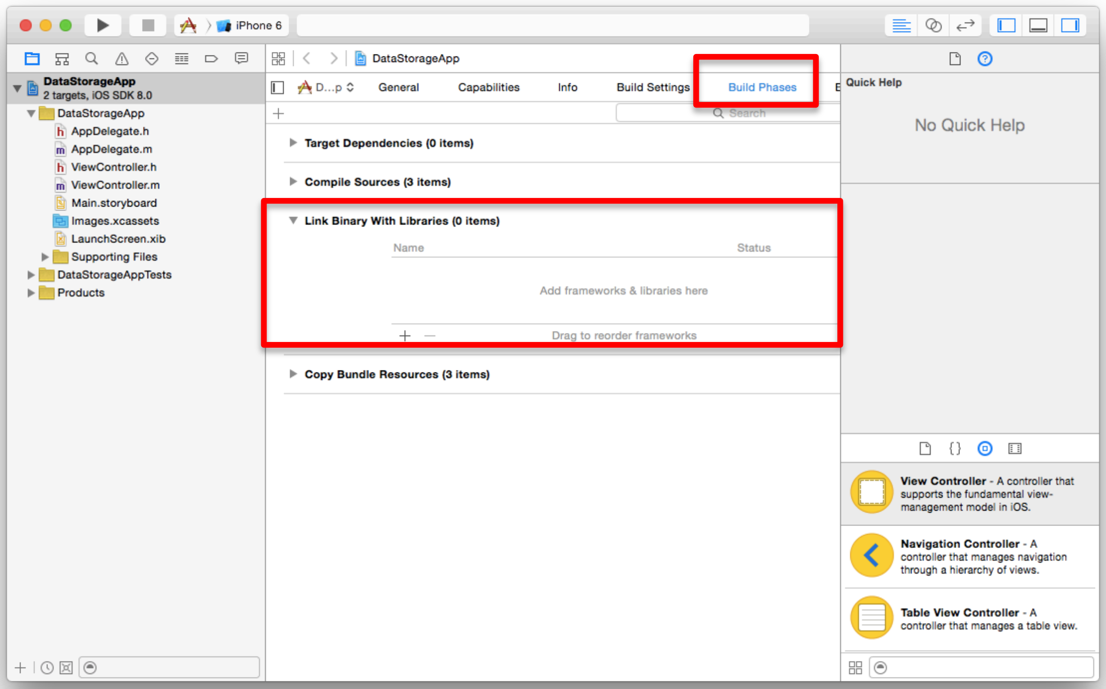
Practical 07: Using SQLite for Data Persistency

In this lab, we will make use of SQLite, a database engine used in iOS application to handle data.

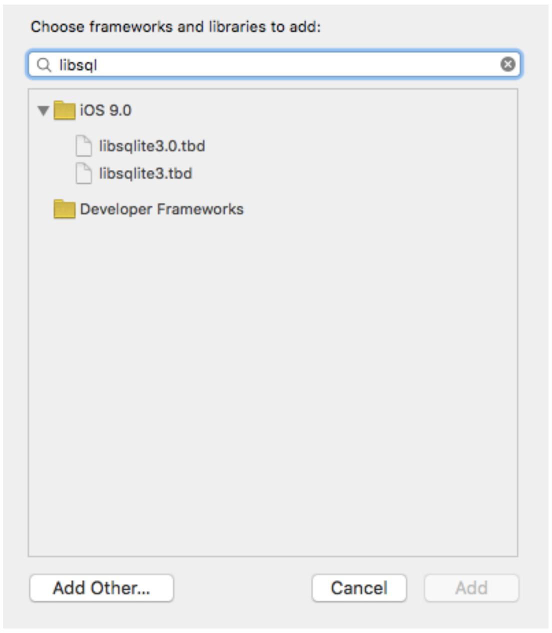
Section 1: Add SQLite framework

1. Download the MovieApp from Blackboard for Practical 7.
2. Using Xcode, open the MovieApp.
3. Select the **Build Phases** at the top bar of the editor pane. Expand the **Link Binary with Libraries**. A framework is a collection of related classes that you can add to target.


iPhone 6
DataStorageApp
2 targets, iOS SDK 8.0
DataStorageApp
AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files
DataStorageAppTests
Products
General
Capabilities
Info
Build Settings
Build Phases
Target Dependencies (0 items)
Compile Sources (3 items)
Link Binary With Libraries (0 items)
Name
Status
Add frameworks & libraries here
Drag to reorder frameworks
Copy Bundle Resources (3 items)
Quick Help
No Quick Help
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Table View Controller - A controller that manages a table view.

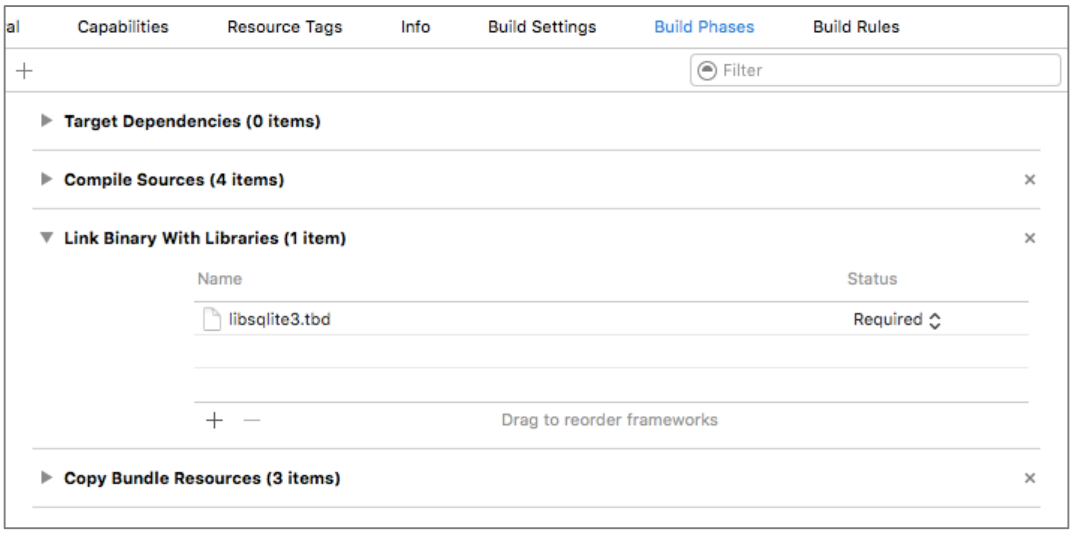
4. Click on **+** button to add new library to the target. Next, select **libsqlite3.tbd** framework and select '**Add**'.

   You will notice that there are 2 SQLite frameworks, **libsqlite3.tbd** and **libsqlite3.0.tbd**. **libsqlite3.tbd** refers to the latest version of SQLite. Use **libsqlite3.0.tbd only** when you want to tie to the exact version of SQLite.


Choose frameworks and libraries to add:

libsql

- iOS 9.0
  - libsqlite3.0.tbd
  - libsqlite3.tbd
- Developer Frameworks



Target Dependencies (0 items)

Compile Sources (4 items)

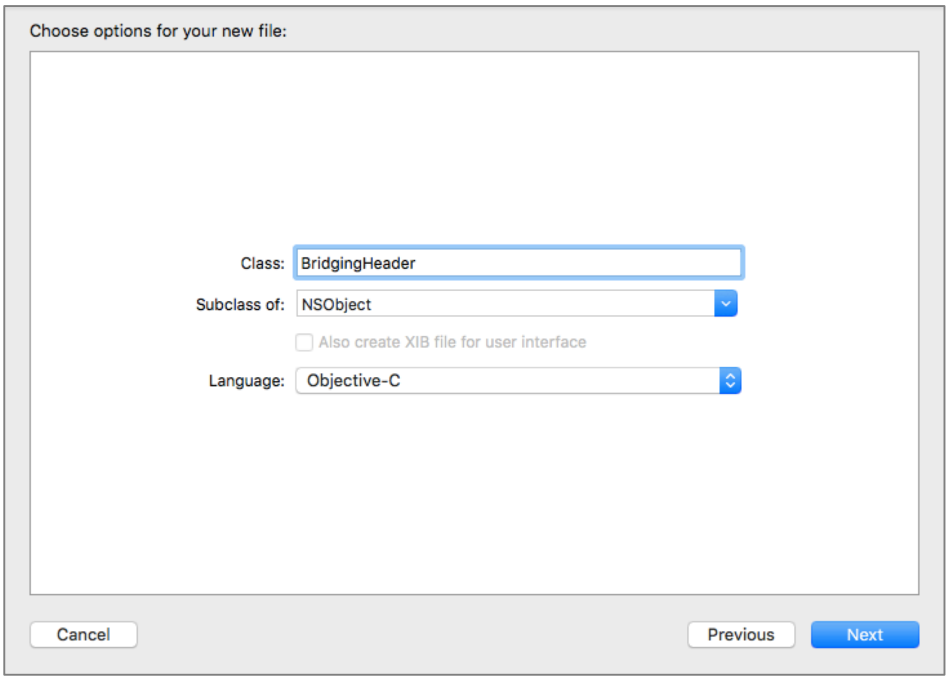
Link Binary With Libraries (1 item)

Name	Status
libsqlite3.tbd	Required

Copy Bundle Resources (3 items)

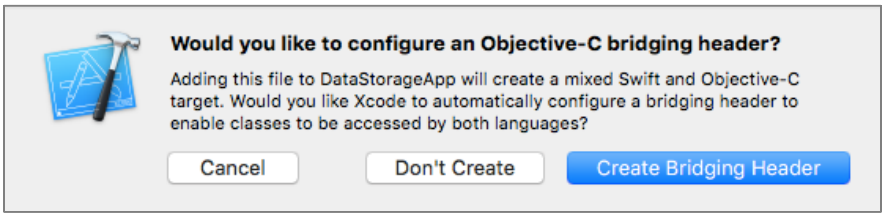
Section 2: Create a Bridging Header

5. Sqlite is a Objective-C library, and before we can use this library in Swift, we will have to create a Bridging Header. To do so, right click on the MovieApp (the one with the folder icon) and click New File. Select **Cocoa Touch** Class, and click Next. Then enter **BridgingHeader** as the Class Name, select **NSObject** as the Subclass, and set **Objective-C** as the language. Then click Next.


Choose options for your new file:

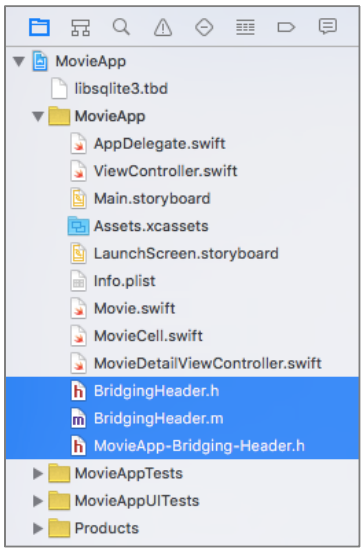
Class: BridgingHeader
Subclass of: NSObject
Also create XIB file for user interface
Language: Objective-C

6. After selecting the folder to create the file, in XCode will prompt to ask if you want to create a Bridging Header. Click **Create Bridging Header** to continue.


Would you like to configure an Objective-C bridging header?

Adding this file to DataStorageApp will create a mixed Swift and Objective-C target. Would you like Xcode to automatically configure a bridging header to enable classes to be accessed by both languages?

7. You should then see 3 files appear in your project – BridgingHeader.h, BridgingHeader.m, MovieApp-Bridging-Header.h


```
MovieApp
  libsqlite3.tbd
  MovieApp
    AppDelegate.swift
    ViewController.swift
    Main.storyboard
    Assets.xcassets
    LaunchScreen.storyboard
    Info.plist
    Movie.swift
    MovieCell.swift
    MovieDetailViewController.swift
    BridgingHeader.h
    BridgingHeader.m
    MovieApp-Bridging-Header.h
  MovieAppTests
  MovieAppUITests
  Products
```

8. Click on the MovieApp-Bridging-Header.h, and add the following line into the file. This allows you to include Objective-C libraries into your Swift code.

```
//
//  Use this file to import your target's public headers that
//  you would like to expose to Swift.
//

#include "sqlite3.h"
```

9. In the project navigator, right-click to add a **Swift** file and name it as **SQLiteDB.swift**. Remember to set the **Swift** as the language. The purpose of this wrapper is to make it easy for us to query the database, because without this, we will have to deal with a lot of low-level Objective-C functions and handling of Objective-C types in Swift.

10. Download the practicalfiles from Blackboard, open the SQLiteDB.swift file and copy the full set of codes into your SQLiteDB.swift file.

    (The version we are using for this practical is a modified copy of the one found at github)

Section 3: Implement DataManager

1. In this section, we will implement a **DataManager.swift** class. This class is responsible for all loading, updating and deleting of movies from the Sqlite database. But this class shall only call the simplified functions created in SQLiteDB to achieve that.

2. In the project navigator, right-click to add a **Cocoa Touch** Class with subclass of **NSObject** and name it as **DataManager**. Remember to set the **Swift** as the language.

3. Now, we are going to implement the **createDatabase** method In **DataManager.swift**, add the following codes.

```
class DataManager: NSObject {

    // Create a new database if it does not already exists
    //
    static func createDatabase()
    {
        SQLiteDB.sharedInstance.execute(sql:
            "CREATE TABLE IF NOT EXISTS " +
            "Movies ( " +
            "    movieID text primary key, " +
            "    movieName text, " +
            "    movieDescription text, " +
            "    runtime int, " +
            "    imagePath text )")
    }
}
```

4. In the **DataManager.swift**, we need to implement the **loadMovies** method.

```
class DataManager: NSObject {

    ...
    // Loads the list of movies from the database
    // and convert it into a [Movie] array
    //
    static func loadMovies() -> [Movie]
    {
        let movieRows = SQLiteDB.sharedInstance.query(sql:
            "SELECT movieID, movieName, " +
            "movieDescription, runtime, imagePath " +
            "FROM Movies")

        var movies : [Movie] = []
        for row in movieRows
        {
             movies.append(Movie(
                 id: row["movieID"] as! String,
                 name: row["movieName"] as! String,
                 description: row["movieDescription"] as! String,
                 runTime: row["runtime"] as! Int,
                 imagePath: row["imagePath"] as! String))
        }
        return movies;
    }
}
```

5. In the **DataManager.swift**, we need to implement the **insertOrReplaceMovie** method.

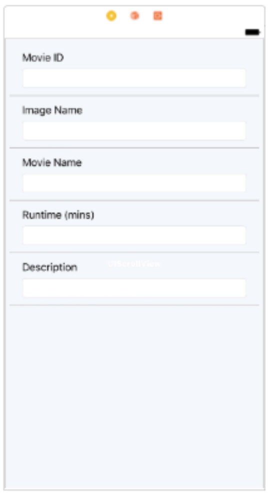
```
class DataManager: NSObject {

    ...

    static func insertOrReplaceMovie (movie: Movie)
    {
        // Since the strings in the Movie class are defined as
        // optionals, we must unwrap them. Instead of using ! to
        // unwrap, we will use a short-hand operator, called
        // the nil-coalesce operator '??'.
        //
        // This operator written this way:
        // X ?? N
        // Which actually means this:
        // if X == nil then use N otherwise use X
        //
        SQLiteDB.sharedInstance.execute(sql:
            "INSERT OR REPLACE INTO Movies (movieID, " +
            "movieName, movieDescription, runtime, imagePath) " +
            "VALUES (?, ?, ?, ?, ?)",
            parameters: [
                movie.movieID ?? "",
                movie.movieName ?? "",
                movie.movieDesc ?? "",
                movie.runtime,
                movie.imagePath ?? ""] )
    }
```

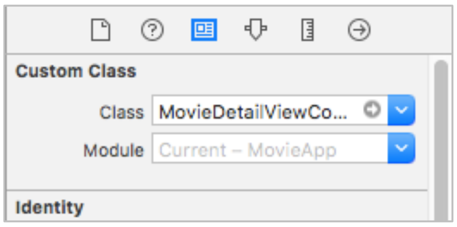
6. In the **DataManager.swift**, we need to implement the **deleteMovie** method.

```
class DataManager: NSObject {

    ...

    // Deletes an existing movie using the movie
    // object's movieID.
    //
    static func deleteMovie(movie: Movie)
    {
        SQLiteDB.sharedInstance.execute(
            sql: "DELETE FROM Movies WHERE MovieID = ?",
            parameters: [movie.movieID])
    }
}
```

Section 4: Build the User Interface

1. On the **Main.storyboard**, note that the a Movie Detail View Controller is already created for you. The MovieDetailViewController class is already created for you, but go to the XCode Identity Inspector to connect the MovieDetailViewController class to the Storyboard.

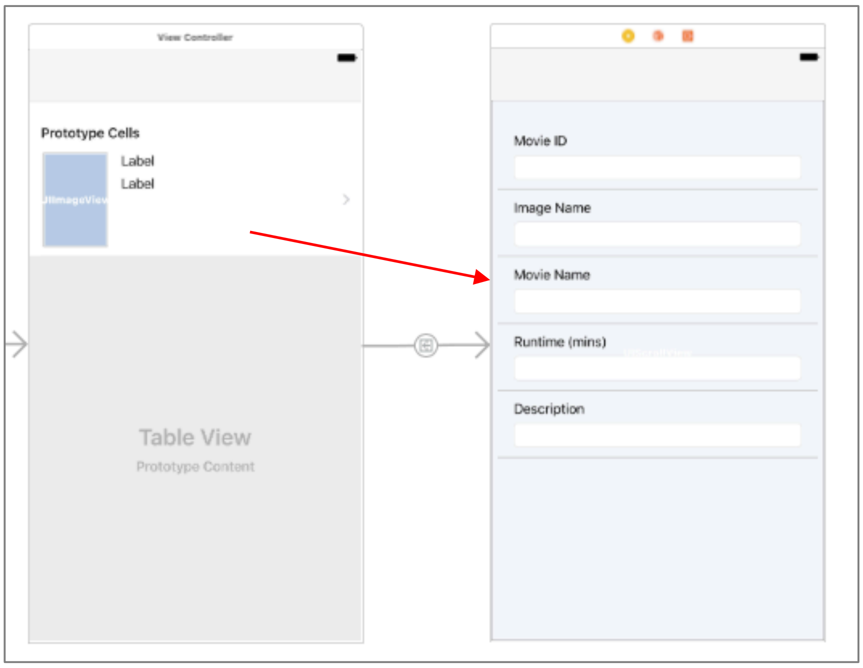



Custom Class
Class MovieDetailViewCo...
Module Current – MovieApp
Identity

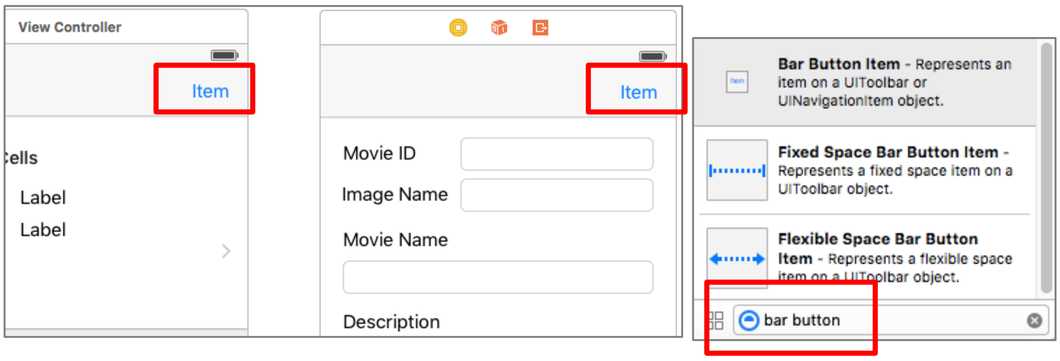
2. Create and connect IBOutlets to all respective TextFields.

```
class MovieDetailViewController: UIViewController
{
    @IBOutlet weak var imageNameTextField: UITextField!

    @IBOutlet weak var movieIDTextField: UITextField!

    @IBOutlet weak var movieNameTextField: UITextField!

    @IBOutlet weak var runtimeField: UITextField!

    @IBOutlet weak var descriptionTextField: UITextField!

    var movieItem : Movie!

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

    }

}
```

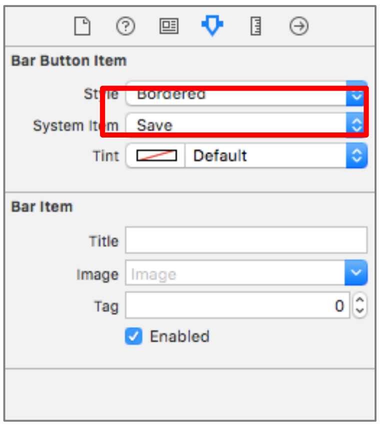
3. Ctrl-click the Table View Cell and connect to your Movie Detail View Controller using the relationship segue **Show**. This tells iOS to push the Movie Detail View Controller on to the top whenever the user taps on the Movie in the Table View.



4. In the **Main.storyboard**, add a Bar Button each to the Root View Controller's and the Movie Detail View Controller's navigation bar.


View Controller
Item
Cells
Label
Label
Item
Movie ID
Image Name
Movie Name
Description
Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.
Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.
Flexible Space Bar Button Item - Represents a flexible space item on a UIToolbar object.
bar button

5. Select the Bar Button in the Root View Controller, go to the Attributes Inspector and set the **System Item** to **Add**. Then, select the bar button in the Movie Detail View Controller, go to the Attributes Inspector and set the **System Item** to **Save**.


Bar Button Item

Style: Bordered

System Item: Save

Tint: Default

Bar Item

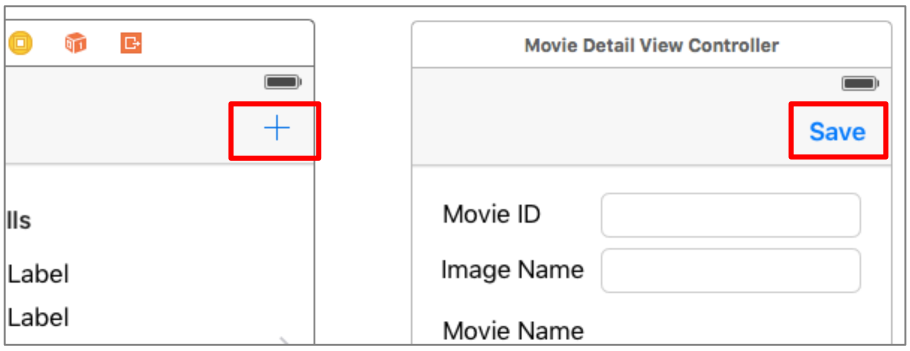
Title:

Image: Image

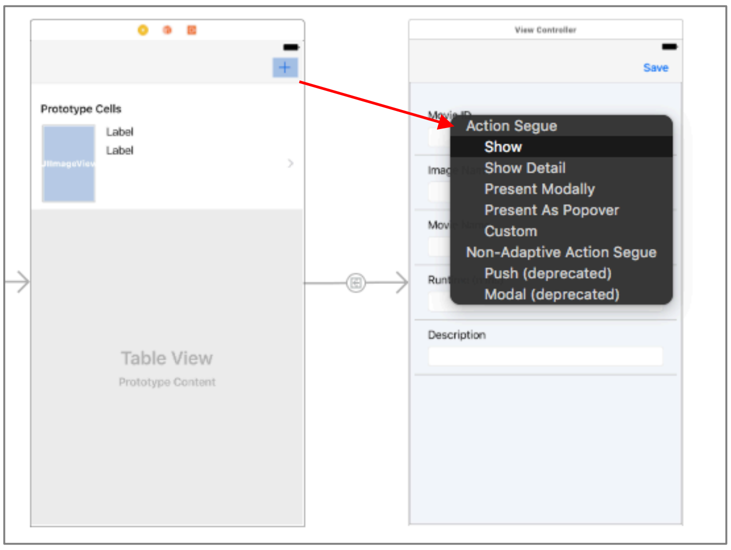
Tag: 0

Enabled ✓

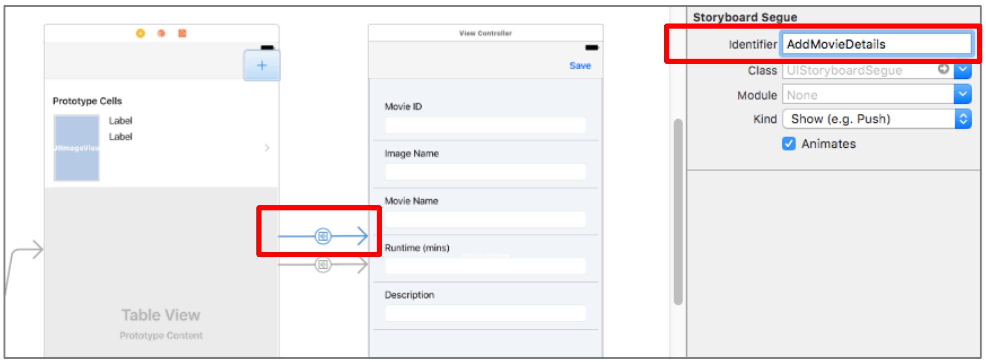
6. Your Storyboard should look like that eventually:



7. Ctrl-Drag from the Add Bar Button to the Movie Detail View Controller, the select the “**Show**” segue.

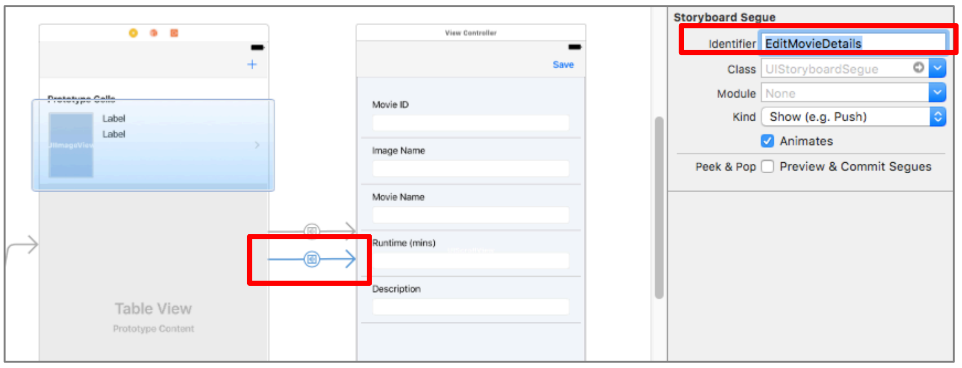


8. You will see 2 segues now coming out from the Root View Controller to the Movie Detail View Controller. One is triggered from the user tapping on a Movie Cell, the other is triggered from the user tapping on the **Add** button. Now, select whichever segue that represents the one coming from the **Add** button. In the Attributes inspector, enter “**AddMovieDetails**” as the segue’s identifier.



Storyboard Segue
Identifier AddMovieDetails
Class UIStoryboardSegue
Module None
Kind Show (e.g. Push)
Animates ✓

9. Click on the other segue (the one triggered by the user tap on the Movie Cell). In the Attributes inspector, enter "**EditMovieDetails**" as the segue's identifier.



Storyboard Segue
Identifier: EditMovieDetails
Class: UIStoryboardSegue
Module: None
Kind: Show (e.g. Push)
Animates: ✓
Peek & Pop: Preview & Commit Segues

10. In **MovieDetailViewController.swift**, create an IBAction **saveButtonPressed**, and connect it to the **Save Bar Button** in your Storyboard.

```
@IBAction func saveButtonPressed(sender: UIButton)
{
}
```

11. Try to run the application in your iOS Simulator.

Section 5: Code the Behaviors into the View Controllers

1. In the **ViewController.swift**, modify the **viewDidLoad** function to create the database tables (if they do not already exist), and also to load up the list of movies from the database.

```
class ViewController: UIViewController,
    UITableViewDelegate, UITableViewDataSource {
    override func viewDidLoad() {
        super.viewDidLoad()

        // When the application first starts,
        // create the database with the necessary tables
        //
        DataManager.createDatabase()

        // Loads up the list of movies from the database
        //
        movieList = DataManager.loadMovies()
    }
}
```

2. In the **ViewController.swift**, modify the **prepare** function create a new and empty Movie object and assign it to the destination Movie Detail View Controller when the user clicks on the **Add** button.

```
class ViewController: UIViewController,
    UITableViewDelegate, UITableViewDataSource
{

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        if(segue.identifier == "EditMovieDetails")
        {
            let detailViewController = segue.destination as!
                MovieDetailViewController
            let myIndexPath = self.tableView.indexPathForSelectedRow

            if(myIndexPath != nil)
            {
                let movie = movieList[(myIndexPath! as
                    NSIndexPath).row]
                detailViewController.movieItem = movie
            }
        }

        // Here we check for the AddMovieDetails segue,
        // which is trigger by the user clicking on the +
        // button at the top of the navigation bar
        //
        if(segue.identifier == "AddMovieDetails")
        {
            let detailViewController = segue.destination as!
                MovieDetailViewController
            let movie = Movie(id: "", name: "", description: "",
                runTime: 0, imagePath: "")
            detailViewController.movieItem = movie
        }

    }
}
```

3. In the **ViewController.swift**, modify the **tableView(tableView: commitEditingStyle: forRowAtIndexPath:)** function to delete the selected record from the database, and update the movieList array.

```
class ViewController: UIViewController,
    UITableViewDelegate, UITableViewDataSource
{

    func tableView(tableView: UITableView,
        commitEditingStyle editingStyle: UITableViewCellEditingStyle,
        forRowAtIndexPath indexPath: NSIndexPath)
    {
        if (editingStyle == UITableViewCellEditingStyle.Delete)
        {
            let movie = movieList[indexPath.row]

            // Here we delete the movie from the database
            // and reload the movieList array again.
            //
            DataManager.deleteMovie(movie: movie)
            movieList = DataManager.loadMovies()

            tableView.deleteRowsAtIndexPaths(
                [indexPath],
                withRowAnimation: UITableViewRowAnimation.fade)
        }
    }

}
```

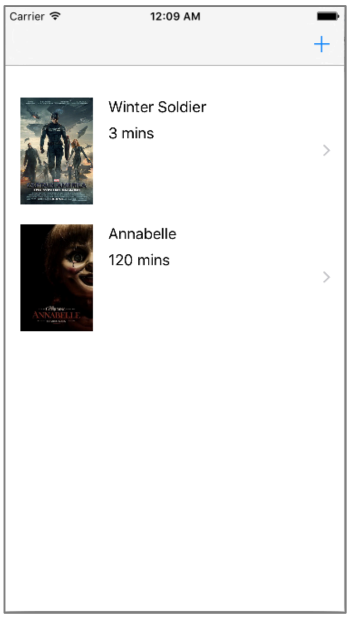
4. In the MovieDetailViewController.swift, modify the viewWillAppear function to bind the Movie object to the text fields on the screen.

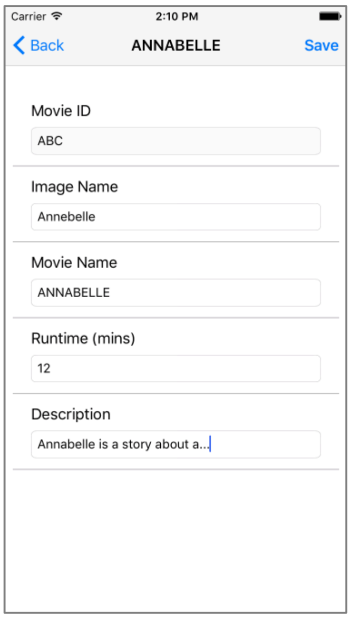
```
class MovieDetailViewController: UIViewController
{
    var movieItem : Movie!

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        if movieItem != nil
        {
            // Assign the fields from the movie object
            // to all the text fields on the screen.
            //
            // Here we use the special nil-coalesce operator again.
            //
            movieIDTextField.text = movieItem!.movieID ?? ""
            movieNameTextField.text = movieItem!.movieName ?? ""
            descriptionTextField.text = movieItem!.movieDesc ?? ""
            imageNameTextField.text = movieItem!.imagePath ?? ""
            runtimeField.text = "\(movieItem!.runtime)"

            // If this is an existing movie (that means the movie
            // ID will have some value), disable the user from
            // modifying the movie ID.
            //
            if movieItem!.movieID != ""
            {
                movieIDTextField.isEnabled = false
            }

            self.navigationItem.title = movieItem!.movieName
        }
    }
}
```

5. In **MovieDetailViewController.swift**, we modify the **saveButtonPressed** IBAction to handle what happens when the user clicks on the Save button.

```
class MovieDetailViewController: UIViewController
{

    @IBAction func saveButtonPressed(sender: UIButton)
    {

        // Validate to ensure that all the fields are
        // entered by the user. If not we show an alert.
        //
        if movieIDTextField.text == "" ||
           movieNameTextField.text == "" ||
           descriptionTextField.text == "" ||
           imageNameTextField.text == "" ||
           runtimeField.text == ""
        {
            let alert = UIAlertController(
                title: "Please enter all fields",
                message: "",
                preferredStyle:
                UIAlertControllerStyle.alert)

            alert.addAction(UIAlertAction(title: "OK",
                style: UIAlertActionStyle.default,
                handler: nil))

            self.present(alert, animated: true, completion: nil)

            return
        }

        // assign the data entered by the user into
        // the movie object
        //
        movieItem!.movieID = movieIDTextField.text
        movieItem!.movieName = movieNameTextField.text
        movieItem!.movieDesc = descriptionTextField.text
        movieItem!.imagePath = imageNameTextField.text

        let runtime = Int(runtimeField.text!)
        movieItem!.runtime = runtime != nil ? runtime! : 0

        // Execute the SQL to insert the data
        // into the database
        //
        DataManager.insertOrReplaceMovie(movie: movieItem!)

        // Calls the root view controller's table view to
        // to refresh itself.
        //
        var viewControllers = self.navigationController?
            .viewControllers
        let parent = viewControllers?[0] as! ViewController
        parent.movieList = DataManager.loadMovies()
        parent.tableView?.reloadData()

        // close this view controller and pop back out to
        // the one that shows the list of movies.
        //
        self.navigationController?.popViewController(animated: true)

    }

}
```

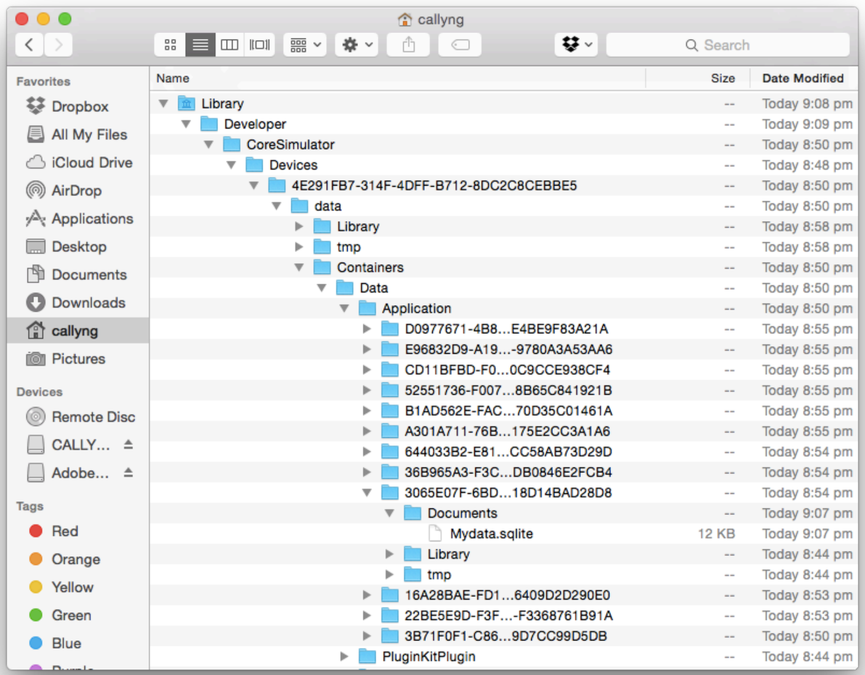
6. Run your application in the iOS simulator. You should be able to add new movies and view the list of movies that you previously added in the UITableView. You are also able to swipe and delete a movie from the UITableView.





Movie ID
ABC
Image Name
Annebelle
Movie Name
ANNABELLE
Runtime (mins)
12
Description
Annabelle is a story about a...

7. After the database is created successfully, it can be found in the Document folder of your application sandbox on the iPhone Simulator in the **~/Library/Developer/CoreSimulator/Devices/<App_ID>/data/Containers/Data/Application/<App_ID>/Documents/** folder.



Name	Size	Date Modified
Library	--	Today 9:08 pm
Developer	--	Today 9:09 pm
CoreSimulator	--	Today 8:50 pm
Devices	--	Today 8:48 pm
4E291FB7-314F-4DFF-B712-8DC2C8CEBBE5	--	Today 8:50 pm
data	--	Today 8:50 pm
Library	--	Today 8:58 pm
tmp	--	Today 8:58 pm
Containers	--	Today 8:50 pm
Data	--	Today 8:50 pm
Application	--	Today 8:50 pm
D0977671-4B8...E4BE9F83A21A	--	Today 8:55 pm
E96832D9-A19...-9780A3A53AA6	--	Today 8:55 pm
CD11BFBD-F0...0C9CCE938CF4	--	Today 8:55 pm
52551736-F007...8B65C841921B	--	Today 8:55 pm
B1AD562E-FAC...70D35C01461A	--	Today 8:55 pm
A301A711-76B...175E2CC3A1A6	--	Today 8:55 pm
644033B2-E81...CC58AB73D29D	--	Today 8:54 pm
36B965A3-F3C...DB0846E2FCB4	--	Today 8:54 pm
3065E07F-6BD...18D14BAD28D8	--	Today 8:54 pm
Documents	--	Today 9:07 pm
Mydata.sqlite	12 KB	Today 9:07 pm
Library	--	Today 8:44 pm
tmp	--	Today 8:44 pm
16A28BAE-FD1...6409D2D290E0	--	Today 8:53 pm
22BE5E9D-F3F...-F3368761B91A	--	Today 8:53 pm
3B71F0F1-C86...9D7CC99D5DB	--	Today 8:50 pm
PluginKitPlugin	--	Today 8:44 pm

Note: If you are unable to see the folder Library in your user directory, you can run this command in the Terminal command to unhide the folder.

```
chflags nohidden ~/Library/
```

Challenge

1. Ensure that the user is not allowed to add a new movie if one with the same movie ID already exists.

2. Show the image of the movie after entering the Image Name.

3. Consider adding a button “Sort” at the top of the Root View Controller (where the Table View of movies it). When this button is clicked, push another view with the following 4 buttons to allow users to sort the movie list by selecting from one of the following options:

   Sort by name (ascending)

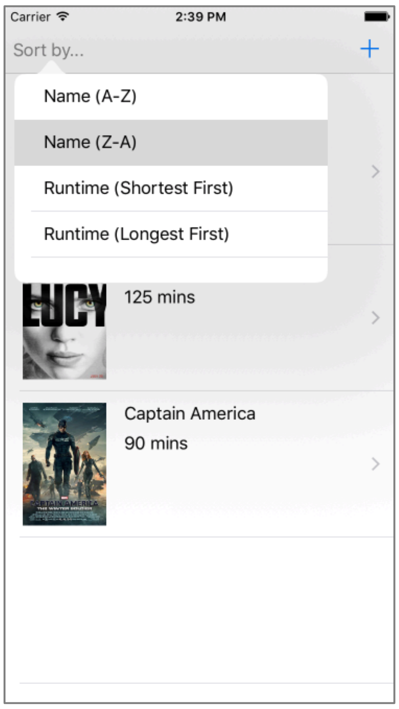
   Sort by name (descending)

   Shortest to longest running time

   Longest to shortest running time.

   Once the sorting order is selected, go back to the movie list screen and reload the movie list using new sorting order.

Try using a Popover to allow the user to select the sorting function, as shown in the following screen shot.



You may refer to the following URL for understanding how to achieve Popovers. Note that the codes are written for Swift 1.2, and you may have to adapt slightly it for Swift 3.

https://richardallen.me/2014/11/28/popovers.html