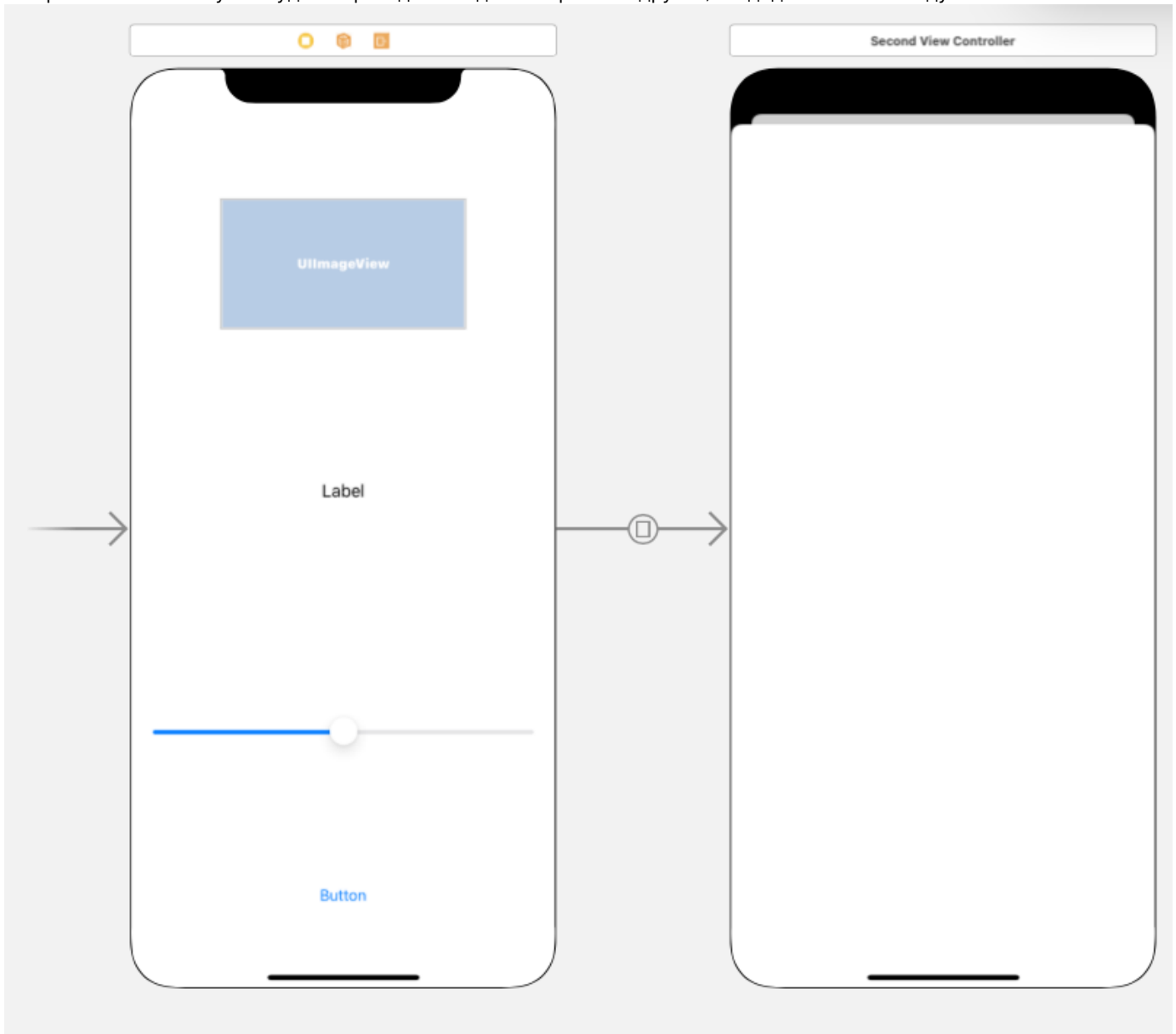


Часть 6. Итоговое приложение, использование сторонних API

Наш финальный проект сможет принести пользу всей компании, ведь мы создадим агрегатор настроения сотрудников! Каждый юзер сможет открыть его, выбрать уровень сегодняшнего настроения, а приложение посчитает его голос и покажет средний результат среди всех проголосовавших.

Для начала спроектируем интерфейс. На этот раз дадим фантазии разыграться - все элементы мы придумаем сами, без помощи макетов в Фигме. В качестве стартового набора возьмем 2 экрана (добавим новый *ViewController*). На первом отобразим *UIImageView*, *Label* с текущей оценкой, кнопку для публикации этой оценки и *Slider* для выбора настроения. Поскольку мы будем переходить с одного экрана на другой, создадим сегвей между ними

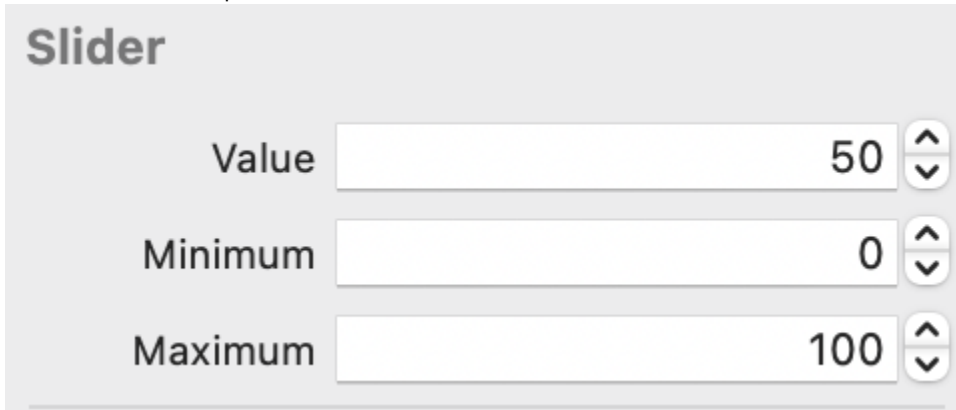


Следующим шагом протянем связи от всех элементов главного контроллера в его класс - *ViewController.swift*

```
@IBOutlet weak var moodSlider: UISlider!  
@IBOutlet weak var moodLabel: UILabel!  
@IBOutlet weak var publishMoodButton: UIButton!  
@IBOutlet weak var moodImageView: UIImageView!
```

И создадим там же два `IBAction` - один для кнопки, а второй для слайдера. Обратите внимание - при создании функции для слайдера в поле *Event* появилось новое значение - *ValueChanged* - это значит, что функция будет выполняться каждый раз, когда у слайдера меняется значение

Для того, чтобы определить, по какой шкале пользователь будет выставлять оценку, откроем сториборд и посмотрим на свойства слайдера. Здесь можно выставить его минимальное, максимальное и первоначальное значения



Теперь определим логику работы слайдера. Когда у него появляется новое значение, наш *Label* должен его отображать, а картинка в *ImageView* будет иллюстрировать выбранную оценку. Начнем с лейбла. Сделаем так, чтобы значение слайдера округлилось (для этого есть функция *rounded*), потом сконвертируем его в строку и поместим в текст лейбла

```
@IBAction func sliderValueChanged() {
    let roundedValue = moodSlider.value.rounded()
    moodLabel.text = String(format: "%.0f", roundedValue)
}
```

Использование `format: "%.0f"` в этой функции нужно для того, чтобы отформатировать числовое значение, которое даже после округления выглядит как число с плавающей точкой (например, 85.0). Форматирование сообщает, что мы хотим видеть число, при этом после точки должно быть 0 знаков.

Для установки разных картинок вспомним про оператор условия. У него есть "многоэтажная" конструкция - если первое условие не выполнится, мы можем проверить еще одно, и еще, и еще

```
if <1> {
    <1>
} else if <2> {
    <2>
} else if <3> {
    <3>
}
```

Применим эту логику к нашему слайдеру. Если его значения ниже 20, покажем грустную картинку, если нет - проверим, вдруг оно ниже 40, иначе проверим что оно ниже 60 и так далее. Эту логику также добавим внутрь функции `sliderValueChanged()`

```

if slider.value < 20 {
    moodImageView.image = UIImage(systemName: "cloud.rain")
} else if slider.value < 40 {
    moodImageView.image = UIImage(systemName: "cloud")
} else if slider.value < 60 {
    moodImageView.image = UIImage(systemName: "faceid")
} else if slider.value < 80 {
    moodImageView.image = UIImage(systemName: "face.smiling")
} else {
    moodImageView.image = UIImage(systemName: "flame")
}

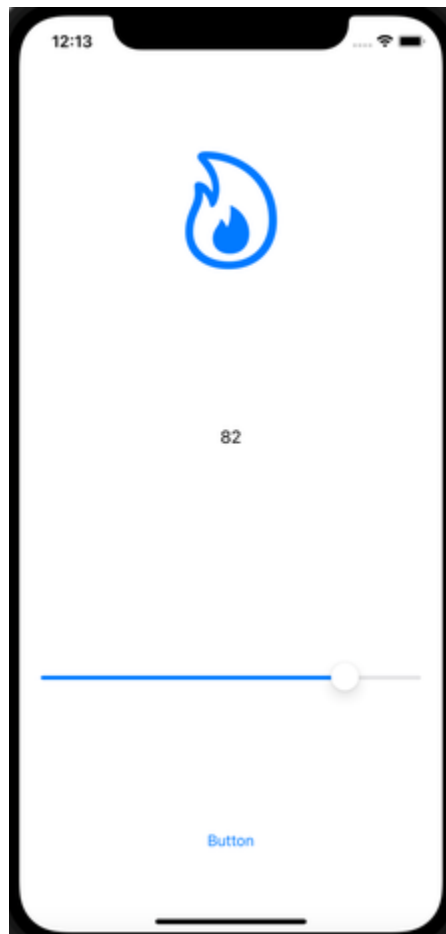
```

Здесь мы используем системные иконки. Если хотите персонализировать интерфейс, нужно добавить свои картинки в папку Assets и вызывать их, используя такую конструкцию (вместо *myImageName* нужно вписать название своей картинки)

```

imageView.image = UIImage(named: "myImageName")

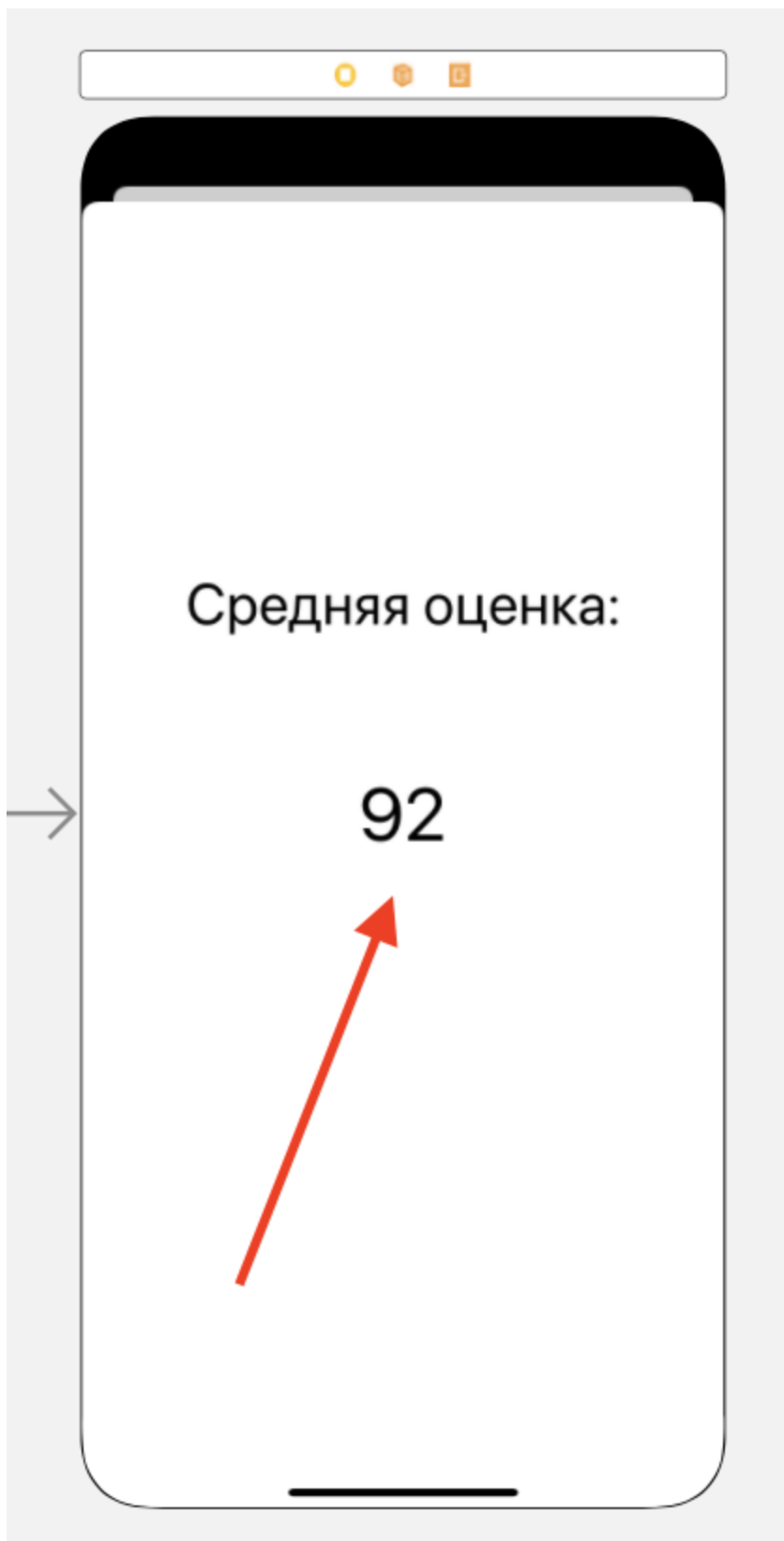
```



Для кнопки внизу добавим сегвей (перед этим обязательно пропишем его *identifier* в сториборде)

```
@IBAction func didTapMoodButton() {  
    performSegue(withIdentifier: "goToSecondScreen", sender: self)  
}
```

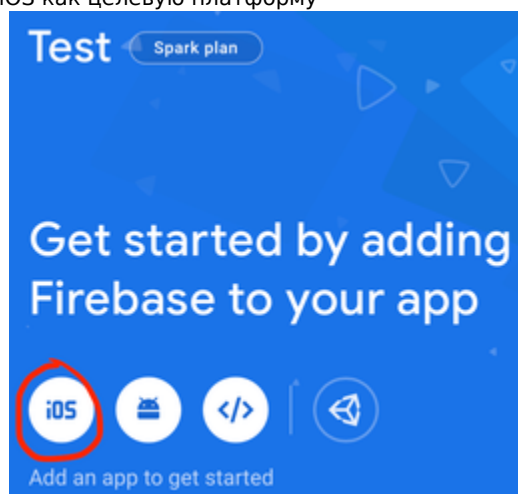
И сконфигурируем второй экран. Как и в [предыдущем приложении](#), создадим класс *SecondViewController* и укажем его в *Identity Inspector*'е второго экрана. Добавим пару лейблов, и соединим один из них с кодом - в нём будет отображаться средняя оценка настроения



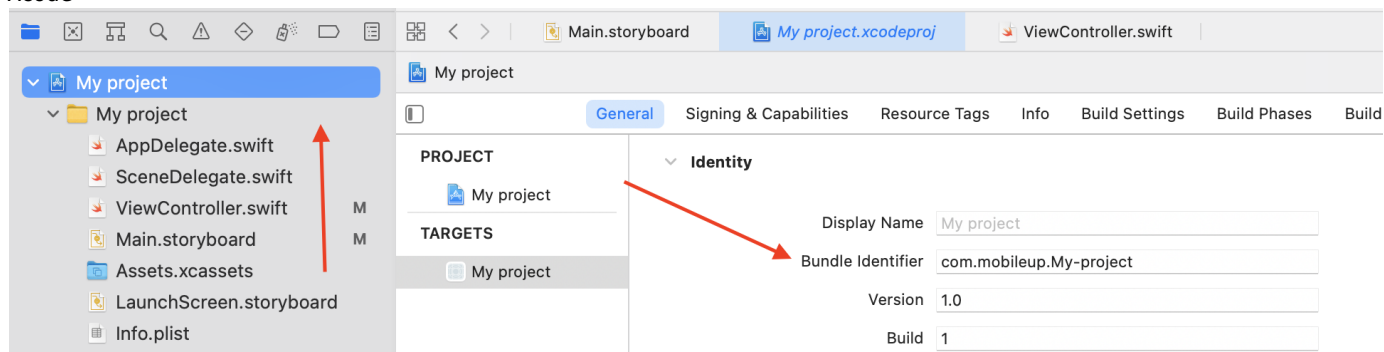
Впереди самый важный шаг - добавление базы данных! Для этого нам понадобится сторонняя библиотека *Firebase* от *Google*. По сути это целый комбайн из сервисов, среди которых есть облачные данные и вычисления, пуш-уведомления и сами базы данных. Для их использования нужно создать проект *Firebase* [по этой ссылке](#) (большая кнопка *Add Project*) и пройти несколько шагов:

1. Придумать имя проекта (оно может быть любым) и перейти на следующий экран
2. Отключить Google-аналитику (она нам не пригодится) и нажать *Create Project*

3. На появившемся экране выбрать iOS как целевую платформу



Далее нам предлагается зарегистрировать приложение в Firebase. На первом шаге нужно заполнить поле *iOS Bundle ID*. Он должен совпадать с таковым в нашем проекте. Посмотреть *Bundle ID* можно, выбрав файл проекта в левой панели Xcode












Шаг второй - добавить файл конфигурации. Для этого нужно скачать *GoogleService-Info.plist*, который предлагается на сайте, и перетащить его к другим файлам в Xcode

Шаг третий - добавить библиотеку от Google в наше приложение. Вопреки инструкции, мы используем более простой и современный метод. Вместо написания кода в командной строке, просто откроем Xcode и в верхней строке выберем File → Swift Packages → Add Package Dependency... Далее в появившемся окне вставим ссылку

<https://github.com/firebase/firebase-ios-sdk.git>

Choose Package Repository:





















Name	Last Updated	Owner
denis.sushkov (GitLab.com)		
 MileOnAir iOS		mobileup
 MU Framework iOS		mobileup
 Mutal		mobileup
Aerostat (GitLab.com)		
 Aerostat Android		mobileup
 Aerostat iOS		mobileup
 Aerostat Mobile API		mobileup
 Aerostat Website		mobileup
Anketka (GitLab.com)		
 Anketka Android		mobileup

CancelPreviousNext

И после нажатия кнопки “далее”, по этой ссылке Xcode сам найдет и загрузит нужную нам библиотеку! Соглашаемся со всеми настройками, ждём загрузку, и в окне с выбором библиотек отмечаем галочкой только пару нужных

Add Package to My project:

Choose package products and targets:

Package Product	Kind	Add to Target
<input type="checkbox"/>  FirebaseAuth	Library	 My project ⇅
<input type="checkbox"/>  FirebaseCrashlytics	Library	 My project ⇅
<input checked="" type="checkbox"/>  FirebaseDatabase	Library	 My project ⇅
<input checked="" type="checkbox"/>  FirebaseDatabaseSwift-B	Library	 My project ⇅
<input type="checkbox"/>  FirebaseDynamicLinks	Library	 My project ⇅
<input type="checkbox"/>  FirebaseFirestore	Library	 My project ⇅
<input type="checkbox"/>  FirebaseFirestoreSwift-Be	Library	 My project ⇅
<input type="checkbox"/>  FirebaseFunctions	Library	 My project ⇅
<input type="checkbox"/>  FirebaseInAppMessaging-	Library	 My project ⇅

CancelPreviousFinish

Теперь у нашего приложения есть доступ к стороннему API! Чтобы воспользоваться его функциями, в файле с кодом нужно в самом верху импортировать эту библиотеку. Так и поступим в следующем шаге.

Согласно подсказке от гугла, нам нужно внести изменения в файл *AppDelegate.swift*. Это файл отвечает за глобальные изменения в приложении в моменты запуска, закрытия или сворачивания в фоновый режим. Настройка состоит всего из двух строчек (во вставке ниже это 2 и 8 строки)

```
import UIKit
import Firebase

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [UIApplication.
LaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()
        return true
    }

    <...>

}
```

На второй строке мы импортировали библиотеку, чтобы файл мог использовать все свойства и методы библиотеки Firebase, а на восьмой - используя эти методы, включили настройку Firebase при запуске приложения.

Вернемся в консоль Firebase, чтобы создать базу данных с необходимыми полями. В панели справа выберем пункт *Realtime Database* и нажмем кнопку *Create Database* в шапке страницы. Регион базы данных оставим по умолчанию и на следующем шаге выберем пункт *Start In Test Mode*

Set up database



1 Database options — 2 Security rules

Once you have defined your data structure **you will have to write rules to secure your data.**

[Learn more](#)

☐ Start in **locked mode**

Your data is private by default. Client read/write access will only be granted as specified by your security rules.

☒ Start in **test mode**

Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
{
  "rules": {
    ".read": "now < 1630962000000", // 2021-9-7
    ".write": "now < 1630962000000", // 2021-9-7
  }
}
```

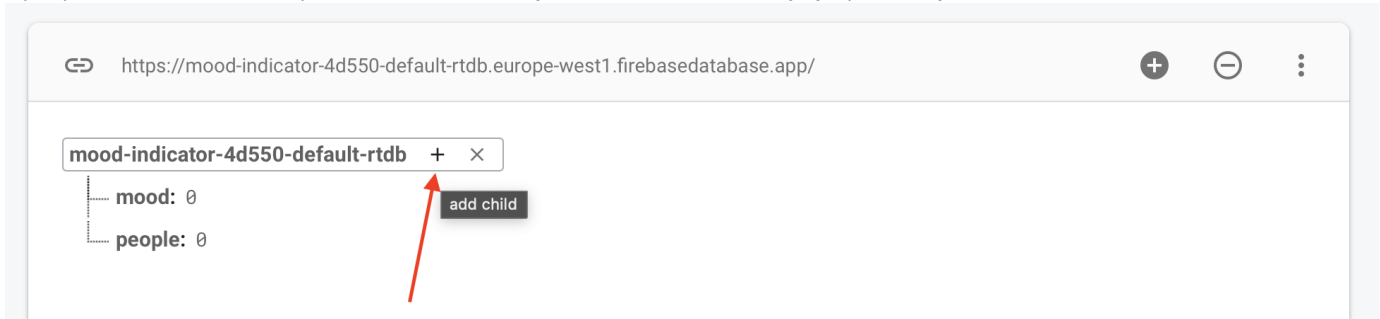


The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days

Cancel

Enable

Наша новая база данных пока пуста. По нажатию на “+” создадим в ней два поля: *mood* для общей оценки настроения и *people* для количества проголосовавших. По умолчанию оба поля будут равны нулю



И снова вернемся к нашему приложению, чтобы настроить получение и отправку этих данных. Google предоставляет набор функций, которые позволяют это сделать, поэтому просто добавим настройку базы данных и получение нужных полей в файл контроллера

```
var databaseReference: DatabaseReference!

var mood: Float = -1
var people: Int = -1
var isAvailable: Bool = true

override func viewDidLoad() {
    super.viewDidLoad()

    databaseReference = Database.database().reference()
    getMood()
}

func getMood() {
    databaseReference.getData { [weak self] (error, snapshot) in

        if let error = error {
            print("Error getting data \(error)")
        } else if snapshot.exists() {
            self?.mood = snapshot.childSnapshot(forPath: "mood").
value as! Float
            self?.people = snapshot.childSnapshot(forPath:
"people").value as! Int
        } else {
            print("No data available")
        }
    }
}
```

Здесь мы объявляем ссылку (референс) на базу данных, создаем три переменные (настроение, количество людей и “доступность”, о которой поговорим позже), и сразу после загрузки контроллера вызываем функцию *getMood()* - она обращается к Firebase и записывает данные в переменные *mood* и *value*

А по нажатию нашей кнопки текущие данные будут дополняться новыми и записываться в базу - но только если выполняются два условия! Первое - *mood* и *value* больше нуля (это означает, что мы получили данные, и их значения

больше не равны -1), второе `isAvailable == true` (это поможет нам запретить юзеру голосовать дважды - как только данные запишутся, переменная переключится на *false*)

```
@IBAction func moodButtonTapped() {
    if mood >= 0, people >= 0 {
        if isAvailable == true {
            databaseReference.child("people").setValue(people + 1)
            databaseReference.child("mood").setValue(mood +
moodSlider.value)
            isAvailable = false
            performSegue(withIdentifier: "goToSecondScreen",
sender: self)
        } else {
            moodLabel.text = "Try again tomorrow!"
        }
    }
}
```

Как можно заметить, после записи новых данных, мы выполняем переход на второй экран - он будет показывать среднюю оценку. Для этого экрана создадим свой класс *SecondViewController* (по аналогии с [предыдущим занятием](#)), добавим *Label* на экран и протянем связь к новому файлу.

Второй контроллер также будет запрашивать данные (которые к этому моменту уже обновились). Поэтому функция `getMood()` будет похожа - но в этот раз нам не потребуется добавлять переменные для целого класса, обойдемся парой локальных констант и сразу покажем результат.

```
var databaseReference: DatabaseReference!

override func viewDidLoad() {
    super.viewDidLoad()

    databaseReference = Database.database().reference()
    getMood()
}

func getMood() {
    databaseReference.getData { [weak self] (error, snapshot) in

        if let error = error {
            print("Error getting data \(error)")
        } else if snapshot.exists() {
            let mood = snapshot.childSnapshot(forPath: "mood").
value as! Float
            let people = snapshot.childSnapshot(forPath: "people").
value as! Int

            let averageMood = mood / Float(people)

            self?.averageMoodLabel.text = (String(format: "%.0f",
averageMood))
        }
    }
}
```

```
    }  
    else {  
        print("No data available")  
    }  
}  
}
```

И кстати, не забудьте добавить строчку `import Firebase` в заголовке файла!

Обновление базы данных в реальном времени можно наблюдать в консоли Firebase. А чтобы объединить все наши приложения для работы с одной базой, достаточно всем указать один и тот же Bundle Identifier в настройках проекта (см. выше) и заменить файл `GoogleServiceInfo.plist` на тот, который был создан для этого идентификатора.

Поздравляю! Приложение, работающее с сетью и базой данных готово! Дальше - дело за вами) Как оно будет выглядеть? Может, дополнить его новыми фичами вроде индикатора загрузки? Или сделать крутой дизайн и похвастаться друзьям? Следующие шаги можете уверенно делать сами - как настоящие iOS разработчики)

Нашкодили! Ура!