audreywelch / **ios-albums**
forked from LambdaSchool/ios-albums

No description, website, or topics provided.

Edit

Manage topics

2 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is even with LambdaSchool:master.

SpencerCurtis Update README.md

README.md	Update README.md
exampleAlbum.json	Add project instructions and example JSON

Clone with HTTPS | Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/audreywelch/ios-al

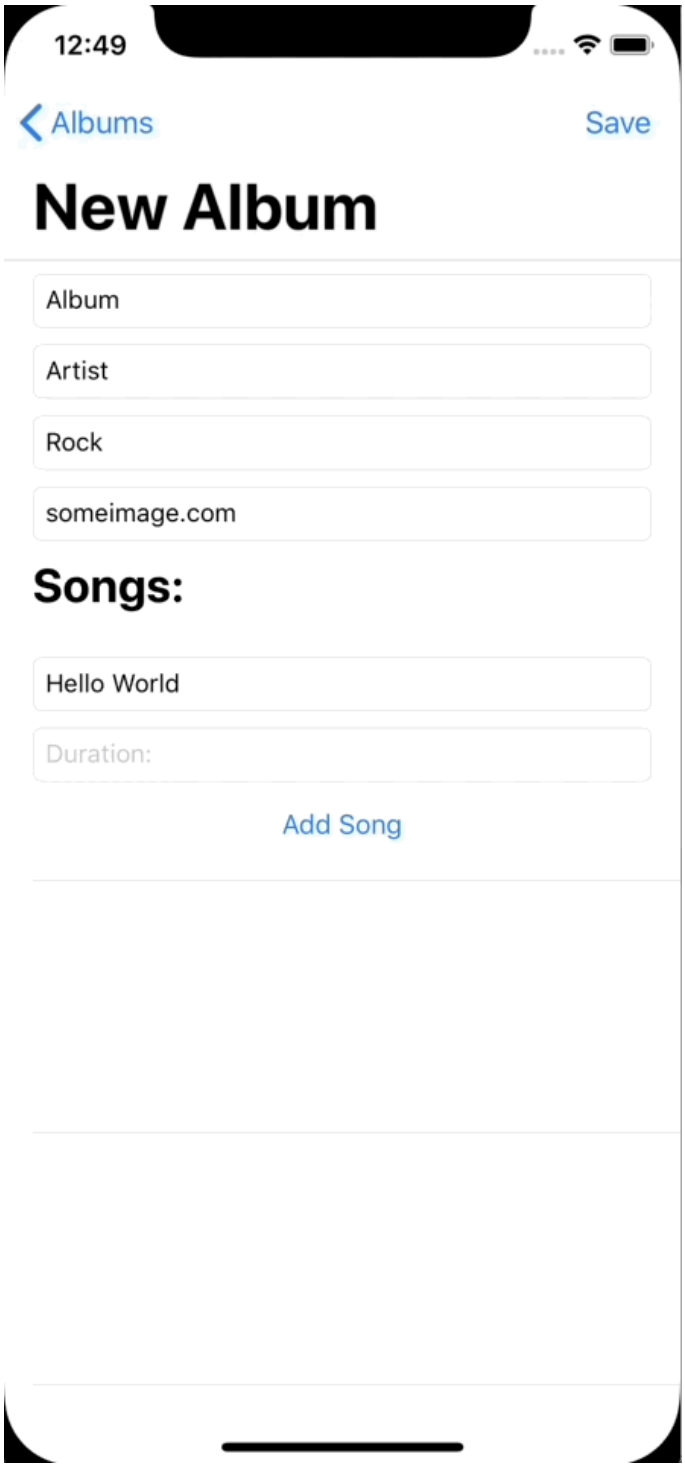
Open in Desktop | Download ZIP

README.md

Albums

Introduction

Please look at the screen recording below to know what the finished project should look like:



Instructions

Please fork and clone this repository. This repository does not have a starter project but, so create one inside of the cloned repository folder. This repository does contain a file with example JSON that you will need in order to set up your model objects.

Part 1 - Storyboard Setup

AlbumsTableViewController

1. Remove the view controller scene the Main.storyboard comes with.
2. Add a table view controller scene and embed it in a navigation controller. Set the navigation controller as the initial view controller.
3. Create a Cocoa Touch subclass of `UITableViewController` called `AlbumsTableViewController` and set this table view controller's class to it.
4. Add a bar button item to the table view controller's navigation item. Change its system item to "Add".
5. Set the prototype cell's style to "Subtitle", and give the cell a reuse identifier.

AlbumDetailViewController

1. Add a second table view controller scene. Create a show segue from the first table view controller's cell, and another show segue from its bar button item.
2. Add a navigation item to this view controller scene. Then add a bar button item and set its system item to "Save".
3. Add a `UIView` as the detail table view's header view. In this header view, add 4 text fields. The text field's placeholder text should show to the user that they must supply:
   - The album's name.
   - The artist.
   - The genres (separated by commas).
   - URLs to the cover art (separated by commas).
4. Create a Cocoa Touch subclass of `UITableViewController` called `AlbumDetailTableViewController` and set this table view controller's class to it. Create outlets from the four text fields, and an action from the bar button item.
5. Each cell will represent a song. In the prototype cell, add two text fields. The first should take in the song's title, and the second should take in the duration of the song.
6. Add a button in the cell also. This will add the song to the album. Set its title to "Add Song".
7. Create a Cocoa Touch subclass of `UITableViewCell` called `SongTableViewCell` and set this cell's class to it. Create outlets from the text fields and the button. Also create an action from the button.

Part 2 - Manual Codable Conformance

The whole purpose of this project is to help you understand how `Codable` works under the hood when it automatically synthesizes conformance to the protocol, as well as giving you the ability to implement it yourself when necessary.

1. Create a new Swift file called "Album.swift". Create a model object called `Album`.
2. Using the JSON provided in the "ExampleAlbum.json" file, parse the JSON to figure out what properties your `Album` model should have.
   - **NOTE:** Create a second model object called `Song` for the array of Songs. Your `Song` should have 3 properties.
   - **NOTE:** You must have a property for every value the JSON contains. There should be 6 properties.
3. Adopt just `Decodable` on both model objects for now. Starting with `Song`, implement the required `init(from decoder: Decoder) throws` initializer.
   - The goal of implementing this initializer yourself is to avoid using nested structs, and keeping your model object "flat".
   - Use the example JSON to figure out how to decode it into your model objects.
   - Assume that all properties in the model objects are **not optional** and are variables.
4. Add the "exampleAlbum.json" file to your project. Make sure you check "Copy file(s) if needed", and add it to your target.
5. Create a new Swift file called "AlbumController.swift". Create an `AlbumController` class.

6. Create a function in the `AlbumController` called `testDecodingExampleAlbum()`. This should:
    - Get the JSON data from the "exampleAlbum.json" file. (`Data(contentsOf: URL)`)
    - Try to decode the JSON using JSONDecoder just like you would if you got this data from an API.
    - Check for errors. This is important because it will help you make sure you've correctly implemented the `init(from decoder: Decoder) throws` initializer in your model objects by giving you an error about what you have potentially done wrong.
7. Run this function in the `AppDelegate`. Make sure you don't get any errors when decoding the example JSON before you move on.
8. Back in the "Album.swift" file, now adopt `Codable` in both model objects.
9. Implement the `encode(to encoder: Encoder) throws` function. This function should encode the JSON back into its original nested state (i.e. the encoded JSON should match the structure of the example JSON exactly).
10. Create a function in the `AlbumController` called `testEncodingExampleAlbum()`. Copy and paste the code from the `testDecodingExampleAlbum()` method. Then simply try encoding the newly decoded `Album`. Again, check for errors to make sure you're encoding correctly.

Part 3 - AlbumController

Now you will add the functionality to fetch Albums from and send them to an API. In the `AlbumController`, create and the following:

1. An `albums: [Album]` variable that will be the data source for the application
2. A `baseURL: URL` property. Create or use an existing Firebase Database for the base URL.
3. A function called `getAlbums`. It should have a completion handler that takes in an optional `Error`. This function should perform a `URLSessionDataTask` that fetches the albums from the `baseURL`, decodes them, and sets the `albums` array to the decoded albums. **Note:** You should decode the JSON data as `[String: Album].self` here.
4. A function called `put(album: Album)`. This should use a `URLSessionDataTask` to PUT the album passed into the function to the API. Add the album's identifier to the base URL so it gets put in a unique location in the API.
5. A function called `createAlbum`. It should take in the necessary properties as parameters in order to initialize a new `Album`. Create an `Album` from the method parameters, then append it to the `albums` array. Then call the `put(album: Album)` method so the new `Album` gets saved to the API.
6. A function called `createSong`. It should take in the necessary properties as parameters to be able to initialize a `Song`. The function should return a `Song`. In the method, simply initialize a new song from the method parameters and return it.
7. A function called `update`. This should take in an `Album` and a parameter for each of the `Album` object's properties that can be changed (This should be every property). Update the values of the `Album` parameter, then send those changes to the API by calling the `put(album: Album)` method.

Test the `createAlbum` method by either using the example JSON or passing in your own `Album` information. Make sure it gets sent to the API, and in the correct structure.

Part 4 - Wiring Up The Views

In the `AlbumsTableViewController`:

1. Create an `albumController: AlbumController?` variable.
2. In the `viewDidLoad`, call the `getAlbums` method of the `albumController`. Reload the table view in its completion closure.
3. Implement the required `UITableViewDataSource` methods. The table view should display the albums in the `albumController`'s `albums` array. The cells should show the album's name and artist.

4. Go to the `AlbumDetailTableViewController`. Add the following:
    - An `albumController: AlbumController?` variable.
    - An `album: Album?` variable.
5. Back in the `AlbumsTableViewController`, implement the `prepare(for segue: ...)` method. If the segue is triggered from the bar button item, it should pass the `albumController`. If it's triggered from tapping a cell, it should pass the `albumController` and the `Album` that corresponds to the cell.

In the `SongTableViewCell`:

1. Create a `song: Song?` variable.
2. Create an `updateViews` method. It should:
    - Check if the song exists. If it does, set the text fields' text to the corresponding values of the `Song`.
    - If the song exists, also hide the button.
3. Implement the `prepareForReuse()` method. Clear the text fields' text, and unhide the button.
4. Create a class protocol above or below the `SongTableViewCell` class called `SongTableViewCellDelegate`. It should have a single function: `func addSong(with title: String, duration: String)`.
5. Create a `weak var delegate: SongTableViewCellDelegate?`.
6. In the action of the bar button item, call `delegate?.addSong(with title: ...)`. Pass in the unwrapped text from the text fields for the parameters to the method.

In the `AlbumDetailTableViewController`:

1. Create a `tempSongs: [Song] = []` array. This will be used to temporarily hold the songs the user adds until they tap the Save bar button item to save the album (or changes to it).
2. Create an `updateViews` method. It should
    - Take the appropriate values from the `album` (if it isn't nil) and place them in the corresponding text fields. You can use the `.joined(separator: ...)` method to combine the urls and genres into strings.
    - Set the title of the view controller to the album's name or "New Album" if the album is nil.
    - Set the `tempSongs` array to the album's array of `Songs`.
3. Call `updateViews()` in the `didSet` property observer of the `album` variable, and in the `viewDidLoad()`. Remember to make sure the view is loaded before trying to set the values of the outlets or the app will crash.
4. Adopt the `SongTableViewCellDelegate` protocol.
5. Add the `addSong` method from the delegate you just adopted. In it:
    - Create a `Song` using the `createSong` method in the `albumController`.
    - Append the song to the `tempSongs` array
    - Reload the table view
    - Call `tableView.scrollToRow(at: IndexPath, ...)` method. You will need to manually create an `IndexPath`. Use 0 for the section and the `count` of the `tempSongs` for the row.
6. Implement the `numberOfRowsInSection` method using the `tempSongs` array. Return the amount of items in the array plus one. This will allow there to be an empty cell for the user to add a new song to.
7. Implement the `cellForRowAt` method. Set this table view controller as the cell's `delegate`.
8. Implement the `heightForRowAt` method. Set the cell's height to something that looks good. Account for the cells whose buttons will be hidden, and the last cell whose button should be unhidden. In the screen recording, the hidden button cells' height is 100, and the last cell's height is 140.
9. Finally, in the action of the "Save" bar button item:
    - Using optional binding, unwrap the text from the text fields.
    - If there is an album, call the `update(album: ...)` method, if not, call the `createAlbum` method using the

unwrapped text, and the `tempSongs` array.
    - Pop the view controller from the navigation controller.

Go Further

- Add a method to fetch the image(s) from an album. Add a collection view either directly to the `AlbumDetailTableViewController` or in a container view in it that shows the album cover(s).
- Add the ability to search for albums and songs using an external API such as the iTunes Search API. (Note: the iTunes Search API does not include song durations, so you may have to include a default value for the duration.