NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

HOMEWORK #2

dedicated to Research Seminar
«Immersion in iOS Development»

MOSCOW 2023

Theme: Using UIKit and Auto-layout to grant simple wishes.
Objective: Learn how to **never use Storyboard ever again**. Gain experience with UIKit and auto-layout, constraints in particular.

Task description

You will create a WishMaker app without a storyboard. You will add the first wish-granting feature: superpower to change colors! To do this we will learn auto-layout and some common UIKit components. You will be developing the WishMaking app in HW3 and HW4 too, so try and write the code in a way that won't haunt you in the future.

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	The app has no Main storyboard and builds succesfully.
2	Running the app shows a WishMakerViewController.
3	WishMakerViewController has a welcoming title and a description.
4	Created a custom slider.
5	Custom sliders change background color.
6	Code does not contain magic numbers.
7	All **«Questions»** are answered in project's README.md.
8	Color changing sliders can be hidden and shown again by a button press.
10	3 or more ways to change background color: (HEX, color picker, random).

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.

Do not hesitate to ask any 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.

Code style violations and poor solutions will lead to a grade decrease starting homework #3.

Tutorial

Name your project as follows: *your study email before «@»*PW*homework number*. Example: gmsosnovskiyPW2. **Make sure** you selected the **Storyboard** as an interface for this task.

Point 1:

During our previous task, we learned how to use a storyboard for iOS app building. Sadly, the storyboard has many issues that prevent us from using it in real life. To enable you to have the most practical skills that can be monetized in the future we have to **forbid you** from using storyboards during this course.

- **Question:** What issues prevent us from using storyboards in real projects?

Now we need to learn how to erase storyboards from existence! Press cmd + 4 or press the search icon in your left side menu for your project (Image 6).

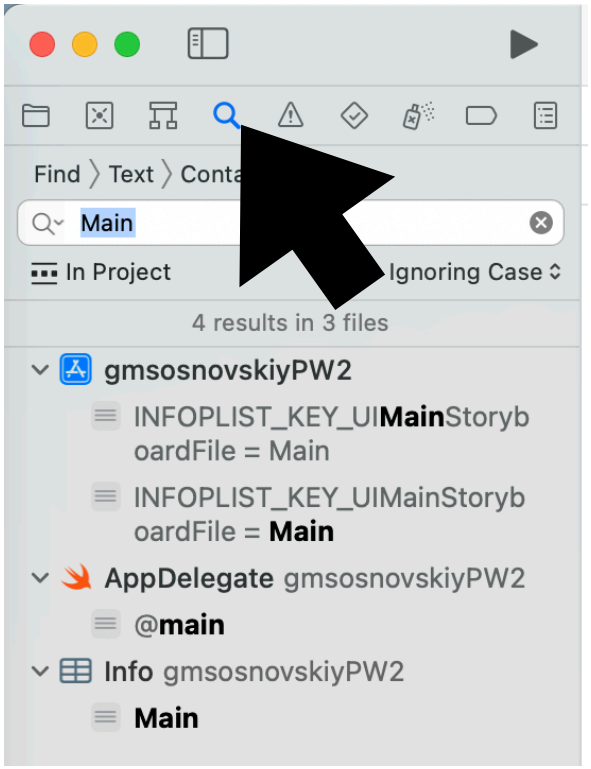

Image 6

Here we can see 4 occurrences of the main in our project. Two of these need to go. First, select the one inside info.plist (the bottom one on Image 6). We can remove the entire row, so just press a minus icon (Image 7). One gone, one more to go.

Image 7

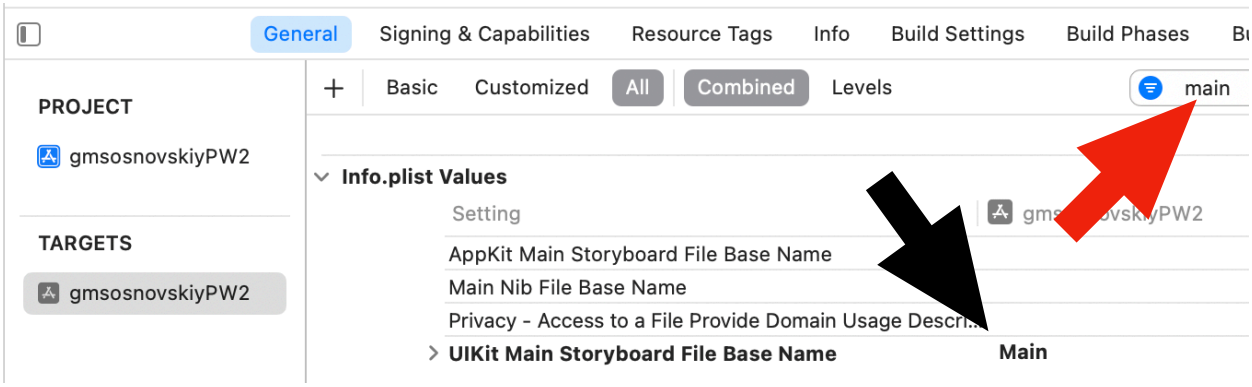

Image 8

Next, we need to delete the value of the top main (Image 6). The issue is when you press on the search result it shows you the general tab, but not the main itself. Use a filter on the top right to find the main culprit (Image 8, red arrow).
Now that we found the Base Name value (Image 8, black arrow) we can select it by tapping on it **once** and pressing backspace. This should remove the main from the value column. This makes it two mains removed from the project.

But our battle is not over yet! The last step is to press cmd + 1 or tap the folder icon (Image 9, red arrow) and move Main (Image 9, black arrow) to the trash by pressing the backspace. Run your app to confirm the successful build. You are going to see a black screen, but we will change that in the next point.

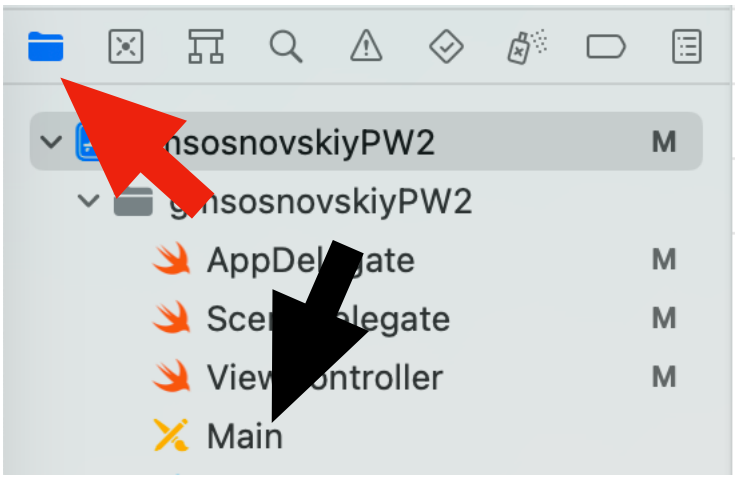

Image 9

Point 2:

Heartless darkness is not something the WishMaker app should look like. We want to add a screen to our app. To do so let's create a new WishMakerViewController:

```
import UIKit

final class WishMakerViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemPink
    }
}
```

Don't forget to rename or delete the old ViewController file so that the only file with a UIViewController heir is WishMakerViewController. By setting the view's background to a custom color we will be able to see whether or not we put our view controller above heartless darkness. The next step is to put our screen as the root view controller of our app. To do so we will open the SceneDelegate file and find **func scene(_ scene: UIScene, willConnectTo session: …)** method. There we set our WishMakerViewController as the root view controller by writing this:

```
guard let windowScene = (scene as? UIWindowScene) else { return }

let window = UIWindow(windowScene: windowScene)
window.rootViewController = WishMakerViewController()
self.window = window
window.makeKeyAndVisible()
```

Window property already exists in the SceneDelegate, so we assign a new instance with our screen set as root. Making it key and visible allows us to put it in the screen stack of our app. UIWindow is a wrap for all the screens in our app.

If we were to develop this app with iPhone 6 users in mind we would have to support iOS 12. By default, any app you create supports the latest iOS available by the time Xcode was released. This is not ideal, since this eliminates 2 previous iOS versions that deserve your attention for all those users who refuse to update their devices. Try setting your supported iOS version to iOS 13 (Image 10). This version has all the most used development tools present and teaches you code style and what features were added in what version.

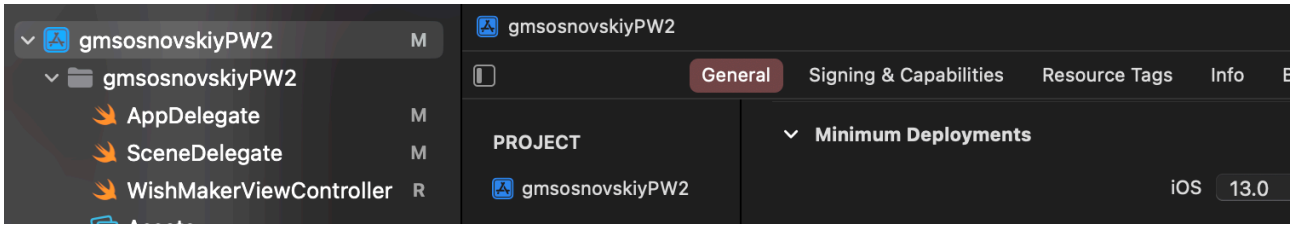

Image 10

Point 3:

To add a title, we must learn best practices for building UI through code via UIKit. There are two ways of building.

The first method is using frames. Any UIView has a frame and bounds. The frame represents the view's x and y coordinates relative to the top left corner of the superview. Superview is a parent view that contains the child view. Bounds represent the view's own coordinate system, with its origin at the top-left corner of the view itself. This approach is efficient when it comes to rendering since there are no complex calculations that define the view's position, but a simple combination of two coordinates, width and height. This combination in Swift is called core graphics rectangle or CGRect for short.

The second method is using autolayout. We are going to focus on this approach. Autolayout in iOS replaces fixed coordinates with rules and constraints. These constraints define how UI elements relate to each other and adapt to different screen sizes and orientations. It ensures a responsive and visually consistent user interface across various iOS devices, making it essential for your career and this course.

Let us create a private function named **configure**UI() and call it from the viewDidLoad (after the super). This is a good practice. Sometimes instead of «**configure**» people use «**setup**» in their UI-related code, this would indicate them being new to iOS development.

```
17.     private func configureUI() {
18.         view.backgroundColor = .systemPink
19.
20.         configureTitle()
21.     }
22.
23.     private func configureTitle() {
24.         let title = UILabel() // make it a private field titleView above viewDidLoad()
25.         title.translatesAutoresizingMaskIntoConstraints = false
26.         title.text = "WishMaker"
27.         title.font = UIFont.systemFont(ofSize: 32)
28.
29.         view.addSubview(title)
30.         NSLayoutConstraint.activate([
31.             title.centerXAnchor.constraint(equalTo: view.centerXAnchor),
32.             title.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 20),
33.             title.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 30)
34.         ])
35.     }
```

On line 24 we create a new label, that is going to be our title. We set its text and font (lines 26-27). Lines 25 and 29 are extremely important. First, run the app with both of them present. Then remove line 25 and see what changed. Bring it back and try running the app without line 29.

Question: What does the code on lines 25 and 29 do?

Lines 30 to 34 are quite long, the font size had to be 8 for them to fit on this page. Those are constraints we set to describe our label's location on the screen. On line 31 we set the label's center to be equal to our screen's horizontal center. On line 32 we say, that the left side of our label must be equal to the screen's left side, but 20 points to the right. On line 33 we make the top of our label to be placed 50 points below the screen's safe area's top anchor.

Question: What is a safe area layout guide?

This point is not over yet! You need to add a description based on the title example. A perfect student would also make the title **bold** and change its text color. Perhaps, we can use the UIColor extension from the previous homework? The description label must be added in a function configureDescription(), and be placed 20 points below the bottom of the title. Horizontal constraints can be copied from the title.

Point 4:

Creating custom views in iOS is an extremely common task. We have to learn how to do it:

```
10.final class CustomSlider: UIView {
11.    var valueChanged: ((Double) -> Void)?
12.
13.    var slider = UISlider()
14.    var titleView = UILabel()
15.
16.    init(title: String, min: Double, max: Double) {
17.        super.init(frame: .zero)
18.        titleView.text = title
19.        slider.minimumValue = Float(min)
20.        slider.maximumValue = Float(max)
21.        slider.addTarget(self, action: #selector(sliderValueChanged), for: .valueChanged)
22.        configureUI()
23.    }
24.
25.    @available(*, unavailable)
26.    required init?(coder: NSCoder) {
27.        fatalError("init(coder:) has not been implemented")
28.    }
29.
30.    private func configureUI() {
31.        backgroundColor = .white
32.        translatesAutoresizingMaskIntoConstraints = false
33.
34.        for view in [slider, titleView] {
35.            addSubview(view)
36.            view.translatesAutoresizingMaskIntoConstraints = false
37.        }
38.
39.        NSLayoutConstraint.activate([
40.            titleView.centerXAnchor.constraint(equalTo: centerXAnchor),
41.            titleView.topAnchor.constraint(equalTo: topAnchor, constant: 10),
42.            titleView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 20),
43.
44.            slider.topAnchor.constraint(equalTo: titleView.bottomAnchor),
45.            slider.centerXAnchor.constraint(equalTo: centerXAnchor),
46.            slider.bottomAnchor.constraint(equalTo: bottomAnchor, constant: -10),
47.            slider.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 20)
48.        ])
49.    }
50.
51.    @objc
52.    private func sliderValueChanged() {
53.        valueChanged?(Double(slider.value))
54.    }
55.}
56.
```

This is a simple custom view since we inherit UIView on line 10. It consists of a slider on line 13 and a title on line 14. If you want you can add one more label that will show the slider's current value. On line 16 we have 3 parameters as an initializer's semantics. The first parameter is a text for our slider's title, the second is the minimum value of the slider, and the third is the maximum value. Moving the slider's head will change the slider's value to a new one in the range between min and max correlating with the head position. Now we can create custom sliders like this:

```
let sliderRed = CustomSlider(title: "Red", min: 0, max: 1)
```

Point 5:

In this tutorial, we help you learn the best ways to develop iOS apps, but you are more than capable of doing some tasks yourself. Here we provide you with a way of placing the sliders on the screen:

```
1.      private func configureSliders() {
2.          let stack = UIStackView()
3.          stack.translatesAutoresizingMaskIntoConstraints = false
4.          stack.axis = .vertical
5.          view.addSubview(stack)
6.          stack.layer.cornerRadius = 20
7.          stack.clipsToBounds = true
8.
9.          let sliderRed = CustomSlider(title: "Red", min: 0, max: 1)
10.         let sliderBlue = CustomSlider(title: "Blue", min: 0, max: 255)
11.         let sliderGreen = CustomSlider(title: "Green", min: 0, max: 255)
12.
13.         for slider in [sliderRed, sliderBlue, sliderGreen] {
14.             stack.addArrangedSubview(slider)
15.         }
16.
17.         NSLayoutConstraint.activate([
18.             stack.centerXAnchor.constraint(equalTo: view.centerXAnchor),
19.             stack.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 20),
20.             stack.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: -40)
21.         ])
22.
23.         sliderRed.valueChanged = { [weak self] value in
24.             self?.view.backgroundColor = ...
25.         }
26.     }
```

On line 11 of our CustomSlider we have a valueChanged field, that is a placeholder for an anonymous function. It can be used as shown above on lines 23 to 25. The value on the line 23 is a new value of the red slider. The anonymous function will be called once the user changes the value of the red slider.

On line two we create a thing known as a UIStackView (UIStack for short). This is a stack view that allows you to show arranged subviews (line 14) without setting constraints for all of them. The stack acts as a wrap and setting it's width and bottom anchor is enough to arrange all views (if they have a defined height).

What to do next in this case is up to you! Come up with the best way to change the background color and maybe gain extra points.

Question: What is [weak self] on line 23 and why it is important?
Question: What does clipsToBounds mean?
Question: What is the valueChanged type? What is Void and what is Double?

Point 6:

In the first tutorial, one of the bonus points was to remove magic numbers and place MARKs in your code. This time we will showcase the best way to store your values in the code to show the interviewers how amazing your iOS development skills are! In Swift, enums act as a namespace, where you can add static properties. This allows you to extract random numbers, text, and even fonts to an enum on the top of your class, which makes it easy to change later.

Order of definition inside a class:

- Typealiases
- Enums
- Fields
- Properties
- Lifecycle methods
- Public methods
- Private methods

Our code with magic numbers looks something like this:

```
stack.layer.cornerRadius = 20
stack.clipsToBounds = true

let sliderRed = CustomSlider(title: "Red", min: 0, max: 1)
let sliderBlue = CustomSlider(title: "Blue", min: 0, max: 1)
let sliderGreen = CustomSlider(title: "Green", min: 0, max: 1)

for slider in [sliderRed, sliderBlue, sliderGreen] {
    stack.addArrangedSubview(slider)
}

NSLayoutConstraint.activate([
    stack.centerXAnchor.constraint(equalTo: view.centerXAnchor),
    stack.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 20),
    stack.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: -40)
])
```

After moving all the ambiguous values to the Constants enum our code will look like this:

```
enum Constants {
    static let sliderMin: Double = 0
    static let sliderMax: Double = 1

    static let red: String = "Red"
    static let green: String = "Green"
    static let blue: String = "Blue"

    static let stackRadius: CGFloat = 20
    static let stackBottom: CGFloat = -40
    static let stackLeading: CGFloat = 20
}

...

private func configureSliders() {
    let stack = UIStackView()
    stack.translatesAutoresizingMaskIntoConstraints = false
    stack.axis = .vertical
    view.addSubview(stack)
    stack.layer.cornerRadius = Constants.stackRadius
    stack.clipsToBounds = true

    let sliderRed = CustomSlider(title: Constants.red, min: Constants.sliderMin, max: Constants.sliderMax)
    let sliderGreen = CustomSlider(title: Constants.green, min: Constants.sliderMin, max: Constants.sliderMax)
    let sliderBlue = CustomSlider(title: Constants.blue, min: Constants.sliderMin, max: Constants.sliderMax)

    for slider in [sliderRed, sliderGreen, sliderBlue] {
        stack.addArrangedSubview(slider)
    }

    NSLayoutConstraint.activate([
        stack.centerXAnchor.constraint(equalTo: view.centerXAnchor),
        stack.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: Constants.stackLeading),
        stack.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: Constants.stackBottom)
    ])
}
```

This code is lengthier but is more readable and modifiable. Try to do this to all magic numbers from now on!

Point 7:

For your qualifications as an iOS developer, it is essential to know the answers to the questions provided. Try to understand the answers you give, this will be on the test.

Conclusion:

This app does not do much, but it serves as a foundation for something bigger. If done right it will be a good addition to your portfolio.

The code provided in the tutorial utilizes the most basic constraints usage. During the seminar, we created a pin methods extension for UIView to make it easier. Example:

```
// Classic approach
stack.translatesAutoresizingMaskIntoConstraints = false
NSLayoutConstraint.activate([
    stack.centerXAnchor.constraint(equalTo: view.centerXAnchor),
    stack.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: Constants.stackLeading),
    stack.bottomAnchor.constraint(equalTo: view.bottomAnchor, constant: Constants.stackBottom)
])

// Pin layout
stack.pinBottom(to: view, -1 * Constants.stackBottom) // bottom makes constant negative for us
stack.pinHorizontal(to: view, Constants.stackLeading)
```

If you successfully finish this tutorial you can get up to **7 points**. This is a good grade, to earn more you need to do the remaining points on your own. You would have to learn to hide views and other ways of setting a background color. Show some creativity, and show off your programming skills. **Astonish** homework inspectors and gain bonus points and respect!

Example of the finished app: