

Министерство образования Республики Беларусь

Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

## **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ МОБИЛЬНЫХ ПЛАТФОРМ**

*Рекомендовано УМО по образованию  
в области информатики и радиоэлектроники  
в качестве учебно-методического пособия для специальности  
1-40 01 01 «Программное обеспечение информационных технологий»*

Минск БГУИР 2020

УДК 004.42:621.395.721.5(076)  
ББК 32.973.3я73+32.884.1я73  
Т38

Авторы:

В. В. Трус, И. Л. Калитеня, А. М. Бакунов,  
И. Н. Коренская, О. М. Бакунова

Рецензенты:

заведующий кафедрой информатики и веб-дизайна учреждения образования  
«Белорусский государственный технологический университет»  
кандидат технических наук, доцент Д. М. Романенко;

доцент кафедры управления информационными ресурсами  
Академии управления при Президенте Республики Беларусь  
кандидат технических наук, доцент В. В. Лабоцкий

Т38 **Технологии** программирования мобильных платформ : учеб.-метод.  
пособие / В. В. Трус [и др.]. – Минск : БГУИР, 2020. – 76 с. : ил.  
ISBN 978-985-543-549-6.

Приведено шесть лабораторных работ на языке swift в среде разработки Xcode с примерами выполнения, в которых содержатся теоретические и практические сведения.

Предназначено для студентов, изучающих дисциплину «Разработка программного обеспечения для мобильных платформ».

УДК 004.42:621.395.721.5(076)  
ББК 32.973.3я73+32.884.1я73

ISBN 978-985-543-549-6

© УО «Белорусский государственный университет информатики и радиоэлектроники», 2020

## СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1 ПРИЛОЖЕНИЕ ДЛЯ ВЕДЕНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ЗАМЕТОК .....	4
ЛАБОРАТОРНАЯ РАБОТА №2 ПРИЛОЖЕНИЕ ДЛЯ ПОЛУЧЕНИЯ ИНФОРМАЦИИ С ВЕБ-СТРАНИЦ С ИСПОЛЬЗОВАНИЕМ RSS-ФАЙЛОВ .....	24
ЛАБОРАТОРНАЯ РАБОТА №3 ПРИЛОЖЕНИЕ ДЛЯ СЧИТЫВАНИЯ И ГЕНЕРИРОВАНИЯ QR-КОДОВ.....	36
ЛАБОРАТОРНАЯ РАБОТА №4 ПРИЛОЖЕНИЕ С ИСПОЛЬЗОВАНИЕМ ДОПОЛНЕННОЙ РЕАЛЬНОСТИ ARKIT .....	46
ЛАБОРАТОРНАЯ РАБОТА №5 КРОССПЛАТФОРМЕННАЯ ИГРА DOODLE JUMP .....	59
ЛАБОРАТОРНАЯ РАБОТА №6 ВИДЖЕТЫ.....	68
ЛИТЕРАТУРА.....	76

# ЛАБОРАТОРНАЯ РАБОТА №1

## ПРИЛОЖЕНИЕ ДЛЯ ВЕДЕНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ЗАМЕТОК

**Цель работы:** закрепить навыки работы со средой разработки и основными методами языка программирования.

### 1.1 Краткие теоретические сведения

Основной средой разработки для устройств под управлением ОС Mac OS и iOS является среда разработки Xcode. Данная среда обеспечивает нативную разработку, что дает лучшую совместимость и переносимость кода на различные подсемейства систем.

Интерфейс среды содержит четыре основных раздела: Navigator (навигатор), Editor (редактор), Debug Area (область отладки) и Utility Area (утилиты).

Вы можете менять размер этих окон (достаточно навести курсор на границы области) или по необходимости скрывать различные области в правом верхнем углу (рисунок 1.1).



Рисунок 1.1 – Окно управления областями видимости

#### **Область навигатора (Navigator)**

В этой панели присутствует несколько навигаторов, переключение между ними осуществляется с помощью *Navigator selector bar*. Три часто используемых навигатора: проект (Project), поиск (Search), проблема (Issue).

#### **Навигатор проекта (Project Navigator)**

Здесь содержатся все файлы проекта (рисунок 1.2).

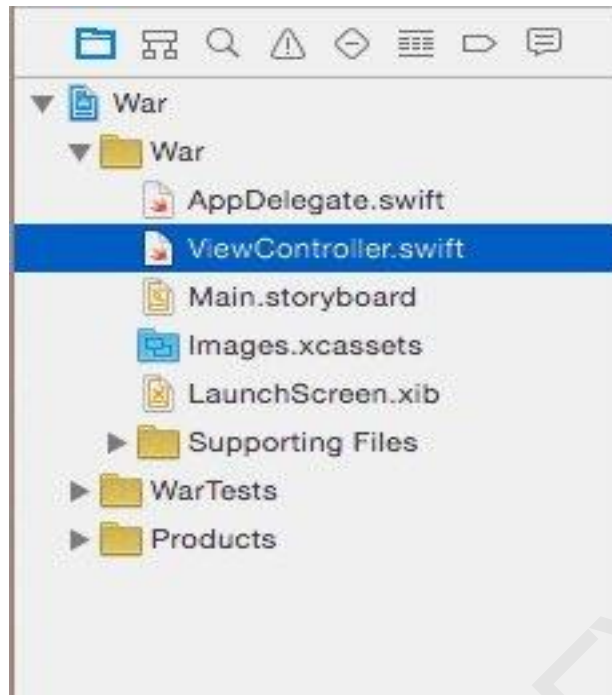


Рисунок 1.2 – Окно навигатора проекта

В рамках навигатора проекта можно создавать группы файлов: щелкнуть правой кнопкой мыши и выбрать Create Group. Группа существует только в проекте Xcode, в файловой системе папка не создается, т. е. на жестком диске файлов не будет.

Корневой узел навигатора проекта – это файл проекта Xcode (отмеченный синим значком). Если щелкнуть по значку, свойства проекта откроются в области редактора (рисунок 1.3).

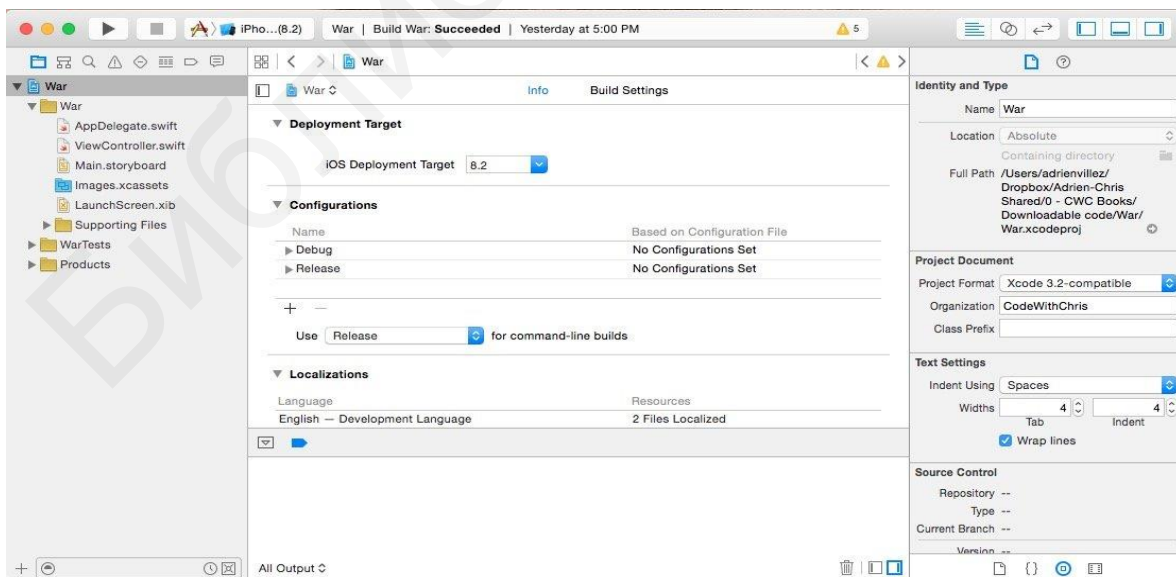


Рисунок 1.3 – Окно свойств проекта

## Навигатор проблем (Issue Navigator)

В результате разработки приложения нередко случаи возникновения ошибок. Для их устранения в среде Xcode предусмотрен навигатор проблем (рисунок 1.4). Он покажет все имеющиеся у приложения проблемы и вероятные причины возникновения ошибок.

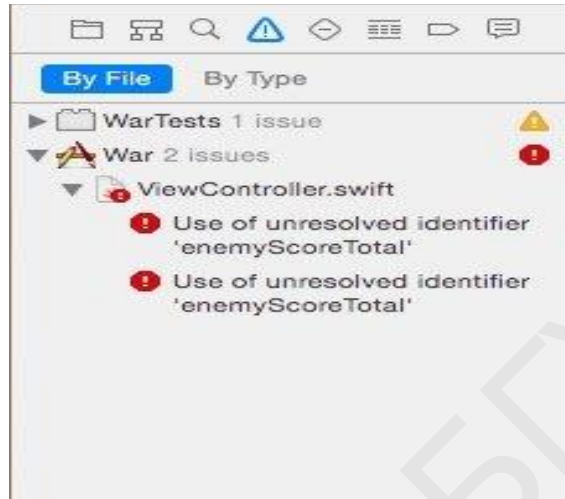


Рисунок 1.4 – Окно навигатора проблем

Если обнаружатся определенные дефекты, которые будут мешать Xcode создавать приложение, последует остановка, и ошибки в навигаторе выделятся красным цветом.

При щелчке кнопкой мыши редактор по ошибке покажет соответствующие файлы и линии. В навигаторе Issue также появляются предупреждения, выделенные желтым цветом. Предупреждение может указать на наличие определенных проблем.

## Область редактора (Editor)

Область редактора – основное рабочее окно, в котором и происходит работа с кодом (рисунок 1.5).

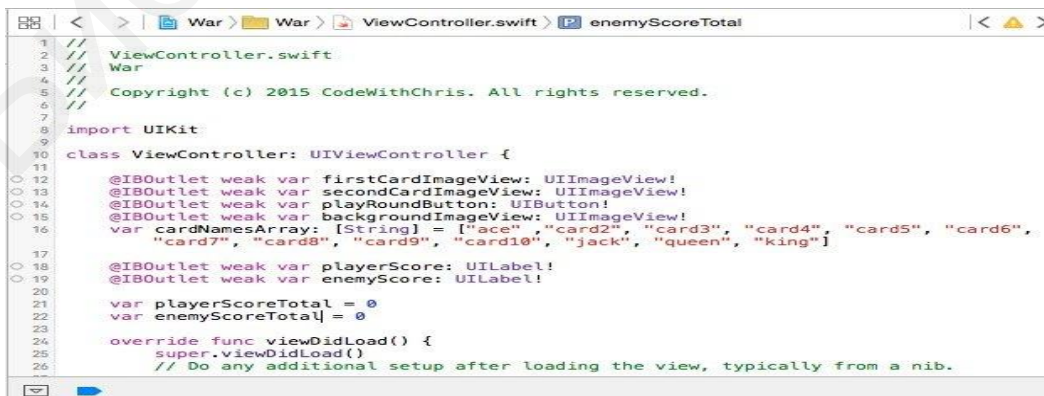


Рисунок 1.5 – Окно редактора кода

## Окно конструктора интерфейса

В Xcode интегрирован конструктор интерфейса (рисунок 1.6), в котором можно просматривать Storyboard или файлы XIB. Область редактора становится визуальным конструктором.

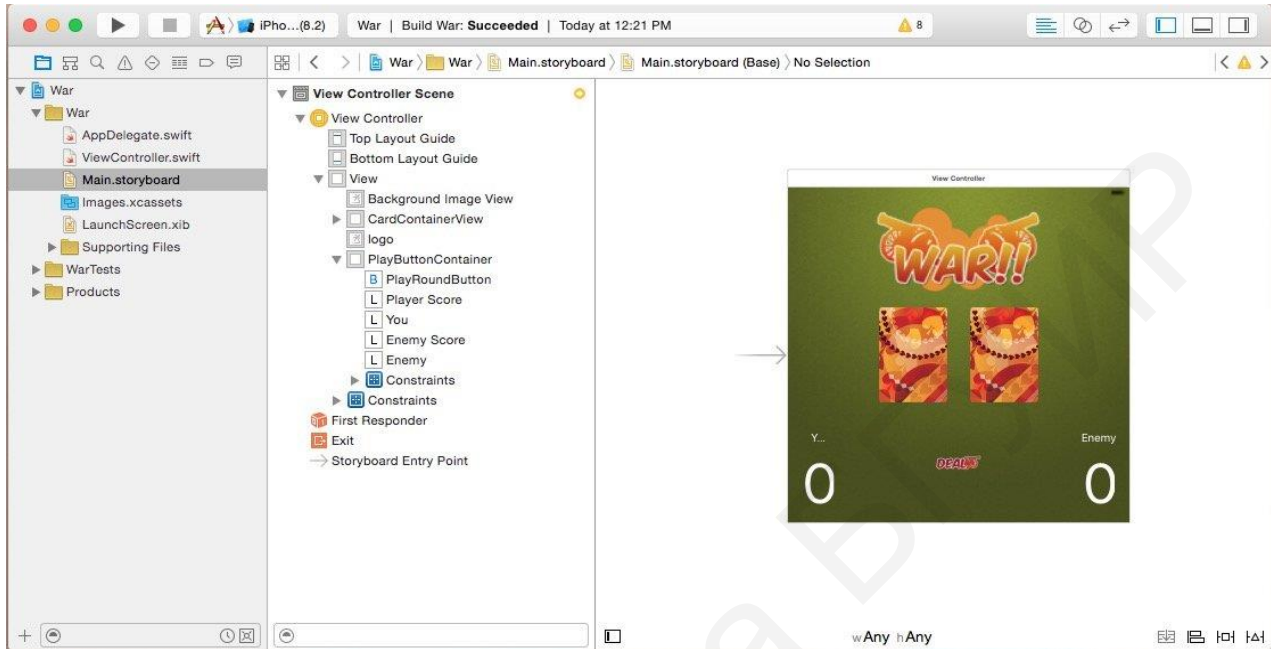


Рисунок 1.6 – Окно конструктора интерфейсов

## Контрольные точки

В данном разделе можно установить контрольные точки и точки остановки в определенных линиях кода. Устанавливая эти точки, вы заметите индикатор синего цвета. Чтобы отменить контрольную точку, щелкните по синему индикатору – он станет темнее (рисунок 1.7).

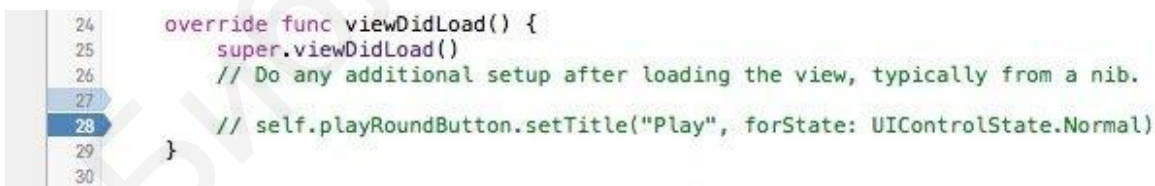


Рисунок 1.7 – Окно с установленными контрольными точками

Если щелкнуть по синему индикатору и одновременно перетащить его, контрольная точка полностью удалится.

Просмотреть список всех контрольных точек проекта можно с помощью вкладки **Навигатор контрольных точек** в области навигатора.

## Область утилит

Область утилит Xcode состоит из двух панелей: панель-инспектор и библиотека (рисунок 1.8).



Рисунок 1.8 – Окно области утилит

Панель-инспектор предоставляет подробную информацию о файле, который выделен в навигаторе проекта.

## Область отладки

При запуске приложения раздел отладки покажет консольный вывод и состояние различных переменных (рисунок 1.9).

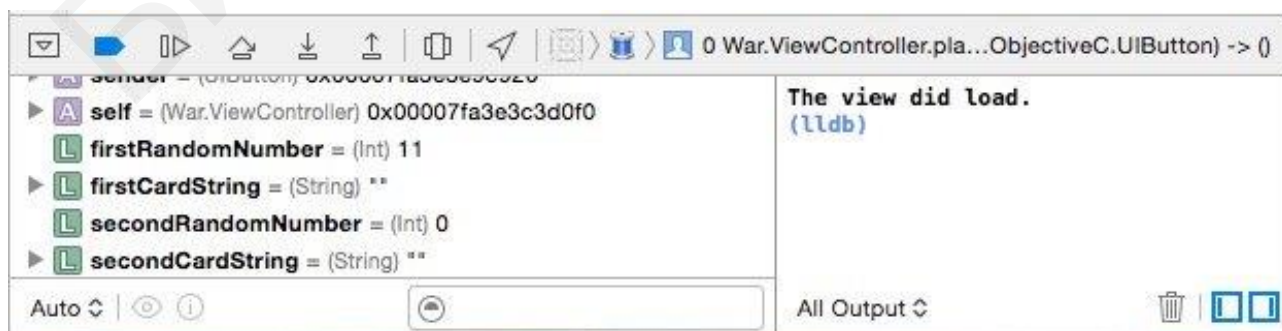


Рисунок 1.9 – Окно отладки



## Панель инструментов

На рисунке 1.10 представлена панель инструментов.

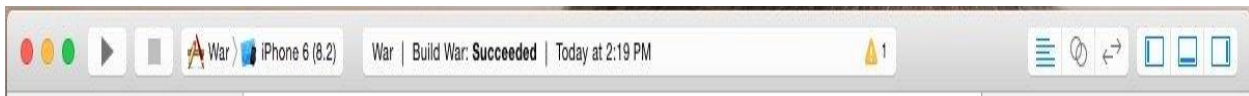


Рисунок 1.10 – Окно панели инструментов

Слева расположена кнопка запуска (Run). Она создает и запускает приложение, а также запускает приложение в iOS-эмуляторе. Соседняя кнопка останавливает приложение и возвращает пользователя к Xcode.

Если нажать и удерживать кнопку Run, появятся следующие опции для работы с приложением:

- Test запускает модульное тестирование, если это предусмотрено в проекте.
- Profile измеряет различные показатели приложения, такие как производительность, использование памяти и др.
- Analyze дает возможность анализировать код и выявлять потенциальные ошибки.

Больше информации об этом предоставлено в официальной документации [Apple iOS Documentation](#).

В выпадающем списке возле кнопки Stop содержатся названия устройств, на которых будет работать приложение. Можно выбрать эмуляторы iPhone или iPad (или другие версии, если они установлены). Раздел iOS Device запускает приложения на электронном устройстве, если оно подключено и должным образом настроено (рисунок 1.11).

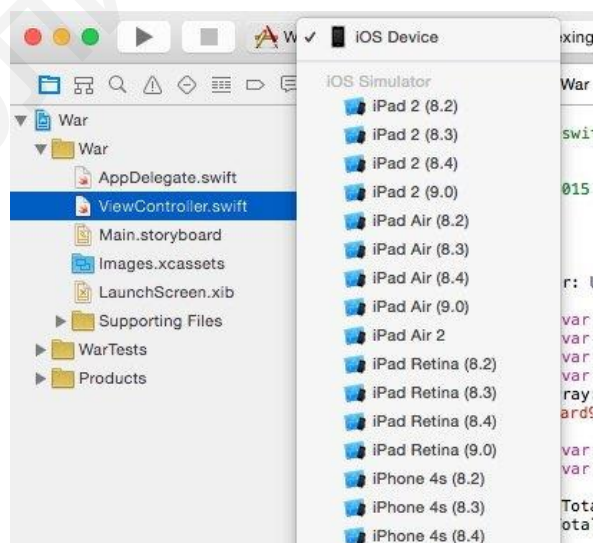


Рисунок 1.11 – Окно управления областями видимости

## iOS-эмулятор

К Xcode 8 прилагается несколько полезных эмуляторов для тестирования. Большую часть времени разработки вы можете использовать эмуляторы, а для тестирования, когда уже практически все сделано, выбрать смартфон или планшет.

Окно эмулятора представлено на рисунке 1.12.



Рисунок 1.12 – Окно эмулятора

Эмулятор позволяет:

- менять расположения экрана (ландшафтный/портретный режимы);
- моделировать различные GPS-координаты;
- моделировать сценарии с низким объемом памяти.

С выходом версии Xcode 6.3 появился эмулятор Apple Watch для тестирования приложений под эту операционную систему (ОС).

## 1.2 Необходимое ПО

1. Установленная среда разработки **Xcode** на ПК.
2. Операционная система **iOS** на мобильном устройстве или эмуляторе.

## 1.3 Порядок выполнения лабораторной работы

Для создания проекта необходимо выполнить следующие действия:

- 1) открыть среду разработки **Xcode** (рисунок 1.13);
- 2) выбрать **Create a new Xcode project** (рисунок 1.14);
- 3) выбрать **SingleViewApp**;

4) ввести данные для разрабатываемого проекта, обязательно выбрав **Use Core Data** (рисунок 1.15).

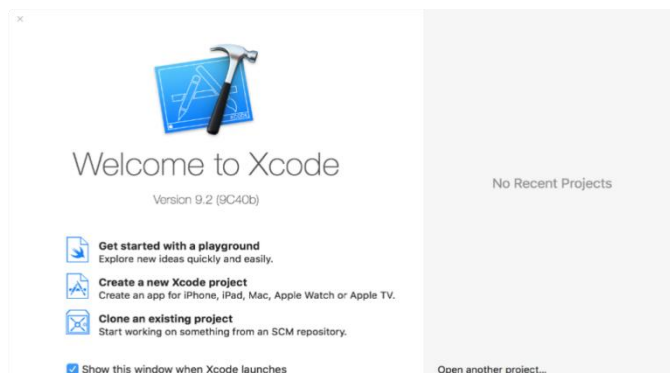


Рисунок. 1.13 – Начальное окно Xcode

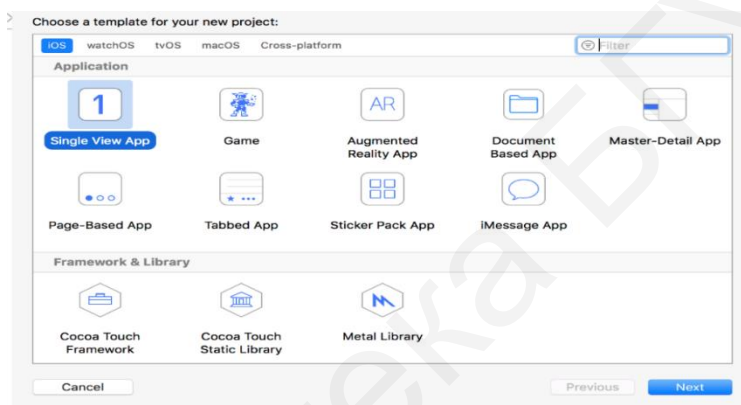


Рисунок 1.14 – Создание нового Xcode проекта

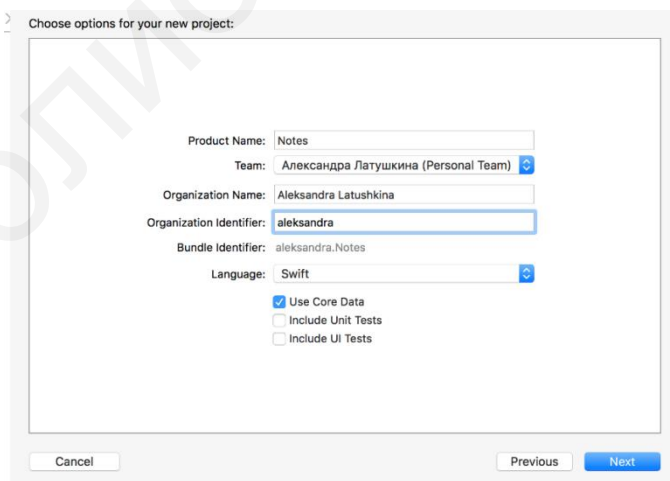




Рисунок 1.15 – Ввод данных для проекта

## Создание пользовательского интерфейса

Для создания пользовательского интерфейса необходимо:

1. Перейти в файл **Main.storyboard**.
2. На вкладке **Object library** добавить **Table View Controller** и два **View Controller**:
  - выбрать **Table View Controller**;
  - выполнить следующие действия:
    - Editor;
    - Embed In;
    - Navigation Controller.
3. Расставить элементы, как показано на рисунке 1.16. Для кнопок добавления и сохранения надо добавить фоновые изображения  и .
4. Добавить связи между контроллерами (рисунок 1.16). Для этого при нажатой клавише **Ctrl** перетащить курсор кнопкой мыши от **Bar Button Item** к **View Controller**, который будет отвечать за добавление заметки, и выбрать **Present Modally**. Для добавления следующей связи при нажатой клавише **Ctrl** перетащить курсор кнопкой мыши от **Table View Controller** к **View Controller**, который будет отвечать за изменение и просмотр заметки. Присвоить идентификатор связи – **UpdateVC**.

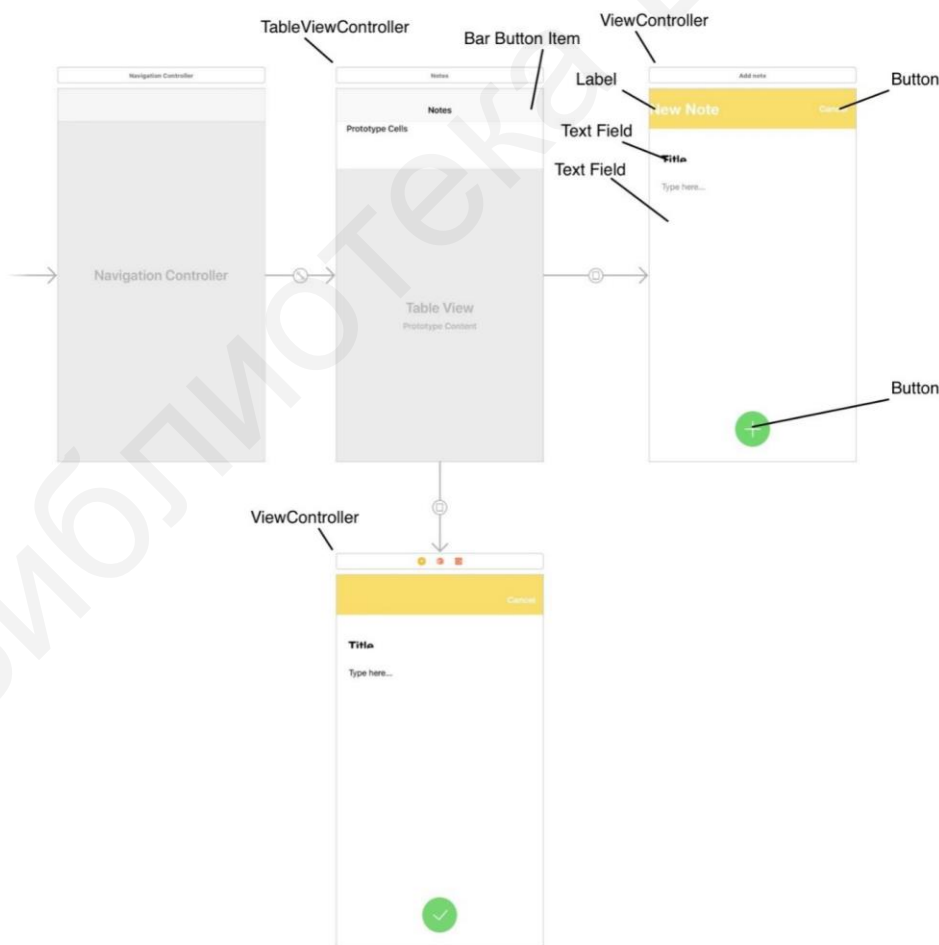


Рисунок 1.16 – Расстановка элементов

## Создание сущности и ее атрибутов

Для создания сущности и ее атрибутов необходимо (рисунок 1.17):

1. В навигаторе проекта выбрать файл **Notes.xcdatamodeld**.
2. Если он отсутствует, то следует выполнить:
  - File → New → File...;
  - iOS → Core Data → Data Model;
  - ввести название файла;
  - нажать **Create**.
3. Создать сущность:
  - выбрать **AddEntity**;
  - добавить атрибуты с необходимым типом.

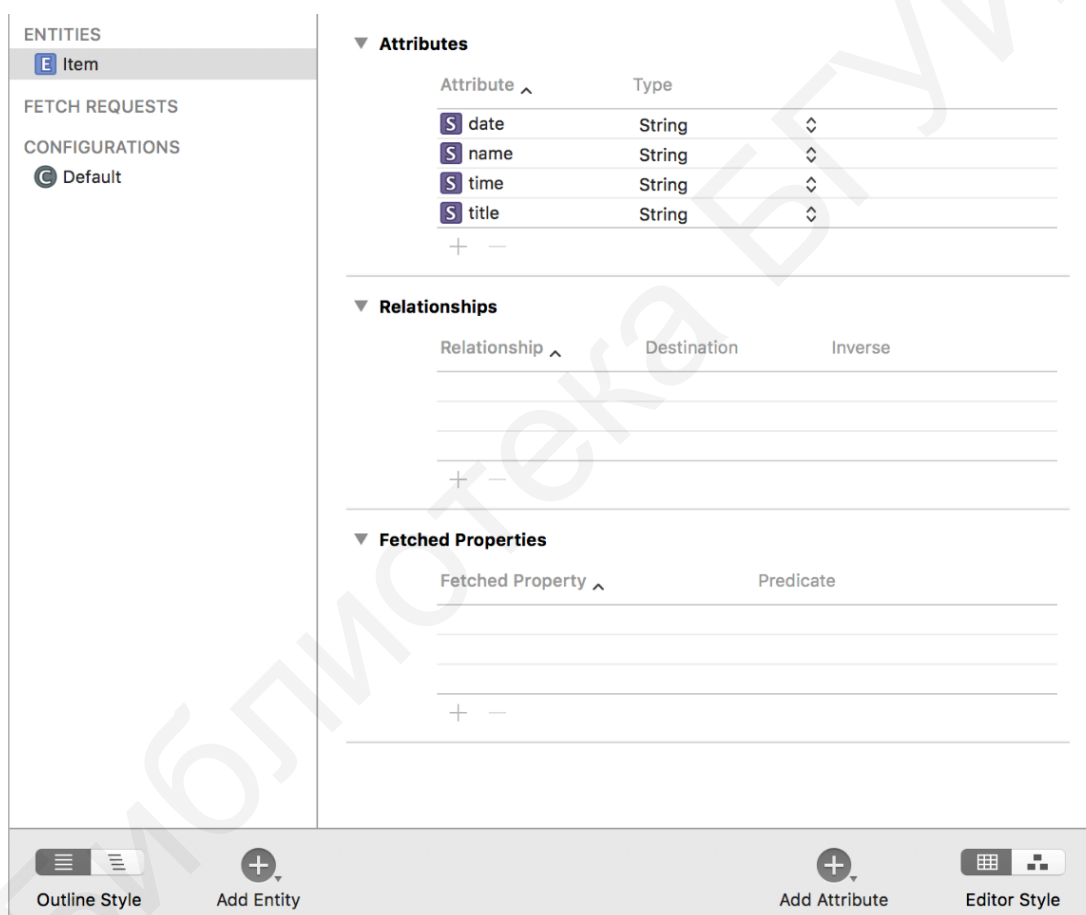


Рисунок 1.17 – Создание сущности и ее атрибутов

## Создание ячейки

Для создания кастомной ячейки надо выполнить следующие действия:

1. File → New → File... .
2. iOS → Source → Cocoa Touch Class.
3. Выбрать подкласс **UITableViewCell** → ввести название **CustomTableViewCell** → поставить в поле галочку **Also create XIB file** → Create.

4. Перейти в созданный файл **CustomTableViewCell.xib**.
5. Расставить элементы в соответствии с рисунком 1.18.
6. Перейти в файл **Main.storyboard**, выбрать ячейку в таблице и присвоить ей класс **CustomTableViewCell**.
7. Перейти в файл **CustomTableViewCell.swift** и удалить все содержимое класса **CustomTableViewCell**.
8. Объявить элементы, перетаскивая их кнопкой мыши при нажатой клавише **Ctrl** от каждого Label созданной ячейки в данный класс (рисунок 1.19).

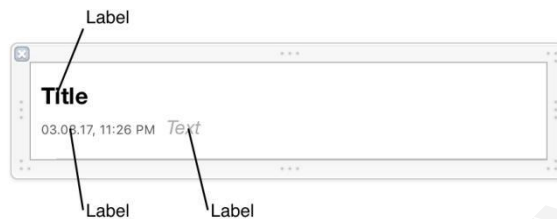


Рисунок 1.18 – Расстановка элементов на форме

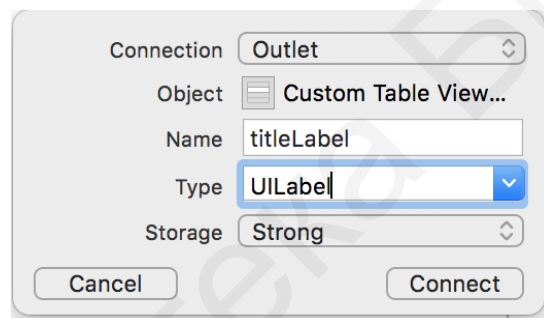


Рисунок 1.19 – Объявление элементов

В итоге содержимое файла должно выглядеть следующим образом:

```
import UIKit
class CustomTableViewCell: UITableViewCell {
    @IBOutlet var titleLabel: UILabel!
    @IBOutlet var textLabel: UILabel!
    @IBOutlet var dateTimeLabel: UILabel!
}
```

### Создание классов

Необходимо создать три класса:

- 1) displayTableViewController;
- 2) addItemViewController;
- 3) updateItemViewController.

Эти классы будут отвечать соответственно за отображение списка заметок, добавление и изменение заметок.

Для создания классов требуется выполнить:

- 1) File → New → File...;
- 2) iOS → Source → Cocoa Touch Class;
- 3) выбрать подкласс **UIViewController** → ввести название → поставить галочку в поле **Also create XIB file** → нажать **Create**.

### Код файла `displayTableViewController.swift`

```
import UIKit
class displayTableViewController: UITableViewController, UISearchBarDelegate
{
let context=(UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext
var items: [Item] = []
var selectedIndex: Int!
var filteredData: [Item] = []

override func viewDidLoad() {
super.viewDidLoad()
navigationController?.navigationBar.barTintColor = UIColor(red: 249/255, green:
215/255, blue: 84/255, alpha: 1.0)
createSearchBar() //цвет панели навигации
self.tableView.estimatedRowHeight = 66 //приблизительная высота ячейки
}
override func didReceiveMemoryWarning() {
super.didReceiveMemoryWarning()
}
//функция, определяющая действия при появлении контроллера
override func viewWillAppear(_ animated: Bool) {
fetchData() //вызов функции обновления таблицы
}
func fetchData() //функция для обновления таблицы
do {
items = try context.fetch(Item.fetchRequest())
filteredData = items
DispatchQueue.main.async {
self.tableView.reloadData()
}
} catch {
print("Couldn't Fetch Data")
}
}
}
extension displayTableViewController { //определение содержимого ячейки
```

```

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {
    let cell= Bundle.main.loadNibNamed("CustomTableViewCell", owner: self,
    options: nil)?.first as! CustomTableViewCell
    cell.titleLabel.text = filteredData[indexPath.row].title
    cell.textLabel.text = filteredData[indexPath.row].name
    let date = filteredData[indexPath.row].date
    let time = filteredData[indexPath.row].time
    if let date = date, let time = time {
    let timeStamp = "\\(date), \\(time)"
    cell.dateTimeLabel.text = timeStamp
    }
    return cell
    }
    //задание количества строк таблицы
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
    Int) -> Int {
    return filteredData.count
    }
    //функция перехода при нажатии на ячейку
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
    IndexPath) {
        selectedIndex = indexPath.row
    performSegue(withIdentifier: "UpdateVC", sender: self)
        tableView.deselectRow(at: indexPath, animated: true)
    }
    //добавление кнопки «Удалить» при свайпе ячейки влево
    override func tableView(_ tableView: UITableView, editActionsForRowAt indexPath:
    IndexPath) -> [UITableViewRowAction]? {
    let delete=UITableViewRowAction(style:.default,title:"Delete"){(action,indexPath) in
    let item = self.filteredData[indexPath.row]
    self.context.delete(item)
    (UIApplication.shared.delegate as! AppDelegate).saveContext()
    self.filteredData.remove(at: indexPath.row)
    tableView.deleteRows(at: [indexPath], with: .fade)
    }
    delete.backgroundColor=UIColor(red:237/255,green:106/255,blue:95/255,alpha: 1.0)
    return [delete]
    }
    //задание высоты строки
    override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
    IndexPath) ->CGFloat {
    return 0
    }

```



```

}
//передача данных в контроллер изменения заметки
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "UpdateVC" {
        let updateVC = segue.destination as! updateItemViewController
        updateVC.item = filteredData[selectedIndex!]
    }
}
func createSearchBar() {//создание search bar
    let searchBar = UISearchBar()
    searchBar.showsCancelButton = false
    searchBar.placeholder = "Search"
    searchBar.searchBarStyle = UISearchBarStyle.minimal
    searchBar.delegate = self
    self.navigationItem.titleView = searchBar
}
//добавление кнопки отмены при нажатии на поиск
func searchBarTextDidBeginEditing(_ searchBar: UISearchBar) {
    searchBar.showsCancelButton = true
    let view: UIView = searchBar.subviews[0] as UIView
    let subViewsArray = view.subviews
    for subView: UIView in subViewsArray {
        if subView is UIButton {
            subView.tintColor = UIColor.white
        }
    }
}
//скрытие кнопки отмены
func searchBarCancelButtonClicked(_ searchBar: UISearchBar) {
    searchBar.showsCancelButton = false
    searchBar.text = ""
    searchBar.resignFirstResponder()
}
//осуществление поиска
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
    if searchText.isEmpty { filteredData = items }
    else {
        //фильтрация
        filteredData=items.filter{($0.name?.lowercased()).contains(searchText.lowercased())!}
    }
    DispatchQueue.main.async {
        self.tableView.reloadData()
    }
}

```

```
}  
}
```

### Код файла addItemViewController.swift

```
import UIKit  
class addItemViewController: UIViewController, UITextViewDelegate,  
UITextFieldDelegate {  
    @IBOutlet var addButton: UIButton!  
    @IBOutlet var nameLabel: UILabel!  
    //выход из контроллера при нажатии отмены  
    @IBAction func cancelAction(_ sender: Any) {  
        dismiss(animated: true, completion: nil)  
    }  
    @IBOutlet var itemTitle: UITextField!  
    @IBOutlet var itemEntryTextView: UITextView!  
    //сохранение заметки  
    @IBAction func saveItem(_ sender: Any) {  
        //проверка на содержание полей  
        if (itemEntryTextView?.text.isEmpty)! || itemEntryTextView?.text == "Type here..."  
        {  
            print("No Data")  
            let alert = UIAlertController(title: "Please type something", message:  
            "Your note was left blank", preferredStyle: .alert)  
            alert.addAction(UIAlertAction(title: "Okay", style: .default) { action in })  
            self.present(alert, animated: true, completion: nil)  
        }  
        else { //сохранение данных из полей  
            let date = Date()  
            let formatter = DateFormatter()  
            formatter.dateStyle = .short  
            formatter.dateFormat = "dd.MM.YY"  
            let currentDate = formatter.string(from: date)  
            let timeFormatter = DateFormatter()  
            timeFormatter.timeStyle = .short  
            let currentTime = timeFormatter.string(from: date)  
            let context=(UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext  
            let newEntry = Item(context: context)  
            newEntry.name = itemEntryTextView?.text!  
            newEntry.date = currentDate  
            newEntry.time = currentTime  
            let timeStamp = "\\(currentDate), \\(currentTime)"  
            if (itemTitle?.text?.isEmpty)! {  
                newEntry.title = timeStamp  
            }  
        }  
    }  
}
```



```

    }
}
//если поле ввода заметки пустое, то возвращаем "Type here..."
func textViewDidEndEditing(_ textView: UITextView) {
    if textView.text.isEmpty {
        textView.text = "Type here..."
        textView.textColor = UIColor.lightGray
    }
}
func textView(_ textView: UITextView, shouldChangeTextIn range: NSRange,
replacementText text: String) -> Bool {
    if (text == "\n") {
        textView.resignFirstResponder()
        return false
    }
    return true
}
}

```

### **Код файла updateItemViewController.swift**

```

import UIKit
class updateItemViewController: UIViewController, UITextViewDelegate,
UITextFieldDelegate {
    let context = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext
    var item: Item!
    @IBOutlet var updateButton: UIButton!
    @IBOutlet weak var entryText: UITextView!
    @IBOutlet weak var entryTitle: UITextField!
    @IBOutlet var textLabel: UILabel!
    //выход из контроллера при нажатии отмены
    @IBAction func cancelAction(_ sender: Any) {
        dismiss(animated: true, completion: nil)
    }
    //обновление выбранной заметки
    @IBAction func updateAction(_ sender: Any) {
        //определение формата даты и времени
        let date = Date()
        let formatter = DateFormatter()
        formatter.dateStyle = .short
        formatter.dateFormat = "dd.MM.YY"
        let currentDate = formatter.string(from: date)
    }
}

```

```

let timeFormatter = DateFormatter()
timeFormatter.timeStyle = .short
let currentTime = timeFormatter.string(from: date)
guard let newText = entryText.text else {
return
}
guard let newTitle = entryTitle.text else {
return
}
//присвоение атрибутам значений из полей
item.title = newTitle
item.name = newText
item.date = currentDate
item.time = currentTime
(UIApplication.shared.delegate as! AppDelegate).saveContext()
dismiss(animated: true, completion: nil)
}
override func viewDidLoad() { //действия при загрузке контроллера
super.viewDidLoad()
navigationController?.navigationBar.prefersLargeTitles = true
navigationController?.navigationBar.barStyle = UIBarStyle.black
navigationController?.navigationBar.tintColor = UIColor.white
entryText!.delegate = self
entryText!.becomeFirstResponder()
configureEntryData(entry: item!)
print(item)
self.entryText.delegate = self
self.entryTitle.delegate = self
self.entryTitle.placeholder = "Title"
updateButton.layer.cornerRadius = self.updateButton.bounds.size.height / 2
updateButton.clipsToBounds = true
updateButton.contentMode = .center
titleLabel.textAlignment = .center
}
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
self.view.endEditing(true)
}
func textViewShouldBeginEditing(_ textView: UITextView) -> Bool {
entryText.resignFirstResponder()
return (true)
}
}
//удаление "Type here..." изменение цвета при нажатии на поле ввода
func textViewDidBeginEditing(_ textView: UITextView) {

```

```

if (textView.text == "Type here...")
{
    textView.text = ""
    textView.textColor = UIColor.black
}
}
//при пустом поле ввода заметки возвращается "Type here..."
func textViewDidEndEditing(_ textView: UITextView) {
    if textView.text.isEmpty {
        textView.text = "Type here..."
        textView.textColor = UIColor.lightGray
    }
}
override func didReceiveMemoryWarning() {
super.didReceiveMemoryWarning()//Dispose of any resources that can be recreated
}
func configureEntryData(entry: Item) {
    guard let text = entry.name else {return }
    guard let title = entry.title else {return }
    entryText!.text = text
    entryTitle!.text = title
}
func textView(_ textView: UITextView, shouldChangeTextIn range: NSRange,
replacementText text: String) -> Bool {
    if (text == "\n") {
        textView.resignFirstResponder()
        return false
    }
    return true
}
}

```

#### **1.4 Результат выполнения лабораторной работы**

Для запуска приложения в симуляторе выбирается любая модель смартфона или планшета (рисунок 1.20).

Для запуска приложения на телефоне требуется наличие аккаунта разработчика, а также выполнение следующих действий (рисунок 1.21):

- 1) подключить аккаунт к проекту;
- 2) при запуске выбрать смартфон, который подключен к компьютеру.

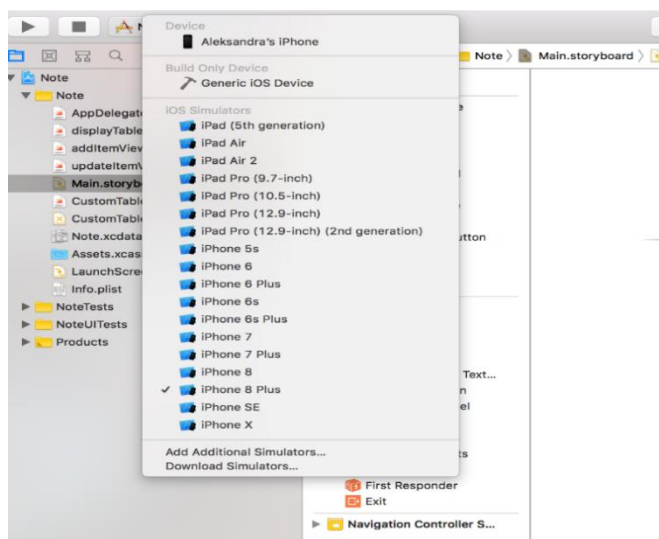


Рисунок 1.20 – Выбор модели телефона или смартфона

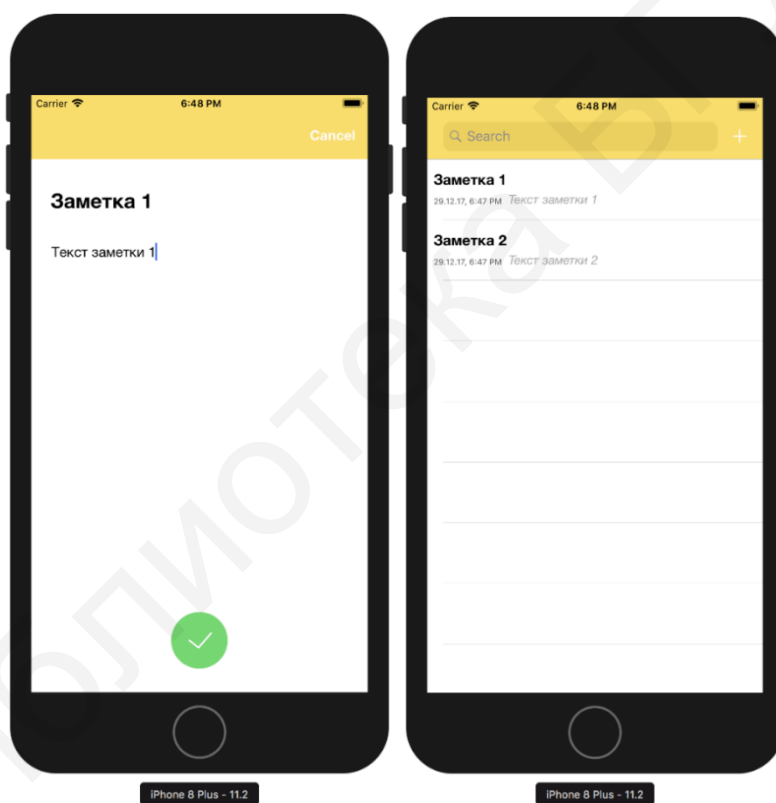


Рисунок 1.21 – Запуск приложения на телефоне

## Выводы

В результате выполнения лабораторной работы должно быть разработано мобильное приложение с пользовательским интерфейсом. Приложение должно поддерживать добавление, изменение информации и работать на основе операционной системы iOS.

## ЛАБОРАТОРНАЯ РАБОТА №2

### ПРИЛОЖЕНИЕ ДЛЯ ПОЛУЧЕНИЯ ИНФОРМАЦИИ С ВЕБ-СТРАНИЦ С ИСПОЛЬЗОВАНИЕМ RSS-ФАЙЛОВ

**Цель работы:** закрепить навыки работы с данными в формате rss-файлов.

#### 2.1 Необходимое ПО

1. Наличие установленной библиотеки **Cordova**.
2. Установленное приложение **AdobePhoneGap**.

#### 2.2 Порядок выполнения лабораторной работы

Для выполнения данной лабораторной работы необходимо выполнить следующие действия:

1. Создать проект (рисунок 2.1):
  - запустить **PhoneGap** на ПК;
  - нажать «+»;
  - выбрать **Create PhoneGap project**;
  - поставить флажок на пункте **Framework 7**;
  - нажать кнопку **Next**;
  - выбрать расположение проекта;
  - ввести название проекта и id (bundle, три слова через точку);
  - нажать кнопку **Create project**.

Структуру проекта после его создания можно увидеть в папке проекта (рисунок 2.2).

Папка **www** является основной папкой проекта, в ней находятся все файлы, которые будут использоваться в web-view.

Также для удобной организации работы со страницами будут использоваться **AngularJs** и **jQuery**.

Для работы понадобится **Framework7**, поставляющий js и css для готовых компонентов приложения, стилизованных под native-компоненты мобильных платформ (стили зависят от платформы, на которой запускается приложение, а также есть ряд особых компонентов под каждую из платформ).



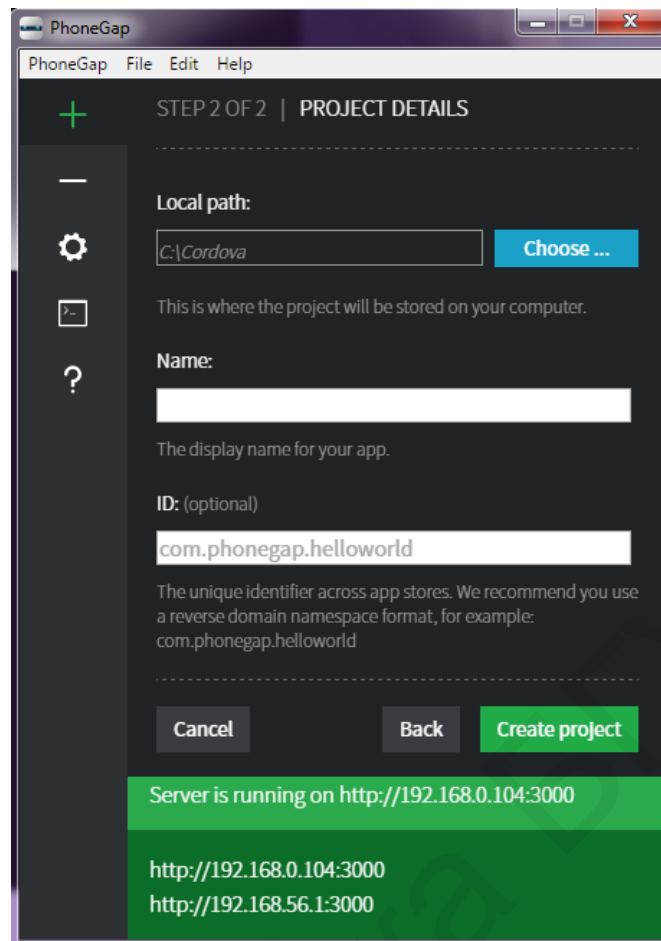


Рисунок 2.1 – Создание проекта

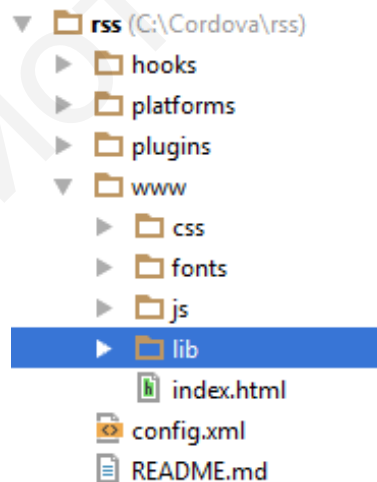


Рисунок 2.2 – Структура проекта

2. Выбрать сайт для получения информации.

В качестве исходной информации для парсинга можно взять rss-ленту, однако возможности web-view и js (в данном случае его фреймворка jQuery) позволяют получить информацию с любого веб-ресурса.

В качестве примера рассмотрим сайт <http://the-flow.ru>.

Для выполнения задания надо проанализировать структуру его страниц.

### 3. Реализовать проект.

Перед началом реализации необходимо скачать и подключить два фреймворка, которые следует добавить в папку **lib** данного проекта (рисунок 2.3):

- **AngularJs** и файл его интеграции с Framework7;
- **jQuery**.

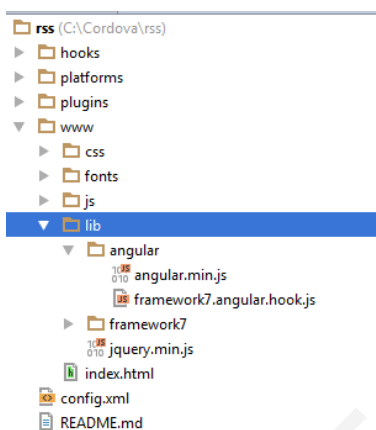


Рисунок 2.3 – Добавление фреймворков в папку **lib** данного проекта

Данные фреймворки необходимо подключить в конец главного файла **index.html** перед закрывающимся тэгом `</body>`:

```
<script type="text/javascript" src="lib/jquery.min.js"></script>  
<script type="text/javascript" src="lib/angular/angular.min.js"></script>
```

Необходимо скачать и установить **font-awesome** для бесплатных значков и расположить файлы в соответствующих папках (рисунок 2.4):

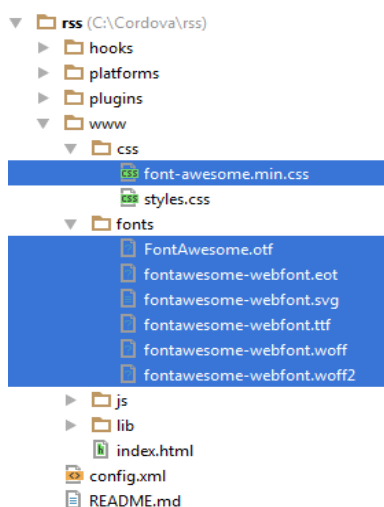


Рисунок 2.4 – Размещение файлов, необходимых для выполнения задания

Для удобства создается и подключается в **index.html** отдельный объект для хранения информации о публикации: файл **publication-item.js** в папке **js** (рисунок 2.5).

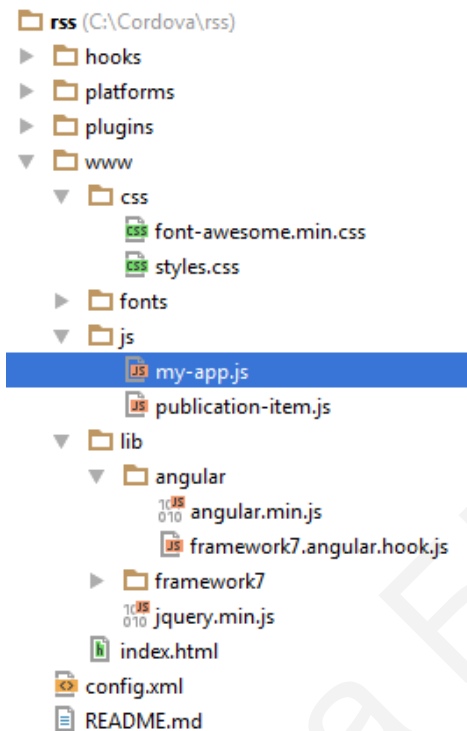


Рисунок 2.5 – Файл **publication-item.js** в папке **js**

В целях подключения объекта для хранения информации о публикации в **index.html** используется следующий скрипт:

```
<script type="text/javascript" src="js/publication-item.js"></script>
```

### Программный код

Программный код объекта **publication-item.js** включает:

- 1) вызов метода **createFromDomElement** для объекта **PublicationItem**;
- 2) передачу в него DOM-объекта публикации;
- 3) селекторы соответствующих блоков с информацией.

```
Function PublicationItem()  
{  
this.category= null;  
this.date= null;  
this.title= null;  
this.text = null;  
this.href= null;  
this.image= null;  
this.createFromDomElement = function(element)
```

```

{
this.category= element.find('.cat_name_inner').text().trim();
this.date= element.find('.publication__item-date').text().trim();
this.title= element.find('.publication__item-title').text().trim();
this.text = element.find('.publication__item-text').text().trim();
this.href='http://the-flow.ru'+element.find('.publication__item-title > a').attr('href');
this.src = 'http://the-flow.ru' + element.find('img').attr('src');
return this;
};
}

```

Далее необходимо расширить файл **index.html** и создать в нем объект для новостей.

Полный код файла **index.html**:

```

<!DOCTYPE html>
<html ng-app="flowApp">
<head>
<!--
Customize this policy to fit your own app's needs. For more guidance, see:
https://github.com/apache/cordova-plugin-whitelist/blob/master/README.md#content-security-policy
Some notes:
* gap: is required only on iOS (when using UIWebView) and is needed for JS->native communication
* https://ssl.gstatic.com is required only on Android and is needed for TalkBack to function properly
* Disables use of inline scripts in order to mitigate risk of XSS vulnerabilities. To change this:
* Enable inline JS: add 'unsafe-inline' to default-src
→
<meta http-equiv="Content-Security-Policy" content="default-src *; style-src * 'unsafe-inline'; script-src * 'unsafe-inline' 'unsafe-eval'; media-src *; img-src * filesystem: data:">
<!--Required meta tags→
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no, minimal-ui">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="format-detection" content="telephone=no">
<meta name="28isapplication-tap-highlight" content="no">

```

<!--This template defaults to the iOS CSS theme. To support both iOS and material design themes, see the Framework7 Tutorial at the link below:

<http://www.idangero.us/framework7/tutorials/maintain-both-ios-and-material-themes-in-single-app.html>

→

```
<link rel="stylesheet" href="lib/framework7/css/framework7.material.min.css">
```

```
<link rel="stylesheet" href="lib/framework7/css/framework7.material.colors.min.css">
```

```
<link rel="stylesheet" href="css/font-awesome.min.css">
```

```
<link rel="stylesheet" href="css/styles.css">
```

```
</head>
```

```
<body>
```

```
<!--Status bar overlay for full screen mode (PhoneGap) →
```

```
<div class="statusbar-overlay"></div>
```

```
<!--Views →
```

```
<div class="views">
```

```
<!--Your main view, should have "view-main" class →
```

```
<div class="view view-main">
```

```
<!--Pages container, because we use fixed-through navbar and toolbar, it has additional appropriate classes →
```

```
<div class="pages navbar-fixed" ng-class="{ 'toolbar-fixed': catCtrl.categoryView }">
```

```
<!--Page, "data-page" contains page name →
```

```
<div data-page="index" class="page" ng-controller="CategoriesCtrl as catCtrl">
```

```
<div class="navbar">
```

```
<div class="navbar-inner">
```

```
<div class="left">
```

```
<a href="#" class="link icon-only" ng-click="catCtrl.hideArticle()" ng-if="!catCtrl.categoryView">
```

```
<€ class="fa fa-arrow-left" aria-hidden="true"></i>
```

```
</a>
```

```
</div>
```

```
<div class="center">The-flow</div>
```

```
<div class="right">
```

```
<a href="#" class="link icon-only" ng-click="catCtrl.refresh()" ng-if="catCtrl.categoryView">
```

```
<€ class="fa fa-refresh" aria-hidden="true"></i>
```

```
</a>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
<div class="toolbar tabbar tabbar-scrollable" ng-if="catCtrl.categoryView">
```

```
<div class="toolbar-inner">
```

```
<a href="#" class="tab-link"
```

```
ng-repeat="category in catCtrl.categories"
```

```

ng-bind="category.name"
  ng-class="{ active: catCtrl.isActiveCategory(category) }"
  ng-click="catCtrl.setActiveCategory(category)"></a>
</div>
</div>
<!--Additional "tabbar-scrollable" class →
<div class="page-content">
<div class="list-block media-list" ng-if="catCtrl.categoryView">
<ul>
<li ng-repeat="publication in catCtrl.publications">
<a href="#" class="item-link item-content" ng-click="catCtrl.showArticle(publica-
tion.href)">
<div class="item-media">
<div class="item-image" ng-style="{ 'background-image': 'url('+publica-
tion.src+')' }">
</div>
</div>
<div class="item-inner">
<div class="item-title-row">
<div class="item-title"></div>
<div class="item-after" ng-bind="publication.date"></div>
</div>
<div class="item-subtitle" ng-bind="publication.title"></div>
<div class="item-text" ng-bind="publication.text"></div>
</div>
</a>
</li>
</ul>
</div>
<div class="webview-wrapper" ng-if="!catCtrl.categoryView">
<iframe ng-src="{ { catCtrl.activeArticleLink } }" frameborder="0"></iframe>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
<script type="text/javascript" src="cordova.js"></script>
<script type="text/javascript" src="lib/jquery.min.js"></script>
<script type="text/javascript" src="lib/angular/angular.min.js"></script>
<script type="text/javascript" src="lib/framework7/js/framework7.min.js"></script>
<script type="text/javascript" src="js/publication-item.js"></script>
<script type="text/javascript" src="js/my-app.js"></script>

```

```
</body>
</html>
```

Код файла **my-app.js** содержит логику работы приложения. Он включает:

- модуль приложения;
- контроллер для компонента View с логикой работы приложения.

```
Var myApp = {};
var mainView = {};
try
{
angular.module('flowApp', [])
.config(function($httpProvider){
delete $httpProvider.defaults.headers.common['X-Requested-With'];
})
.run(function($http, $sce) {
$http.defaults.headers.common.ContentType = 'text/html';
$sce.trustAsResourceUrl('http://the-flow.ru');
myApp = new Framework7({
modalTitle: 'Framework7',
pushState: true,
angular: true
});
mainView = myApp.addView('.view-main', {});
})
.controller('CategoriesCtrl', ['$scope', '$http', '$sce', function($scope, $http, $sce) {
this.categoryView= true;
this.activeArticleLink= '';
this.categories= {
photo: {
name: 'Фото',
link: '/foto'
},
videos: {
name: 'Клипы',
link: '/videos'
},
music: {
name: 'Рецензии',
link: '/music'
},
releases: {
name: 'Альбомы',
```

```

link: '/releases'
},
features: {
name: 'Тексты',
link: '/features'
},
news: {
name: 'Новости',
link: '/news'
}
};
this.activeCategory= this.categories.photo;
this.publications= [];
this.setActiveCategory = function(category) {
if (!this.isActiveCategory(category)) {
myApp.showPreloader('Загрузка');
this.activeCategory= category;
this.publications= [];
var ctrl = this;
$http.get('http://the-flow.ru' + category.link).then(function (response)
{
var page = $(response.data),
items=page.find('.content__left_column section.publication .items .publication__item');
if (items.length> 0) {
angular.forEach(items, function (element) {
ctrl.publications.push((new PublicationItem()).createFromDomElement($(element)));
});
}
myApp.hidePreloader();
}, function (responseData) {
//alert(responseData);
myApp.hidePreloader();
});
}
};
this.isActiveCategory = function(category) {
return this.activeCategory=== category;
};
this.refresh = function() {
var cat = this.activeCategory;
this.activeCategory= null;
this.setActiveCategory(cat);
};

```



```

this.showArticle = function(link) {
this.categoryView= false;
this.activeArticleLink= $sce.trustAsResourceUrl(link);
setTimeout(function() {
$('.webview-wrapper iframe').css({
transform: 'scale(' + $('.webview-wrapper').outerWidth()/1280 + ')',
height:$('.webview-wrapper').outerHeight()/($('.webview-wrapper').outerWidth()/1280)+'px'
})
});
};
this.hideArticle = function() {
this.categoryView= true;
this.activeArticleLink= '';
};
(function construct(context) {
context.refresh()
})(this);
});
} catch€ {
alert€;
}

```

В приведенном коде контроллер состоит из объекта с категориями, которые содержат:

- название категории;
- ссылку на страницу категории;
- массив для загруженных публикаций;
- поле с активной категорией (по умолчанию является категория **photo**);
- главный метод **setActiveCategory**, который в качестве параметра принимает ссылку на объект категории.

Метод **setActiveCategory** ставит категорию из параметра как активную, выводит **preloader** и начинает загрузку страницы, ссылка на которую содержится в объекте категории. После загрузки страницы по селектору выбираются все блоки публикаций, циклом создаются и заносятся в общий массив публикаций объекты **PublicationItem**, которым в метод **createFromDomElement** передается соответствующий элемент публикации. В соответствии с новыми данными **AngularJs** автоматически обновит **view**.

Метод **showArticle** меняет тип отображения компонента **View** и достает соответствующую информацию из пришедшей параметром публикации (взаимодействие **View** и контроллера содержится в коде **index.html**, все методы навешиваются на соответствующие элементы **View**).

## 2.3 Результат выполнения лабораторной работы

Перед запуском приложения на мобильном устройстве необходимо выполнить следующие действия (рисунок 2.6):

- скачать приложение **PhoneGap**;
- войти в сеть, в которой компьютер и смартфон подключены к одному роутеру;
- нажать кнопку запуска в приложении **PhoneGap** для ПК;
- ввести соответствующий ip в приложении на телефоне.

Результаты выполненных действий можно увидеть на рисунке 2.7.

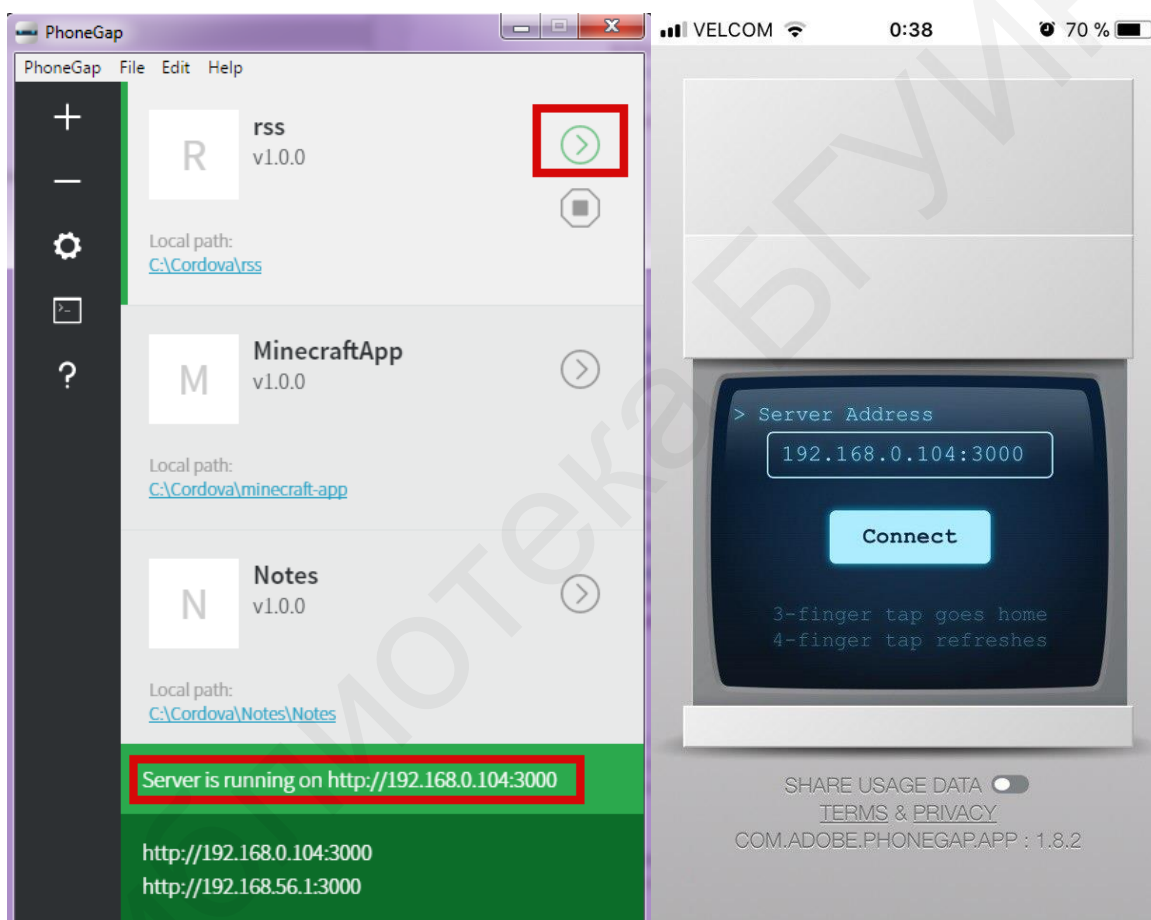


Рисунок 2.6 – Настройка компьютера и смартфона

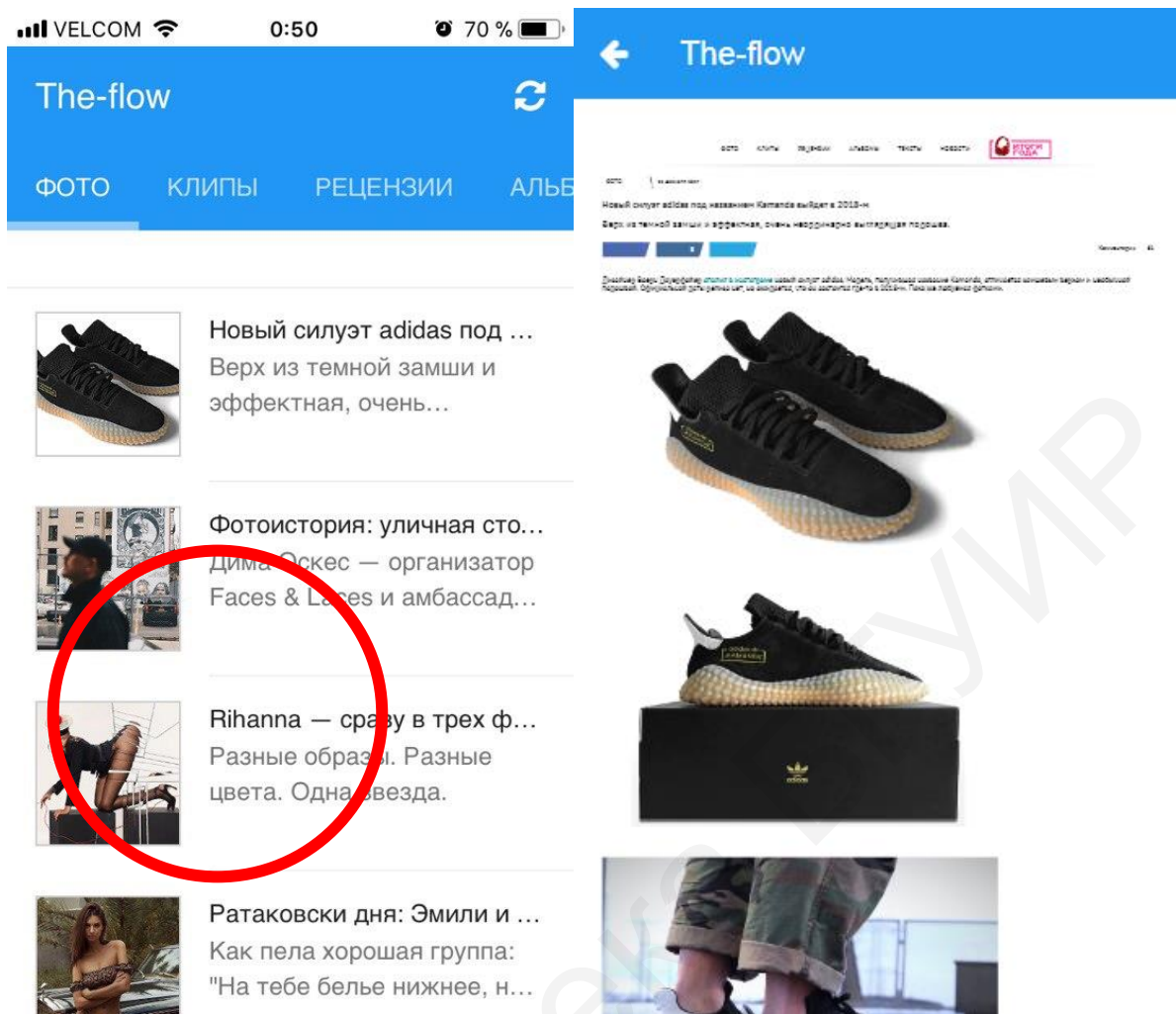


Рисунок 2.7 – Результаты работы приложения

Для работы проекта на операционной системе Android достаточно выполнить следующее:

- зарегистрироваться по ссылке на сайте **Phonegap**;
- сохранить в своем кабинете архив с папкой проекта;
- нажать кнопку **Download .apk**.

### Выводы

В результате выполнения лабораторной работы должно быть разработано мобильное приложение для получения информации с веб-ресурсов, использующее для работы rss-файлы.

## ЛАБОРАТОРНАЯ РАБОТА №3

### ПРИЛОЖЕНИЕ ДЛЯ СЧИТЫВАНИЯ И ГЕНЕРИРОВАНИЯ QR-КОДОВ

**Цель работы:** закрепить навыки работы с языком программирования Swift и обработкой изображения, получаемого с камеры мобильных устройств.

#### 1.1 Необходимое ПО

1. Установленная среда разработки **Xcode** на ПК.
2. Операционная система **iOS** на мобильном устройстве или эмуляторе.

#### 1.2 Порядок выполнения лабораторной работы

Последовательность действий для выполнения лабораторной работы приводится для операционной системы **iOS**:

1. Запустить среду разработки **Xcode**.
2. Запустить приложение.

После запуска появится окно приветствия, в котором справа отобразятся ранее созданные проекты. Для создания нового проекта необходимо нажать **Create a new Xcode project** (рисунок 3.1).



Рисунок 3.1 – Создание нового проекта

3. Выбрать **SingleViewApp** в появившемся окне выбора шаблона приложения (рисунок 3.2).

4. Нажать кнопку **Next**.

5. Задать имя приложения, организации и идентификатора (рисунок 3.3).

В открывшемся окне выбора опций приложения указать имя приложения, организации и идентификатора, учитывая, что к названию идентификатора приписывается com. Например, com.yourname.

6. Выбрать язык **Swift**.

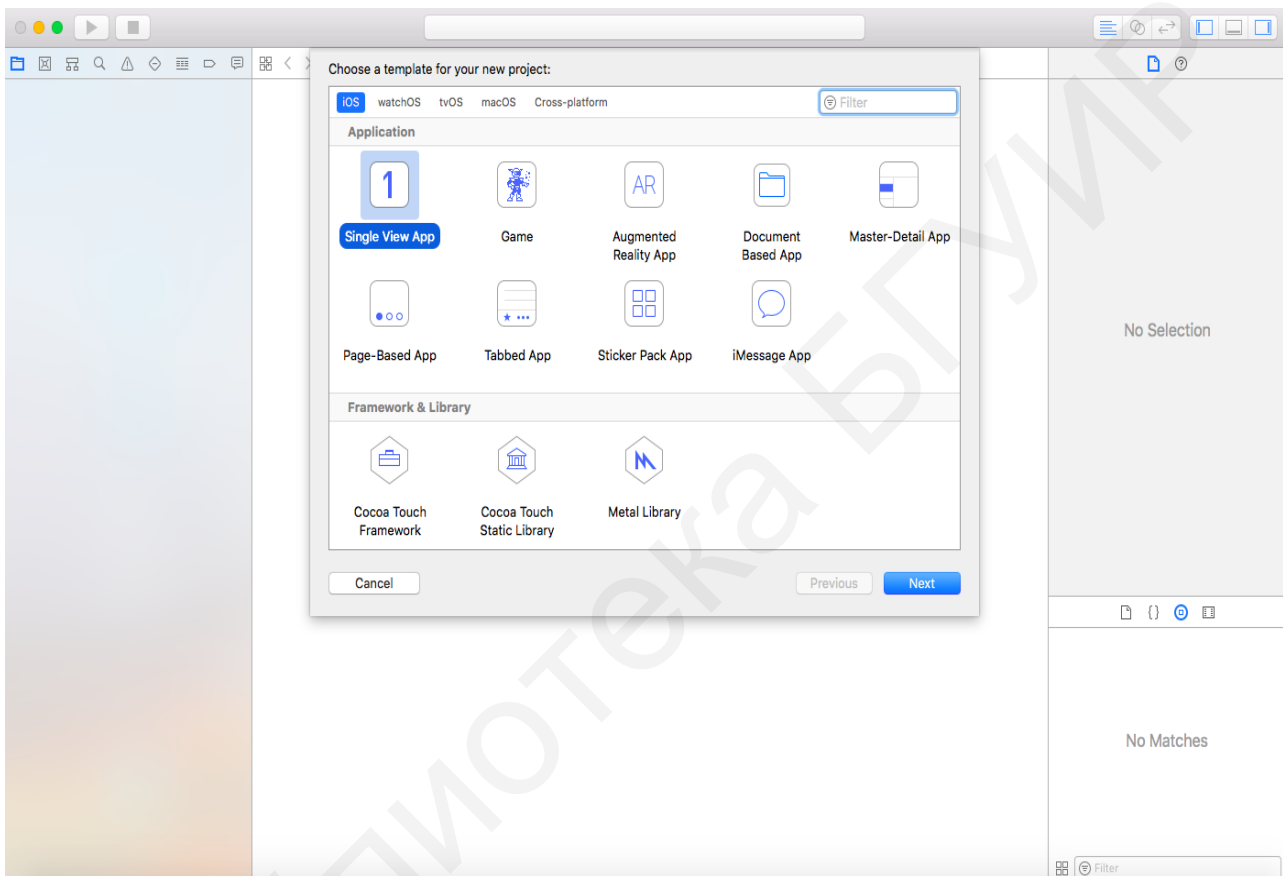


Рисунок 3.2 – Окно выбора шаблона приложения

7. Нажать кнопку **Next**.

8. Выбрать место для сохранения проекта (рисунок 3.4).

9. Нажать кнопку **Create**.

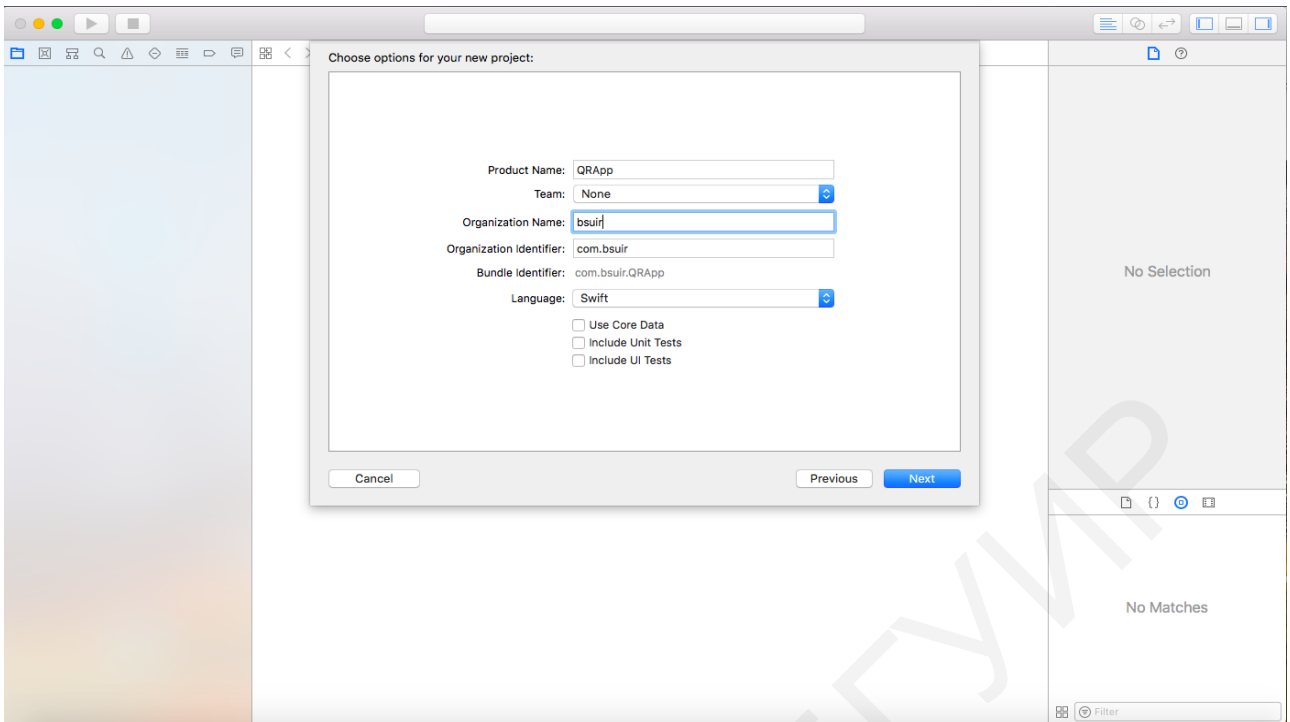


Рисунок 3.3 – Окно выбора опций приложения

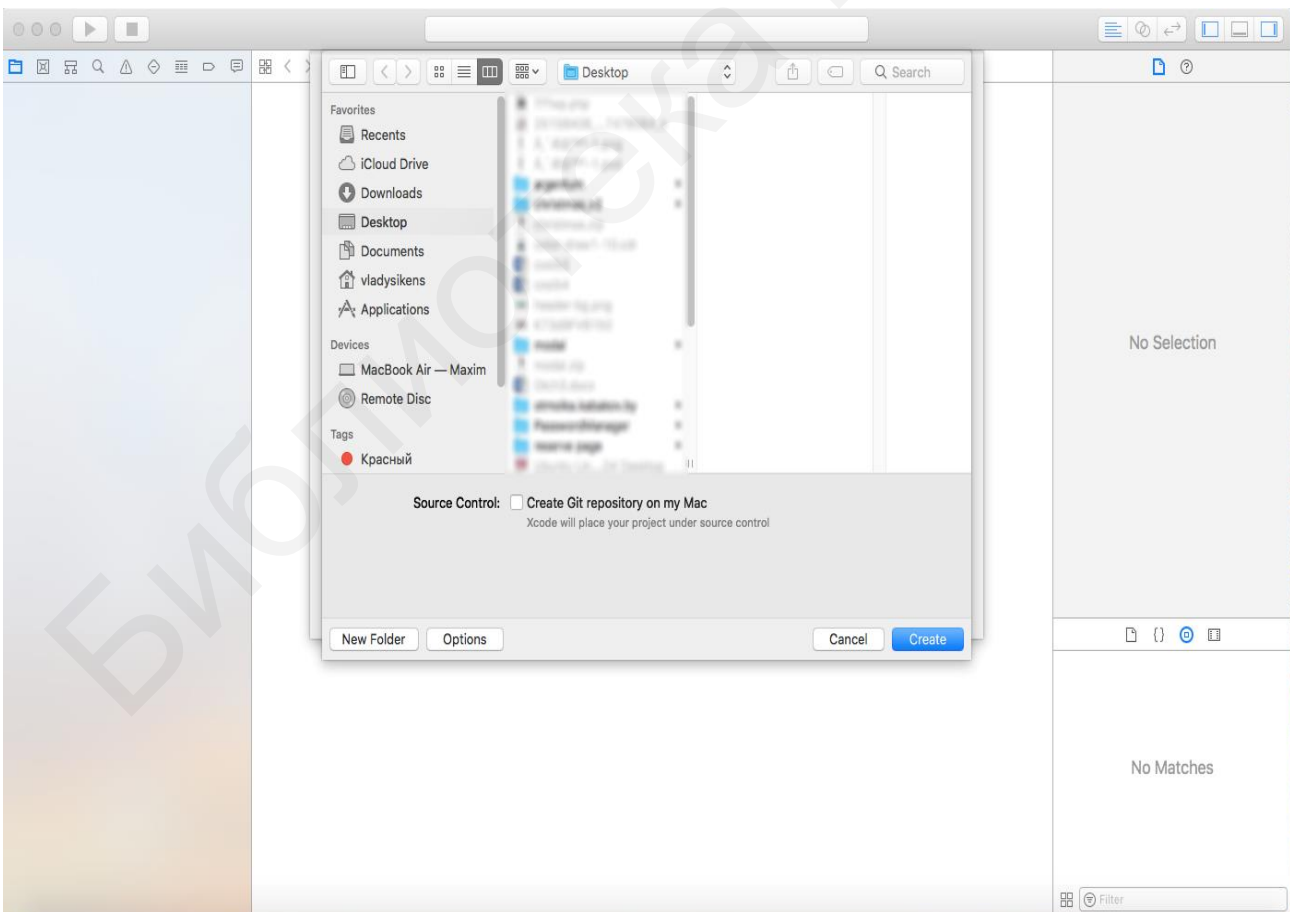


Рисунок 3.4 – Выбор места для сохранения проекта

После создания проекта откроется его главное окно (рисунок 3.5).

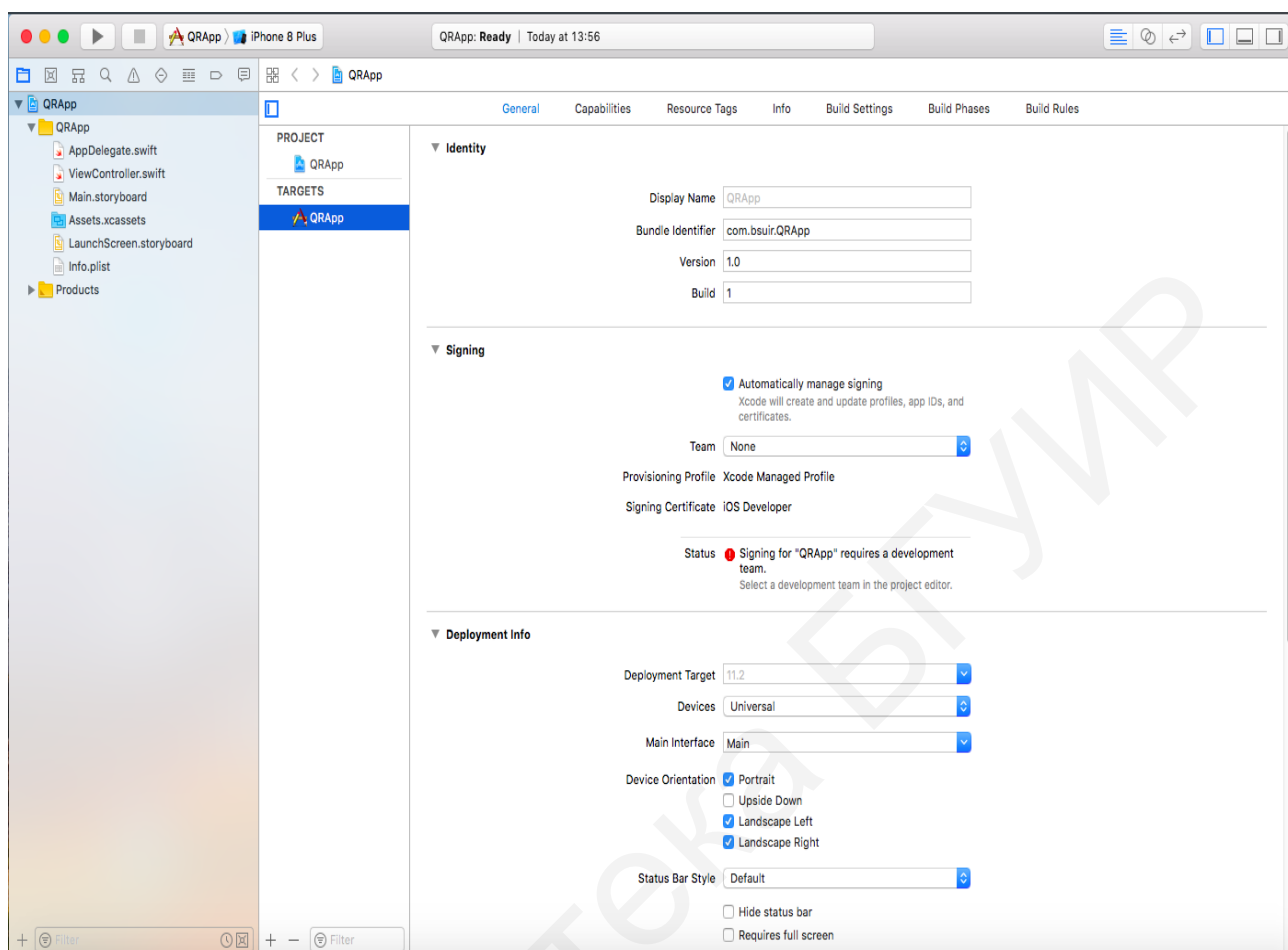


Рисунок 3.5 – Главное окно проекта

Функция сканирования штрих-кода добавлена в структуру AVFoundation, так как любое сканирование штрих-кода в iOS, включая сканирование QR-кода, полностью основано на захвате видео.

Приложение работает практически так же, как приложение для захвата видео, но без функции записи. При запуске приложение использует заднюю камеру iOS-устройства для определения QR-кода и автоматически распознает его.

Создание приложения можно начать с загрузки шаблона проекта, перейдя по ссылке <https://github.com/appcoda/QRCodeReader>.

Основной экран проекта связан с классом **QRCodeViewController**, а экран сканера – с классом **QRScannerController** (рисунок 3.6).

Метка в пользовательском интерфейсе используется для отображения декодированной информации QR-кода и связана со свойством **messageLabel** класса **QRScannerController**.

Для реализации функции сканирования QR-кода надо открыть файл **QRScannerController.swift** и импортировать фреймворк, учитывая структуру **AVFoundation**: `import AVFoundation`.

Протокол с расширением имеет вид

```
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

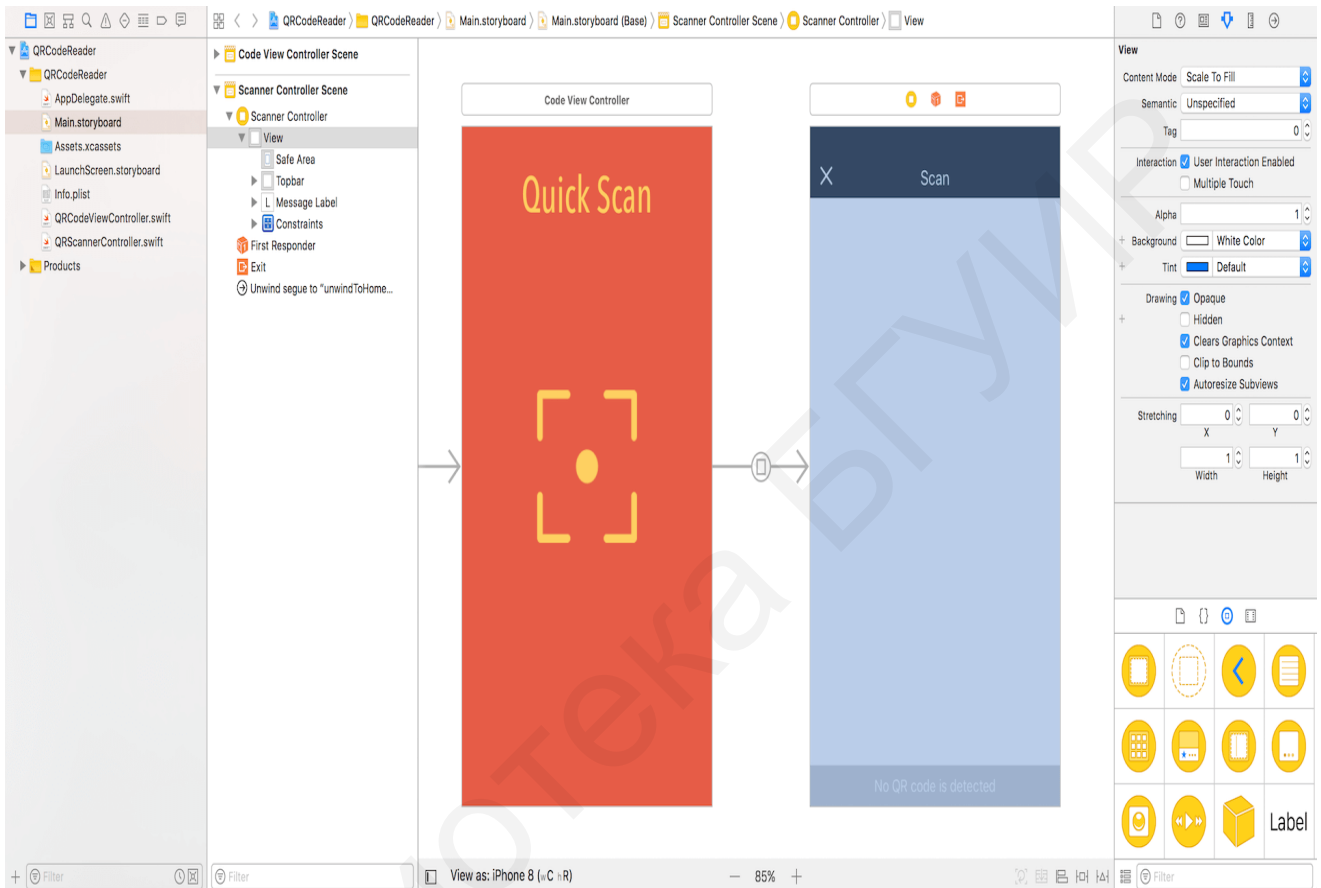


Рисунок 3.6 – Режим просмотра

Перед перемещением необходимо объявить переменные в классе **QRScannerController**:

```
var captureSession: AVCaptureSession?  
var videoPreviewLayer: AVCaptureVideoPreviewLayer?  
var qrCodeFrameView: UIView?
```

В метод **viewDidLoad** класса **QRScannerController** следует вставить следующий код:

```
let deviceDiscoverySession = AVCaptureDevice.DiscoverySession(deviceTypes:  
[.builtInDualCamera], mediaType: AVMediaType.video, position: .back)
```



```

guard let captureDevice = deviceDiscoverySession.devices.first else {
print("Failed to get the camera device")
return
}
do {
let input = try AVCaptureDeviceInput(device: captureDevice)
captureSession.addInput(input)
} catch {
print(error)
return
}

```

Класс **AVCaptureDevice.DiscoverySession** предназначен для поиска всех доступных устройств захвата, соответствующих определенному типу устройства. Приведенный код содержит действия для получения устройства, поддерживающего тип носителя **AVMediaType.video**.

Для захвата в режиме реального времени используется объект **AVCaptureSession**, необходимый для координации потока данных с устройства ввода видео на наш выход, и добавляется входное устройство видеозахвата.

В этом случае на выходе сеанса устанавливается объект **AVCaptureMetadataOutput**. Класс **AVCaptureMetadataOutput** является основной частью чтения QR-кода. Этот класс в сочетании с протоколом **AVCaptureMetadataOutputObjectsDelegate** используется для перехвата любых метаданных, найденных на устройстве ввода, например, QR-кода, захваченного камерой устройства, и перевода его в удобочитаемый для человека формат.

В блок **do** метода **viewDidLoad** записываются следующие строки кода:

```

let captureMetadataOutput = AVCaptureMetadataOutput()
captureSession.addOutput(captureMetadataOutput)

```

Затем добавляются строки кода, показанные ниже. При этом необходимо установить себя как делегат объекта **captureMetadataOutput**. Поэтому класс **QRReaderViewController** использует протокол **AVCaptureMetadataOutputObjectsDelegate**.

```

captureMetadataOutput.setMetadataObjectsDelegate (self, queue: DispatchQueue.main)
captureMetadataOutput.metadataObjectTypes = [AVMetadataObject.ObjectType.qr]

```

При захвате новых объектов метаданных они перенаправляются на объект делегата для дальнейшей обработки. В приведенном коде очередь отправки используется для выполнения методов делегата. Очередь отправки может быть последовательной или параллельной, но согласно документации Apple, она должна

быть последовательной. Таким образом, **DispatchQueue.main** используется для получения серийной очереди по умолчанию.

Свойство **metadataObjectTypes** содержит массив строк, идентифицирующих типы метаданных для приложения. Объект **AVMetadataObject.ObjectType.qr** применяется при сканировании QR-кода.

После установки и настройки объекта **AVCaptureMetadataOutput** надо отобразить видео, снятое камерой на экране устройства. Это можно сделать с помощью слоя **AVCaptureVideoPreviewLayer**, который на самом деле является **CALayer**. Этот слой предварительного просмотра используется совместно с сеансом захвата AV для отображения видео. Уровень предварительного просмотра добавляется как подуровень текущего вида.

Следующий код надо вставить в блок блокировки:

```
videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
videoPreviewLayer?.videoGravity = AVLayerVideoGravity.resizeAspectFill
videoPreviewLayer?.frame = view.layer.bounds
view.layer.addSublayer(videoPreviewLayer!)
```

Далее вызывается метод **startRunning** для сеанса захвата видео:

```
captureSession.startRunning()
```

После компиляции и запуска приложения на реальном устройстве iOS может появиться ошибка при выполнении сканирования:

*This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSCameraUsageDescription key with a string value explaining to the user how the app uses this data.*

Это означает, что iOS требует от разработчиков приложений получить разрешение пользователя, прежде чем разрешить доступ к камере. Для этого необходимо добавить в файл **Info.plist** ключ с именем **NSCameraUsageDescription**:

1. Открыть файл.
2. Щелкнуть правой кнопкой мыши по любой пустой области для добавления новой строки.
3. Установить ключ в «Конфиденциальность – описание использования камеры» и значение «Нам нужно получить доступ к вашей камере для сканирования QR-кода» (рисунк 3.7).

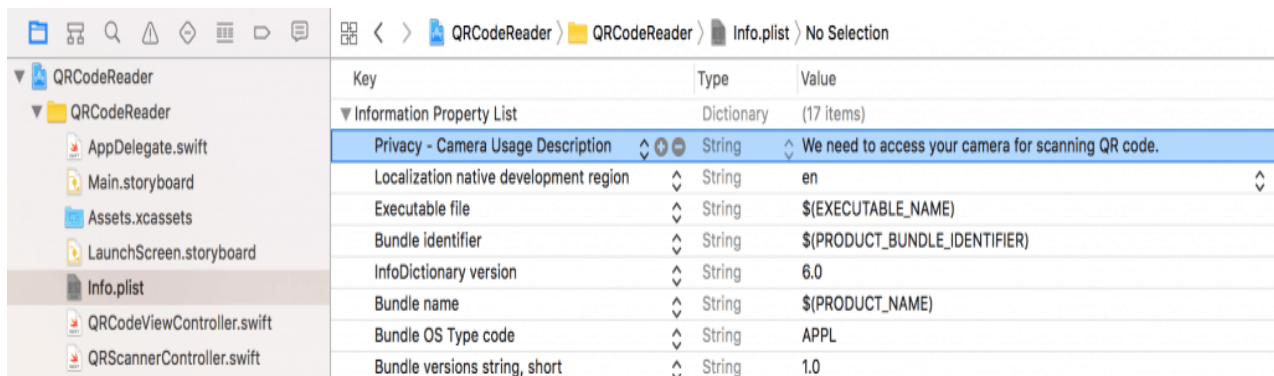


Рисунок 3.7 – Установка ключа

После завершения редактирования следует развернуть приложение и снова запустить его на реальном устройстве. При нажатии кнопки сканирования должна отображаться встроенная камера и запускаться видео. В этот момент ярлык сообщения и верхняя панель скрыты. Для исправления этого недостатка надо добавить следующую строку кода, выполнение которого приведет к перемещению поверх видео слоя метки сообщения и верхней панели.

```
view.bringSubview(toFront: messageLabel)
view.bringSubview(toFront: topbar)
```

После внесения изменений и перезапуска приложения на экране появится метка сообщения «*Нет QR-кода*».

В блок **do** метода **viewDidLoad** необходимо добавить следующий код:

```
qrCodeFrameView = UIView()
if let qrCodeFrameView = qrCodeFrameView {
    qrCodeFrameView.layer.borderColor = UIColor.green.cgColor
    qrCodeFrameView.layer.borderWidth = 2
    view.addSubview(qrCodeFrameView)
    view.bringSubview(toFront: qrCodeFrameView)
}
```

Переменная **qrCodeFrameView** невидима на экране, так как размер объекта **UIView** по умолчанию равен нулю. Поэтому при обнаружении QR-кода надо изменить его размер и превратить в зеленый квадрат.

При распознании объектом **AVCaptureMetadataOutput** QR-кода вызывается метод делегирования **AVCaptureMetadataOutputObjectsDelegate**:

```
optional func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput metadataObjects: [AVMetadataObject], from connection: AVCaptureConnection)
```

Без этого метода приложение не может перевести QR-код. Для захвата QR-кода и декодирования информации необходимо реализовать данный метод для выполнения дополнительной обработки объектов метаданных. Код метода делегирования **AVCaptureMetadataOutputObjectsDelegate**:

```
func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput metadataObjects:
[AVMetadataObject], from connection: AVCaptureConnection) {
    if metadataObjects.count == 0 {
        qrCodeFrameView?.frame = CGRect.zero
        messageLabel.text = "No QR code is detected"
        return
    }
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject
    if metadataObj.type == AVMetadataObject.ObjectType.qr {
        let barCodeObject=videoPreviewLayer?.transformedMetadataObject(for:metadataObj)
        qrCodeFrameView?.frame = barCodeObject!.bounds
        if metadataObj.stringValue != nil {
            messageLabel.text = metadataObj.stringValue
        }
    }
}
```

Второй параметр метода **MetadataObjects** – объект массива, который содержит все объекты прочитанных метаданных. Сначала надо убедиться, что этот массив не равен **null** и содержит хотя бы один объект. Иначе следует сбросить размер **qrCodeFrameView** до нуля и установить **messageLabel** в сообщение по умолчанию.

После того как объект метаданных найден, выполняется проверка, является ли он QR-кодом, и осуществляется поиск границ QR-кода. Данный код используется для настройки зеленого поля в целях выделения кода QR. При вызове метода **transformMetadataObject** в классе **viewPreviewLayer** визуальные свойства объекта метаданных преобразуются в координаты слоя и при этом находятся границы QR-кода для построения зеленой коробки.

Далее QR-код декодируется в удобочитаемую информацию. Доступ к декодированной информации можно получить, используя свойство **stringValue** объекта **AVMetadataMachineReadableCode**.

### 3.3 Результат выполнения лабораторной работы

После запуска приложения надо:

- 1) нажать кнопку сканирования;
- 2) навести устройство на QR-код.

Приложение обнаруживает код и декодирует информацию (рисунок 3.8).

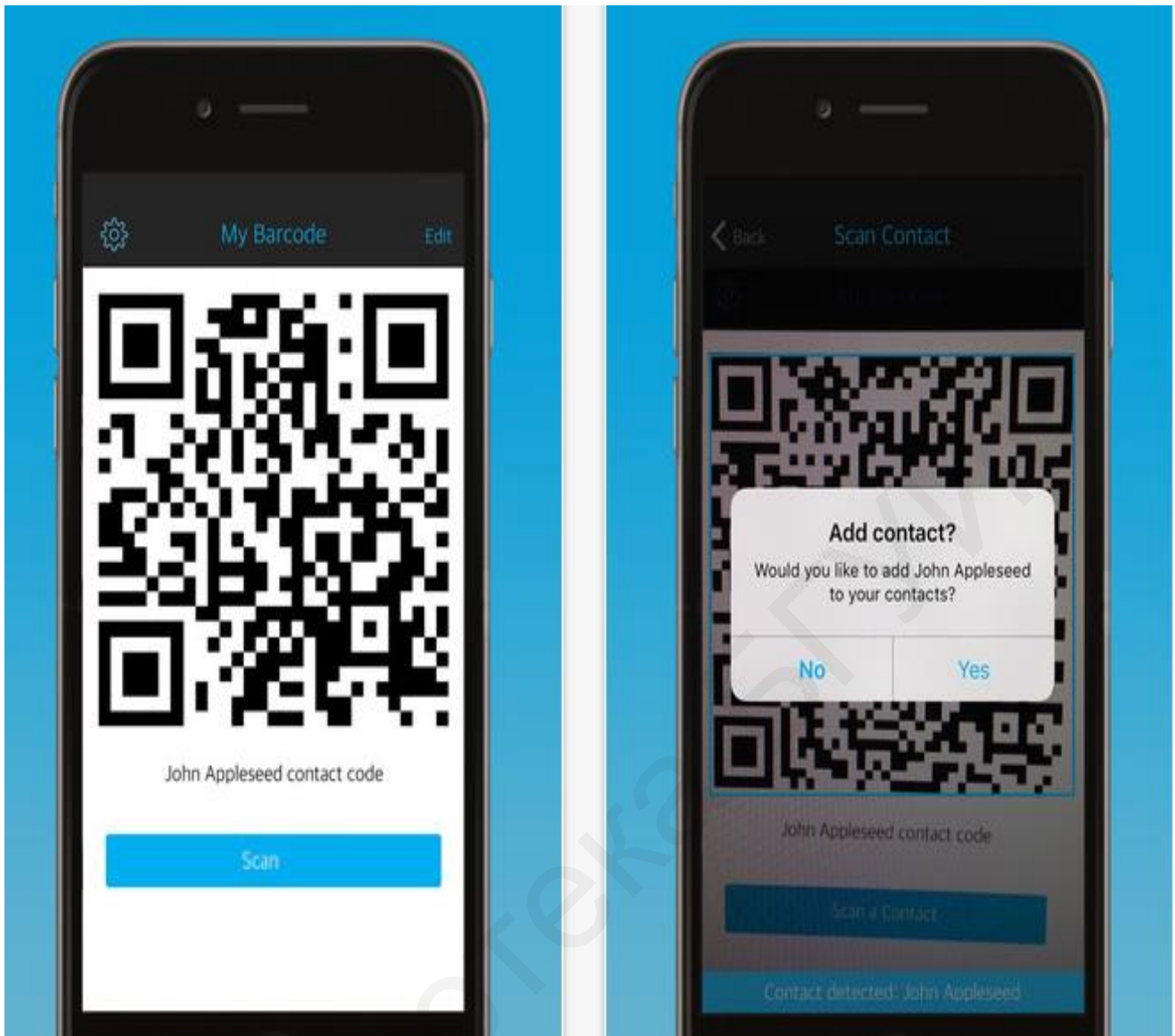


Рисунок 3.8 – Результат работы приложения

### **Выводы**

В результате выполнения лабораторной работы приложение должно считывать информацию, представленную в виде QR-кода, и отображать ее в виде текста.

## ЛАБОРАТОРНАЯ РАБОТА №4

### ПРИЛОЖЕНИЕ С ИСПОЛЬЗОВАНИЕМ ДОПОЛНЕННОЙ РЕАЛЬНОСТИ ARKIT

**Цель работы:** реализовать простое приложение, которое будет использовать фреймворк дополненной реальности **ARKit** (быстрый старт).

#### 4.1 Необходимое ПО

1. Установленная среда **Xcode** версии 9 и выше.
2. Смартфон **iPhone** или **iPad** на процессоре A9 и выше (для использования всех ключевых функций **ARKit** необходим процессор A9 и выше).
3. Аккаунт разработчика.

#### 4.2 Теоретические сведения

**ARKit** – инструмент, который умеет грамотно обрабатывать большое количество данных, полученных от устройства. Благодаря камере и датчикам движения фреймворк отслеживает движение, находит поверхности и определяет освещенность. После анализа данных формируется представление об окружающем мире в виде точек пересечения, координат поверхностей и положении камеры в пространстве.

Основной задачей **ARKit** является слежение за окружающим миром (World Tracking) для создания виртуальной модели реального мира. Фреймворк распознает особенности видеок кадров, отслеживает изменения их положения и сравнивает эту информацию с данными от датчиков движения. Результатом является виртуальная модель реального мира. Дополненная реальность **ARKit** имеет возможность распознавания плоских горизонтальных поверхностей: находит плоскости и сообщает об их расположении и размерах.

Слежение за окружающим миром требует анализа картинки, получаемой от камеры. Для достижения наилучшего результата необходимо хорошее освещение.

#### 4.3 Создание проекта

Для создания проекта необходимо выполнить следующие действия:

1. В верхнем меню нажать **File → New → Project** или вызвать меню создания нового проекта, используя «горячие» клавиши **cmd+Shift+N**.
2. Выбрать тип проекта **AugmentedRealityApp** (рисунок 4.1).
3. Нажать **Next**.

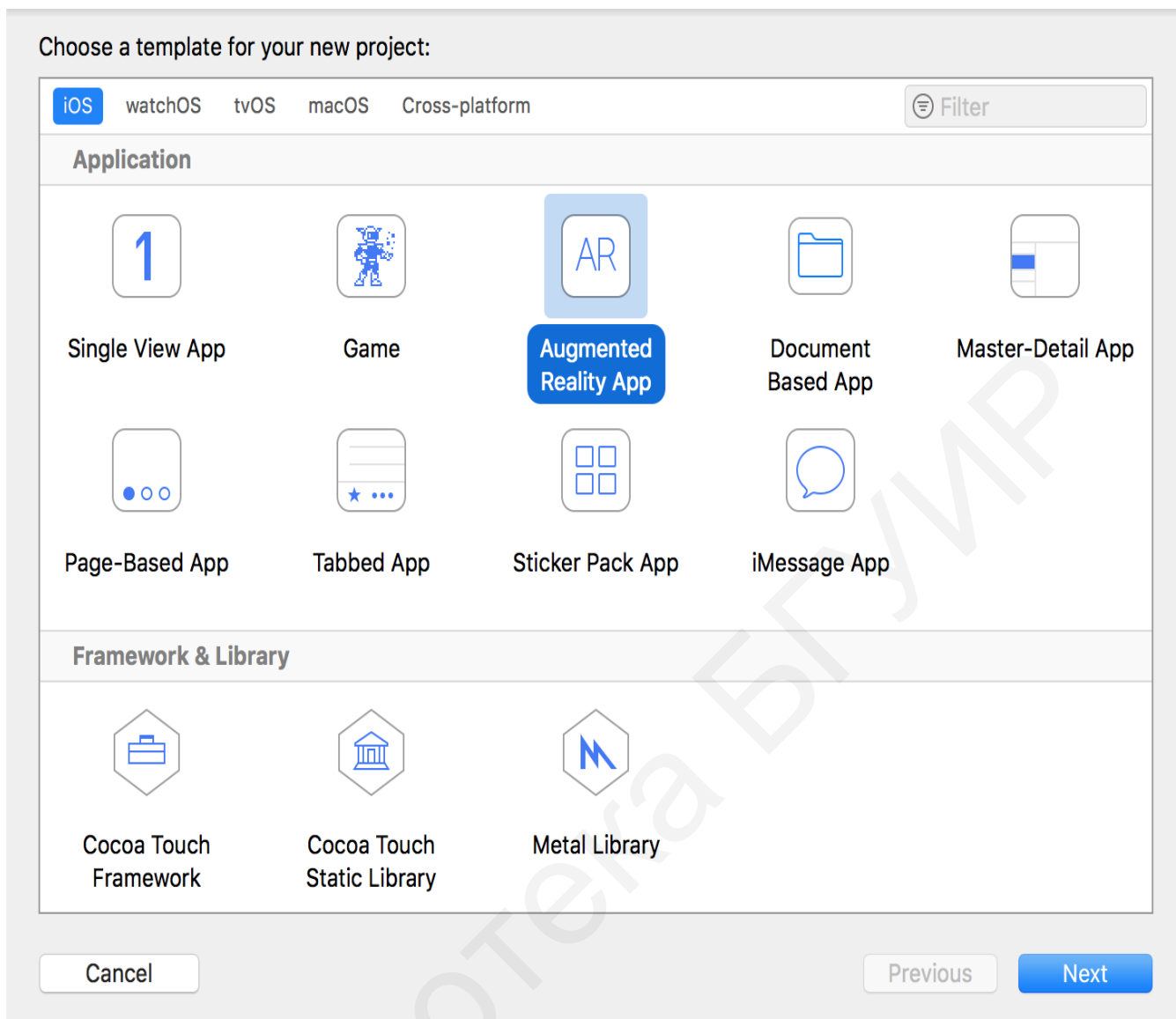


Рисунок 4.1 – Выбор типа проекта

4. Ввести название проекта, при этом необходимо выбрать команду разработки и указать **BundleID** – уникальный идентификатор приложения в AppStore (рисунок 4.2).

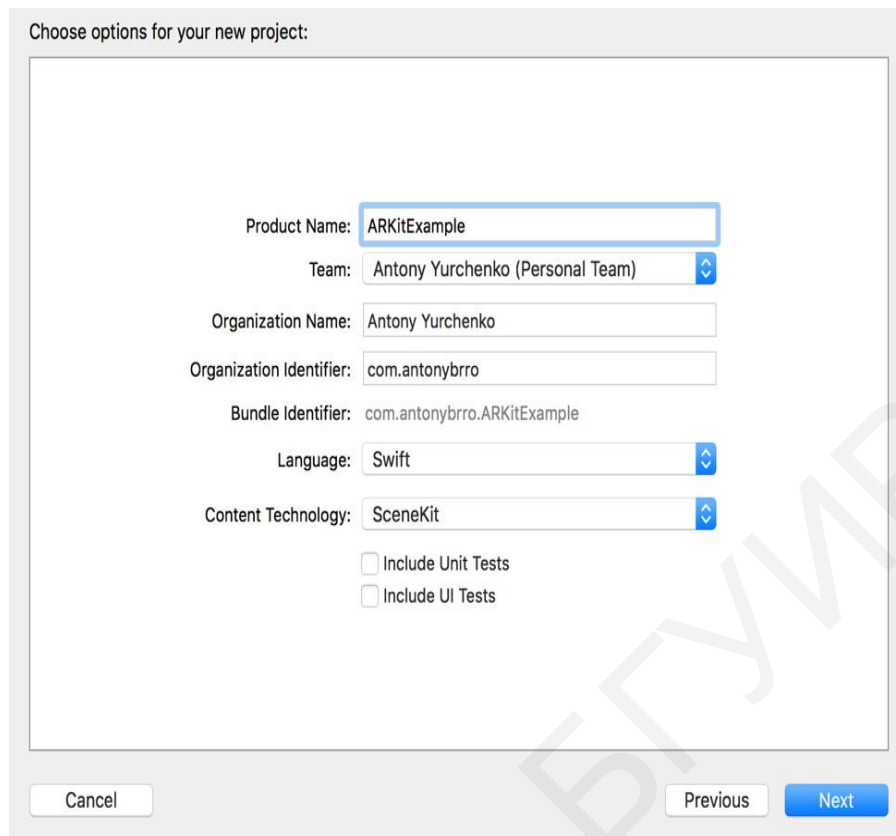


Рисунок 4.2 – Изменение опций нового проекта

5. Выбрать директорию для хранения проекта.
  6. Для запуска проекта подключить iPhone или iPad к Mac (рисунок 4.3).
- Для этого необходимо:
- в верхнем меню выбрать устройство,
  - нажать кнопку запуска приложения.

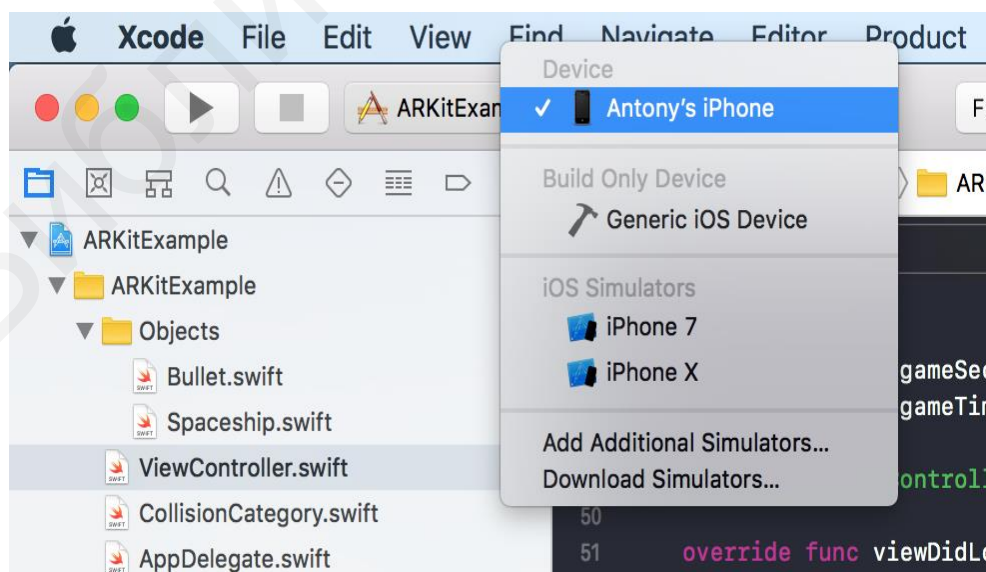


Рисунок 4.3 – Запуск проекта



7. После запуска проекта появится стандартная неподвижная виртуальная модель самолета, вокруг которой можно передвигаться в пространстве и наблюдать за моделью как за объектом реального мира (рисунок 4.3).



Рисунок 4.4 – Результаты выполнения работы

#### 4.4 Написание кода

Основой **ARKit** являются **ARSCNView** и **ARSKView**. Они служат для отображения **live**-видео и рендеринга **3D**- и **2D**-изображений и происходят от **SCNView** и **SKView**. Следовательно, **ARKit** не привносит особенностей в отображение данных. Поэтому движки для работы с **2D**- и **3D**-графикой неизменны.

**ARSCNView** и **ARSKView** содержат в себе класс **ARSession**, который имеет средства для работы с дополненной реальностью. Для запуска **ARSession** необходимо передать конфигурацию работы сессии. Для этого следует:

1. Выбрать тип конфигурации, определяющий стиль и качество работы AR, которое может быть достигнуто:

- на устройствах с процессором A9 и новее, где можно использовать **ARWorldTrackingConfiguration** и все возможности нового фреймворка, что поможет расположить виртуальные объекты с максимальной точностью;
- на остальных устройствах, поддерживающих **ARKit**, где будет доступна только **AROrientationTrackingConfiguration**. Базовый класс предоставляет информацию о движении устройства в пространстве, но не строит виртуаль-

ные модели. Необходимый эффект не будет достигнут, так как возможность фиксации виртуальных объектов относительно объектов реального мира будет недоступна.

2. Открыть файл **ViewController.swift** и удалить содержимое класса **ViewController**.

3. Добавить **IBOutlet** для **ARSCNView**, которая выполняет рендеринг виртуальных объектов. Для этого необходимо:

- открыть **Main.storyboard**;
- в меню сверху выбрать **View** → **AssistantEditor** → **ShowAssistantEditor** (**Opt+cmd+Enter**);
- выбрать **ARSCNView**;
- перетащить **IBOutlet** в класс (рисунок 4.5).

4. Ввести данные **IBOutlet** (рисунок 4.6).

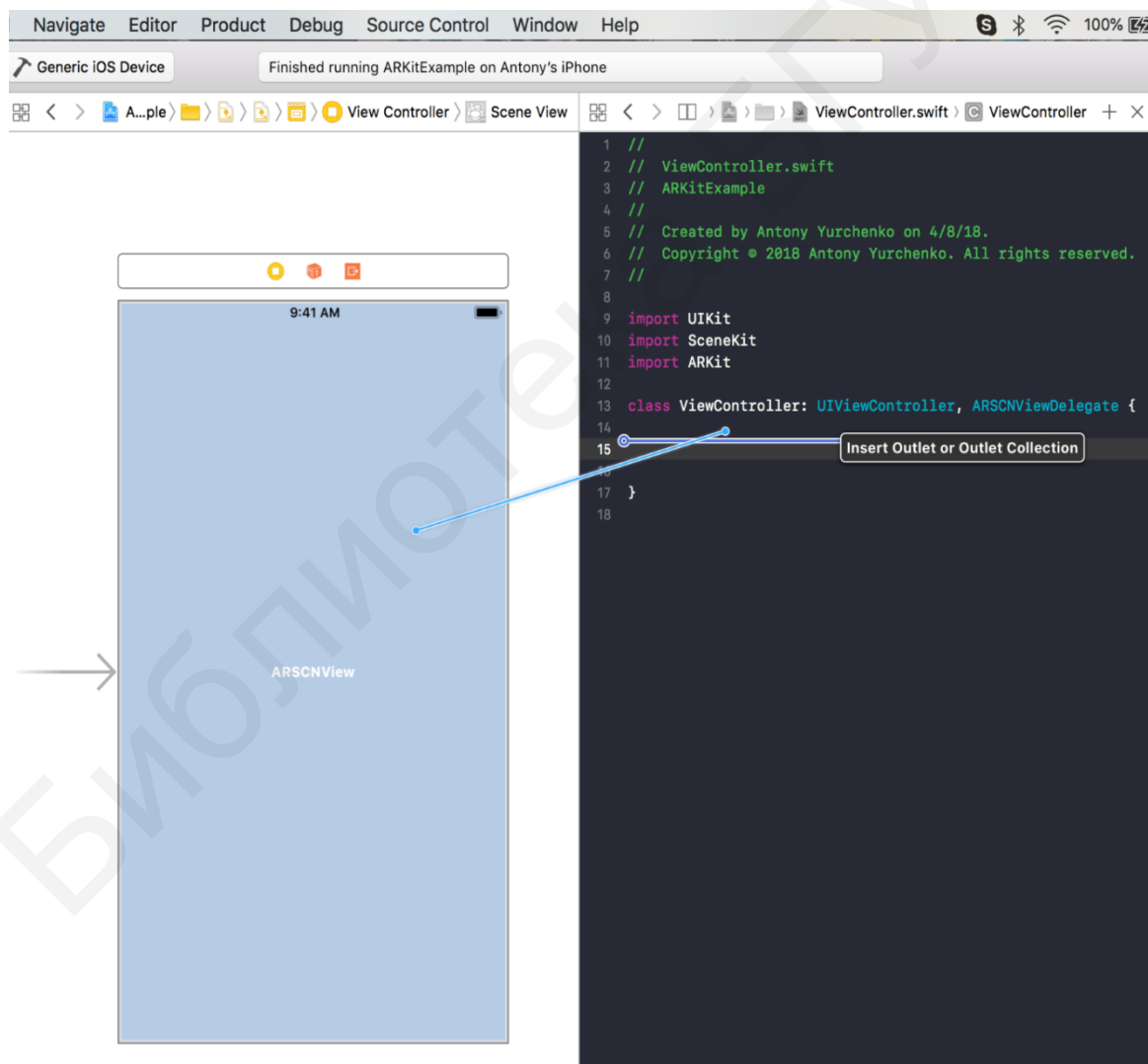


Рисунок 4.5 – Добавление **IBOutlet** в класс

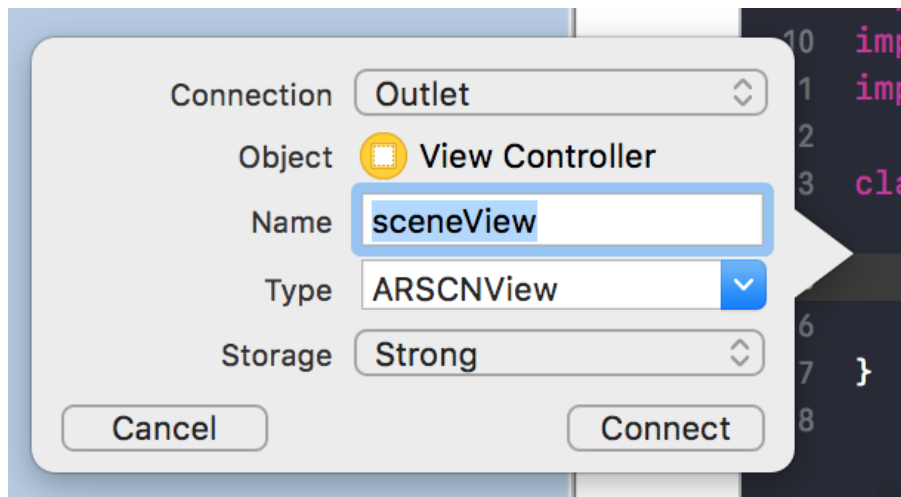


Рисунок 4.6 – Ввод данных

В классе **ViewController** должен появиться **IBOutlet** (рисунок 4.7).

```
class ViewController: UIViewController, ARSCNViewDelegate {  
  
    @IBOutlet var sceneView: ARSCNView!  
  
}
```

Рисунок 4.7 – IBOutlet

5. После выбора типа конфигурации создать ее экземпляр, произвести настройку и запустить сессию (рисунок 4.8).

```
override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
  
    // Create a session configuration  
    let configuration = ARConfiguration.isSupported ?  
        ARWorldTrackingConfiguration() : AROrientationTrackingConfiguration()  
  
    // Run the view's session  
    sceneView.session.run(configuration)  
}
```

Рисунок 4.8 – Команда запуска сессии

6. Сделать контроллер делегатом контактов физического мира и вывести статистику. Настроить запуск и паузу сессии при появлении и скрытии контроллера (рисунок 4.9).

```
// MARK: - ViewController life cycle

override func viewDidLoad() {
    super.viewDidLoad()

    sceneView.scene.physicsWorld.contactDelegate = self

    // Show statistics such as fps and timing information
    sceneView.showsStatistics = true

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(tapAction))
    sceneView.addGestureRecognizer(tapGestureRecognizer)

    spaceshipCount = 0
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    // Create a session configuration
    let configuration = ARConfiguration.isSupported ? ARWorldTrackingConfiguration()
        : AROrientationTrackingConfiguration()

    // Run the view's session
    sceneView.session.run(configuration)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // Pause the view's session
    sceneView.session.pause()
    stopGame()
}
```

Рисунок 4.9 – Настройка запуска и паузы сессии

7. Создать физический объект. Для этого необходимо:

- задать фигуру определенного размера;
- создать фигуру с физическими свойствами для контролирования контактов с другими объектам;

– создать физическое тело для описания поведения объекта при соприкосновении;

– задать текстуры.

На рисунках 4.10 и 4.11 приводятся примеры реализации классов патрона в виде шара и космического корабля в виде куба с нужными текстурами.

```
import UIKit
import SceneKit

final class Bullet: SCNNode {

    private static let sphereRadius: CGFloat = 0.025

    override init() {
        super.init()
        initialization()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        initialization()
    }

    private func initialization() {
        let arKitBox = SCNSphere(radius: Bullet.sphereRadius)
        self.geometry = arKitBox
        let shape = SCNPhysicsShape(geometry: arKitBox, options: nil)
        self.physicsBody = SCNPhysicsBody(type: .dynamic, shape: shape)
        self.physicsBody?.isAffectedByGravity = false

        self.physicsBody?.categoryBitMask = CollisionCategory.bullet.rawValue
        self.physicsBody?.contactTestBitMask = CollisionCategory.spaceship.rawValue

        // add texture
        let material = SCNMaterial()
        material.diffuse.contents = UIImage(named: "art.scnassets/bullet.jpg")
        self.geometry?.materials = [material]
    }
}
```

Рисунок 4.10 – Пример реализации класса патрона в виде шара

```

import UIKit
import SceneKit

final class Spaceship: SCNNode {

    private static let boxSide: CGFloat = 0.1

    override init() {
        super.init()
        initialization()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        initialization()
    }

    private func initialization() {
        let logo = SCNBox(width: Spaceship.boxSide,
                          height: Spaceship.boxSide,
                          length: Spaceship.boxSide,
                          chamferRadius: 0)

        self.geometry = logo
        let shape = SCNPhysicsShape(geometry: logo, options: nil)

        self.physicsBody = SCNPhysicsBody(type: .dynamic, shape: shape)
        self.physicsBody?.isAffectedByGravity = false

        self.physicsBody?.categoryBitMask = CollisionCategory.spaceship.rawValue
        self.physicsBody?.contactTestBitMask = CollisionCategory.bullet.rawValue

        // add texture
        let material = SCNMaterial()
        material.diffuse.contents = UIImage(named: "art.scnassets/spaceship.jpg")
        self.geometry?.materials = Array(repeating: material, count: 6)
    }
}

```

Рисунок 4.11 – Пример реализации класса патрона в виде космического корабля

8. Определить тип объекта при контакте. Для определения использовать структуру **CollisionCategory** (рисунок 4.12).

```

struct CollisionCategory: OptionSet {

    let rawValue: Int

    static let bullet = CollisionCategory(rawValue: 1 << 0)
    static let spaceship = CollisionCategory(rawValue: 1 << 1)

}

```

Рисунок 4.12 – Структура **CollisionCategory**

9. В классе **ViewController** реализовать делегат, отвечающий за обработку контактов виртуальных объектов, которые могут взаимодействовать при контакте (рисунок 4.13).

```

extension ViewController: SCNPhysicsContactDelegate {

    func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact: SCNPhysicsContact) {
        guard let nodeABitMask = contact.nodeA.physicsBody?.categoryBitMask,
              let nodeBBitMask = contact.nodeB.physicsBody?.categoryBitMask,
              nodeABitMask & nodeBBitMask == CollisionCategory.spaceship.rawValue & CollisionCategory.bullet.rawValue else {
            return
        }

        contact.nodeB.removeFromParentNode()
        spaceshipCount -= 1

        if spaceshipCount == 0 {
            DispatchQueue.main.async {
                self.stopGame()
            }
        }

        DispatchQueue.main.asyncAfter(deadline: .now() + 0.5, execute: {
            contact.nodeA.removeFromParentNode()
        })
    }
}

```

Рисунок 4.13 – Реализация делегатов в классе **ViewController**



10. В класс **ViewController** добавить два метода: для добавления кораблей и для выстрела (рисунки 4.14, 4.15).

```
private func addSpaceship() {
    guard let currentFrame = sceneView.session.currentFrame else {
        return
    }

    let spaceship = Spaceship()
    sceneView.scene.rootNode.addChildNode(spaceship)

    var translation = matrix_identity_float4x4
    translation.columns.3.z = ViewController.cameraToSpaceshipSpace
    spaceship.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)

    spaceshipCount += 1
    if spaceshipCount == ViewController.spaceshipMaxCount {
        startGame()
    }
}
```

Рисунок 4. 14 – Метод для добавления кораблей

```
private func shoot() {
    let bullet = Bullet()

    let (direction, position) = cameraVector
    bullet.position = position

    let bulletDirection = direction
    bullet.physicsBody?.applyForce(bulletDirection, asImpulse: true)
    sceneView.scene.rootNode.addChildNode(bullet)
}
```

Рисунок 4.15 – Метод для добавления выстрела



11. С помощью расширения класса **ViewController** реализовать логику игры (рисунок 4.16).

```
// MARK: - Game logic

extension ViewController {

    fileprivate func startGame() {
        state = .shooting

        gameTimer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true, block: { [weak self] _ in
            self?.gameSeconds += 1
            DispatchQueue.main.async {
                self?.configureTimeLabel()
            }
        })
    }

    fileprivate func stopGame() {
        state = .placing

        gameTimer?.invalidate()
        gameTimer = nil

        gameSeconds = 0
        spaceshipCount = 0

        configureTimeLabel()
    }

    fileprivate func configureTimeLabel() {
        timeLabel.isHidden = self.gameSeconds == 0

        let seconds = self.gameSeconds % 60
        let minutes = (self.gameSeconds / 60) % 60

        timeLabel.text = String(format: "%02d:%02d", minutes, seconds)
    }
}
}
```

Рисунок 4.16 – Реализация логики игры

12. Собрать проект и запустить приложение на устройстве (рисунок 4.17).

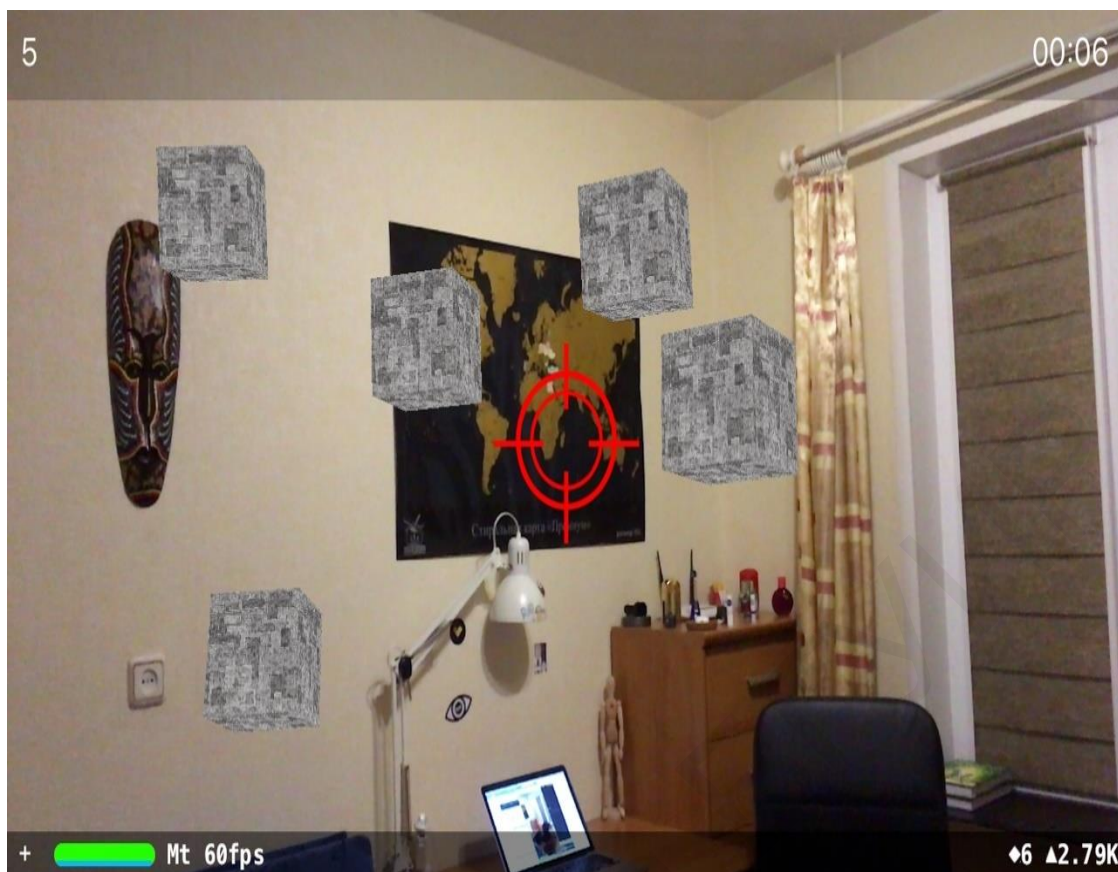


Рисунок 4.17 – Результаты работы проекта

**Примечание** – Репозиторий с готовым проектом из данной лабораторной работы можно скачать на сайте: <https://github.com/antonybro/ARKitExample>. Официальная документация Apple ARKit: <https://developer.apple.com/documentation/arkit>. ARKit демо-проекты, рекомендуемые к ознакомлению: <https://habrahabr.ru/post/336964/>.

### **Выводы**

В результате выполнения лабораторной работы должно быть разработано мобильное приложение для работы с дополненной реальностью ARKIT.

## ЛАБОРАТОРНАЯ РАБОТА №5

### КРОССПЛАТФОРМЕННАЯ ИГРА DOODLE JUMP

**Цель работы:** научиться разрабатывать приложение с фреймворком PointJS.

#### 5.1 Необходимое ПО

1. Установленная среда разработки **IntelXDK** (официальный сайт <https://software.intel.com/en-us/xdk>).
2. Игровой движок **PointJS** (<https://mult-uroki.ru/pointjs>).
3. **HTML5**.
4. **JavaScript**.

#### 5.2 Порядок выполнения лабораторной работы

##### 5.2.1 Создание нового проекта

Сначала следует создать новый проект (рисунок 5.1):

1. Запустить среду IntelXDK.
2. В открывшемся окне внизу нажать кнопку **Startnewproject**.
3. Во вкладке **Templates** выбрать **Standart HTML5**.
4. Нажать кнопку **Create**.

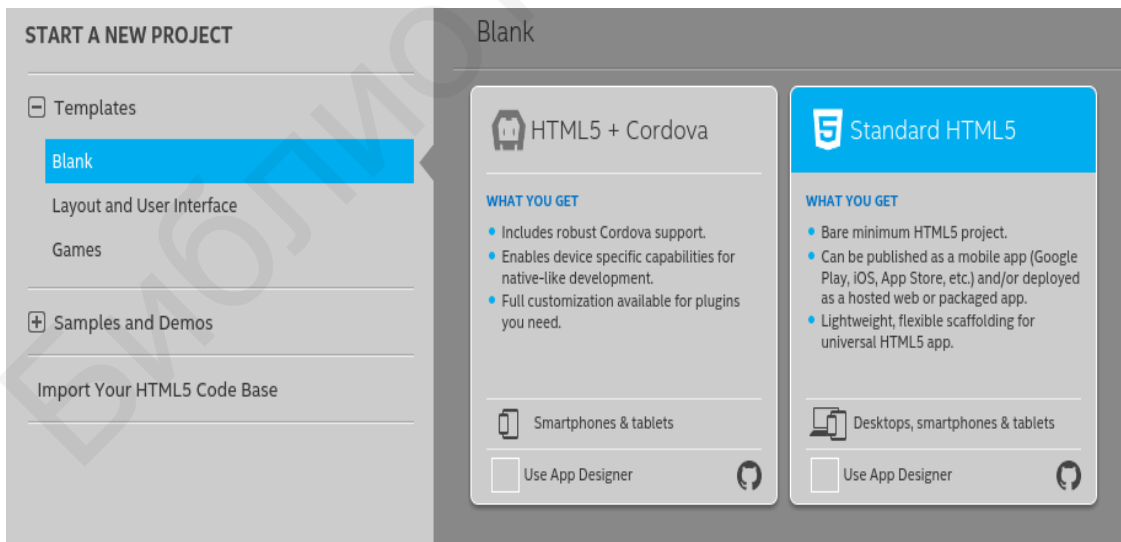


Рисунок 5.1 – Создание нового проекта

## 5.2.2 Настройка проекта

После создания проект следует настроить (все настройки логично понятны):

1. Щелкнуть по синей папке в левом верхнем углу окна проекта.
2. Выбрать платформу Android (рисунок 5.2).

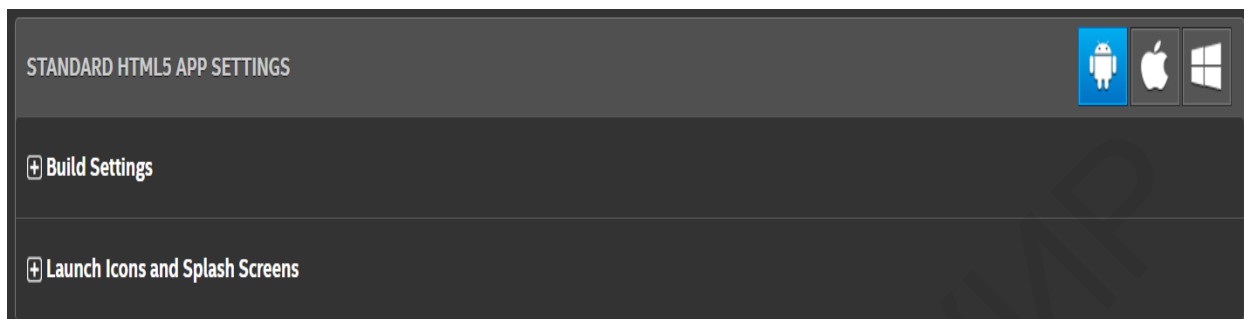


Рисунок 5.2 – Выбор платформы Android

## 5.2.3 Подготовка файлов проекта

Затем необходимо подготовить файлы проекта. Для этого следует:

1. Перейти на вкладку **DEVELOP**.
2. В появившемся слева списке файлов проекта щелкнуть правой кнопкой мыши по **www** (рабочая папка).
3. Нажать кнопку **Показать в проводнике**.
4. Удалить все.
5. Скопировать движок и файл **index.html** (рисунок 5.3).

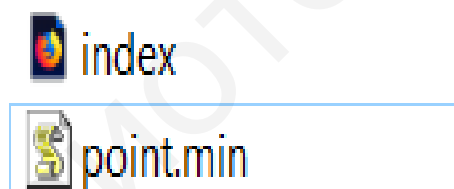


Рисунок 5.3 – Настройка проекта

## 5.2.4 Создание файлов игры

### 5.2.4.1 Создание файла Game.js

Код файла **Game.js** следующий:

```
var pjs = new PointJS('2d', 400, 400);  
pjs.system.initFullPage();
```

```
var vector = pjs.vector;  
var log = pjs.system.log;
```

```

var game = pjs.game;
var point = vector.point;
var size = vector.size;
var camera = pjs.camera;
var brush = pjs.brush;
var OOP = pjs.OOP;
var math = pjs.math;

var touch = pjs.touchControl;
touch.initTouchControl();

var width = game.getWH().w; // ширина экрана после разворачивания игры
var height = game.getWH().h; // высота экрана после разворачивания игры
// пересчет размеров объектов с учетом коэффициента единого размера
var del = height / 1000 / 5;
var dt; // записывается delta-time для плавности движения в игре
var score = 0; // счет игрока во время прохождения игры
var levelScore = 0; // счет, набравшийся за уровень, после проигрыша
var tmpScore = localStorage.getItem('score'); // лучший рекорд в oldScore как счет из
локального хранилища (если есть)
var oldScore = tmpScore ? tmpScore : 0;

```

#### 5.2.4.2 Создание файла grid.js

Файл используется для реализации фона «тетрадный лист»:

```

var sizeX = 200 * del; // размеры ячейки с учетом коэффициента del
var sizeY = 200 * del;
var drawGrid = function () { // функция отрисовки
var x = width / sizeX; // определение количества повторений клетки по X и Y
y = height / sizeY;
OOP.forXY(x, y, function (x, y) { // отрисовка прямоугольника
brush.drawRectS({ // первые два аргумента повторения по X и Y, затем функция
x : (sizeX + 2)*x, // позиция по X
y : (sizeY + 2)*y, // позиция по Y
fillColor : 'white', // цвет заливки
w : sizeX, h : sizeY // ширина и высота
});
});
};

```

### 5.2.4.3 Создание файла menu.js

Для создания меню используется файл **menu.js** с кодом:

```
// создание объекта, по нажатию на который игрока перекинет в игру
var newGame = game.newTextObject({
    text : 'Новая игра', // надпись на объекте
    font : 'serif', // шрифт надписи
    size : 300*del, // размер шрифта (помним про del)
    color : '#363636' // цвет текста
});
//позиционирование объекта относительно экранных координат центром в центр
newGame.setPositionCS(point(width/2, height/2));
var drawMenu = function () { // функция для отрисовки и обработки
newGame.draw(); // рисование объекта, созданного выше
brush.drawTextLinesS({ // рисование рекордов в две строки
x : newGame.x, // выравнивание текста по границе надписи "Новая игра"
y : newGame.y + newGame.h, // установка верхней границы отрисовки текста
lines : ['Рекорд: '+Math.abs(oldScore), 'За уровень: '+Math.abs(levelScore)],
font : 'serif', // шрифт
size : 250*del, // размер
color : '#363636' // цвет
});
// после создания ссылки на объект touch выполняется проверка выбора "Новой
игры"
if (touch.isPeekObject(newGame)) { // выбор "Новой игры"
createLevel(10); // создание уровня из начальных десяти блоков
return game.setLoop('game'); // выход из текущего игрового цикла в цикл 'game'
}
};
```

### 5.2.4.4 Создание файла pl.js

Код данного файла:

```
var pl = game.newImageObject({ // создание объекта плеера
    x : 0, y : 0, // начальная позиция (не требуется)
    w : 744 * del, h : 670 * del, // размеры 744 на 670, потом уменьшается
    file : 'img/pl.png' // путь к картинке
});
var dy = 2 * del; // сила, движущая объект вниз или вверх
var dmax = 50 * del; // максимальное ускорение
var dx = 0; // сила, движущая объект влево или вправо
var drawPlayer = function () { // функция отрисовки персонажа
if (pl.y < score) { // подсчет счета (вверх ось Y уменьшается!)
    score = Math.ceil(pl.y);
}
```

```

    }
    pl.draw(); // рисование персонажа
    pl.move(point(0, dy*dt)); // его движение вниз или вверх с учетом delta-time
    dy+=dy <dmax?del*dt:0; //реализация гравитации: ускорение,
        //если максимум скорости вниз не достигнут
    if (touch.isDown()) { // если игрок нажимает на экран
    if (touch.getPositionS().x > width/2) // нажатие по правой области
        //экрана (более половины ширины экрана)
        dx = 30*del; // движение вправо: скорость движения положительна
    else// иначе
        dx = -30*del; // движение влево: скорость движения отрицательна
    // определение направления движения объекта
    if (dx >0) { // движение вправо
        pl.setFlip(0, 0); //сброс зеркалирования (по умолчанию картинка «смотрит» вправо)
        if (pl.x > width) { // выход за пределы экрана
            pl.x = -pl.w; // телепортация объекта ниже
        }
    }
    elseif (dx <0) { // при движении влево
        pl.setFlip(1, 0); // зеркалирование картинки
        if (pl.x+pl.w <0) { // выход за пределы экрана
            pl.x = width; // телепортация объекта ниже
        }
    }
    pl.move(point(dx*dt, 0)); // движение объекта влево или вправо
}
// уменьшение счета при поднятии объекта вверх или его падении (спуск ниже
//максимальной позиции данного уровня – игрок проиграл)
if (pl.getPositionS().y > score + 5000 * del) {
    levelScore = score; // установка счета уровня
    if (score < oldScore) { // текущий счет больше предыдущего
        oldScore = score; // установка нового рекорда
        localStorage.setItem('score', score); // сохранение рекорда
    }
    return game.setLoop('menu'); // выход в меню при проигрыше
}
// при отрисовке персонажа надо следить за ним камерой
camera.moveTimeC(vector.pointPlus(point(width/2, pl.getPositionC().y), point(0, -
500*del)), 10);
};

```

### 5.2.4.5 Создание файла blocks.js

Код файла **blocks.js**:

```
//пустой массив с блоками для расстояний между последним и новым блоком
var blocks=[];
    dy2 = false;
var createBlock =function () {// создание одного блока (добавляет после послед-
него)
// установка начальной позиции первого блока, если последний блок не определен,
if (dy2 === false) {
    dy2 = height - 60*del*4; // расположим блок почти внизу экрана
    } else { // если позиция имеется, к ней рандомно прибавляется половина
высоты прыжка (получено экспериментально)
    dy2 = blocks[blocks.length-1].y - 500*del -
    math.random(500*del, 800*del); // рандом
    }
blocks.push(game.newImageObject({ // новый блок в массив
w :200 * del*4, h : 60 * del*4, // размеры
file :'img/block.png', // картинка для блока
x:math.random(0,width-200*del*4), //случайное расположение по горизонтали
y : dy2, // по высоте используется подсчитанное ранее значение
}));
});
// определение переменной для хранения последнего блока, который последним
// был перемещен наверх
var oldBlock;
// функция создания уровня (рассмотрена ранее: кнопка "Новая игра")
var createLevel = function (i) { // передача количества блоков для игры
    pl.y = 0; //так как создается уровень сначала, то игрок ставится в нуль
    pl.x = 0;
    score = 0; // обнуление текущего счета
    OOP.forInt(i, function () { // запуск цикла по созданию блоков
        createBlock(); // вызов функции нужное количество раз
    });
// запоминание последнего созданного блока ( самый верхний ВАЖНО)
    oldBlock = blocks[blocks.length-1]
};
// чтобы не создавать новые блоки следует: если блок окажется сильно внизу, //
перекинуть его наверх,
varrePositionBlock = function (e1) { // определение блока для перекидывания
// рандомная генерация позиции блока по горизонтали
var x=math.random(0,width-200*del*4),
y=oldBlock.y-500*del-// oldBlock последний телепортированный наверх блок
```



```

math.random(500*del, 800*del); // рандом, прибавляемый к высоте блока
el.setPosition(point(x, y)); // перепозиционирование блока наверх
oldBlock=el; // блок выше последнего становится последним (самым высоким)
};
// функция отрисовки всех блоков и проверки столкновения с ними плеера
var drawLevel = function () {
  OOP.forArr(blocks, function (el) { // цикл по блокам
    // проверка, не оказался ли блок сильно внизу
    if (camera.getPosition().y + height + 2000*del < el.y) { // если оказался
      rePositionBlock(el); // его перепозиционирование
    }
    el.draw(); // отрисовка
    // проверка на столкновение с игроком: учитывается падение вниз
    if (pl.isStaticIntersect(el.getStaticBox()) && dy >0) { // столкновение
    // проверка нижней позиции игрока: выше ли она одной трети от верха блока,
    // чтобы отталкивался он только тогда, когда его ноги выше центра
    if (pl.y+pl.h < el.y+el.h/3)
      dy = -50*del; // установка силы, двигающей объект
    }
  });
};
};

```

### 5.2.5 Подключение файлов игры

Осталось подключить все файлы и записать функции в **index.html**.

```

<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <meta name="viewport" content="width=device-width, user-scalable=no"/>
  <title>PointJS Game</title>
</head>
<body>
<script type="text/javascript" src="point.min.js"></script>
<script type="text/javascript" src="game.js"></script>
<script type="text/javascript" src="blocks.js"></script>
<script type="text/javascript" src="grid.js"></script>
<script type="text/javascript" src="menu.js"></script>
<script type="text/javascript" src="pl.js"></script>
<script type="text/javascript">

```

```

game.newLoop('menu', function () {
  dt = game.getDT(20);

```

```

    game.fill('#D9D9D9');
    drawGrid();
    drawMenu();
});

game.newLoop('game', function () {
    dt = game.getDT(20);
    game.fill('#D9D9D9');
    drawGrid();
    drawLevel();
    drawPlayer();

    brush.drawTextS({
        x : 10, y : 10,
        text : 'Счет: '+Math.abs(score),
        size : 300*del,
        color : '#515151',
        font : 'serif'
    });
});
game.startLoop('menu');
</script>
</body>
</html>

```

Затем следует нарисовать блок (рисунок 5.3) и персонажа игры (рисунок 5.4).



Рисунок 5.3 – Изображение блока

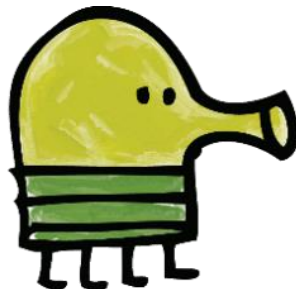


Рисунок 5.4 – Персонаж игры – Дудлик

### 5.2.6 Тестирование проекта

Для проверки проекта (рисунок 5.5) необходимо:

1. Перейти на вкладку **SIMULATE**.
2. Выбрать устройство.
3. Нажать кнопку **Start Simulator**.



Рисунок 5.5 – Тестирование проекта

### 5.2.7 Запуск проекта

Для запуска проекта на смартфоне (рисунок 5.6) необходимо:

1. Скачать приложение **AppPreview** в **Play Market**.
2. Зарегистрироваться.
3. Войти в свой аккаунт.

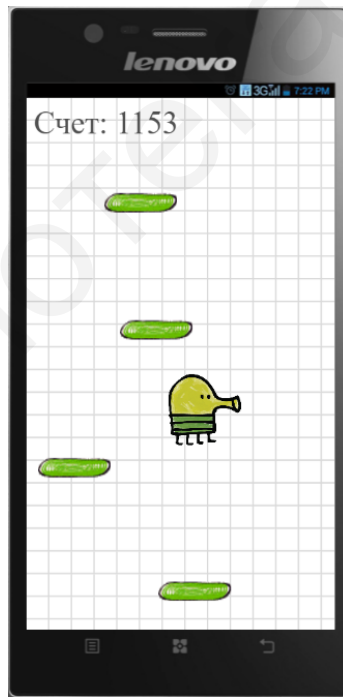


Рисунок 5.5 – Запуск проекта на смартфоне

### Выводы

В результате выполнения лабораторной работы должно быть разработано мобильное приложение в виде игры Doodle Jump.

# ЛАБОРАТОРНАЯ РАБОТА №6

## ВИДЖЕТЫ

**Цель работы:** разработать мобильное приложение с виджетом, которое получает погодную сводку.

### 6.1 Необходимое ПО

1. Установленная среда разработки **IntelXDK** (официальный сайт <https://software.intel.com/en-us/xdk>).
2. Игровой движок **PointJS** (<https://mult-uroki.ru/pointjs>).
3. **HTML5**.
4. **JavaScript**.

### 6.2 Порядок выполнения лабораторной работы

Приложение разрабатывается под мобильную систему Android.

Для получения сводки погоды используется сервис **openweathermap**.

В главном окне приложения размещены семь **textview** для отображения информации о погоде и местоположении:

```
<TextView
    android:id="@+id/city_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:text=""
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="#FFFFFF" />
```

```
<TextView
    android:id="@+id/updated_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/city_field"
    android:layout_centerHorizontal="true"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="#FFFFFF"
    android:textSize="13sp" />
```

```
<TextView
    android:id="@+id/weather_icon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="#FFFFFF"
    android:textSize="90sp" />
```

```
<TextView
    android:id="@+id/current_temperature_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="#FFFFFF"
    android:textSize="50sp" />
```

```
<TextView
    android:id="@+id/details_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/weather_icon"
    android:layout_centerHorizontal="true"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="#FFFFFF" />
```

```
<TextView
    android:id="@+id/humidity_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/details_field"
    android:layout_centerHorizontal="true"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="#FFFFFF" />
```

```
<TextView
    android:id="@+id/pressure_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/humidity_field"
```

```
android:layout_centerHorizontal="true"
android:textAppearance="?android:attr/textAppearanceMedium"
android:textColor="#FFFFFF" />
```

Получение данных о погоде:

```
private static final String OPEN_WEATHER_MAP_URL =
"http://api.openweathermap.org/data/2.5/weather?id=625144&units=metric";
```

Переменная, хранящая в себе запрос для получения сводки, где `id` – это город, а `units` – система мер, в данном случае метрическая.

```
private static final String OPEN_WEATHER_MAP_API = "*****"
```

Переменная, хранящая в себе ключ для API:

```
public static JSONObject getWeatherJSON(String lat, String lon){
try {
    URL url = new URL(String.format(OPEN_WEATHER_MAP_URL, lat, lon));
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    connection.setRequestProperty("x-api-key", OPEN_WEATHER_MAP_API);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
    StringBuffer json = new StringBuffer(1024);
    String tmp="";
    while((tmp=reader.readLine())!=null
        json.append(tmp).append("\n");
    reader.close();
    JSONObject data = new JSONObject(json.toString());

    // This value will be 404 if the request was notsuccessful
    if(data.getInt("cod") != 200){
        return null;
    }
    return data;
} catch(Exception e){
    return null;
}
}
```

Ниже представлен запрос, который получает json-объект, используя url и ключ api:

```
protected void onPostExecute(JSONObject json) {
    try {
        if(json != null){
            JSONObject details = json.getJSONArray("weather").getJSONObject(0);
            JSONObject main = json.getJSONObject("main");
            DateFormat df = DateFormat.getDateInstance();
            String city = json.getString("name").toUpperCase(Locale.US) + ", " +
                json.getJSONObject("sys").getString("country");
            String description = details.getString("description").toUpperCase(Locale.US);
            String temperature = String.format("%.2f", main.getDouble("temp"))+ "°";
            String humidity = main.getString("humidity") + "%";
            String pressure = main.getString("pressure") + " hPa";
            String updatedOn = df.format(new Date(json.getLong("dt")*1000));
            String iconText = setWeatherIcon(details.getInt("id"),
                json.getJSONObject("sys").getLong("sunrise") * 1000,
                json.getJSONObject("sys").getLong("sunset") * 1000);
            delegate.processFinish(city, description, temperature, humidity, pressure,
                updatedOn,iconText,""+(json.getJSONObject("sys").getLong("sunrise")* 1000));
        }
    } catch (JSONException e) {
        //Log.e(LOG_TAG, "Cannot process JSON results", e);
    }
}
```

При завершении процесса получения json-данных объект извлекает необходимую информацию:

```
public class MainActivity extends AppCompatActivity {
    private Timer autoUpdate;
    TextView cityField, detailsField, currentTemperatureField, humidity_field,
    pressure_field, weatherIcon, updatedField;
    Typeface weatherFont;
    public String city,temp,humidity,pressure;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportActionBar().hide();
        setContentView(R.layout.activity_main);
    }
    @Override
```

```

    public void onResume() {
super.onResume();
autoUpdate = new Timer();
autoUpdate.schedule(new TimerTask() {
    @Override
    public void run() {
runOnUiThread(new Runnable() {
    public void run() {
        updateWeather();
    }
    });
    }, 0, 40000); // updates each 40 secs
}
private void updateWeather(){
// your logic here
    weatherFont = Typeface.createFromAsset(getApplicationContext().getAssets(),
"fonts/weathericons-regular-webfont.ttf");
cityField = (TextView)findViewById(city_field);
updatedField = (TextView)findViewById(R.id.updated_field);
detailsField = (TextView)findViewById(R.id.details_field);
currentTemperatureField=(TextView)findViewById(R.id.current_temperature_field);
humidity_field = (TextView)findViewById(R.id.humidity_field);
pressure_field = (TextView)findViewById(R.id.pressure_field);
weatherIcon = (TextView)findViewById(R.id.weather_icon);
weatherIcon.setTypeface(weatherFont);

```

```

Function.placeIdTask asyncTask=
new Function.placeIdTask(new Function.AsyncResponse() {
    public void processFinish(String weather_city, String weather_description,
String weather_temperature, String weather_humidity,
String weather_pressure, String weather_updatedOn,
String weather_iconText, String sun_rise) {
cityField.setText(weather_city);
city = weather_city;
updatedField.setText(weather_updatedOn);
detailsField.setText(weather_description);
currentTemperatureField.setText(weather_temperature);
temp = weather_temperature;
humidity_field.setText("Humidity: "+weather_humidity);
humidity = weather_humidity;
pressure_field.setText("Pressure: "+weather_pressure);
pressure = weather_pressure;

```



```

weatherIcon.setText(Html.fromHtml(weather_iconText));
SharedPreferences preferences=getSharedPreferences("info",Context.MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putString("city",city);
editor.putString("temp",temp);
editor.putString("humidity",humidity);
editor.putString("pressure",pressure);
editor.apply();
}
});
//city = "jhj";
asyncTask.execute("53.901135","27.557255");//asyncTask.execute("Latitude","Longitude")
//city = "kjk";
//city = cityField.toString();
//city = ((TextView) findViewById(city_field)).getText().toString();
/*SharedPreferences preferences = getSharedPreferences("info",
Context.MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putString("city",city);
editor.apply();*/
}
@Override
public void onPause() {
autoUpdate.cancel();
super.onPause();
}
}

```

Пока приложение запущено, оно обновляется каждые 40 с. Из функции Function принимаются данные и присваиваются соответствующим полям в главном окне. Также при помощи **SharedPreferences** эти данные становятся доступными в виджете.

**SharedPreferences** является одним из способов передачи данных между **activity**.

```

public class Widget extends AppWidgetProvider {
Context pref;
static void updateAppWidget(Context context,AppWidgetManager
appWidgetManager, int appWidgetId) {
CharSequence widgetText = context.getString(R.string.appwidget_text);

// Construct the RemoteViews object
RemoteViews views = new RemoteViews(context.getPackageName(), widget);

```

```

views.setTextViewText(R.id.city, widgetText);

// Instruct the widget manager to update the widget
appWidgetManager.updateAppWidget(appWidgetId, views);
}
@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
// There may be multiple widgets active, so update all of them
SharedPreferences prfs = context.getSharedPreferences("info",
Context.MODE_PRIVATE);
String city = prfs.getString("city", "");
String temp = prfs.getString("temp", "");
String humidity = prfs.getString("humidity", "");
String pressure = prfs.getString("pressure", "");

for (int appWidgetId : appWidgetIds) {
updateAppWidget(context, appWidgetManager, appWidgetId);
RemoteViews views = new RemoteViews(context.getPackageName(),
widget);
views.setTextViewText(R.id.city, city);
views.setTextViewText(R.id.temp, temp);
views.setTextViewText(R.id.humidity, humidity);
views.setTextViewText(R.id.pressure, pressure);
appWidgetManager.updateAppWidget(appWidgetId, views);
}
}
@Override
public void onEnabled(Context context) {
// Enter relevant functionality for when the first widget is created
}
@Override
public void onDisabled(Context context) {
// Enter relevant functionality for when the last widget is disabled
}
}

```

При использовании **SharedPreferences** получаются данные из **MainActivity**, которые и отображаются на форме виджета.

```

<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
android:initialKeyguardLayout="@layout/widget"
android:initialLayout="@layout/widget"
android:minHeight="120dp"

```

```
android:minWidth="180dp"  
android:previewImage="@drawable/example_appwidget_preview"  
android:resizeMode="horizontal|vertical"  
android:updatePeriodMillis="1800000"  
android:widgetCategory="home_screen"></appwidget-provider>
```

Виджет обновляется каждые 1 800 000 мс.

На рисунке 6.1 приведен вид главного окна проекта.



Рисунок 6.1 – Главное окно проекта

В результате выполненной работы получается вид приложения, представленный на рисунке 6.2.



Рисунок 6.2 – Виджет

### **Выводы**

В результате выполнения лабораторной работы должно быть разработано мобильное приложение с использованием виджета погодной сводки, предоставляющее пользователю оперативную метеорологическую сводку.

## ЛИТЕРАТУРА

1. Усов, В. Swift. Основы разработки приложений под iOS / В. Усов. – СПб. : Питер, 2016. – 304 с.
2. Конвэй, Д. Программирование под iOS / Д. Конвэй, А. Хиллегасс. – СПб. : Питер, 2013. – 608 с.
3. Нойбург, М. Программирование для iOS 7. Основы Objective-C, Xcode и Cocoa / М. Нойбург. – М. : Вильямс, – 2014. – 384 с.
4. Программирование под Android / З. Медникс [и др.]. – СПб. : Питер, 2013. – 496 с.
5. Марк, Д. Разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK / Д. Марк, К. Топли, Д. Ламарш. – М. : Вильямс, 2016. – 816 с.

Библиотека БГУИР

*Учебное издание*

**Трус Владимир Викторович**  
**Калитеня Иван Леонидович**  
**Бакунов Александр Михайлович и др.**

**ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ  
МОБИЛЬНЫХ ПЛАТФОРМ**

**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ**

Редактор *М. А. Зайцева*  
Корректор *Е. Н. Батурчик*  
Компьютерная правка, оригинал-макет *О. И. Толкач*

Подписано в печать 04.06.2020. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».  
Отпечатано на ризографе. Усл. печ. л. 4,65. Уч.-изд л. 5,2. Тираж 70 экз. Заказ 370.

Издатель и полиграфическое исполнение: учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники».  
Свидетельство о государственной регистрации издателя, изготовителя,  
распространителя печатных изданий №1/238 от 24.03.2014,  
№2/113 от 07.04.2014, №3/615 от 07.04.2014.  
Ул. П. Бровки, 6, 220013, г. Минск