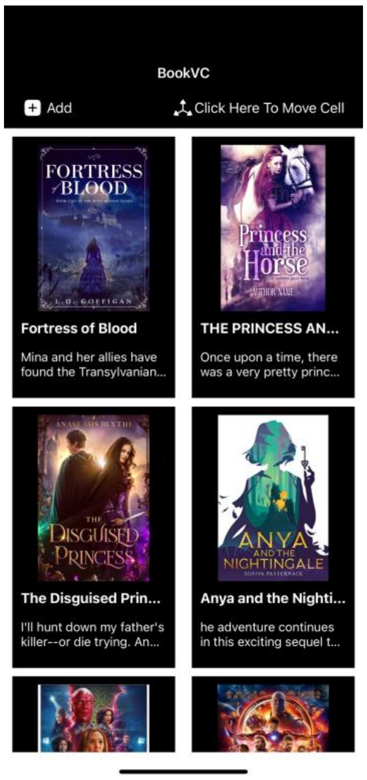
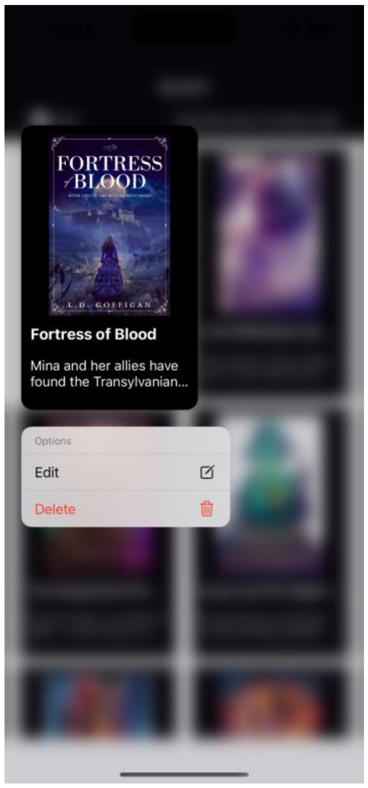
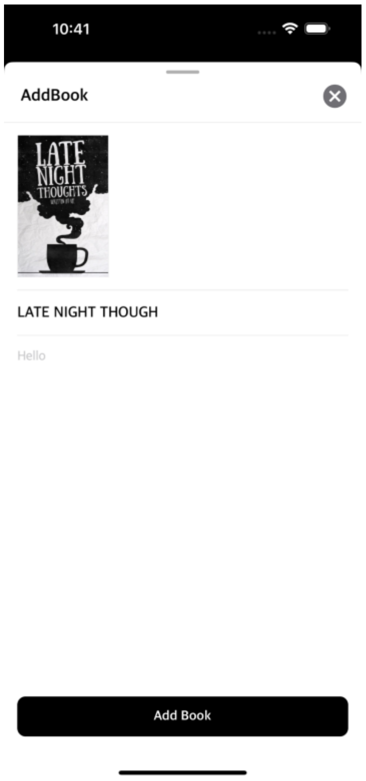
Content

1. Introduction

An object that manages an ordered collection of data items and presents them using customizable layouts.

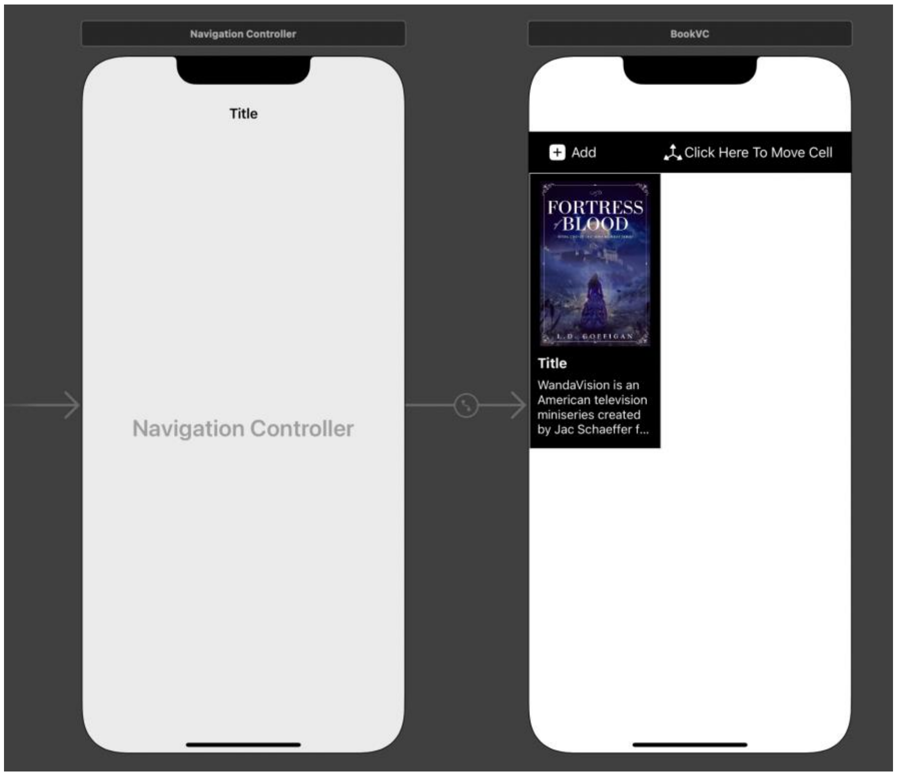
2. Sample project 1: show how to add, edit, delete, detail, moveCell in collectionView [Book → Self → BookSB & Book → BookAdd → BookAddSB]


BookVC
Add
Click Here To Move Cell
Fortress of Blood
Mina and her allies have found the Transylvanian...
THE PRINCESS AN...
Once upon a time, there was a very pretty princ...
The Disguised Prin...
I'll hunt down my father's killer--or die trying. An...
Anya and the Nighti...
he adventure continues in this exciting sequel t...


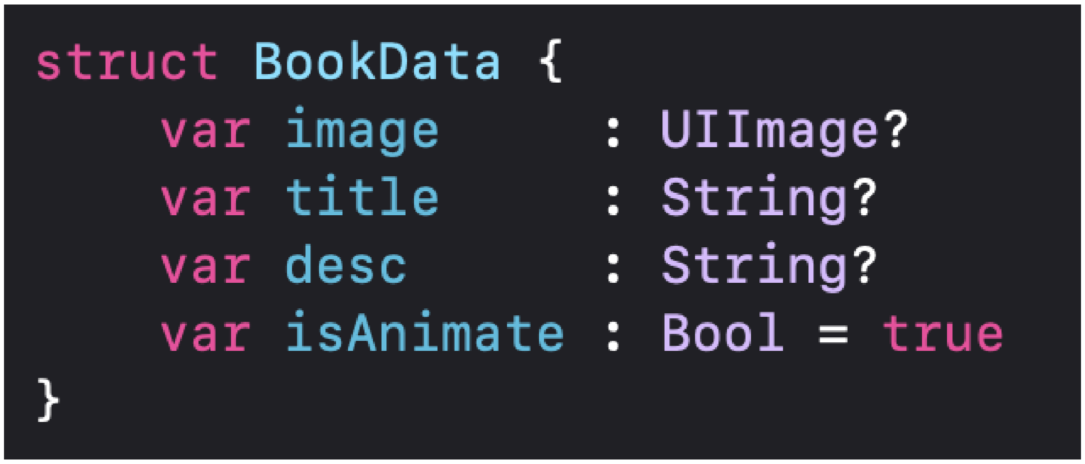


❖ Book → Self

a. **Step1:** In View create BookSB, design TopView that contains btnAdd & btnMoveCell. Below TopView contain collectionView that has image, labelTitle & labelDesc

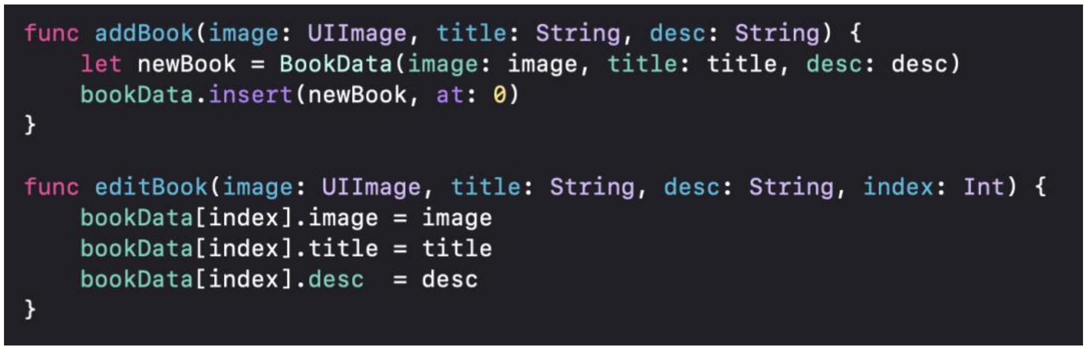


b. **Step2:** In Model create struct BookData that contains image, title, desc, isAnimate

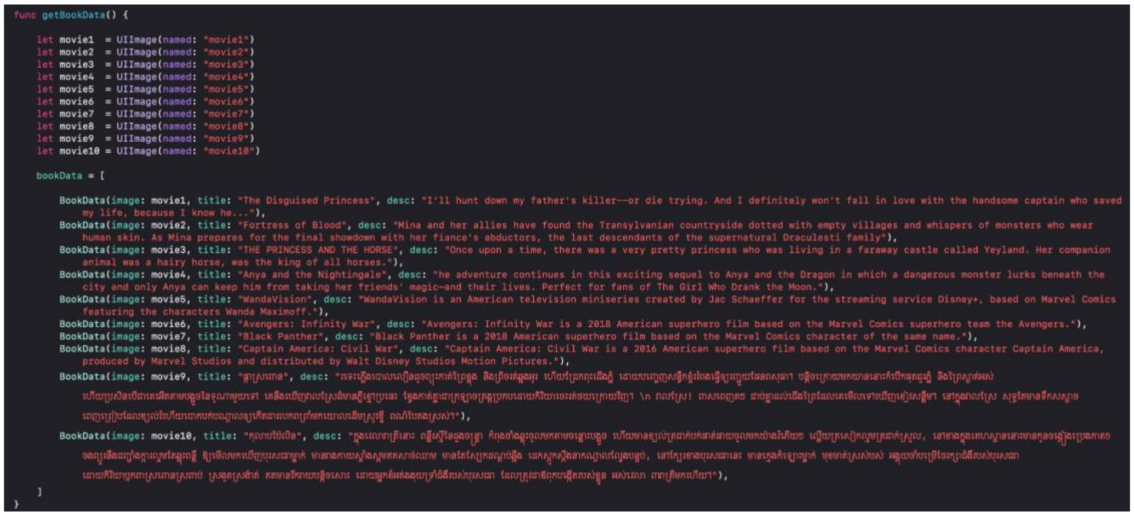


```swift
struct BookData {
    var image     : UIImage?
    var title     : String?
    var desc      : String?
    var isAnimate : Bool = true
}
```

c. **Step3:** In ViewModel create class BookVM that contain func addBook, editBook & getBookData

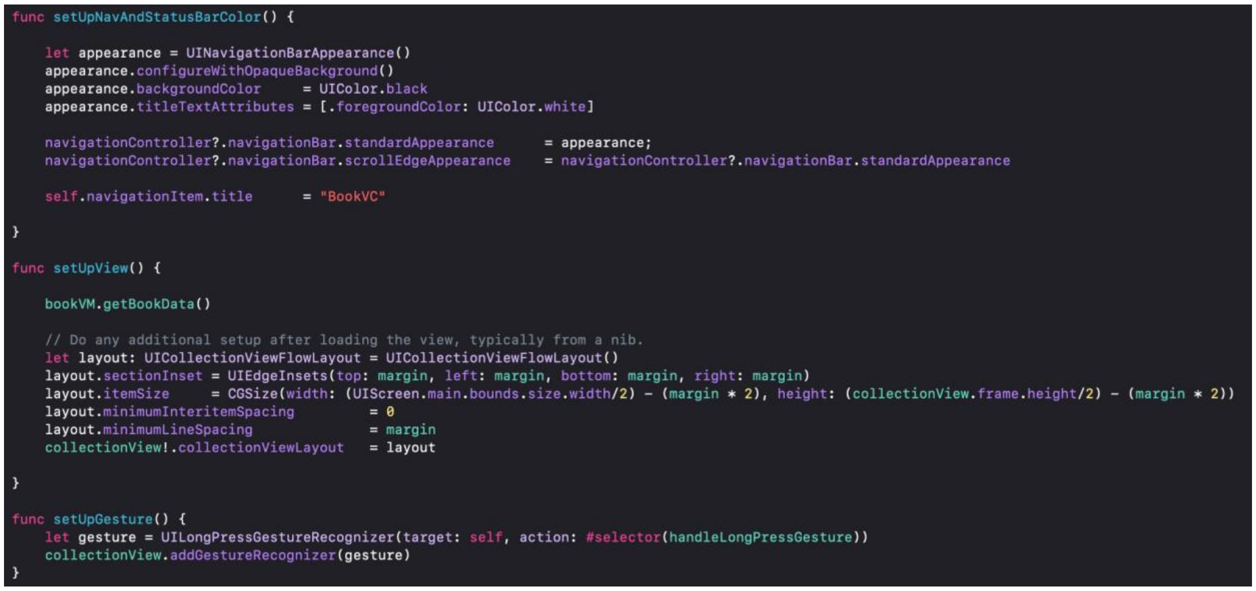


```swift
func addBook(image: UIImage, title: String, desc: String) {
    let newBook = BookData(image: image, title: title, desc: desc)
    bookData.insert(newBook, at: 0)
}

func editBook(image: UIImage, title: String, desc: String, index: Int) {
    bookData[index].image = image
    bookData[index].title = title
    bookData[index].desc  = desc
}
```

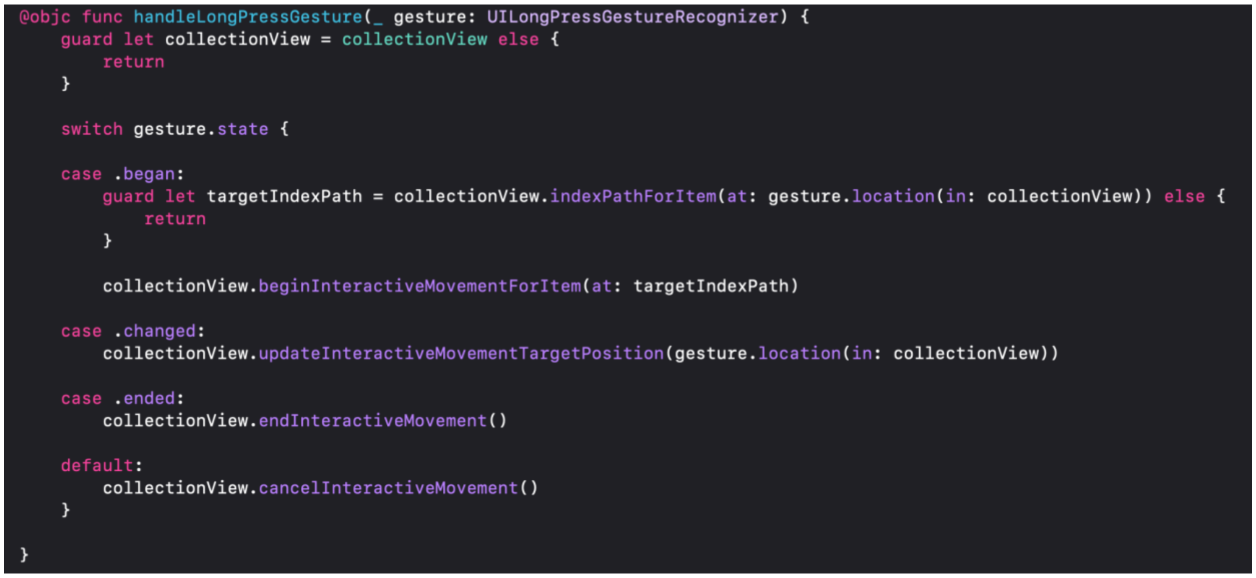


```swift
func getBookData() {

    let movie1  = UIImage(named: "movie1")
    let movie2  = UIImage(named: "movie2")
    let movie3  = UIImage(named: "movie3")
    let movie4  = UIImage(named: "movie4")
    let movie5  = UIImage(named: "movie5")
    let movie6  = UIImage(named: "movie6")
    let movie7  = UIImage(named: "movie7")
    let movie8  = UIImage(named: "movie8")
    let movie9  = UIImage(named: "movie9")
    let movie10 = UIImage(named: "movie10")

    bookData = [

        BookData(image: movie1, title: "The Disguised Princess", desc: "I'll hunt down my father's killer--or die trying. And I definitely won't fall in love with the handsome captain who saved
            my life, because I know he..."),
        BookData(image: movie2, title: "Fortress of Blood", desc: "Mina and her allies have found the Transylvanian countryside dotted with empty villages and whispers of monsters who wear
            human skin. As Mina prepares for the final showdown with her fiance's abductors, the last descendants of the supernatural Draculesti family"),
        BookData(image: movie3, title: "THE PRINCESS AND THE HORSE", desc: "Once upon a time, there was a very pretty princess who was living in a faraway castle called Yeyland. Her companion
            animal was a hairy horse, was the king of all horses."),
        BookData(image: movie4, title: "Anya and the Nightingale", desc: "he adventure continues in this exciting sequel to Anya and the Dragon in which a dangerous monster lurks beneath the
            city and only Anya can keep him from taking her friends' magic—and their lives. Perfect for fans of The Girl Who Drank the Moon."),
        BookData(image: movie5, title: "WandaVision", desc: "WandaVision is an American television miniseries created by Jac Schaeffer for the streaming service Disney+, based on Marvel Comics
            featuring the characters Wanda Maximoff."),
        BookData(image: movie6, title: "Avengers: Infinity War", desc: "Avengers: Infinity War is a 2018 American superhero film based on the Marvel Comics superhero team the Avengers."),
        BookData(image: movie7, title: "Black Panther", desc: "Black Panther is a 2018 American superhero film based on the Marvel Comics character of the same name."),
        BookData(image: movie8, title: "Captain America: Civil War", desc: "Captain America: Civil War is a 2016 American superhero film based on the Marvel Comics character Captain America,
            produced by Marvel Studios and distributed by Walt Disney Studios Motion Pictures."),
        BookData(image: movie9, title: "ផ្កាស្រពោន", desc: "[illegible]
            [illegible] \n [illegible]
            [illegible]"),

        BookData(image: movie10, title: "កុលាបប៉ៃលិន", desc: "[illegible]
            [illegible]
            [illegible]"),
    ]
}
```

d. **Step4:** In View create BookVC, on viewDidLoad call func setUpNavAndStatusBarColor, setUpView, setUpGesture

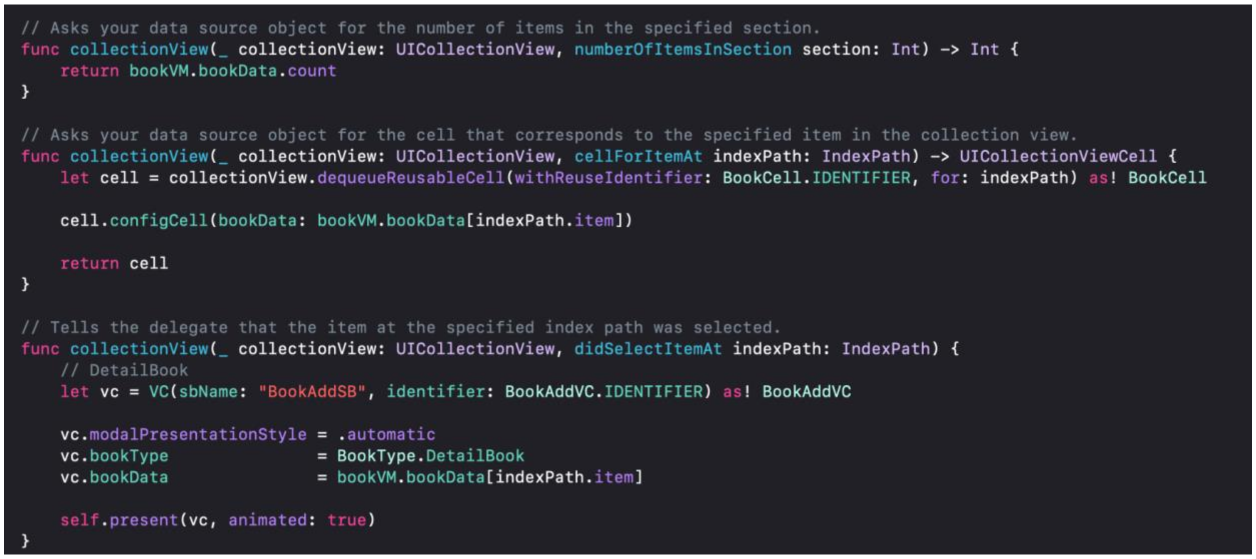


```swift
func setUpNavAndStatusBarColor() {

    let appearance = UINavigationBarAppearance()
    appearance.configureWithOpaqueBackground()
    appearance.backgroundColor     = UIColor.black
    appearance.titleTextAttributes = [.foregroundColor: UIColor.white]

    navigationController?.navigationBar.standardAppearance      = appearance;
    navigationController?.navigationBar.scrollEdgeAppearance    = navigationController?.navigationBar.standardAppearance

    self.navigationItem.title      = "BookVC"

}

func setUpView() {

    bookVM.getBookData()

    // Do any additional setup after loading the view, typically from a nib.
    let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
    layout.sectionInset = UIEdgeInsets(top: margin, left: margin, bottom: margin, right: margin)
    layout.itemSize     = CGSize(width: (UIScreen.main.bounds.size.width/2) - (margin * 2), height: (collectionView.frame.height/2) - (margin * 2))
    layout.minimumInteritemSpacing         = 0
    layout.minimumLineSpacing              = margin
    collectionView!.collectionViewLayout   = layout

}

func setUpGesture() {
    let gesture = UILongPressGestureRecognizer(target: self, action: #selector(handleLongPressGesture))
    collectionView.addGestureRecognizer(gesture)
}
```

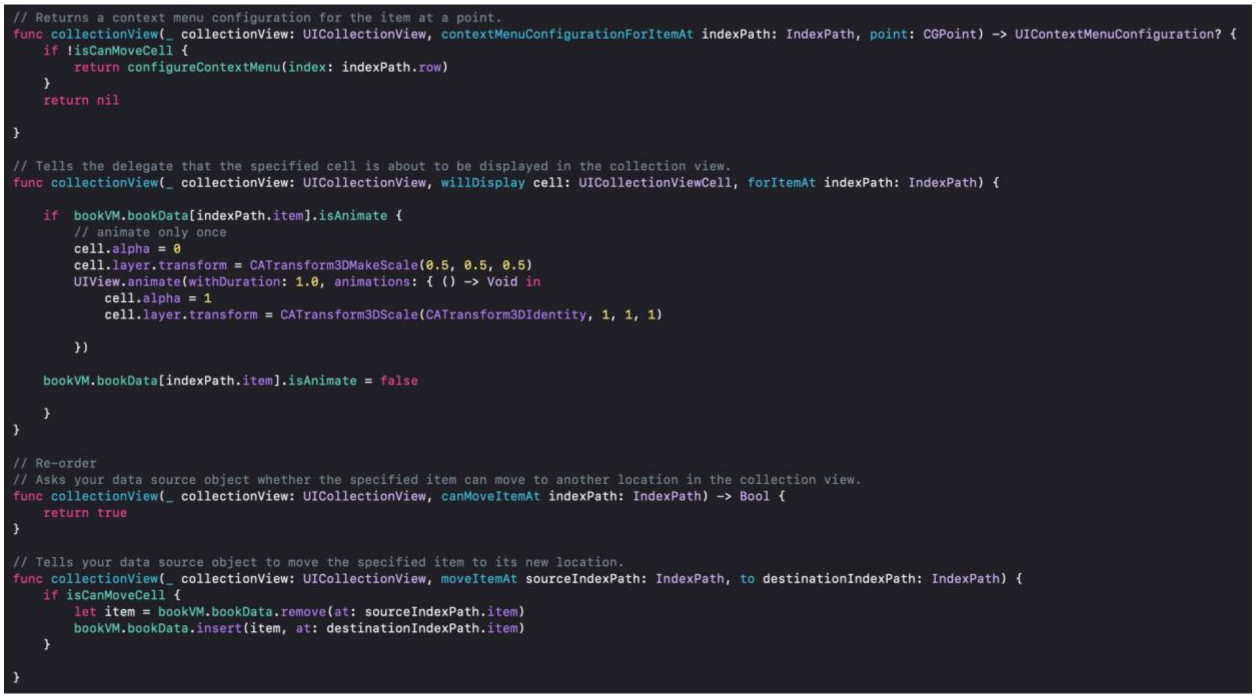


```swift
@objc func handleLongPressGesture(_ gesture: UILongPressGestureRecognizer) {
    guard let collectionView = collectionView else {
        return
    }

    switch gesture.state {

    case .began:
        guard let targetIndexPath = collectionView.indexPathForItem(at: gesture.location(in: collectionView)) else {
            return
        }

        collectionView.beginInteractiveMovementForItem(at: targetIndexPath)

    case .changed:
        collectionView.updateInteractiveMovementTargetPosition(gesture.location(in: collectionView))

    case .ended:
        collectionView.endInteractiveMovement()

    default:
        collectionView.cancelInteractiveMovement()
    }

}
```

e. **Step5:** extension BookVC: UICollectionViewDelegate, UICollectionViewDataSource
   - ➔ When didSelectItemAt it will go to BookAddVC

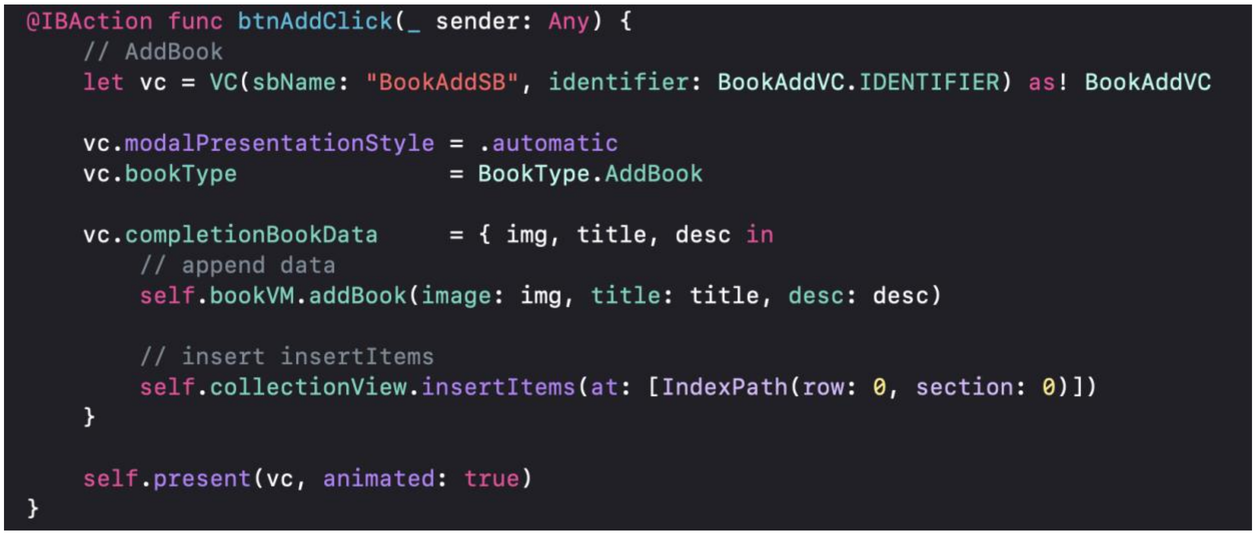


```swift
// Asks your data source object for the number of items in the specified section.
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    return bookVM.bookData.count
}

// Asks your data source object for the cell that corresponds to the specified item in the collection view.
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: BookCell.IDENTIFIER, for: indexPath) as! BookCell

    cell.configCell(bookData: bookVM.bookData[indexPath.item])

    return cell
}

// Tells the delegate that the item at the specified index path was selected.
func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
    // DetailBook
    let vc = VC(sbName: "BookAddSB", identifier: BookAddVC.IDENTIFIER) as! BookAddVC

    vc.modalPresentationStyle = .automatic
    vc.bookType               = BookType.DetailBook
    vc.bookData               = bookVM.bookData[indexPath.item]

    self.present(vc, animated: true)
}
```



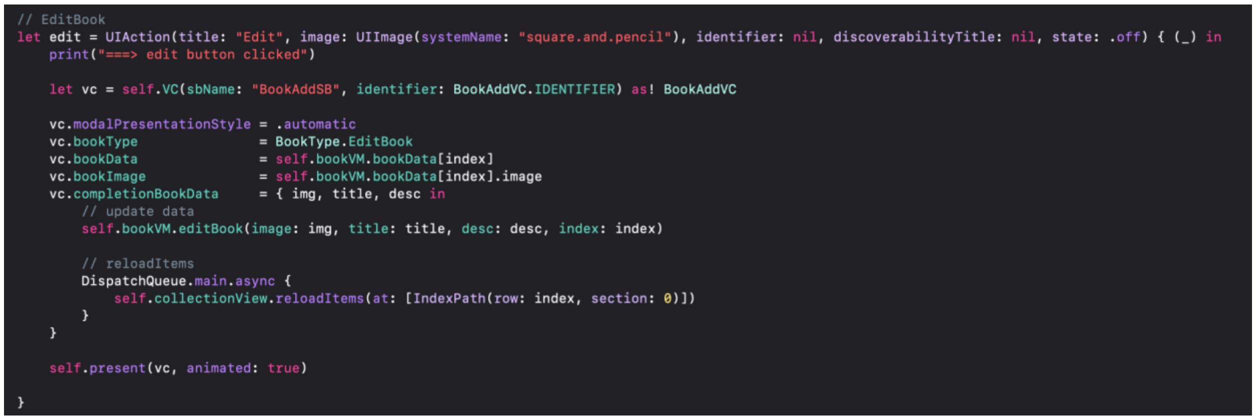
```swift
// Returns a context menu configuration for the item at a point.
func collectionView(_ collectionView: UICollectionView, contextMenuConfigurationForItemAt indexPath: IndexPath, point: CGPoint) -> UIContextMenuConfiguration? {
    if !isCanMoveCell {
        return configureContextMenu(index: indexPath.row)
    }
    return nil

}

// Tells the delegate that the specified cell is about to be displayed in the collection view.
func collectionView(_ collectionView: UICollectionView, willDisplay cell: UICollectionViewCell, forItemAt indexPath: IndexPath) {

    if  bookVM.bookData[indexPath.item].isAnimate {
        // animate only once
        cell.alpha = 0
        cell.layer.transform = CATransform3DMakeScale(0.5, 0.5, 0.5)
        UIView.animate(withDuration: 1.0, animations: { () -> Void in
            cell.alpha = 1
            cell.layer.transform = CATransform3DScale(CATransform3DIdentity, 1, 1, 1)

        })

    bookVM.bookData[indexPath.item].isAnimate = false

    }
}

// Re-order
// Asks your data source object whether the specified item can move to another location in the collection view.
func collectionView(_ collectionView: UICollectionView, canMoveItemAt indexPath: IndexPath) -> Bool {
    return true
}

// Tells your data source object to move the specified item to its new location.
func collectionView(_ collectionView: UICollectionView, moveItemAt sourceIndexPath: IndexPath, to destinationIndexPath: IndexPath) {
    if isCanMoveCell {
        let item = bookVM.bookData.remove(at: sourceIndexPath.item)
        bookVM.bookData.insert(item, at: destinationIndexPath.item)
    }

}
```

f. **Step6:** On btnAddClick, it will go to BookAddVC, when BookAddVC dismiss we get completionBookData to addBook in VM & then we insertItems



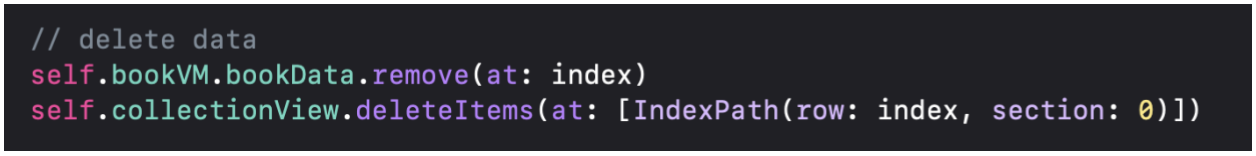
```swift
@IBAction func btnAddClick(_ sender: Any) {
    // AddBook
    let vc = VC(sbName: "BookAddSB", identifier: BookAddVC.IDENTIFIER) as! BookAddVC

    vc.modalPresentationStyle = .automatic
    vc.bookType               = BookType.AddBook

    vc.completionBookData     = { img, title, desc in
        // append data
        self.bookVM.addBook(image: img, title: title, desc: desc)

        // insert insertItems
        self.collectionView.insertItems(at: [IndexPath(row: 0, section: 0)])
    }

    self.present(vc, animated: true)
}
```

g. **Step7:** On editAction, it will go to BookAddVC, when BookAddVC dismiss we get completionBookData to editBook in VM & then we reloadItems



```swift
// EditBook
let edit = UIAction(title: "Edit", image: UIImage(systemName: "square.and.pencil"), identifier: nil, discoverabilityTitle: nil, state: .off) { (_) in
    print("===> edit button clicked")

    let vc = self.VC(sbName: "BookAddSB", identifier: BookAddVC.IDENTIFIER) as! BookAddVC

    vc.modalPresentationStyle = .automatic
    vc.bookType               = BookType.EditBook
    vc.bookData               = self.bookVM.bookData[index]
    vc.bookImage              = self.bookVM.bookData[index].image
    vc.completionBookData     = { img, title, desc in
        // update data
        self.bookVM.editBook(image: img, title: title, desc: desc, index: index)

        // reloadItems
        DispatchQueue.main.async {
            self.collectionView.reloadItems(at: [IndexPath(row: index, section: 0)])
        }
    }

    self.present(vc, animated: true)

}
```

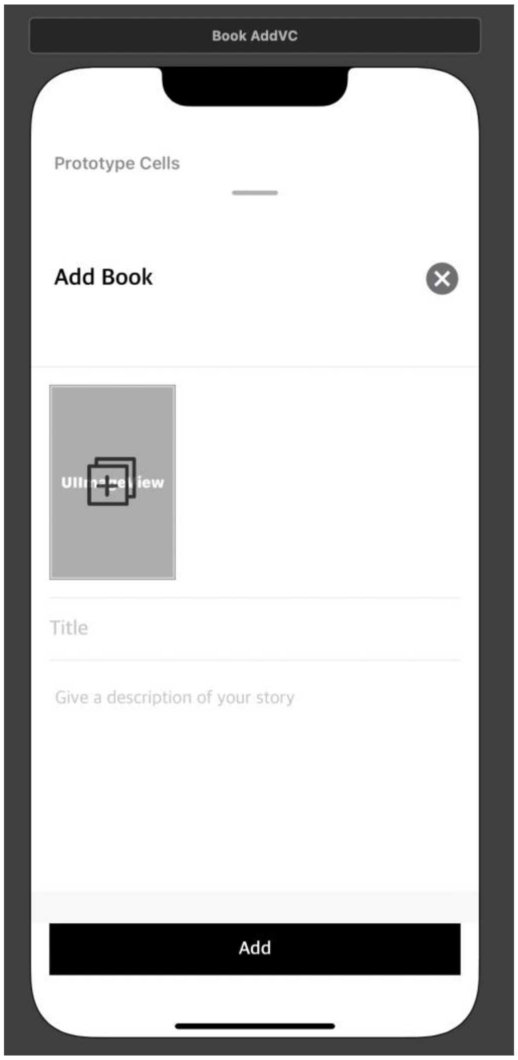
h. **Step8:** when click Yes we remove data in VM at specific index & then we deleteItems



```swift
// delete data
self.bookVM.bookData.remove(at: index)
self.collectionView.deleteItems(at: [IndexPath(row: index, section: 0)])
```

❖ Book → BookAdd

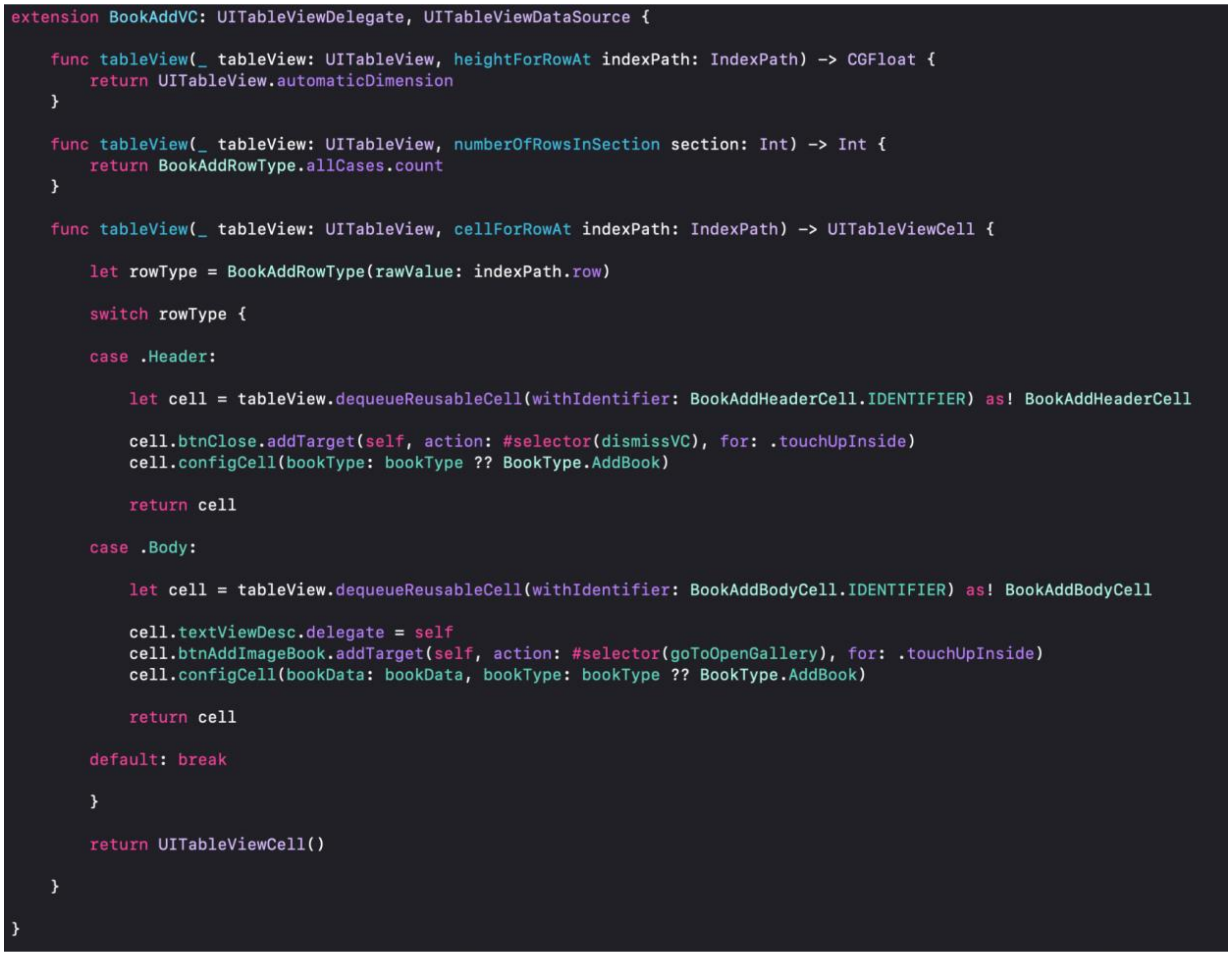
i. **Step9:** In View create BookAddSB, add TableView that contains BookAddHeaderCell & BookAddBodyCell, below TableView contain brnAdd



j. **Step10:** In BookAddVC on viewDidLoad call setUpView to checkBookType for hide show btnAdd & setTitle

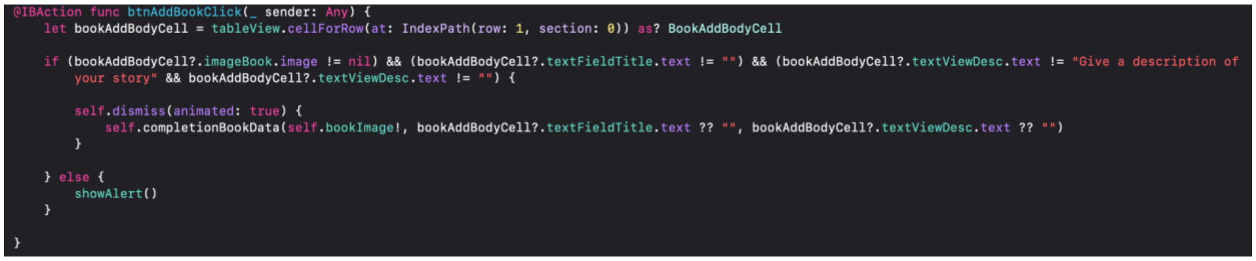
```
func checkBookType() {
    if bookType == BookType.DetailBook {
        btnAdd.isHidden                  = true
        constraintBtnAddHeight.constant = 0
    } else {
        btnAdd.isHidden                  = false
        constraintBtnAddHeight.constant = 48
        btnAdd.setTitle(bookType == BookType.AddBook ? "Add Book" : "Update Book", for: .normal)
    }
}
```

k. **Step11:** extension BookAddVC: UITableViewDelegate, UITableViewDataSource, we get data from BookVC to show & check conditions depending on the type of book



```swift
extension BookAddVC: UITableViewDelegate, UITableViewDataSource {

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return UITableView.automaticDimension
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return BookAddRowType.allCases.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

        let rowType = BookAddRowType(rawValue: indexPath.row)

        switch rowType {

        case .Header:

            let cell = tableView.dequeueReusableCell(withIdentifier: BookAddHeaderCell.IDENTIFIER) as! BookAddHeaderCell

            cell.btnClose.addTarget(self, action: #selector(dismissVC), for: .touchUpInside)
            cell.configCell(bookType: bookType ?? BookType.AddBook)

            return cell

        case .Body:

            let cell = tableView.dequeueReusableCell(withIdentifier: BookAddBodyCell.IDENTIFIER) as! BookAddBodyCell

            cell.textViewDesc.delegate = self
            cell.btnAddImageBook.addTarget(self, action: #selector(goToOpenGallery), for: .touchUpInside)
            cell.configCell(bookData: bookData, bookType: bookType ?? BookType.AddBook)

            return cell

        default: break

        }

        return UITableViewCell()

    }

}
```

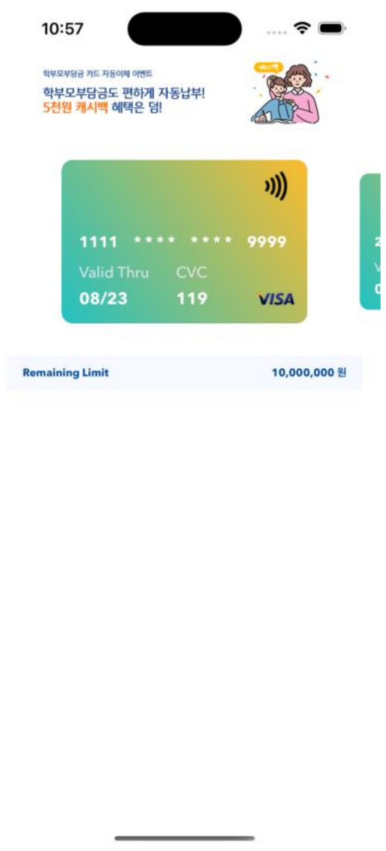
l. **Step12:** when clicking btnAddBookClick, we check some conditions if true we dismiss & completionBookData else showAlert



```swift
@IBAction func btnAddBookClick(_ sender: Any) {
    let bookAddBodyCell = tableView.cellForRow(at: IndexPath(row: 1, section: 0)) as? BookAddBodyCell

    if (bookAddBodyCell?.imageBook.image != nil) && (bookAddBodyCell?.textFieldTitle.text != "") && (bookAddBodyCell?.textViewDesc.text != "Give a description of
        your story" && bookAddBodyCell?.textViewDesc.text != "") {

        self.dismiss(animated: true) {
            self.completionBookData(self.bookImage!, bookAddBodyCell?.textFieldTitle.text ?? "", bookAddBodyCell?.textViewDesc.text ?? "")
        }

    } else {
        showAlert()
    }

}
```

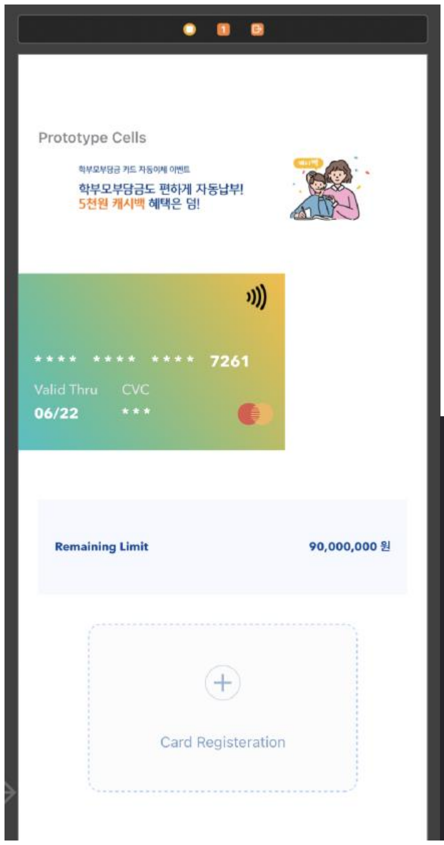
3. Sample project 2: show how to do swipe card step by step in collectionView [Card → CardListSB]

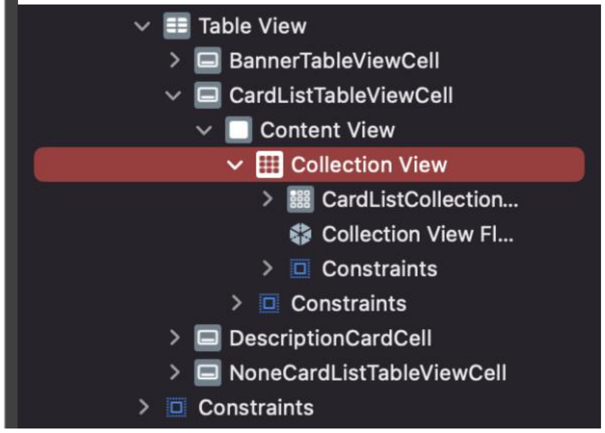
Reference:



❖ **Card**

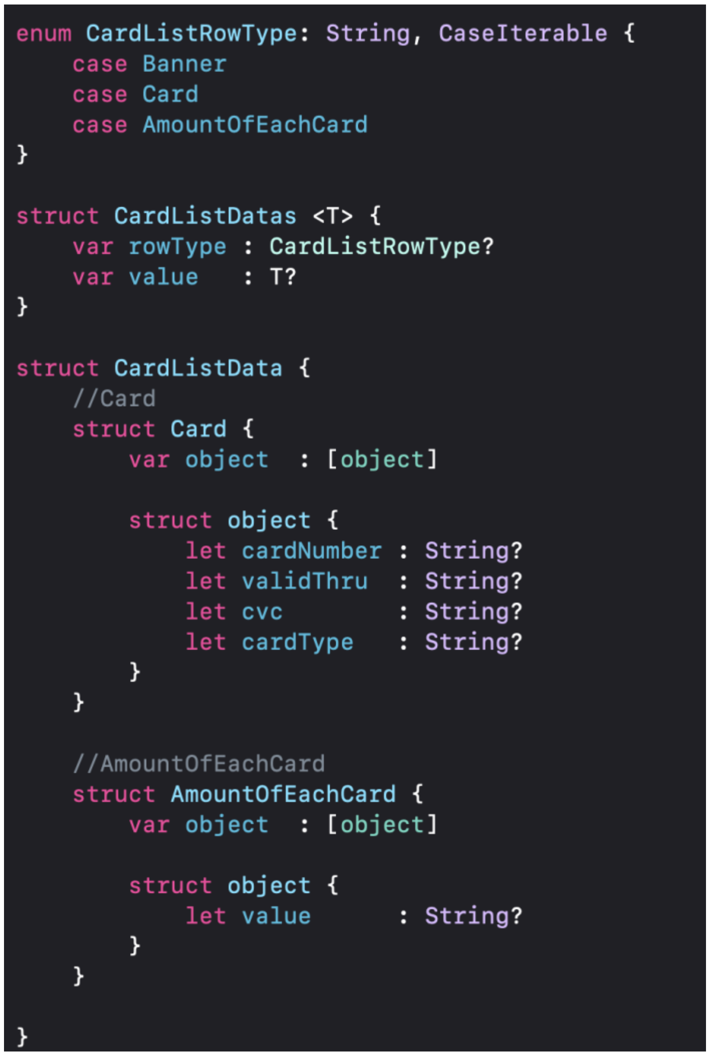
a. **Step1:** In View create CardListSB, add TableView that contains BannerTableViewCell, CardListTableViewCell contain collectionView, DescriptionCardCell, NoneCardListTableViewCell





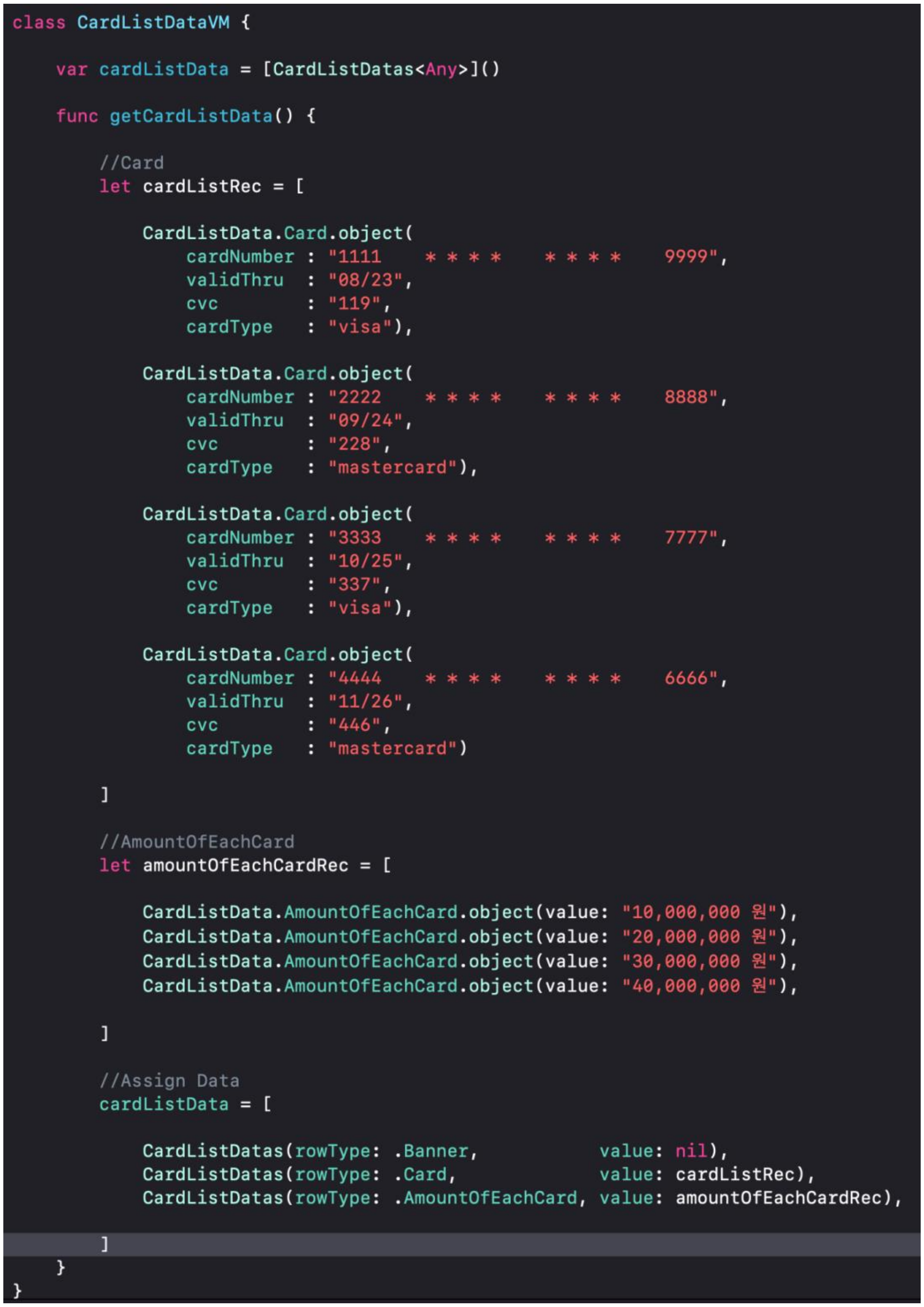
```
⌄ Table View
    › BannerTableViewCell
    ⌄ CardListTableViewCell
        ⌄ Content View
            ⌄ Collection View
                › CardListCollection...
                  Collection View Fl...
                › Constraints
            › Constraints
    › DescriptionCardCell
    › NoneCardListTableViewCell
› Constraints
```

b. **Step2:** In Model create struct CardListDatas <T> that contains rowType is CardListRowType & value is T type(get any value not define type)



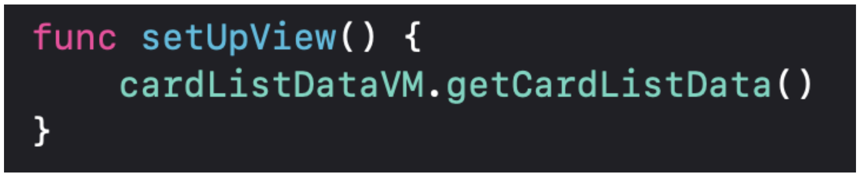
```swift
enum CardListRowType: String, CaseIterable {
    case Banner
    case Card
    case AmountOfEachCard
}

struct CardListDatas <T> {
    var rowType : CardListRowType?
    var value   : T?
}

struct CardListData {
    //Card
    struct Card {
        var object  : [object]

        struct object {
            let cardNumber : String?
            let validThru  : String?
            let cvc        : String?
            let cardType   : String?
        }
    }

    //AmountOfEachCard
    struct AmountOfEachCard {
        var object  : [object]

        struct object {
            let value      : String?
        }
    }

}
```

c. **Step3:** In ViewModel create class CardListDataVM that contains var cardListData & func getCardListData for init data to var cardListData



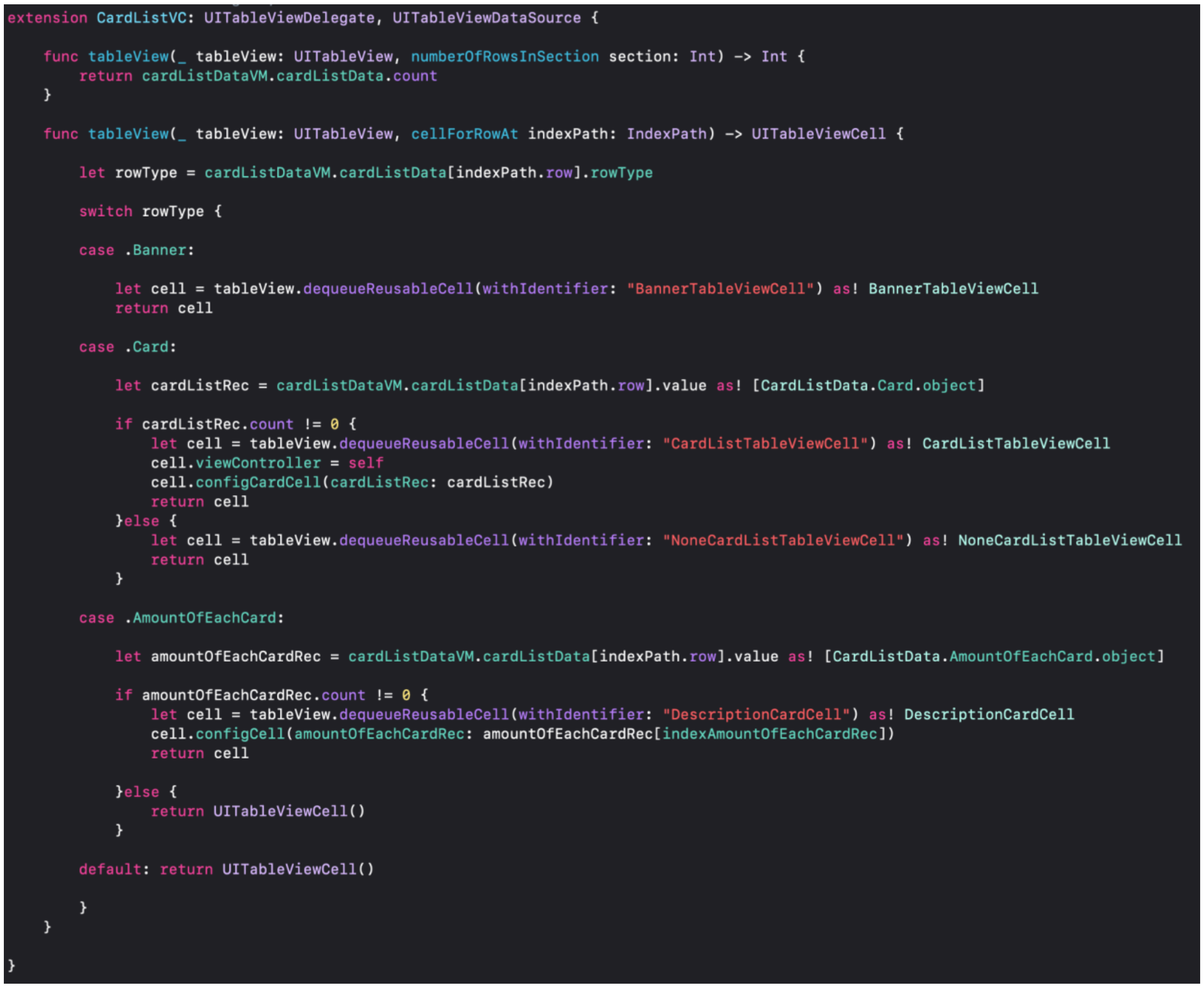
```swift
class CardListDataVM {

    var cardListData = [CardListDatas<Any>]()

    func getCardListData() {

        //Card
        let cardListRec = [

            CardListData.Card.object(
                cardNumber : "1111    * * * *    * * * *    9999",
                validThru  : "08/23",
                cvc        : "119",
                cardType   : "visa"),

            CardListData.Card.object(
                cardNumber : "2222    * * * *    * * * *    8888",
                validThru  : "09/24",
                cvc        : "228",
                cardType   : "mastercard"),

            CardListData.Card.object(
                cardNumber : "3333    * * * *    * * * *    7777",
                validThru  : "10/25",
                cvc        : "337",
                cardType   : "visa"),

            CardListData.Card.object(
                cardNumber : "4444    * * * *    * * * *    6666",
                validThru  : "11/26",
                cvc        : "446",
                cardType   : "mastercard")

        ]

        //AmountOfEachCard
        let amountOfEachCardRec = [

            CardListData.AmountOfEachCard.object(value: "10,000,000 원"),
            CardListData.AmountOfEachCard.object(value: "20,000,000 원"),
            CardListData.AmountOfEachCard.object(value: "30,000,000 원"),
            CardListData.AmountOfEachCard.object(value: "40,000,000 원"),

        ]

        //Assign Data
        cardListData = [

            CardListDatas(rowType: .Banner,           value: nil),
            CardListDatas(rowType: .Card,             value: cardListRec),
            CardListDatas(rowType: .AmountOfEachCard, value: amountOfEachCardRec),

        ]
    }
}
```

d. **Step4:** In View create CardListVC, on viewDidLoad call func setUpView to call getCardListData



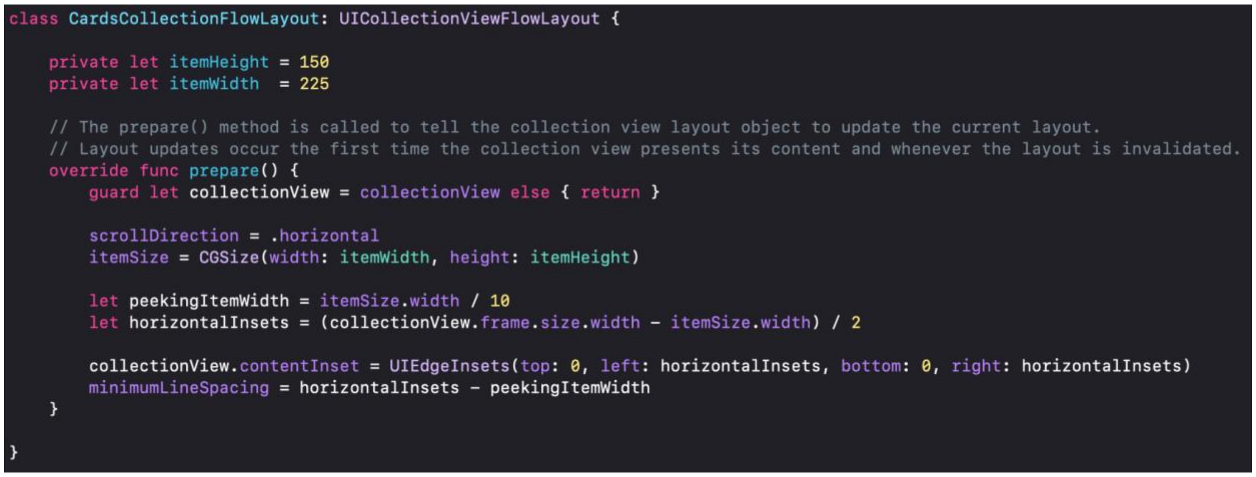
```swift
func setUpView() {
    cardListDataVM.getCardListData()
}
```

e. **Step5:** extension CardListVC: UITableViewDelegate, UITableViewDataSource, we get data from cardListDataVM.cardListData to show & check conditions depending on rowType



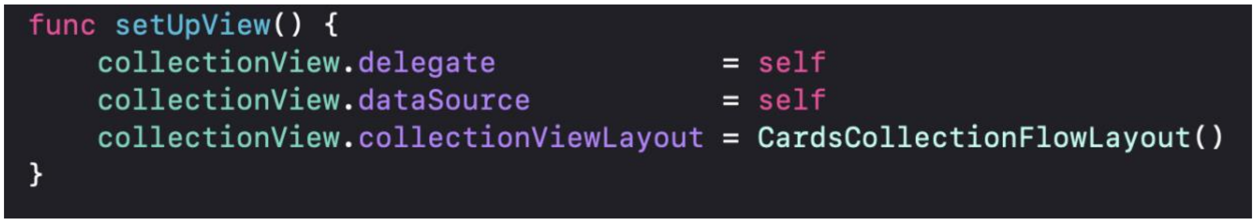
```swift
extension CardListVC: UITableViewDelegate, UITableViewDataSource {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return cardListDataVM.cardListData.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

        let rowType = cardListDataVM.cardListData[indexPath.row].rowType

        switch rowType {

        case .Banner:

            let cell = tableView.dequeueReusableCell(withIdentifier: "BannerTableViewCell") as! BannerTableViewCell
            return cell

        case .Card:

            let cardListRec = cardListDataVM.cardListData[indexPath.row].value as! [CardListData.Card.object]

            if cardListRec.count != 0 {
                let cell = tableView.dequeueReusableCell(withIdentifier: "CardListTableViewCell") as! CardListTableViewCell
                cell.viewController = self
                cell.configCardCell(cardListRec: cardListRec)
                return cell
            }else {
                let cell = tableView.dequeueReusableCell(withIdentifier: "NoneCardListTableViewCell") as! NoneCardListTableViewCell
                return cell
            }

        case .AmountOfEachCard:

            let amountOfEachCardRec = cardListDataVM.cardListData[indexPath.row].value as! [CardListData.AmountOfEachCard.object]

            if amountOfEachCardRec.count != 0 {
                let cell = tableView.dequeueReusableCell(withIdentifier: "DescriptionCardCell") as! DescriptionCardCell
                cell.configCell(amountOfEachCardRec: amountOfEachCardRec[indexAmountOfEachCardRec])
                return cell

            }else {
                return UITableViewCell()
            }

        default: return UITableViewCell()

        }
    }

}
```

f. **Step6: Creating Spacing Between Cards (Centered Pagination):**
A good way to control how the collection view cells are positioned and arranged on the screen is to subclass the UICollectionViewFlowLayout. We'll create a custom class that handles the centered pagination for us:



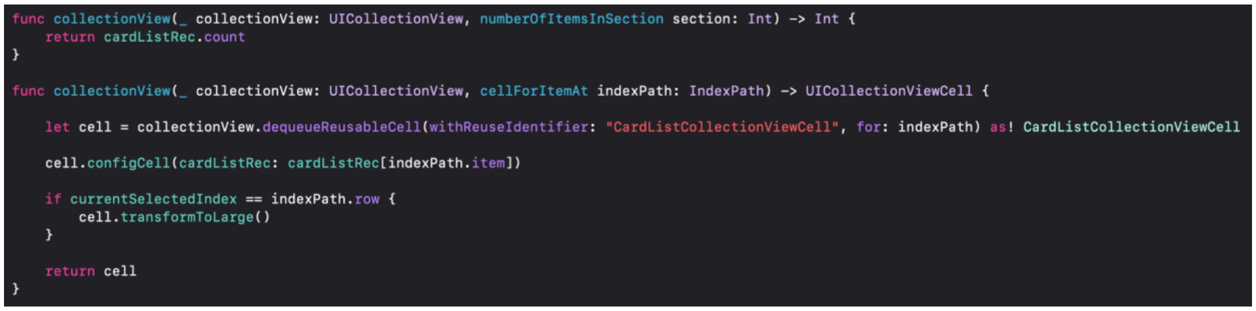
```swift
class CardsCollectionFlowLayout: UICollectionViewFlowLayout {

    private let itemHeight = 150
    private let itemWidth  = 225

    // The prepare() method is called to tell the collection view layout object to update the current layout.
    // Layout updates occur the first time the collection view presents its content and whenever the layout is invalidated.
    override func prepare() {
        guard let collectionView = collectionView else { return }

        scrollDirection = .horizontal
        itemSize = CGSize(width: itemWidth, height: itemHeight)

        let peekingItemWidth = itemSize.width / 10
        let horizontalInsets = (collectionView.frame.size.width - itemSize.width) / 2

        collectionView.contentInset = UIEdgeInsets(top: 0, left: horizontalInsets, bottom: 0, right: horizontalInsets)
        minimumLineSpacing = horizontalInsets - peekingItemWidth
    }

}
```

g. **Step7:** In CardListTableViewCell, on awakeFromNib call func setUpView & then set it to the collection view, preferably in the ViewController class like this:



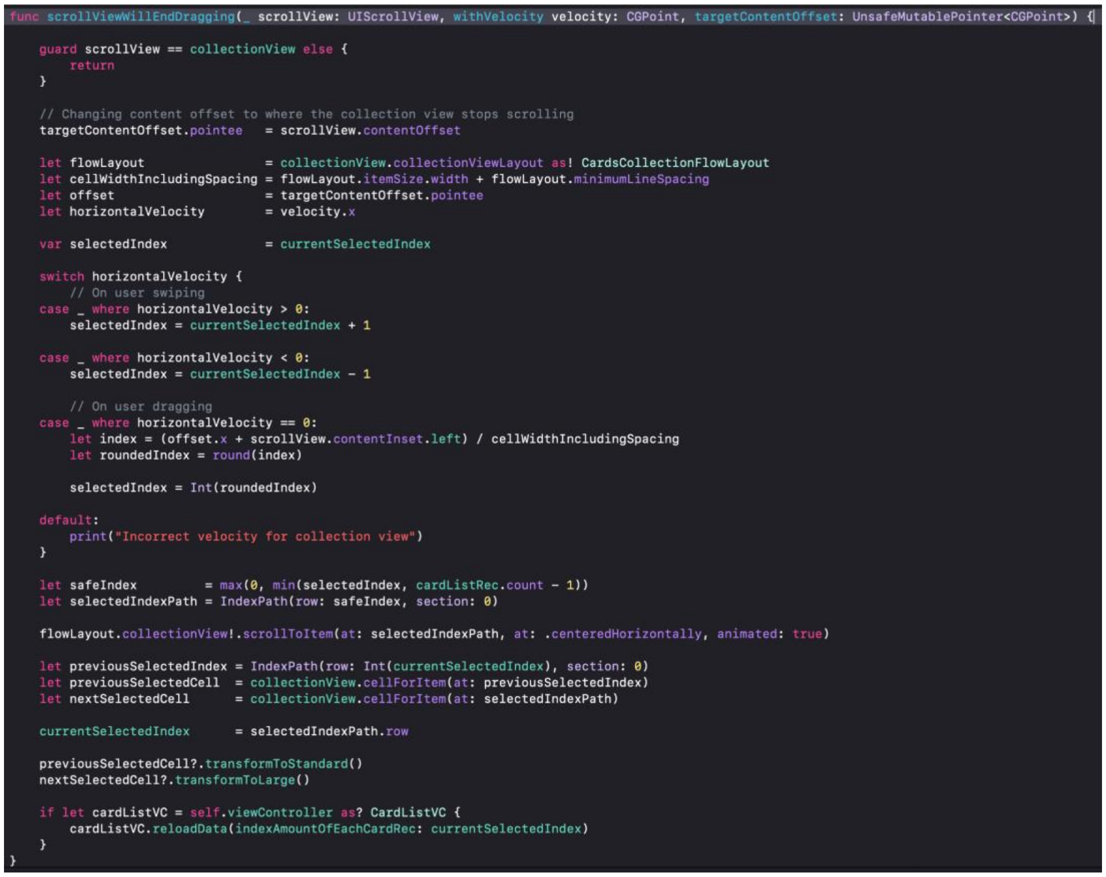
```swift
func setUpView() {
    collectionView.delegate             = self
    collectionView.dataSource           = self
    collectionView.collectionViewLayout = CardsCollectionFlowLayout()
}
```

h. **Step8:** eextension CardListTableViewCell: UICollectionViewDelegate, UICollectionViewDataSource, we get data from CardListVC to show



```swift
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    return cardListRec.count
}

func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "CardListCollectionViewCell", for: indexPath) as! CardListCollectionViewCell

    cell.configCell(cardListRec: cardListRec[indexPath.item])

    if currentSelectedIndex == indexPath.row {
        cell.transformToLarge()
    }

    return cell
}
```

i. **Step9: Adding a Snap-on-Scroll Behavior**
The scrollViewWillEndDragging(_:withVelocity:target:ContentOffset:) instance method can be used to determine when the user finishes scrolling and which cell the collection view needs to scroll to:

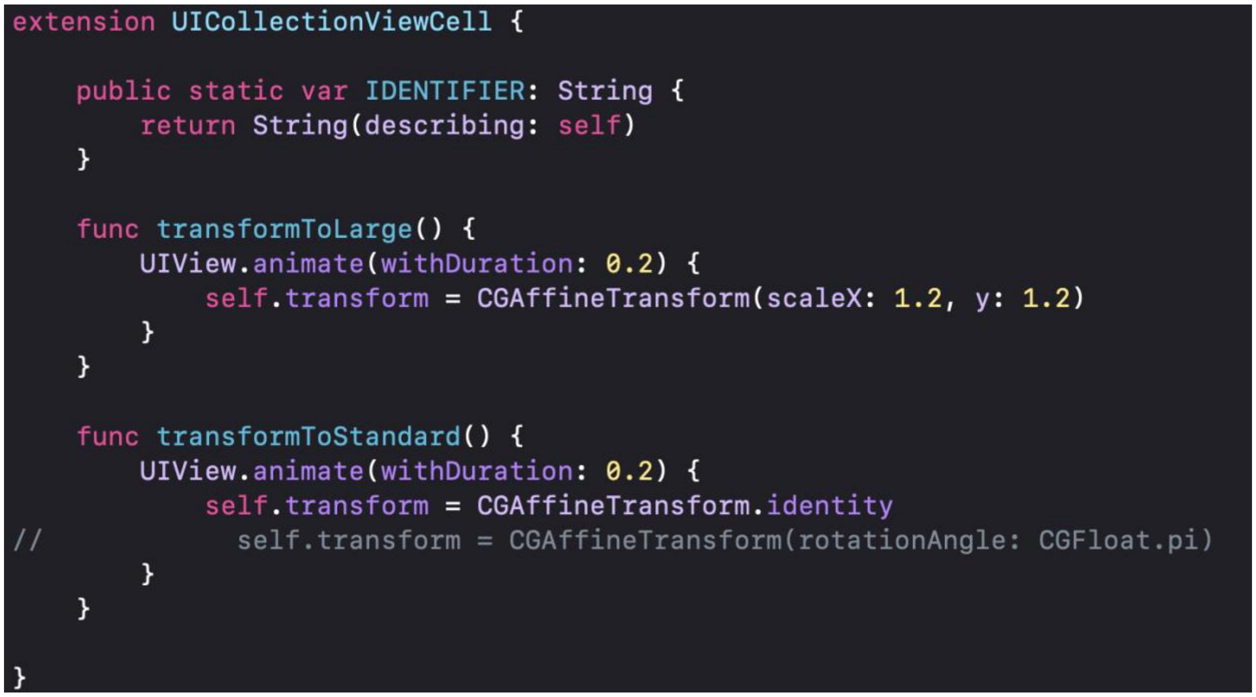


```swift
func scrollViewWillEndDragging(_ scrollView: UIScrollView, withVelocity velocity: CGPoint, targetContentOffset: UnsafeMutablePointer<CGPoint>) {

    guard scrollView == collectionView else {
        return
    }

    // Changing content offset to where the collection view stops scrolling
    targetContentOffset.pointee   = scrollView.contentOffset

    let flowLayout                = collectionView.collectionViewLayout as! CardsCollectionFlowLayout
    let cellWidthIncludingSpacing = flowLayout.itemSize.width + flowLayout.minimumLineSpacing
    let offset                    = targetContentOffset.pointee
    let horizontalVelocity        = velocity.x

    var selectedIndex             = currentSelectedIndex

    switch horizontalVelocity {
        // On user swiping
    case _ where horizontalVelocity > 0:
        selectedIndex = currentSelectedIndex + 1

    case _ where horizontalVelocity < 0:
        selectedIndex = currentSelectedIndex - 1

        // On user dragging
    case _ where horizontalVelocity == 0:
        let index = (offset.x + scrollView.contentInset.left) / cellWidthIncludingSpacing
        let roundedIndex = round(index)

        selectedIndex = Int(roundedIndex)

    default:
        print("Incorrect velocity for collection view")
    }

    let safeIndex         = max(0, min(selectedIndex, cardListRec.count - 1))
    let selectedIndexPath = IndexPath(row: safeIndex, section: 0)

    flowLayout.collectionView!.scrollToItem(at: selectedIndexPath, at: .centeredHorizontally, animated: true)

    let previousSelectedIndex = IndexPath(row: Int(currentSelectedIndex), section: 0)
    let previousSelectedCell  = collectionView.cellForItem(at: previousSelectedIndex)
    let nextSelectedCell      = collectionView.cellForItem(at: selectedIndexPath)

    currentSelectedIndex      = selectedIndexPath.row

    previousSelectedCell?.transformToStandard()
    nextSelectedCell?.transformToLarge()

    if let cardListVC = self.viewController as? CardListVC {
        cardListVC.reloadData(indexAmountOfEachCardRec: currentSelectedIndex)
    }
}
```

j. **Step10: Scaling Size of Cards on Scroll**

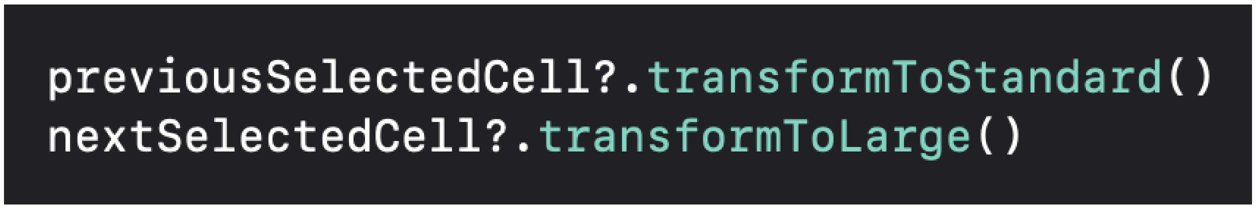
After the user has finished scrolling and the collection view has scrolled to the selected card, we can update the sizes of the previousSelectedCell and currentSelectedCell.

We'll create two extension methods that will handle this:



```swift
extension UICollectionViewCell {

    public static var IDENTIFIER: String {
        return String(describing: self)
    }

    func transformToLarge() {
        UIView.animate(withDuration: 0.2) {
            self.transform = CGAffineTransform(scaleX: 1.2, y: 1.2)
        }
    }

    func transformToStandard() {
        UIView.animate(withDuration: 0.2) {
            self.transform = CGAffineTransform.identity
//              self.transform = CGAffineTransform(rotationAngle: CGFloat.pi)
        }
    }

}
```

We can call these methods in the scrollViewWillEnd method after the collectionView has scrolled:



```swift
previousSelectedCell?.transformToStandard()
nextSelectedCell?.transformToLarge()
```

GitHub

- https://github.com/lymanny/UICollectionView-Sample-Project.git