

**Санкт-Петербургский политехнический университет**

**ПЕТРА ВЕЛИКОГО**

**Институт компьютерных наук и технологий**

**Кафедра «Распределенные вычисления и компьютерные сети»**

**КУРСОВОЙ ПРОЕКТ**

**Автоматизированная система «Гараж»**

**по дисциплине**

**Программная инженерия**

Выполнил  
студент гр. 33507/1

С. Д. Копытов

Руководитель  
профессор, к.т.н

А.В. Самочадин

Санкт-Петербург

2016

Постановка задачи	3
Описание проделанной работы	4
Начало работы над проектом	4
Описание процесса разработки	5
Структура проекта	5
База данных	6
Программирование	7
Интерфейс	10
Тестирование	11
Подведение итогов	12
Надежность	12
Вывод	13

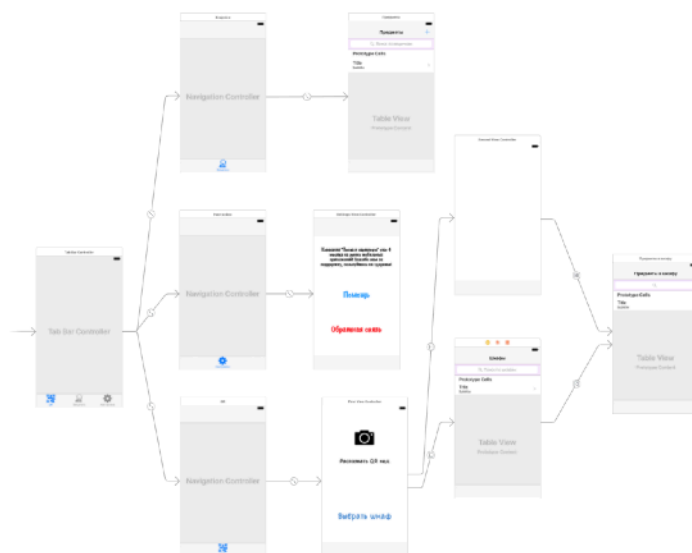
# Постановка задачи

Требовалось создать приложение, позволяющее быстро и эффективно создавать, дополнять и использовать некую базу хранимых товаров (различных вещей, материалов, инструментов и др.). Продукт должен работать стабильно (не вылетать например), а также справляться с объемом работы качественно и надежно.

# Описание проделанной работы

## Начало работы над проектом

В проекте моя роль была - роль главного разработчика. Ввиду некоторого опыта разработки под операционную систему iOS, целевой системой была выбрана именно она. После согласования некоторых вещей, наша команда составила приблизительный проект, который в дальнейшем нужно было запрограммировать. Макет представлял собой несколько экранов, собранных вместе при помощи UITabBarController, а также нескольких экранов в каждой вкладке, собранных под UINavigationController. Именно контроллер навигации обеспечивает надежное использование нашего приложения, так как вместо создания нового экземпляра при переходе между сценами (экранами), он обеспечивает создания лишь одной копии и дальнейшее её открытие, это позволяет избежать ошибки переполнения стека. Ниже приведен скриншот окончательного макета приложения.



# Описание процесса разработки

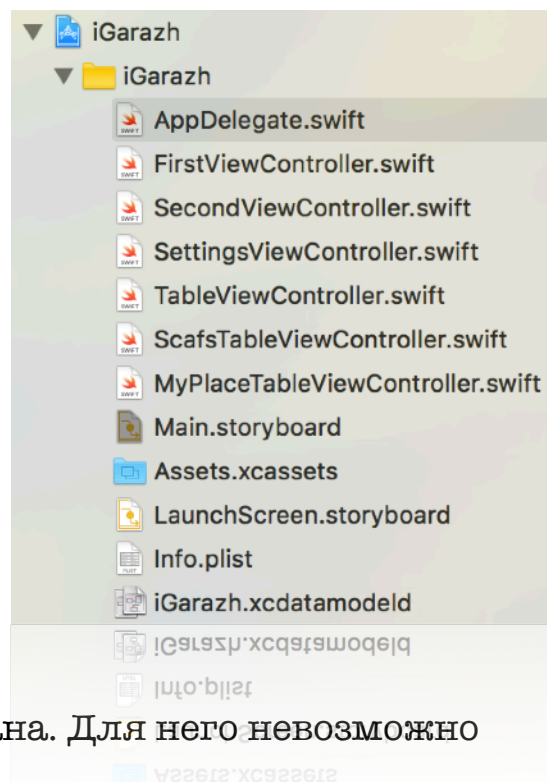
## Структура проекта

Как только приблизительный макет приложения был готов, было принято решение разграничить роли всех участников проекта. Для помощи в разработке под систему iOS я выбрал Алексея Шаланкина. Позже мною был создан приватный репозиторий GitHub для удобного процесса разработки. Таким образом, я получал задачи от Дмитрия в Redmine, затем обсуждал их либо при личной встрече, либо на просторах интернета, после этого старался выполнить их в срок, а также принести свои дополнения в проект, ну и, повсеместно, производя коммиты в репозиторий.

В итоге структура проекта представляла собой 12 файлов.

Файлы с расширением «.swift» содержат в себе непосредственно исполняемый код (о нем поподробнее чуть позже).

Файлы с расширением «.storyboard» представляют собой файлы с интерфейсом, «LaunchScreen» - это интерфейс, подгружаемый в первую очередь, вроде загрузочного экрана. Для него невозможно написать исполнимый код, в отличие от «Main», содержащий интерфейс контроллеров, чей функционал описан в файлах «.swift». Работа контроллеров представляет собой «базу данных» (образно говоря), то есть при их открытии они



отправляют запросы в файл, присвоенный им при проектировании структуры, ответы на которые, как раз, должен написать программист.

Файл с расширением «.xcassets» содержит хранилище всех внешних файлов проекта, например картинки.

Файл с расширением «.plist» хранит в себе большую базу информации. По сути, это аналог xml, но представленный в более удобном формате. Именно в этом файле указываются поля основного языка приложения, поля используемых ресурсов (вроде камеры), а также текст в запросе на разрешение использования их, имя продукта и многое другое.

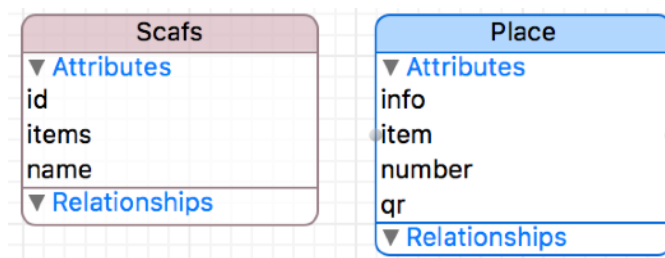
Файл с расширением «.xcdatamodeld» позволяет настроить свою «модель данных», и, в дальнейшем, при помощи фреймворка «Core Data», использовать ее, как базу данных.

## **База данных**

После создания макета проекта, а также всех необходимых файлов, я принялся за разработку базы данных. Вместе с Алексеем, я построил модель данных и представил её в проекте при помощи фреймворка «Core Data» (как и говорилось ранее). Суть работы данного фреймворка заключается не в формировании постоянных запросов, а в хранении данных в формате модели и запросов к ней, для выгрузки данных. То есть мы обращаемся к данным только при инициализации переменных (массива) в котором они будут храниться, а затем уже обращаемся к переменной (массиву). Вместе с командой разработки мы пришли к выводу, что данные удобнее всего будет



хранить в двух моделях, модели товара и модели шкафа, таким



образом мы получили базу данных с двумя таблицами. Таблица «Place» стала базой информации о товаре, а таблица «Scafs» стала базой информации о шкафах.

В общем случае данная база используется, как массив переменных некоего класса, содержащего поля с информацией. Базы связывали два поля: «id» - «qr» и «name» - «number». Первое поле содержит уникальный идентификатор (к нему мы вернемся позже), а второе - название шкафа. В поле «items» хранится информация о количестве вещей, содержащихся в шкафу, поле «item» хранит название товара, а поле «info» - его описание (например номер полки).

## Программирование

Для начала я заполнил информацию о проекте в файл «info.plist», затем создал все необходимые переменные, такие как массивы (для базы данных), а также переменные интерфейса и их соединения (названные IBOutlet или IBAction).

Так как, я живу по принципу «сначала легкое, потом тяжелое», то начал я, как ни странно, с файла «Settings», в котором, при поддержке библиотеки «MessageUI» и протокола «MFMailComposeViewControllerDelegate» я написал код, позволяющий сделать обратную связь легкой и доступной по нажатию лишь одной кнопки. Также я написал информацию о проекте и создал кнопку «помощь».

После этого я решил заняться созданием представления базы данных, для чего был создан «UITableView». Далее я сформировал все необходимые ответы на запросы данной таблицы, а также сделал поиск (с помощью протокола «UISearchBarDelegate»), принцип работы которого заключается в фильтрации исходного массива и сохранении результатов в новый.

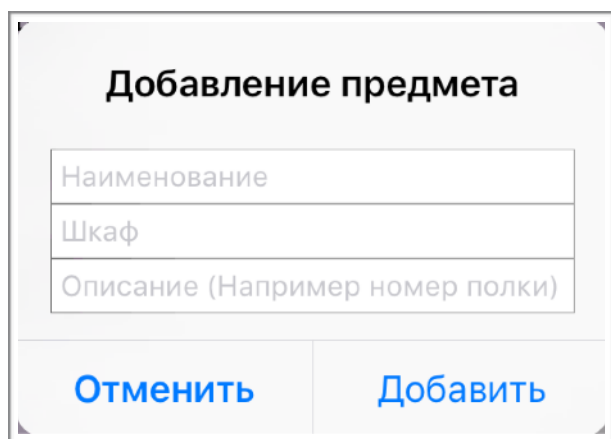
```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
    filtered = mas.filter({ (text) -> Bool in
        let tmp: String = text.item! as String
        let range = tmp.range(of: searchText, options: String.CompareOptions.caseInsensitive)
        return range != nil
    })

    if(filtered.isEmpty){
        SActive = false;
    } else {
        SActive = true;
    }

    self.tableView.reloadData()
}
```

После того, как таблица была сформирована, я решил заняться формированием данных из базы данных. Затем и создал новый экран, который выводил полную информацию от товаре

(при первой презентации проекта все было именно так), позже я упразднил этот переход, сделав вывод информации в всплывающем окне. Помимо этого, я добавил кнопку «+» в правый верхний угол, по нажатию на которую откроется



всплывающее окно с полями ввода информации о товаре, причем поля «наименование» и «шкаф» являются обязательными, в отличие от поля «описание».



Теперь все было готово к созданию универсального ключа для каждого шкафа (QR кода), чем я и решил заняться. В данные кода я решил сохранять идентификатор шкафа, таким образом не будет производиться утечка неверной информации (например при считывании QR кода другим человеком). Для генерации изображения была

написана новая функция, а также, для сохранения изображения я создал расширения для класса «UIImage»,

```
func generateQRCode(from string: String) -> UIImage? {
    let data = string.data(using: String.Encoding.ascii)

    if let filter = CIFilter(name: "CIQRCodeGenerator") {
        filter.setValue(data, forKey: "inputMessage")
        let transform = CGAffineTransform(scaleX: 100, y: 100)

        if let output = filter.outputImage?.applying(transform) {
            return UIImage(cgImage: output)
        }
    }

    return nil
}
```

позволяющее мне сжимать картинки, так как изображение, генерируемое моей функцией, не могло сохраниться.

Ну а дальше нужно было написать сам «считыватель» QR-кода. Для этого я создал отдельную клавишу на первом контроллере (первый, то есть он первый в Tab Bar и первый, который открывается), и по клавише создал сегвей (на языке xcode - это переход) на новый экран. В файле с кодом нового контроллера я описал считывание кода. Принцип его работы довольно прост: в первую очередь запускается видоискатель на весь экран, затем он ищет различные метаданные и, как только находит метаданные типа QR код, окрашивает их в красную рамочку и совершает новый сегвей на новую таблицу, передавая при этом данные, считанные из QR кода. По этим данным воссоздается нужный нам шкаф.

Затем я решил, что хорошо бы было получать доступ к шкафам без считывания QR кода, например, если хочу узнать, что

лежит в конкретном шкафу, находясь далеко от него. Так, на главном экране появилась кнопка «Выбор шкафа», переходя по которой открывается таблица, в ней можно выбрать нужный шкаф и, либо по нажатию перейти к новой таблице с обзором нужного шкафа, либо по свайпу влево на ячейке шкафа можно получить доступ к кнопке «share», созданной для передачи изображения с QR кодом выбранного шкафа. Также функция «share» доступна и в таблице шкафа, и в основной таблице. В основной таблице, помимо функции «share», доступны функции «delete» и «edit». Они дают пользователю возможность удалить или изменить информацию о товаре. При изменении откроется похожее окно, каждое поле в котором будет необязательным, но при изменении вся информация будет занесена в базу.

## **Интерфейс**

Теперь пришло время интерфейса, ведь именно с ним взаимодействует пользователь напрямую. При построении внешнего вида, мне пришла в голову идея - сделать все простым и понятным с первого взгляда, без лишних подписей и экранов. Так, ввод информации представляет собой легкое всплывающее окно, а не громоздкий новый экран.

При формировании TableView, я старался описать как можно больше системных методов, для того, чтобы таблица выглядела свежо и нативно, а также была простой и понятной в использовании.

После описания всех интерфейсных оболочек программным кодом, формирующим их, я занялся визуализацией. Первое - я подобрал к приложению фон. Совместно с Дмитрием было

принято решение использовать простую и шуточную картинку. После поисков в google, а также небольших манипуляций в photoshop, картинка была готова. Я сразу наложил её на фон, а также сделал ячейки таблиц прозрачными, чтобы картинку было видно. После этого была выбрана иконка и воссоздана для хранилища иконок в xcode при помощи MakeAppIcon. Затем мы выбирали шрифты, цвет и размер текста и прочие вещи, пока не были довольны. Итак, «дизайн» был готов.

## **Тестирование**

Самая важная часть разработки проекта - проверка его работоспособности! Как только приложение было почти готово, я решил начать его тестировать. Для этого я установил его на свой смартфон и начал не только использовать его по назначению, но и нажимать на все кнопки подряд, проверяя его стабильность. Также, я установил приложение на телефоны своих друзей и попросил их о помощи в тестировке. В итоге приложение было проверено на разных версиях смартфона от Apple.

# Подведение итогов

## Надежность

Как я уже и говорил, в QR коде хранится значение поля «id», в котором, в свою очередь, хранится уникальный идентификационный номер, генерируемый функцией «UUID», лежащей в библиотеке «UIKit». Данный код представляет собой последовательность из 32 чисел и букв, разбитых на 5 групп, которые в свою очередь разбиты дефисом (пример кода: 159CAFBC-36DC-4E84-92F5-D7137A7875FC). Функция «UUID» генерирует уникальный ключ, а его сложность и длина позволяют исключить вероятность повторения. Получаем, что имя шкафа никто, кроме хозяина не узнает, считая QR-код.

База данных, в которой хранится вся информация, лежит локально в папке с приложением, к ней нет удаленного доступа. Таким образом утечка данных исключена. В нашем приложении можно хранить расположение личных вещей, не переживая при этом за попадание этой информации в чужие руки.

Считывание QR кода происходит при поддержке стандартной библиотеки, а это значит, что вероятность ошибки в данном процессе настолько невелико, насколько качественно в Apple создают библиотеки для Swift.

После нескольких дней использования приложения на различных устройствах (соответственно после последнего исправления багов) ошибок не было выявлено, что означает, что при регулярном использовании, приложение работает верно, а также стабильно.

## **Вывод**

Итогом всей проделанной работы стало приложение, позволяющее качественно, быстро и надежно создать собственное виртуальное хранилище вещей, а также управлять всей этой базой, например в случае перемещения объектов. Также, наше приложение позволяет наиболее эффективно использовать возможности современных мобильных устройств, а именно их камеры, при помощи которых пользователю открывается возможность, не открывая шкаф, узнать о всех товарах, лежащих в нем. Конечно, пользователем предварительно должна быть составлена вся его база верно, кроме того QR коды должны располагаться также верно. Ошибки пользователя могут привести к ошибкам в работе программы.

Подводя итоги, я считаю, что созданное нашей командой приложение прекрасно справляется со своими задачами, упрощая банальный ежедневный поиск предметов.