceivedArray)
    {
        Article *article = [[Article alloc] init];
        article.title = [dictionaryFromJSON objectForKey@”title”];
        // repeat for other properties
        [articlesMutableArray addObject:article];
    }

    self.articles = [NSArray arrayWithArray:articlesMutableArray];
    [self.tableView reloadData];
}
```

Implement the HTTP requets

Implement create new article

emo

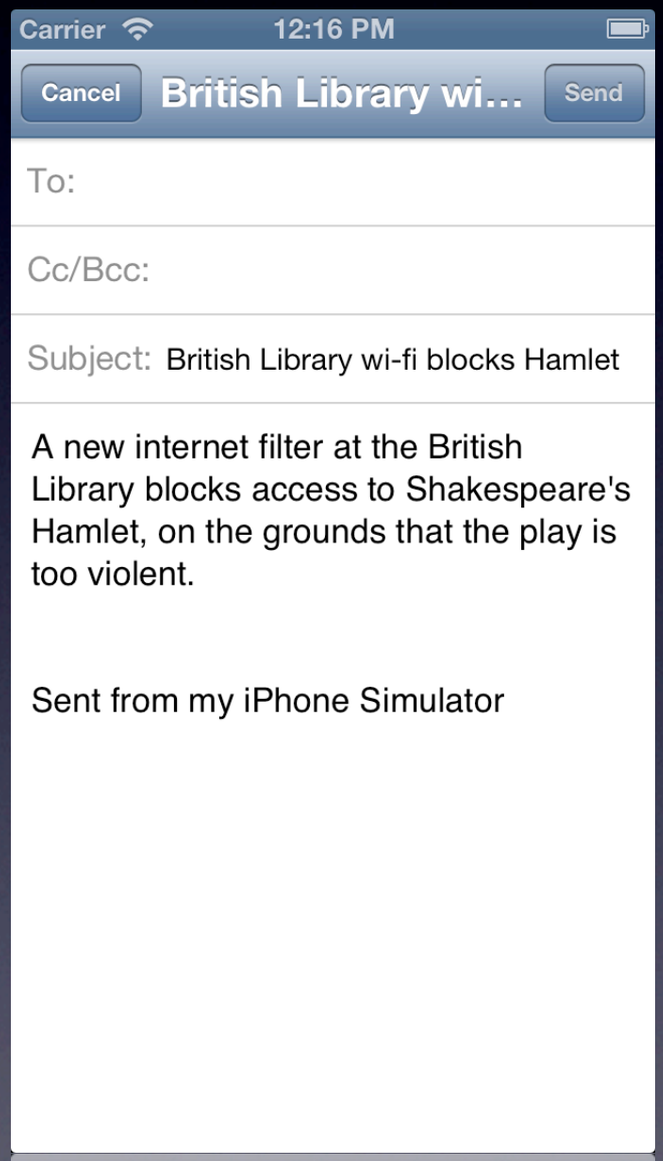
MFMailComposeViewController

Creating and sending emails

```
MFMailComposeViewController *mail =
[[MFMailComposeViewController alloc] init];

[mail setMailComposeDelegate:self];
[mail setSubject:@”Hello”];
[mail setMessageBody:@”Email sent from our app”
isHTML:NO];

[self presentViewController:mail animated:YES
completion:nil];
```

MFMailComposeViewController

Creating and sending emails



Implement the mail compose view controller

if time, implement search, pull to refresh

Demo

More!

Things we didn't have time for...

More!

Things we didn't have time for...

- Unit Testing

More!

Things we didn't have time for...

- Unit Testing
- Core Data

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase
- Inter-app Communication

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase
- Inter-app Communication
- Localisation

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase
- Inter-app Communication
- Localisation
- Accessibility

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase
- Inter-app Communication
- Localisation
- Accessibility
- Maps

More!

Things we didn't have time for...

- Unit Testing
- Core Data
- Local & Remote Notifications
- In-App Purchase
- Inter-app Communication
- Localisation
- Accessibility
- Maps
- Etc...

More!

Things we didn't have time for...

Documentation
http://developer.apple.com

Developer Forums
http://devforums.apple.com

Stack Overflow
http://stackoverflow.com

Online Tutorials
http://raywenderlich.com

Questions?