HIGHER SCHOOL OF ECONOMICS UNIVERSITY

HOMEWORK #4

dedicated to Research Seminar
«Development of applications for the Apple iOS platform»

MOSCOW 2022

Theme and Objective

Theme	Working with the UITablwView
Objective	Mastering UITabView and UITableViewCell setup

Task description

You should create a second NoteViewController, which appears modally when note button(see previous homework) is tapped. The controller must display 2 section with cells. First section: One cell with UITextView for quick note input, UIButton for saving note into array. Second section: Contains all the cells created from array of notes or contains none (if the array is empty).

Task requirements

During this task:

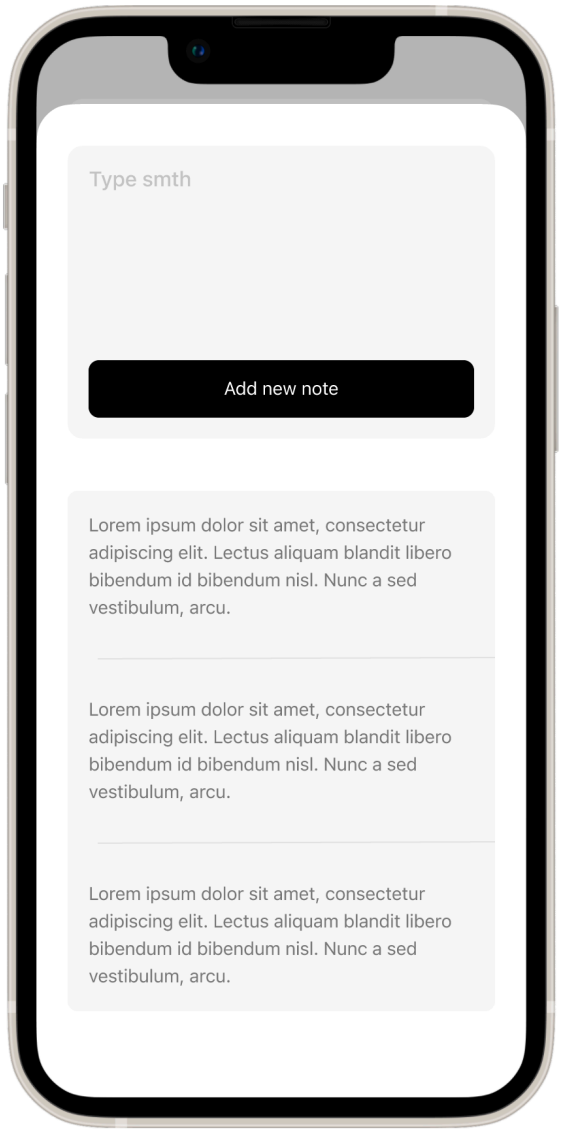
- You are not allowed to use React, SwiftUI, Objc-C or anything except vanilla Swift.
- You cannot use StoryBoard to layout the app.
- You are allowed to use any architecture for this app.
- You can use UIView extension provided to you before.

Grading

Grade	Task
0	The task was not submitted or it contains nothing
1	The task has nothing to do with the assignment.
2	NotesViewController is created
3	NotesViewController appears when you press the corresponding button
4	NotesViewController displays cells
5	Cells contain UITextView (of the note) and represents the array of notes
6	There are to sections with cells: one for input note’s text (1 cell: UITextView + Button to save), other for representing saved notes
7	Cells can be added
8	Cells can be deleted

9	Input cell contains placeholder (note cannot be saved if we see the placeholder or input nothing)
10	Notes are not deleted when the app is closed
Bonus	Some functionality of IOS15 is used

Visualisation



Tutorial

Point 2

The time has come for the new controller creation. Press CMND + N and name new file NotesViewController.

```
import UIKit

final class NotesViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .systemBackground
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        setupView()
    }

    private func setupView() {

        setupNavBar()
    }

    private func setupNavBar() {
        self.title = "Notes"
    }
}
```

Point 3

We need to add the @objc function which will be presenting our controller and attach it to the corresponding notes button. Warning: we are using here some presentational features of iOS15, if your app target is lover you can present controller modally.

Point 4

To display cells our ViewController must contain UITableView and conform to UITableViewDelegate, UITableViewDataSource protocols.

Key functions:

```
func tableView(_ , numberOfRowsInSection) -> Int { }
```

Returns number of rows of cells with respect to section in our tableview.

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

Returns cell with respect to index path. Each index in an index path represents the

index into an array of children from one node in the tree to another, deeper, node.

```
import UIKit

final class NotesViewController: UIViewController {

    private let tableView = UITableView(frame: .zero, style: .insetGrouped)
    private var dataSource = [ShortNote]()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .systemBackground
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        setupView()
    }

    private func setupView() {
        setupTableView()
        setupNavBar()
    }

    private func setupTableView() {
        tableView.register(NoteCell.self, forCellReuseIdentifier:
NoteCell.reuseIdentifier)

        view.addSubview(tableView)
        tableView.backgroundColor = .clear
        tableView.keyboardDismissMode = .onDrag
        tableView.dataSource = self
        tableView.delegate = self

        view.addSubview(tableView)
        tableView.pin(to: self.view)
    }

    private func setupNavBar() {
        self.title = "Notes"
    }

    private func handleDelete(indexPath: IndexPath) {
        dataSource.remove(at: indexPath.row)
        tableView.reloadData()
    }
}

extension NotesViewController: UITableViewDataSource {

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

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
            let note = dataSource[indexPath.row]
            if let noteCell = tableView.dequeueReusableCell(withIdentifier:
NoteCell.reuseIdentifier, for: indexPath) as? NoteCell {
                noteCell.configure(note)
                return noteCell
            }
        }
        return UITableViewCell()
    }
}

extension NotesViewController: UITableViewDelegate { }
```

As you can see we are using array of ShortNotes as dataSource. ShortNote is the naive struct, which represents the model of our note. We should create new file with the following structure.

```
import Foundation

struct ShortNote {
    var text: String
}
```

Also we are using NoteCell class. It's our custom cell for note representation.

Point 5

This is already done in previous point. To check that this requirement is satisfied we can «fake» the dataSource array. Lets manually insert several Nates into array and check.

```
private var dataSource = [
    ShortNote(text: "a"),
    ShortNote(text: "b"),
    ShortNote(text: "c")
]
```

Point 6

Now It's time for input cell creation. First of all we need to create the AddNoteCell class.

Note: reuseIdentifier must be unique for every type of the cell in order. UITableView uses it in order to reuse Cells (for performance reasons). We implemented it as a static property for ease of use and clarity in the code.

We also need to register cell with its corresponding reuseIdentifier in the tableView as we did before. We did it in order to provide tableview with information how to create new cells. If a cell of the specified type isn't currently in a reuse queue, the table view uses the provided information to create a new cell object automatically. Note, If you previously registered a class or nib file with the same reuse identifier, the class you specify in the cellClass parameter replaces the old entry.

```
import UIKit

final class AddNoteCell: UITableViewCell {
    static let reuseIdentifier = "AddNoteCell"
    private var textView = UITextView()
    public var addButton = UIButton()

    // MARK: - Init
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        self.selectionStyle = .none
        setupView()
    }

    override func layoutSubviews() {
        super.layoutSubviews()
    }

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

    private func setupView() {
        textView.font = .systemFont(ofSize: 14, weight: .regular)
        textView.textColor = .tertiaryLabel
        textView.backgroundColor = .clear
        textView.setHeight(to: 140)

        addButton.setTitle("Add new note", for: .normal)
        addButton.titleLabel?.font = .systemFont(ofSize: 16, weight: .medium)
        addButton.setTitleColor(.systemBackground, for: .normal)
        addButton.backgroundColor = .label
        addButton.layer.cornerRadius = 8
        addButton.setHeight(to: 44)
        addButton.addTarget(self, action: #selector(addButtonTapped(_:)),
for: .touchUpInside)
        addButton.isEnabled = false
        addButton.alpha = 0.5

        let stackView = UIStackView(arrangedSubviews: [textView, addButton])
        stackView.axis = .vertical
        stackView.spacing = 8
        stackView.distribution = .fill

        contentView.addSubview(stackView)
        stackView.pin(to: contentView, [.left: 16, .top: 16, .right: 16, .bottom: 16])
        contentView.backgroundColor = .systemGray5
    }

    @objc
    private func addButtonTapped(_ sender: UIButton) {

    }
}
```

We need two separate sections for these cell types. Let's code this.

```
func numberOfSections(in tableView: UITableView) -> Int { }
```

Using the following function we can tell how many different sections we want to have in the tableview.

In order to return values with respect to the section in other functions - we switch section to indexPath.section

```
extension NotesViewController: UITableViewDataSource {
    func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
{
        switch section {
        case 0:
            return 1
        default:
            return dataSource.count
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        switch indexPath.section {
        case 0:
            if let addNewCell = tableView.dequeueReusableCell(withIdentifier:
AddNoteCell.reuseIdentifier, for: indexPath) as? AddNoteCell {
                addNewCell.delegate = self
                return addNewCell
            }

        default:
            let note = dataSource[indexPath.row]
            if let noteCell = tableView.dequeueReusableCell(withIdentifier:
NoteCell.reuseIdentifier, for: indexPath) as? NoteCell {
                noteCell.configure(note)
                return noteCell
            }
        }
        return UITableViewCell()
    }
}
```

Point 7

In order to pass text string to our NoteView Controller we should implement AddNoteDelegate protocol

```
protocol AddNoteDelegate: AnyObject {
    func newNoteAdded(note: ShortNote)
}

final class AddNoteCell: UITableViewCell {
    static let reuseIdentifier = "AddNoteCell"
    private var textView = UITextView()
    public var addButton = UIButton()

    var delegate: AddNoteDelegate?

    //..

    @objc
    private func addButtonTapped(_ sender: UIButton) {
        updateUI()
        delegate?.newNoteAdded(note: ShortNote(text: textView.text))
        clearTextView()
    }
}
```

And our NoteViewController should conform to NewNoteDelegate protocol.

```
extension NotesViewController: AddNoteDelegate {
    func newNoteAdded(note: ShortNote) {
        dataSource.insert(note, at: 0)
        tableView.reloadData()
    }
}
```

Point 8

Now we need to implement deletion of the row.

```
//inside NotesViewController class
private func handleDelete(indexPath: IndexPath) {
        dataSource.remove(at: indexPath.row)
        tableView.reloadData()
    }
```

```
extension NotesViewController: UITableViewDelegate {
    func tableView(_ tableView: UITableView, trailingSwipeActionsConfigurationForRowAt
indexPath: IndexPath) -> UISwipeActionsConfiguration? {

        let deleteAction = UIContextualAction(
            style: .destructive,
            title: .none
        ) { [weak self] (action, view, completion) in
            self?.handleDelete(indexPath: indexPath)
            completion(true)
        }
        deleteAction.image = UIImage(
            systemName: "trash.fill",
            withConfiguration: UIImage.SymbolConfiguration(weight: .bold)
        )?.withTintColor(.white)
        deleteAction.backgroundColor = .red
        return UISwipeActionsConfiguration(actions: [deleteAction])
    }
}
```

This tableView function allows to configure swipe actions to display on the trailing edge of the row. In the similar way functionality of «Mark as Favorite/Read» can be implemented.

Also let's make life of the user easier and add button to NotesViewController to dismiss it.

```
private func setupNavBar() {
        self.title = "Notes"

        let closeButton = UIButton(type: .close)
        closeButton.addTarget(self, action: #selector(dismissViewController(_:)),
for: .touchUpInside)
        self.navigationItem.rightBarButtonItem = UIBarButtonItem(customView:
closeButton)
    }
```

```
@objc
    func dismissViewController(_ sender: UIButton) {
        self.dismiss(animated: true, completion: nil)
    }
```

Info

If your project contains interesting solutions, exceptional code style, good presentation, unique references or well designed clean code (patterns, paradigms) up to two bonus points might be added. This means the maximum grade a student can receive per homework is 12. If interesting solution repeats in different projects bonus points won't be added. It is pointless to try and prove to us that your solution is interesting. There won't be any appeals regarding bonus points.

If the task is overdue, for each day after deadline your maximum grade goes down by 10%. Bonus points might not be added for projects submitted late. The maximum penalty is 50%. Outdated submission can get you up to 5 points, but if you already submitted and received a grade you can submit points you didn't manage to do in time. Therefor you can make your grades better.

The grade will be decreased for code style violation and bad code. If you stumble across any questions feel free to contact teachers or assistants! Our goal is to help you develop skills required to create iOS Apps.

Conclusion

We have mastered working with UITableView and UITableViewCells, also learned how to pass values between cell and table and created nice looking quick note app.