МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

Отчет
по научно-исследовательской работе
Тема: Разработка фреймворка для оптимизации создания iOS приложений

Студент гр. 4303 _____ Ахриев Р.А.

Преподаватель _____

Санкт-Петербург
2019

ЗАДАНИЕ

НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ

Студент Ахриев Р.А.

Группа 4303

Задание на НИР:

1. Исследовать существующие фреймворки для оптимизации создания iOS приложения.
2. Разработать фреймворк для оптимизации создания iOS приложения.

Дата сдачи отчета: 26.12.2019

Дата защиты отчет: 27.12.2019

Студент _____ Ахриев Р.А

Преподаватель _____

АННОТАЦИЯ

Задача НИР – разработать фреймоврк для оптимизации разработки iOS приложения. Для решения этой задачи были изучены различные библиотеки, расширяющие возможности iOS API. Далее было разработан компонент ListController, решающий ряд повседневных задач iOS разработчика.

SUMMARY

The task of research is to develop a framework for optimizing the development of iOS applications. To solve this problem, various libraries were expanded that expand the capabilities of the iOS API. Further, the ListController component was developed, which solves a number of everyday tasks of the iOS developer.

ВВЕДЕНИЕ

Мобильные телефоны играют важную роль в нашем, современном мире. Практически у каждого человека сейчас есть мобильный телефон. Смартфоны имеют широкий функционал. Пользователи используют смартфоны для совершения звонков, просмотра видео, прослушивания аудиокниг, съемки. Мобильные приложения, создаваемы для смартфонов позволяют существенно расширить базовый функционал телефона. Существуют различные приложения для самых разнообразных целей.

В данный момент более 99% рынка занимают операционные системы iOS (14%) и Android (85%).

В данной дипломной работе речь пойдет о создании фреймворка для оптимизации создания iOS приложений.

1. Среда разработки

1.1. Выбор IDE

Xcode IDE является главным инструментов, предоставляемый компанией разработки продуктов Apple. Xcode, тесно интегрированный с платформами Cocoa и Cocoa Touch, является невероятно продуктивной средой для создания приложений для Mac, iPhone, iPad, Apple Watch и Apple TV [1] (Рисунок 1).

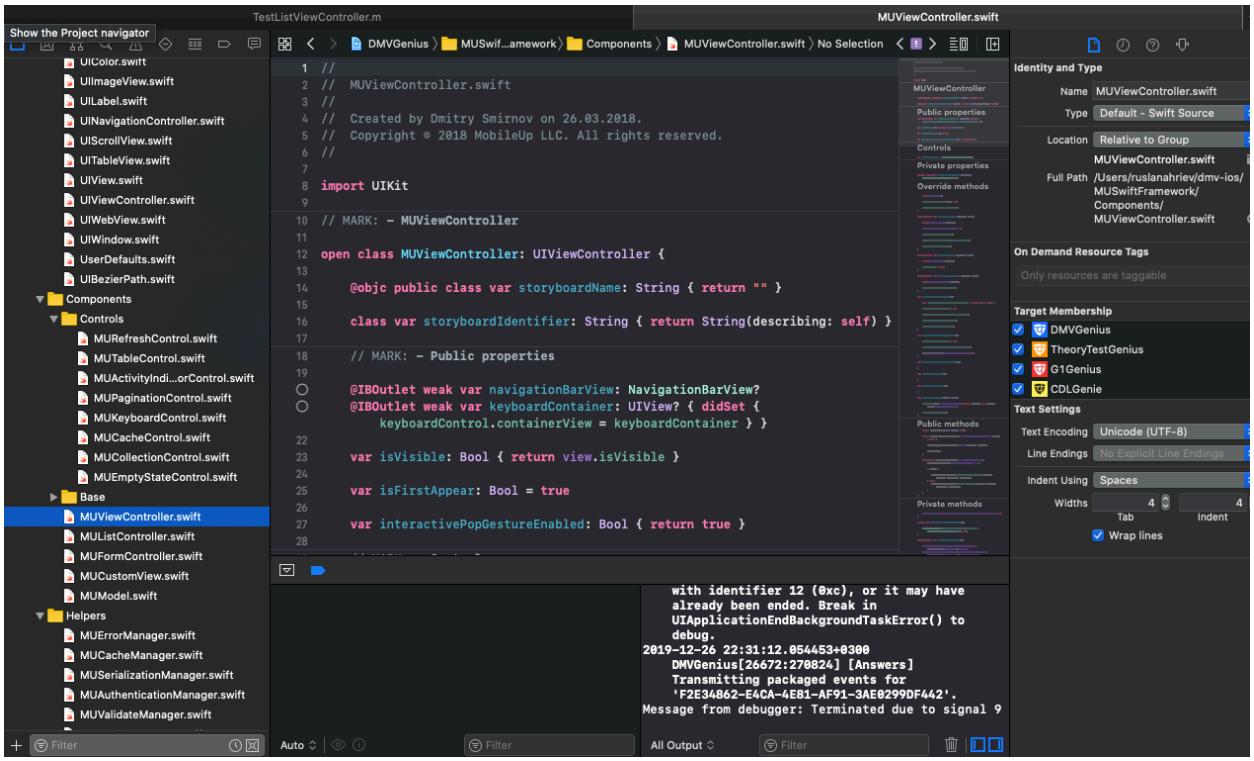

Рисунок 1 – Xcode IDE

2. Разработка компонента ListController

2.1. Базовый функционал

Основной задачей UITableView[2], чьим расширением и является ListController, является отображения списка ячеек. Дополняя функционал данного класса мы получаем мощный инструмент, который умеет эффективно работать со списками, подгружать их из сети и сортировать в зависимости от критерия (Рисунок 2).

```
// MARK: - DemoListController

class DemoListController: ListController {

    class override var storyboardName: String { return "DemoList" }

    // MARK: - Override properties

    override var hasRefresh: Bool { return true }

    override var hasPagination: Bool { return true }

    override var hasEmptyState: Bool { return true }

    // MARK: - Private properties

    @IBOutlet private weak var tableProvider: UITableView! { didSet { tableView = tableProvide

    @IBOutlet private weak var emptyViewProvider: UIView! { didSet { emptyView = emptyViewProv

    // MARK: - Override methods

    override func beginRequest() {

        DemoService.getAll() { [weak self] (objects) in

            self?.update(objects: objects)
        }
    }
}
```

Рисунок 2 – Листинг примера реализации ListController.

В данном примере продемонстрирована возможность ListControler-а при помощи метода «beginRequest» отправить запрос на сервис, получить данные и вызвав метод «update» обновить ячейки таблицы. Реализация класса ListController находится в приложении А.

3. Разработка компонента MUViewController

3.1. Базовый функционал

Базовым UI компонентом в приложениях под iOS является ViewController, по этой причине его функционал был расширен следующими функциями:

- Роутинг
- Получение ошибок API
- Контейнер над клавиатурой

```
DemoController.push(to: self)

DemoController.push(to: navigationController)

DemoController.push(to: self) { instance in

 instance.foo = self.foo
}

DemoController.present(in: self, asRoot: true)

DemoController.present(in: self, style: .overCurrentContext)

DemoController.insert(controller: TargetController.self, into: targetView)
```

Рисунок 3 – Листинг примера использования вариаций роутинга.

3. Вывод

Apple предоставляет возможность разрабатывать приложения под систему iOS, при помощи Cocoa Touch, UIKit, Foundation и многих других базовых библиотек. С течением времени задачи, касающиеся iOS разработчика стали типичными и стали требовать структурированности. По данной причине был создан фреймворк обеспечивающий возможность ускорять процесс разработки путем использования компонентов с расширенным функционалом базовых классов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Xcode [Электронный ресурс]. URL: https://developer.apple.com/xcode/ide/ —дата обращения: 24.12.2019).

[2] UITableView [Электронный ресурс]. URL: https://developer.apple.com/documentation/uikit/uitableview—дата обращения: 24.12.2019).

Приложение А

```
import UIKit

// MARK: - MUListControlDelegate

@objc protocol MUListControlDelegate: class {
    func cellIdentifier(for object: MUModel, at indexPath: IndexPath) -> String?
    func cellIdentifier(at indexPath: IndexPath) -> String?
    func cellDidSelected(for object: MUModel)
    func getSection(for object: MUModel) -> String?
    func isObjectChanged(for object: MUModel) -> Bool
    func objectDidChanged(with objects: [MUModel])
    func scrollDidScroll(_ scrollView: UIScrollView)

    // MARK: - UITableViewDataSource

    @objc optional func tableView(_ tableView: UITableView,
```

```
numberOfRowsInSection section: Int) -> Int

    @objc optional func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell

    // MARK: - UITableViewDelegate

    @objc optional func tableView(_ tableView: UITableView, heightForRowAt
indexPath: IndexPath) -> CGFloat

    @objc optional func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath)

    // MARK: - UICollectionViewDataSource

    @objc optional func collectionView(_ collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int

    @objc optional func collectionView(_ collectionView: UICollectionView,
cellForItemAt indexPath: IndexPath) -> UICollectionViewCell
```

```
    // MARK: - UICollectionViewDelegateFlowLayout

    @objc optional func collectionView(_ collectionView: UICollectionView, layout: UICollectionViewLayout, section : Int) -> CGFloat

    // MARK: - UICollectionViewDelegate

    @objc optional func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath)
}

// MARK: - MUListController

public class MUListController: MUViewController, MUListControlDelegate {

    // MARK: - Public properties

    var hasRefresh: Bool { return false }
```

```
    var hasPagination: Bool { return false }

    var hasEmptyState: Bool { return false }

    var hasCache: Bool { return false }

    @IBOutlet weak var tableView: UITableView?

    @IBOutlet weak var collectionView: UICollectionView?

    @IBOutlet weak var emptyView: UIView? { didSet {
emptyStateControl.emptyView = emptyView } }

    var objects: [MUModel] {

        set { setObjects(with: newValue)  }

        get { return getObjects() }
```

```
}

// MARK: - Controls

var tableControl = MUTableControl()

var collectionControl = MUCollectionControl()

var emptyStateControl = MUEmptyStateControl()

var refreshControl = MURefreshControl()

var paginationControl = MUPaginationControl()

var cacheControl: MUCacheControlProtocol? { return nil }


// MARK: - Private properties


private var activityTimer: Timer?

// MARK: - Override methods

override public func viewDidLoad() {

    super.viewDidLoad()

    cacheControl?.setup(with: self)

    tableControl.setup(with: self)
```

```
    collectionControl.setup(with: self)
    paginationControl.setup(with: self)
    refreshControl.setup(with: self)
    emptyStateControl.setup(with: self)
}
override public func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
}
override public func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    emptyStateControl.updateLayout()
}

override func appErrorDidBecome(error: Error) {
    super.appErrorDidBecome(error: error)
    paginationControl.cancelLastPage()
    hideActivityIndicators(withDelay: 0.3)
}
```

```
    // MARK: - Public methods
    func hideActivityIndicators() {
        hideActivityIndicator()
        refreshControl.stopAnimation()
        emptyStateControl.stopAnimation()
        paginationControl.stopAnimation()
    }

    func hideActivityIndicators(withDelay delay: TimeInterval) {
        activityTimer?.invalidate()
        activityTimer = Timer.scheduledTimer(withTimeInterval: delay, repeats:
false, block: { [weak self] timer in

            self?.hideActivityIndicators()
        })
    }
    // MARK: - Request
    func beginRequest() {
```

```
        update(objects: [])
    }
    func update(objects newObjects: [MUModel]) {
        if refreshControl.isRefreshing {
            updateWithTimeInterval(objects: newObjects)
        } else {
            updateWithoutTimeInterval(objects: newObjects)
        }
        hideActivityIndicators()
    }

    func updateWithTimeInterval(objects newObjects: [MUModel], interval:
Double = 0.1) {

        Timer.scheduledTimer(withTimeInterval: interval, repeats: true, block: {
[weak self] (timer) in
            if self?.refreshControl.isRefreshing ?? false == false,
self?.tableView?.contentOffset.y ?? 0 == 0 {
                self?.updateWithoutTimeInterval(objects: newObjects)
```

```
            timer.invalidate()
        }
    })
}

func updateWithoutTimeInterval(objects newObjects: [MUModel]) {
    if paginationControl.page > 1 {

        objects += newObjects
    } else {
        objects = newObjects
    }
}
func requestObjects(withIndicator: Bool = true) {
    if withIndicator {
        showActivityIndicator()
    }
    beginRequest()
```

```
}

// MARK: - Private methods

private func getObjects() -> [MUModel] {
    if tableView != nil {
        return tableControl.objects
    } else {
        return collectionControl.objects
    }
}

private func setObjects(with newObjects: [MUModel]) {
    if tableView != nil {
        tableControl.objects = newObjects
    } else {
        collectionControl.objects = newObjects
    }
}

// MARK: - MUListControlDelegate

func cellIdentifier(for object: MUModel, at indexPath: IndexPath) -> String? {
```

```
    return "Cell"
}
func cellIdentifier(at indexPath: IndexPath) -> String? {
    return nil
}
func cellDidSelected(for object: MUModel) {
}

func getSection(for object: MUModel) -> String? {
    return ""
}
func isObjectChanged(for object: MUModel) -> Bool {
    return false
}
func objectDidChanged(with objects: [MUModel]) {
}
func scrollDidScroll(_ scrollView: UIScrollView) {
    paginationControl.scroll(with: scrollView)
```

```
    }
}
// MARK: - MURefreshControlDelegate
extension MUListController: MURefreshControlDelegate {
    func refreshControlDidRefresh() {
        paginationControl.reset()
        requestObjects(withIndicator: false)
    }
}
// MARK: - MUPaginationControlDelegate
extension MUListController: MUPaginationControlDelegate {

    func paginationControlDidRequestMore(page: Int) {
        requestObjects(withIndicator: false)
    }
}
```