IvanCaldwell / **Art-Gallery**
forked from LambdaSchool/Art-Gallery

No description, website, or topics provided.

Edit

Manage topics

7 commits | 1 branch | 0 releases | 2 contributors

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

This branch is even with LambdaSchool:master. Pull request Compare

erica Revising README and removing skeleton Latest commit 64137b9 on Oct 3

PaintingAssets.xcassets	Revising README and removing skeleton	a month ago
README.md	Revising README and removing skeleton	a month ago

README.md

Art Gallery

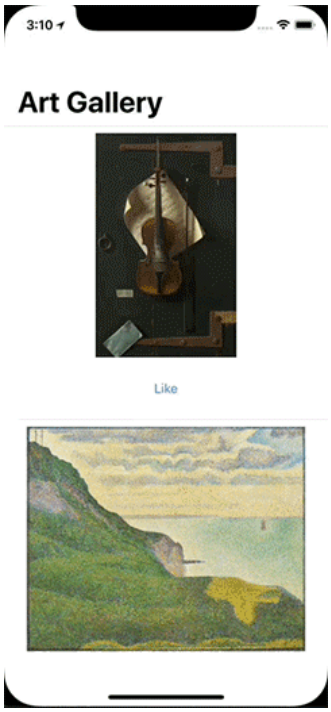
This project helps you practice today's objective goals. Working on this project helps you:

- understand and explain what a protocol is and common scenarios for their use
- define a custom protocol, and make a class or struct conform to it
- understand and explain the purpose of UITableViewController
- use a regular UIViewController to display a UITableView
- create a custom UITableViewCell
- understand and explain the delegate pattern and why it is used

You will repeat and expand upon the material from this morning's lecture. Be sure to start from scratch and use your notes and source as reference material only. Avoid cutting and pasting from the morning work.

Introduction

Your Art Gallery application displays a list of paintings, enabling users to pick the items they like. A fully built application looks like this:



Setting up the project

The project practices using a standalone table view set apart and distinct from using a UITableViewController. You need to create a view controller in Interface Builder, add a table view as its child, and stretch it to completely fill its parent with 20-point gaps on the two sides.

Create classes for:

- PaintingViewController: Your primary view controller class, renamed from ViewController
- PaintingTableViewCell: The cells for your table view

Assign these to the controller and table view in Interface Builder using the identity inspector.

Embed your controller inside a Navigation Controller and name your View Controller's title to "Lambda Gallery"

Set up your Controller

Use IB to connect your table view to a `tableView` outlet. Implement `viewWillAppear` to reload your table data.

Set up your model

Create three files:

- Painting.swift: The basic painting instance type
- PaintingTableViewCellDelegate.swift: A custom protocol for communicating between your cell and your model
- PaintingModel.swift: The model that will store your application state.

In Painting.swift create a Painting struct. Include two fields: an `image` constant (of `UIImage`) and a variable `isLiked` Boolean. Add an initializer. Default `isLiked` to false in the initializer.

Your protocol defines a single `tappedLikeButton(on:)` method declaration. Remember this is a protocol, so you do no implementation here. Your protocol must conform to `class` for it to work with this example:

```
protocol PaintingTableViewCellDelegate: class {
```

```
    func tappedLikeButton(on cell: PaintingTableViewCell)
}
```

Create a `PaintingModel` class that descends from NSObject and conforms to both `UITableViewDataSource` and `PaintingTableViewCellDelegate`. Add a weak `tableView` optional variable, so you can reference your main table view controller. You need this reference to be able to respond to the `tappedLikeButton` call. It's slightly a mess because this app isn't built the way you'd normally build one (with a UITVC).

Create a variable property called `paintings`, which is an array of `Painting`. Set its initial value to an empty array.

Build an initializer that uses the numbers 1 through 12 to load assets into an Image Array.

Set up your Table Delegate

In Interface Builder create a new object, set its identity to PaintingsModel and set it as your table view's delegate.

In `PaintingsModel.swift`, add the three core table delegate methods. In `numberOfSections`, set your weak tableView property.

Establish a `reuseIdentifier` property called `cell` and name the prototype cell in Interface Builder.

Set up your interface

Create a vertical stack view, add it to your cell and constrain the view to the top and bottom, leaving a 20 point gap on both sides. Stretch right and left.

Add an image view and a button. Set the button's text size to 14 points. Set the Image View height to 256. Set the view content mode to aspect fit.

Set the stack view's separation to 20.

Setting up the cell

Connect outlets for `portraitView` and `likeButton` in your table view cell. Add a weak delegate variable of `PaintingTableViewCellDelegate?` type.

Use IB to connect the button to the cell class to create `toggledAppreciation`. Have it relay the action to the delegate to let it handle `tappedLikeButton(on: self)`

Finish your Model code

Implement `tappedLikeButton(on:)` to toggle the paintings `isLiked` property. Change the text on the cell's button to match this.

Finish setting up your cell code in `cellForRoad` by setting the `likeButton` 's text and adding the painting image to the `portraitView`.

Go Farther

- Create an information screen to present when you tap on an image view that shows information (you can just use "blah blah" or lorem ipsum for this). Present it with a "show" segue. You do not need to prepare for the segue as you'll be showing static content each time.
- Change your segue from "show" to present it modally, allowing the information screen to slide up. You'll need to add a done button that allows the user to dismiss the model view when they are done reading.

Stretch:

- Stylize the portraits by rounding the corners of each portrait and adding a border

- Make the cells dynamically resize according to the image's height.

Progress Dashboard

Download checked items