Sample Code

Creating a Sticker App with a Custom Layout

Expand on the Messages sticker app template to create an app with a customized user interface.

Download

iOS 14.0+ | iPadOS 14.0+ | Xcode 13.2+

Overview

Xcode provides a template for creating sticker pack apps, but the template's presets limit customization. The Stickers sample project demonstrates how to create your own sticker pack app using an application template instead of the Xcode template, and how to customize the app's layout.

The Xcode project consists of two targets:

- An iOS application that has no functionality in this sample, but usually contains your main application.
- An iMessage Extension target that contains all logic for building and displaying the stickers.

When the iMessage Extension target first initializes, the `viewDidAppear` method of the `MessagesViewController` configures and displays the `UICollectionView`, which contains several sections of stickers, as well as actions to create new ones.

Customize the Sticker Layout

The Stickers app uses a `UICollectionView` to display its stickers. While the Xcode template only allows for two, three, or four items per row, the sample displays five items per row using a

`UICollectionViewCompositionalLayout`.

The following code shows an example of a custom layout:

```
// Creates a custom layout using section providers.
private static func createLayout() -> UICollectionViewLayout {
    let sectionProvider = { (sectionIndex: Int, layoutEnvironment: NSCollectionLayou
        let headerFooterSize = NSCollectionLayoutSize(
            widthDimension: .fractionalWidth(1.0),
            heightDimension: sectionIndex < 2 ? .absolute(64) : .estimated(44))

        switch sectionIndex {
            // texts
            case 0:
                let textSection = MessagesViewController.oneItemSection
                let textSectionHeader = NSCollectionLayoutBoundarySupplementaryItem(
                    layoutSize: headerFooterSize, elementKind: TextSupplementaryVie
                textSection.boundarySupplementaryItems = [textSectionHeader]
                return textSection
            // pictures
            case 1:
                let pictureSection = MessagesViewController.twoItemSection
                let pictureSectionHeader = NSCollectionLayoutBoundarySupplementaryIt
                    layoutSize: headerFooterSize, elementKind: PictureSupplementaryV
                pictureSection.boundarySupplementaryItems = [pictureSectionHeader]
                return pictureSection
            // animals
            case 2:
                let animalSection = MessagesViewController.fiveItemSection
                let titleSectionHeader = NSCollectionLayoutBoundarySupplementaryItem
                    layoutSize: headerFooterSize, elementKind: TitleSupplementaryVie
                animalSection.boundarySupplementaryItems = [titleSectionHeader]
                return animalSection
            // trees
            case 3:
                let treeSection = MessagesViewController.fiveItemSection
                let titleSectionHeader = NSCollectionLayoutBoundarySupplementaryItem
                    layoutSize: headerFooterSize, elementKind: TitleSupplementaryVie
                treeSection.boundarySupplementaryItems = [titleSectionHeader]
                return treeSection
            // days
            case 4:
                let daySection = MessagesViewController.oneItemSection
```

```
                let titleSectionHeader = NSCollectionLayoutBoundarySupplementaryItem
                    layoutSize: headerFooterSize, elementKind: TitleSupplementaryVie
                daySection.boundarySupplementaryItems = [titleSectionHeader]
                return daySection
            default: return nil
        }
    }
    return UICollectionViewCompositionalLayout(sectionProvider: sectionProvider)
}
```

The Stickers app uses `UICollectionViewDiffableDataSource` to populate the `UICollectionView`. The sections are `String` values that denote each section and the values are `MSSticker` instances. The app creates the stickers and applies them to the data source, which displays the stickers to the user.

The following code shows how the Stickers app populates a data source:

```
var snapshot = NSDiffableDataSourceSnapshot<SectionType, MSSticker>()
snapshot.appendSections(SectionType.allCases)

snapshot.appendItems(["animal-1", "animal-2", "animal-3", "animal-4"].compactMap({ (
    guard let url = Bundle.main.url(forResource: name, withExtension: "png") else {
        return nil
    }
    return try? MSSticker(contentsOfFileURL: url, localizedDescription: name)
}), toSection: .animals)

snapshot.appendItems(["tree-1", "tree-2", "tree-3", "tree-4"].compactMap({ (name) ->
    guard let url = Bundle.main.url(forResource: name, withExtension: "png") else {
        return nil
    }
    return try? MSSticker(contentsOfFileURL: url, localizedDescription: name)
}), toSection: .trees)

snapshot.appendItems(["Happy " + formatter.string(from: Date())].compactMap({ (name)
    let label = UILabel()
    label.text = name
    guard let image = label.image(),
          let data = image.pngData(),
          let baseURL = FileManager.default.urls(for: .documentDirectory, in: .userD
          let url = URL(string: "\(baseURL.appendingPathComponent("\(name).png"))"),
          (try? data.write(to: url)) != nil else {
        print("write failed")
```

```
        return nil
    }
    return try? MSSticker(contentsOfFileURL: url, localizedDescription: name)
}), toSection: .days)

datasource.apply(snapshot, animatingDifferences: true, completion: nil)
```

Create a Custom Cell

The `UICollectionView` includes a custom `UICollectionViewCell`. The cells use a custom `UIContentConfiguration` to provide an `MSStickerView`, each of which contains an `MSSticker` instance.

The following code shows how to provide data to a custom cell using `UICollectionView.CellRegistration`:

```
let cellRegistration = UICollectionView.CellRegistration<UICollectionViewCell, MSSti
    cell.contentConfiguration = CustomContentConfiguration(sticker: item)
}

datasource = UICollectionViewDiffableDataSource<SectionType, MSSticker>(
    collectionView: collectionView, cellProvider: { (collectionView, indexPath, item
        return collectionView.dequeueConfiguredReusableCell(using: cellRegistration,
    }
)

let titleHeaderRegistration = UICollectionView.SupplementaryRegistration
<TitleSupplementaryView>(elementKind: TitleSupplementaryView.reuseIdentifier) {
    (supplementaryView, string, indexPath) in
    supplementaryView.label.text = SectionType.allCases[indexPath.section].rawValue
}

let textHeaderRegistration = UICollectionView.SupplementaryRegistration
<TextSupplementaryView>(elementKind: TextSupplementaryView.reuseIdentifier) {
    (supplementaryView, string, indexPath) in
    supplementaryView.field.delegate = self
}

let pictureHeaderRegistration = UICollectionView.SupplementaryRegistration
<PictureSupplementaryView>(elementKind: PictureSupplementaryView.reuseIdentifier) {
    (supplementaryView, string, indexPath) in
    supplementaryView.button.addTarget(self, action: #selector(self.pickImage), for:
```

The Stickers app creates a cell with a custom `UIContentConfiguration`, as the following code demonstrates:

```
// Set up the internal view and apply the custom configuration.
init(configuration: CustomContentConfiguration) {
    super.init(frame: .zero)
    setupInternalViews()
    apply(configuration: configuration)
}

private func setupInternalViews() {
    stickerView.translatesAutoresizingMaskIntoConstraints = false
    addSubview(stickerView)
    NSLayoutConstraint.activate([
        stickerView.leadingAnchor.constraint(equalTo: leadingAnchor),
        stickerView.trailingAnchor.constraint(equalTo: trailingAnchor),
        stickerView.topAnchor.constraint(equalTo: topAnchor),
        stickerView.bottomAnchor.constraint(equalTo: bottomAnchor)
    ])
}

private var appliedConfiguration = CustomContentConfiguration()

// Apply the custom configuration.
private func apply(configuration: CustomContentConfiguration) {
    guard appliedConfiguration != configuration else { return }
    appliedConfiguration = configuration
    stickerView.sticker = appliedConfiguration.sticker
}
```

See Also

Custom iMessage app interface

{} IceCreamBuilder: Building an iMessage Extension

Allow users to collaborate on the design of ice cream sundae stickers.

`class` `MSMessagesAppViewController`

The principal view controller for iMessage apps.

`protocol` MSMessagesAppTranscriptPresentation

A protocol that provides support for displaying live messages in the transcript of the Messages app.

`enum` MSMessagesAppPresentationStyle

Presentation styles that describe your iMessage app's appearance.