NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

HOMEWORK #4

dedicated to Research Seminar
«Immersion in iOS Development»

MOSCOW 2024

Theme: Add a collection view triggers for wish-granting time.
Objective: Hone collection view skills, and learn to work with a calendar.

Task description

You will continue working on the WishMaker app. If you skipped this task ask the assistant to give you the best solution yet. You will add the third and last wish-granting feature: the power to schedule tasks to bring wishes closer to being granted! This requires you to use the collection view.

Task requirements

- You must delete Storyboard and use Swift + UIKit to complete this task.
- **You must use one of the following architectures: MVC, MVVM, or VIPER.**
- To use code from this tutorial or the internet you must understand it first.

Grading

Grade	Task
1	Add new button for scheduling wish-granting activity.
2	Buttons color changes depending on the background color.
3	Pressing the schedule button pushes WishCalendarViewController.
4	WishCalendarViewController has a collection view in it.
5	WishEventCells can be created with event title, description, start and end dates.
6	Events for cells can be created in a popup shown using present().
7	Event is created in the calendar.
8	Events are saved in app so that they stay after the app is closed.
9	A wish can be picked from saved wished list when scheduling an event.
10	The color picked on the WishMakerViewController is now passed to be background color for other screens.

Disclaimer

Overdue submissions are **punished** by a decrease in the grade equal to the number of days past the due date. This punishment is limited to **minus 5 points**. Please keep this in mind and plan your week accordingly.

Ask all the questions regarding this task or iOS development in general. Our goal is to enable you to become **the best iOS developer** possible. Teachers and assistants are here to **help**!

Submissions containing extraordinary solutions, exceptional code style, pattern usage, and entertaining features can lead to more points added. This is non-negotiable. Repeating or copying solutions that earned bonus points previously does not grant bonus points.

Starting this homework **your bad code** will **decrease your grade**. Remember to remove magic numbers and write clean code.

Tutorial

Continue working on your project from the previous homework. But when making commits write HW-5: in your commit messages. Try committing every point separately for bonus points and glorious success later in (real) life.

Point 1:

You've created enough buttons to do it yourself. know how to add a button. Let us do it properly together one more time for the last time. Here is a snippet to group our two buttons. Use it if you want.

```
private func configureActionStack() {
    actionStack.axis = .vertical
    view.addSubview(actionStack)
    actionStack.spacing = Constants.spacing

    for button in [addMoreWishesButton, scheduleWishesButton] {
        actionStack.addArrangedSubview(button)
    }

    configureAddMoreWishes()
    configureScheduleMissions()

    actionStack.pinBottom(to: view, Constants.stackBottom)
    actionStack.pinHorizontal(to: view, Constants.stackLeading)
}
```

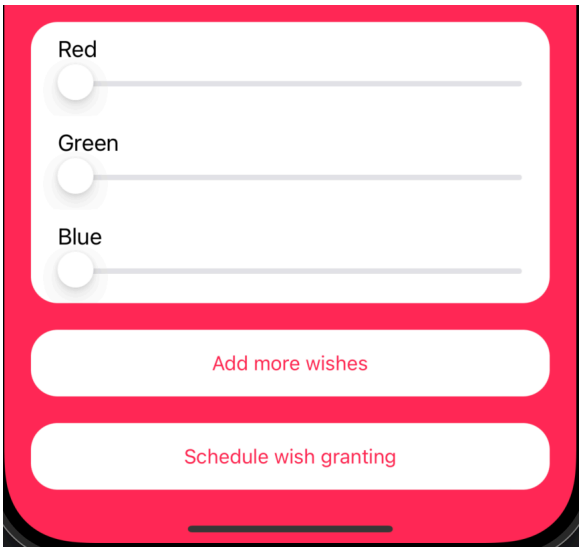

Image 12. Example for the buttons.

Point 2:

We already have the **colors** property with didSet that updates the background color. All you need to do is to change it in a way where it is also set as the title color for the buttons. If you read future points you might want to extract the current color into a property, so that it can be passed to other controllers.

Point 3:

Create a new **WishCalendarViewController**. The thing is, pushing the view controller is different from presenting it. Present allows us to show a simple popup screen with no depth. We want to utilize all routing technologies available to iOS app developers. So here we will have to change the AppDelegate. Right now it has **WishMakerViewController** as the rootViewController. We will need to create an instance of a UINavigationController.

```
let navC = UINavigationController()
```

This controller allows us to pun view controllers on top of each other in a stack. This small change allows us to create complex multiscreen applications. But we have to set a rootViewController for both the navigation controller and the app window.

```
let navC = UINavigationController(rootViewController: WishMakerViewController())
window?.rootViewController = navC
```

This allows us to have multiple screen layers. Routing between several screens allows users to jump to different parts of the application at any moment.

Now we need to push our new screen when the button is pressed. Insert the following code into **scheduleWishButtonPressed**:

```
let vc = WishCalendarViewController()
navigationController?.pushViewController(vc, animated: true)
```

Run your app and see what happens when you press the button. Confirm the new screen and move on. Check the presence of the UINavigationController below our screen. We will be able to push screens on top of it.

Point 4:

Previously you did work with UITableView, now it is time to work with UICollectionView. The first difference lies in its initialization:

```
private let collectionView: UICollectionView = UICollectionView(
    frame: .zero,
    collectionViewLayout: UICollectionViewFlowLayout()
)
```

If you try to initiate the collection without UICollectionViewFlowLayout it will throw an exception. Once you have a collection it is time to configure it:

```
private func configureCollection() {
    collectionView.delegate = self
    collectionView.dataSource = self
    collectionView.backgroundColor = .white
    collectionView.alwaysBounceVertical = true
    collectionView.showsVerticalScrollIndicator = false
    collectionView.contentInset = Constants.contentInset

    /* Temporary line */
    collectionView.register(UICollectionViewCell.self, forCellWithReuseIdentifier: "cell")

    view.addSubview(collectionView)

    collectionView.pinHorizontal(to: view)
    collectionView.pinBottom(to: view.safeAreaLayoutGuide.bottomAnchor)
    collectionView.pinTop(to: view.safeAreaLayoutGuide.topAnchor, Constants.collectionTop)
}
```

Pasting this code will cause build errors, due to 2 missing protocol conformations. Let us add them the proper way:

```
// MARK: - UICollectionViewDataSource
extension WishCalendarViewController: UICollectionViewDataSource {
    func collectionView(
        _ collectionView: UICollectionView,
        numberOfItemsInSection section: Int
    ) -> Int {
        return 10
    }

    func collectionView(
        _ collectionView: UICollectionView,
        cellForItemAt indexPath: IndexPath
    ) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "cell", for: indexPath)

        return cell
    }
}
```

```
// MARK: - UICollectionViewDelegateFlowLayout
extension WishCalendarViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(
        _ collectionView: UICollectionView,
        layout collectionViewLayout: UICollectionViewLayout,
        sizeForItemAt indexPath: IndexPath
    ) -> CGSize {
        // Adjust cell size as needed
        return CGSize(width: collectionView.bounds.width - 10, height: 100)
    }

    func collectionView(
        _ collectionView: UICollectionView,
        didSelectItemAt indexPath: IndexPath
    ) {
        print("Cell tapped at index \(indexPath.item)")
    }
}
```

Run the code and confirm you called the **configureCollection** method. You should see nothing unless you change **WishCalendarViewController backgroundColor**. In that case, confirm the UICollectionView using the debug view hierarchy.

Point 5:

Create a new file in your architecture view folder. Call it WishEventCell, and inherit it from UICollectionViewCell. We can start by adding the required views and identification property for registration:

```
static let reuseIdentifier: String = "WishEventCell"

private let wrapView: UIView = UIView()
private let titleLabel: UILabel = UILabel()
private let descriptionLabel: UILabel = UILabel()
private let startDateLabel: UILabel = UILabel()
private let endDateLabel: UILabel = UILabel()
```

Next, we can override the initializer and add UI configuration:

```
// MARK: - Lifecycle
override init(frame: CGRect) {
    super.init(frame: frame)

    configureWrap()
    configureTitleLabel()
    configureDescriptionLabel()
    configureStartDateLabel()
    configureEndDateLabel()
}

@available(*, unavailable)
required init?(coder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

// MARK: - Cell Configuration
func configure(with event: WishEventModel) {
    titleLabel.text = event.title
    descriptionLabel.text = event.description
    startDateLabel.text = "Start Date: \(event.startDate)"
    endDateLabel.text = "End Date: \(event.endDate)"
}
```

Next, you have to implement configuration methods. Be creative! Think about how you would want the UI to look. Take inspiration from Apple, Notion, Google, etc. Here is an example of how to start. It also shows good decomposition and marking practices.

```
// MARK: - UI Configuration
private func configureWrap() {
    addSubview(wrapView)

    wrapView.pin(to: self, Constants.offset)
    wrapView.layer.cornerRadius = Constants.cornerRadius
    wrapView.backgroundColor = Constants.backgroundColor
}

private func configureTitleLabel() {
    addSubview(titleLabel)
    titleLabel.textColor = .black
    titleLabel.pinTop(to: wrapView, Constants.titleTop)
    titleLabel.font = Constants.titleFont
    titleLabel.pinLeft(to: wrapView, Constants.titleLeading)
}
```

Once you are finished you can register the cell as follows:

```
if let layout = collectionView.collectionViewLayout as?
UICollectionViewFlowLayout {
    layout.minimumInteritemSpacing = 0
    layout.minimumLineSpacing = 0

    layout.invalidateLayout()
}
/* Temporary line */
collectionView.register(
    WishEventCell.self,
    forCellWithReuseIdentifier: WishEventCell.reuseIdentifier
)
```

Configure the cells before showing them on the screen in **UICollectionViewDataSource**. It can be done like this:

```
let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
WishEventCell.reuseIdentifier, for: indexPath)

guard let wishEventCell = cell as? WishEventCell else {
    return cell
}

wishEventCell.configure(
    with: WishEventModel(
        title: "Test",
        description: "Test description",
        startDate: "Start date",
        endDate: "End date"
    )
)

return cell
```

Point 6:

You have done something similar in previous homework, so I trust you to do the popup yourself. Name it **WishEventCreationView** and inherit it from UIViewController. Present it by pressing a plus button. What plus button? The one you will add of course. It should add a new event somewhere in your app, preferably in CoreData to achieve 8 points.

Point 7:

Now let us learn how to create calendar events in our iOS app. First, we need to press cmd + 1, select our app at the top of the file hierarchy, go to info.plist, and add the Privacy - calendars usage description. Provide the teaching assistant with a calming message on why you need access to the calendar:

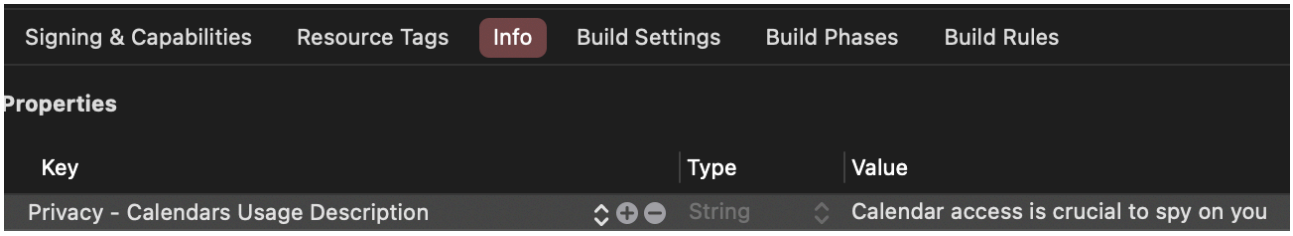

Image 13. Calendar access example.

Now that this is done we can set up a CalendarEventManager or CalendarService or *insert better naming here*. Here is the protocol for it and a simple model you can use:

```
import EventKit

protocol CalendarManaging {
    func create(eventModel: CalendarEventModel) -> Bool
}

struct CalendarEventModel {
    var title: String
    var startDate: Date
    var endDate: Date
    var note: String?
}
```

The implementation is on the next page. This point is the last one for this homework that has instructions. Points 8-10 need to be done by you. But the instructions don't end there. You spent significant time working on this project, it is a pretty good first iOS project you can add to your GitHub. But it'll need a little more work. First, confirm that the architecture you are using is implemented correctly. Then you can add the date sections to better separate events. Editing events is also a good addition. The inspiration for this homework is an amazing app called Just Button. It can give you more insight on how to do a good pet project for your portfolio. Next time you will be doing a news app, so prepare for a new project.

```swift
final class CalendarManager: CalendarManaging {
    private let eventStore : EKEventStore = EKEventStore()

    func create(eventModel: CalendarEventModel) -> Bool {
        var result: Bool = false
        let group = DispatchGroup()
        group.enter()

        create(eventModel: eventModel) { isCreated in
            result = isCreated
            group.leave()
        }

        group.wait()

        return result
    }

    func create(eventModel: CalendarEventModel, completion: ((Bool) -> Void)?) {
        let createEvent: EKEventStoreRequestAccessCompletionHandler = { [weak self] (granted,
error) in
            guard granted, error == nil, let self else {
                completion?(false)
                return
            }

            let event: EKEvent = EKEvent(eventStore: self.eventStore)

            event.title = eventModel.title
            event.startDate = eventModel.startDate
            event.endDate = eventModel.endDate
            event.notes = eventModel.note
            event.calendar = self.eventStore.defaultCalendarForNewEvents

            do {
                try self.eventStore.save(event, span: .thisEvent)
            } catch let error as NSError {
                print("failed to save event with error : \(error)")
                completion?(false)
            }

            completion?(true)
        }

        if #available(iOS 17.0, *) {
            eventStore.requestFullAccessToEvents(completion: createEvent)
        } else {
            eventStore.requestAccess(to: .event, completion: createEvent)
        }
    }
}
```

Conclusion:

Great job completing the WishMaker app! You've delved into iOS development, focusing on stacks, table view, collection view, and calendar integration.

Remember to uphold coding best practices, maintain clean code with no magic numbers, and consider this project as a valuable addition to your GitHub portfolio. Keep refining your skills by communicating with teaching staff and assistants.

If you follow provided instructions you can get up to 7 points. If you struggle doing points we entrusted you to complete based on previous assignments feel free to revisit homeworks 1-3 to regain the skills learned then.