**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Руководитель проекта,  Приглашенный преподаватель Базовой кафедры «Системное программирование» Института системного программирования им. В.П. Иванникова РАН (ИСП РАН)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Кучук  **«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_** 2021 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов  **«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_** 2021 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.01-01 ТЗ 01-1-ЛУ | | **КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ГЕОТАРГЕТИРОВАННОЙ РЕКЛАМЫ «ГЕОФУД». МОБИЛЬНОЕ ПРИЛОЖЕНИЕ**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 ТЗ 01-1-ЛУ** | | |
|  |  | |
| Исполнитель:  студент группы БПИ194  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Е. В. Аникеев /  **«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** 2021 г. | |
|  | | |
|  | |  |

**Москва 2021**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 ТЗ 01-1-ЛУ |  |  | | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.01-01 ТЗ 01-1-ЛУ | | **КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ГЕОТАРГЕТИРОВАННОЙ РЕКЛАМЫ «ГЕОФУД». МОБИЛЬНОЕ ПРИЛОЖЕНИЕ**  **Текст программы**  **RU.17701729.04.01-01 ТЗ 01-1**  **Листов 112** | | | |
|  |  | | |
|  | | |
|  | | | |
|  | | |  |

**Москва** **2021**

Содержание

[1. Текст программы 4](#_Toc71752330)

[1.1. LoginForm.swift 4](#_Toc71752331)

[1.2. LoginEntryChecker.swift 4](#_Toc71752332)

[1.3. MapExtension.swift 5](#_Toc71752333)

[1.4. NextButton.swift 6](#_Toc71752334)

[1.5. InputTextField.swift 7](#_Toc71752335)

[1.6. TitledTextField.swift 9](#_Toc71752336)

[1.7. AppDelegate.swift 12](#_Toc71752337)

[1.8. SceneDelegate.swift 13](#_Toc71752338)

[1.9. RestaurantSaleModel.swift 14](#_Toc71752339)

[1.10. RestaurantModel.swift 16](#_Toc71752340)

[1.11. UserModel.swift 18](#_Toc71752341)

[1.12. User+CoreDataClass.swift 18](#_Toc71752342)

[1.13. User+CoreDataProperties.swift 19](#_Toc71752343)

[1.14. Restaurant+CoreDataClass.swift 21](#_Toc71752344)

[1.15. Restaurant+CoreDataProperties.swift 21](#_Toc71752345)

[1.16. Sale+CoreDataClass.swift 23](#_Toc71752346)

[1.17. Sale+CoreDataProperties.swift 23](#_Toc71752347)

[1.18. Location+CoreDataClass.swift 24](#_Toc71752348)

[1.19. Location+CoreDataProperties.swift 25](#_Toc71752349)

[1.20. CoreDataStore.swift 25](#_Toc71752350)

[1.21. CoordinateRequestModel.swift 27](#_Toc71752351)

[1.22. ImageLoader.swift 28](#_Toc71752352)

[1.23. Endpoints.swift 29](#_Toc71752353)

[1.24. UserService.swift 30](#_Toc71752354)

[1.25. RequestFactory.swift 34](#_Toc71752355)

[1.26. LocationManager.swift 37](#_Toc71752356)

[1.27. UserViewModel.swift 37](#_Toc71752357)

[1.28. AccountPresenter.swift 38](#_Toc71752358)

[1.29. AccountViewController.swift 39](#_Toc71752359)

[1.30. AccountInteractor.swift 42](#_Toc71752360)

[1.31. AccountRouter.swift 43](#_Toc71752361)

[1.32. AccountConfigurator.swift 43](#_Toc71752362)

[1.33. AccountConfigurator.swift 44](#_Toc71752363)

[1.34. MainViewController.swift 44](#_Toc71752364)

[1.35. RestaurantViewController.swift 45](#_Toc71752365)

[1.36. RestaurantHeaderView.swift 54](#_Toc71752366)

[1.37. SaleCell.swift 57](#_Toc71752367)

[1.38. SpecialSaleCollectionViewCell.swift 60](#_Toc71752368)

[1.39. RestaurantViewModel.swift 65](#_Toc71752369)

[1.40. SaleViewModel.swift 66](#_Toc71752370)

[1.41. RestaurantPresenter.swift 67](#_Toc71752371)

[1.42. RestaurantConfigurator.swift 68](#_Toc71752372)

[1.43. RestaurantInteractor.swift 69](#_Toc71752373)

[1.44. RestaurantAnnotation.swift 70](#_Toc71752374)

[1.45. RestaurantTypeCell.swift 71](#_Toc71752375)

[1.46. MapViewController.swift 73](#_Toc71752376)

[1.47. MapViewModel.swift 80](#_Toc71752377)

[1.48. MapRestaurantViewModel.swift 81](#_Toc71752378)

[1.49. MapPresenter.swift 82](#_Toc71752379)

[1.50. MapConfigurator.swift 86](#_Toc71752380)

[1.51. MapInteractor.swift 86](#_Toc71752381)

[1.52. MapRouter.swift 88](#_Toc71752382)

[1.53. RegistrationViewController.swift 89](#_Toc71752383)

[1.54. RegistrationConfigurator.swift 94](#_Toc71752384)

[1.55. RegistrationPresenter.swift 94](#_Toc71752385)

[1.56. RegistrationInteractor.swift 96](#_Toc71752386)

[1.57. RegistrationRouter.swift 98](#_Toc71752387)

[1.58. AuthorizationViewController.swift 99](#_Toc71752388)

[1.59. AuthorizationPresenter.swift 105](#_Toc71752389)

[1.60. AuthorizationInteractor.swift 107](#_Toc71752390)

[1.61. AuthorizationRouter.swift 109](#_Toc71752391)

[1.62. AuthorizationConfigurator.swift 110](#_Toc71752392)

[2. Список литературы 111](#_Toc71752393)

[3. Лист регистрации изменений 112](#_Toc71752394)

1. Текст программы
   1. LoginForm.swift

//

// AccountRouter.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Входные методы роутера

**protocol** AccountRouterInput: AnyObject {

/// Вернуться в корневой вью

**func** popBack()

}

**class** AccountRouter: AccountRouterInput {

/// Контроллер аккаунта

**private** **var** view: AccountViewController!

/// Конструктор с контроллером

/// - **Parameter** view: Контроллер аккаунта

**init**(with view: AccountViewController) {

**self**.view = view

}

/// Вернуться в корневой вью

**func** popBack() {

view.navigationController?.popViewController(animated: **true**)

}

}

* 1. LoginEntryChecker.swift

//

// LoginEntryChecker.swift

// GeoFood

//

// Created by Егор on 16.03.2021.

//

**import** Foundation

/// Протокол проверки данных на авторизацию/регистрацию

**protocol** LoginEntryCheckerProtocol: class {

/// Проверка формата строки для почты

/// - **Parameter** email: проверяемая строка

**static** **func** checkEmail(**\_** email: String) -> Bool

/// Провекра формата строки с паролем

/// - **Parameter** password: Проверяемая строка

**static** **func** checkPassword(**\_** password: String) -> Bool

}

/// Объект для проверки введенных данных по формату

**class** LoginEntryChecker: LoginEntryCheckerProtocol {

/// Проверка формата строки для почты

/// - **Parameter** email: проверяемая строка

**static** **func** checkEmail(**\_** email: String) -> Bool {

**let** regex = **try**! NSRegularExpression(pattern: "[A-Z0-9a-z.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,64}")

**let** range = NSRange(location: 0, length: email.count)

**return** regex.firstMatch(in: email, options: [], range: range) != **nil**

}

/// Провекра формата строки с паролем

/// - **Parameter** password: Проверяемая строка

**static** **func** checkPassword(**\_** password: String) -> Bool {

password.count > 8

}

}

* 1. MapExtension.swift

//

// MapExtension.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** Foundation

**import** MapKit

**extension** MKMapView {

/// Координаты верхней центральной точки карты

/// - **Returns**: верхняя точка видимой области карты

**private** **func** topCenterCoordinate() -> CLLocationCoordinate2D {

**return** **self**.convert(CGPoint(x: **self**.frame.size.width / 2.0, y: 0), toCoordinateFrom: **self**)

}

/// Расчёт радиуса от середины до верхней точки видимости карты

/// - **Returns**: расстояние от центра до верхней точки

**func** currentRadius() -> Double {

**let** centerLocation = CLLocation(latitude: **self**.centerCoordinate.latitude, longitude: **self**.centerCoordinate.longitude)

**let** topCenterCoordinate = **self**.topCenterCoordinate()

**let** topCenterLocation = CLLocation(latitude: topCenterCoordinate.latitude, longitude: topCenterCoordinate.longitude)

**return** centerLocation.distance(from: topCenterLocation)

}

}

* 1. NextButton.swift

//

// NextButton.swift

// GeoFood

//

// Created by Егор on 21.04.2021.

//

**import** UIKit

**@IBDesignable**

/// Кнопка со стрелкой

**class** NextButton: UIButton {

/// Конструктор класса

**init**() {

**super**.init(frame: .zero)

setUp()

}

/// Конструктор класса

/// - **Parameter** coder: Кодер

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

/// Пересчитать размеры и положение объектов

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

imageEdgeInsets.left = **self**.bounds.width - **self**.imageView!.bounds.width - 10;

titleEdgeInsets.right = **self**.imageView!.bounds.width + 15

titleEdgeInsets.left = 1

}

/// Конфигурация внутренних вью

**private** **func** setUp() {

setTitleColor(UIColor(named: "dark\_blue"), for: .normal)

backgroundColor = UIColor(named: "light\_green")

imageView?.tintColor = UIColor(named: "dark\_blue")

**self**.layer.cornerRadius = 10

setImage(UIImage(named: "right\_arrow"), for: .normal)

}

}

* 1. InputTextField.swift

//

// InputTextField.swift

// GeoFood

//

// Created by Егор on 21.04.2021.

//

**import** UIKit

**@IBDesignable**

/// Поле ввода с синими полями и радиусами

**class** InputTextField: UITextField {

/// Левый отступ

**@IBInspectable** **var** leftPadding: CGFloat = 0

/// Дополнительный отступ от картинки

**@IBInspectable** **var** gapPadding: CGFloat = 0

/// Цвет placeholder

**@IBInspectable** **var** color: UIColor? = .lightGray {

**didSet** {

updateView()

}

}

/// Картинка слева

**@IBInspectable** **var** leftImage: UIImage? {

**didSet** {

updateView()

}

}

/// Расчет расстояния слева для текста

**private** **var** textPadding: UIEdgeInsets {

**let** p: CGFloat = leftPadding + gapPadding + (leftView?.frame.width ?? 0)

**return** UIEdgeInsets(top: 15, left: p, bottom: 15, right: 5)

}

/// Область начала для текста

/// - **Parameter** bounds: Область вью

/// - **Returns**: Область для текста

**override** **func** leftViewRect(forBounds bounds: CGRect) -> CGRect {

**var** textRect = **super**.leftViewRect(forBounds: bounds)

textRect.origin.x += leftPadding

**return** textRect

}

/// Конструктор класса

**init**() {

**super**.init(frame: .zero)

updateView()

}

/// Конструктор класса

/// - **Parameter** coder: coder

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

/// Размеры текста

/// - **Parameter** bounds: Общая область

/// - **Returns**: Область текста

**override** **open** **func** textRect(forBounds bounds: CGRect) -> CGRect {

**return** bounds.inset(by: textPadding)

}

/// Размеры placeholder

/// - **Parameter** bounds: Общая область

/// - **Returns**: Область placeholder

**override** **open** **func** placeholderRect(forBounds bounds: CGRect) -> CGRect {

**return** bounds.inset(by: textPadding)

}

/// Размеры редактируемого текста

/// - **Parameter** bounds: Общая область

/// - **Returns**: Область редактируемого текста

**override** **open** **func** editingRect(forBounds bounds: CGRect) -> CGRect {

**return** bounds.inset(by: textPadding)

}

/// Конфигурировать вью по данным

**private** **func** updateView() {

**self**.layer.cornerRadius = 10

**self**.layer.borderColor = UIColor(named: "dark\_blue")?.cgColor

**self**.layer.borderWidth = 1

**guard** **let** image = leftImage **else** {

leftViewMode = UITextField.ViewMode.never

leftView = **nil**

**return**;

}

leftViewMode = UITextField.ViewMode.always

**let** imageView = UIImageView(frame: CGRect(x: 20, y: 0, width: 40, height: 20))

imageView.contentMode = .scaleAspectFit

imageView.image = image

imageView.tintColor = color

leftView = imageView

**guard** **let** color = color **else** {

**return**;

}

attributedPlaceholder = NSAttributedString(string: placeholder != **nil** ? placeholder! : "", attributes:[NSAttributedString.Key.foregroundColor: color])

}

}

* 1. TitledTextField.swift

//

// TitledTextField.swift

// GeoFood

//

// Created by Егор on 21.04.2021.

//

**import** UIKit

**@IBDesignable**

**class** TitledTextField: UIView {

/// Заголовок текстового поля

**private** **let** titleLabel = UILabel(frame: .zero)

/// Поле ввода

**private** **let** inputTextField = InputTextField()

/// Цвет заголовка и поля ввода

**@IBInspectable** **var** color: UIColor? = .gray

/// Текст заголовка

**@IBInspectable** **var** titleText: String? {

**didSet** {

updateView()

}

}

/// Введенный текст

**var** text: String {

inputTextField.text ?? ""

}

/// Является ли текстовое поле защищенным

**@IBInspectable** **var** isSecure: Bool = **false** {

**didSet** {

inputTextField.isSecureTextEntry = isSecure

}

}

/// Картинка поля ввода

**@IBInspectable** **var** textFieldImage: UIImage? {

**didSet** {

updateView()

}

}

/// Placeholder поля ввода

**@IBInspectable** **var** placeholder: String? {

**didSet** {

updateView()

}

}

/// Конструктор

**init**() {

**super**.init(frame: .zero)

addSubview(titleLabel)

addSubview(inputTextField)

titleLabel.translatesAutoresizingMaskIntoConstraints = **false**

inputTextField.translatesAutoresizingMaskIntoConstraints = **false**

inputTextField.font = UIFont.systemFont(ofSize: 16, weight: .medium)

updateView()

}

/// Конструктор

/// - **Parameter** coder: кодер

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

/// Инициализация констреинтов

**override** **func** layoutSubviews() {

NSLayoutConstraint.activate([

titleLabel.topAnchor.constraint(equalTo: topAnchor),

titleLabel.leadingAnchor.constraint(equalTo: leadingAnchor),

titleLabel.trailingAnchor.constraint(equalTo: trailingAnchor),

inputTextField.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 4),

inputTextField.leadingAnchor.constraint(equalTo: leadingAnchor),

inputTextField.trailingAnchor.constraint(equalTo: trailingAnchor),

inputTextField.bottomAnchor.constraint(equalTo: bottomAnchor)

])

}

/// Фабрика поля ввода для почты

/// - **Returns**: TitledTextField для почты

**static** **func** emailTextField() -> TitledTextField {

**let** emailTextField = TitledTextField()

emailTextField.placeholder = "example@example.com"

emailTextField.textFieldImage = UIImage(named: "mail")

emailTextField.color = UIColor(named: "dark\_blue")

emailTextField.titleText = "Email"

emailTextField.inputTextField.keyboardType = .emailAddress

**return** emailTextField

}

/// Фабрика поля ввода для пароля

/// - **Returns**: TitledTextField для пароля

**static** **func** passwordTextField() -> TitledTextField {

**let** passwordTextField = TitledTextField()

passwordTextField.titleText = "Пароль"

passwordTextField.textFieldImage = UIImage(named: "lock")

passwordTextField.color = UIColor(named: "dark\_blue")

passwordTextField.isSecure = **true**

passwordTextField.placeholder = String(repeating: "\u{2022}", count: 8)

**return** passwordTextField

}

/// Конфигуратор вью

**private** **func** updateView() {

**self**.isUserInteractionEnabled = **true**

inputTextField.color = color

inputTextField.gapPadding = 15

inputTextField.leftPadding = 15

titleLabel.text = titleText

titleLabel.textColor = color

inputTextField.placeholder = placeholder

inputTextField.leftImage = textFieldImage

inputTextField.attributedPlaceholder = NSAttributedString(string: placeholder ?? "", attributes: [NSAttributedString.Key.foregroundColor: UIColor.lightGray])

}

}

* 1. AppDelegate.swift

//

// AppDelegate.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** UIKit

**@main**

**class** AppDelegate: UIResponder, UIApplicationDelegate {

**func** application(**\_** application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: **Any**]?) -> Bool {

// Override point for customization after application launch.

**return** **true**

}

// MARK: **UISceneSession Lifecycle**

**func** application(**\_** application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {

// Called when a new scene session is being created.

// Use this method to select a configuration to create the new scene with.

**return** UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)

}

**func** application(**\_** application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {

// Called when the user discards a scene session.

// If any sessions were discarded while the application was not running, this will be called shortly after application:didFinishLaunchingWithOptions.

// Use this method to release any resources that were specific to the discarded scenes, as they will not return.

}

}

* 1. SceneDelegate.swift

//

// SceneDelegate.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** UIKit

**class** SceneDelegate: UIResponder, UIWindowSceneDelegate {

**var** window: UIWindow?

**func** scene(**\_** scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {

// Use this method to optionally configure and attach the UIWindow `window` to the provided UIWindowScene `scene`.

// If using a storyboard, the `window` property will automatically be initialized and attached to the scene.

// This delegate does not imply the connecting scene or session are new (see `application:configurationForConnectingSceneSession` instead).

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

window = UIWindow(frame: UIScreen.main.bounds)

window?.windowScene = scene

**let** mainView = MainViewController()

//let navigationController = UINavigationController(rootViewController: MapViewController())

window?.rootViewController = mainView

window?.makeKeyAndVisible()

}

**func** sceneDidDisconnect(**\_** scene: UIScene) {

// Called as the scene is being released by the system.

// This occurs shortly after the scene enters the background, or when its session is discarded.

// Release any resources associated with this scene that can be re-created the next time the scene connects.

// The scene may re-connect later, as its session was not necessarily discarded (see `application:didDiscardSceneSessions` instead).

}

**func** sceneDidBecomeActive(**\_** scene: UIScene) {

// Called when the scene has moved from an inactive state to an active state.

// Use this method to restart any tasks that were paused (or not yet started) when the scene was inactive.

}

**func** sceneWillResignActive(**\_** scene: UIScene) {

// Called when the scene will move from an active state to an inactive state.

// This may occur due to temporary interruptions (ex. an incoming phone call).

}

**func** sceneWillEnterForeground(**\_** scene: UIScene) {

// Called as the scene transitions from the background to the foreground.

// Use this method to undo the changes made on entering the background.

}

**func** sceneDidEnterBackground(**\_** scene: UIScene) {

// Called as the scene transitions from the foreground to the background.

// Use this method to save data, release shared resources, and store enough scene-specific state information

// to restore the scene back to its current state.

}

}

* 1. RestaurantSaleModel.swift

//

// RestaurantStockModel.swift

// GeoFood

//

// Created by Егор on 25.03.2021.

//

**import** Foundation

**import** UIKit

/// Модель акции кафе

**class** RestaurantSaleModel: Codable {

/// id акции

**var** id: Int32

/// Название акции

**var** name: String

/// Промокод акции

**var** promo: String?

/// Картинка акции

**var** image: UIImage?

/// Изначальная цена товара

**var** oldPrice: Int32?

/// Новая цена по акции

**var** newPrice: Int32

/// Является ли акция специальной

**var** special: Bool

/// Ключи для парсинга

**enum** CodingKeys: CodingKey {

**case** id

**case** name

**case** promo

**case** oldPrice

**case** newPrice

**case** special

}

/// Конструктор класса

**init**() {

id = 0

name = "Test"

promo = "Test"

oldPrice = 200

newPrice = 300

special = **false**

}

/// Конструктор из модели акции для CoreData

/// - **Parameter** model: Модель акции для CoreData

**init**(from model: Sale) {

**self**.id = model.id

**self**.name = model.name

**self**.promo = model.code

**self**.newPrice = model.price

**self**.special = model.isSpecial

**if** **let** data = model.image {

**self**.image = UIImage(data: data)

} **else** {

**self**.image = UIImage(named: "empty")

}

}

}

* 1. RestaurantModel.swift

//

// RestaurantModel.swift

// GeoFood

//

// Created by Егор on 25.03.2021.

//

**import** Foundation

**import** UIKit

/// Модель кафе

**class** RestaurantModel: Codable {

/// id кафе

**var** id: Int32

/// Название кафе

**var** name: String

/// Долгота позиции кафе

**var** longitude: Double

/// Широта позиции кафе

**var** latitude: Double

/// Картинка кафе

**var** logoImage: UIImage?

/// Тип кафе

**var** type: RestaurantType

/// Адрес кафе

**var** location: String

/// Акции ресторана

**var** sales: [RestaurantSaleModel]?

/// Конструктор из модели кафе для CoreData

/// - **Parameter** model: Модель кафе для CoreData

**init**(from model: Restaurant) {

id = model.id

name = model.name

longitude = model.longitude

latitude = model.latitude

type = model.type

location = model.address

sales = model.sales?.compactMap{ $0 **as**? RestaurantSaleModel }

}

/// Конструктор без параметров

**init**() {

id = 0

name = "Test"

longitude = 37.615

latitude = 55.75222

type = .coffee

location = ""

}

/// Ключи для десериализации

**enum** CodingKeys: CodingKey {

**case** id

**case** name

**case** longitude

**case** latitude

**case** location

**case** type

}

}

/// Перечисление типов ресторана

**@objc** **public** **enum** RestaurantType: Int32, Codable {

**case** coffee = 0

**case** fastFood = 1

/// Картинка для типа

**var** image: UIImage {

**switch** **self** {

**case** .coffee:

**return** UIImage(named: "coffee")!

**case** .fastFood:

**return** UIImage(named: "fast-food")!

}

}

/// Текст названия типа

**var** text: String {

**switch** **self** {

**case** .coffee:

**return** "Кофе"

**case** .fastFood:

**return** "ФастФуд"

}

}

/// Количество типов

**static** **let** typesCount = 2

}

* 1. UserModel.swift

//

// UserModel.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Модель пользователя

**class** UserModel {

/// Токен авторизации пользователя

**let** token: String

/// Почта пользователя

**let** login: String

/// Пароль польователя

**let** password: String

/// Инициализация из модели пользователя из CoreData

/// - **Parameter** model: модель пользователя из CoreData

**init**(from model: User) {

token = model.token ?? ""

login = model.login

password = model.password

}

}

* 1. User+CoreDataClass.swift

//

// User+CoreDataClass.swift

//

//

// Created by Егор on 25.04.2021.

//

//

**import** Foundation

**import** CoreData

**@objc(User)**

/// Модель пользователя

**public** **class** User: NSManagedObject {

}

* 1. User+CoreDataProperties.swift

//

// User+CoreDataProperties.swift

//

//

// Created by Егор on 26.04.2021.

//

//

**import** Foundation

**import** CoreData

**extension** User {

/// Запрос для получения объектов

/// - **Returns**: Запрос

**@nonobjc** **public** **class** **func** fetch() -> NSFetchRequest<User> {

**return** NSFetchRequest<User>(entityName: "User")

}

/// Токен

**@NSManaged** **public** **var** token: String?

/// Почта

**@NSManaged** **public** **var** login: String

/// Пароль

**@NSManaged** **public** **var** password: String

/// Кафе

**@NSManaged** **public** **var** restaurants: NSSet?

/// Неотправленные позиции

**@NSManaged** **public** **var** untrackedLocations: NSSet?

}

// MARK: **Generated accessors for restaurants**

**extension** User {

/// Добавить кафе пользователю

/// - **Parameter** value: Кафе для добавления

**@objc(addRestaurantsObject:)**

**@NSManaged** **public** **func** addToRestaurants(**\_** value: Restaurant)

/// Удалить кафе из пользователя

/// - **Parameter** value: Удаляемое кафе

**@objc(removeRestaurantsObject:)**

**@NSManaged** **public** **func** removeFromRestaurants(**\_** value: Restaurant)

/// Добавление множества кафе

/// - **Parameter** values: Множество кафе для добавления

**@objc(addRestaurants:)**

**@NSManaged** **public** **func** addToRestaurants(**\_** values: NSSet)

/// Удаление множества кафе из пользователя

/// - **Parameter** values: Множество кафе для удаления

**@objc(removeRestaurants:)**

**@NSManaged** **public** **func** removeFromRestaurants(**\_** values: NSSet)

}

// MARK: **Generated accessors for untrackedLocations**

**extension** User {

/// Добавить неотправленную позицию

/// - **Parameter** value: Неотправленная позиция для добавления

**@objc(addUntrackedLocationsObject:)**

**@NSManaged** **public** **func** addToUntrackedLocations(**\_** value: Location)

/// Удалить неотправленную позицию

/// - **Parameter** value: Неотправленная позиция для удаления

**@objc(removeUntrackedLocationsObject:)**

**@NSManaged** **public** **func** removeFromUntrackedLocations(**\_** value: Location)

/// Добавить неотправленные позиции

/// - **Parameter** value: Неотправленные позиции для добавления

**@objc(addUntrackedLocations:)**

**@NSManaged** **public** **func** addToUntrackedLocations(**\_** values: NSSet)

/// Удалить неотправленные позиции

/// - **Parameter** value: Неотправленные позиции для удаления

**@objc(removeUntrackedLocations:)**

**@NSManaged** **public** **func** removeFromUntrackedLocations(**\_** values: NSSet)

}

* 1. Restaurant+CoreDataClass.swift

//

// Restaurant+CoreDataClass.swift

//

//

// Created by Егор on 25.04.2021.

//

//

**import** Foundation

**import** CoreData

**@objc(Restaurant)**

/// Модель кафе

**public** **class** Restaurant: NSManagedObject {

}

* 1. Restaurant+CoreDataProperties.swift

//

// Restaurant+CoreDataProperties.swift

//

//

// Created by Егор on 26.04.2021.

//

//

**import** Foundation

**import** CoreData

**extension** Restaurant {

/// Запрос на загрузку объектов из CoreData

/// - **Returns**: Запрос

**@nonobjc** **public** **class** **func** fetch() -> NSFetchRequest<Restaurant> {

**return** NSFetchRequest<Restaurant>(entityName: "Restaurant")

}

/// id кафе

**@NSManaged** **public** **var** id: Int32

/// Адрес кафе

**@NSManaged** **public** **var** address: String

/// Широта кафе

**@NSManaged** **public** **var** latitude: Double

/// Долгота кафе

**@NSManaged** **public** **var** longitude: Double

/// Картинка кафе

**@NSManaged** **public** **var** logoImage: Data?

/// Название кафе

**@NSManaged** **public** **var** name: String

/// Акции кафе

**@NSManaged** **public** **var** sales: NSSet?

/// Тип кафе

**@NSManaged** **public** **var** type: RestaurantType

/// Конструктор кафе из модели кафе

/// - **Parameters**:

/// - plainModel: Модель кафе

/// - context: Контекст CoreData

**convenience** **init**(with plainModel: RestaurantModel, context: NSManagedObjectContext) {

**self**.init(context: context)

id = plainModel.id

address = plainModel.location

latitude = plainModel.latitude

longitude = plainModel.longitude

type = plainModel.type

logoImage = plainModel.logoImage?.pngData()

}

}

// MARK: **Generated accessors for sales**

**extension** Restaurant {

/// Добавить акцию к кафе

/// - **Parameter** value: Добавляемая акция

**@objc(addSalesObject:)**

**@NSManaged** **public** **func** addToSales(**\_** value: Sale)

/// Удалить акцию из кафе

/// - **Parameter** value: Удаляемая акция

**@objc(removeSalesObject:)**

**@NSManaged** **public** **func** removeFromSales(**\_** value: Sale)

/// Добавить множество акций к кафе

/// - **Parameter** values: Множество акций для добавления

**@objc(addSales:)**

**@NSManaged** **public** **func** addToSales(**\_** values: NSSet)

/// Удалить множество акций из кафе

/// - **Parameter** values: Множество акций для удаления

**@objc(removeSales:)**

**@NSManaged** **public** **func** removeFromSales(**\_** values: NSSet)

}

* 1. Sale+CoreDataClass.swift

//

// Sale+CoreDataClass.swift

//

//

// Created by Егор on 25.04.2021.

//

//

**import** Foundation

**import** CoreData

**@objc(Sale)**

/// Модель акции

**public** **class** Sale: NSManagedObject {

}

* 1. Sale+CoreDataProperties.swift

//

// Sale+CoreDataProperties.swift

//

//

// Created by Егор on 26.04.2021.

//

//

**import** Foundation

**import** CoreData

**extension** Sale {

/// Запрос на получение объектов из CoreData

/// - **Returns**: Запрос

**@nonobjc** **public** **class** **func** fetch() -> NSFetchRequest<Sale> {

**return** NSFetchRequest<Sale>(entityName: "Sale")

}

/// id акции

**@NSManaged** **public** **var** id: Int32

/// Промокод акции

**@NSManaged** **public** **var** code: String

/// Картинка акции

**@NSManaged** **public** **var** image: Data?

/// Название акции

**@NSManaged** **public** **var** name: String

/// Цена акции

**@NSManaged** **public** **var** price: Int32

/// Специальная ли акция

**@NSManaged** **public** **var** isSpecial: Bool

/// Кафе, которое предоставляет акцию

**@NSManaged** **public** **var** currentRestaurant: Restaurant?

/// Конструктор

/// - **Parameters**:

/// - plainModel: Модель акции

/// - context: Контекст CoreData

**convenience** **init**(with plainModel: RestaurantSaleModel, context: NSManagedObjectContext) {

**self**.init(context: context)

id = plainModel.id

code = plainModel.promo ?? "promo"

name = plainModel.name

price = plainModel.newPrice

isSpecial = plainModel.special

image = plainModel.image?.pngData()

}

}

* 1. Location+CoreDataClass.swift

//

// Location+CoreDataClass.swift

//

//

// Created by Егор on 26.04.2021.

//

//

**import** Foundation

**import** CoreData

**@objc(Location)**

/// Позиция пользователя

**public** **class** Location: NSManagedObject {

}

* 1. Location+CoreDataProperties.swift

//

// Location+CoreDataProperties.swift

//

//

// Created by Егор on 26.04.2021.

//

//

**import** Foundation

**import** CoreData

**extension** Location {

/// Запрос на получение объектов

/// - **Returns**: Запрос

**@nonobjc** **public** **class** **func** fetch() -> NSFetchRequest<Location> {

**return** NSFetchRequest<Location>(entityName: "Location")

}

/// Широта

**@NSManaged** **public** **var** latitude: Double

/// Долгота

**@NSManaged** **public** **var** longitude: Double

/// Дата когда пользователь был на позиции

**@NSManaged** **public** **var** date: Date

/// Пользователь

**@NSManaged** **public** **var** user: User?

}

* 1. CoreDataStore.swift

//

// CoreDataStore.swift

// GeoFood

//

// Created by Егор on 25.04.2021.

//

**import** Foundation

**import** CoreData

/// Сервис взаимодействия с CoreData

**class** CoreData {

/// Общий экземпляр типа

**static** **let** shared = CoreData()

/// Контейнер CoreData

**private** **var** persistentContainer: NSPersistentContainer = {

**let** container = NSPersistentContainer(name: "GeoFood")

container.loadPersistentStores(completionHandler: { (storeDescription, error) **in**

**if** **let** error = error **as** NSError? {

fatalError("Unresolved error \(error), \(error.userInfo)")

}

})

**return** container

}()

/// Контекст CoreData

**var** viewContext: NSManagedObjectContext {

persistentContainer.viewContext

}

/// Получить сохраненного пользователя

/// - **Returns**: Сохраненный пользователь или nil

**func** loadUser() -> User? {

**let** fetchRequest = User.fetch()

**return** **try**? viewContext.fetch(fetchRequest).first

}

/// Получить сохраненные кафе

/// - **Returns**: Массив кафе или nil

**func** loadRestaurants() -> [Restaurant]? {

**let** fetchRequest = Restaurant.fetch()

**return** **try**? viewContext.fetch(fetchRequest)

}

/// Сохранить изменения

**func** saveData() {

**if** viewContext.hasChanges {

**try**? viewContext.save()

}

}

/// Получить сохраненные позиции пользователя

/// - **Returns**: Позиции пользователя или nil

**func** loadUntrackedLocations() -> [Location]? {

**let** fetchRequest = Location.fetch()

**return** **try**? viewContext.fetch(fetchRequest)

}

/// Удалить объект из база

/// - **Parameter** object: Удаляемый объект

**func** removeObject(**\_** object: NSManagedObject) {

viewContext.delete(object)

}

/// Удалить текущего пользователя

**func** clearUser() {

**let** fetchRequest = User.fetch()

**guard** **let** users = **try**? viewContext.fetch(fetchRequest) **else** {

**return**

}

**for** user **in** users {

**self**.removeObject(user)

}

**self**.saveData()

}

}

* 1. CoordinateRequestModel.swift

//

// CoordinateRequestModel.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** Foundation

**import** MapKit

/// Модель данных для запроса кафе в окружности в центре геоточки и радиусе

**struct** CoordinateRequestModel: Codable {

/// Долгота точки

**let** longitude: Double

/// Широта точки

**let** latitude: Double

/// Радиус поиска

**let** radius: Double

/// Конструктор

/// - **Parameters**:

/// - coordinates: Координаты геоточки

/// - radius: Радиус поиска

**init**(coordinates: CLLocationCoordinate2D, radius: Double) {

longitude = coordinates.longitude

latitude = coordinates.latitude

**self**.radius = radius

}

}

* 1. ImageLoader.swift

//

// ImageLoader.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** Foundation

**import** Alamofire

/// Протокол загрузчика фотографий

**protocol** ImageLoaderProtocol: class {

/// Загрузить картинку для кафе

/// - **Parameters**:

/// - restId: id ресторана

/// - completion: Блок выполнения

**static** **func** loadRestaurantImage(restId: Int32, completion: **@escaping** (**\_** data: Data?) -> ())

/// Загрузить картинку для акции

/// - **Parameters**:

/// - saleId: id акции

/// - completion: Блоак выполнения

**static** **func** loadSaleImage(saleId: Int32, completion: **@escaping** (**\_** data: Data?) -> ())

}

/// Загрузчик фотографий из сети

**class** ImageLoader: ImageLoaderProtocol {

/// Загрузить картинку для кафе

/// - **Parameters**:

/// - restId: id ресторана

/// - completion: Блок выполнения

**static** **func** loadRestaurantImage(restId: Int32, completion: **@escaping** (**\_** data: Data?) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.loadRestaurantImage(for: restId)) { data, response, error **in**

completion(data)

}

dataTask.resume()

}

/// Загрузить картинку для акции

/// - **Parameters**:

/// - saleId: id акции

/// - completion: Блоак выполнения

**static** **func** loadSaleImage(saleId: Int32, completion: **@escaping** (**\_** data: Data?) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.loadSaleImage(for: saleId)) { data, response, error **in**

completion(data)

}

dataTask.resume()

}

}

* 1. Endpoints.swift

//

// Endpoints.swift

// GeoFood

//

// Created by Егор on 15.03.2021.

//

**import** Foundation

/// Маппинг url сервера

**enum** Endpoints {

/// Базовый url

**private** **static** **let** base = "http://178.20.41.6:8080/"

**case** register

**case** login

**case** getInfo

**case** getRestaurants

**case** getStocks

**case** loadShopImage

**case** loadSaleImage

**case** locationUpdate

/// Строковое значение url

**var** stringValue: String {

**switch** **self** {

**case** .register: **return** "\(Endpoints.base)reg/user"

**case** .login: **return** "\(Endpoints.base)auth"

**case** .getInfo: **return** "\(Endpoints.base)user/get"

**case** .getRestaurants: **return** "\(Endpoints.base)user/shops"

**case** .getStocks: **return** "\(Endpoints.base)user/stocks"

**case** .loadShopImage: **return** "\(Endpoints.base)shop/img"

**case** .loadSaleImage: **return** "\(Endpoints.base)stock/img"

**case** .locationUpdate: **return** "\(Endpoints.base)movement"

}

}

/// Получить url

**var** url: URL {

**return** URL(string: stringValue)!

}

}

* 1. UserService.swift

//

// UserService.swift

// GeoFood

//

// Created by Егор on 25.04.2021.

//

**import** Foundation

/// Сервис пользователя

**class** UserService {

/// Сервис CoreData

**private** **lazy** **var** coreDataStore = CoreData.shared

/// Общий объект сервиса

**static** **var** shared: UserService = UserService()

/// Конструктор

**private** **init**() {

currentUser = coreDataStore.loadUser()

**if** currentUser == **nil** {

currentUser = User(context: coreDataStore.viewContext)

}

}

/// Авторизован ли пользователь

**var** isUserAuth: Bool {

**return** currentUser?.token != **nil** && currentUser?.token != ""

}

/// Текущий пользователь

**var** currentUser: User? = **nil** {

**didSet** {

DispatchQueue.global(qos: .background).async { [**weak** **self**] **in**

**self**?.syncUserData()

}

}

}

/// Синхронизировать позиции пользователя с сервером

**private** **func** syncUserData() {

**if** (currentUser?.untrackedLocations?.count == 0) {

**return**;

}

**guard** **let** locations = coreDataStore.loadUntrackedLocations() **else** {

**return**

}

**for** location **in** locations {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.locationRequest(with: currentUser?.token ?? "", longitude: location.longitude, latitude: location.latitude, date: location.date)) { [**unowned** **self**] data, response, error **in**

**if** error != **nil** && (response **as**! HTTPURLResponse).statusCode == 500 {

**return**

}

**self**.coreDataStore.removeObject(location)

**self**.coreDataStore.saveData()

}

dataTask.resume()

}

}

/// Авторизировать пользователя

/// - **Parameters**:

/// - form: Форма авторизации

/// - completion: Блок выполнения

/// - **Returns**: Прошла ли авторизация пользователя успешно

**func** authUser(with form: LoginForm, completion: **@escaping** (**\_** isSuccess: Bool) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.authRequest(with: form)) { [**unowned** **self**] data, response, error **in**

**guard** **let** data = data,

**let** json = **try**? JSONSerialization.jsonObject(with: data, options: []),

**let** dict = (json **as**? [String: **Any**]),

**let** token = dict["token"] **as**? String

**else** {

completion(**false**)

**return**

}

coreDataStore.clearUser()

**let** user = User(context: coreDataStore.viewContext)

user.token = token

user.login = form.login

user.password = form.password

user.untrackedLocations = **self**.currentUser?.untrackedLocations

**self**.currentUser = user

**self**.coreDataStore.saveData()

completion(**true**)

}

dataTask.resume()

}

/// Регистрация пользователя

/// - **Parameters**:

/// - form: Форма регистрации

/// - completion: Блок выполнения

/// - **Returns**: прошла ли регистрация успешно

**func** registerUser(with form: LoginForm, completion: **@escaping** (**\_** isSuccess: Bool) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.registrationRequest(with: form)) { data, response, error **in**

completion(error == **nil** && (response **as**! HTTPURLResponse).statusCode != 500)

}

dataTask.resume()

}

/// Получить кафе рядом с положением пользователя

/// - **Parameters**:

/// - coordinate: Координаты пользователя и радиус

/// - completion: Блок выполнения

/// - **Returns**: Загруженные модели ресторанов

**func** getRestaurantsNear(coordinate: CoordinateRequestModel, completion: **@escaping** ([RestaurantModel]?) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.restaurantsRequest(with: **self**.currentUser?.token ?? "", data: coordinate)) { data, response, error **in**

**if** error != **nil** {

completion(**nil**)

**return**

}

**guard** **let** data = data,

**let** json = **try**? JSONDecoder().decode([RestaurantModel].**self**, from: data)

**else** {

completion(**nil**)

**return**

}

completion(json)

}

dataTask.resume()

}

/// Получить акции кафе

/// - **Parameters**:

/// - restaurantId: id кафе

/// - completion: Блок выполнения

/// - **Returns**: Загруженные модели акций

**func** getRestaurantSales(restaurantId: Int32, completion: **@escaping** (**\_** sales: [RestaurantSaleModel]?) -> ()) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.restaurantSalesRequest(with: currentUser?.token ?? "", restaurantId: restaurantId)) { data, response, error **in**

**guard** **let** data = data,

**let** json = **try**? JSONDecoder().decode([RestaurantSaleModel].**self**, from: data)

**else** {

**let** rests = **self**.coreDataStore.loadRestaurants()?.map{ RestaurantModel(from: $0) }

**let** sales = rests?.first(where: { $0.id == restaurantId })?.sales

completion(sales)

**return**

}

completion(json)

}

dataTask.resume()

}

/// Отправить обновление позиции пользователя

/// - **Parameters**:

/// - latitude: Долгота позиции

/// - longitude: Широта позиции

**func** updateUserLocation(latitude: Double, longitude: Double) {

**let** dataTask = URLSession.shared.dataTask(with: RequestFactory.locationRequest(with: currentUser?.token ?? "", longitude: longitude, latitude: latitude, date: Date())) { [**unowned** **self**] data, response, error **in**

**if** error != **nil** {

**let** untrackedLocation = Location(context: **self**.coreDataStore.viewContext)

untrackedLocation.latitude = latitude

untrackedLocation.longitude = longitude

untrackedLocation.date = Date()

currentUser?.addToUntrackedLocations(untrackedLocation)

**self**.coreDataStore.saveData()

**return**

}

DispatchQueue.global(qos: .background).async {

**self**.syncUserData()

}

}

dataTask.resume()

}

/// Выйти из аккаунта пользователя

/// - **Parameter** completion: Блок выполнения

/// - **Returns**: Прошел ли выход успешно

**func** logout(completion: (**\_** isSuccess: Bool) -> ()) {

coreDataStore.clearUser()

coreDataStore.saveData()

currentUser = **nil**

completion(**true**)

}

}

* 1. RequestFactory.swift

//

// RequestFactory.swift

// GeoFood

//

// Created by Егор on 25.04.2021.

//

**import** Foundation

/// Фабрика запросов

**final** **class** RequestFactory {

/// Запрос авторизации пользователя

/// - **Parameter** form: Форма авторизации

/// - **Returns**: Сформированный запрос

**static** **func** authRequest(with form: LoginForm) -> URLRequest {

**let** url = Endpoints.login.url

**var** request = URLRequest(url: url)

**let** data = ["login": form.login, "password": form.password]

request.httpMethod = "POST"

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.httpBody = **try**! JSONSerialization.data(withJSONObject: data, options: .prettyPrinted)

**return** request

}

/// Запрос регистрации пользователя

/// - **Parameter** form: Форма регистрации

/// - **Returns**: Сформированный запрос

**static** **func** registrationRequest(with form: LoginForm) -> URLRequest {

**let** url = Endpoints.register.url

**let** data = ["login" : form.login, "password" : form.password]

**var** request = URLRequest(url: url)

request.httpMethod = "POST"

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.httpBody = **try**! JSONSerialization.data(withJSONObject: data, options: .prettyPrinted)

**return** request

}

/// Запрос получения акций кафе

/// - **Parameters**:

/// - token: токен пользователя

/// - restaurantId: id кафе

/// - **Returns**: Сформированный запрос

**static** **func** restaurantSalesRequest(with token: String, restaurantId: Int32) -> URLRequest {

**var** request = URLRequest(url: Endpoints.getStocks.url)

request.httpMethod = "POST"

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

request.httpBody = **try**! JSONSerialization.data(withJSONObject: ["id" : restaurantId], options: .prettyPrinted)

**return** request

}

/// Запрос кафе рядом с позицией

/// - **Parameters**:

/// - token: токен пользователя

/// - coordinate: Координаты для запроса

/// - **Returns**: Сформированный запрос

**static** **func** restaurantsRequest(with token: String, data coordinate: CoordinateRequestModel) -> URLRequest {

**var** request = URLRequest(url: Endpoints.getRestaurants.url)

request.httpMethod = "POST"

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

**let** opts = **try**! JSONEncoder().encode(coordinate)

request.httpBody = opts

**return** request

}

/// Запрос обновления позиции пользователя

/// - **Parameters**:

/// - token: Токен пользователя

/// - longitude: Долгота позиции

/// - latitude: Широта позиции

/// - date: Дата отправки запроса

/// - **Returns**: Сформированный запрос

**static** **func** locationRequest(with token: String, longitude: Double, latitude: Double, date: Date) -> URLRequest {

**var** request = URLRequest(url: Endpoints.locationUpdate.url)

request.httpMethod = "POST"

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.addValue("Bearer \(token)", forHTTPHeaderField: "Authorization")

**let** dateFormat = DateFormatter()

dateFormat.dateFormat = "yyyy-MM-dd HH:mm:ss"

**let** data = **try**! JSONSerialization.data(withJSONObject: ["longitude": longitude, "latitude": latitude, "date": dateFormat.string(from: date)])

request.httpBody = **try**! JSONSerialization.data(withJSONObject: ["longitude": longitude, "latitude": latitude, "date": dateFormat.string(from: date)], options: .prettyPrinted)

**return** request

}

/// Запрос загрузки картинки кафе

/// - **Parameter** restaurantId: id кафе

/// - **Returns**: Сформированный запрос

**static** **func** loadRestaurantImage(for restaurantId: Int32) -> URLRequest {

**var** request = URLRequest(url: Endpoints.loadShopImage.url)

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.httpMethod = "POST"

request.httpBody = **try**! JSONSerialization.data(withJSONObject: ["shopId": restaurantId], options: .prettyPrinted)

**return** request

}

/// Запрос загрузки картинки акции

/// - **Parameter** saleId: id акции

/// - **Returns**: Сформированный запрос

**static** **func** loadSaleImage(for saleId: Int32) -> URLRequest {

**var** request = URLRequest(url: Endpoints.loadSaleImage.url)

request.addValue("application/json", forHTTPHeaderField: "Content-Type")

request.httpMethod = "POST"

request.httpBody = **try**! JSONSerialization.data(withJSONObject: ["stockId": saleId], options: .prettyPrinted)

**return** request

}

}

* 1. LocationManager.swift

//

// LocationManager.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

**import** MapKit

/// Сервис позиции пользователя

**final** **class** LocationManager: CLLocationManager {

/// Общий объект сервиса

**static** **let** shared = LocationManager()

/// Расстояние от позиции пользователя до геоточки

/// - **Parameter** location: Геоточка

/// - **Returns**: Расстояние

**func** userDistance(to location: CLLocation) -> Double {

**return** **self**.location?.distance(from: location) ?? 0

}

}

* 1. UserViewModel.swift

//

// UserViewModel.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Модель представления пользователя

**class** UserViewModel {

/// Логин пользователя

**let** login: String

/// Конструктор из модели пользователя

/// - **Parameter** model: Модель пользователя

**init**(from model: UserModel) {

**self**.login = model.login

}

}

* 1. AccountPresenter.swift

//

// AccountPresenter.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Выходные методы презентера

**protocol** AccountPresenterOutput: AnyObject {

/// Метод выхода пользователя из аккаунта

**func** logoutUser()

}

/// Входные методы презентера

**protocol** AccountPresenterInput: AnyObject {

/// Метод успешного выхода из аккаунта

**func** logoutSuccess()

}

/// Презентер аккаунта

**class** AccountPresenter: AccountPresenterInput {

/// Контроллер аккаунта

**weak** **var** view: AccountViewInput!

/// Интерактор аккаунта

**var** interactor: AccountInteractor!

/// Роутер аккаунта

**var** router: AccountRouterInput!

/// Конструктор с контроллером

/// - **Parameter** view: Контроллер

**init**(with view: AccountViewInput) {

**self**.view = view

}

/// Метод успешного выхода из аккаунта

**func** logoutSuccess() {

router.popBack()

}

}

**extension** AccountPresenter: AccountViewOutput {

/// Событие нажатия кнопки выхода

**func** logoutTapped() {

interactor.logoutUser()

}

/// Контроллер загрузился

**func** viewDidLoad() {

**guard** **let** user = interactor.currentUser **else** {

router.popBack()

**return**

}

view.configure(with: UserViewModel(from: user))

}

}

* 1. AccountViewController.swift

//

// AccountViewController.swift

// GeoFood

//

// Created by Егор on 04.05.2021.

//

**import** UIKit

/// Выходные методы контроллера

**protocol** AccountViewOutput: AnyObject {

/// Событие нажатия кнопки выхода

**func** logoutTapped()

/// Контроллер загружен

**func** viewDidLoad()

}

/// Входные методы контроллера

**protocol** AccountViewInput: AnyObject {

/// Сконфигурировать контроллер по модели данных пользователя

/// - **Parameter** viewModel: Модель данных пользователя

**func** configure(with viewModel: UserViewModel)

}

**class** AccountViewController: UIViewController, AccountViewInput {

/// Кнопка выхода из аккаунта

**var** logoutButton = UIButton()

/// Заголовок поля почты

**var** emailTitle = UILabel()

/// Текст почты пользователя

**var** userEmail = UILabel()

/// Presenter модуля

**var** output: AccountViewOutput!

/// Контроллер загружен

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Аккаунт"

**let** background = UIImage(named: "background")

**let** imageView = UIImageView(frame: view.bounds)

imageView.contentMode = .scaleAspectFill

imageView.clipsToBounds = **true**

imageView.image = background

imageView.center = view.center

view.addSubview(imageView)

**self**.view.sendSubviewToBack(imageView)

navigationItem.setHidesBackButton(**true**, animated: **false**)

logoutButton.translatesAutoresizingMaskIntoConstraints = **false**

logoutButton.setTitle("Выйти", for: .normal)

logoutButton.setTitleColor(UIColor(named: "dark\_blue"), for: .normal)

logoutButton.layer.borderColor = UIColor(named: "light\_green")?.cgColor

logoutButton.layer.borderWidth = 2

logoutButton.layer.cornerRadius = 10

logoutButton.backgroundColor = UIColor(named: "lime")

logoutButton.titleLabel?.font = UIFont.systemFont(ofSize: 16, weight: .semibold)

logoutButton.addTarget(**self**, action: **#selector**(logoutButtonTapped), for: .touchUpInside)

view.addSubview(logoutButton)

emailTitle.translatesAutoresizingMaskIntoConstraints = **false**

emailTitle.attributedText = NSAttributedString(string: "Ваша почта:", attributes: [NSAttributedString.Key.underlineStyle: NSUnderlineStyle.thick.rawValue])

emailTitle.font = UIFont.systemFont(ofSize: 24, weight: .semibold)

emailTitle.textColor = UIColor(named: "dark\_blue")

view.addSubview(emailTitle)

userEmail.translatesAutoresizingMaskIntoConstraints = **false**

userEmail.font = UIFont.systemFont(ofSize: 20, weight: .semibold)

userEmail.textColor = UIColor(named: "light\_blue")

view.addSubview(userEmail)

NSLayoutConstraint.activate([

logoutButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),

logoutButton.centerYAnchor.constraint(equalTo: view.centerYAnchor),

logoutButton.widthAnchor.constraint(equalToConstant: 200),

logoutButton.heightAnchor.constraint(equalToConstant: 50),

emailTitle.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 10),

emailTitle.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 20),

userEmail.topAnchor.constraint(equalTo: emailTitle.bottomAnchor, constant: 10),

userEmail.leadingAnchor.constraint(equalTo: emailTitle.leadingAnchor)

])

output.viewDidLoad()

}

/// Контроллер появился на экране

/// - **Parameter** animated: анимировано ли появился

**override** **func** viewDidAppear(**\_** animated: Bool) {

**super**.viewDidAppear(animated)

**self**.navigationController?.tabBarItem.title = **self**.title

}

/// Событие нажатия на кнопку выхода из аккаунта

**@objc** **private** **func** logoutButtonTapped() {

output.logoutTapped()

}

/// Сконфигурировать контроллер по модели данных пользователя

/// - **Parameter** viewModel: Модель данных пользователя

**func** configure(with viewModel: UserViewModel) {

**self**.userEmail.text = viewModel.login

}

}

* 1. AccountInteractor.swift

//

// AccountInteractor.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Интерактор аккаунта

**class** AccountInteractor {

/// Сервис пользователя

**let** userService = UserService.shared

/// Презентер аккаунта

**weak** **var** presenter: AccountPresenterInput!

/// Текущий пользователя

**var** currentUser: UserModel? {

**if** **let** user = userService.currentUser {

**return** UserModel(from: user)

}

**return** **nil**

}

/// Конструктор

/// - **Parameter** presenter: Презентер аккаунта

**init**(with presenter: AccountPresenterInput) {

**self**.presenter = presenter

}

}

**extension** AccountInteractor: AccountPresenterOutput {

/// Выйти из аккаунта пользователя

**func** logoutUser() {

userService.logout { isSuccess **in**

presenter.logoutSuccess()

}

}

}

* 1. AccountRouter.swift

//

// AccountRouter.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

**import** Foundation

/// Входные методы роутера

**protocol** AccountRouterInput: AnyObject {

/// Вернуться в корневой вью

**func** popBack()

}

**class** AccountRouter: AccountRouterInput {

/// Контроллер аккаунта

**private** **var** view: AccountViewController!

/// Конструктор с контроллером

/// - **Parameter** view: Контроллер аккаунта

**init**(with view: AccountViewController) {

**self**.view = view

}

/// Вернуться в корневой вью

**func** popBack() {

view.navigationController?.popViewController(animated: **true**)

}

}

* 1. AccountConfigurator.swift

//

// AccountConfigurator.swift

// GeoFood

//

// Created by Егор on 05.05.2021.

//

* 1. AccountConfigurator.swift

**import** Foundation

/// Конфигуратор модуля аккаунта

**class** AccountConfigurator {

/// Создает зависимости между компонентами модуля

/// - **Returns**: Контроллер модуля

**static** **func** assembly() -> AccountViewController {

**let** view = AccountViewController()

**let** presenter = AccountPresenter(with: view)

view.output = presenter

**let** interactor = AccountInteractor(with: presenter)

presenter.interactor = interactor

presenter.router = AccountRouter(with: view)

**return** view

}

}

* 1. MainViewController.swift

//

// MainViewController.swift

// GeoFood

//

// Created by Егор on 06.04.2021.

//

**import** UIKit

/// TabBar контроллер для карты и авторизации

**class** MainViewController: UITabBarController {

/// Вью загрузилось

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

**let** authorizationNavController = UINavigationController(rootViewController: AuthorizationViewController())

**let** tabBarItem = UITabBarItem(title: "Аккаунт", image: UIImage(systemName: "person.crop.circle"), tag: 1)

authorizationNavController.tabBarItem = tabBarItem

authorizationNavController.navigationBar.layoutMargins.left = 32

authorizationNavController.navigationBar.largeTitleTextAttributes = [.foregroundColor: UIColor(named: "dark\_blue")!]

//authorizationNavController.tabBarItem.badgeColor = UIColor(named: "dark\_blue")

**let** mapNavController = UINavigationController(rootViewController: MapViewController())

mapNavController.navigationBar.layoutMargins.left = 32

mapNavController.navigationBar.largeTitleTextAttributes = [.foregroundColor: UIColor(named: "dark\_blue")!]

mapNavController.tabBarItem = UITabBarItem(title: "Карта", image: UIImage(systemName: "map"), tag: 0)

viewControllers = [mapNavController, authorizationNavController]

tabBar.barTintColor = UIColor(named: "light\_green")

tabBar.tintColor = UIColor(named: "dark\_blue")

tabBar.unselectedItemTintColor = UIColor(named: "light\_blue")

}

/// Перерисовка tabbar

**override** **func** viewDidLayoutSubviews() {

**super**.viewDidLayoutSubviews()

tabBar.backgroundColor = UIColor(named: "light\_green")

view.layer.cornerRadius = 10

tabBar.layer.cornerRadius = 30

tabBar.layer.masksToBounds = **true**

tabBar.frame = CGRect(x: 30, y: tabBar.frame.minY, width: view.frame.width - 60, height: 60)

tabBar.removeConstraints(tabBar.constraints)

}

}

* 1. RestaurantViewController.swift

//

// RestaurantViewController.swift

// GeoFood

//

// Created by Егор on 17.03.2021.

//

**import** UIKit

/// Выходные методы контроллера

**protocol** RestaurantViewControllerOutput: AnyObject {

**func** viewDidLoad()

}

/// Контроллер кафе

**class** RestaurantViewController: UIViewController {

/// Взаимодействует ли пользователь с скроллом

**var** isDragging = **false**

/// Скроллится ли скролл

**var** isDecelerating = **false**

/// Последняя позиция контента скролла

**var** lastSalesTableScrollY: CGFloat = 0

/// Презентер кафе

**var** presenter: RestaurantViewControllerOutput!

/// Верхняя вью с информацией о кафе

**private** **let** header = RestaurantHeaderView()

/// Таблица общих акций

**private** **let** salesTable = UITableView()

/// Коллекция специальных акций

**private** **let** specialSalesCollection: UICollectionView = {

**let** layout = UICollectionViewFlowLayout()

layout.scrollDirection = .horizontal

**return** UICollectionView(frame: .zero, collectionViewLayout: layout)

}()

/// Точки под коллекцией с указанием текущей акции

**private** **let** pageControl = UIPageControl()

/// Общий скролл

**private** **let** scrollView = UIScrollView()

/// Контент скролла

**private** **let** contentView = UIView()

/// id ячейки коллекции

**private** **let** collectionCellIdentifier = "stockCell"

/// Модель данных ресторана

**private** **var** viewModel: RestaurantViewModel?

/// Верхний констреинт таблицы общих акций

**private** **var** salesTableTopAnchor: NSLayoutYAxisAnchor?

/// Перерисовать вью

**override** **func** viewDidLayoutSubviews() {

**super**.viewDidLayoutSubviews()

**self**.scrollView.contentSize = contentView.bounds.size

}

/// Контроллер загрузился

**override** **func** viewDidLoad() {

view.backgroundColor = .white

title = "Акции"

**self**.navigationController?.navigationBar.prefersLargeTitles = **true**

**self**.navigationController?.setNavigationBarHidden(**false**, animated: **true**)

**let** background = UIImage(named: "background")

**var** imageView : UIImageView!

imageView = UIImageView(frame: view.bounds)

imageView.contentMode = .scaleAspectFill

imageView.clipsToBounds = **true**

imageView.image = background

imageView.center = view.center

view.addSubview(imageView)

**self**.view.sendSubviewToBack(imageView)

configureSubviews()

addAllSubviews()

initConstraints()

presenter.viewDidLoad()

}

/// Конфигурация отображения внутренних вью

**func** configureSubviews() {

header.translatesAutoresizingMaskIntoConstraints = **false**

salesTable.translatesAutoresizingMaskIntoConstraints = **false**

salesTable.delegate = **self**

salesTable.dataSource = **self**

salesTable.register(SaleCell.**self**, forCellReuseIdentifier: collectionCellIdentifier)

salesTable.backgroundColor = .clear

salesTable.separatorStyle = .none

salesTable.allowsSelection = **false**

salesTable.isScrollEnabled = **false**

salesTable.showsVerticalScrollIndicator = **false**

salesTableTopAnchor = salesTable.topAnchor

specialSalesCollection.backgroundColor = .clear

specialSalesCollection.translatesAutoresizingMaskIntoConstraints = **false**

specialSalesCollection.dataSource = **self**

specialSalesCollection.register(SpecialSaleCollectionViewCell.**self**, forCellWithReuseIdentifier: "collectionCell")

specialSalesCollection.showsHorizontalScrollIndicator = **false**

specialSalesCollection.isPagingEnabled = **true**

specialSalesCollection.layer.masksToBounds = **false**

**let** layout = specialSalesCollection.collectionViewLayout **as**! UICollectionViewFlowLayout

layout.itemSize = CGSize(width: **self**.view.frame.width - 60, height: 160)

layout.sectionInset = UIEdgeInsets(top: 0, left: 30, bottom: 0, right: 30)

layout.minimumLineSpacing = 60

pageControl.translatesAutoresizingMaskIntoConstraints = **false**

pageControl.pageIndicatorTintColor = .lightGray

pageControl.currentPageIndicatorTintColor = .darkGray

pageControl.isUserInteractionEnabled = **false**

scrollView.translatesAutoresizingMaskIntoConstraints = **false**

scrollView.delegate = **self**

contentView.translatesAutoresizingMaskIntoConstraints = **false**

contentView.layer.masksToBounds = **true**

}

/// Добавить все вью в контроллер

**func** addAllSubviews() {

view.addSubview(scrollView)

scrollView.addSubview(contentView)

contentView.addSubview(header)

contentView.addSubview(specialSalesCollection)

contentView.addSubview(salesTable)

contentView.addSubview(pageControl)

}

/// Активировать констреинты

**func** initConstraints() {

NSLayoutConstraint.activate([

scrollView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor),

scrollView.leadingAnchor.constraint(equalTo: view.leadingAnchor),

scrollView.trailingAnchor.constraint(equalTo: view.trailingAnchor),

scrollView.bottomAnchor.constraint(equalTo: view.bottomAnchor),

contentView.topAnchor.constraint(equalTo: scrollView.topAnchor),

contentView.widthAnchor.constraint(equalTo: scrollView.widthAnchor),

contentView.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor),

header.topAnchor.constraint(equalTo: contentView.topAnchor, constant: 12),

header.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 33),

header.trailingAnchor.constraint(equalTo: contentView.trailingAnchor, constant: -33),

header.heightAnchor.constraint(equalToConstant: 60),

specialSalesCollection.topAnchor.constraint(equalTo: header.bottomAnchor, constant: 10),

specialSalesCollection.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),

specialSalesCollection.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),

specialSalesCollection.heightAnchor.constraint(equalToConstant: 160),

pageControl.topAnchor.constraint(equalTo: specialSalesCollection.bottomAnchor, constant: 5),

pageControl.leadingAnchor.constraint(greaterThanOrEqualTo: contentView.leadingAnchor, constant: 10),

pageControl.trailingAnchor.constraint(lessThanOrEqualTo: contentView.trailingAnchor, constant: -10),

pageControl.centerXAnchor.constraint(equalTo: contentView.centerXAnchor),

pageControl.heightAnchor.constraint(equalToConstant: 20),

salesTable.topAnchor.constraint(equalTo: pageControl.bottomAnchor, constant: 10),

salesTable.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

salesTable.trailingAnchor.constraint(lessThanOrEqualTo: contentView.trailingAnchor, constant: -20),

salesTable.widthAnchor.constraint(equalTo: contentView.widthAnchor, constant: -40),

salesTable.heightAnchor.constraint(equalTo: scrollView.heightAnchor),

contentView.bottomAnchor.constraint(equalTo: salesTable.bottomAnchor)

])

scrollView.contentSize = contentView.bounds.size

}

}

**extension** RestaurantViewController: RestaurantPresenterOutput {

/// Сконфигурировать вью по модели данных кафе

/// - **Parameter** vm: Модель данных кафе

**func** configure(with vm: RestaurantViewModel) {

**self**.viewModel = vm

header.configure(with: vm)

specialSalesCollection.isHidden = vm.specialSales.count == 0

pageControl.isHidden = vm.specialSales.count == 0

**if** specialSalesCollection.isHidden {

salesTable.topAnchor.constraint(equalTo: header.bottomAnchor, constant: 10).isActive = **true**

} **else** {

salesTable.topAnchor.constraint(equalTo: pageControl.bottomAnchor, constant: 10).isActive = **true**

}

salesTable.reloadData()

specialSalesCollection.reloadData()

}

}

**extension** RestaurantViewController: UITableViewDataSource {

/// Создание ячейки для таблицы

/// - **Parameters**:

/// - tableView: Таблица, для которой создается ячейка

/// - indexPath: Номер ячейки

/// - **Returns**: Ячейка с данными

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

**guard** **let** cell = tableView.dequeueReusableCell(withIdentifier: collectionCellIdentifier) **as**? SaleCell,

**let** viewModel = **self**.viewModel **else** {

**return** UITableViewCell()

}

**let** sale = viewModel.sales[indexPath.row]

cell.configure(with: sale)

**return** cell

}

/// Количество ячеек

/// - **Parameters**:

/// - tableView: Таблица, для которой определяется количество ячеек

/// - section: Секция, в которой будут ячейки

/// - **Returns**: Число ячеек

**func** tableView(**\_** tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

viewModel?.sales.count ?? 0

}

/// Вычисление высоты для ячейки таблицы

/// - **Parameters**:

/// - tableView: Таблица, для которой определяется высота ячейки

/// - indexPath: Индекс ячейки

/// - **Returns**: Высота ячейки

**func** tableView(**\_** tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

**return** 90

}

}

**extension** RestaurantViewController: UITableViewDelegate {

/// Создание вью с заголовком для таблицы

/// - **Parameters**:

/// - tableView: Таблица

/// - section: Секция таблицы

/// - **Returns**: Вью с заголовком

**func** tableView(**\_** tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {

**let** view = UIView(frame: CGRect(x: 0, y: 0, width: tableView.frame.width, height: 40))

**let** label = UILabel(frame: CGRect(x: 3, y: 0, width: view.frame.width - 3, height: 30))

label.text = "Все акции:"

label.textColor = UIColor(named: "dark\_blue")

label.font = UIFont.systemFont(ofSize: 19, weight: .semibold)

view.addSubview(label)

view.backgroundColor = .clear

**return** view

}

/// Высота вью с заголовком

/// - **Parameters**:

/// - tableView: Таблица

/// - section: Секция таблицы

/// - **Returns**: Высота

**func** tableView(**\_** tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {

40

}

}

**extension** RestaurantViewController: UICollectionViewDataSource {

/// Вычисление количества ячеек в коллекции

/// - **Parameters**:

/// - collectionView: Коллекция

/// - section: Секция коллекции

/// - **Returns**: Количество ячеек

**func** collectionView(**\_** collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {

**self**.pageControl.numberOfPages = viewModel?.specialSales.count ?? 0

**return** viewModel?.specialSales.count ?? 0

}

/// Конфигурация ячейки для коллекции

/// - **Parameters**:

/// - collectionView: Коллекция

/// - indexPath: Индекс коллекции

/// - **Returns**: Ячейка коллекции

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

**guard** **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: "collectionCell", for: indexPath) **as**? SpecialSaleCollectionViewCell,

**let** viewModel = **self**.viewModel

**else** {

**return** UICollectionViewCell()

}

**let** sale = viewModel.specialSales[indexPath.row]

cell.configure(with: sale)

**return** cell

}

}

**extension** RestaurantViewController: UIScrollViewDelegate {

/// Индекс отображаемой ячейки коллекции

**func** getCurrentPage() {

**let** visibleRect = CGRect(origin: specialSalesCollection.contentOffset, size: specialSalesCollection.bounds.size)

**let** visiblePoint = CGPoint(x: visibleRect.midX, y: visibleRect.midY)

**if** **let** visibleIndexPath = specialSalesCollection.indexPathForItem(at: visiblePoint) {

**self**.pageControl.currentPage = visibleIndexPath.row

}

}

/// Событие завершения скролла

/// - **Parameters**:

/// - scrollView: Скролл

/// - decelerate: Будет ли пролистывание дальше

**func** scrollViewDidEndDragging(**\_** scrollView: UIScrollView, willDecelerate decelerate: Bool) {

**if** scrollView == **self**.scrollView && !isDecelerating {

**if** scrollView.contentOffset.y > 20 {

scrollView.setContentOffset(CGPoint(x: 0, y: salesTable.frame.minY), animated: **true**)

scrollView.isScrollEnabled = **false**

**self**.salesTable.isScrollEnabled = **true**

**self**.isModalInPresentation = **true**

isDragging = **true**

} **else** {

scrollView.setContentOffset(CGPoint(x: 0, y: 0), animated: **true**)

}

}

**if** scrollView == salesTable {

**if** scrollView.contentOffset.y <= 0 && lastSalesTableScrollY <= 0 {

**self**.scrollView.setContentOffset(CGPoint(x: 0, y: 0), animated: **true**)

**self**.scrollView.isScrollEnabled = **true**

scrollView.isScrollEnabled = **false**

**self**.isModalInPresentation = **false**

}

lastSalesTableScrollY = scrollView.contentOffset.y

}

}

/// Скролл завершил пролистывание

/// - **Parameter** scrollView: Скролл

**func** scrollViewDidEndDecelerating(**\_** scrollView: UIScrollView) {

**if** scrollView == **self**.scrollView && !isDragging {

**if** scrollView.contentOffset.y > 20 {

scrollView.setContentOffset(CGPoint(x: 0, y: salesTable.frame.minY), animated: **true**)

scrollView.isScrollEnabled = **false**

**self**.salesTable.isScrollEnabled = **true**

**self**.isModalInPresentation = **true**

isDecelerating = **true**

} **else** {

scrollView.setContentOffset(CGPoint(x: 0, y: 0), animated: **true**)

}

}

**if** scrollView == salesTable {

**if** scrollView.contentOffset.y <= 0 && lastSalesTableScrollY <= 0 {

**self**.scrollView.setContentOffset(CGPoint(x: 0, y: 0), animated: **true**)

**self**.scrollView.isScrollEnabled = **true**

scrollView.isScrollEnabled = **false**

**self**.isModalInPresentation = **false**

isDragging = **false**

isDecelerating = **false**

}

lastSalesTableScrollY = scrollView.contentOffset.y

}

**if** scrollView == specialSalesCollection {

getCurrentPage()

}

}

/// Скролл скролится пользователем

/// - **Parameter** scrollView: Скролл

**func** scrollViewDidScroll(**\_** scrollView: UIScrollView) {

**if** scrollView == specialSalesCollection {

getCurrentPage()

}

}

/// Скролл закончил касание пользователя

/// - **Parameters**:

/// - scrollView: Скролл

/// - velocity: Разница при скроле

/// - targetContentOffset: Текущая точка

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

**if** scrollView == specialSalesCollection {

getCurrentPage()

}

}

}

* 1. RestaurantHeaderView.swift

//

// RestaurantHeaderView.swift

// GeoFood

//

// Created by Егор on 23.04.2021.

//

**import** UIKit

/// Вью информации о кафе

**class** RestaurantHeaderView: UIView {

/// Картинка кафе

**private** **let** logoImage = UIImageView()

/// Картинка типа кафе

**private** **let** smallImage = UIImageView()

/// Название кафе

**private** **let** nameLabel = UILabel()

/// Адрес кафе

**private** **let** addressLabel = UILabel()

/// Расстояние пользователя до кафе

**private** **let** distanceLabel = UILabel()

/// Конструктор

**init**() {

**super**.init(frame: .zero)

setupUI()

}

/// Конструктор

/// - **Parameter** frame: Фрейм

**override** **init**(frame: CGRect) {

**super**.init(frame: frame)

setupUI()

}

/// Конструктор

/// - **Parameter** coder: Кодер

**required** **init**?(coder: NSCoder) {

**super**.init(coder: coder)

setupUI()

}

/// Конфигурировать представление вью

**private** **func** setupUI() {

configureSubviews()

addAllSubviews()

}

/// Конфигурировать представление внутренних вью

**private** **func** configureSubviews() {

configureLogoImage()

configureSmallImage()

configureNameLabel()

configureAddressLabel()

configureDistanceLabel()

}

/// Конфигурировать логотип компании

**private** **func** configureLogoImage() {

logoImage.translatesAutoresizingMaskIntoConstraints = **false**

logoImage.layer.cornerRadius = 13

logoImage.layer.shadowColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.25).cgColor

logoImage.layer.shadowRadius = 1

logoImage.layer.shadowOffset = CGSize(width: 0, height: 4)

logoImage.layer.masksToBounds = **true**

}

/// Конфигурировать картинку типа компании

**private** **func** configureSmallImage() {

smallImage.translatesAutoresizingMaskIntoConstraints = **false**

smallImage.tintColor = UIColor(named: "dark\_blue")

}

/// Конфигурировать текст названия компании

**private** **func** configureNameLabel() {

nameLabel.translatesAutoresizingMaskIntoConstraints = **false**

nameLabel.font = UIFont.systemFont(ofSize: 16, weight: .bold)

nameLabel.textColor = UIColor(named: "dark\_blue")

}

/// Конфигурировать текст адреса компании

**private** **func** configureAddressLabel() {

addressLabel.translatesAutoresizingMaskIntoConstraints = **false**

addressLabel.font = UIFont.systemFont(ofSize: 14, weight: .regular)

addressLabel.textColor = .lightGray

}

/// Конфигурировать текст расстояния

**private** **func** configureDistanceLabel() {

distanceLabel.translatesAutoresizingMaskIntoConstraints = **false**

distanceLabel.font = UIFont.systemFont(ofSize: 13, weight: .medium)

distanceLabel.textColor = .lightGray

}

/// Добавить все вью

**private** **func** addAllSubviews() {

addSubview(logoImage)

addSubview(smallImage)

addSubview(nameLabel)

addSubview(addressLabel)

addSubview(distanceLabel)

}

/// Перерисовать вью

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

NSLayoutConstraint.activate([

logoImage.leadingAnchor.constraint(equalTo: leadingAnchor),

logoImage.heightAnchor.constraint(equalTo: heightAnchor),

logoImage.widthAnchor.constraint(equalTo: logoImage.heightAnchor),

logoImage.topAnchor.constraint(equalTo: topAnchor),

smallImage.leadingAnchor.constraint(equalTo: logoImage.trailingAnchor, constant: 14),

smallImage.widthAnchor.constraint(equalToConstant: 20),

smallImage.heightAnchor.constraint(equalTo: smallImage.widthAnchor),

smallImage.topAnchor.constraint(equalTo: topAnchor, constant: 10),

nameLabel.centerYAnchor.constraint(equalTo: smallImage.centerYAnchor),

nameLabel.leadingAnchor.constraint(equalTo: smallImage.trailingAnchor, constant: 4),

nameLabel.trailingAnchor.constraint(lessThanOrEqualTo: trailingAnchor),

addressLabel.leadingAnchor.constraint(equalTo: smallImage.leadingAnchor),

addressLabel.topAnchor.constraint(equalTo: smallImage.bottomAnchor, constant: 6),

addressLabel.trailingAnchor.constraint(lessThanOrEqualTo: trailingAnchor),

addressLabel.bottomAnchor.constraint(lessThanOrEqualTo: bottomAnchor),

distanceLabel.topAnchor.constraint(equalTo: topAnchor),

distanceLabel.trailingAnchor.constraint(equalTo: trailingAnchor)

])

}

/// Конфигурировать вью по модели представления кафе

/// - **Parameter** vm: Модель представления кафе

**func** configure(with vm: RestaurantViewModel) {

**self**.logoImage.image = vm.logoImage

nameLabel.text = vm.name

addressLabel.text = vm.address

smallImage.image = vm.typeImage

distanceLabel.text = vm.distance

}

}

* 1. SaleCell.swift

//

// SailCell.swift

// GeoFood

//

// Created by Егор on 24.04.2021.

//

**import** UIKit

/// Ячейка таблицы с акцией

**class** SaleCell: UITableViewCell {

/// Задняя картинка для акции

**let** backImage = UIImageView()

/// Внешняя вью

**let** frontView = UIView()

/// Название акции

**let** saleName = UILabel()

/// Промокод акции

**let** saleCode = UILabel()

/// Цена акции

**let** salePrice = UILabel()

/// Конструктор

/// - **Parameters**:

/// - style: Стиль ячейки

/// - reuseIdentifier: id ячейки

**override** **init**(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {

**super**.init(style: style, reuseIdentifier: reuseIdentifier)

setupUI()

}

/// Конструктор

/// - **Parameter** coder: кодер

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

/// Вью загрузилось из storyboard

**override** **func** awakeFromNib() {

**super**.awakeFromNib()

configureSubviews()

addAllSubviews()

}

/// Конфигурировать вью

**private** **func** setupUI() {

configureSubviews()

addAllSubviews()

}

/// Конфигурировать внутренние вью

**private** **func** configureSubviews() {

layer.cornerRadius = 17

contentView.layer.cornerRadius = 17

contentView.layer.masksToBounds = **true**

**let** shadowPath = UIBezierPath(roundedRect: contentView.bounds, cornerRadius: 17)

contentView.layer.shadowPath = shadowPath.cgPath

contentView.layer.shadowColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.2).cgColor

contentView.layer.shadowOpacity = 1

contentView.layer.shadowRadius = 6

contentView.layer.shadowOffset = CGSize(width: 0, height: 0)

backgroundColor = .clear

backImage.translatesAutoresizingMaskIntoConstraints = **true**

backImage.layer.masksToBounds = **true**

contentView.translatesAutoresizingMaskIntoConstraints = **false**

frontView.layer.cornerRadius = layer.cornerRadius

frontView.translatesAutoresizingMaskIntoConstraints = **false**

frontView.layer.cornerRadius = 17

frontView.backgroundColor = .white

saleName.translatesAutoresizingMaskIntoConstraints = **false**

saleName.font = UIFont.systemFont(ofSize: 15, weight: .semibold)

saleName.textColor = UIColor(named: "dark\_blue")

saleName.numberOfLines = 2

saleName.lineBreakMode = .byWordWrapping

saleCode.translatesAutoresizingMaskIntoConstraints = **false**

saleCode.font = UIFont.systemFont(ofSize: 13, weight: .light)

saleCode.textColor = UIColor(named: "dark\_blue")

salePrice.translatesAutoresizingMaskIntoConstraints = **false**

salePrice.font = UIFont.systemFont(ofSize: 14, weight: .regular)

salePrice.textColor = UIColor(named: "dark\_blue")

}

/// Добавить все вью

**private** **func** addAllSubviews() {

contentView.addSubview(backImage)

contentView.addSubview(frontView)

frontView.addSubview(saleName)

frontView.addSubview(saleCode)

frontView.addSubview(salePrice)

}

/// Перерисовать вью

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

NSLayoutConstraint.activate([

contentView.topAnchor.constraint(equalTo: topAnchor, constant: 5),

contentView.bottomAnchor.constraint(equalTo: bottomAnchor, constant: -5),

contentView.leadingAnchor.constraint(equalTo: leadingAnchor),

contentView.trailingAnchor.constraint(equalTo: trailingAnchor),

frontView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 70),

frontView.topAnchor.constraint(equalTo: topAnchor),

frontView.trailingAnchor.constraint(equalTo: trailingAnchor),

frontView.bottomAnchor.constraint(equalTo: bottomAnchor),

saleName.topAnchor.constraint(equalTo: frontView.topAnchor, constant: 20),

saleName.leadingAnchor.constraint(equalTo: frontView.leadingAnchor, constant: 15),

saleName.trailingAnchor.constraint(lessThanOrEqualTo: salePrice.leadingAnchor, constant: -5),

saleCode.topAnchor.constraint(equalTo: saleName.bottomAnchor, constant: 2),

saleCode.leadingAnchor.constraint(equalTo: saleName.leadingAnchor),

saleCode.trailingAnchor.constraint(equalTo: frontView.trailingAnchor),

saleCode.bottomAnchor.constraint(lessThanOrEqualTo: frontView.bottomAnchor),

salePrice.topAnchor.constraint(equalTo: frontView.topAnchor, constant: 20),

salePrice.trailingAnchor.constraint(equalTo: frontView.trailingAnchor, constant: -11),

])

backImage.frame = CGRect(x: 0, y: 0, width: 85, height: contentView.frame.height)

}

/// Сконфигурировать вью по модели представления акции

/// - **Parameter** vm: Модель представления

**func** configure(with vm: SaleViewModel) {

**self**.backImage.image = vm.image

saleName.text = vm.name

salePrice.text = vm.price

saleCode.text = vm.code

}

}

* 1. SpecialSaleCollectionViewCell.swift

//

// SpecialSaleCollectionViewCell.swift

// GeoFood

//

// Created by Егор on 28.04.2021.

//

**import** UIKit

**import** CoreImage.CIFilterBuiltins

/// Ячейка специальной акции

**class** SpecialSaleCollectionViewCell: UICollectionViewCell {

/// Контекст для генерации qr-кода

**private** **let** context = CIContext()

/// Фильтр для генерации qr-кода

**private** **let** filter = CIFilter.qrCodeGenerator()

/// Картинка акции

**private** **let** image = UIImageView()

/// Название акции

**private** **let** title = UILabel()

/// Промокод акции

**private** **let** codeLabel = UILabel()

/// Кнопка получения акции

**private** **let** getButton = NextButton()

/// Вью акции

**private** **let** frontView = UIView()

/// Вью qr-кода

**private** **let** backView = UIView()

/// Картинка qr-кода

**private** **let** qrImage = UIImageView()

/// Кнопка закрытия qr-кода

**private** **let** closeQRButton = UIImageView()

/// Открыта ли вью акции

**private** **var** isFront = **true**

/// Конструктор

/// - **Parameter** frame: Фрейм ячейки

**override** **init**(frame: CGRect) {

**super**.init(frame: frame)

setupUI()

}

/// Конструктор

/// - **Parameter** coder: кодер

**required** **init**?(coder: NSCoder) {

**super**.init(coder: coder)

setupUI()

}

/// Конфигурировать вью

**private** **func** setupUI() {

backgroundColor = .clear

contentView.backgroundColor = .white

contentView.layer.cornerRadius = 17

**let** shadowPath = UIBezierPath(roundedRect: contentView.bounds, cornerRadius: 17)

contentView.layer.shadowPath = shadowPath.cgPath

contentView.layer.shadowColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.2).cgColor

contentView.layer.shadowOpacity = 1

contentView.layer.shadowRadius = 6

contentView.layer.shadowOffset = CGSize(width: 0, height: 0)

image.translatesAutoresizingMaskIntoConstraints = **false**

image.layer.cornerRadius = 17

image.contentMode = .scaleAspectFill

image.layer.masksToBounds = **true**

frontView.addSubview(image)

title.translatesAutoresizingMaskIntoConstraints = **false**

title.textColor = UIColor(named: "dark\_blue")

title.font = UIFont.systemFont(ofSize: 15, weight: .semibold)

title.numberOfLines = 3

title.lineBreakMode = .byWordWrapping

frontView.addSubview(title)

codeLabel.translatesAutoresizingMaskIntoConstraints = **false**

codeLabel.textColor = UIColor(named: "dark\_blue")

codeLabel.font = UIFont.systemFont(ofSize: 13, weight: .light)

frontView.addSubview(codeLabel)

getButton.translatesAutoresizingMaskIntoConstraints = **false**

getButton.setTitle("Получить", for: .normal)

getButton.titleLabel?.font = UIFont.systemFont(ofSize: 16, weight: .semibold)

getButton.addTarget(**self**, action: **#selector**(didTapButton), for: .touchUpInside)

frontView.addSubview(getButton)

qrImage.translatesAutoresizingMaskIntoConstraints = **false**

backView.addSubview(qrImage)

closeQRButton.translatesAutoresizingMaskIntoConstraints = **false**

closeQRButton.image = UIImage(systemName: "xmark.circle")

closeQRButton.tintColor = .lightGray

backView.addSubview(closeQRButton)

frontView.backgroundColor = .white

backView.backgroundColor = .white

frontView.layer.cornerRadius = 17

backView.layer.cornerRadius = 17

frontView.translatesAutoresizingMaskIntoConstraints = **false**

backView.translatesAutoresizingMaskIntoConstraints = **false**

contentView.addSubview(backView)

contentView.addSubview(frontView)

**let** tapGesture = UITapGestureRecognizer(target: **self**, action: **#selector**(didTapButton))

backView.addGestureRecognizer(tapGesture)

}

/// Отрисовать внутренние вью

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

NSLayoutConstraint.activate([

frontView.topAnchor.constraint(equalTo: contentView.topAnchor),

frontView.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),

frontView.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),

frontView.bottomAnchor.constraint(equalTo: contentView.bottomAnchor),

backView.topAnchor.constraint(equalTo: contentView.topAnchor),

backView.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),

backView.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),

backView.bottomAnchor.constraint(equalTo: contentView.bottomAnchor),

image.topAnchor.constraint(equalTo: frontView.topAnchor, constant: 12),

image.leadingAnchor.constraint(equalTo: frontView.leadingAnchor, constant: 12),

image.heightAnchor.constraint(equalToConstant: 136),

image.widthAnchor.constraint(equalTo: image.heightAnchor),

title.topAnchor.constraint(equalTo: image.topAnchor),

title.leadingAnchor.constraint(equalTo: image.trailingAnchor, constant: 12),

title.trailingAnchor.constraint(equalTo: frontView.trailingAnchor, constant: -15),

codeLabel.topAnchor.constraint(equalTo: title.bottomAnchor, constant: 10),

codeLabel.leadingAnchor.constraint(equalTo: title.leadingAnchor),

codeLabel.trailingAnchor.constraint(equalTo: title.trailingAnchor),

getButton.trailingAnchor.constraint(equalTo: frontView.trailingAnchor, constant: -15),

getButton.bottomAnchor.constraint(equalTo: image.bottomAnchor),

getButton.heightAnchor.constraint(equalToConstant: 50),

getButton.widthAnchor.constraint(equalToConstant: 120),

qrImage.centerXAnchor.constraint(equalTo: backView.centerXAnchor),

qrImage.centerYAnchor.constraint(equalTo: backView.centerYAnchor),

qrImage.heightAnchor.constraint(equalToConstant: 100),

qrImage.widthAnchor.constraint(equalTo: qrImage.heightAnchor),

closeQRButton.heightAnchor.constraint(equalToConstant: 20),

closeQRButton.widthAnchor.constraint(equalTo: closeQRButton.heightAnchor),

closeQRButton.topAnchor.constraint(equalTo: backView.topAnchor, constant: 10),

closeQRButton.trailingAnchor.constraint(equalTo: backView.trailingAnchor, constant: -10)

])

}

/// Событие нажатия на кнопку открытия/закрытия акции

**@objc** **private** **func** didTapButton() {

**let** fromView = isFront ? frontView : backView

**let** toView = isFront ? backView : frontView

UIView.transition(from: fromView, to: toView, duration: 0.5, options: [.curveEaseInOut, .transitionFlipFromLeft, .showHideTransitionViews])

isFront.toggle()

}

/// Сконфигурировать вью по модели представления акции

/// - **Parameter** vm: Модель представления акции

**func** configure(with vm: SaleViewModel) {

title.text = vm.name

codeLabel.text = vm.code

**self**.image.image = vm.image

qrImage.image = generateQRCode(from: vm.code)

}

/// Генерация картинки qr-кода

/// - **Parameter** string: Строка qr-кода

/// - **Returns**: Картинка с qr-кодом

**private** **func** generateQRCode(from string: String) -> UIImage? {

**let** data = Data(string.utf8)

**guard** **let** qrFilter = CIFilter(name: "CIQRCodeGenerator") **else** { **return** **nil** }

qrFilter.setValue(data, forKey: "inputMessage")

**guard** **let** qrImage = qrFilter.outputImage?.transformed(by: CGAffineTransform(scaleX: 10, y: 10)) **else** { **return** **nil** }

**return** UIImage(ciImage: qrImage)

}

}

* 1. RestaurantViewModel.swift

//

// RestaurantViewModel.swift

// GeoFood

//

// Created by Егор on 24.04.2021.

//

**import** Foundation

**import** UIKit

**import** MapKit

/// Модель представления данных кафе

**class** RestaurantViewModel {

/// Названия кафе

**let** name: String

/// Адрес кафе

**var** address: String

/// Расстояние до кафе

**let** distance: String

/// Картинка типа акции

**let** typeImage: UIImage

/// Логотип кафе

**let** logoImage: UIImage

/// Акции кафе

**let** sales: [SaleViewModel]

/// Специальные кафе

**let** specialSales: [SaleViewModel]

/// Конструктор

/// - **Parameters**:

/// - model: Модель представления данных кафе

/// - allSales: Все акции кафе

**init**(with model: RestaurantModel, allSales: [RestaurantSaleModel]) {

name = model.name

address = model.location

**let** locationCoords = CLLocation(latitude: model.latitude, longitude: model.longitude)

**let** distanceValue = LocationManager.shared.userDistance(to: locationCoords) / 1000

distance = "\(String(format: "%.1f", distanceValue))км"

logoImage = model.logoImage ?? UIImage(named: "empty")!

typeImage = model.type.image

**var** sales = [SaleViewModel]()

**var** specialSales = [SaleViewModel]()

**for** sale **in** allSales {

**if** sale.special {

specialSales.append(SaleViewModel(with: sale))

} **else** {

sales.append(SaleViewModel(with: sale))

}

}

**self**.sales = sales

**self**.specialSales = specialSales

**let** group = DispatchGroup()

**let** geoCoder = CLGeocoder()

group.enter()

geoCoder.reverseGeocodeLocation(locationCoords, completionHandler: { placemarks, error **in**

**guard** **let** placemarks = placemarks **else** {

**return**

}

**let** placemark = placemarks[0]

**let** houserNumber = placemark.subThoroughfare

**let** street = placemark.thoroughfare

**if** street == **nil** {

**self**.address = ""

} **else** **if** houserNumber == **nil** {

**self**.address = "\(street!)"

} **else** {

**self**.address = "\(street!), \(houserNumber!)"

}

group.leave()

})

group.wait()

}

}

* 1. SaleViewModel.swift

//

// SaleViewModel.swift

// GeoFood

//

// Created by Егор on 24.04.2021.

//

**import** Foundation

**import** UIKit

/// Модель представления данных акции

**class** SaleViewModel {

/// Имя акции

**let** name: String

/// Цена акции

**let** price: String

/// Промокод акции

**let** code: String

/// Картинка акции

**let** image: UIImage

/// Конструктор из модели акции

/// - **Parameter** model: Модель акции

**init**(with model: RestaurantSaleModel) {

name = model.name

price = "\(model.newPrice)\u{20BD}"

code = "Промокод: \(model.promo ?? "F2C4")"

image = model.image ?? UIImage(named: "empty")!

}

}

* 1. RestaurantPresenter.swift

//

// RestaurantPresenter.swift

// GeoFood

//

// Created by Егор on 02.05.2021.

//

**import** Foundation

/// Выходные данные презентера кафе

**protocol** RestaurantPresenterOutput: AnyObject {

**func** configure(with vm: RestaurantViewModel)

}

/// Презентер кафе

**class** RestaurantPresenter {

/// Текущее кафе

**let** currentRestaurant: RestaurantModel

/// Контроллер кафе

**weak** **var** view: RestaurantViewController!

/// Интерактор кафе

**var** interactor: RestaurantInteractorInput!

/// Конструктор

/// - **Parameters**:

/// - view: Контроллер кафе

/// - currentRestaurant: Текущее кафе

**init**(with view: RestaurantViewController, currentRestaurant: RestaurantModel) {

**self**.view = view

**self**.currentRestaurant = currentRestaurant

}

}

**extension** RestaurantPresenter: RestaurantViewControllerOutput {

/// Контроллер загрузился

**func** viewDidLoad() {

interactor.loadSales(for: currentRestaurant.id)

}

}

**extension** RestaurantPresenter: RestaurantInteractorOutput {

/// Акции кафе загружены

/// - **Parameter** sales: Массив загруженных акций

**func** loadedSales(**\_** sales: [RestaurantSaleModel]) {

**let** viewModel = RestaurantViewModel(with: currentRestaurant, allSales: sales)

DispatchQueue.main.async {

**self**.view.configure(with: viewModel)

}

}

}

* 1. RestaurantConfigurator.swift

//

// RestaurantConfigurator.swift

// GeoFood

//

// Created by Егор on 04.05.2021.

//

**import** Foundation

/// Конфигуратор модуля кафе

**class** RestaurantConfigurator {

/// Конфигурировать модуль

/// - **Parameter** restaurant: Текущее кафе

/// - **Returns**: Сконфигурированный контроллер модуля

**static** **func** assembly(with restaurant: RestaurantModel) -> RestaurantViewController {

**let** view = RestaurantViewController()

**let** presenter = RestaurantPresenter(with: view, currentRestaurant: restaurant)

view.presenter = presenter

**let** interactor = RestaurantInteractor(with: presenter)

presenter.interactor = interactor

**return** view

}

}

* 1. RestaurantInteractor.swift

//

// RestaurantInteractor.swift

// GeoFood

//

// Created by Егор on 04.05.2021.

//

**import** Foundation

**import** UIKit

/// Входные методы интерактора

**protocol** RestaurantInteractorInput: AnyObject {

/// Загрузить акции кафе

/// - **Parameter** restId: id кафе

**func** loadSales(for restId: Int32)

}

/// Выходные методы интерактора

**protocol** RestaurantInteractorOutput: AnyObject {

/// Акции, загруженные из сети

/// - **Parameter** sales: Массив загруженных акций

**func** loadedSales(**\_** sales: [RestaurantSaleModel])

}

**class** RestaurantInteractor: RestaurantInteractorInput {

/// Сервис пользователя

**let** userService = UserService.shared

/// Презентер кафе

**weak** **var** output: RestaurantInteractorOutput!

/// Конструктор

/// - **Parameter** output: Презентер кафе

**init**(with output: RestaurantInteractorOutput) {

**self**.output = output

}

/// Загрузить акции

/// - **Parameter** restId: id кафеы

**func** loadSales(for restId: Int32) {

**self**.userService.getRestaurantSales(restaurantId: restId){ sales **in**

**guard** **let** sales = sales **else** {

**self**.output.loadedSales([])

**return**

}

**let** group = DispatchGroup()

**for** sale **in** sales {

group.enter()

ImageLoader.loadSaleImage(saleId: sale.id) { data **in**

**if** **let** data = data {

sale.image = UIImage(data: data)

} **else** {

sale.image = UIImage(named: "empty")

}

group.leave()

}

}

group.notify(queue: DispatchQueue.global(qos: .utility), work: DispatchWorkItem {

**self**.output.loadedSales(sales)

})

}

}

}

* 1. RestaurantAnnotation.swift

//

// StockAnnotation.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** UIKit

**import** MapKit

/// Аннотация кафе на карте

**class** RestaurantAnnotation: MKAnnotationView {

/// Кафе для аннотации

**var** restaurant: MapRestaurantViewModel!

/// Конструктор

/// - **Parameters**:

/// - annotation: Исходная аннотация

/// - reuseIdentifier: id аннотации

**override** **init**(annotation: MKAnnotation?, reuseIdentifier: String?) {

**super**.init(annotation: annotation, reuseIdentifier: reuseIdentifier)

}

/// Конфигурировать аннотацию по модели представления кафе

/// - **Parameter** vm: Модель представления кафе

**func** configure(with vm: MapRestaurantViewModel) {

**self**.restaurant = vm

**self**.image = vm.image

}

/// Конструктор

/// - **Parameter** aDecoder: Кодер

**required** **init**?(coder aDecoder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

}

* 1. RestaurantTypeCell.swift

//

// RestaurantTypeCell.swift

// GeoFood

//

// Created by Егор on 26.04.2021.

//

**import** UIKit

/// Ячейка типа ресторана

**class** RestaurantTypeCell: UICollectionViewCell {

/// Название типа

**private** **let** textLabel = UILabel()

/// Картинка типа

**private** **let** image = UIImageView()

/// Выбран ли тип

**var** selectionMode = **false**

/// Конструктор

/// - **Parameter** frame: Фрейм ячейки

**override** **init**(frame: CGRect) {

**super**.init(frame: .zero)

setupUI()

}

/// Конфигурировать вью

**private** **func** setupUI() {

textLabel.font = UIFont.systemFont(ofSize: 10, weight: .semibold)

textLabel.textColor = UIColor(named: "dark\_blue")

textLabel.translatesAutoresizingMaskIntoConstraints = **false**

image.translatesAutoresizingMaskIntoConstraints = **false**

image.tintColor = UIColor(named: "dark\_blue")

layer.cornerRadius = 15

layer.masksToBounds = **true**

layer.borderColor = UIColor(named: "light\_green")!.cgColor

**let** blurEffect = UIBlurEffect(style: .light)

**let** blurEffectView = UIVisualEffectView(effect: blurEffect)

blurEffectView.frame = contentView.bounds

blurEffectView.autoresizingMask = [.flexibleWidth, .flexibleHeight]

contentView.addSubview(blurEffectView)

backgroundColor = UIColor(named: "lime")

contentView.addSubview(textLabel)

contentView.addSubview(image)

}

/// Перерисовать вью

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

NSLayoutConstraint.activate([

image.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 15),

image.centerYAnchor.constraint(equalTo: contentView.centerYAnchor),

image.heightAnchor.constraint(equalToConstant: 20),

image.widthAnchor.constraint(equalTo: image.heightAnchor),

textLabel.leadingAnchor.constraint(equalTo: image.trailingAnchor, constant: 5),

textLabel.centerYAnchor.constraint(equalTo: image.centerYAnchor),

textLabel.trailingAnchor.constraint(lessThanOrEqualTo: contentView.trailingAnchor, constant: -5)

])

}

/// Конфигурировать ячейку по типу кафе

/// - **Parameter** type: Тип кафе

**func** configure(with type: RestaurantType) {

textLabel.text = type.text

image.image = type.image

}

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

}

* 1. MapViewController.swift

//

// MapViewController.swift

// GeoFood

//

// Created by Егор on 17.03.2021.

//

**import** UIKit

**import** MapKit

/// Входные методы карты

**protocol** MapViewInput: AnyObject {

/// Конфигурировать вью по модели представления карты

/// - **Parameter** vm: Модель представления карты

**func** configure(with vm: MapViewModel)

}

/// Контроллер карты

**class** MapViewController: UIViewController {

/// Конфигуратор модуля карты

**let** configurator: MapConfiguratorProtocol = MapConfigurator()

/// Последняя позиция пользователя

**var** lastLocation: CLLocation?

/// Презентер карты

**var** presenter: MapPresenterInput!

/// Карта

**let** mapView = MKMapView(frame: .zero)

/// Сервис положения пользователя

**let** locationManager = LocationManager.shared

/// Модель представления карты

**var** viewModel: MapViewModel?

/// id аннотации

**private** **let** annotationId = "restaurantAnnotation"

/// Первое ли отображение карты

**private** **var** isFirstAppear = **true**

/// Коллекция типов кафе

**private** **var** typesCollection: UICollectionView = {

**let** layout = UICollectionViewFlowLayout()

layout.scrollDirection = .horizontal

**return** UICollectionView(frame: .zero, collectionViewLayout: layout)

}()

/// Переместить карту на текущую позицию пользователя

**private** **let** currentLocationButton = UIButton()

/// Контроллер загрузился

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

configurator.configure(with: **self**)

view.backgroundColor = .white

mapView.showsUserLocation = **true**

mapView.showsCompass = **false**

navigationController?.setNavigationBarHidden(**true**, animated: **false**)

mapView.translatesAutoresizingMaskIntoConstraints = **false**

mapView.register(RestaurantAnnotation.**self**, forAnnotationViewWithReuseIdentifier: annotationId)

locationManager.requestWhenInUseAuthorization()

locationManager.startUpdatingLocation()

locationManager.delegate = **self**

mapView.delegate = **self**

view.addSubview(mapView)

typesCollection.translatesAutoresizingMaskIntoConstraints = **false**

typesCollection.dataSource = **self**

typesCollection.delegate = **self**

typesCollection.register(RestaurantTypeCell.**self**, forCellWithReuseIdentifier: "typeCell")

typesCollection.backgroundColor = .clear

typesCollection.showsHorizontalScrollIndicator = **false**

typesCollection.contentInset = UIEdgeInsets(top: 0, left: 30, bottom: 0, right: 30)

view.addSubview(typesCollection)

currentLocationButton.frame = CGRect(x: view.frame.maxX - 70, y: view.frame.minY + 50, width: 50, height: 50)

currentLocationButton.layer.cornerRadius = currentLocationButton.frame.width / 2

currentLocationButton.backgroundColor = .white

currentLocationButton.imageView?.tintColor = UIColor(named: "dark\_blue")

currentLocationButton.addTarget(**self**, action: **#selector**(changeFollowMode), for: .touchUpInside)

view.addSubview(currentLocationButton)

NSLayoutConstraint.activate([

mapView.topAnchor.constraint(equalTo: view.topAnchor),

mapView.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor),

mapView.trailingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.trailingAnchor),

mapView.bottomAnchor.constraint(equalTo: view.bottomAnchor),

typesCollection.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 0),

typesCollection.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: 0),

typesCollection.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor, constant: -15),

typesCollection.heightAnchor.constraint(equalToConstant: 40)

])

}

/// Контроллер отобразился

/// - **Parameter** animated: Анимировано ли отобразился

**override** **func** viewDidAppear(**\_** animated: Bool) {

**super**.viewDidAppear(animated)

**self**.navigationController?.setNavigationBarHidden(**true**, animated: **false**)

**if** isFirstAppear {

mapView.setUserTrackingMode(.followWithHeading, animated: **false**)

isFirstAppear = **false**

}

checkAuthorizationStatus()

presenter.viewDidAppear()

}

/// Изменить способ следования за пользователем

**@objc** **func** changeFollowMode() {

**if** mapView.userTrackingMode == .none {

mapView.setUserTrackingMode(.follow, animated: **true**)

} **else** {

mapView.setUserTrackingMode(.followWithHeading, animated: **true**)

}

}

/// Проверка возможности отслеживать позицию пользователя

**private** **func** checkAuthorizationStatus() {

**if** locationManager.authorizationStatus == .denied {

**let** alert = UIAlertController(title: "Ошибка", message: "Разрешите доступ к вашей геопозиции", preferredStyle: .alert)

**let** action = UIAlertAction(title: "Перейти в настройки", style: .default, handler: { **\_** **in**

**guard** **let** settingsUrl = URL(string: UIApplication.openSettingsURLString) **else** {

**return**

}

**if** UIApplication.shared.canOpenURL(settingsUrl) {

UIApplication.shared.open(settingsUrl)

}

})

alert.addAction(action)

present(alert, animated: **true**, completion: **nil**)

}

}

}

**extension** MapViewController: MapViewInput {

/// Конфигурировать по модели представления карты

/// - **Parameter** vm: Модель представления карты

**func** configure(with vm: MapViewModel) {

**self**.viewModel = vm

**if** vm.deleteAll {

**self**.mapView.removeAnnotations(mapView.annotations)

}

**self**.mapView.removeAnnotations(**self**.viewModel?.deletedRestaurants.map{ $0.annotation } ?? [])

**for** rest **in** viewModel!.mapRestaurants {

mapView.addAnnotation(rest.annotation)

}

}

}

**extension** MapViewController: MKMapViewDelegate {

/// Событие изменения отображаемого региона на карте

/// - **Parameters**:

/// - mapView: Карта

/// - animated: Анимировано ли отобразился

**func** mapView(**\_** mapView: MKMapView, regionDidChangeAnimated animated: Bool) {

presenter.regionDidChange(region: mapView.region, radius: mapView.currentRadius())

}

/// Сконфигурировать вью для аннотации

/// - **Parameters**:

/// - mapView: Карта, на которой отображается аннотация

/// - annotation: Аннотация, для которой конфигурируется вью

/// - **Returns**: Вью для аннотации

**func** mapView(**\_** mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {

**if** annotation **is** MKUserLocation {

**let** userAnnotation = MKUserLocationView()

userAnnotation.annotation = annotation

userAnnotation.tintColor = UIColor(named: "dark\_blue")

**return** userAnnotation

}

**guard** **let** restaurantVM = **self**.viewModel?.mapRestaurants.first(where: { $0.annotation.coordinate.latitude == annotation.coordinate.latitude && $0.annotation.coordinate.longitude == annotation.coordinate.longitude })

**else** {

**return** **nil**

}

**guard** **let** annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: annotationId) **as**? RestaurantAnnotation **else** {

**let** restAnnotation = RestaurantAnnotation(annotation: annotation, reuseIdentifier: annotationId)

restAnnotation.restaurant = restaurantVM

**return** restAnnotation

}

annotationView.configure(with: restaurantVM)

annotationView.bounds = CGRect(x: 0, y: 0, width: 40, height: 40)

annotationView.layer.masksToBounds = **true**

annotationView.layer.cornerRadius = 20

**return** annotationView

}

/// Событие клика на аннотацию

/// - **Parameters**:

/// - mapView: Карта

/// - view: Выбранное вью

**func** mapView(**\_** mapView: MKMapView, didSelect view: MKAnnotationView) {

mapView.deselectAnnotation(view.annotation, animated: **false**)

presenter.annotationTapped(view)

}

/// Событие изменения способа следования за пользователем

/// - **Parameters**:

/// - mapView: Карта

/// - mode: Способ следования

/// - animated: Анимировано ли изменен

**func** mapView(**\_** mapView: MKMapView, didChange mode: MKUserTrackingMode, animated: Bool) {

**if** mode == .follow {

currentLocationButton.setImage(UIImage(systemName: "location.fill"), for: .normal)

} **else** **if** mode == .followWithHeading {

currentLocationButton.setImage(UIImage(systemName: "location.fill"), for: .normal)

} **else** {

currentLocationButton.setImage(UIImage(systemName: "location"), for: .normal)

}

}

}

**extension** MapViewController: CLLocationManagerDelegate {

/// Событие изменения позиции пользователя

/// - **Parameters**:

/// - manager: Сервис

/// - locations: Все позиции пользователя

**func** locationManager(**\_** manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

**self**.presenter.updateUserLocation(locations)

}

/// Событие изменения доступа к отслеживанию позиции пользователя

/// - **Parameter** manager: Сервис

**func** locationManagerDidChangeAuthorization(**\_** manager: CLLocationManager) {

checkAuthorizationStatus()

}

}

**extension** MapViewController: UICollectionViewDataSource {

/// Вычисление количества ячеек в коллекции

/// - **Parameters**:

/// - collectionView: Коллекция типов

/// - section: Секция коллекции

/// - **Returns**: Количество ячеек

**func** collectionView(**\_** collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {

RestaurantType.typesCount

}

/// Конфигурация ячейки коллекции

/// - **Parameters**:

/// - collectionView: Коллекция типов

/// - indexPath: Индекс ячейки

/// - **Returns**: Сконфигурированная ячейка

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

**guard** **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: "typeCell", for: indexPath) **as**? RestaurantTypeCell **else** {

**return** RestaurantTypeCell()

}

cell.configure(with: RestaurantType(rawValue: Int32(indexPath.row)) ?? .coffee)

**return** cell

}

/// Событие выбора ячейки коллекции

/// - **Parameters**:

/// - collectionView: Коллекция типов

/// - indexPath: индекс выбранной ячейки

**func** collectionView(**\_** collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {

**guard** **let** cell = collectionView.cellForItem(at: indexPath) **as**? RestaurantTypeCell **else** {

**return**

}

cell.selectionMode.toggle()

cell.layer.borderWidth = cell.selectionMode ? 3 : 0

**self**.presenter.changeFilter(for: indexPath.row, state: cell.selectionMode)

}

}

**extension** MapViewController: UICollectionViewDelegateFlowLayout {

/// Вычисление размера ячейки коллекции

/// - **Parameters**:

/// - collectionView: Коллекция типов

/// - collectionViewLayout: Layout коллекции

/// - indexPath: Индекс ячейки

/// - **Returns**: Размер ячейки

**func** collectionView(**\_** collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {

**return** CGSize(width: 120, height: 40)

}

}

* 1. MapViewModel.swift

//

// MapModel.swift

// GeoFood

//

// Created by Егор on 01.05.2021.

//

**import** Foundation

/// Модель представления карты

**class** MapViewModel {

/// Кафе, которые необходимо добавить на карту

**var** mapRestaurants: [MapRestaurantViewModel] = []

/// Кафе, которые необходимо удалить с карты

**var** deletedRestaurants: [MapRestaurantViewModel] = []

/// Все кафе

**var** allRests: [MapRestaurantViewModel] = []

/// Необходимо ли удалить все кафе с карта

**var** deleteAll = **false**

/// Конструктор слияния старой модели с новыми кафе

/// - **Parameters**:

/// - restaurants: Новые кафе

/// - oldModel: Старая модель

**init**(with restaurants: [RestaurantModel], oldModel: MapViewModel?) {

**if** **let** oldModel = oldModel {

**if** restaurants.count == 0 {

deleteAll = **true**

**return**

}

deletedRestaurants = oldModel.mapRestaurants.filter { old **in** !restaurants.contains(where: { old.id == $0.id }) }

**for** rest **in** restaurants {

**if** **let** oldRest = oldModel.allRests.first(where: { $0.id == rest.id }) {

allRests.append(oldRest)

} **else** {

allRests.append(MapRestaurantViewModel(with: rest))

}

}

mapRestaurants = allRests.filter{ rest **in** !oldModel.mapRestaurants.contains(where: { rest.id == $0.id }) }

} **else** {

allRests = restaurants.map{ MapRestaurantViewModel(with: $0) }

mapRestaurants = allRests

}

}

/// Обновить кафе по фильтру

/// - **Parameter** types: Типы кафе для фильтрации

**func** updateForTypes(**\_** types: [RestaurantType]) {

mapRestaurants = allRests.filter{ types.count == 0 || types.contains($0.type) }

deletedRestaurants = allRests.filter{ !types.contains($0.type) && types.count != 0 }

}

}

* 1. MapRestaurantViewModel.swift

//

// MapRestaurantModel.swift

// GeoFood

//

// Created by Егор on 01.05.2021.

//

**import** Foundation

**import** MapKit

/// Модель представления кафе на карте

**class** MapRestaurantViewModel {

/// id кафе

**let** id: Int32

/// Картинка кафе

**let** image: UIImage

/// Аннотация кафе

**let** annotation: MKPointAnnotation = MKPointAnnotation()

/// Тип кафе

**let** type: RestaurantType

/// Конструктор из модели кафе

/// - **Parameter** model: Модель кафе

**init**(with model: RestaurantModel) {

image = model.logoImage ?? UIImage(named: "empty")!

id = model.id

annotation.coordinate = CLLocationCoordinate2D(latitude: model.latitude, longitude: model.longitude)

**self**.type = model.type

}

}

* 1. MapPresenter.swift

//

// MapPresenter.swift

// GeoFood

//

// Created by Егор on 17.03.2021.

//

**import** Foundation

**import** MapKit

/// Входные методы презентера

**protocol** MapPresenterInput: class {

/// Регион, отображаемый на карте, изменен

/// - **Parameters**:

/// - region: Отображаемый регион

/// - radius: Радиус отображаемого региона

**func** regionDidChange(region: MKCoordinateRegion, radius: Double)

/// Событие нажатия на аннотацию

/// - **Parameter** annotation: Выбраная аннотация

**func** annotationTapped(**\_** annotation: MKAnnotationView)

/// Обновить позицию пользователя если она изменилась на 100 метров

/// - **Parameter** locations: Новая позиция пользователя

**func** updateUserLocation(**\_** locations: [CLLocation])

/// Событие изменения выбранных фильров

/// - **Parameters**:

/// - index: Числовое значение фильтра

/// - state: Состояние фильтра

**func** changeFilter(for index: Int, state: Bool)

/// Контроллер отобразился

**func** viewDidAppear()

}

/// Презентер кафе

**class** MapPresenter: MapPresenterInput {

/// Последняя позиция пользователя

**private** **var** lastLocation: CLLocation?

/// Типы кафе для фильтрации

**private** **var** filterTypes: [RestaurantType] = []

/// Вью кафе

**private** **weak** **var** view: MapViewInput!

/// Интерактор кафе

**var** interactor: MapInteractorInput!

/// Роутер кафе

**var** router: MapRouterProtocol!

/// Все кафе

**var** restaurants: [RestaurantModel] = []

/// Текущая модель представления

**private** **var** currentViewModel: MapViewModel?

/// Последний отображаемый на карте регион

**private** **var** lastRegion: MKCoordinateRegion?

/// Конструктор

/// - **Parameter** view: Контроллер карты

**required** **init**(view: MapViewController) {

**self**.view = view

}

/// Регион, отображаемый на карте, изменен

/// - **Parameters**:

/// - region: Отображаемый регион

/// - radius: Радиус отображаемого региона

**func** regionDidChange(region: MKCoordinateRegion, radius: Double) {

**if** **let** lastRegion = lastRegion {

**let** lastLocation = CLLocation(latitude: lastRegion.center.latitude, longitude: lastRegion.center.longitude)

**let** currentLocation = CLLocation(latitude: region.center.latitude, longitude: region.center.longitude)

**if** lastLocation.distance(from: currentLocation).isLessThanOrEqualTo(100) {

**return**

}

}

lastRegion = region

**self**.interactor.loadRestaurants(in: region.center, radius: radius)

}

/// Событие нажатия на аннотацию

/// - **Parameter** annotation: Выбраная аннотация

**func** annotationTapped(**\_** annotation: MKAnnotationView) {

**guard** **let** restAnnotation = annotation **as**? RestaurantAnnotation,

**let** restaurant = **self**.restaurants.first(where: { $0.id == restAnnotation.restaurant.id })

**else** {

**return**

}

**self**.router.openRestaurantView(with: restaurant)

}

/// Обновить позицию пользователя если она изменилась на 100 метров

/// - **Parameter** locations: Новая позиция пользователя

**func** updateUserLocation(**\_** locations: [CLLocation]) {

**guard** **let** newLocation = locations.last **else** {

**return**

}

**if** **let** lastLocation = lastLocation {

**if** lastLocation.distance(from: newLocation).isLessThanOrEqualTo(100) {

**return**

}

**self**.lastLocation = newLocation

} **else** {

**self**.lastLocation = newLocation

}

**self**.interactor.sendUserLocationUpdate(lastLocation!)

}

/// Событие изменения выбранных фильров

/// - **Parameters**:

/// - index: Числовое значение фильтра

/// - state: Состояние фильтра

**func** changeFilter(for index: Int, state: Bool) {

**if** state {

addFilter(for: index)

} **else** {

removeFilter(for: index)

}

}

/// Добавить фильтр

/// - **Parameter** index: Числовое значение филтра

**private** **func** addFilter(for index: Int) {

**guard** **let** newFilter = RestaurantType(rawValue: Int32(index)) **else** {

**return**

}

filterTypes.append(newFilter)

**guard** **let** currentViewModel = currentViewModel **else** {

**return**

}

currentViewModel.updateForTypes(filterTypes)

**self**.view.configure(with: currentViewModel)

}

/// Удалить фильтр

/// - **Parameter** index: Числовое значение фильтра

**private** **func** removeFilter(for index: Int) {

filterTypes.removeAll(where: { $0.rawValue == Int32(index) })

**guard** **let** currentViewModel = currentViewModel **else** {

**return**

}

currentViewModel.updateForTypes(filterTypes)

**self**.view.configure(with: currentViewModel)

}

/// Контроллер отобразился

**func** viewDidAppear() {

**guard** **let** coordinate = LocationManager.shared.location **else** {

**return**

}

interactor.loadRestaurants(in: coordinate.coordinate, radius: 1000)

}

}

**extension** MapPresenter: MapInteractorOutput {

/// Кафе загружены

/// - **Parameter** restaurants: Массив загруженных кафе

**func** restaurantsLoad(restaurants: [RestaurantModel]) {

**self**.restaurants = restaurants

**let** mapVM = MapViewModel(with: **self**.restaurants, oldModel: **self**.currentViewModel)

mapVM.updateForTypes(filterTypes)

**self**.currentViewModel = mapVM

DispatchQueue.main.async {

**self**.view.configure(with: mapVM)

}

}

}

* 1. MapConfigurator.swift

//

// MapConfigurator.swift

// GeoFood

//

// Created by Егор on 25.03.2021.

//

**import** Foundation

/// Протокол конфигуратора модуля карты

**protocol** MapConfiguratorProtocol: class {

/// Конфигурировать модуль по контроллеру

/// - **Parameter** with: Контроллер карты

**func** configure(with: MapViewController)

}

/// Конфигуратор модуля карты

**class** MapConfigurator: MapConfiguratorProtocol {

/// Конфигурировать модуль по контроллеру

/// - **Parameter** with: Контроллер карты

**func** configure(with view: MapViewController) {

**let** presenter = MapPresenter(view: view)

**let** interactor = MapInteractor(presenter: presenter, service: UserService.shared)

**let** router = MapRouter(view: view)

presenter.interactor = interactor

presenter.router = router

view.presenter = presenter

}

}

* 1. MapInteractor.swift

//

// MapInteractor.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** Foundation

**import** MapKit

/// Входные методы интеракторы карты

**protocol** MapInteractorInput: class {

/// Загрузить кафе

/// - **Parameters**:

/// - region: Отображаемый регион

/// - radius: Радиус отображения

**func** loadRestaurants(in region: CLLocationCoordinate2D, radius: Double)

/// Отправить изменение позиции пользователя

/// - **Parameter** location: Позиция пользователя

**func** sendUserLocationUpdate(**\_** location: CLLocation)

}

/// Выходные методы интерактора

**protocol** MapInteractorOutput: class {

/// Кафе загружены

/// - **Parameter** restaurants: Массив загруженных кафе

**func** restaurantsLoad(restaurants: [RestaurantModel])

}

**class** MapInteractor {

/// Презентер карты

**private** **weak** **var** presenter: MapInteractorOutput!

/// Сервис пользователя

**private** **var** service: UserService

/// Конструктор

/// - **Parameters**:

/// - presenter: Презентер карты

/// - service: Сервис пользователя

**required** **init**(presenter: MapInteractorOutput, service: UserService) {

**self**.presenter = presenter

**self**.service = service

}

}

**extension** MapInteractor: MapInteractorInput {

/// Загрузить кафе

/// - **Parameters**:

/// - region: Отображаемый регион

/// - radius: Радиус отображения

**func** loadRestaurants(in region: CLLocationCoordinate2D, radius: Double) {

**let** requestData = CoordinateRequestModel(coordinates: region, radius: radius)

service.getRestaurantsNear(coordinate: requestData) { restaurants **in**

**guard** **let** restaurants = restaurants **else** {

**self**.presenter.restaurantsLoad(restaurants: [])

**return**

}

**let** group = DispatchGroup()

**for** rest **in** restaurants {

group.enter()

ImageLoader.loadRestaurantImage(restId: rest.id) { data **in**

**if** **let** data = data {

rest.logoImage = UIImage(data: data)

} **else** {

rest.logoImage = UIImage(named: "empty")

}

group.leave()

}

}

group.notify(queue: DispatchQueue.global(qos: .utility), work: DispatchWorkItem {

**self**.presenter.restaurantsLoad(restaurants: restaurants)

})

}

}

/// Отправить изменение позиции пользователя

/// - **Parameter** location: Позиция пользователя

**func** sendUserLocationUpdate(**\_** location: CLLocation) {

service.updateUserLocation(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)

}

}

* 1. MapRouter.swift

//

// MapRouter.swift

// GeoFood

//

// Created by Егор on 26.03.2021.

//

**import** Foundation

**import** UIKit

/// Протокол роутера карты

**protocol** MapRouterProtocol: class {

/// Открыть модуль кафе

/// - **Parameters**:

/// - restaurant: Кафе для модуля

**func** openRestaurantView(with restaurant: RestaurantModel)

}

/// Роутер карты

**class** MapRouter: MapRouterProtocol {

/// Контроллер карты

**private** **var** view: MapViewController

/// Конструктор

/// - **Parameter** view: Контроллер карты

**required** **init**(view: MapViewController) {

**self**.view = view

}

/// Открыть модуль кафе

/// - **Parameters**:

/// - restaurant: Кафе для модуля

**func** openRestaurantView(with restaurant: RestaurantModel) {

view.present(UINavigationController(rootViewController: RestaurantConfigurator.assembly(with: restaurant)), animated: **true**, completion: **nil**)

}

}

* 1. RegistrationViewController.swift

//

// RegistrationViewController.swift

// GeoFood

//

// Created by Егор on 13.03.2021.

//

**import** UIKit

/// Выходные методы контроллера регистрации

**protocol** RegistrationViewOutput: class {

**func** setNavigationBarHidden(**\_** isHidden: Bool, animated: Bool)

**func** showAlert(title: String, message: String)

**func** getNavigationController() -> UINavigationController?

}

/// Контроллер регитсрации

**class** RegistrationViewController: UIViewController {

/// Конфигуратор модуля

**let** configurator: RegistrationConfiguratorProtocol = RegistrationConfigurator()

/// Презентер регистрации

**var** presenter: RegistrationPresenterInput!

/// Поле ввода почты

**var** emailTextField: TitledTextField = TitledTextField.emailTextField()

/// Поле ввода пароля

**var** passwordTextField: TitledTextField = TitledTextField.passwordTextField()

/// Поле повторного ввода пароля

**var** repeatPasswordTextField = TitledTextField.passwordTextField()

/// Кнопка регистрации

**let** registrationButton = UIButton()

/// Кнопка возвращения на экран авторизации

**let** backButton = UIButton()

/// Контроллер загружен

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

configurator.configure(with: **self**)

view.backgroundColor = .systemBackground

title = "Регистрация"

**self**.navigationItem.setHidesBackButton(**true**, animated: **false**)

configureSubviews()

addAllSubviews()

initConstraints()

}

/// Контроллер отобразился

/// - **Parameter** animated: Анимировано ли

**override** **func** viewDidAppear(**\_** animated: Bool) {

**super**.viewDidAppear(animated)

**self**.navigationController?.tabBarItem.title = **self**.title

presenter.viewDidAppear()

}

/// Конфигурировать вью

**private** **func** configureSubviews() {

configureEmailTextField()

configurePasswordTextField()

configureRepeatPasswordTextField()

configureRegistrationButton()

configureBackButton()

}

/// Конфигурировать поле ввода почты

**private** **func** configureEmailTextField() {

emailTextField.translatesAutoresizingMaskIntoConstraints = **false**

}

/// Конфигурировать поле ввода пароля

**private** **func** configurePasswordTextField() {

passwordTextField.translatesAutoresizingMaskIntoConstraints = **false**

}

/// Конфигурировать поле ввода пароля повторно

**private** **func** configureRepeatPasswordTextField() {

repeatPasswordTextField.translatesAutoresizingMaskIntoConstraints = **false**

repeatPasswordTextField.titleText = "Повторите пароль"

}

/// Конфигурировать кнопку регистрации

**private** **func** configureRegistrationButton() {

registrationButton.translatesAutoresizingMaskIntoConstraints = **false**

registrationButton.setTitle("Зарегистрироваться", for: .normal)

registrationButton.setTitleColor(UIColor(named: "dark\_blue"), for: .normal)

registrationButton.layer.borderColor = UIColor(named: "light\_green")?.cgColor

registrationButton.layer.borderWidth = 2

registrationButton.layer.cornerRadius = 10

registrationButton.backgroundColor = UIColor(named: "lime")

registrationButton.titleLabel?.font = UIFont.systemFont(ofSize: 16, weight: .semibold)

registrationButton.addTarget(**self**, action: **#selector**(registrationButtonTapped), for: .touchUpInside)

}

/// Конфигурировать кнопку возврата

**private** **func** configureBackButton() {

backButton.translatesAutoresizingMaskIntoConstraints = **false**

**let** attrString = NSAttributedString(string: "Войти", attributes: [NSAttributedString.Key.underlineStyle: NSUnderlineStyle.thick.rawValue])

backButton.setAttributedTitle(attrString, for: .normal)

backButton.setTitleColor(UIColor(named: "dark\_blue"), for: .normal)

backButton.addTarget(**self**, action: **#selector**(backButtonTapped), for: .touchUpInside)

}

/// Добавить все вью

**private** **func** addAllSubviews() {

view.addSubview(emailTextField)

view.addSubview(passwordTextField)

view.addSubview(repeatPasswordTextField)

view.addSubview(registrationButton)

view.addSubview(backButton)

}

/// Активировать констреинты

**private** **func** initConstraints() {

NSLayoutConstraint.activate([

emailTextField.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 22),

emailTextField.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 32),

emailTextField.trailingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.trailingAnchor, constant: -32),

emailTextField.heightAnchor.constraint(equalToConstant: 80),

passwordTextField.topAnchor.constraint(equalTo: emailTextField.bottomAnchor, constant: 7),

passwordTextField.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 32),

passwordTextField.trailingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.trailingAnchor, constant: -32),

passwordTextField.heightAnchor.constraint(equalToConstant: 80),

repeatPasswordTextField.topAnchor.constraint(equalTo: passwordTextField.bottomAnchor, constant: 7),

repeatPasswordTextField.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 32),

repeatPasswordTextField.trailingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.trailingAnchor, constant: -32),

repeatPasswordTextField.heightAnchor.constraint(equalToConstant: 80),

registrationButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),

registrationButton.topAnchor.constraint(equalTo: repeatPasswordTextField.bottomAnchor, constant: 22),

registrationButton.leadingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.leadingAnchor, constant: 46),

registrationButton.trailingAnchor.constraint(equalTo: view.safeAreaLayoutGuide.trailingAnchor, constant: -46),

registrationButton.heightAnchor.constraint(equalToConstant: 50),

backButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),

backButton.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor, constant: -66),

backButton.topAnchor.constraint(greaterThanOrEqualTo: registrationButton.bottomAnchor, constant: 20)

])

}

/// Проверить повторно введенные пароль

/// - **Parameter** password: Строка пароля

/// - **Returns**: Результат проверки

**private** **func** repeatPasswordValidator(**\_** password: String) -> Bool {

passwordTextField.titleText! == password

}

/// Событие нажатия на кнопку регистрации

**@objc** **func** registrationButtonTapped() {

presenter.registrationButtonTapped(withEmail: emailTextField.text, password: passwordTextField.text, passwordRepeat: repeatPasswordTextField.text)

}

/// Событие нажатия на кнопку выхода из вью

**@objc** **func** backButtonTapped() {

presenter.backButtonTapped()

}

}

**extension** RegistrationViewController: RegistrationViewOutput {

/// Поставить видимость навигации

/// - **Parameters**:

/// - isHidden: Спрятан ли

/// - animated: Анимировано

**func** setNavigationBarHidden(**\_** isHidden: Bool, animated: Bool) {

navigationController?.setNavigationBarHidden(**false**, animated: **true**)

}

/// Показать сообщение

/// - **Parameters**:

/// - title: Заголовок сообщения

/// - message: Текст сообщения

**func** showAlert(title: String, message: String) {

**let** alert = UIAlertController(title: title, message: message, preferredStyle: .alert)

**let** action = UIAlertAction(title: "Ok", style: .default, handler: **nil**)

alert.addAction(action)

present(alert, animated: **true**, completion: **nil**)

}

/// Получить контроллер навигации

/// - **Returns**: Контроллер навигации

**func** getNavigationController() -> UINavigationController? {

navigationController

}

}

* 1. RegistrationConfigurator.swift

//

// RegistrationConfigurator.swift

// GeoFood

//

// Created by Егор on 15.03.2021.

//

**import** Foundation

/// Протокол конфигуратора модуля

**protocol** RegistrationConfiguratorProtocol: class {

**func** configure(with: RegistrationViewController)

}

/// Конфигуратор модуля

**class** RegistrationConfigurator: RegistrationConfiguratorProtocol {

/// Конфигурировать модуль регистрации

/// - **Parameter** view: Контроллер модуля

**func** configure(with view: RegistrationViewController) {

**let** presenter = RegistrationPresenter(view: view)

**let** interactor = RegistrationInteractor(presenter: presenter)

view.presenter = presenter

presenter.interactor = interactor

presenter.router = RegistrationRouter(view: view)

}

}

* 1. RegistrationPresenter.swift

//

// RegistrationPresenter.swift

// GeoFood

//

// Created by Егор on 15.03.2021.

//

**import** Foundation

/// Входные методы презентера

**protocol** RegistrationPresenterInput: class {

/// Контроллер отобразился

**func** viewDidAppear()

/// Кнопка регистрации нажата

/// - **Parameters**:

/// - withEmail: Введенные пароль

/// - password: Введенный пароль

/// - passwordRepeat: Введенный повторно пароль

**func** registrationButtonTapped(withEmail: String, password: String, passwordRepeat: String)

/// Событие нажатия кнопки возврата

**func** backButtonTapped()

}

/// Презентер регистрации

**class** RegistrationPresenter: RegistrationPresenterInput {

/// Контроллер регистрации

**weak** **var** view: RegistrationViewOutput!

/// Интерактор регистрации

**var** interactor: RegistrationInteractorInput!

/// Роутер регистрации

**var** router: RegistrationRouterInput!

/// Конструктор

/// - **Parameter** view: Контроллер

**required** **init**(view: RegistrationViewOutput) {

**self**.view = view

}

/// Контроллер отобразился

**func** viewDidAppear() {

view.setNavigationBarHidden(**false**, animated: **true**)

}

/// Кнопка регистрации нажата

/// - **Parameters**:

/// - withEmail: Введенные пароль

/// - password: Введенный пароль

/// - passwordRepeat: Введенный повторно пароль

**func** registrationButtonTapped(withEmail: String, password: String, passwordRepeat: String) {

**if** /\*LoginEntryChecker.checkPassword(password) && password == passwordRepeat\*/ **true** {

interactor.registerUser(withEmail: withEmail, password: password)

} **else** {

view.showAlert(title: "Ошибка", message: "Проверьте введенные данные")

}

}

/// Событие нажатия кнопки возврата

**func** backButtonTapped() {

router.popBack()

}

}

**extension** RegistrationPresenter: RegistrationPresenterOutput {

/// Регистрация прошла успешно

**func** registrationSuccessfully() {

DispatchQueue.main.async {

**self**.router.openAccountView()

}

}

/// Регистрация не удалась

**func** registrationUnsuccessfully() {

DispatchQueue.main.async {

**self**.view.showAlert(title: "Ошибка", message: "Не удалось зарегистрироваться")

}

}

}

* 1. RegistrationInteractor.swift

//

// RegistrationInteractor.swift

// GeoFood

//

// Created by Егор on 15.03.2021.

//

**import** Foundation

/// Входные методы интерактора

**protocol** RegistrationInteractorInput: class {

/// Зарегистрировать пользователя

/// - **Parameters**:

/// - withEmail: Почта пользователя

/// - password: Пароль

**func** registerUser(withEmail: String, password: String)

}

/// Выходные методы интерактора

**protocol** RegistrationPresenterOutput: class {

/// Регистрация прошла успешно

**func** registrationSuccessfully()

/// Не удалось зарегистрироваться

**func** registrationUnsuccessfully()

}

/// Интерактор регистрации

**class** RegistrationInteractor: RegistrationInteractorInput {

/// Презентер регистрации

**weak** **var** presenter: RegistrationPresenterOutput!

/// Сервис пользователя

**var** userService: UserService = UserService.shared

/// Конструктор

/// - **Parameter** presenter: Презентер регистрации

**required** **init**(presenter: RegistrationPresenterOutput) {

**self**.presenter = presenter

}

/// Зарегистрировать пользователя

/// - **Parameters**:

/// - withEmail: Почта пользователя

/// - password: Пароль

**func** registerUser(withEmail: String, password: String) {

userService.registerUser(with: LoginForm(login: withEmail, password: password)) { isSuccess **in**

**if** isSuccess {

**self**.userService.authUser(with: LoginForm(login: withEmail, password: password)) { isSuccess **in**

**if** isSuccess {

**self**.presenter.registrationSuccessfully()

} **else** {

**self**.presenter.registrationUnsuccessfully()

}

}

} **else** {

**self**.presenter.registrationUnsuccessfully()

}

}

}

}

* 1. RegistrationRouter.swift

//

// RegistrationRouter.swift

// GeoFood

//

// Created by Егор on 15.03.2021.

//

**import** Foundation

/// Протокол роутера регистрации

**protocol** RegistrationRouterInput: class {

/// Вернуться на прошлый модуль

**func** popBack()

/// Открыть модуль аккаунта

**func** openAccountView()

}

/// Роутер регистрации

**class** RegistrationRouter: RegistrationRouterInput {

/// Контроллер регистрации

**var** view: RegistrationViewOutput

/// Конструктор

/// - **Parameter** view: Контроллер регистрации

**required** **init**(view: RegistrationViewOutput) {

**self**.view = view

}

/// /// Вернуться на прошлый модуль

**func** popBack() {

view.getNavigationController()?.popViewController(animated: **true**)

}

/// Открыть модуль аккаунта

**func** openAccountView() {

view.getNavigationController()?.pushViewController(AccountConfigurator.assembly(), animated: **true**)

}

}

* 1. AuthorizationViewController.swift

//

// ViewController.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** UIKit

/// Протокол контроллера авторизации

**protocol** AuthorizationViewProtocol: class {

/// Показать сообщение

/// - **Parameters**:

/// - title: Заголовок сообщения

/// - message: Текст сообщения

**func** showAlert(title: String, message: String)

/// Изменить видимость навигации

/// - **Parameters**:

/// - isHidden: Спрятан ли

/// - animated: Анимировано

**func** setNavigationBarHidden(**\_** isHidden: Bool, animated: Bool)

/// Начать анимацию индикатора загрузки

**func** startAnimatingActivityIndicator()

/// Завершить анимацию индикатора загрузки

**func** stopAnimatingActivityIndicator()

/// Получить контроллер навигации

**func** getNavigationController() -> UINavigationController?

}

/// Контроллер авторизации

**class** AuthorizationViewController: UIViewController {

/// Презентер авторизации

**var** presenter: AuthorizationPresenterProtocol!

/// Конфигуратор модуля

**var** configurator = AuthorizationConfigurator()

/// Картинка фона

**let** backgroundImage = UIImageView(image: UIImage(named: "auth\_background"))

/// Поле ввода почты

**let** emailTextField = TitledTextField.emailTextField()

/// Поле ввода почты

**let** passwordTextField = TitledTextField.passwordTextField()

/// Кнопка авторизации

**let** authorizationButton = NextButton()

/// Кнопка открытия модуля регистрации

**let** registrationButton = UIButton(frame: .zero)

/// Индикатор загрузки

**let** activityIndicator = UIActivityIndicatorView(frame: .zero)

/// Текст подзаголовка

**let** subtitleLabel = UILabel(frame: .zero)

/// Текст заголовка

**let** titleLabel = UILabel(frame: .zero)

/// 2 уровень подзаголовка

**let** smallSubtitleLabel = UILabel(frame: .zero)

/// Нижняя подпись

**let** footerLabel = UILabel(frame: .zero)

/// Контроллер загрузился

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

UITabBar.appearance().tintColor = UIColor(named: "dark\_blue")

configurator.configure(with: **self**)

view.backgroundColor = UIColor.systemBackground

navigationController?.navigationBar.prefersLargeTitles = **true**

configureSubviews()

addAllSubviews()

initConstraints()

presenter.viewDidLoad()

}

/// Контроллер отобразился

/// - **Parameter** animated: Анимировано

**override** **func** viewDidAppear(**\_** animated: Bool) {

**super**.viewDidAppear(animated)

**self**.navigationController?.tabBarItem.title = "Авторизация"

}

/// Конфигурировать вью

**private** **func** configureSubviews() {

configureBackgroundImage()

configureSubtitleLabel()

configureTitleLabel()

configureSmallSubtitleLabel()

configureEmailTextField()

configurePasswordTextField()

configureAuthorizationButton()

configureRegistrationButton()

configureFooterLabel()

configureActivityIndicator()

}

/// Конфигурировать картинку фона

**private** **func** configureBackgroundImage() {

backgroundImage.translatesAutoresizingMaskIntoConstraints = **false**;

}

/// Конфигурировать подзаголовка

**private** **func** configureSubtitleLabel() {

subtitleLabel.translatesAutoresizingMaskIntoConstraints = **false**;

subtitleLabel.font = UIFont.systemFont(ofSize: 20, weight: .light)

subtitleLabel.textColor = UIColor(named: "dark\_blue")

subtitleLabel.text = "Welcome to"

}

/// Конфигурировать заголовок

**private** **func** configureTitleLabel() {

titleLabel.translatesAutoresizingMaskIntoConstraints = **false**

titleLabel.textColor = UIColor(named: "dark\_blue")

titleLabel.font = UIFont.boldSystemFont(ofSize: 38)

titleLabel.text = "GeoFood"

}

/// Конфигурировать 2 уровень подзаголовка

**private** **func** configureSmallSubtitleLabel() {

smallSubtitleLabel.translatesAutoresizingMaskIntoConstraints = **false**

smallSubtitleLabel.font = UIFont.systemFont(ofSize: 14, weight: .medium)

smallSubtitleLabel.textColor = UIColor(named: "light\_blue")

smallSubtitleLabel.numberOfLines = 2

smallSubtitleLabel.text = """

Введите Email и пароль для

авторизации

"""

}

/// Конфигурировать поле ввода почты

**private** **func** configureEmailTextField() {

emailTextField.translatesAutoresizingMaskIntoConstraints = **false**

}

/// Конфигурировать поле ввода пароля

**private** **func** configurePasswordTextField() {

passwordTextField.translatesAutoresizingMaskIntoConstraints = **false**

}

/// Конфигурировать кнопку авторизации

**private** **func** configureAuthorizationButton() {

authorizationButton.translatesAutoresizingMaskIntoConstraints = **false**

authorizationButton.setTitle("Войти", for: .normal)

authorizationButton.addTarget(**self**, action: **#selector**(authorizationButtonTapped), for: .touchUpInside)

}

/// Конфигурировать нижний текст

**private** **func** configureFooterLabel() {

footerLabel.translatesAutoresizingMaskIntoConstraints = **false**;

footerLabel.textColor = UIColor(named: "dark\_blue")

footerLabel.font = UIFont.systemFont(ofSize: 16, weight: .light)

footerLabel.text = "Нет аккаунта?"

}

/// Конфигурировать кнопку перехода на модуль авторизации

**private** **func** configureRegistrationButton() {

registrationButton.translatesAutoresizingMaskIntoConstraints = **false**

**let** attrString = NSAttributedString(string: "Зарегистрироваться", attributes: [NSAttributedString.Key.underlineStyle: NSUnderlineStyle.thick.rawValue])

registrationButton.setAttributedTitle(attrString, for: .normal)

registrationButton.setTitleColor(UIColor(named: "dark\_blue"), for: .normal)

registrationButton.addTarget(**self**, action: **#selector**(registrationButtonTapped), for: .touchUpInside)

}

/// Конфигурировать индикатор загрузки

**private** **func** configureActivityIndicator() {

activityIndicator.translatesAutoresizingMaskIntoConstraints = **false**

activityIndicator.style = .large

activityIndicator.color = .darkGray

}

/// Добавить все вью

**private** **func** addAllSubviews() {

view.addSubview(backgroundImage)

view.addSubview(subtitleLabel)

view.addSubview(titleLabel)

view.addSubview(smallSubtitleLabel)

view.addSubview(emailTextField)

view.addSubview(passwordTextField)

view.addSubview(authorizationButton)

view.addSubview(footerLabel)

view.addSubview(footerLabel)

view.addSubview(registrationButton)

view.addSubview(activityIndicator)

}

/// Активировать констреинты

**private** **func** initConstraints() {

NSLayoutConstraint.activate([

backgroundImage.topAnchor.constraint(equalTo: view.topAnchor),

backgroundImage.leadingAnchor.constraint(equalTo: view.leadingAnchor),

backgroundImage.trailingAnchor.constraint(equalTo: view.trailingAnchor),

backgroundImage.heightAnchor.constraint(equalTo: view.heightAnchor, multiplier: 0.385),

subtitleLabel.topAnchor.constraint(equalTo: backgroundImage.bottomAnchor, constant: -10),

subtitleLabel.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 30),

subtitleLabel.trailingAnchor.constraint(lessThanOrEqualTo: view.trailingAnchor, constant: -100),

titleLabel.topAnchor.constraint(equalTo: subtitleLabel.bottomAnchor, constant: 4),

titleLabel.leadingAnchor.constraint(equalTo: subtitleLabel.leadingAnchor),

titleLabel.trailingAnchor.constraint(equalTo: subtitleLabel.trailingAnchor),

smallSubtitleLabel.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 5),

smallSubtitleLabel.leadingAnchor.constraint(equalTo: titleLabel.leadingAnchor),

smallSubtitleLabel.trailingAnchor.constraint(equalTo: titleLabel.trailingAnchor),

emailTextField.topAnchor.constraint(equalTo: smallSubtitleLabel.bottomAnchor, constant: 3),

emailTextField.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 30),

emailTextField.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -30),

emailTextField.heightAnchor.constraint(equalToConstant: 80),

passwordTextField.topAnchor.constraint(equalTo: emailTextField.bottomAnchor, constant: 7),

passwordTextField.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 30),

passwordTextField.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -30),

passwordTextField.heightAnchor.constraint(equalTo: emailTextField.heightAnchor),

authorizationButton.topAnchor.constraint(equalTo: passwordTextField.bottomAnchor, constant: 26),

authorizationButton.trailingAnchor.constraint(equalTo: passwordTextField.trailingAnchor),

authorizationButton.heightAnchor.constraint(equalToConstant: 50),

authorizationButton.widthAnchor.constraint(equalToConstant: 120),

footerLabel.centerXAnchor.constraint(equalTo: view.centerXAnchor),

footerLabel.topAnchor.constraint(equalTo: authorizationButton.bottomAnchor, constant: 25),

footerLabel.leadingAnchor.constraint(greaterThanOrEqualTo: view.leadingAnchor, constant: 30),

footerLabel.trailingAnchor.constraint(lessThanOrEqualTo: view.trailingAnchor, constant: -30),

registrationButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),

registrationButton.topAnchor.constraint(equalTo: footerLabel.bottomAnchor, constant: 1.54),

registrationButton.leadingAnchor.constraint(greaterThanOrEqualTo: view.leadingAnchor, constant: 30),

registrationButton.trailingAnchor.constraint(lessThanOrEqualTo: view.trailingAnchor, constant: -30),

activityIndicator.centerXAnchor.constraint(equalTo: view.centerXAnchor),

activityIndicator.centerYAnchor.constraint(equalTo: view.centerYAnchor),

])

}

/// Событие нажатия на кнопку авторизации

**@objc** **private** **func** authorizationButtonTapped() {

presenter.authorizationButtonTapped(withEmail: emailTextField.text, password: passwordTextField.text)

}

/// Событие нажатия на кнопку регистрации

**@objc** **private** **func** registrationButtonTapped() {

presenter.registrationButtonTapped()

}

}

**extension** AuthorizationViewController: AuthorizationViewProtocol {

/// Начать анимацию индикатора загрузки

**func** startAnimatingActivityIndicator() {

DispatchQueue.main.async {

**self**.activityIndicator.startAnimating()

}

}

/// Завершить анимацию индикатора загрузки

**func** stopAnimatingActivityIndicator() {

DispatchQueue.main.async {

**self**.activityIndicator.stopAnimating()

}

}

/// Показать сообщение

/// - **Parameters**:

/// - title: Заголовок сообщения

/// - message: Текст сообщения

**func** showAlert(title: String, message: String) {

**let** alert = UIAlertController(title: title, message: message, preferredStyle: .alert)

**let** alertAction = UIAlertAction(title: "Попробовать снова", style: .default, handler: **nil**)

alert.addAction(alertAction)

present(alert, animated: **true**, completion: **nil**)

}

/// Изменить видимость навигации

/// - **Parameters**:

/// - isHidden: Спрятан ли

/// - animated: Анимировано

**func** setNavigationBarHidden(**\_** isHidden: Bool, animated: Bool) {

navigationController?.setNavigationBarHidden(isHidden, animated: animated)

}

/// Получить контроллер навигации

**func** getNavigationController() -> UINavigationController? {

navigationController

}

}

* 1. AuthorizationPresenter.swift

//

// AuthorizationPresenter.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** Foundation

/// Протокол презентера авторизации

**protocol** AuthorizationPresenterProtocol: class {

/// Контроллер загрузился

**func** viewDidLoad()

/// Событие нажатия на кнопку авторизации

/// - **Parameters**:

/// - withEmail: Введенная почта

/// - password: Введенный пароль

**func** authorizationButtonTapped(withEmail: String, password: String)

/// Событие нажатия на кнопку регистрации

**func** registrationButtonTapped()

}

/// Презентер авторизации

**class** AuthorizationPresenter: AuthorizationPresenterProtocol {

/// Контроллер авторизации

**weak** **var** view: AuthorizationViewProtocol!

/// Интерактор авторизации

**var** interactor: AuthorizationInteractorProtocol!

/// Роутер авторизации

**var** router: AuthorizationRouterProtocol!

/// Конструктор

/// - **Parameter** view: Контроллер

**required** **init**(view: AuthorizationViewProtocol) {

**self**.view = view

}

/// Контроллер загрузился

**func** viewDidLoad() {

**if** interactor.isUserAuth {

**self**.router.openAccountView(animated: **false**)

}

}

/// Событие нажатия на кнопку авторизации

/// - **Parameters**:

/// - withEmail: Введенная почта

/// - password: Введенный пароль

**func** authorizationButtonTapped(withEmail: String, password: String) {

**if** /\*LoginEntryChecker.checkEmail(withEmail) && LoginEntryChecker.checkPassword(password)\*/ **true** {

view.startAnimatingActivityIndicator()

interactor.authUser(withEmail: withEmail, password: password)

} **else** {

view.showAlert(title: "Неверные данные", message: "Проверьте введенные email и пароль")

}

}

/// Событие нажатия на кнопку регистрации

**func** registrationButtonTapped() {

router.openRegistrationView()

}

}

**extension** AuthorizationPresenter: AuthorizationInteractorOutputProtocol {

/// Авторизация прошла неуспешно

**func** authorizationUnsuccessfully() {

DispatchQueue.main.async { [**unowned** **self**] **in**

view.stopAnimatingActivityIndicator()

view.showAlert(title: "Не удалось авторизоваться", message: "Проверьте введенные email и пароль")

}

}

/// Авторизация прошла успешно

**func** authorizationSuccessfully() {

DispatchQueue.main.async { [**unowned** **self**] **in**

view.stopAnimatingActivityIndicator()

router.openAccountView(animated: **true**)

}

}

}

* 1. AuthorizationInteractor.swift

//

// AuthorizationInteractor.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** Foundation

/// Входные методы интерактора

**protocol** AuthorizationInteractorProtocol: class {

/// Авторизовать пользователя

/// - **Parameters**:

/// - withEmail: Почта

/// - password: Пароль

**func** authUser(withEmail: String, password: String)

**var** isUserAuth: Bool { **get** }

}

/// Выходные методы интерактора

**protocol** AuthorizationInteractorOutputProtocol: class {

/// Авторизация прошла неуспешно

**func** authorizationUnsuccessfully()

/// Авторизация прошла успешно

**func** authorizationSuccessfully()

}

/// Интерактор авторизации

**class** AuthorizationInteractor: AuthorizationInteractorProtocol {

/// Авторизован ли пользователь

**var** isUserAuth: Bool {

**return** userService.isUserAuth

}

/// Презентер авторизации

**weak** **var** presenter: AuthorizationInteractorOutputProtocol!

/// Сервис пользователя

**var** userService = UserService.shared

/// Конструктор

/// - **Parameter** presenter: Презентер

**required** **init**(presenter: AuthorizationInteractorOutputProtocol) {

**self**.presenter = presenter

}

/// Авторизовать пользователя

/// - **Parameters**:

/// - withEmail: Почта

/// - password: Пароль

**func** authUser(withEmail: String, password: String) {

userService.authUser(with: LoginForm(login: withEmail, password: password)) { isSuccess **in**

**if** !isSuccess {

**self**.presenter.authorizationUnsuccessfully()

**return**

}

**self**.presenter.authorizationSuccessfully()

}

}

}

* 1. AuthorizationRouter.swift

//

// AuthorizationRouter.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** Foundation

/// Протокол роутера авторизации

**protocol** AuthorizationRouterProtocol: class {

/// Открыть модуль регистрации

**func** openRegistrationView()

/// Открыть модуль аккаунта

/// - **Parameter** animated: Анимировано

**func** openAccountView(animated: Bool)

}

/// Роутер авторизации

**class** AuthorizationRouter: AuthorizationRouterProtocol {

/// Контроллер авторизации

**var** view: AuthorizationViewProtocol!

/// Конструктор

/// - **Parameter** view: Контроллер авторизации

**required** **init**(view: AuthorizationViewController) {

**self**.view = view

}

/// Открыть модуль регистрации

**func** openRegistrationView() {

**let** registrationVc = RegistrationViewController()

view.getNavigationController()?.pushViewController(registrationVc, animated: **true**)

}

/// Открыть модуль аккаунта

/// - **Parameter** animated: Анимировано

**func** openAccountView(animated: Bool) {

**self**.view.getNavigationController()?.pushViewController(AccountConfigurator.assembly(), animated: animated)

}

}

* 1. AuthorizationConfigurator.swift

//

// AuthorizationConfigurator.swift

// GeoFood

//

// Created by Егор on 11.03.2021.

//

**import** Foundation

/// Протокол конфигуратора авторизации

**protocol** AuthorizationConfiguratorProtocol: class {

/// Конфигурировать модуль авторизации

/// - **Parameter** with: Контроллер авторизации

**func** configure(with: AuthorizationViewController)

}

/// Конфигуратор авторизации

**class** AuthorizationConfigurator: AuthorizationConfiguratorProtocol {

/// Конфигурировать модуль авторизации

/// - **Parameter** with: Контроллер авторизации

**func** configure(with view: AuthorizationViewController) {

**let** presenter = AuthorizationPresenter(view: view)

**let** interactor = AuthorizationInteractor(presenter: presenter)

**let** router = AuthorizationRouter(view: view)

presenter.interactor = interactor

presenter.router = router

view.presenter = presenter

}

}

1. Список литературы
   * 1. ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению. – М.: ИПК Издательство стандартов, 2001.
     2. Документация по UIKit [Электронный ресурс]// URL: https://developer.apple.com/documentation/uikit (Дата обращения: 5.05.2021, режим доступа: свободный)
     3. Документация по Swift [Электронный ресур]// URL: https://swift.org/documentation/ (Дата обращения: 5.05.2021, режим доступа свободный)
2. Лист регистрации изменений

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |