Министерство образования Республики Беларусь

Учреждение образования

«БелорусскиЙ государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Основы программирования информационных систем»

|  |
| --- |
| *К защите допустить*: |
| Руководитель курсовой работы  магистр технических наук, ассистент |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. М. Бруй  \_\_\_.\_\_\_. 20\_\_\_ |

**Пояснительная записка**

к курсовой работе

на тему

**Разработка IOS-приложения «Cryptocurrency»**

БГУИР КР 1-40 05 01-10 № 139 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент | (подпись студента) | А.Г. Батура |
|  |  | Курсовая работа представлена на проверку \_\_\_.\_\_\_. 20\_\_\_ |
|  |  | (подпись студента) |

Минск 2020

**РЕФЕРАТ**

БГУИР КР 1-40 05 01-10 № 139 ПЗ

Батура, А.Г. Разработка IOS-приложения «CryptoCurrency» / А.Г. Батура. – Минск: БГУИР, 2020. – 38 с.

Ключевые слова: мобильное приложение, криптовалюта.

Предмет: Объектно-ориентированное программирование, А.В.Михалькевич.

Объект: шаблоны проектирования, разработка пользовательского интерфейса и бизнес-логики.

Цель: разработка IOS-приложения «CryptoCurrency».

Методология проведения работы: разработка дизайна мобильного приложения, разработка логики приложение в соответствии с архитектурным паттерном MVC, выполнение HTTP запроса для получения актуального курса криптовалют.

Ссылка на проект: https://github.com/artyombatura/CryptoCurrency\_v2

Результаты работы: разработано мобильное приложение, полностью подходящее для просмотра курсов ряда криптовалют (количество ограничено биржей), просмотра графика дневного изменения цен (с возможностью выбора нацинальной валюты, поддерживаемой биржей), конвертации криптовалюты в, выбранную пользователем, национальную валюту.

Область применения результатов: предназначено для пользователей, которые нуждаются в простом и удобном приложении для просмотра актуальных цен на криптовалюту, а также в быстрой конвертации валют.

Содержание

[Введение 4](#_Toc40051518)

[1 Описание проект 5](#_Toc40051519)

[1.1 Клиентская часть 5](#_Toc40051520)

[1.2 Серверная часть 10](#_Toc40051521)

[2 Обоснование выбора технологий 13](#_Toc40051522)

[2.1 Объектно-ориентированное программирование 13](#_Toc40051523)

[2.2 Язык программирования Swift 16](#_Toc40051524)

[2.3 Технологии IOS разработки 17](#_Toc40051525)

[3 Инструментарий 18](#_Toc40051526)

[3.1 Xcode 18](#_Toc40051527)

[3.2 Система контроля версий GIT 18](#_Toc40051528)

[3.3 Postman 22](#_Toc40051529)

4 Архитектурный шаблон проектирования MVC………………………23

[5 Шаблоны проектирования 25](#_Toc40051531)

[Список использованных источников 31](#_Toc40051532)

[Приложение А (обязательное) 32](#_Toc40051533)

**ВВЕДЕНИЕ**

Приложение для быстрого показа актуальных цен на криптовалюты является важным инструментом для пользователя, заинтересованного этой сферой. Мобильное приложение, полностью подходит для просмотра курсов ряда криптовалют (количество ограничено только биржей), просмотра графика дневного изменения цен (с возможностью выбора нацинальной валюты, поддерживаемой биржей), конвертации криптовалюты в, выбранную пользователем, национальную валюту.

В курсовой работе поставлена цель разработать IOS-приложение, используя архитектуру MVС и сторонний API для получения актуальной информации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Проектирование пользовательского интерфейса.
2. Построение структуры приложения, соответствующей архитектуре MVС.
3. Построение запроса на получение погоды через внешний API.

**1 ОПИСАНИЕ ПРОЕКТА**

* 1. **Клиентская часть**

Клиентская часть — это мобильное приложение, которое показывает пользовательский интерфейс и функции для достижения пользовательских целей. В разрабатываемом приложении пользователь взаимодействует с текущим и недельным прогнозом погоды. Для создания пользовательского интерфейса в IOS используют сториборды.

Сториборд - замечательная функция Xcode, которая была представлена вместе с iOS 5 и сэкономила уйму времени для создания пользовательского интерфейса ваших приложений. Вы можете не знать, что конкретно делает ваше приложение, однако вы можете смотреть какие страницы оно имеет и как они взаимосвязаны между собой. В этом и есть сила сториборд.

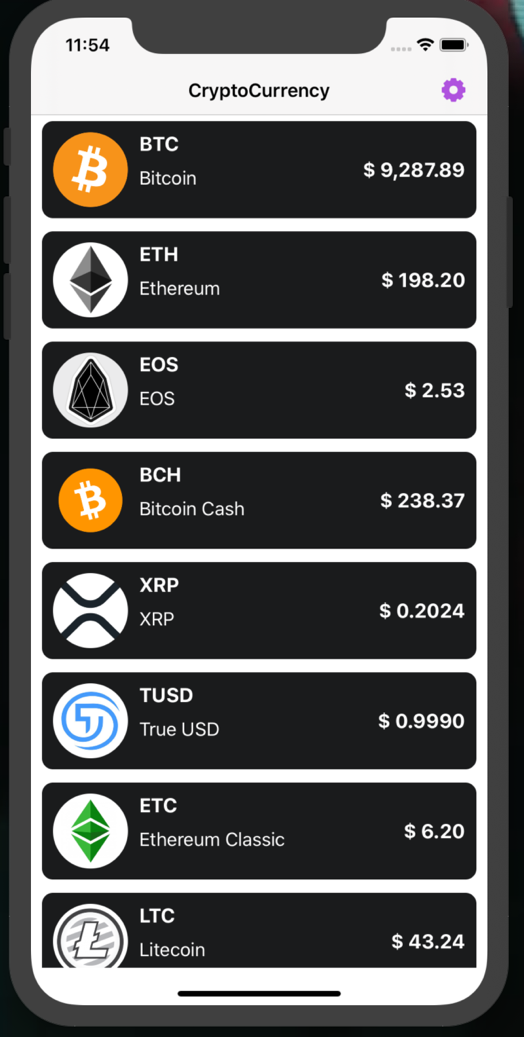
При создании приложения с большим количеством разных страниц сториборд позволяют существенно уменьшить количество склеивающего их кода, чтобы обеспечить переход от одной страницы к другой. В стародавние времена разработчикам приходилось создавать отдельные файлы (файлы nib или xib) интерфейса для каждого отдельного ViewController(страницы). Но теперь ваши приложения используют единственный сториборд, который включает в себя дизайн всех страниц и который определяет взаимодействие между ними.

Сториборды имеют ряд достоинств:

В сториборде очень удобно наблюдать общую картину приложения и взаимосвязи между его страницами. В сториборде можно отслеживать все что угодно, потому что общий дизайн приложения содержится в одном единственном файле, а не распределен между несколькими файлами nib.

* Сториборды могут описывать переходы между различными окнами. Эти переходы называются "segues", которые создаются путем соединения двух страниц прямо в сториборде. Благодаря этим segues, есть возможность писать меньше кода для пользовательского интерфейса.
* Сториборды облегчают работу с тибличными типами, с ячейками. Возможность создать таблицы практически полностью из сториборда, это именно то, что уменьшает в разы код, который бы пришлось написать.
* Сториборды упрощают работу при использовании автопозиционирования. Автопозиционирование - мощная функция, которая позволяет определять математические взаимоотношения между элементами, которые имеют определенные размеры и позиции, и так же упрощает работу по отображению вашего приложения на разных устройствах с разными расширениями экрана.

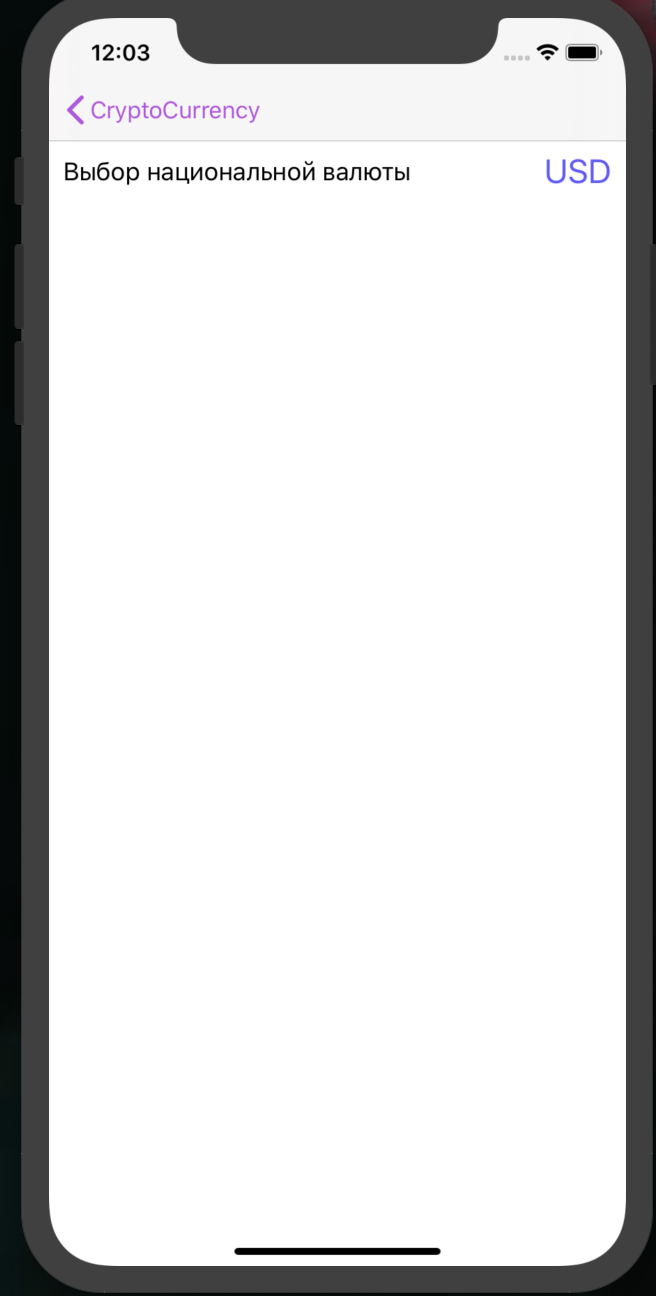
Запустив приложение, пользователь попадает на основной экран, где он видит основную информацию о криптовалюте: название, короткое название, цену и логотип.



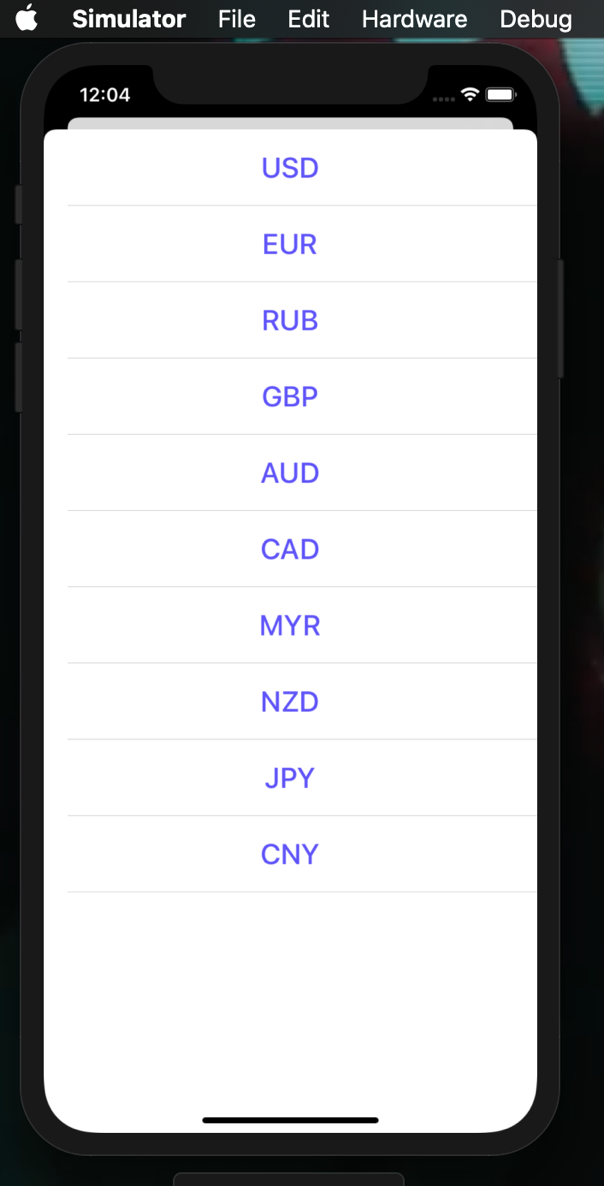
Основной элемент экрана это UITableView, элемент фреймворка UIKit, позволяющий осуществить вывод любой информации в виде таблице. На UITableView расположена UITableViewCell, которая полностью была переопределена мной, чтобы пользователь видел кастомизированный внешний вид ячейки, который улучшает восприятие пользователем.

При первом запуске приложение в качестве автоматически выбранной национальной валюты выступает Американский доллар (USD). В приложении предусмотрена возможность выбора национальной валюты, в которой будут отображаться цены на криптовалюту. Попасть на этот экран можно нажав на кнопку настроек в верхнем правом углу, открыв при этом экран пользовательских настроек, далее выбираем пункт «Выбор национальной валюты».

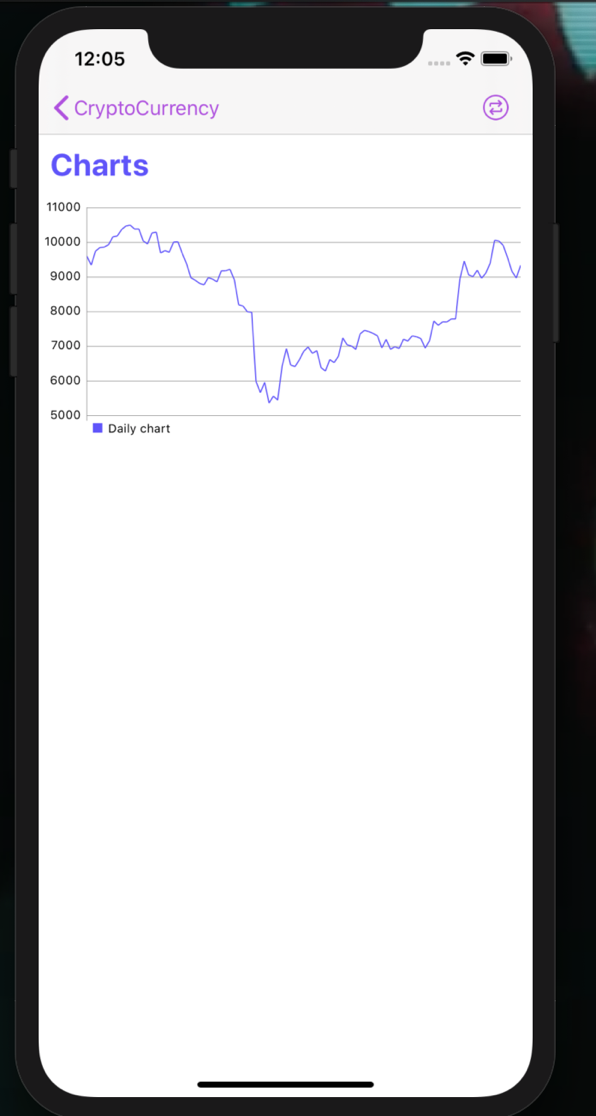




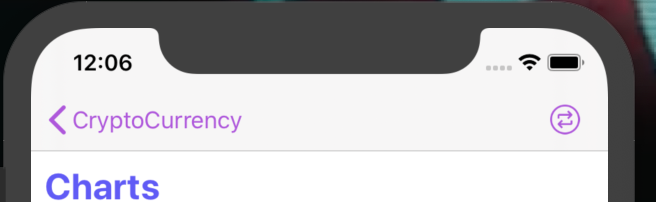
Попав на экран выбора национальной валюты, мы можем выбрать любую, доступную валюту на бирже, просто нажав на ячейку таблицу.



По нажатию на ячейку пользователь попадает на экран просмотра дневного графика изменения цен.



При нажатии на кнопку «Обмен валют» в правом верхнем углу, пользователь попадает на экран «Обмен валют», где может произвести обмен валют в обе стороны. Верхнее поле для ввода/отображения предназначено для ввода выбранной криптовалюты, нижнее для ввода/отображения выбранной национальной валюты.





* 1. **Серверная часть**

Мобильный разработчик не разрабатывает серверную часть напрямую сам. Однако у приложения должен быть реализован свой сетевой слой, который будет отвечать за отправление запросов и задание нужных параметров для этих запросов. Для этих целей в приложении используется сторонняя библиотека Alamofire.

Alamofire - это HTTP сетевая библиотека на Swift для iOS и Mac OS X. Она обеспечивает элегантный интерфейс сетевого стека Foundation от Apple, упрощающего ряд общих сетевых задач.

Я определил свое перечисление для создания запросов. Здесь хранится вся информация, нужная для создания запроса. Созданы вычисляемые переменные который и представляют собой все нужные параметры. Вычисляются они в зависимости от выбранного запроса.

Создана функция для создания запроса, которая использует все эти переменные как входные параметры, создает запрос на сервер и возвращает в колбэке результат, полученный с сервера, в формате JSON.

**import** UIKit

**import** Alamofire

**import** SwiftyJSON

**public** **let** baseUrl = "https://cryptocompare.com"

**public** **let** minApiBaseUrl = "https://min-api.cryptocompare.com/data"

**public** **let** internationalCurrencyCodesAPI = "https://openexchangerates.org/api/currencies.json"

**private** **let** apiKey = "247f4279d26a7e83c5256cc4a26f624006a0c8f5a220b6c0afbd2f0e09860c4f"

**enum** NetworkManager {

**case** GetAllCurrencies(tsym: String, page: Int, limit: Int)

**case** GetNationalCurrencyCodes

**case** GetChartData(tsym: String, fsym: String)

**private** **var** fullUrl: URL? {

**get** {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**if** **let** url = URL(string: minApiBaseUrl + "/top/totalvolfull") {

**return** url

}

**case** .GetNationalCurrencyCodes:

**return** URL(string: internationalCurrencyCodesAPI)

**case** .GetChartData(**\_**, **\_**):

**return** URL(string: minApiBaseUrl + "/v2/histoday")

}

**return** **nil**

}

}

**private** **var** headers: HTTPHeaders {

**get** {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**return** HTTPHeaders(["apiKey":apiKey])

**case** .GetNationalCurrencyCodes:

**return** HTTPHeaders([:])

**case** .GetChartData(**\_**, **\_**):

**return** HTTPHeaders([:])

}

}

}

**private** **var** parameters: Parameters {

**switch** **self** {

**case** .GetAllCurrencies(**let** tsym, **let** page, **let** limit):

**return** ["tsym": tsym, "limit": limit, "page": page]

**case** .GetNationalCurrencyCodes:

**return** [:]

**case** .GetChartData(**let** tsym, **let** fsym):

**return** ["tsym": tsym, "fsym": fsym, "limit": "100"]

}

}

**private** **var** method: HTTPMethod {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**return** .get

**case** .GetNationalCurrencyCodes:

**return** .get

**case** .GetChartData(**\_**, **\_**):

**return** .get

}

}

**public** **func** createNetworkSession(completion: **@escaping** (JSON)->()) {

**guard** **let** fullUrlUnwrapped = fullUrl **else** { **return** }

AF.request(fullUrlUnwrapped, method: method, parameters: parameters, encoding: URLEncoding.default, headers: headers, interceptor: **nil**, requestModifier: **nil**).responseJSON { (response) **in**

**switch** response.result {

**case** .success(**let** json):

print("SUCCESS")

**let** json = JSON(arrayLiteral: json)

completion(json)

**case** .failure(**let** error):

print("ERROR: \(error)")

}

}

}

}

Пример создания запроса и обработки полученных данных (получение Coins):

//MARK: **- FetchCoins**

**extension** CurrenciesViewController {

**private** **func** fetchCoins() {

**var** tsym = "USD"

**if** **let** tsymFromUserConfig = userdefaults.value(forKey: getInternationalCurrencyCode) **as**? String {

tsym = tsymFromUserConfig

}

**let** networkSession = NetworkManager.GetAllCurrencies(tsym: tsym, page: pageToDownload, limit: 10)

networkSession.createNetworkSession { (json) **in**

**let** coins = Coin.getAllCoinsArray(fromJSON: json)

coins.forEach { (coin) **in**

**self**.coins.append(coin)

}

**self**.currenciesTableView.reloadData()

}

}

}

# 

# 2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

**2.1** **Объектно-ориентированное программирование**

Объектно-ориентированное программирование (в дальнейшем ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В центре ООП находится понятие объекта.

Объект — это сущность, экземпляр класса, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования. Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм, то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения не значимые. Соответственно, абстракция — это набор всех таких характеристик.

В контексте ООП абстракция — это обобщение данных и поведения для типа, находящегося выше текущего класса по иерархии. Перемещая переменные или методы из подкласса в супер класс, вы обобщаете их. Но язык добавляет также понятия абстрактных классов и абстрактных методов.

Абстрактный класс является классом, для которого нельзя создать экземпляр.

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя. Инкапсуляция позволит скрыть детали реализации, и открыть только то что необходимо в последующем использовании. Другими словами, инкапсуляция – это механизм контроля доступа.

Цель инкапсуляции – уйти от зависимости внешнего интерфейса класса (то, что могут использовать другие классы) от реализации. Чтобы малейшее изменение в классе не влекло за собой изменение внешнего поведения класса.

Существует 4 вида модификаторов доступа: public, protected, private и default.

Public – уровень предполагает доступ к компоненту с этим модификатором из экземпляра любого класса и любого пакета.

Protected – уровень предполагает доступ к компоненту с этим модификатором из экземпляров родного класса и классов-потомков, независимо от того, в каком пакете они находятся.

Default – уровень предполагает доступ к компоненту с этим модификатором из экземпляров любых классов, находящихся в одном пакете с этим классом.

Private – уровень предполагает доступ к компоненту с этим модификатором только из этого класса.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом.

Наследование — это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов. Применение иерархии классов делает управляемыми большие потоки информации.

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

В более общем смысле, концепцией полиморфизма является идея “один интерфейс, множество методов“. Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор.

Говоря про ООП нужно сказать про принципы SOLID, каждое грамотное приложение соответствует этим принципам.

SOLID — это аббревиатура пяти основных принципов проектирования в объектно-ориентированном программировании — Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion. В переводе на русский: принципы единственной ответственности, открытости / закрытости, подстановки Барбары Лисков, разделения интерфейса и инверсии зависимостей)

Аббревиатура SOLID была предложена Робертом Мартином, автором нескольких книг, широко известных в сообществе разработчиков. Эти принципы позволяют строить на базе ООП масштабируемые и сопровождаемые программные продукты с понятной бизнес-логикой.

**Расшифровка:**

* Single responsibility — принцип единственной ответственности
* Open-closed — принцип открытости / закрытости
* Liskov substitution — принцип подстановки Барбары Лисков
* Interface segregation — принцип разделения интерфейса
* Dependency inversion — принцип инверсии зависимостей

**Принцип единственной обязанности / ответственности** (single responsibility principle / SRP) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности

**Принцип открытости / закрытости** (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.

**Принцип подстановки Барбары Лисков** (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом».

**Принцип разделения интерфейса** (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

**Принцип инверсии зависимостей** (dependency inversion principle / DIP) — модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций.

**2.2 Язык программирования Swift**

Swift – это новый язык программирования для разработки iOS и OS X приложений, который сочетает в себе все лучшее от C и Objective-C, но лишен ограничений, накладываемых в угоду совместимости с C. В Swift используются паттерны безопасного программирования и добавлены современные функции, превращающие создание приложения в простой, более гибкий и увлекательный процесс. Swift – это возможность заново представить себе, как разрабатываются приложения.

Swift разрабатывался несколько лет. Основой нового языка программирования послужили существующие компилятор, отладчик и фреймворки. Процесс управления памятью упрощен с помощью механизма автоматического подсчета ссылок – Automatic Reference Counting (ARC). Фреймворки также подверглись серьезной модернизации. Objective-C начал поддерживать блоки, литералы и модули – все это создало благоприятные условия для внедрения современных технологий. Именно эта подготовительная работа послужила фундаментом для нового языка программирования, который будет применяться для разработки будущих программных продуктов для Apple.

Разработчикам Objective-C Swift покажется знакомым. Он сочетает в себе читабельность именованных параметров и мощь динамической объектной модели Objective-C. Он открывает доступ к уже существующим фреймворкам Cocoa и совместим с кодом, написанным на Objective-C. Построенный на этой общей основе язык предлагает множество новых возможностей и унифицирует процедурные и объектно-ориентированные аспекты языка программирования.

Swift не отпугнет и начинающих программистов. Это первый мощный язык программирования, такой же понятный и увлекательный, как скриптовый язык. Он поддерживает так называемые playground-ы, которые позволяют программистам экспериментировать с кодом, видя результат в режиме реального времени без необходимости компилировать и запускать приложение.

Swift вобрал в себя все лучшее от современных языков и разработан с учетом обширного опыта компании Apple. Компилятор спроектирован таким образом, чтобы было легко разработать первое приложение «hello, world!», и даже целую операционную систему. Все это делает Swift важным инструментом для разработчиков и для самой компании Apple.

Swift – это новый фантастический способ создавать приложения для iOS и OS X, с каждым годом добавляется новый функционал и предоставляет новые возможности для разработчиков.

**2.3 Технологии IOS разработки**

Foundation определяет базовый уровень функциональности, который требуется практически для всех приложений. Он предоставляет примитивные классы и вводит несколько парадигм, которые определяют функциональные возможности, не предоставляемые языком или средой выполнения. Он разработан с учетом следующих целей:

- Предоставить небольшой набор базовых служебных классов.

- Упростите разработку программного обеспечения, введя согласованные соглашения.

- Поддержка интернационализации и локализации, чтобы сделать программное обеспечение доступным для пользователей по всему миру.

- Обеспечить уровень независимости от ОС, чтобы улучшить переносимость.

В то время как платформа Foundation определяет классы, протоколы и функции для разработки под iOS и OS X, платформа UIKit предназначена исключительно для разработки под iOS. Это эквивалент Application Kit или платформы AppKit для разработки под OS X.

UIKit как и Foundation определяет классы, протоколы, функции, типы данных и константы. Он также добавляет дополнительные функциональные возможности к различным базовым классам, таким как NSObject, NSString и NSValue, благодаря использованию категорий Objective-C.

# 3 ИНСТРУМЕНТАРИЙ

**3.1 Xcode**

Xcode — интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple. Первая версия выпущена в 2003 году. Стабильные версии распространяются бесплатно через Mac App Store. Зарегистрированные разработчики также имеют доступ к бета-сборкам через сайт Apple Developer.

Xcode позволяет работать быстрее. В редакторе исходного кода есть функция автозаполнения, а цвета улучшают восприятие, независимо от языка программирования, на котором вы пишете. Мощные компиляторы и инструменты для сборки идентифицируют баги, графический отладчик быстро находит практически любую проблему и помогает ее решить. Есть возможность на ходу менять значения переменных и устанавливать точки останова, на которых будет прерываться программа. Приложение позволяет обрабатывать любые типы данных — от изображений до JSON и PLIST. Есть возможность добавить в проект модели машинного обучения — они мгновенно станут доступны в коде.

**3.2 Система контроля версий GIT**

Git — это система управления версиями. У Git две основных задачи: первая — хранить информацию о всех изменениях в коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом. (рис. 13)

Репозиторий Git — это место, где хранится код и вся информация о его изменениях. Репозитории могут находиться на компьютере, на компьютерах коллег и на удалённом сервере.

Главные преимущества систем управлениями версиями:

* Доступ к коду. Исходники кода хранятся в удаленном репозитории (хранилище данных), куда обращаются разработчики, чтобы забрать актуальную версию файлов или внести изменения. Так выстраивается командная разработка.
* Логирование изменений в коде. Отслеживание коммитов (внесений изменений в код), помогает найти кто, что и когда менял, решить конфликты при модифицировании одних и тех же файлов, откатиться на любое предыдущее состояние.
* Ветвление разработки. Программисты параллельно ведут разработку нового функционала в отдельных ветках, не затрагивая работоспособности старого.
* Поддержка версионности продуктов. При выпуске обновлений программных продуктов, мы обозначаем релизные версии, например, с помощью тегов, чтобы зафиксировать их в этом состоянии, для дебага или ретроспективы.

Главные преимущества Git по сравнению с другими системами управления версиями:

* Распределённая разработка;
* Транзакционный подход в управлении пакетами;
* Простота управления исходным кодом;
* Простота и удобство создания патчей;
* Интеграция в VCS апстрима, в том числе и по истории;
* Прозрачная сборка как локально, так и на сервере (в последнем случае нужно лишь создать подписанный тег и задать команду на сборку по ssh);
* Быстрое бэкпортирование;
* Малый трафик при постановке на сборку в репозиторий;
* Разнообразные проверки собираемых пакетов.

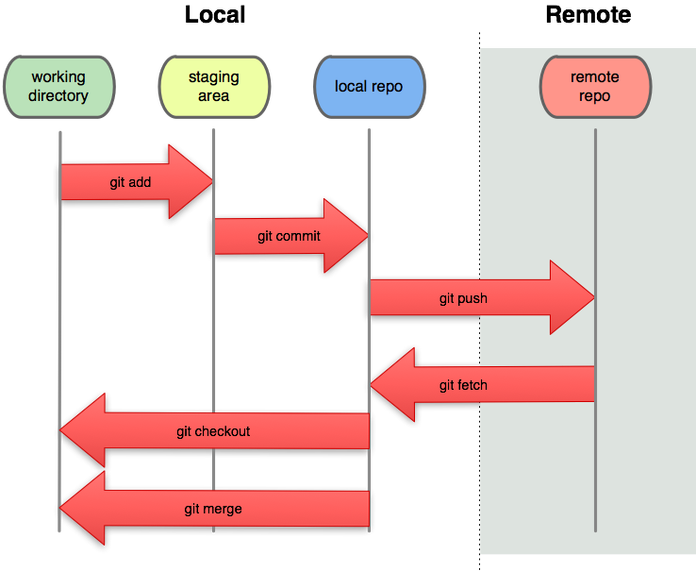


Рисунок 13 – принцип работы с Git

Gitflow является методологией работы с Git. Это значит, она определяет, какие ветки нужно создать и как производить их слияние. Набор инструментов git-flow нужно установить отдельно. Пакеты команд git-flow доступны во многих операционных системах. Для системы OSX можно выполнить brew install git-flow. После установки git-flow необходимо выполнить команду git flow init. Git-flow является оберткой для Git. Команда git flow init является расширением стандартной команды git init и ничего не меняет в репозитории, кроме того, что создает ветки.

На этой схеме показан принцип того, как работает git-flow (рис. 14):

Изображение выглядит как часы

Автоматически созданное описание

Рисунок 14 – Базовое ветвление

Вместо использования одной ветки master, в этой модели используется две ветки для записи истории проекта. В ветке master хранится официальная история релиза, а ветка develop служит в качестве интеграционной ветки для новых функций. Также, удобно тегировать все коммиты в ветке master номером версии.

Изображение выглядит как компьютер

Автоматически созданное описание

Рисунок 15 – Продвинутое ветвление

Каждая новая функциональность должна разрабатываться в отдельной ветке, которую можно отправлять в центральный репозиторий для создания резервной копии/для совместной работы команды (рис. 15). Ветки функций создаются не на основе master, a на основе develop. Когда работа над новой функциональностью завершена, она вливается назад в develop. Новый код не должен отправляться напрямую в master.

Изображение выглядит как компьютер

Автоматически созданное описание

Рисунок 16 – Ветвление на реальном проекте

Когда в ветку develop уже слито достаточно нового кода для релиза (или подходит установленная дата предрелиза), от ветки develop создается ветка release. Создание данной ветки означает начало следующего цикла релиза, в ходе которой новая функциональность уже не добавляется, а производится только отладка багов, создание документации и решение других задач, связанных с релизом. Когда все готово, ветка release сливается в master, и ей присваивается тег с версией. Кроме этого, она должна быть также слита обратно в ветку develop, в которой с момента создания ветки релиза могли добавляться изменения с момента создания ветки релиза.

Ветки hotfix используются для быстрого внесения исправлений в рабочую версию кода (рис. 16). Ветки hotfix очень похожи на ветки release и feature, за исключением того, что они созданы от master, а не от develop. Это единственная ветка, которая должна быть создана непосредственно от master. Как только исправление завершено, ветка hotfix должна быть объединена как с master, так и с develop (или с веткой текущего релиза), а master должен быть помечен обновленным номером версии.

Наличие специальной ветки для исправления ошибок позволяет команде решать проблемы, не прерывая остальную часть рабочего процесса и не ожидая следующего цикла подготовки к релизу. Можно говорить о ветках hotfix как об особых ветках relese, которые работают напрямую с master.

**3.3 Postman**

Основное предназначение Postman — создание коллекций с запросами к API. Любой разработчик или тестировщик, открыв коллекцию, может с лёгкостью разобраться в работе сервиса. Postman позволяет проектировать дизайн API и создавать на его основе Mock-сервер. Реализацию сервера и клиента можно запустить одновременно. Тестировщики могут писать тесты и производить автоматизированное тестирование прямо из Postman. А инструменты для автоматического документирования по описаниям из коллекций сэкономят время на выполнение других задач. Авторы приложения предусмотрели возможность создания коллекций для мониторинга сервисов.

Коллекция — отправная точка для нового API. Можно рассматривать коллекцию, как файл проекта. Коллекция объединяет в себе все связанные запросы. Обычно API описывается в одной коллекции, но при необоходимости нет никаких ограничений сделать по-другому. Коллекция может иметь свои скрипты и переменные.

Запрос — основная составляющая коллекции. Запрос создается в конструкторе. Конструктор запросов это главное пространство, с которым придётся работать. Postman умеет выполнять запросы с помощью всех стандартных HTTP методов, все параметры запроса под контролем разработчика. Есть возможность поменять или добавить необходимые заголовки, cookie, и тело запроса. У запроса есть свои скрипты. Вкладки "Pre-request Script" и "Tests" среди параметров запроса. Они позволяют добавить скрипты перед выполнением запроса и после. Именно эти две возможности делают Postman мощным инструментом помогающим при разработке и тестировании.

1. **АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC**

**Model-View-Controller** (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, [пользовательского интерфейса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

* ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении [бизнес-логики](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) (*модели*) от её визуализации (*представления*, *вида*). За счёт такого разделения повышается возможность [повторного использования кода](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0). Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной *модели* можно присоединить несколько *видов*, при этом не затрагивая реализацию *модели*. Например, некоторые данные могут быть одновременно представлены в виде [электронной таблицы](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0), [гистограммы](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D1%81%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) и [круговой диаграммы](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D1%83%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0);

Не затрагивая реализацию *видов*, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) — для этого достаточно использовать другой *контроллер*;

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), либо разрабатывают [бизнес-логику](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0). Поэтому возможно добиться того, что программисты, занимающиеся разработкой [бизнес-логики](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) (*модели*), вообще не будут осведомлены о том, какое *представление* будет использоваться.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента:

**Модель**

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает, как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «[наблюдателей](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

**Представление**

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

**Контроллер**

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

**Функциональные возможности и расхождения**

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями.

Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Также не указано, где должна находиться проверка введённых пользователем данных. Простая валидация может встречаться даже в представлении, но чаще они встречаются в контроллере или модели.

Интернационализация и форматирование данных также не имеет четких указаний по расположению.

Схема алгоритма, диаграмма последовательности и диаграмма состояний приведены в Приложении пояснительной записки.

**5 ШАБЛОНЫ ПРОЕКТИРОВАНИЯ**

Шаблон проектирования, или паттерн, в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования, в рамках некоторого часто возникающего контекста.

Шаблоны бывают следующих трех видов:

1. Порождающие;
2. [Структурные](https://tproger.ru/translations/design-patterns-simple-words-2/);
3. [Поведенческие](https://tproger.ru/translations/design-patterns-simple-words-3/).

Если говорить простыми словами, то это шаблоны, которые предназначены для создания экземпляра объекта или группы связанных объектов

Порождающие шаблоны — шаблоны проектирования, которые имеют дело с процессом создания объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту. Эти шаблоны оказываются важны, когда система больше зависит от композиции объектов, чем от наследования классов. Получается так, что основной упор делается не на жестком кодировании фиксированного набора поведений, а на определении небольшого набора фундаментальных поведений, с помощью композиции которых можно получать любое число более сложных. Таким образом, для создания объектов с конкретным поведением требуется нечто большее, чем простое инстанцирование класса.

Порождающие шаблоны инкапсулируют знания о конкретных классах, которые применяются в системе, то есть скрывают детали того, как эти классы создаются и стыкуются. Единственная информация об объектах, известная системе, — это их интерфейсы, определенные с помощью абстрактных классов. Следовательно, порождающие шаблоны обеспечивают большую гибкость при решении вопроса о том, что создается, кто это создает, как и когда. Можно собрать систему из «готовых» объектов с самой различной структурой и функциональностью статически (на этапе компиляции) или динамически (во время выполнения).

Фабричный метод (Factory Method) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам. Фабричный метод стоит применять в таких случаях:

* Когда заранее неизвестно, объекты каких типов необходимо создавать;
* Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать;
* Когда создание новых объектов необходимо делегировать из базового класса классам наследникам.

Паттерн "Абстрактная фабрика" (Abstract Factory) предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов. Данный паттерн стоит применять в таких случаях:

* Когда система не должна зависеть от способа создания и компоновки новых объектов;
* Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными.

Одиночка (Singleton, Синглтон) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа. Этот паттерн стоит использовать когда необходимо, чтобы для класса существовал только один экземпляр. Синглтон позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие синглтона от глобальных переменных.

Поведенческие шаблоны — [шаблоны проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов.

В поведенческих шаблонах уровня класса используется наследование, чтобы определить поведение для различных классов. В поведенческих шаблонах уровня объекта используется композиция. Некоторые из них описывают, как с помощью кооперации несколько равноправных объектов работают над заданием, которое они не могут выполнить по отдельности. Здесь важно то, как объекты получают информацию о существовании друг друга. Объекты-коллеги могут хранить ссылки друг на друга, но это усиливает степень связанности системы. При высокой связанности каждому объекту пришлось бы иметь информацию обо всех остальных. Некоторые из шаблонов решают эту проблему.

Паттерн Стратегия (Strategy) представляет шаблон проектирования, который определяет набор алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм. Этот паттерн стоит использовать в таких случаях:

* Когда есть несколько родственных классов, которые отличаются поведением. Можно задать один основной класс, а разные варианты поведения вынести в отдельные классы и при необходимости их применять;
* Когда необходимо обеспечить выбор из нескольких вариантов алгоритмов, которые можно легко менять в зависимости от условий;
* Когда необходимо менять поведение объектов на стадии выполнения программы;
* Когда класс, применяющий определенную функциональность, ничего не должен знать о ее реализации.

Паттерн "Наблюдатель" (Observer) представляет поведенческий шаблон проектирования, который использует отношение "один ко многим". В этом отношении есть один наблюдаемый объект и множество наблюдателей. И при изменении наблюдаемого объекта автоматически происходит оповещение всех наблюдателей.

Данный паттерн еще называют Publisher-Subscriber (издатель-подписчик), поскольку отношения издателя и подписчиков характеризуют действие данного паттерна: подписчики подписываются email-рассылку определенного сайта. Сайт-издатель с помощью email-рассылки уведомляет всех подписчиков о изменениях. А подписчики получают изменения и производят определенные действия: могут зайти на сайт, могут проигнорировать уведомления и т.д.

Паттерн стоит использовать в таких случаях:

* Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях;
* Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друга от друга;
* Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и процессе работы программы может изменяться.

Структурные паттерны рассматривают вопросы о компоновке системы на основе классов и объектов. При этом могут использоваться следующие механизмы:

* Наследование, когда базовый класс определяет интерфейс, а подклассы - реализацию. Структуры на основе наследования получаются статичными;
* Композиция, когда структуры строятся путем объединения объектов некоторых классов. Композиция позволяет получать структуры, которые можно изменять во время выполнения.

Фасад (Facade) представляет шаблон проектирования, который позволяет скрыть сложность системы с помощью предоставления упрощенного интерфейса для взаимодействия с ней. Фасад стоит использовать в этих случаях:

* Когда имеется сложная система, и необходимо упростить с ней работу. Фасад позволит определить одну точку взаимодействия между клиентом и системой;
* Когда надо уменьшить количество зависимостей между клиентом и сложной системой. Фасадные объекты позволяют отделить, изолировать компоненты системы от клиента и развивать и работать с ними независимо;
* Когда нужно определить подсистемы компонентов в сложной системе. Создание фасадов для компонентов каждой отдельной подсистемы позволит упростить взаимодействие между ними и повысить их независимость друг от друга.

Декоратор (Decorator) представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Для определения нового функционала в классах нередко используется наследование. Декораторы же предоставляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов. Стоит использовать когда надо динамически добавлять к объекту новые функциональные возможности. При этом данные возможности могут быть сняты с объекта

Применение наследования неприемлимо, если нам надо определить множество различных функциональностей и для каждой функциональности наследовать отдельный класс, то структура классов может очень сильно разрастись. Еще больше она может разрастись, если нам необходимо создать классы, реализующие все возможные сочетания добавляемых функциональностей.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были изучены материалы, связанные с разработкой IOS-приложении на архитектурном паттерне MVC. В результате курсовой работы было написано приложение, позволяющее просматривать актуальную информацию о курсе. Была углублено изучена и применена на практике объектно-ориентированная технология программирования и архитектурный паттерн MVC. Код программы написан на языке Swift версии 5.1. В результате работы с программой Sketch был спроектирован пользовательский интерфейс. Разработанное приложение доступно и может использоваться на практике конечным пользователем.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] ERUD.BY [Электронный ресурс]. – Режим доступа: http://erud.by/object\_orient\_program

[2] HABR [Электронный ресурс]. – Режим доступа: https://habr.com/

[3] APPLE [Электронный ресурс]. – Режим доступа: https://apple.com/

[4] MEDIUM [Электронный ресурс]. – Режим доступа: https://medium.com/

[5] SWIFTBOOK [Электронный ресурс]. – Режим доступа: https://swiftbook.com /

# ПРИЛОЖЕНИЕ А

Листинг кода

//

// CurrenciesViewController.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/9/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** UIKit

**import** SDWebImage

**protocol** CurrenciesViewControllerDelegate {

**func** updateCurrenciesAfterDismiss()

}

**class** CurrenciesViewController: UIViewController, CurrenciesViewControllerDelegate {

//MARK: **- Outlets**

**@IBOutlet** **weak** **var** currenciesTableView: UITableView!

**@IBOutlet** **weak** **var** activityIndicator: UIActivityIndicatorView!

//MARK: **- Variables**

**private** **var** coins: [Coin] = [Coin]()

**private** **var** pageToDownload: Int = 0

**let** userdefaults = UserDefaults.standard

//MARK: **- Lifecycle**

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

**super**.viewDidLoad()

navigationItem.title = "CryptoCurrency"

navigationItem.largeTitleDisplayMode = .always

navigationItem.rightBarButtonItem = UIBarButtonItem(image: UIImage(named: "SettingsSymbol")!, style: .plain, target: **self**, action: **#selector**(openSettingsScreen))

navigationController?.navigationBar.tintColor = UIColor.systemPurple

currenciesTableView.delegate = **self**

currenciesTableView.dataSource = **self**

currenciesTableView.tableFooterView = UIView(frame: .zero)

fetchCoins()

}

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

**super**.viewWillAppear(**true**)

}

//MARK: **- Actions**

**@objc** **func** openSettingsScreen() {

**self**.coins = [Coin]()

**self**.pageToDownload = 0

**let** settingsVC = SettingsViewController.storyboardInstance

settingsVC.currenciesVCDelegate = **self**

navigationController?.pushViewController(settingsVC, animated: **true**)

}

//MARK: **- CurrenciesViewControllerDelegate**

**func** updateCurrenciesAfterDismiss() {

fetchCoins()

}

}

//MARK: **- Storyboard Instance**

**extension** CurrenciesViewController {

**static** **unowned** **var** storyboardInstance: CurrenciesViewController {

**return** UIStoryboard(name: String(describing: **self**), bundle: **nil**).instantiateInitialViewController() **as**! CurrenciesViewController

}

}

//MARK: **- UITableViewDelegate & UITableViewDataSource**

**extension** CurrenciesViewController: UITableViewDelegate, UITableViewDataSource {

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

**return** 100.0

}

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

**let** count = coins.count

**if** count == 0 {

activityIndicator.startAnimating()

activityIndicator.isHidden = **false**

currenciesTableView.isHidden = **true**

} **else** {

activityIndicator.stopAnimating()

activityIndicator.isHidden = **true**

currenciesTableView.isHidden = **false**

}

**return** count

}

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

**guard** **let** cell = currenciesTableView.dequeueReusableCell(withIdentifier: CURRENCY\_CELL\_IDENTIFIER, for: indexPath) **as**? CurrencyCell **else** { **return** UITableViewCell() }

cell.backgroundGradientView.backgroundColor = cellTopGradientColor

cell.shortNameLabel.text = coins[indexPath.row].shortName

cell.fullNameLabel.text = coins[indexPath.row].fullName

cell.priceLabel.text = coins[indexPath.row].price

cell.logoImageView.layer.cornerRadius = cell.logoImageView.frame.height / 2

cell.logoImageView.sd\_setImage(with: URL(string: baseUrl + coins[indexPath.row].imgURL), completed: **nil**)

**return** cell

}

**func** tableView(**\_** tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

currenciesTableView.deselectRow(at: indexPath, animated: **true**)

**let** chartsVC = ChartsViewController.storyboardInstance

chartsVC.coin = coins[indexPath.row]

navigationController?.pushViewController(chartsVC, animated: **true**)

}

**func** tableView(**\_** tableView: UITableView, willDisplay cell: UITableViewCell, forRowAt indexPath: IndexPath) {

**if** indexPath.row == Int(**self**.coins.count / 2) {

pageToDownload += 1

**for** coin **in** **self**.coins {

print(coin.imgURL)

}

**self**.fetchCoins()

}

}

}

//MARK: **- FetchCoins**

**extension** CurrenciesViewController {

**private** **func** fetchCoins() {

**var** tsym = "USD"

**if** **let** tsymFromUserConfig = userdefaults.value(forKey: getInternationalCurrencyCode) **as**? String {

tsym = tsymFromUserConfig

}

**let** networkSession = NetworkManager.GetAllCurrencies(tsym: tsym, page: pageToDownload, limit: 10)

networkSession.createNetworkSession { (json) **in**

**let** coins = Coin.getAllCoinsArray(fromJSON: json)

coins.forEach { (coin) **in**

**self**.coins.append(coin)

}

**self**.currenciesTableView.reloadData()

}

}

}

//

// SettingsViewController.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/11/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** UIKit

**import** Alamofire

**import** SwiftyJSON

**protocol** SettingsViewControllerDelegate {

**func** controllerWillAppearAfterDismiss()

}

**class** SettingsViewController: UIViewController, SettingsViewControllerDelegate {

//MARK: **- Outlets**

**@IBOutlet** **weak** **var** settingsTableView: UITableView!

//MARK: **- Variables**

**var** settingsCells: [String] = ["Выбор национальной валюты"]

**let** userdefaults = UserDefaults.standard

**var** currenciesVCDelegate: CurrenciesViewControllerDelegate!

//MARK: **- Lifecycle**

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

**super**.viewDidLoad()

settingsTableView.delegate = **self**

settingsTableView.dataSource = **self**

settingsTableView.tableFooterView = UIView(frame: .zero)

}

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

**super**.viewWillAppear(**true**)

settingsTableView.reloadData()

}

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

**super**.viewWillDisappear(**true**)

currenciesVCDelegate.updateCurrenciesAfterDismiss()

}

//MARK: **- Delegate actions**

**func** controllerWillAppearAfterDismiss() {

settingsTableView.reloadData()

}

}

//MARK: **- Storyboard Instance**

**extension** SettingsViewController {

**static** **unowned** **var** storyboardInstance: SettingsViewController {

**return** UIStoryboard(name: String(describing: **self**), bundle: **nil**).instantiateInitialViewController() **as**! SettingsViewController

}

}

//MARK: **- TableViewDelegate & DataSource**

**extension** SettingsViewController: UITableViewDelegate, UITableViewDataSource {

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

**return** 44.0

}

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

**return** settingsCells.count

}

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

**guard** **let** cell = settingsTableView.dequeueReusableCell(withIdentifier: SETTING\_CELL\_IDENTIFIER, for: indexPath) **as**? SettingCell **else** { **return** UITableViewCell() }

cell.settingNameLabel.text = settingsCells[indexPath.row]

**if** **let** tsym = userdefaults.value(forKey: getInternationalCurrencyCode) **as**? String {

cell.tsymLabel.text = tsym

}

**return** cell

}

**func** tableView(**\_** tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

settingsTableView.deselectRow(at: indexPath, animated: **true**)

**let** nextVC = TSYMPickerViewController.storyboardInstance

nextVC.settingsVCDelegate = **self**

navigationController?.present(nextVC, animated: **true**, completion: **nil**)

}

}

//

// TSYMPickerViewController.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/12/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** UIKit

**class** TSYMPickerViewController: UIViewController {

//MARK: **- Outlets**

**@IBOutlet** **weak** **var** tsymTableView: UITableView!

**var** settingsVCDelegate: SettingsViewControllerDelegate!

//MARK: **- Variables**

**private** **var** tsyms: [String] = ["USD", "EUR", "RUB", "GBP", "AUD", "CAD", "MYR", "NZD", "JPY", "CNY"]

//[String]()

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

**super**.viewDidLoad()

tsymTableView.delegate = **self**

tsymTableView.dataSource = **self**

tsymTableView.tableFooterView = UIView(frame: .zero)

// fetchInternationalCurrenciesCodes()

}

}

//MARK: **- Storyboard Instance**

**extension** TSYMPickerViewController {

**static** **unowned** **var** storyboardInstance: TSYMPickerViewController {

**return** UIStoryboard(name: String(describing: **self**), bundle: **nil**).instantiateInitialViewController() **as**! TSYMPickerViewController

}

}

/\*

//MARK: **- Fetch international currencies codes**

extension TSYMPickerViewController {

private func fetchInternationalCurrenciesCodes() {

let networkManager: NetworkManager = NetworkManager.GetNationalCurrencyCodes

var tsyms = [String]()

networkManager.createNetworkSession { (json) in

guard let dict = json[0].dictionaryObject as? [String:String]

else { return }

for (key, \_) in dict {

tsyms.append(key)

}

self.tsyms = tsyms

self.tsymTableView.reloadData()

}

}

}

\*/

//MARK: **- TableViewDelegate & DataSource**

**extension** TSYMPickerViewController: UITableViewDelegate, UITableViewDataSource {

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

**return** 64.0

}

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

**return** tsyms.count

}

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

**guard** **let** cell = tsymTableView.dequeueReusableCell(withIdentifier: TSYM\_CELL\_IDENTIFIER, for: indexPath) **as**? TSYMCell **else** { **return** UITableViewCell() }

cell.tsymName.text = tsyms[indexPath.row]

**return** cell

}

**func** tableView(**\_** tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

**if** !tsyms.isEmpty {

**if** tsyms[indexPath.row] != "" {

**let** userdefaults = UserDefaults.standard

userdefaults.set(tsyms[indexPath.row] **as** **Any**, forKey: getInternationalCurrencyCode)

**self**.settingsVCDelegate.controllerWillAppearAfterDismiss()

dismiss(animated: **true**, completion: **nil**)

}

}

}

}

//

// ChartsViewController.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/13/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** UIKit

**import** Charts

**class** ChartsViewController: UIViewController {

//MARK: **- Variables**

**public** **var** coin: Coin!

**private** **var** chartDataArray: [ChartData] = [ChartData]()

//MARK: **- Outlets**

**@IBOutlet** **weak** **var** firstChart: LineChartView!

//MARK: **- Lifecycle**

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

**super**.viewDidLoad()

uiSetting()

fetchData()

}

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

**super**.viewWillAppear(**true**)

navigationItem.rightBarButtonItem = UIBarButtonItem(image: UIImage(named: "exchangeSymbol"), style: .plain, target: **self**, action: **#selector**(goToExchangeScreen))

}

//MARK: **- Actions**

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

**let** exchangeScreen = ExchangeViewController.storyboardInstance

exchangeScreen.coinToExchange = coin

navigationController?.present(exchangeScreen, animated: **true**, completion: **nil**)

}

}

//MARK: **- Storyboard Instance**

**extension** ChartsViewController {

**static** **unowned** **var** storyboardInstance: ChartsViewController {

**return** UIStoryboard(name: String(describing: **self**), bundle: **nil**).instantiateInitialViewController() **as**! ChartsViewController

}

}

//MARK: **- UI**

**extension** ChartsViewController {

**func** uiSetting() {

setChartView()

}

}

//MARK: **- Creating test chart dataset**

**extension** ChartsViewController {

**private** **func** setChartView() {

firstChart.scaleYEnabled = **false**

firstChart.scaleXEnabled = **false**

firstChart.xAxis.enabled = **false**

firstChart.rightAxis.enabled = **false**

firstChart.drawGridBackgroundEnabled = **false**

}

**private** **func** setChartDataset() {

**var** lineChartEntry = [ChartDataEntry]()

**if** **self**.chartDataArray.count == 0 {

lineChartEntry.append(ChartDataEntry())

} **else** {

**for** i **in** 0..<**self**.chartDataArray.count {

**let** entry = ChartDataEntry()

entry.x = chartDataArray[i].time

entry.y = chartDataArray[i].value

lineChartEntry.append(entry)

}

}

**let** line1 = LineChartDataSet(entries: lineChartEntry, label: "Daily chart")

line1.circleColors = [mainBlueColor **as** NSUIColor]

line1.circleHoleColor = mainBlueColor **as** NSUIColor

line1.circleRadius = 0.0

line1.lineWidth = 1.0

line1.colors = [mainBlueColor **as** NSUIColor]

**let** data = LineChartData()

data.addDataSet(line1)

firstChart.data = data

firstChart.notifyDataSetChanged()

}

}

//MARK: **- Data fetch**

**extension** ChartsViewController {

**private** **func** fetchData() {

**var** tsym = "USD"

**if** **let** tsymUD = UserDefaults.standard.value(forKey: getInternationalCurrencyCode) **as**? String {

tsym = tsymUD

}

**let** networkManager = NetworkManager.GetChartData(tsym: tsym, fsym: coin.shortName)

networkManager.createNetworkSession { (json) **in**

**self**.chartDataArray = ChartData.getDataArray(fromJSON: json)

// print(self.chartDataArray)

**self**.setChartDataset()

}

}

}

//

// ExchangeViewController.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/13/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** UIKit

**class** ExchangeViewController: UIViewController {

//MARK: **- Outlets**

**@IBOutlet** **weak** **var** parentBackgroundView: UIView!

**@IBOutlet** **weak** **var** fromTextField: UITextField!

**@IBOutlet** **weak** **var** toTextField: UITextField!

//MARK: **- Variables**

**public** **var** coinToExchange: Coin!

//MARK: **- Lifecycle**

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

**super**.viewDidLoad()

uiSetting()

}

}

//MARK: **- Storyboard Instance**

**extension** ExchangeViewController {

**static** **unowned** **var** storyboardInstance: ExchangeViewController {

**return** UIStoryboard(name: String(describing: **self**), bundle: **nil**).instantiateInitialViewController() **as**! ExchangeViewController

}

}

//MARK: **- UI**

**extension** ExchangeViewController {

**private** **func** uiSetting() {

parentBackgroundView.layer.masksToBounds = **false**

parentBackgroundView.layer.shadowColor = UIColor.black.cgColor

parentBackgroundView.layer.shadowOpacity = 0.4

parentBackgroundView.layer.shadowOffset = .zero

parentBackgroundView.layer.shadowRadius = 2

parentBackgroundView.layer.shouldRasterize = **true**

fromTextField.delegate = **self**

toTextField.delegate = **self**

fromTextField.tag = 111

toTextField.tag = 222

fromTextField.placeholder = coinToExchange.shortName

**if** **let** tsym = UserDefaults.standard.value(forKey: getInternationalCurrencyCode) **as**? String {

toTextField.placeholder = tsym

}

fromTextField.addTarget(**self**, action: **#selector**(textFieldDidChange), for: .editingChanged)

toTextField.addTarget(**self**, action: **#selector**(textFieldDidChange), for: .editingChanged)

}

}

//MARK: **- TFDelegate**

**extension** ExchangeViewController: UITextFieldDelegate {

**func** textFieldDidBeginEditing(**\_** textField: UITextField) {

fromTextField.text = ""

toTextField.text = ""

}

//Custom func: tracking tf changed

**@objc** **private** **func** textFieldDidChange(**\_** sender: UITextField) {

**if** sender.tag == fromTextField.tag {

**if** **let** from = fromTextField.text {

**if** **let** fromAsDouble = Double(from) {

**let** result = fromAsDouble \* coinToExchange.priceValue

toTextField.text = String(result)

}

}

} **else** **if** sender.tag == toTextField.tag {

**if** **let** from = toTextField.text {

**if** **let** fromAsDouble = Double(from) {

**let** result = fromAsDouble / coinToExchange.priceValue

fromTextField.text = String(result)

}

}

}

}

}

//

// NetworkManager.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/10/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** Foundation

**import** UIKit

**import** Alamofire

**import** SwiftyJSON

**public** **let** baseUrl = "https://cryptocompare.com"

**public** **let** minApiBaseUrl = "https://min-api.cryptocompare.com/data"

**public** **let** internationalCurrencyCodesAPI = "https://openexchangerates.org/api/currencies.json"

**private** **let** apiKey = "247f4279d26a7e83c5256cc4a26f624006a0c8f5a220b6c0afbd2f0e09860c4f"

**enum** NetworkManager {

**case** GetAllCurrencies(tsym: String, page: Int, limit: Int)

**case** GetNationalCurrencyCodes

**case** GetChartData(tsym: String, fsym: String)

**private** **var** fullUrl: URL? {

**get** {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**if** **let** url = URL(string: minApiBaseUrl + "/top/totalvolfull") {

**return** url

}

**case** .GetNationalCurrencyCodes:

**return** URL(string: internationalCurrencyCodesAPI)

**case** .GetChartData(**\_**, **\_**):

**return** URL(string: minApiBaseUrl + "/v2/histoday")

}

**return** **nil**

}

}

**private** **var** headers: HTTPHeaders {

**get** {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**return** HTTPHeaders(["apiKey":apiKey])

**case** .GetNationalCurrencyCodes:

**return** HTTPHeaders([:])

**case** .GetChartData(**\_**, **\_**):

**return** HTTPHeaders([:])

}

}

}

**private** **var** parameters: Parameters {

**switch** **self** {

**case** .GetAllCurrencies(**let** tsym, **let** page, **let** limit):

**return** ["tsym": tsym, "limit": limit, "page": page]

**case** .GetNationalCurrencyCodes:

**return** [:]

**case** .GetChartData(**let** tsym, **let** fsym):

**return** ["tsym": tsym, "fsym": fsym, "limit": "100"]

}

}

**private** **var** method: HTTPMethod {

**switch** **self** {

**case** .GetAllCurrencies(**\_**, **\_**, **\_**):

**return** .get

**case** .GetNationalCurrencyCodes:

**return** .get

**case** .GetChartData(**\_**, **\_**):

**return** .get

}

}

**public** **func** createNetworkSession(completion: **@escaping** (JSON)->()) {

**guard** **let** fullUrlUnwrapped = fullUrl **else** { **return** }

AF.request(fullUrlUnwrapped, method: method, parameters: parameters, encoding: URLEncoding.default, headers: headers, interceptor: **nil**, requestModifier: **nil**).responseJSON { (response) **in**

**switch** response.result {

**case** .success(**let** json):

print("SUCCESS")

**let** json = JSON(arrayLiteral: json)

completion(json)

**case** .failure(**let** error):

print("ERROR: \(error)")

}

}

}

}

// YOU GOT INTO MY GENJUTSU

//Ooh, yeah  
//Lil Pump!  
//Yeah

//

// Constants.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/9/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** Foundation

**import** UIKit

**public** **let** CURRENCY\_CELL\_IDENTIFIER = "CurrencyCell"

**public** **let** SETTING\_CELL\_IDENTIFIER = "SettingCell"

**public** **let** TSYM\_CELL\_IDENTIFIER = "TSYMCell"

**public** **let** cellTopGradientColor = UIColor(red: 26.0/255.0, green: 27.0/255.0, blue: 28.0/255.0, alpha: 1.0)

**public** **let** cellBottomGradientColor = UIColor(red: 33.0/255.0, green: 34.0/255.0, blue: 35.0/255.0, alpha: 1.0)

**public** **let** mainBlueColor = UIColor(red: 95.0/255.0, green: 84.0/255.0, blue: 250.0/255.0, alpha: 1.0)

//MARK: **- UserDefaults keys**

**public** **let** firstAppLogin = "FIRST\_APP\_LOGIN"

**public** **let** getInternationalCurrencyCode = "InternationalCurrencyCode"

//

// Coin.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/10/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** Foundation

**import** UIKit

**import** SwiftyJSON

**class** Coin {

**var** shortName: String = ""

**var** fullName: String = ""

**var** price: String = ""

**var** imgURL: String = ""

**var** priceValue: Double = 0.0

**init**(shortName: String, fullName: String, price: String, imgURL: String, priceValue: Double) {

**self**.shortName = shortName

**self**.fullName = fullName

**self**.price = price

**self**.imgURL = imgURL

**self**.priceValue = priceValue

}

**init**() {

}

**static** **func** getAllCoinsArray(fromJSON json: JSON) -> [Coin] {

**var** coins = [Coin]()

**let** dataArray = json[0]["Data"].array

print(dataArray)

**if** **let** dataArrayUnwrapped = dataArray {

**for** data **in** dataArrayUnwrapped {

**let** coin = Coin()

**if** **let** shortName = data["CoinInfo"]["Name"].string {

coin.shortName = shortName

}

**if** **let** fullName = data["CoinInfo"]["FullName"].string {

coin.fullName = fullName

}

**if** **let** tsymFromUserConfig = UserDefaults.standard.value(forKey: getInternationalCurrencyCode) **as**? String {

**if** **let** price = data["DISPLAY"][tsymFromUserConfig]["PRICE"].string {

coin.price = price

}

**if** **let** priceValue = data["RAW"][tsymFromUserConfig]["PRICE"].double {

coin.priceValue = priceValue

}

}

**if** **let** imageURL = data["CoinInfo"]["ImageUrl"].string {

coin.imgURL = imageURL

}

coins.append(coin)

}

}

**return** coins

}

}

//

// ChartData.swift

// CryptoCurrency\_v2.0

//

// Created by Artyom on 5/13/20.

// Copyright © 2020 Artyom. All rights reserved.

//

**import** Foundation

**import** UIKit

**import** SwiftyJSON

**class** ChartData {

**var** time: Double = 0.0

**var** value: Double = 0.0

**init**(time: Double, value: Double) {

**self**.time = time

**self**.value = value

}

**init**() {

}

**static** **func** getDataArray(fromJSON json: JSON) -> [ChartData] {

**var** chartDataArray = [ChartData]()

**let** dataArray = json[0]["Data"]["Data"].array

**if** **let** dataArrayUnwrapped = dataArray {

**for** data **in** dataArrayUnwrapped {

**let** chartData = ChartData()

**if** **let** time = data["time"].double {

chartData.time = time

}

**if** **let** value = data["high"].double {

chartData.value = value

}

chartDataArray.append(chartData)

}

}

**return** chartDataArray

}

}