ITP 342
Mobile App Dev

Scenes & Segues

New Scene

- We want to create a new scene to allow the user to input data
  - For our project, we want the user to add a new quote and its author
- To create a new scene, we need a View Controller


Carrier
11:48 AM
Cancel
Add Quote
Save
Author


Carrier
11:49 AM
Cancel
Add Quote
Save
Today is a great day!
Trina Gregory
“Gregory”
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return

New Scene

- Create a new Single View iPhone app named **AddQuote**
- Refactor ViewController to **AddQuote**ViewController

Adding Text View & Text Field

- MainStoryboard.storyboard
- Drag a Text View into the view for the quote
  - IBOutlet: quoteTextView
- Drag a Text Field into the view for the author
  - IBOutlet: authorTextField

First Responder

- A small user experience touch
  - Make the keyboard automatically appear
  - Don't make users tap in the text view
- A focused text view becomes the first responder
  - All keyboard input goes to the text view

```
- (void) viewWillAppear: (BOOL) animated {
    [super viewWillAppear:animated];
    [self.quoteTextView becomeFirstResponder];
}
```

What We Get for Free

- Keyboard appears when focused
  - Doesn't seem to go away
- Autocorrect
- Cut, copy, & paste

What We Want

- When we hit return, we want
  - The keyboard will disappear
  - We'll log the text from the text view and the text field to the console
  - The text in the text view and the text field will be cleared

Delegation

- Delegates communicate information back to an "owner" for response or decision
- Customization without subclassing

Delegates

- The text field communicated with the view controller through Apple's **UITextFieldDelegate** protocol
  - Delegate mechanism lets us handle text input without subclassing UITextField
  - We'll handle the return key through a delegate method
- **UITextViewDelegate** protocol

Delegate Mechanism

- UITextField.h contains a list of delegate methods
- UITextField has a delegate property of type **UITextFieldDelegate**
- These methods are optional
  - If the method isn't implemented, it won't be called

UITextFieldDelegate Protocol

```
@protocol UITextFieldDelegate <NSObject>

@optional

- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField;
- (void)textFieldDidBeginEditing:(UITextField *)textField;
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField;
- (void)textFieldDidEndEditing:(UITextField *)textField;

- (BOOL)textField:(UITextField *)textField
      shouldChangeCharactersInRange:(NSRange)range
      replacementString:(NSString *)string;

- (BOOL)textFieldShouldClear:(UITextField *)textField;
- (BOOL)textFieldShouldReturn:(UITextField *)textField;

@end
```

Becoming a Delegate

- Adopt the UITextFieldDelegate protocol in AddQuoteViewController (.h or .m)
- Set our view controller as the text field's delegate
- Implement all required methods in the protocol
- Implement any optional methods of interest

Declaring the Protocol

- Add the protocol declaration in the implementation file to the class extension
- This says that this view controller can implement any of the protocol methods
- We don't have to declare these methods in the header

```
@interface AddQuoteViewController () <UITextFieldDelegate>

@property (weak, nonatomic) IBOutlet UITextView *quoteTextView;
@property (weak, nonatomic) IBOutlet UITextField *authorTextField;

@end
```

Connecting the Delegate

- Set the AddQuoteViewController as the text field's delegate
- We can do this in code or in the storyboard
- Storyboard
  - Click on text field and view Connections Inspector
  - Click from Delegate to Add Quote View Controller

UITextFieldDelegate Protocol

Overview

The `UITextFieldDelegate` protocol defines the messages sent to a text field delegate as part of the sequence of editing its text. All of the methods of this protocol are optional.

Tasks

Managing Editing

- `textFieldShouldBeginEditing:`
- `textFieldDidBeginEditing:`
- `textFieldShouldEndEditing:`
- `textFieldDidEndEditing:`

Editing the Text Field's Text

- `textField:shouldChangeCharactersInRange:replacementString:`
- `textFieldShouldClear:`
- `textFieldShouldReturn:`

Handle the Return Key

- Implement the `textFieldShouldReturn` method
- The way delegates are called is something like this:
  - if (a method with this name exists) { send this message }
  - if you spell the method name wrong, the method just won't be called

textFieldShouldReturn:

```
- (BOOL) textFieldShouldReturn: (UITextField *) textField {

    [textField resignFirstResponder];

    // Print the text to the console window
    NSLog(@"%@", textField.text);

    // Set the text to nothing
    textField.text = nil;
    return YES;
}
```

Navigation Bar

- MainStoryboard.storyboard
  - Drag a UINavigationBar from the library
  - Drag two bar button items into the navbar
    - Identifier – Save
    - Identifier – Cancel
  - Create an IBOutlet for the Save button
    - saveButton

Form Validation

- How do we check the text as it is typed?

```
textField:
shouldChangeCharactersInRange:
replacementString:
```

- Evaluate the post-typing string length
- Enable / disable the save button accordingly

Enable Save Button

```
-(void) enableSaveButtonForQuote: (NSString *) quoteText
    author: (NSString *) authorText {

    self.saveButton.enabled = (quoteText.length > 0 &&
                               authorText.length > 0);

}
```

TextField

```
- (BOOL) textField: (UITextField *) textField
    shouldChangeCharactersInRange: (NSRange) range
    replacementString: (NSString *) string {

    NSString *changedString = [textField.text
        stringByReplacingCharactersInRange: range
                                withString: string];

    [self enableSaveButtonForQuote: self.quoteTextView.text
                            author: changedString];

    // Do not actually replace the text field's text!
    // Return YES and let UIKit do it
    return YES;
}
```

Moving Data Between Scenes

- Text field sends its text to the Add Quote View Controller
- How do we get that text to the QuotesTableView Controller?
  - There are a few options…

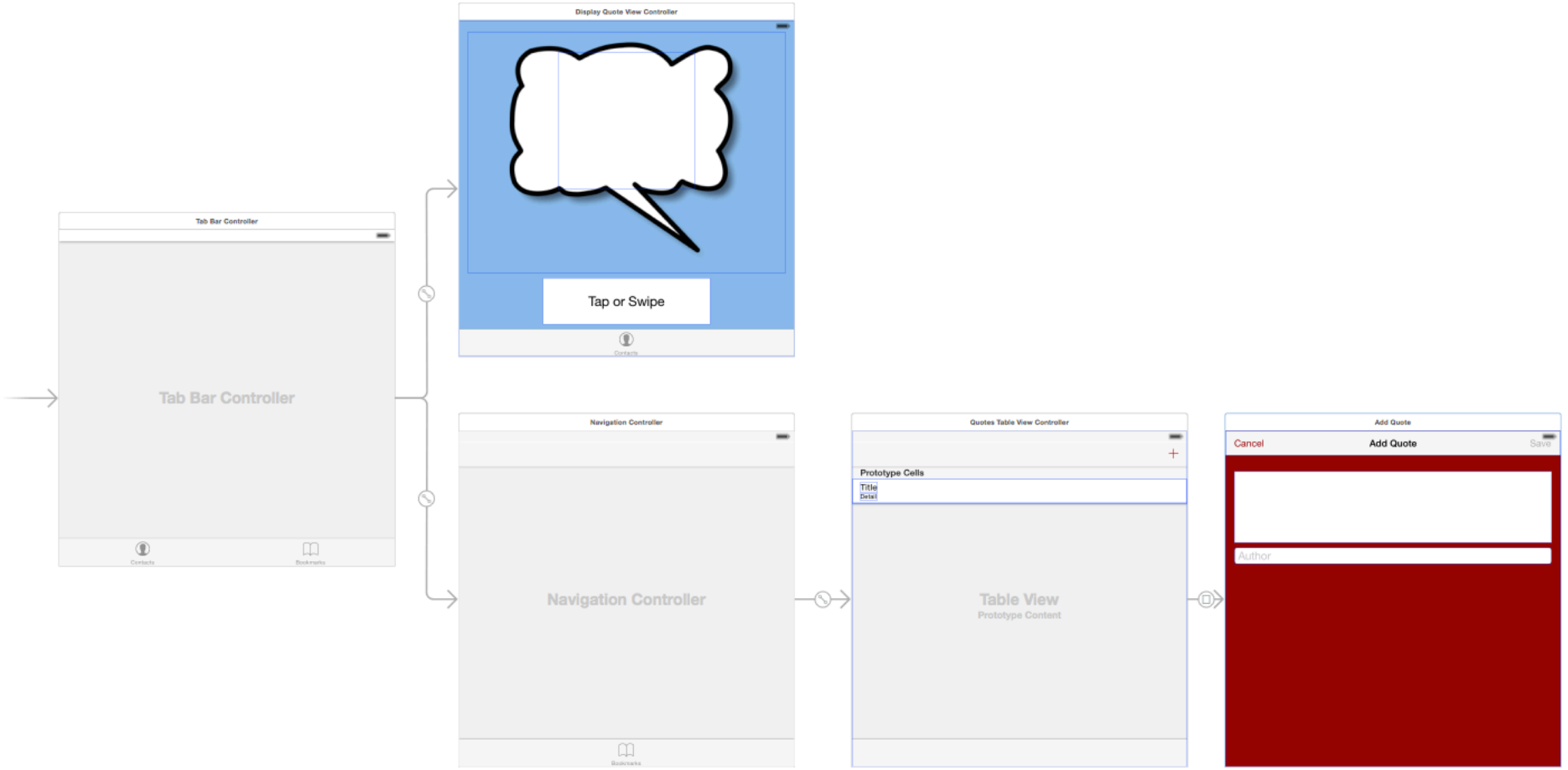
One (Wrong) Solution

- Set the parent scene as the text field's delegate
  - Too much coupling between view controllers
  - Deactivates all of our validation logic
    - AddQuoteViewController loses its delegate messages

Blocks as Callbacks

- We previously used blocks to handle animation
- Now we'll use them for communication between scenes (view controllers)
- AddQuoteViewController will accept a block to invoke when tapping Save or Cancel
  - It does not need to know about the model
- QuotesTableViewController will supply that block when the add quote view controller is presented

Storyboard


Display Quote View Controller
Tap or Swipe
Tab Bar Controller
Tab Bar Controller
Navigation Controller
Navigation Controller
Quotes Table View Controller
Prototype Cells
Title
Detail
Table View
Prototype Content
Add Quote
Cancel
Add Quote
Save
Author

Declaring a Completion Block

- Xcode helps you define blocks as objects
  - This lets your store them in properties for later use
- Type `typedef` in AddQuoteViewController.h
  - Xcode will autocomplete a block definition

```
// AddQuoteViewController.h

typedef returntype(^<#block name#>)(<#arguments#>);
```

Declaring a Completion Block

- Every block has
  - A return type
  - A type name
  - Zero or more arguments
- We set these based on our requirements

```
// AddQuoteViewController.h

typedef void(^AddQuoteCompletionHandler)(NSString *quote,
                                         NSString *author);
```

Designing a Block Callback

- Think about the problems you need to solve
- What should the input view pass through?
  - The newly-entered text, if any
- What does the input view need in return?
  - Nothing

Accepting a Completion Block

- The input view must hold on to the handler
- Create a property with the newly-defined type
  - Block properties should always be copy
- Invoke it at the relevant times

```
// AddQuoteViewController.h

typedef void(^AddQuoteCompletionHandler)(NSString *quote,
                                         NSString *author);

@interface AddQuoteViewController : UIViewController

@property (copy, nonatomic) AddQuoteCompletionHandler completionHandler;

@end
```

Cancel and Save Button

- Create IBActions for nav bar items
  - saveButtonTapped
  - cancelButtonTapped
- Implement methods
  - Invoke the completion handler (block)
    - Always check if the block exists first
    - Blocks are invoked as C functions
  - Pass nil for the cancel case

Invoking a Completion Block

```
// AddQuoteViewController.m

- (IBAction) cancelButtonTapped: (id)sender {
   // add code to make keyboard go away (resignFirstResponder)

   if (self.completionHandler) {
      self.completionHandler(nil, nil);
   }

   // add code to set the text view and text field strings to nil
}

- (IBAction) saveButtonTapped: (id)sender {
   // add code to make keyboard go away (resignFirstResponder)

   if (self.completionHandler) {
      self.completionHandler(self.quoteTextView.text,
                             self.authorTextField.text);
   }

   // add code to set the text view and text field strings to nil
}
```

Update

- Update `textFieldShouldReturn:` with the same code as the `saveButtonTapped:`

```
// AddQuoteViewController.m

-(BOOL) textFieldShouldReturn: (UITextField *) textField {
   [textField resignFirstResponder];

   if (self.completionHandler) {
      self.completionHandler(self.quoteTextView.text
                             self.authorTextField.text);
   }

   self.quoteTextView.text = nil;
   self.authorTextField.text = nil;

   return YES;
}
```

Blocks as Contracts

- Strict, consistent terms for using the input view
  - Same mechanism every time
- Consumers know how to get input back
  - No coupling
  - No interference
  - No confusion

Clean and Reusable

- AddQuoteViewController is now portable
- Any project can safely reuse it
  - Set a completion block to handle new text entry
  - No dependencies on internal implementation details

Back to Quotes

Integrate AddQuoteViewController into Quotes Project

Adding View Controller

- Open AddQuote project
  - View storyboard
  - Select the AddQuoteViewController
    - Should see the blue outline
  - Copy
- Open Quotes project
  - View storyboard
  - Paste

Adding AddQuoteViewController

- Open AddQuote project
  - Show the Project Navigator
  - Select the AddQuoteViewController.h and AddQuoteViewController.m files
  - Drag them over to the Quotes project
- Quotes project
  - In the options for adding the files
    - Check **Copy items into destination group's folder**
    - Select **Create groups for any added folders**
    - Check **Add to targets** for Quotes

Configure the Add Quote Controller

- There can only be one initial scene
- This new scene needs an Identifier
- In the Identity Inspector, for the Storyboard ID, add something like AddQuoteInputViewController

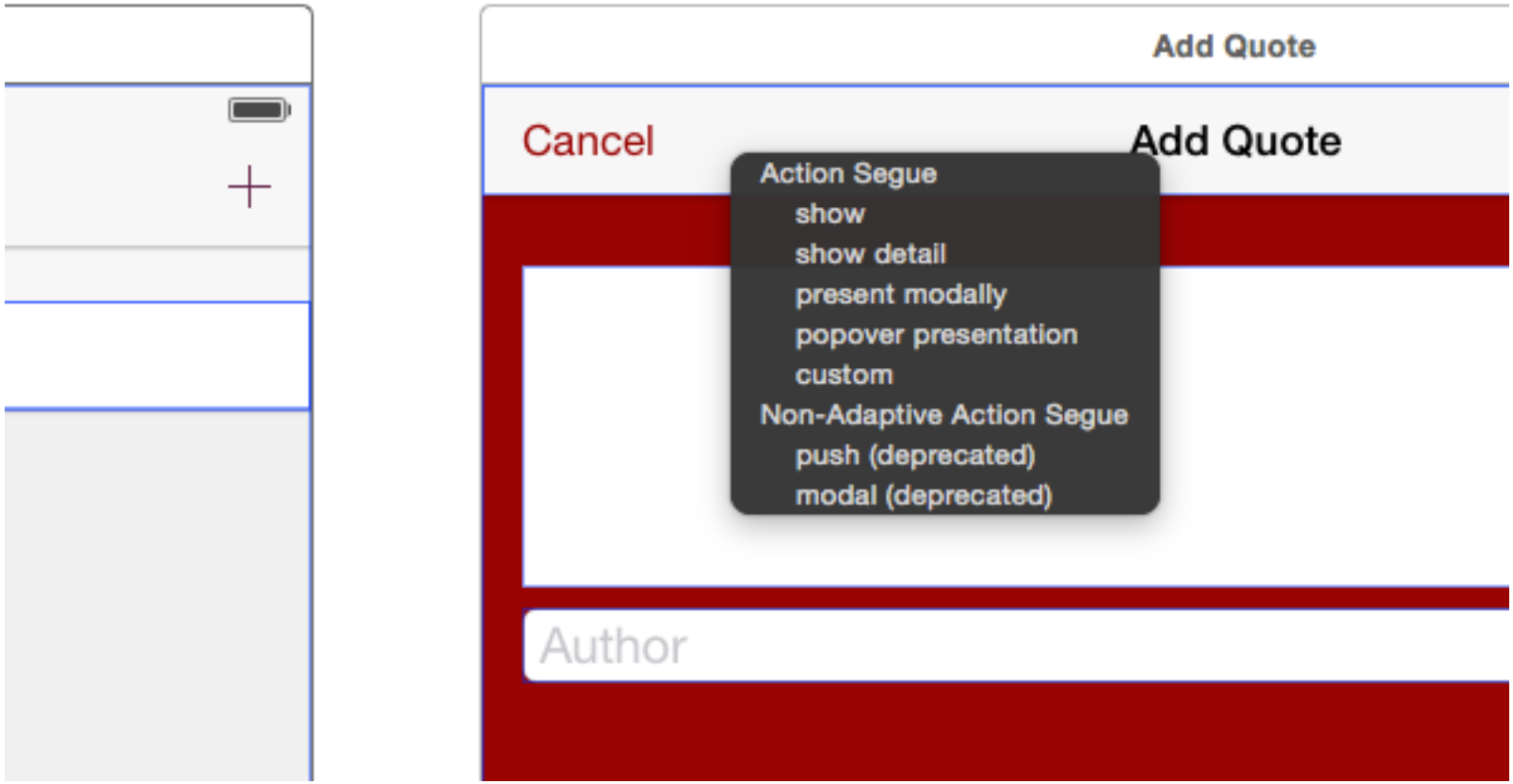
Add Table View Controller

- Add the Table View Controller to the Quotes project
- Add its corresponding ViewController class (.h & .m) to the Quotes project

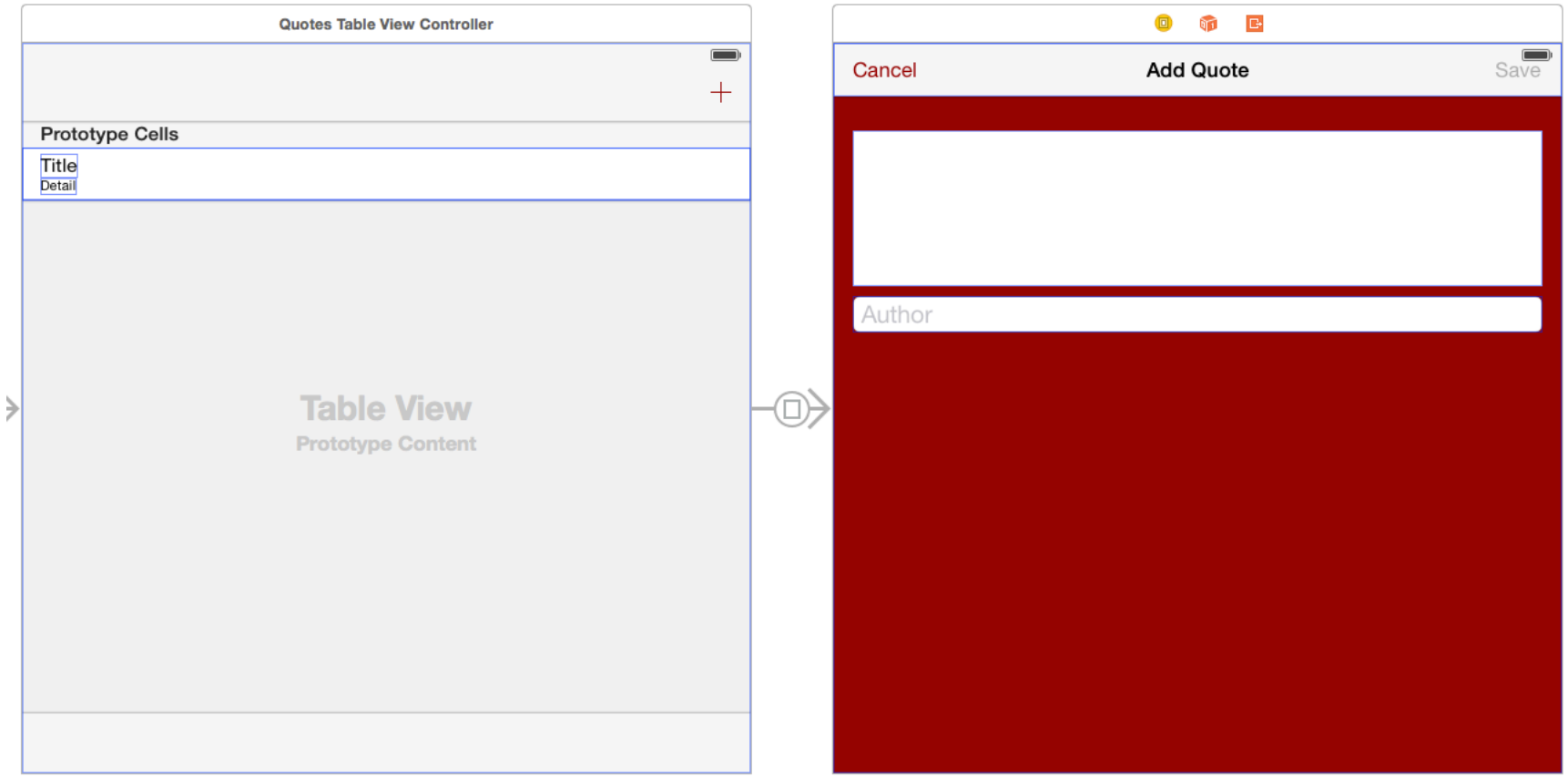
Connect

- Connect the Add Quote View Controller to the Table View Controller in the Storyboard using a segue
- Control-click drag from + on Table View Controller to Add Quote View Controller
- Select Present Modally

Add a Segue


Add Quote
Cancel
Add Quote
Action Segue
show
show detail
present modally
popover presentation
custom
Non-Adaptive Action Segue
push (deprecated)
modal (deprecated)
Author

Segue Object



What's a segue?

- A transition between storyboard scenes
- Two common segue styles:
  - Show (e.g. Push)

    - Left-to-right navigation
    - Adds another View Controller to the navigation stack
    - Used with Navigation Controllers
    - Xcode: Editor → Embed In → Navigation Controller
  - **Present Modally**

    - Full-screen cover
    - Allows one View Controller to present another View Controller modally

Life Cycle of a Segue

- Segue objects are instances of UIStoryboardSegue or one of its subclasses.
  - Your app never creates segue objects directly; they are always created on your behalf by iOS when a segue is triggered.
- Here's what happens:
  1. The destination controller is created and initialized.
  2. The segue object is created and its `initWithIdentifier: source: destination:` method is called.
  3. The source view controller's `prepareForSegue: sender:` method is called.
  4. The segue object's perform method is called. This method performs a transition to bring the destination view controller on-screen.
  5. The reference to the segue object is released, causing it to be deallocated.

Segue Identifiers

- Allow code to distinguish between segues
  - Very important as projects become more complex
- Pick a segue identifier right away
  - Set in Interface Builder under the Attributes Inspector
  - We'll use addQuote

Segues Explained

- The storyboard performs segues automatically
- … after calling `prepareForSegue:`
  - Inherited from UIViewController
  - Includes source and destination view controllers
  - Do any preparation or configuration here
    - Use identifier to distinguish between multiple segues

Providing a Completion Block

- Use `prepareForSegue:` to connect the scenes
  - Get the new view controller using [segue destinationViewController]
  - Set the completion handler
    - Insert new quote into the model
    - Have the table view reload its data
    - Dismiss the view controller

Prepare For Segue

```
// QuotesTableViewController.m

- (void) prepareForSegue: (UIStoryboardSegue *) segue
      sender: (id) sender {

   AddQuoteViewController *addQuoteVC =
      segue.destinationViewController;
   addQuoteVC.completionHandler = ^(NSString *quote,
                                    NSString *author) {
      if (quote != nil) {
         [self.model insertQuote:quote author:author];
         [self.tableView reloadData];
      }
      [self dismissViewControllerAnimated:YES completion:nil];
   };
}
```