МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2021 г.

Разработка iOS-приложения для поиска мастера на час

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2021.308-281.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-402
_____ А.В. Загоскин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2021 г.

Челябинск, 2021 г

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2021 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-402
Загоскину Артему Викторовичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. Тема работы (утверждена приказом ректора от 26.04.2021 г. № 714-13/12)
Разработка iOS-приложения для поиска мастера на час.
2. Срок сдачи студентом законченной работы: 05.06.2021 г.
3. Исходные данные к работе
3.1. The Swift Programming Language. [Электронный ресурс] URL: https://developer.apple.com/documentation/swift (дата обращения 05.04.2021 г.).
3.2. XCode. [Электронный ресурс] URL: https://developer.apple.com/documentation/xcode/ (дата обращения 05.04.2021 г.).
4. Перечень подлежащих разработке вопросов
4.1. Провести анализ предметной области.
4.2. Спроектировать мобильное приложение.
4.3. Реализовать мобильное приложение.
4.4. Протестировать мобильное приложение.
5. Дата выдачи задания: 08.02.2021 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению А.В. Загоскин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

На сегодняшний день все большее количество людей начинают переносить банальные дела в онлайн: заказывают еду вместо того, чтобы идти в супермаркет, покупают одежду в мобильных приложениях и на сайтах, а не в торговых центрах и специализированных магазинах [1]. Во время пандемии Covid-19 это стало особо актуальным, а рынок онлайн-магазинов вырос почти в два раза только за апрель 2020 года [2]. Но и действовать привычно, например, идти в магазин у дома за хлебом не составляет проблемы. Когда же дело доходит до, например, установки шкафа или каких-то технических установок в доме, людям приходится искать знакомых, способных помочь им, или учиться этому на курсах или в интернете. К сожалению, не всем это интересно, а порой времени на обучение нет совсем. Именно для этого и необходимо приложение, в котором можно выбрать и без рисков риска того, что это мошенник, нанять специалиста, который приедет для выполнения необходимой задачи.

В настоящее время самыми популярными и используемыми устройствами являются смартфоны. Ими пользуются чаще, чем какими-либо другими устройствами. Устройства iPhone от компании Apple, работающие на операционной системе iOS, продаются в большом количестве – они занимают 23,4% рынка смартфонов [3].

Исходя из приведенных данных был сделан вывод, что разработка приложения для iOS, благодаря которому люди смогут искать специалистов, оказывающих услуги в разных направлениях, является актуальной задачей.

Цели и задачи работы

Целью проекта является разработка iOS-приложения для поиска мастера на час. Для реализации цели были поставлены следующие задачи:

1) провести анализ предметной области и обзор аналогичных проектов;

2) спроектировать мобильное приложение;
3) реализовать мобильное приложение;
4) протестировать мобильное приложение.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы – 31 страница, объем списка литературы – 24 источника.

В главе «Анализ предметной области» приведено описание предметной области проекта, рассмотрены аналогичные решения.

В главе «Проектирование» приведены функциональные и нефункциональные требования к программной системе и рассмотрены актеры и варианты использования разрабатываемого мобильного приложения.

В главе «Архитектура системы» выделены основные компоненты системы и представлена диаграмма использования.

В главе «Реализация» описана реализация основных компонентов мобильного приложения, взаимодействия с сервером и представлены вспомогательные средства реализации.

Глава «Тестирование» посвящена тестированию разработанного мобильного приложения.

В заключении описаны основные результаты работы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Основной функционал представленного iOS-приложения, направленного на поиск исполнителей для профессиональных задач, должен включать в себя:

1) возможность авторизоваться;
2) возможность просмотра списка всех задач;
3) возможность просмотра списка пользовательских задач;
4) возможность просмотра списка откликов на определенную задачу;
5) возможность просмотра профиля;
6) возможность оставления обратной связи;
7) возможность добавления адреса;
8) возможность создания задачи.

1.2. Анализ схожих проектов

YouDo – сервис в виде web-сайта и мобильного приложения для iOS и Android. Помогает быстро находить исполнителей для любой работы и поручений: уборка в квартире, перевозка грузов, бытовой ремонт, компьютерная помощь, строительство, фриланс-услуги, вызов курьера, фотографа, мужа на час и т.д.

Среди главных достоинств сервиса – приятный интерфейс, быстрое создание заданий, получение откликов в короткие сроки и возможность опираться на цену, рейтинг и отзывы при выборе исполнителя.

К минусам приложения относится то, что безопасность сделок не гарантирована.

На рисунке 1 представлен интерфейс мобильного приложения YouDo.

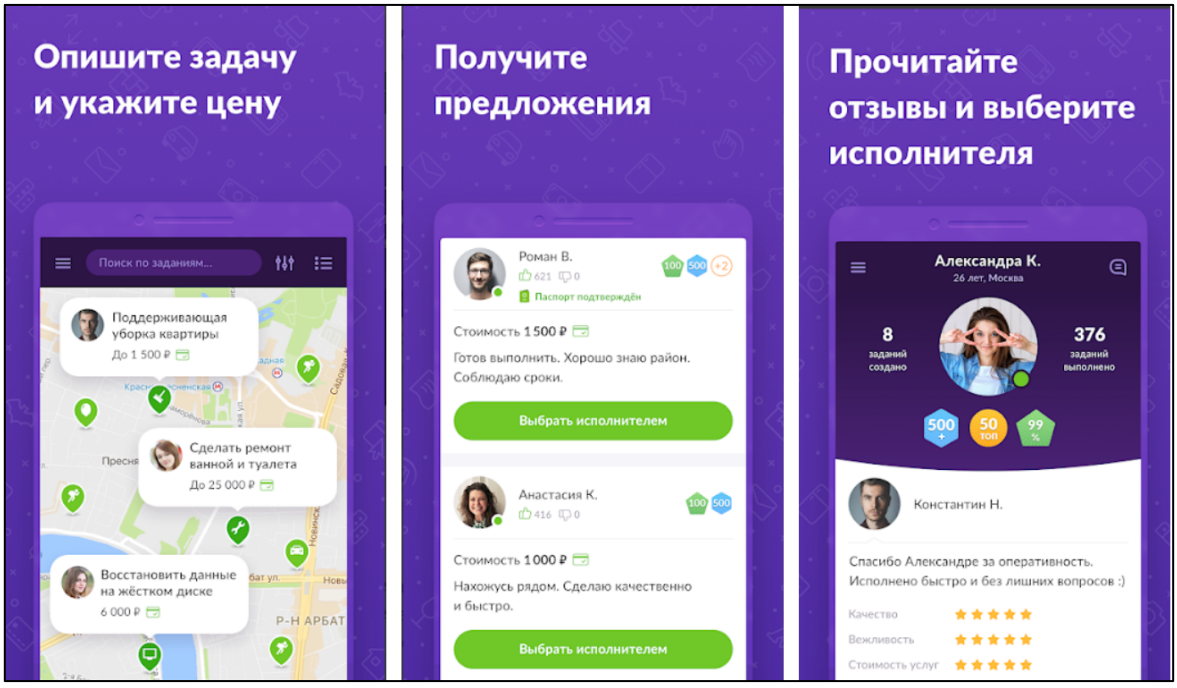

Рисунок 1 – Мобильное приложение YouDo

PROFI.RU – это рекомендательный сервис по подбору частных специалистов.

Сервис PROFI.RU бесплатный только для клиентов сервиса. Услуги по организации работы оплачивают специалисты.

К сожалению, регистрация в сервисе происходит медленно, модераторы, судя по отзывам, часто удаляют негативные отзывы, а одни и те же элементы интерфейса могут быть сделаны в разном дизайне, что может запутать пользователя.

На рисунке 2 представлен интерфейс мобильного приложения Profi.ru.

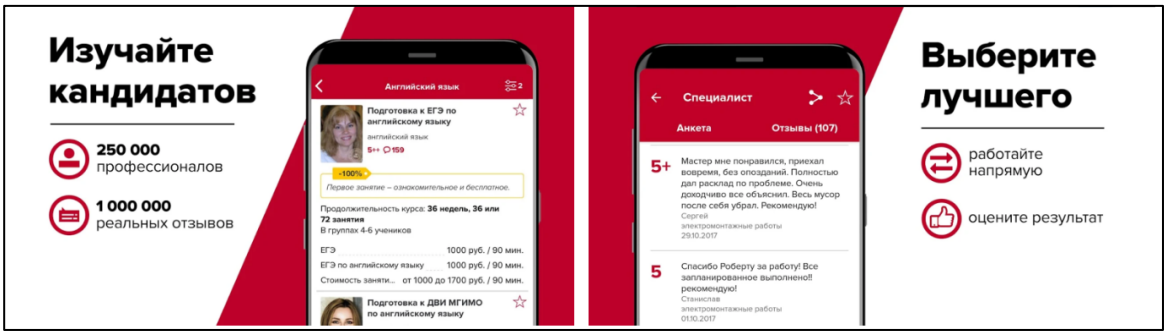

Рисунок 2 – Мобильное приложение Profi.ru

СберУслуги – платформа, которая обеспечивает безопасные сделки по предоставлению услуг. Участники застрахованы в экосистеме компании, а исполнители проверены через Сбер ID.

К минусам можно отнести отсутствие в приложении списка исполнителей для выбора – при создании задачи появляются карточки исполнителей поочередно, а не сразу все.

На рисунке 3 представлен интерфейс мобильного приложения СберУслуги.

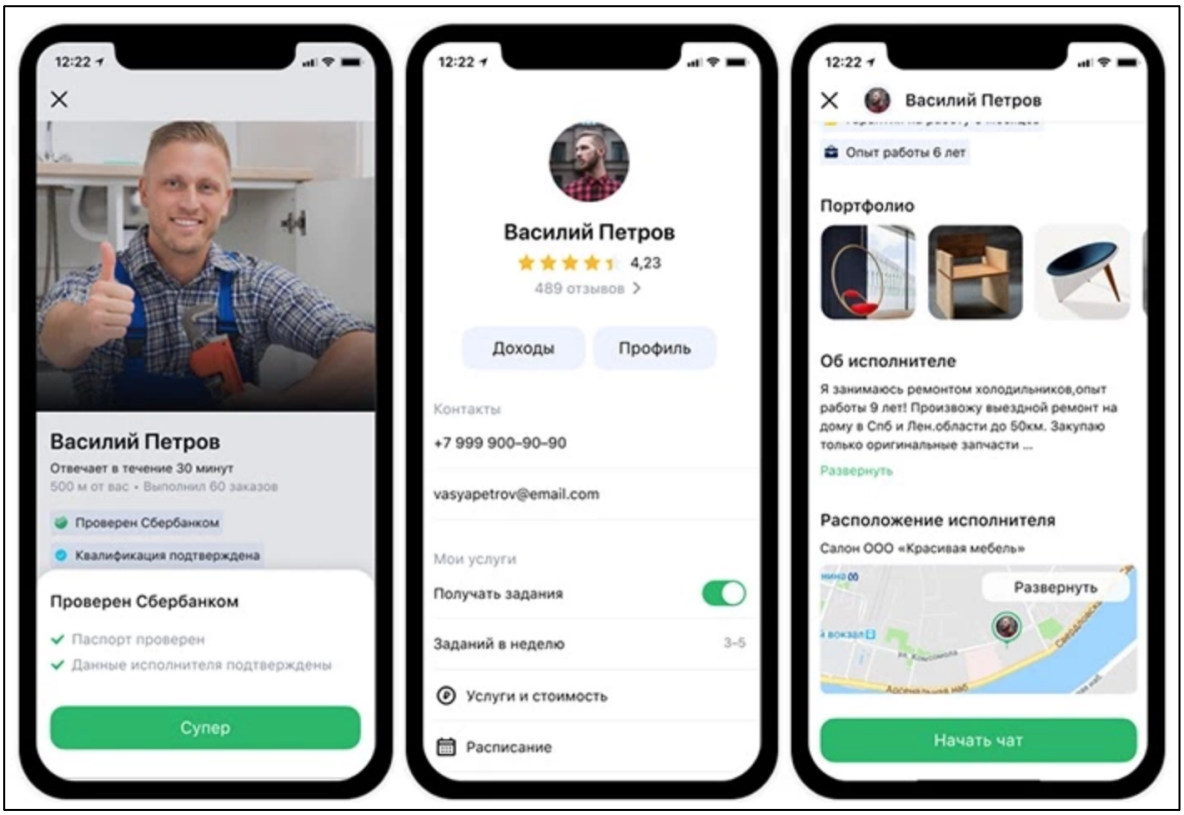

Рисунок 3 – Мобильное приложение СберУслуги

В результате обзора аналогов было выявлено, что разрабатываемому сервису необходимо иметь приятный однородный интерфейс, удобную и быструю регистрации, чтобы оттолкнуть пользователя на раннем этапе, и возможность видеть всех откликнувшихся на задачу исполнителей для более удобного выбора.

2. ПРОЕКТИРОВАНИЕ

2.1. Требования к проектируемой системе

В ходе анализа предметной области были определены следующие функциональные требования:

1) мобильное приложение должно предоставлять гостю возможность регистрации и авторизации;

2) мобильное приложение должно предоставлять пользователю возможность просматривать список задач, доступных для исполнения;

3) мобильное приложение должно предоставлять пользователю возможность фильтровать список задач;

4) мобильное приложение должно предоставлять пользователю возможность отклика на задачу;

5) мобильное приложение должно предоставлять пользователю возможность создания своей задачи;

6) мобильное приложение должно предоставлять пользователю возможность добавлять информацию о себе;

7) мобильное приложение должно предоставлять пользователю возможность выбирать исполнителя для созданной задачи;

8) мобильное приложение должно предоставлять пользователю возможность оценить исполнителя задачи;

9) мобильное приложение должно предоставлять пользователю возможность просмотра информации об исполнителе;

10) мобильное приложение должно предоставлять пользователю возможность обращения в техническую поддержку;

11) мобильное приложение должно предоставлять пользователю возможность добавления фотографий и файлов;

12) мобильное приложение должно предоставлять пользователю возможность выхода из аккаунта;

13) мобильное приложение должно предоставлять пользователю возможность загрузки документов.

Также были определены нефункциональные требования:

1) мобильное приложение должно быть написано на языке Swift для платформы iOS;

2) мобильное приложение должно быть написано с помощью интегрированной среды разработки XCode;

3) мобильное приложение должно функционировать на устройствах iPhone седьмого поколения (iPhone 6) и выше;

4) мобильное приложение должно функционировать на устройствах с операционной системой iOS версии не ниже 11.0;

5) мобильное приложение должно быть доступно только в портретной ориентации.

2.2. Варианты использования системы

Для проектирования мобильного приложения был использован язык графического описания для объектного моделирования UML [4]. Была построена диаграмма вариантов использования, отражающая взаимодействие внешнего актера с системой.

В ходе проектирования было выделено три актера.

Гость – пользователь, которому доступны возможности регистрации и авторизации.

Заказчик – гость, прошедший авторизацию и имеющий доступ к полному функционалу мобильного приложения за исключением возможности оставлять отклики на задачи.

Исполнитель – заказчик, загрузивший документы и имеющий доступ оставлять отклики на задачи заказчиков.

Диаграмма вариантов использования мобильного приложения представлена на рисунке 4.

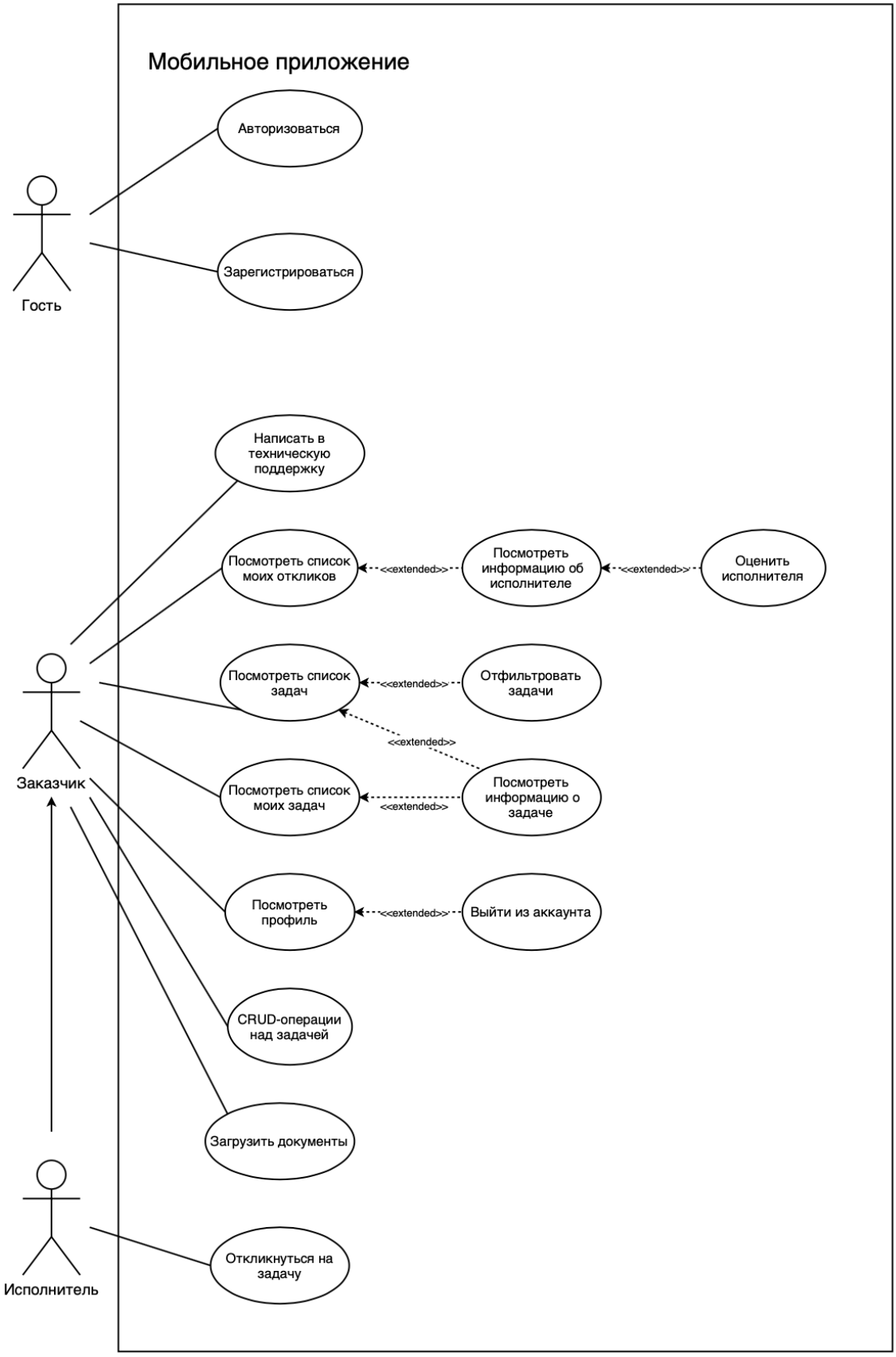

Рисунок 4 – Варианты использования мобильного приложения

Ниже представлено краткое описание вариантов использования мобильного приложения:

1) авторизоваться – войти в свой аккаунт;

2) зарегистрироваться – зарегистрировать аккаунт в системе;

3) посмотреть список задач – просмотр всех актуальных задач;

4) отфильтровать задачи – фильтрация списка задач по разным параметрам: тип задачи, удаленность места исполнения задачи и категория задачи;

5) посмотреть информацию о задаче – просмотр полной информации о задаче;

6) просмотреть список моих задач – просмотр списка созданных пользователем задач;

7) откликнуться на задачу – оставить отклик на определенную задачу;

8) просмотреть список моих откликов – просмотр списка сделанных исполнителем откликов;

9) загрузить документы – добавить фотографии пользователя и двух оборотов паспорта для прохождения проверки безопасности;

10) CRUD-операции над задачей – возможность создания, изменения и удаления пользовательской задачи;

11) просмотреть профиль – просмотр профиля пользователя;

12) написать в техническую поддержку – отправить письмо с вопросом или отзывом о приложении в техническую поддержку сервис;

13) выйти из аккаунта – вернуться на экран авторизации с понижением прав пользователя до прав гостя.

2.3. Архитектура системы

На рисунке 5 представлена диаграмма компонентов мобильного приложения для платформы iOS. В качестве архитектурного паттерна в

процессе проектирования приложения был выбран MVVM (Model-View-ViewModel) [5]. Система состоит из модели, представления и модели представления. Паттерн был выбран из-за слабой связанности компонентов и возможности переиспользования модели представлений для того, чтобы не допустить нарушение принципа DRY (Don’t repeat yourself) [6] – не повторять свой код.

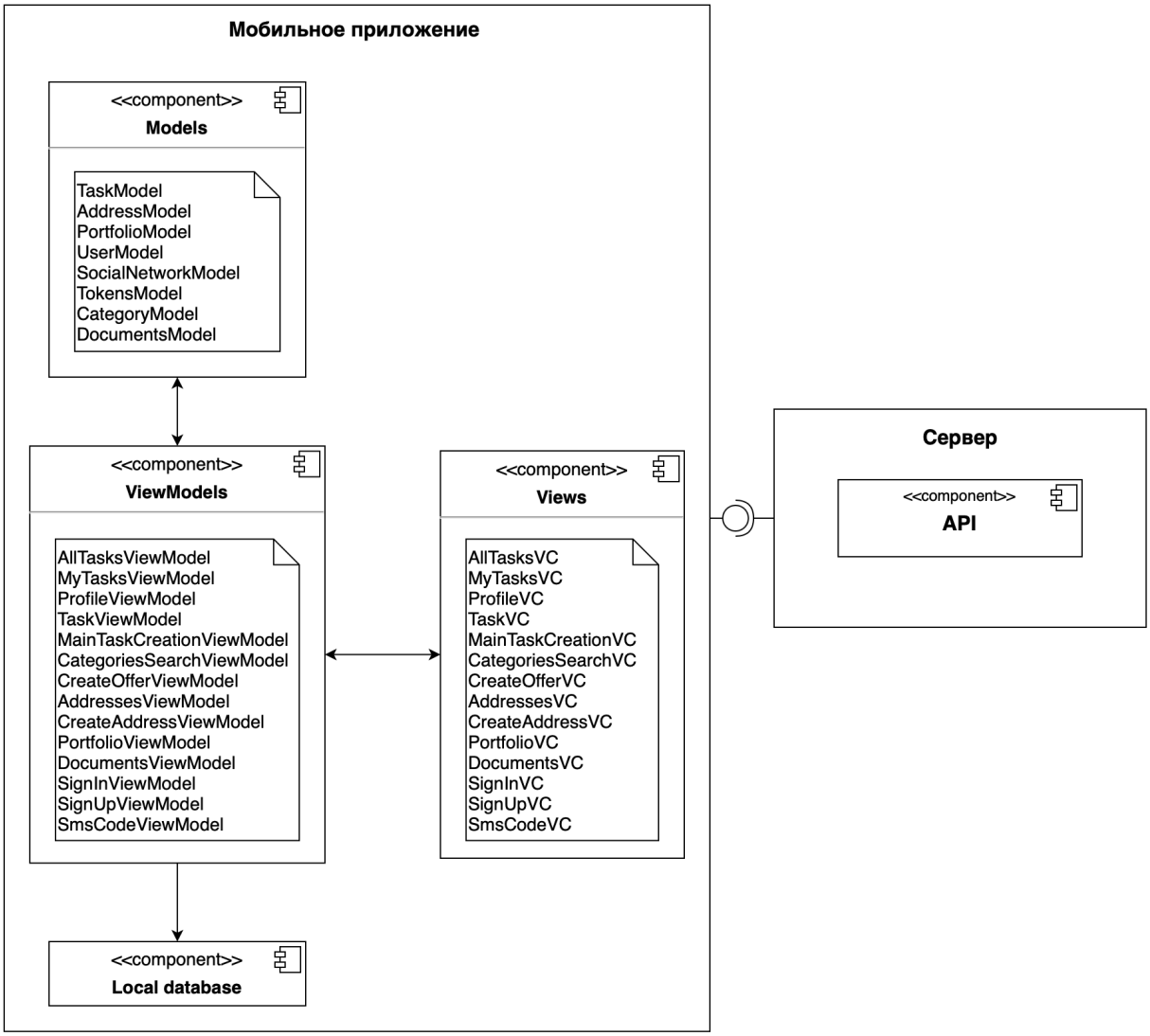

Рисунок 5 – Диаграмма компонентов системы

Компонент **Server** представляет собой серверную часть программного кода, осуществляющую хранение и предоставление данных. Взаимодействие мобильного приложения с компонентом Server осуществляется с помощью библиотек Alamofire [7] и SwaggerClient [8].

Компонент **Local database** – внутренняя база данных мобильного приложения. Компонент **Model** содержит данные приложения и предназначен для хранения данных с сервера или из локального хранилища. Компонент **ViewModel** – модель представления, связывающая модель и представление через механизм привязки данных. При изменении каких-либо данных автоматически изменяются отображаемые данные в представлении. Компонент **View** определяет визуальный интерфейс, через который пользователь взаимодействует с мобильным приложением.

Основные представления (далее – контроллеры) разрабатываемого мобильного приложения:

1) AllTasksVC – контроллер, отображающий список доступных для отклика задач;

2) MyTasksVC – контроллер, отображающий либо созданные пользователем задачи, либо задачи, на которые он откликнулся;

3) ProfileVC – контроллер, отображающий данные и содержащий разделы о пользователе;

4) TaskVC – контроллер, отображающий информацию о задаче;

5) CategoriesSearchVC – контроллер, отображающий категории для создания задачи;

6) CreateOfferVC – контроллер для создания задачи;

7) AddressesVC – контроллер, отображающий список пользовательских адресов;

8) CreateAddressVC – контроллер, отвечающий за добавление нового адреса;

9) PortfolioVC– контроллер, отображающий портфолио;

10) DocumentsVC – контроллер, отображающий документы;

11) SignInVC – контроллер с вводом номера телефона;

12) SignUpVC – контроллер, отвечающий за регистрацию пользоавтеля;

13) SmsCodeVC – контроллер, проверяющий код из смс.

2.4. Диаграммы деятельности

На рисунке 6 представлена диаграмма деятельности, описывающая процесс создания задачи. В процессе задействуются пользователь, представление, модель представления и сервер, на которые отправляются данные.

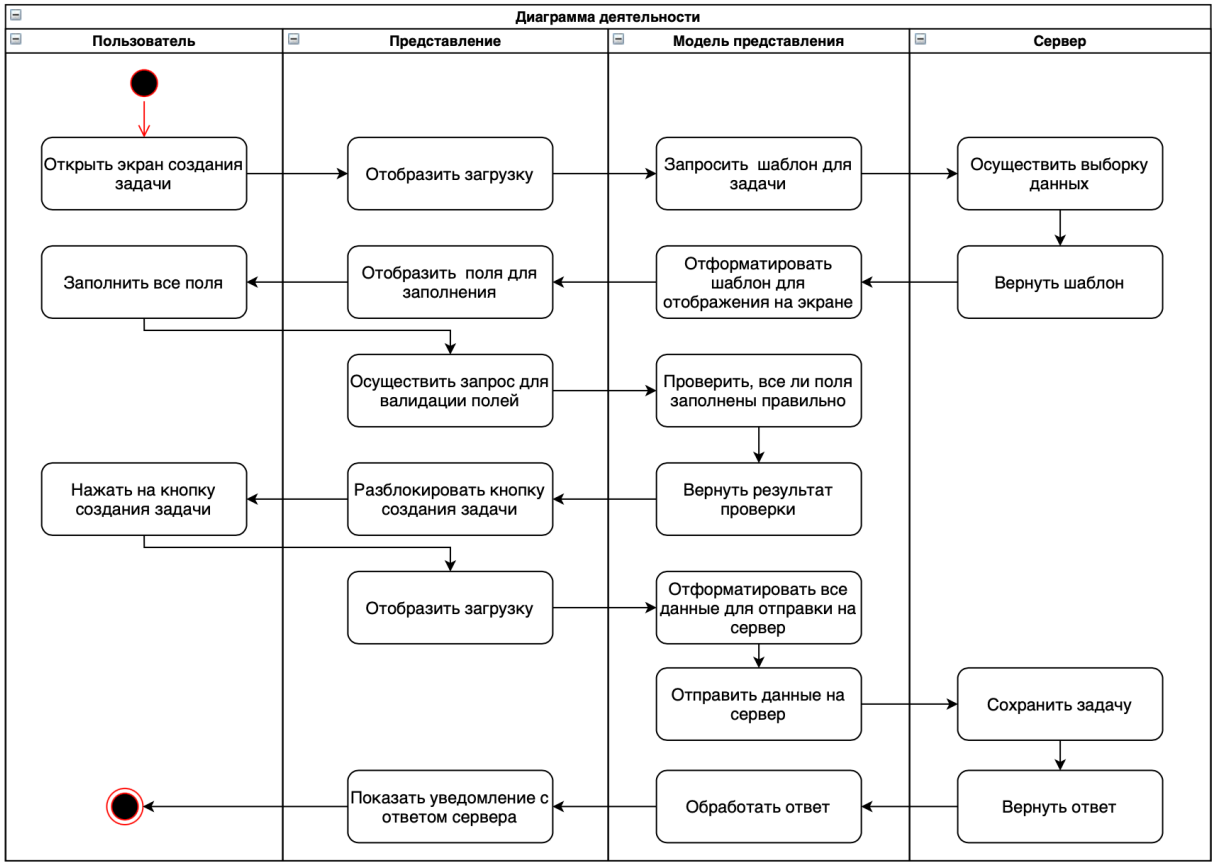

Рисунок 6 – Процесс создания задачи

Диаграмма деятельности демонстрирует взаимодействие как пользователя с приложением, так и его отдельных компонентов между собой. Для начала пользователю надо открыть экран создания задачи, подождать, пока система получит и отформатирует все нужные данные с сервера. Далее, ему необходимо заполнить все поля, система проверит, все ли данные заполнены корректно, и разблокирует кнопку создания задачи. Пользователю необходимо нажать на кнопку создания задачи. Система сформирует запрос к серверу, который, в свою очередь, сохранит задачу и вернет ответ мобильному приложению для отображения пользователю.

2.5. Дизайн-система

На рисунке 7 представлены основные цвета приложения. Они были выбраны на основе цветов стандартных приложений системы с включенной темной темой.

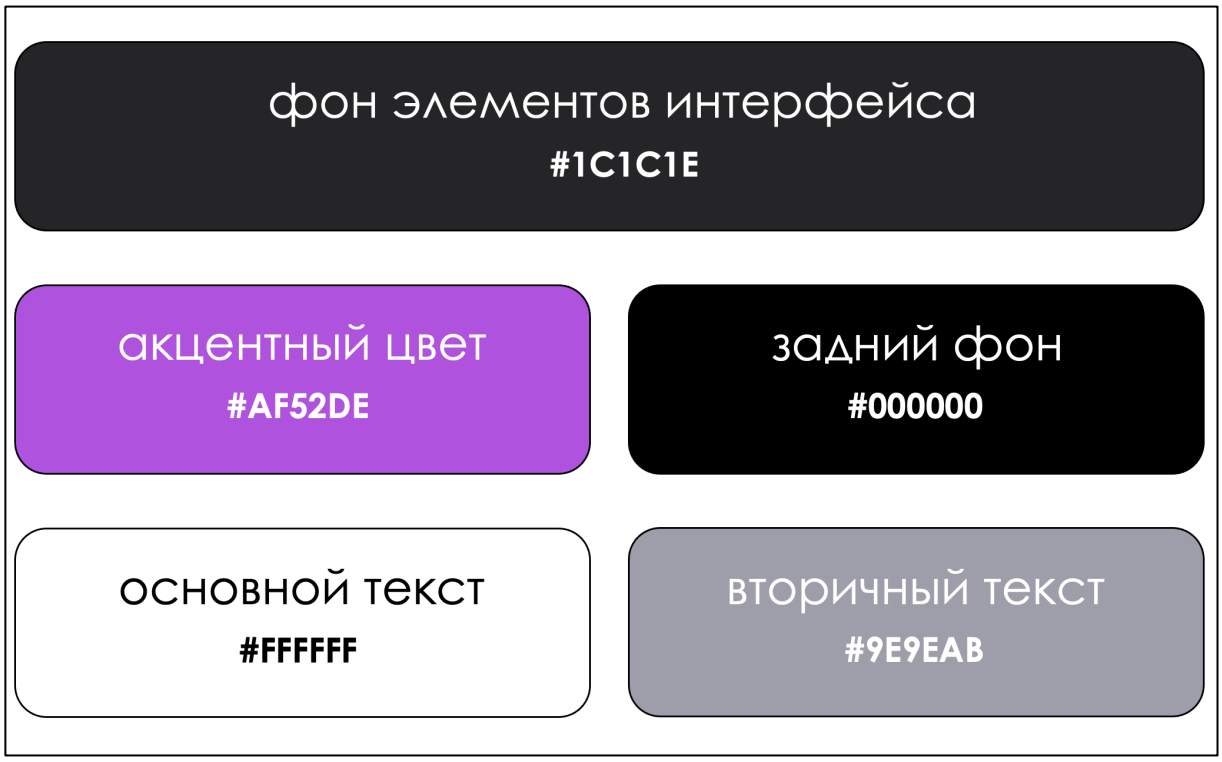

Рисунок 7 – Основные цвета приложения

На рисунке 8 представлен один из элементов интерфейса, выполненный в представленной выше цветовой палитре.

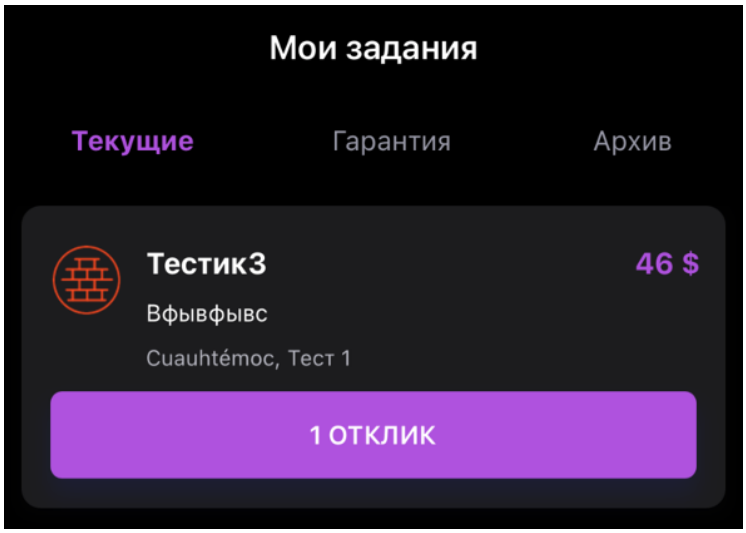

Рисунок 8 – Элемент интерфейса

3. РЕАЛИЗАЦИЯ

3.1. Выбор инструментов разработки

Для решения поставленной задачи существует два подхода: кроссплатформенная и нативная разработка.

Кроссплатформенная разработка – разработка, при которой разработчик реализует приложение сразу для несколько платформ. Для этого используются такие языки программирования, как, например, React Native [9] или Kotlin Multiplatform [10].

К достоинствам React Native можно отнести следующие:

1) достаточно стабильное решение, с помощью которого реализованы некоторые крупные приложение, например, Instagram [11];

2) разработчик создает приложение сразу для двух платформ.

К недостаткам использования React Native можно отнести следующие:

1) технология представлена не окончательно;

2) невозможность реализации некоторых нативных функций.

К достоинствам Kotlin Multiplatform можно отнести экономию времени, так как разработчики разных приложений на разные платформы могут писать одновременно код разных модулей, а потом интегрировать код друг друга.

К недостаткам использования Kotlin Multiplatform относится следующее:

1) многопоточность работает через технологии, не поддерживаемые в языке программирования Swift, поэтому приходится искать пути решения;

2) новая технология, вследствие чего существует не так много литературы;

3) некоторые задачи необходимо писать на разных языках программирования, поэтому все равно необходимо два разработчика.

Нативная разработка – разработка, при которой программисты

используют язык программирования, предоставляемый компанией-разработчиком платформы. Для платформы iOS таким языком программирования является Swift [12].

К плюсам Swift можно отнести следующее:

1) использование технологий, поддерживаемых только платформой iOS;

2) использование стандартных для платформы iOS элементов интерфейса, что хорошо сказывается на пользовательском опыте;

3) лучшая оптимизация приложения.

Основным недостатком Swift является то, что приложение реализуется только для платформы iOS.

Для решения задачи, исходя из анализа существующих решений, был выбран язык программирования Swift. Для нативной разработки мобильных приложений для платформы iOS часто используют интегрированную среду разработки (IDE) компании Apple Xcode. Она предоставляет разработчикам инструменты для создания приложений под iPhone, iPad, Mac, Apple Watch и Apple TV.

3.2. Реализация моделей

Модели разрабатываемой системы содержат в себе информацию о пользователе, задачах, адресах, документах, категориях, токенах. Было реализовано 8 моделей:

1) TaskModel – сущность, содержащая такую информацию о задаче, как название, описание, цена выполнения, адрес и фотографии;

2) AddressModel – сущность, содержащая информацию о стране, городе, улице и доме места исполнения задачи;

3) PortfolioModel – сущность, содержащая такую информацию об исполнителе, как языки, на которых он говорит и адрес мастерской;

4) UserModel – сущность, содержащая имя, фамилию, аватар пользователя и т.д.;

5) SocialNetworkModel – сущность, содержащая информацию о социальной сети, привязанной к аккаунту для быстрого входа в приложение;

6) TokensModel – сущность, содержащая информацию о токенах авторизации;

7) CategoryModel – сущность, содержащая информацию о категории, а именно название и ее подкатегории;

8) DocumentsModel – сущность, содержащая фотографии паспорта и пользователя с паспортом.

3.3. Реализация контроллеров

В контроллерах содержится логика отображения элементов на экране мобильного устройство и логика реакции на действия пользователя. Также контроллер подписывается на изменения модели представления для отображения изменений на экране.

Контроллер MyTasksVC отвечает за экран, на котором отображаются задачи пользователя в виде списка. Для его отображения используется класс UITableView [13], а контроллер реализует два делегата [14]: UITableViewDelegate и UITableViewDataSource. Изначально контроллер отображает загрузку, затем, получив данные от модели представлений, отображает список. Также контроллер подписывается на изменения модели представлений и обновляется каждый раз, когда она изменяется. На рисунке 9 представлен листинг методов делегатов, а на рисунке 10 показана подписка с помощью замыкания [15] на модель представлений.

В методе `tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell` происходит конфигурация ячейки таблицы. А если на экране отображается последняя ячейка, то вызывается запрос следующих элементов. Метод `tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int` отвечает за количество отображаемых ячеек, а метод `tableView(_`

tableView: UITableView, didSelectRowAt indexPath: IndexPath) обрабатывает нажатие на ячейку. Кроме того, в методе tableView(_ tableView: UITableView, estimatedHeightForRowAt indexPath: IndexPath) -> CGFloat устанавливается высота ячейки, а в методе tableView(_ tableView: UITableView, estimatedHeightForRowAt indexPath: IndexPath) -> CGFloat – ее ожидаемая высота для более корректного отображения.

```
final var selectedTab: MyTaskTabModel? { tabs.first(\.$0.tab.isSelected) }
final var currentItemsCount: Int { selectedTab?.items.count ?? 0 }

final func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int { currentItemsCount }

final func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        guard let model = selectedTab?.items[indexPath.row], let identifier
= model.cell?.identifier else {
            return .init()
        }
        let cell = tableView.dequeueReusableCell(withIdentifier:
identifier, for: indexPath, model: model, delegate: self)

        if let selectedTab = selectedTab, indexPath.row ==
selectedTab.items.count - 1, selectedTab.total > selectedTab.items.count {
            viewModel.getOffers(offset: selectedTab.items.count, category:
selectedTab.category) }

        return cell
    }

final func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        guard indexPath.row < currentItemsCount, let offer =
selectedTab?.items[indexPath.row].offer  else { return }

        if offer.status != .warranty {
            openTaskCard(for: offer, at: indexPath)
        } else {
            openWarrantyCard(for: offer, at: indexPath)
        }
    }
final func tableView(_ tableView: UITableView, estimatedHeightForRowAt
indexPath: IndexPath) -> CGFloat {
        guard
            indexPath.row < currentItemsCount,
            let estimatedHeight =
selectedTab?.items[indexPath.row].cell?.estimatedHeight
        else { return ConstantsFile.estimatedCellHeight }
        return estimatedHeight
    }
    final func tableView(_ tableView: UITableView, heightForRowAt
indexPath: IndexPath) -> CGFloat {
        guard indexPath.row < currentItemsCount, height =
selectedTab?.items[indexPath.row].cell?.height { return
UITableView.automaticDimension }
        return height }
```

Рисунок 9 – Листинг обработки делегатов.

```
private func bindViewModel() {
        viewModel.itemsDidChange = { [weak self] items in
            for (
                 index,
                 tab
            ) in self.tabs.enumerated() where tab == self.selectedTab
         {
                self.tabs[index].items = items
                self.hideLoading()
                self.tableView.reloadData()
          }
    }
```

Рисунок 10 – Листинг механизма привязки данных

В представленном методе представлен механизм связывания – как только элементы в модели представления будут обновляться, будет вызываться этот участок кода, в котором происходит обновление отображения таблицы в контроллере.

3.4. Реализация модели представления

Для реализации моделей представления был разработан базовый класс BaseViewModel, содержащий в себе координатор-дженерик [16] для навигации и такие общие методы, как обработка ошибки. На рисунке 11 представлен листинг этого класса.

Модель представления MyTasksViewModel отправляет запрос в сеть, обрабатывает и преобразует полученные данные, после чего отправляет их контроллеру для отображения. На рисунке 12 представлен листинг запроса в сеть.

```
class BaseViewModel<C: Coordinator> {

    weak var coordinator: C?

    init(coordinator: C) {
        self.coordinator = coordinator
    }

    func handleError(error: APIError) {
        coordinator.presentErrorBottomSheet(with: error)
    }

}
```

Рисунок 11 – Листинг класса BaseViewModel

```
private var items: [OffersModel] = [] { didSet { itemsDidChange(items) }}
    func getOffers(offset: Int, category: CategoryModel) {
        let request = GetOffersAction.Request(offset: offset, category:
category)
        send(request) { [weak self] result in
            switch result {
            case let .success(offers):
                self?.items = offers
            case let .failure(error):
                self?.handleError(error: error)
            }
        }
    }
```

Рисунок 12 – Листинг запроса в сеть

В представленном участке кода происходит запрос в сеть внутри модели представлений. Сначала создается модель запроса, потом вызывается сам запрос в сеть. Когда приходит результат – срабатывает код внутри замыкания и обрабатывается ответ – если запрос прошел успешно, то элементы обновляются и вызывается код для обновления таблицы внутри представления (это описано ранее). В случае ошибки вызывается метод `handleError(error: APIError)`, который отображает ошибку на экран.

3.5. Реализация интерфейсов

Для реализации интерфейсов в разработанном мобильном приложении был выбран стандартный инструмент создания интерфейса для мобильных iOS-приложений – InterfaceBuilder. Данный интерфейс предоставляет коллекции объектов пользовательского интерфейса, такие как: формы для ввода даты, таблицы данных, текстовые поля, кнопки и тому подобное.

Для корректного отображения представлений на разных устройствах на платформе iOS использовался Auto Layout [17]. Auto Layout динамически вычисляет позиции и размеры всех объектов в иерархии объектов на основе правил, заданных для того или иного объекта. Эти правила называются layout constraints и задаются вручную, либо с помощью встроенных функций Xcode. Для создания связи с кодом

используются поля класса Outlet, Action, OutletCollection, ссылающиеся на конкретные объекты в InterfaceBuilder.

На рисунке 13 представлена ячейка задачи внутри Interface Builder.

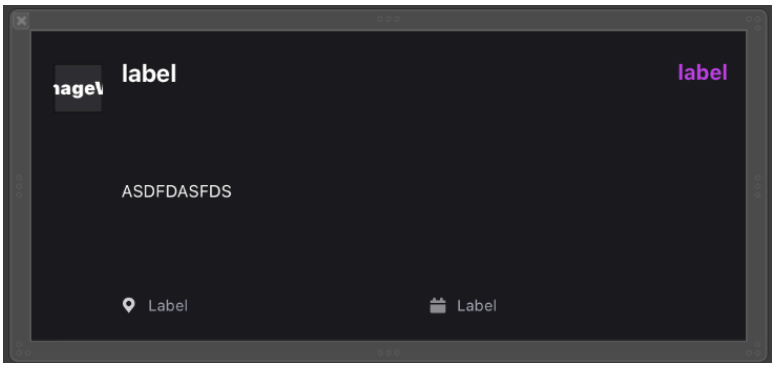

Рисунок 13 – Ячейка задачи внутри Interface Builder

В процессе разработки системы было создано 11 storyboard-файлов [18], содержащих графический интерфейс экранов и связи между ними. На рисунке 14 представлен файл Main.storyboard.

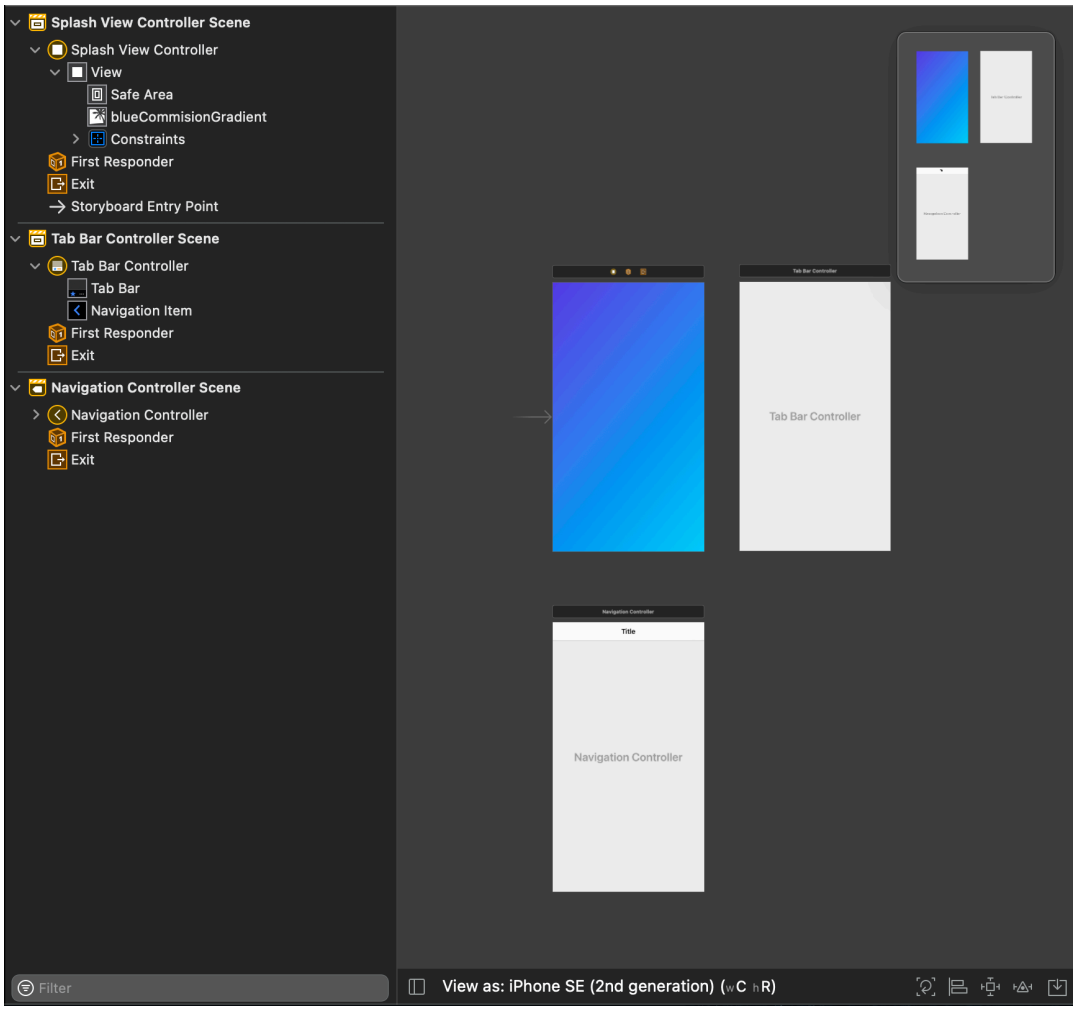

Рисунок 14 – Файл Main.storyboard

3.6. Реализация взаимодействия с локальной базой данных

В качестве локальной базы данных используются UserDefaults [19] и Keychain [20]. Второй инструмент отличается своей безопасностью, поэтому он был выбран для хранения токенов. На рисунке 15 представлен листинг property wrapper'ов [21] для взаимодействия с ними.

```
@propertyWrapper
struct UserDefault<T> {

    private let key: String
    private let defaultValue: T
    private let userDefaults = UserDefaults.standard

    var wrappedValue: T {
        get { userDefaults.object(forKey: key) as? T ?? defaultValue }
        set { userDefaults.set(newValue, forKey: key) }
    }

    init(key: Key, defaultValue: T) {
        self.key = key.rawValue
        self.defaultValue = defaultValue
    }

}

extension UserDefault {

    enum Key: String {
        case firstName
    }

}
```

Рисунок 15 – Листинг взаимодействия с UserDefault

```
@propertyWrapper
struct Keychain {
    private let service: String
    private let account: String
    private let defaultValue: String
    var wrappedValue: String? {
        get {
do {
                return try KeychainItem(service: service, account:
account).readItem()
            } catch { return nil }
        }
        set {
            do {
                if let newValue = newValue {
                    try KeychainItem(service: service, account:
account).saveItem(newValue)
                } else { removeValue() }
            } catch {}
```

Рисунок 16 – Листинг взаимодействия с KeyChain

На рисунке 16 продемонстрирован propertyWrapper для взаимодействия с Keychain. В get-части переменной элемент достается из памяти, а в set-части переменной – помещается в память под нужным ключом.

```
@@UserDefault(key: .firstName, defaultValue: "") private var firstName:
String
```

Рисунок 17 – Пример взаимодействия с UserDefault

На данном участке кода на рисунке 17 из базы данных берется переменная `firstName`. Если такого значения не существует, то данная переменная будет принимать значение, указанное в `defaultValue` – в данном случае это пустая строка.

3.7. Вспомогательные средства реализации

Для интеграции сторонних библиотек используется CocoaPods [22] и Swift Package Manager [23]. В проекте используются следующие библиотеки:

1) библиотека SwaggerClient – используется для кодогенерации данных с сервера;

2) библиотека Alamofire – используется для работы с сетевыми http(s) запросами;

3) библиотека SDWebImage – используется для асинхронной загрузки и кэширования изображений из сети;

4) библиотека FBSDKLoginKit – используется для входа в аккаунт через социальную сеть Facebook;

5) библиотека GoogleSignIn – используется для входа в аккаунт через Google;

6) библиотека GooglePlaces – используется для получения почтового индекса с помощью адреса и наоборот.

4. ТЕСТИРОВАНИЕ

Для разработанной системы было выполнено функциональное и интеграционное тестирование.

4.1. Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения исходным функциональным требованиям. Протокол тестирования на функциональность представлен в таблице 1. Все тесты были пройдены успешно.

Таблица 1 – Протокол тестирования

№	Описание	Шаги	Ожидаемый результат
1	Войти в аккаунт в приложении	1. Запустить мобильное приложение 2. Ввести номер телефона 3. Нажать на кнопку «Получить код» 4. Ввести смс-код	На экране отобразится список актуальных задач пользователя.
2	Просмотр списка задач	1. Нажать на кнопку бокового меню 2. Нажать на кнопку «Задачи»	На экране отобразится список задач в виде таблицы
3	Отфильтровать задачи	1. Нажать на кнопку меню 2. Нажать на кнопку «Задачи» 3. Нажать на кнопку «Фильтр» 4. Выбрать требуемый фильтр	На экране отобразятся только те задачи, которые подходят по параметру фильтрации
4	Создать задачу	1. Нажать на кнопку меню 2. Нажать на кнопку «Создать задачу» 3. Выбрать нужную категорию 4. Выбрать нужную подкатегорию 5. Заполнить все обязательные поля 6. Нажать на кнопку «Создать»	Открывается список задач пользователя с новой созданной задачей
5	Откликнуться на задачу	1. Нажать на кнопку бокового меню 2. Нажать на кнопку «Задачи» 3. Нажать на нужную задачу 4. Нажать на кнопку «Откликнуться» 5. Ввести свою цену и предлагаемое время исполнения 7. Нажать на кнопку «Откликнуться»	Открывается экран задачи с созданным откликом

Окончание таблицы 1

№	Описание	Шаги	Ожидаемый результат
6	Выбрать исполнителя	1. Нажать на кнопку бокового меню 2. Нажать на кнопку «Мои задачи» 3. Нажать на нужную задачу 4. Нажать на кнопку «Отклики» 5. Нажать на исполнителя 6. Нажать на кнопку «Выбрать исполнителя»	Открывается экран задачи с выбранным исполнителем
7	Обратиться в техническую поддержку	1. Нажать на кнопку бокового меню 2. Нажать на кнопку «Обратная связь» 3. Выбрать способ обращения 4. Связаться с технической поддержкой согласно выбранного способа связи	При выборе способа связи через звонок открывается приложение телефона с веденным номером телефона. При выборе способа связи через сообщение открывается форма ввода обращения
8	Выйти из профиля	1. Нажать на кнопку бокового меню 2. Нажать на кнопку «Профиль» 3. Нажать на кнопку «Выйти»	Откроется экран ввода номера телефона

4.2. Интеграционное тестирование

Интеграционное тестирование – тип тестирования, при котором происходит полная проверка разработанной системы с целью выявления ошибок [24].

Для проверки корректности отображения UI-элементов разработанная система была проверена на следующих iOS-устройствах, имеющих разную диагональ экрана: iPhone 6s, iPhone 6s Plus, iPhone X, iPhone 12, iPhone 12 Pro Max, iPhone 12 Mini. В результате проверки было выяснено, что ошибок отображения интерфейса нет.

Также разработанное приложение было проверено на следующих версиях операционной системы: iOS 11, iOS 12, iOS 13, iOS 14. В результате тестирования ошибок не было обнаружено.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы мною была изучена технология создания мобильного приложения для операционной системы iOS на примере реализации простейшего приложения для поиска мастера на час. Приложение включает в себя работу с сетью, а также дополнительное локальное хранение данных для получения доступа к ним в оффлайн-режиме. Объем исходного кода мобильного приложения составил 5198 строк.

Для достижения цели были решены задачи:

1) выполнен анализ предметной области и сравнение схожих приложений;
2) выполнено проектирование мобильного приложения;
3) реализовано мобильное приложение;
4) рассмотрены и произведены два вида тестирования мобильного приложения.

ЛИТЕРАТУРА

1. Онлайн-шопинг в цифрах: главная статистика. [Электронный ресурс] URL: https://www.shopolog.ru/metodichka/analytics/onlayn-shoping-v-cifrah-glavnaya-statistika/ (дата обращения 20.03.2021 г.).

2. Исследование онлайн-продаж в период карантина. [Электронный ресурс] URL: https://www.shopolog.ru/metodichka/analytics/issledovanie-onlayn-prodazh-v-period-karantina/ (дата обращения 23.03.2021 г.).

3. Apple стала крупнейшим в мире продавцом смартфонов с 23,4% мирового рынка. [Электронный ресурс] URL: https://www.forbes.ru/newsroom/tehnologii/419677-apple-stala-krupneyshim-v-mire-prodavcom-smartfonov-s-234-mirovogo-rynka (дата обращения 22.03.2021 г.).

4. What is UML. [Электронный ресурс] URL: https://www.uml.org/what-is-uml.htm/ (дата обращения 24.03.2021 г.).

5. Model View ViewModel Theory. [Электронный ресурс] URL: https://www.raywenderlich.com/books/advanced-android-app-architecture/v1.0/chapters/10-model-view-viewmodel-theory (дата обращения 24.03.2021 г.).

6. Don’t Repeat Yourself. [Электронный ресурс] URL: https://medium.com/code-thoughts/dont-repeat-yourself-caa413910753 (дата обращения 24.03.2021 г.).

7. Alamofire. Elegant networking solution in swift. [Электронный ресурс] URL: https://github.com/Alamofire/Alamofire (дата обращения 26.03.2021 г.).

8. Swagger Codegen. [Электронный ресурс] URL: https://swagger.io/tools/swagger-codegen/ (дата обращения 26.03.2021 г.).

9. React Native. [Электронный ресурс] URL: https://reactnative.dev (дата обращения 05.04.2021 г.).

10. Kotlin multiplatform programming. [Электронный ресурс] URL: https://kotlinlang.org/docs/multiplatform.html (дата обращения 05.04.2021 г.).

11. React Native at Instagram. [Электронный ресурс] URL: https://instagram-engineering.com/react-native-at-instagram-dd828a9a90c7 (дата обращения 05.04.2021 г.).

12. Swift. [Электронный ресурс] URL: https://developer.apple.com/documentation/swift (дата обращения 05.04.2021 г.).

13. UITableView. [Электронный ресурс] URL: https://developer.apple.com/documentation/uikit/uitableview (дата обращения 16.04.2021 г.).

14. Protocols. [Электронный ресурс] URL: https://docs.swift.org/swift-book/LanguageGuide/Protocols.html (дата обращения 16.04.2021 г.).

15. Closures. [Электронный ресурс] URL: https://docs.swift.org/swift-book/LanguageGuide/Closures.html (дата обращения 16.04.2021 г.).

16. Generics. [Электронный ресурс] URL: https://docs.swift.org/swift-book/LanguageGuide/Generics.html (дата обращения 17.04.2021 г.).

17. Understanding Auto Layout. [Электронный ресурс] URL: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html (дата обращения 18.04.2021 г.).

18. Designing with Storyboards. [Электронный ресурс] URL: https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/DesigningwithStoryboards.html (дата обращения 19.04.2021 г.).

19. UserDefaults. [Электронный ресурс] URL: https://developer.apple.com/documentation/foundation/userdefaults (дата обращения 24.03.2021 г.).

20. Storing Keys in the Keychain. [Электронный ресурс] URL: https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_keychain (дата обращения 24.03.2021 г.).

21. Property Wrapper in swift. [Электронный ресурс] URL: https://medium.com/nerd-for-tech/property-wrapper-in-swift-e56e3854650b

(дата обращения 10.05.2021 г.).

22. CocoaPods. [Электронный ресурс] URL: https://cocoapods.org (дата обращения 15.05.2021 г.).

23. Package Manager. [Электронный ресурс] URL: https://swift.org/package-manager/ (дата обращения 15.05.2021 г.).

24. Интеграционное тестирование: что это? [Электронный ресурс] URL: https://logrocon.ru/news/intgration_testing (дата обращения 18.05.2021 г.).