ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Преподаватель департамента программной инженерии факультета компьютерных наук

_____ М. К. Горденко
«_____» _____ 2019 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2019 г.

IOS-ПРИЛОЖЕНИЕ «СОЦИАЛЬНАЯ СЕТЬ ДЛЯ СОТРУДНИКОВ НИУ ВШЭ»

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.03-01 81 01-1-ЛУ

Исполнитель: студент группы БПИ174
_____ Д. Ю. Редникина
«_____» _____ 2019 г.

Москва 2019

УТВЕРЖДЕН
RU.17701729.04.03-01 81 01-1-ЛУ

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

IOS-ПРИЛОЖЕНИЕ «СОЦИАЛЬНАЯ СЕТЬ ДЛЯ СОТРУДНИКОВ НИУ ВШЭ»

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.03-01 81 01-1-ЛУ

Листов 54

Москва 2019

Содержание

1 Введение

1.1 Наименование программы

1.1.1 Наименование программы на русском языке

iOS-приложение «Социальная сеть для сотрудников НИУ ВШЭ»

1.1.2 Наименование программы на английском языке

iOS application «Social network for HSE staff»

1.2 Документы, на основании которых ведется разработка

1. Приказ декана факультета компьютерных наук Национального Исследовательского университета «Высшая школа экономики» № 2.3-02/1012-0 2 от 10.12.18.
2. Техническое задание «iOS-приложение «Социальная сеть для сотрудников НИУ ВШЭ».

2 Назначение и область применения

2.1 Функциональное назначение

Программа применяется как средство коммуникации между сотрудниками НИУ ВШЭ. Сервис также позволяет сотрудникам НИУ ВШЭ координировать совместные исследования и проекты научного характера. Приложение предоставляет доступ к общению посредством сообщений, дает возможность делиться новостями с помощью публикации постов. Также приложение дает возможность фильтровать новости по интересущим темам.

2.2 Эксплуатационное назначение

Программа будет использоваться как средство общения и огранизации работы над совместными научными проектами между сотрудниками с одного или разных направлений/ факультетов/ кампусов.

Таким образом, программный продукт позволит решить проблему коммуникации между факультетами, кампусами НИУ ВШЭ и даст возможность наладить взаимодействие между преподавателями университета, будет способствовать развитию научной деятельности между разными факультетами и кампусами.

2.3 Область применения

НИУ ВШЭ очень большой вуз с кампусами в разных городах России. Преподавателям сложно общаться между собой, узнавать последние новости, связанные с написанием статей, научно-исследовательской деятельностью коллег. Разрабатываемая программа позволит наладить взаимодействие преподавателей и будет способствовать развитию научной деятельности и коммуникации между факультетами и кампусами университета.

Задача социальной сети заключается в обеспечении коммуникации, обмене актуальной научной и общеуниверситетской информацией, знакомстве научных работников и преподавателей с разных факультетов и кампусов.

Разрабатываемая программа является прекрасной площадкой для начала исследований, обсуждения научных работ, общения по научным и университетским тематикам для сотрудников университета НИУ ВШЭ.

3 Технические характеристики

3.1 Постановка задачи на разработку программы

Программа выполняется в рамках темы курсовой работы в соответствии с учебным планом подготовки бакалавров по направлению 09.03.04 «Программная инженерия» Национального исследовательского университета «Высшая школа экономики», факультет компьютерных наук.

Разработка требований велась совместно с командой и заказчиком в рамках предмета «Групповая динамика и коммуникации в программной инженерии»

Состав команды:

- Анна Михалева - android-developer;
- Константин Манежин - web-developer;
- Илья Костюченко - backend-developer;
- Дарья Редникина - iOS-developer;

Заказчик:

- Девятьярова Анна Дмитриевна, Факультет бизнеса и менеджмента/ Кафедра управления человеческими ресурсами.

FR-1. **Авторизация клиента**
Чтобы использовать программу, клиент должен иметь возможность авторизоваться в системе.

FR-1.1. При регистрации в социальной сети клиенту необходимо заполнить обязательные поля регистрации:

1. (*Verified, автор: заказчик*)
   ФИО;
2. (*Verified, автор: заказчик*)
   Факультет;
3. (*Verified, автор: заказчик*)
   Почта;
4. (*Verified, автор: заказчик*)
   Должность;
5. (*Verified, автор: заказчик*)
   Город;

FR-1.2. (*Verified, автор: Илья*)
Уже зарегистрированный клиент для входа в социальную сеть должен ввести свою почту с доменом `@hse.ru`.

FR-1.3. (*Verified, автор: Анна*)
После успешной авторизациии/регистрации пользователю будет выслан код на введенную им почту, который нужно ввести в специальное поле в приложении, только после правильного ввода кода клиент сможет войти в социальную сеть.

FR-2. **Просмотр профиля пользователя**

FR-2.1. (*Verified, автор: Дарья*)
Должна быть возможность редактирования у пользователей выше перечисленных полей (см. требование Постановка задачи на разработку программы), заполненых при регистрации, в настройках профиля.

FR-2.2. (*Verified, автор: команда, Анна*)
На странице профиля должна быть возможность просмотра раннее опубликованных постов человека.

FR-2.3. (*Verified, автор: Константин, заказчик*)
Также на странице должна быть возможность просмотра личных данных пользователя, то есть информации из требования Постановка задачи на разработку программы

FR-2.4. (*Verified, автор: Анна*)
На странице пользователя должна быть возможность написать сообщение пользователю в диалог.

FR-2.5. (*Verified, автор: Анна*)
На странице пользователя должна быть возможность добавить пользователя в существующий канал.

FR-3. **Публикация постов**
Клиент имеет возможность опубликовать текстовую информацию от своего имени, чтобы она отображалась в ленте у других пользователей приложения и у него в профиле.

FR-3.1. (*Verified, автор: Илья*)
При написании поста у пользователя есть возможность добавить хэштеги к текстовой информации.

1. Хэштеги должны состоять из одного слова
2. Хэштеги должны состоять из латинских и русских букв, допустимые символы при написании хэштега: нижнее подчеркивание, цифры.
3. При неверном формате введенного хэштега он не будет опубликован вместе с написанным постом.

FR-3.2. (*Verified, автор: Константин*)
При выборе хэштегов пользователю должны предлагаться autosuggested hashtags, раннее использованные в приложении другими пользователями при публикации постов.

FR-3.3. (*Verified, автор: Анна*)
Должна быть поддержка разметки markdown, syntax highlighting при написании поста.

FR-3.4. (*Verified, автор: Дарья*)
При выходе из раздела создания поста, должна быть возможность сохранить черновик с текущем текстом и набором хэштегов.

FR-4. **Лента**
Пользователь должен имеет возможность, находясь в ленте, совершать поиск по интересущим его хэштегам, смотреть новости.

FR-4.1. (*Verified, автор: Дарья*)
При входе в основную ленту должна быть возможность отображения всех существующих постов только в хронологическом порядке.

FR-4.2. (*Verified, автор: Константин*)
Должна быть возможность осуществления перехода при нажатии на какой-либо хэштег в ленту всех постов с выбранным хэштегом.

FR-4.3. (*Verified, автор: Илья*)
Должна быть возможность подсказки autosuggested hashtags при поиске нужной информации в поисковой строке в ленте.

FR-4.4. (*Verified, автор: Илья*)
Должна быть возможность отображения как preview поста в ленте, так и его полного содержания в отдельном окне при нажатии.

FR-4.5. (*Verified, автор: Анна*)
Должна быть возможность перехода на страницы авторов постов, отображенных в ленте.

FR-5. **Каналы [4.2]**
У клиента приложения должна быть возможность сохранять поисковые фильтры для быстрого доступа к просмотру ленты по нужному множеству хэштегов и людей.

FR-5.1. (*Verified, автор: Константин*)
Должна быть возможноть создания канала, который должен иметь:

1. Название;
2. Еединое множество людей и хэштегов;
3. Функцию «предпросмотр канала»;

FR-5.2. (*Verified, автор: Дарья*)
Должна быть возможноть редактирования каналов, где можно:

1. Изменить название канала;
2. Изменить множество выбранных хэштегов и людей;
3. Перейти к предпросмотру канала;

FR-5.3. (*Verified, автор: Константин*)
Должна быть возможность автоподсказки по хэштегам при редактировании канала;

FR-5.4. (*Verified, автор: заказчик, Анна*)
Просмотр содержимого канала;

FR-5.5. (*Verified, автор: Константин*)
Удаление каналов;

FR-5.6. (*Verified, автор: Анна*)
Должна быть возможность поиска по названию созданных каналов;

FR-6. **Создание чатов для пользователей**
Клиент должен иметь возможность общаться с другими пользователями социальной сети: обмениваться сообщениями в диалогах и групповых беседах.

FR-6.1. (*Verified, автор: Анна, заказчик*)
Должна быть возможность создать групповую беседы (добавление участников, названия чата) размером от 2 до 50 пользователей

FR-6.2. (*Verified, автор: Илья*)
При правах администратора беседы клиент должен иметь возможность:

1. Менять состав участников: добавлять или удалять;
2. Менять название;
3. Делать администраторами других людей из беседы;
4. Лишать их возможности быть администратором;

FR-6.3. (*Verified, автор: Дарья*)
Создатель беседы автоматически должен являться администратором.

FR-6.4. (*Verified, автор: Константин*)
Должна быть возможность просматривать информацию о беседе (состав участников, название беседы, список администраторов).

FR-6.5. (*Verified, автор: Илья*)
Должна быть возможность выйти из беседы.

FR-6.6. (*Verified, автор: Дарья*)
Должна быть возможность написать сообщение другому пользователю.

3.2 Описание алгоритмов и функционирования программы

3.2.1 Описание алгоритмов программы

3.2.1.1 Поиск предложенных хэштегов

Список всех существующих хэштегов хранится в виде дерева, полученного по запросу из JSON файла в структуре.

```
struct CompletionTree: Codable {
    var value: String?
    var subtree: [String : CompletionTree]
}
```

Листинг 1 — Структура объекта дерева автодополнений хэштегов

Сравнение со стандатным алгоритмом:

Стандартный поиск совпадений в массиве и отображении предложенных хэштегов выполняется за $O(q \cdot l)$ (методы из стандартной библиотеки `func filter { }`, `func contains { }` имеет временную сложность $O(q), O(l)$), где q – количество слов в массиве для поиска, а l - максимальная длина слова.

```
var filteredDataSource = [String]()
var dataSource = [String]()
func filterContentForSearchText(_ searchText: String)
{
    filteredDataSource = dataSource.filter {
        $0.lowercased().contains(searchText.lowercased()) }
}
```

Листинг 2 — Стандартный поиск совпадений в массиве за $O(q * l)$

Поиск совпадений в структуре дерева (см. 3.2.1.1) выполняется за $O(n+m)$, где n – количество элементов дерева (введенных букв), m – количество найденных результатов поиска.

Доказательство:

Так как дерево представлено структурой (см. 3.2.1.1), то каждое поддерево доступно за $O(1)$ (поиск по ключу в словаре `subtree`). Пусть n – количество элементов дерева, тогда обход дерева n (поиск поддеревьев, соответсвующих всем введенным буквам), а взятие результата сводится к m, где m – количество результатов поиска. Итого сложность алгоритма $O(n+m)$.

```
static func getCompletion(tree: CompletionTree, word: String) -> [String] {
    if word == "" {
        return getValues(tree: tree)
    }

    let character = String(word.first!)

    if tree.subtree[character] == nil {
        return []
    } else {
        return getCompletion(tree: tree.subtree[character]!,
                             word: String(word.dropFirst()))
    }
}

static func getValues(tree: CompletionTree) -> [String] {
    var out = [String]()

    if let treeVal = tree.value {
        out.append(treeVal)
    }

    for (_, subtree) in tree.subtree {
        out += getValues(tree: subtree)
    }

    return out
}
```

Листинг 3 — Функции для обхода дерева хэштегов

Выше приведен код для обхода дерева и поиска значений `value != nil` в массив предложенных хэштегов.

3.2.2 Описание схемы функционирования программы

3.2.2.1 Application Flow

Приложение разрабатывалось с использованием Coordinator pattern: взаимодействие внутри основных слоев приложения (LogIn, Channels, Messages, Profile) осуществлялось с помощью координатора (см. рис 1) - класса, который управлял основной логикой переходов между ViewControllers, передачей основных данных между общими частями приложения. Этот паттерн применяется в проектировании приложений для избежания повторения кода и отделения логики переходов между экранами в отдельный класс для быстрого изменения flow приложения в будущем.

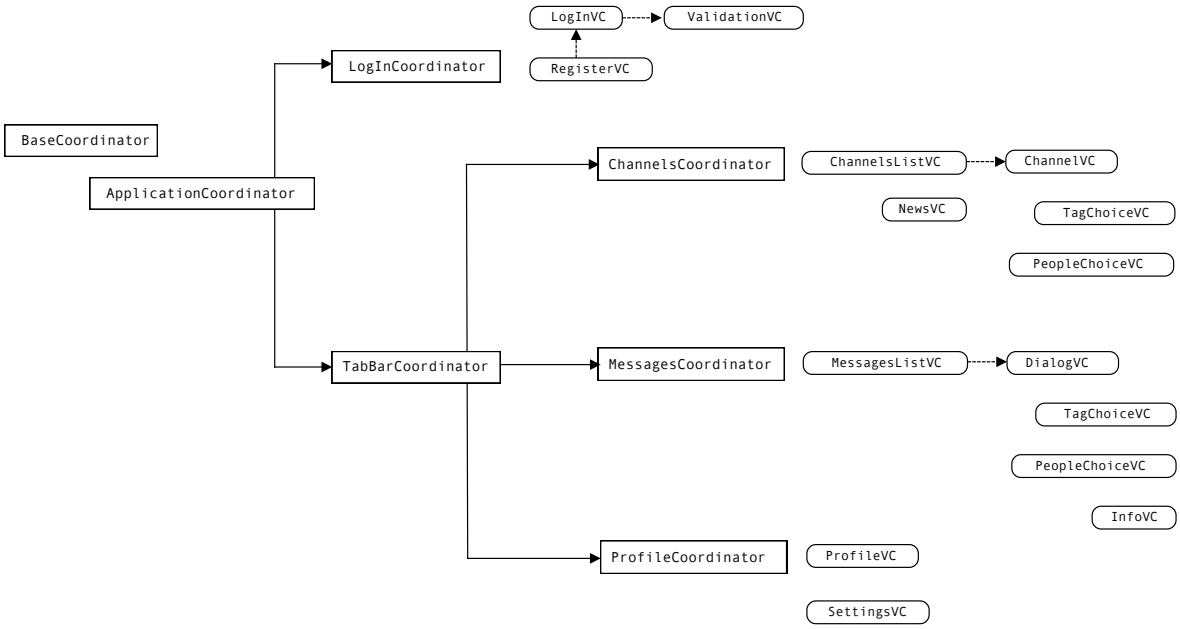

Рисунок 1 — Слои приложения в App Flow

Список слоев управления с описанием их зоны ответственности:

1. AppCoordinator: управление основным Flow программы.
   Наследуется от `BaseCoordinator`, который в свою очередь правляет зависимостями между координаторами и началом и окончанием работы координаторов.
   `AppCoordinator` принимает решение, кому передавать работу: схеме авторизации/входа в аккаунт или же начинать работу с авторизованным пользователем в `MainFlow`

    1. `LogInCoordinator` отвечает за переключение между авторизацией/регистрацией. В случае успешной авторизации передает работу `TabBarCoordinator`
    2. `TabBarCoodinator` отвечает за `MainFlow` программы. Управляет каналами, сообщениями, профилем до выхода из аккаунта (в этом случае передает работу AppCoodrinator, который в свою очередь делегирует работу `LogInCoordinator`)
        1. `ChannelsCoordinator` координирует работу каналов
        2. `MessagesCoordinator` координирует работу сообщений
        3. `ProfileCoordinator` координирует работу

3.2.2.2 Схема отправки запросов серверу и получения ответа

Так как приложение «Социальная сеть для сотрудников НИУ ВШЭ» является клиент-серверным приложением, то ключевая задача приложения - взаимодействие с сервером [3]: отправка и получение данных. Для корректного отправления и получения данных используется библиотека Alomofire [4] и возможности стандартной библиотеки iOS SDK (Codable protocol, URLrequest) [5].

Ниже описан класс стандартного запроса `GeneralRequest`. Этот класс позволяет сериализовать в JSON объект типа `GeneralRequest`, который используется для всех запросов постов/сообщений/постов в каналах:

```
struct GeneralRequest<T: Codable>: Codable
{
    var direction: String
    var limit: Int
    var exclusiveFrom: Int?
    var request: T

    init(direction: String = "forward", limit: Int, exclusiveFrom: Int?,
                                                        request: T) {
        self.direction = direction
        self.limit = limit
        self.exclusiveFrom = exclusiveFrom
        self.request = request
    }

    enum CodingKeys: String, CodingKey {
        case limit
        case exclusiveFrom
        case request
        case direction
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(limit, forKey: .limit)
        try container.encode(exclusiveFrom, forKey: .exclusiveFrom)
        try container.encode(request, forKey: .request)
        try container.encode(direction, forKey: .direction)
    }
}
```

Листинг 4 — Структура запроса на сервер

В классе `Model` содержатся статические методы для отправки конкретных запросов: `POST /posts/forUser`, `POST /posts/last`, `POST /channel/get`, `POST /messages/get/userChat`, `POST /messages/get/groupChat`. Отправка и получение запросов происходит по следующей схеме (ниже приведен пример реализации запроса и возвращения response от сервера):

1. Создание `URLRequest` с помощью url (iOS SDK)
2. Сериализация JSON объекта класса `Channels: Codable` (iOS SDK)
3. Отправка request и получение respose (Alomofire) и валидация полученного `statusCode` от сервера.

4. В некоторых случаях происходит десериализация полученного `JSONresponse` с помощью iOS SKD и через `@escaping` замыкание происходит возвращение полученного объекта

3.2.2.3 Схема работы с каналами

Ниже приведена визуализация управления слоя `ChannelsCoordinator`

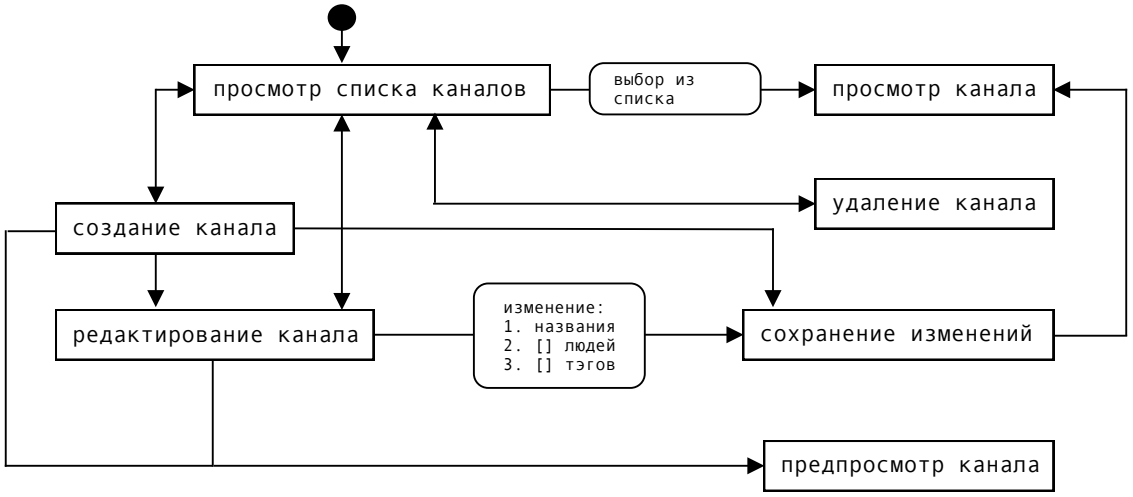

Рисунок 2 — Диаграмма состояний

Процесс работы клиента с каналами выглядит следующим образом (см. рис 2):

- Запрос данных на сервер для отображения списка каналов
  - При успешном запросе пользователь видет список каналов
  - При ошибке запроса (`statusCode != 200`) - демонстрируется пустая таблица
- При редактировании или создании канала происходит вызов методов updateChannel или createChannel
  - При успешном запросе пользователь видет только что созданный/измененный канал в списке каналов
  - При ошибке запроса (`statusCode != 200`) изменения к выбранному каналу не применяются

  Запрос на сервер осуществляется при выходе из режима редактирования канала с последующим обновлением списка каналов
- При выборе режима просмотра содержимого канала отправляется запрос на сервер
  - При успешном запросе пользователь видет ленту, отвечающую множестам подписок на канал
  - При ошибке запроса (`statusCode != 200`) пользователь видит пустую ленту канала

3.2.2.4 Схема работы с сообщениями

Ниже приведена визуализация слоя, отвечающего за работу с сообщениями:

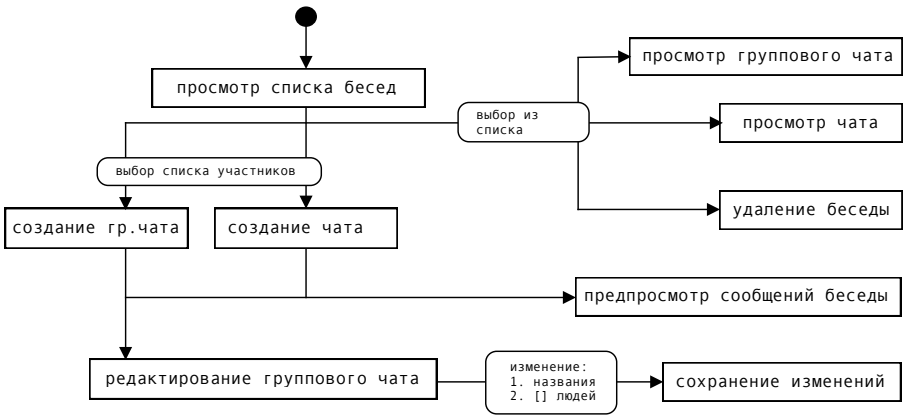

Рисунок 3 — Диаграмма состояний

Процесс работы клиента с сообщениями (см. рис 3, 4) выглядит следующим образом:

– Запрос данных на сервер для отображения списка диалогов

  – При успешном запросе пользователь видет список чатов и бесед

– При просмотре сообщений клиент имеет возможность пролистывать сообщения в обратном хронологическом порядке, для этого серверу отправляются запросы в формате «с какого сообщения», «сколько», «какое направление»

– При выходе из режима редактирования беседы происходит update с отправлением новой информации о чате на сервер

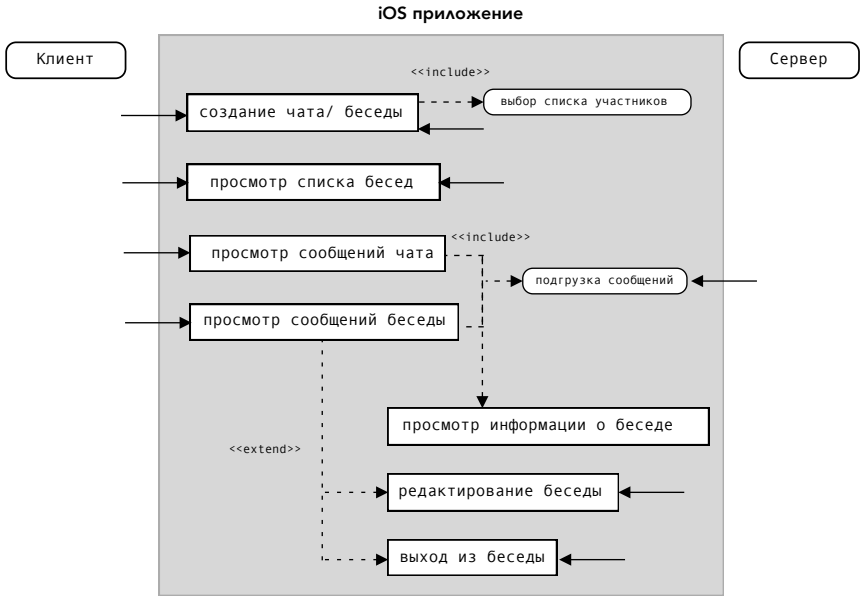

Рисунок 4 — Диаграмма прецедентов

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

3.2.2.5 Схема авторизации и регистрации клиента

При входе в аккаунт пользователю предлагается ввести свою почту (см. рис 5, 6):

1. Процесс входа уже зарегистрированного пользователя:
    1. Ввод email – происходит отправка запрос на сервер для валидации данных, получение ответа от сервера и cookies для header запросов
    2. При успешной валидации почты на сервере, происходит получение письма на введенную почту с кодом подтверждения
    3. Клиент вводит полученный код в специальное поле – при успешной валидации кода полученные cookies становятся активными и клиент успешно входит в приложение
2. Процесс регистрации:
    1. Ввод email – происходит отправка запрос на сервер для валидации данных
    2. При успешной валидации почты на сервере, происходит получение письма на введенную почту с кодом подтверждения
    3. Клиент вводит полученный код в специальное поле – при успешной валидации кода клиент получает cookies для header запросов
    4. Открывается окно с регистрацией: пользователю необходимо заполнить обязательные поля (см. раздел требований 1)
    5. После заполнения пользователь подтверждает создание учетной записи, объект с заполненныи данными пользователя отправляется на сервер, cookies становятся активными и клиент успешно входит в приложение

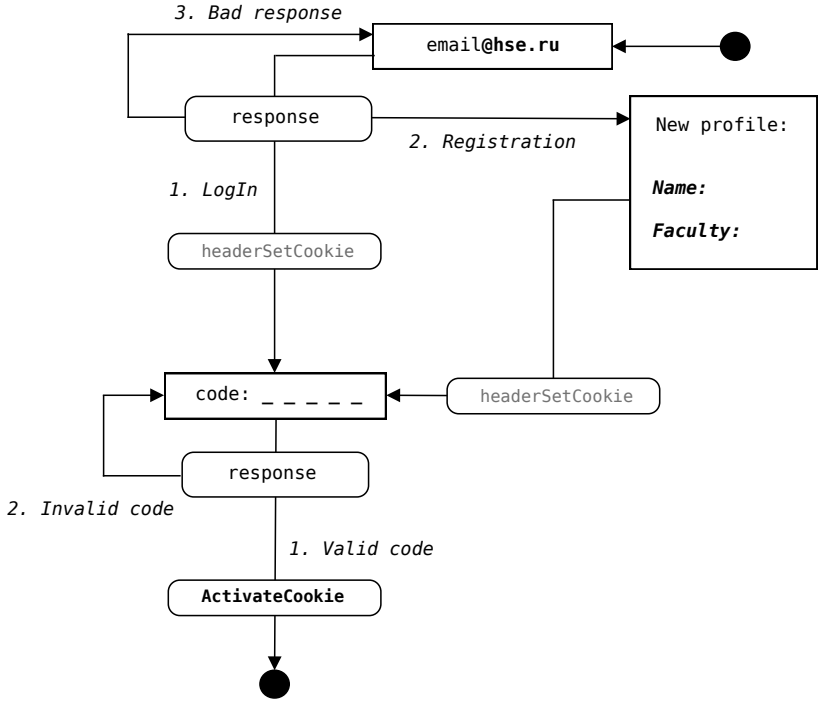

Рисунок 5 — Диаграмма состояний

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

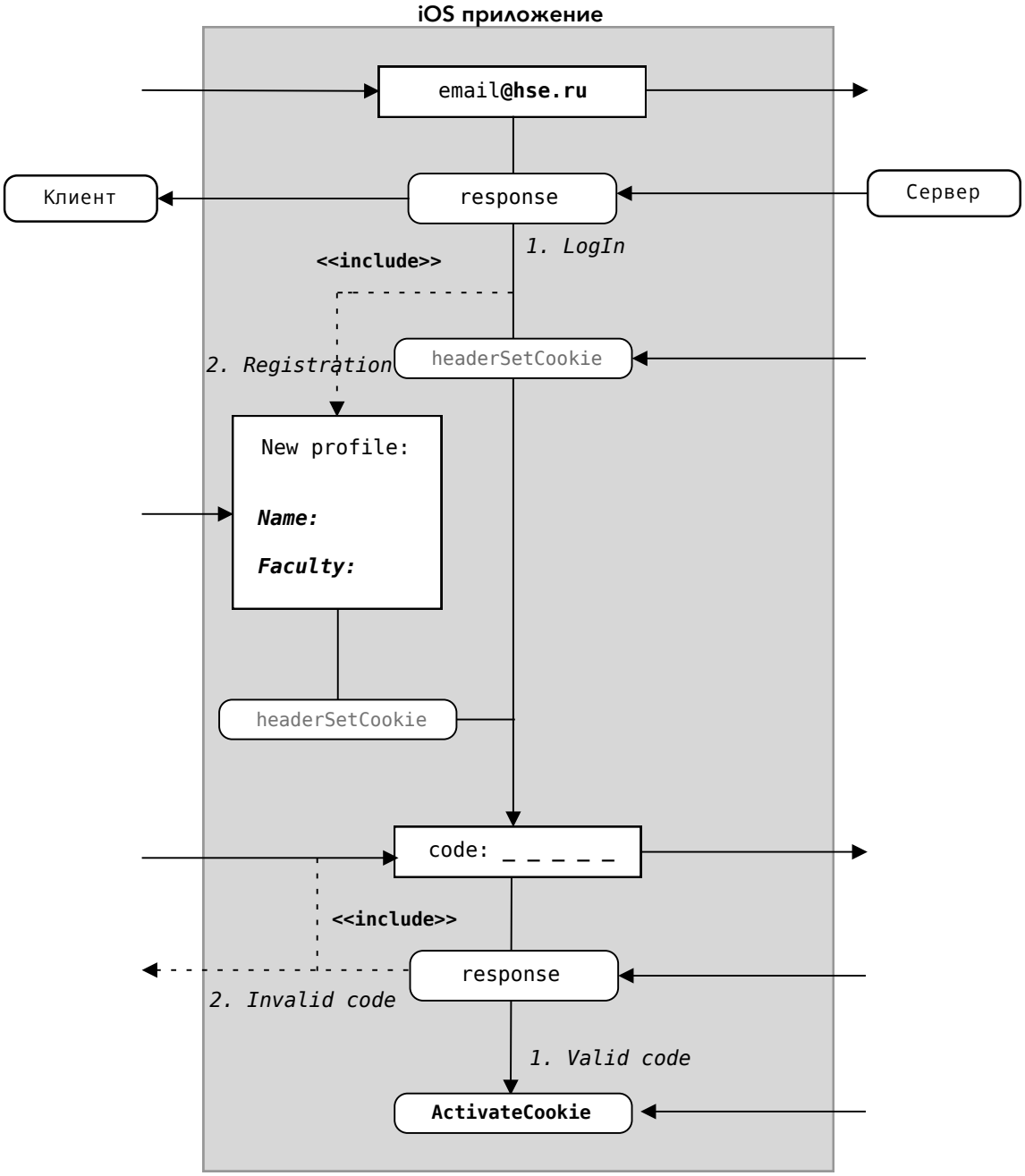

Рисунок 6 — Диаграмма прецедентов

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.03-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

3.2.3 Возможные взаимодействия программы с другими программами

Приложение «Социальная сеть для сотрудников НИУ ВШЭ» разработана с использованием сторонних библиотек:

- github «robb/Cartography»: расстановка ограничений верстки в коде
- github «ivanbruel/MarkdownKit»: отображение markdown текста в постах
- github «roberthein/TinyConstraints»: расстановка ограничений верстки в коде
- github «macteo/Marklight»: поддержка синтаксической подсветки markdown
- github «Alamofire/Alamofire» «5.0.0-beta.2»: для отправки запросов и получения ответов от сервера
- github «ZaidSA/TaggerKit» (forked from nekonora/TaggerKit): создание хэштегов, поиск и автодополнение
- github «ReactiveCocoa/ReactiveCocoa»: валидация почты

3.3 Описание и обоснование выбора метода организации входных и выходных данных

3.3.1 Описание метода организации входных и выходных данных

FR-7. **Входные данные**
Одна из основных задач приложения – создать площадку для коммуникации и обмена информацией между сотрудниками университета.

FR-7.1. При регистрации или входе в аккаунт в качестве входных данных программа будет принимать email только с доменом @hse.ru. (см 1a)

FR-7.2. При общении, создании постов у пользователей должна быть возможность обмениваться текстовой информацией.

FR-8. **Выходные данные**

FR-8.1. Выходные данные должны быть представлены посредством пользовательского интерфейса в качестве информации, полученной через указанные API методы от сервера.

FR-8.2. Все данные, собранные в процессе работы программы, загружаются на сервер во время работы программы и при ее завершении.

3.3.2 Обоснование выбора метода организации входных и выходных данных

Выбор входных и выходных данных обусловлен установленным функционалом программы.

3.4 Описание и обоснование выбора состава технических и программных средств

3.4.1 Состав технических и программных средств

При разработке программного продукта использовались следующие технические и программные средства:

- Язык разработки: Swift 5.0
- Среда разработки: Xcode Version 10.2.1
- Dependency manager: Carthage
- iPhone 7 версии 12.2
- Библиотеки, использованные при разрабоке: github «robb/Cartography», github «ReactiveCocoa/ReactiveCocoa», github «ZaidSA/TaggerKit» (forked from nekonora/TaggerKit), github «ivanbruel/MarkdownKit», github «roberthein/TinyConstraints», github «macteo/Marklight», github «Alamofire/Alamofire»

3.4.2 Обоснование выбора состава технических и программных средств

3.4.2.1 Язык программирования

Компания Apple поддерживает два языка программирования для iOS разработки: Objective-C и Swift.

Для разработки был выбран язык Swift 4.2, так как он более легкий в изучении, прост и приятен в синтаксисе, level of performance выше более чем в 1.5 раза чем у других языков, на которых можно разрабатывать под iOS платформу [7]. В процессе работы проект мигрировал на новую версию Swift 5.0 (первая ABI стабильная версия).

3.4.2.2 Среда разработки

Так как изначально было принято решение разрабатывать нативное приложение на языке программирования Swift, то такие среды разработки как Xamarin не рассматривались. Xcode – бесплатная (в отличие от AppCode) и удобная среда разработки нативных приложений на iOS, разработанная компанией Apple.

3.4.2.3 Шаблон проектирования

Был выбран шаблон проектирования MVC: он используется в iOS SDK и рекомендован производитем платформы. Другие основные паттерны, использованные при разработке: delegation, Coodinator pattern.

3.4.2.4 Библиотеки

Alomofire библиотека HTTP запросов для Swift. Эта библиотека используется для выполнения всех запросов на сервер [3] и получения данных. Удобное API для error handling, responce handling.

TinyConstraints/Cartography - обе библиотеки использовались для расставлений размеров и накладываний ограничений на внешний вид элементов `self.view` в коде. Xcode поддерживает `autolayout` с помощью технологии выставления constraints

в `storyboard`, но этот способ не подходит для сложных лэйаутов/анимаций. Обе библиотеки имеют документацию, содержащую информацию о том, как реализовывать расставление, анимацию constraints непосредственно в коде приложения. Несколько viewControllers были созданы полностью с помощью кода (не использовался `storyboard`), эти библиотеки помогли создать `layout` без xml поддержки. iOS SDK также поддерживает возможность расставлять ограничения в коде, но стандартная библиотека предоставляет неудобный API для этого. В частности Cartography позволяет объединять constraints в группы, эта возможность позволяет в динамике работать со взаимосвязанными объектами и их размерами. Библиотека TinyConstraints - синтаксический сахар, заменяющий длинные строчки кода из стандартной библиотеки на маленький и емкий кусок кода.

MarkdownLight/Marklight/TaggerKit - библиотеки использовались для парсинга маркдауна/подстветки синтаксиса и представления выбора хэштегов в приложении. Все три библиотеки не подразумевают сложного API, не требуют дополнительных знаний

4 Технико-экономические показатели

4.1 Предполагаемая потребность

Программа будет использоваться сотрудниками НИУ ВШЭ для коммуникации между научными сотрудниками, преподавателями, сотрудниками с разных факультетов.

Программа будет использоваться как средство общения и огранизации работы над совместными научными проектами между сотрудниками с одного или разных направлений/ факультетов/ кампусов.

Таким образом, программный продукт позволит решить проблему коммуникации между факультетами, кампусами НИУ ВШЭ и даст возможность наладить взаимодействие между преподавателями университета, будет способствовать развитию научной деятельности между разными факультетами и кампусами.

4.2 Экономические преимущества по сравнению с отечественными и зарубежными аналогами

В данной таблице приведен список прямых и косвенных конкурентов социальной сети.

Таблица 1 — Виды конкурентов

Название	**Описание**	**Вид**
Вк	Социальная сеть	Прямой
Твиттер	Социальная сеть	Прямой
Телеграмм	Социальная сеть	Прямой
Ватсап	Мессенджер	Прямой
Фейсбук	Социальная сеть	Прямой
Лмс	Система организации обучения	Прямой
Официальный сайт	Информационный портал	Прямой
Слак	Социальная сеть	Прямой
Вайбер	Мессенджер	Прямой
LinkedIn	Социальная сеть	Прямой
Одноклассники	Социальная сеть	Прямой
Мероприятия	Конференции	Косвенный
Instagram	Социальная сеть	Прямой
Почта	Способ рассылки сообщений	Косвенный
Brainly	Социальная сеть	Прямой
Skype	Социальная сеть	Прямой
ResearcherGate	Социальная сеть	Прямой
Mendeley	Социальная сеть	Прямой
ieee-collabaratec	Социальная сеть	Прямой
Authorea	Социальная сеть	Прямой
Edmodo	Социальная сеть	Прямой

Большинство аналогов не ориентированы на научную деятельность. Прямые конкуренты (например, LMS) не удовлетворяют всем перечисленным критериям (не имеют приложения на мобильные устройства, нет возможности предпросмотра кода, бесперебойность работы).

Таким образом, приложение «Социальная сеть для сотрудников НИУ ВШЭ» будет пользоваться спросом среди сотрудников НИУ ВШЭ из-за его узкой направленности (приложение могут использовать только сотрудники университета), ориентации на развитие научной деятельности, наличии приложения и бесперебойной работы социальной сети.

Ниже приведена таблица с детальным анализом конкурентов социальной сети. При подсчете финальной оценки кажого конкурента большим приоритетом в формуле обладали показатели критериев «Ориентированность на профессионаоьную и исследовательскую деятельность», «Бесперебойность работы» и «Категоризация информации». Именно эти критерии наиболее релевантны для будущих пользователей разрабатываемого приложения.

Для вычисления финальной оценки был применен следующий алгоритм: критерии оценки были разбиты на следующие кластеры исходя из смысла, для каждой категории был задан вес(от 0 до 10, где 10 – самое важное) исходя из общих рассуждений о важности:

A. Наличие бесед – 10

B. Поиск по категориям – 9

C. Категоризация информации – 9

D. Автоматический выбор релевантной информации – 9

E. Ориентированность на профессиональную деятельность – 8

F. Подгрузка информации из релевантных источников – 3

G. Ориентированность на мобильные устройства – 9

H. Наличие десктопного приложения – 9

I. Наличие web-версии – 9

J. Код – 8

K. Мультиплатформенность – 9

L. Мультимедиа – 6

M. Разделение сообщения/объявления/профиль – 9

N. Бесперебойность работы – 8

Итоговый коэффициент для каждой оценки рассчитывался, как

$$\frac{n}{\sum_{i=1}^{7} n} \tag{1}$$

Финальная оценка рассчитывалась по следующей формуле:

```
import statistics as st

def coeff(i):
    res = 10*(st.mediana([C[i],G[i],D[i],M[i]]))/10*C[i]
    + B[i]/10 + (0.65*E[i]+0.35*L[i])/10*C[i] +
    F[i]/10*C[i] + st.mediana([J[i],H[i]])/10*C[i]
    + K[i]/10*C[i] + N[i]/10*C[i]

    return res
```

Листинг 5 — Функция, рассчитывающая финальную оценку

Таблица 2 — Детальный анализ конкурентов

	Вк	**Твиттер**	**Телеграмм**	**Ватсап**	**Фейсбук**	**Лмс**	**Официальный сайт**	**Слак**	**Вайбер**	**Mendeley**	**Одноклассники**	**Мероприятия**	**Instagram**	**Почта**	**Brainly**	**Skype**	**ResearcherGate**	**ieee-collabratec**	**Authorea**	**Edmodo**
Наличие бесед	7	3	10	8	8	0	0	10	7	2	7	10	4	0	0	6	0	6	0	8
Поиск по категориям	7	8	4	2	7	0	4	9	0	5	4	3	7	5	9	0	9	8	8	8
Категоризация информации	4	5	2	2	4	7	7	8	0	6	4	8	6	5	8	0	9	7	8	7
Ориентированность на профессиональную исследовательскую деятельность	2	2	2	2	2	10	10	9	0	10	2	10	1	8	10	4	10	10	10	6
Подгрузка информации из других релевантных источников	3	3	3	0	5	0	0	10	0	8	0	0	0	0	0	0	0	5	5	8
Автоматический выбор релевантной информации	6	6	0	0	6	0	0	0	0	7	0	0	6	0	0	0	8	7	6	8
Ориентированность на мобильные устройства	9	7	9	9	8	0	5	8	6	4	5	0	8	9	7	7	4	4	5	10
Наличие десктопного приложения	6	0	8	3	0	0	0	8	3	8	0	0	2	9	0	7	0	0	0	0
Наличие web-версии	10	7	8	3	10	6	10	8	4	6	10	4	2	9	10	5	7	3	8	9
Мультимедиа	9	6	7	6	8	2	2	6	5	5	8	4	7	6	4	5	7	9	9	8
Код	0	0	4	0	0	0	0	9	0	0	0	0	0	0	3	0	9	10	9	6
Разделение сообщения/ объявления/ профиль	8	7	4	4	8	7	3	5	4	4	7	7	4	0	2	5	8	7	6	10
Бесперебойность работы	9	8	5	10	10	6	10	10	10	9	10	7	10	10	10	8	10	2	8	10
Финальная оценка	**6.7**	**5.1**	**5.9**	**4.7**	**6.9**	**2.7**	**4.2**	**8.5**	**4.1**	**5.7**	**5.5**	**5.2**	**4.5**	**4.9**	**5.1**	**4.5**	**5.9**	**5.9**	**6**	**8.2**

ПРИЛОЖЕНИЕ А

Список литературы

[1] The Swift Programming Language Documentation [Электронный ресурс] URL: `https://swift.org/documentation/#the-swift-programming-language` (Дата обращения: 16.05.2019, режим доступа: свободный)

[2] Единая система программной документации – М.: ИПК, Издательство стандартов, 2000, 125 стр.

[3] GitHub repository ilyakoo0/denis [Электронный ресурс] URL: `https://github.com/ilyakooo0/denis` (Дата обращения: 16.05.2019, режим доступа: свободный)

[4] GitHub repository Alomofire/Alomofire [Электронный ресурс] URL: `https://github.com/Alamofire/Alamofire` (Дата обращения: 16.05.2019, режим доступа: свободный)

[5] Swift Standard Library [Электронный ресурс] URL: `https://developer.apple.com/documentation/swift/swift_standard_library` (Дата обращения: 16.05.2019, режим доступа: свободный)

[6] LMS [Электронный ресурс] URL: `https://lms.hse.ru` (Дата обращения: 16.05.2019, режим доступа: свободный)

[7] Aticle «9 Reasons to Choose Swift for iOS App Development» [Электронный ресурс] URL: `https://www.upwork.com/hiring/for-clients/9-reasons-to-choose-swift-for-ios-app-development/` (Дата обращения: 16.05.2019, режим доступа: свободный)

ПРИЛОЖЕНИЕ В

Используемые понятия и определения

Социальная сеть – это интернет-площадка, сайт, который позволяет зарегистрированным на нем пользователям размещать информацию о себе и коммуницировать между собой, устанавливая социальные связи

Хэштег – ключевое слово или несколько слов сообщения, тег (пометка), используемый в микроблогах и социальных сетях, облегчающий поиск сообщений по теме или содержанию и начинающийся со знака решётки

Пост – информационный блок, размещённый пользователем в социальной сети на своей странице и содержащий набор хэштегов, по которым его можно найти

Канал – сохраненные ранее созданные фильтры новостей (набор хэштегов) по всем публичным постам

Беседа – чат для пользователей, в котором одновременно могут присутствовать от 3 до 50 участников

Проект – раздел, в котором сотрудники с любых факультетов по приглашению смогут вместе работать над каким-либо научным исследованием. Проект состоит из timeline (лента с новостями для всех участников проекта) и набором чатов и бесед (с разным количеством участников в каждой)

Preview канала – предпросмотр ленты канала с ограничениями на просмотр полной версии поста в ленте, переходом на страницы авторов, нажатия на хэштеги

Preview поста – ограниченный контент (содержание) поста в ленте

Model-View-Controller MVC схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо

Xcode интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple

Dependency manager программный модуль, который координируют интеграцию внешних библиотек или пакетов в стек приложения

Constraints ограничения на размеры и положения объектов на view, необходимые для правильного определения размеров и позиций контейнеров

Storyboard удобный механизм разработки интерфейса программы

ПРИЛОЖЕНИЕ С

Статус требований

Таблица 3 — Статус требований

Proposed	Требование запрошено авторизированным источником
Approved	Требование проанализировано, его влияние на проект просчитано, и оно было размещено в базовой версии определенной версии.
Implemented	Код, реализующий требование, разработан, написан и протестирован. Требование отслежено до соответствующих элементов дизайна и кода
Verified	Корректное функционирование реализованного требования подтверждено в соответствующем продукте. Требование отслежено до соответствующих вариантов тестирования. Теперь требование считается завершенным
Deleted	Утвержденное требование удалено из базовой версии.
Rejected	Требование предложено, но не запланировано для реализации ни в одной будущих версий.

ПРИЛОЖЕНИЕ D

Описание классов, структур, методов, полей

Описание классов и структур

Класс или структура	Описание
`class ChannelController: UIViewController, UITableViewDelegate, UITableViewDataSource, UITextFieldDelegate, UISearchBarDelegate`	Класс, координирующий перемещение в слое каналов приложения
`class TabBarCoordinator: BaseCoordinator`	Класс, отвечающий за координацию Main Flow программы, управляет ChannelsCoordinator, MessagesCoordinator, ProfileCoordinator
`class NewPostViewController: UIViewController, UITextViewDelegate, TagsReceiver`	Класс, отвечающий за создание и публикация нового поста
`class DialogCell: UITableViewCell`	Класс ячейки для представления списка сообщений в диалогах
`class DialogViewController: UIViewController, UpdatableGroup, UITableViewDelegate, UITableViewDataSource`	Класс, содержащий таблицу для отображения сообщений и поле для ввода сообщений
`class MessagesViewController: UITableViewController`	Класс, координирующий перемещение в слое сообщений приложения
`class TaggsSelectionViewController: UIViewController`	Класс, отвечающий за выбор хэштегов
`final class TabbarController: UITabBarController, UITabBarControllerDelegate, TabbarView`	Класс, отвечающий за перемещение между tabBar в приложении
`class ChannelsCoordinator: BaseCoordinator`	Класс, координирующий работу слоя каналов приложения
`class SimplifiedChannelsList: UITableViewController`	Класс, отвечающий за представление списка каналов в упрощенном виде
`class PeopleToWriteViewController: UITableViewController`	Класс, отвечающий за выбор людей
`class MyStackView: UIStackView`	Класс, содержащий переопределение методов для класса StackView
`struct CompletionTree: Codable`	Класс дерева хэштегов, отвечающий за представление подсказок хэштегов в приложении
`class AppDelegate: UIResponder, UIApplicationDelegate`	Главный класс приложения. Точка запуска

`class LogInCoordinator: BaseCoordinator`	Класс, отвечающий за логику регистрации и авторизации
`class ProfileViewController: UIViewController`	Класс, отвечающий за представление профилей пользователей в приложении
`class Model`	Класс, отвечающий за общение с сервером: отправку запросов, получения данных
`class ChannelViewController: UITableViewController, UpdatableName, UpdatableChannel, TagsReceiver`	Класс, отвечающий за представление канала в режиме редактирования и создания
`class InfoCell: UITableViewCell`	Класс, отвечающий за кастомизацию ячейки таблицы информации о пользователе на станице пользователя
`class DataStorage`	Класс, хранящий данные в хранилище приложения о пользователе (токен для запросов и id)
`class MessagesCoordinator: BaseCoordinator`	Класс, координирующий работу слоя приложений
`class BaseCoordinator: Coordinator`	Базовый класс координаторов
`class LogInViewController: UIViewController`	Класс, отвечающий за ввод почты пользователя
`class NewsController: UIViewController, UISearchControllerDelegate, UpdateableWithChannel, UISearchResultsUpdating`	Класс, отвечающий за отображения новостной ленты
`class ChatInfoViewController: UITableViewController`	Класс, отвечающий за отображение информации о беседе
`class ProfileCoordinator: BaseCoordinator`	Класс, координирующий работу слоя профиля приложения
`class PostViewCell: UITableViewCell`	Класс, отвечающий за представление ячейки таблицы поста в новостной ленте
`class BasicInfoController: UIViewController, UITableViewDelegate, UITableViewDataSource`	Класс, отвечающий за предоставление базовой информации о пользователе на его странице
`class ChannelListController: UITableViewController`	Класс, отвечающий за представлене списка каналов в профиле пользователя
`class NewsVC: UIViewController, UITableViewDelegate, UITableViewDataSource`	Класс, отвечающий за предоставление информации о новостной ленте на странице пользователя, в ленте, в ленте каналов
`class FullPostController: UITableViewController`	Класс, отвечающий за представление полного размера поста
`final class ApplicationCoordinator: BaseCoordinator`	Класс, координирующий работу приложения

Описание полей классов и структур

`class TabBarCoordinator: BaseCoordinator`

Поле	Описание
`var didEndFlow: (()->())?`	Лямбда, отвечающая за окончание работы слоя
`private let tabbarView: TabbarView`	Приватное поле, содержащее объект таббара

`class NewPostViewController: UIViewController, UITextViewDelegate, TagsReceiver`

Поле	Описание
`@IBOutlet weak var view1: UIView!`	Экран написания поста
`var currentTags: [String]`	Добавленные к посту тэги
`static var draft: String`	Черновик, сохраняющийся при выходе из режима написания поста
`var textView: UITextView!`	Поле для ввода текста поста
`var moveBackToParentVC: (()->())?`	Переход обратно в ленту

`class DialogCell: UITableViewCell`

Поле	Описание
`let nameLabel: UIButton`	Имя пользователя, написавшего сообщение
`var onUserDisplay: ((Int)->())?`	Лямбда-действие, происходящее при нажатии на имя пользователя
`let timeLabel: UILabel`	Время отправки оообщения
`var user: Model.Users?`	Объект пользователя, написавшего сообщение

`class DialogViewController: UIViewController, UpdatableGroup, UITableViewDelegate, UITableViewDataSource`

Поле	Описание
`var tableView: UITableView`	Таблица для отображения сообщений в диалоге
`let cellId`	reuseIdentifier ячейки
`var onInfoShow: (()->())?`	Лямбда - что делать при нажатии на кнопку «Info»
`var dialog: Model.Dialog?`	Диалог, отображенный в таблице сообщений
`var groupChat: Model.GroupChat?`	Групповой чат, отображенный в таблице сообщений
`var userChat: Model.UserChat?`	Чат, отображенный в таблице сообщений
`var users: [Int: Model.Users]?`	Словарь пользователей, участвующих в беседе

`var messageSendView: UIView`	Окно для отправки сообщения
`var messageTextView: UITextView`	Место для ввода сообщения
`var sendButton: UIButton`	Кнопка для отправки сообщения
`var bottomConstraint: NSLayoutConstraint!`	Constraint для окна для отправки сообщения
`var currentMessagesInChat: [Model.LastMessage]?`	Сообщения, отображенные в чате

`class MessagesViewController: UITableViewController`

Поле	**Описание**
`var currentActiveDialogs: [Model.Dialog]`	Диалоги, отображенные в списке диалогов
`var users: [Int: Model.Users]`	Пользователи, учствующие в диалогах
`var onUserDisplayList: (()->())?`	Лябда, определяющая что делать при нажати на кнопку «Write»
`var onDialogDisplay: (((dialog: Model.Dialog, users: [Int:Model.Users]))->())?`	Лямбда, определеяющая открытие окна отображения диалога
`let searchC`	searchController для поиска в списке диалогов

`class ChannelsCoordinator: BaseCoordinator`

Поле	**Описание**
`let storyboard`	Инстанс сториборда
`private weak var navigationController: UINavigationController?`	navigationController для этого слоя

`struct CompletionTree: Codable`

Поле	**Описание**
`var value: String?`	Либо принимает значение null в случае неполного слова, либо - слово-хэштег
`var subtree: [String : CompletionTree]`	Поддерево хэштегов

`class ProfileViewController: UIViewController`

Поле	**Описание**
`@IBOutlet weak var segmentedControl: UISegmentedControl!`	Определяющий контрол для отображения базовой информации о пользователе на старнице, либо его постов
`@IBOutlet weak var profileView: UIView!`	Окно для отображения пользователя

`@IBOutlet weak var tableView: UITableView!`	Таблица, отвечающая за отображение постов пользователя
`@IBOutlet weak var profileImageView: UIImageView!`	Фотография пользователя
`@IBOutlet weak var surnameLabel: UILabel!`	Лэйбл для фамилии пользователя
`@IBOutlet weak var nameLabel: UILabel!`	Лэйбл для имени пользователя
`@IBOutlet weak var facultyLabel: UILabel!`	Лэйбл для факультета пользователя
`@IBOutlet weak var placeLabel: UILabel!`	Лэйбл для места работы пользователя
`@IBOutlet weak var newMessageButton: UIButton!`	Кнопка для написания сообщения пользователю

`class Model`

Поле	**Описание**
`static let invalidTocken`	Константа статус кода, означающего invalidTocken
`static var hashTagTree: CompletionTree?`	Экземпляр дерева хэштегов
`private static var isValidTocken: ((Int)->())?`	Проверка на валидность токена с помощью переданного статус кода запроса
`private static let baseUrl`	Базовый URL сервера
`static let authenticationURL`	URL для аутентификации в социальной сети
`static let postsLastURL`	URL для запроса постов в общей ленте
`static let postsForUserURL`	URL для запроса постов на странице пользователя
`static let postsPublishURL`	URL для опубликования нового поста
`static let usersURL`	URL для получения объекта пользователя по его id
`static let usersAllURL`	URL для получения всех объектов пользователей соцаильной сети
`static let channelsGetURL`	URL для получения канала
`static let channelsUpdateURL`	URL для обновления объекта канала
`static let channelsListURL`	URL для получения списка всех каналов
`static let channelsCreateURL`	URL для создания канала
`static let channelsDeleteURL`	URL для удаления канала
`static let channelsGetAnonURL`	URL для запроса анонимного канала по структуре канала без id
`static let complexURL`	URL для сложных запросов
`static let hashTagTreeURL`	URL для запроса дерева хэштегов
`static let createGroupChatURL`	URL для создания беседы

`static let chatsGetAllURL`	URL для получения списка всех чатов
`static let getGroupChatURL`	URL для получения объекта чата
`static let leaveGroupChatURL`	URL для покидания группового чата
`static let updateGroupChatURL`	URL для обновления группового чата
`static let messagesGetGroupChatURL`	URL для запроса сообщений в беседе
`static let messagesSendURL`	URL для отправки сообщения
`static let messagesGetUserChatURL`	URL для запроса сообщений в чате с пользователем

`class ChannelViewController: UITableViewController, UpdatableName, UpdatableChannel, TagsReceiver`

Поле	**Описание**
`var onChoosingHashTags: (([String]?)->())?`	Действие при нажатии на ячейку для выбора людей
`var onChoosingPeople: ((Model.Channels?)->())?`	Действие при нажатии на ячейку для выбора хэштегов
`var channel: Model.Channels?`	Отображенный канал
`var onShowingPreview: ((Model.Channels)->())?`	Действие при нажатии на кнопку отображения предпросмотра

`class LogInViewController: UIViewController`

Поле	**Описание**
`var authenticate: ((Int)->())?`	Лямбда для входа в приложение
`let logInLabel: UILabel`	Лэйбл Log In
`let mailTextField: UITextField`	textField для почты пользователя
`let logInButton: UIButton`	Кнопка для входа в приложение
`private lazy var keyboardBar`	Панель для кнопки
`private lazy var contentView`	Содержимое для поля для почты и лэйбла
`private var bottomConstraint: NSLayoutConstraint?`	Нижний constraint панели для кнопки

`class NewsController: UIViewController, UISearchControllerDelegate, UpdateableWithChannel, UISearchResultsUpdating`

Поле	**Описание**
`var changedChannelName: ((String)->())?`	Действие при нажатии на ячейку для изменения имени
`@IBOutlet weak var tableView: UITableView!`	Таблица для отображения постов
`var channel: Model.Channels?`	Отображенный канал

`var anonymousChannel: (users: [Int: Model.Users], posts: [Model.Posts])?`	Анонимный канал
`var onSelectChannel: (() -> Void)?`	Действие при выборе другого канала
`var searchController: UISearchController?`	Поисковая строка
`var news: [Model.Posts]`	Массив новостей
`var type: HeaderType?`	Тип новостной ленты

`class ChatInfoViewController: UITableViewController`

Поле	**Описание**
`weak var delegate: UpdatableGroup?`	Объект-делегат для обновления информации о группе при успешном изменении данных
`var groupChat: Model.Group?`	Объект группы, информация о котором отображается в таблице
`var usersArray: [Int]`	Список людей в этой группе
`var users: [Int: Model.Users]`	Словарь людей в этой группе
`var myPermissions: PersonStatus`	Статус пользователя в беседе

`class PostViewCell: UITableViewCell`

Поле	**Описание**
`var onUserDisplay: ((Int)->())?`	Действие при нажатии на ячейку автора поста
`var onAnonymousChannelDisplay: ((String)->())?`	Действие при нажатии на хэштег
`let nameLabel: UIButton`	Имя автора
`let fullNameLabel: UILabel`	Полное имя автора
`let timeLabel: UILabel`	Время написания поста
`var post: Model.Posts?`	Пост, отображенный в ячейке
`var hashtags`	Хэштеги поста

`class NewsVC: UIViewController, UITableViewDelegate, UITableViewDataSource`

Поле	**Описание**
`var onChannelDidChange: ((([Int: Model.Users],[Model.Posts]))->())?`	Действие, происходящее с изменением выбора канала
`var onFullPost: ((HeaderType, Model.Posts)->())?`	Действие, происходящее с нажатием на просмотр полного содержания поста
`var dictionary: [Int: Model.Users]`	Люди, посты которых отображены в ленте
`var currChannel : Model.Channels?`	Отображенный канал
`var dataSourse: [Model.Posts]`	Данные о постах
`var cellDataSourse: [PostCellData]`	Данные о ячейках для отображения постов

`var type: HeaderType`	Тип новостной ленты
`weak var viewController: UIViewController?`	Класс, который делегирует отображение ленты

`class FullPostController: UITableViewController`

Поле	**Описание**
`var post: Model.Posts?`	Пост, отображенный в ячейке
`var type: HeaderType`	Тип поста

Описание методов классов и структур

`class ChannelController: UIViewController, UITableViewDelegate, UITableViewDataSource, UITextFieldDelegate, UISearchBarDelegate`

Метод	**Описание**
`override func viewDidLoad()`	Загрузка вьюшки
`override func viewWillAppear(_ animated: Bool)`	Метод, вызывающийся при появлении вьюшки
`override func viewWillDisappear(_ animated: Bool)`	Метод, вызывающийся при исчезновении вьюшки
`@objc func showPreview()`	Предпросмотр превью канала
`func setUpController()`	Настройка поисковой строки
`func numberOfSections(in tableView: UITableView) -> Int`	Количество секций в таблице
`func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int`	Количество ячеек в секции
`func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)`	Действие при выборе ячейки
`func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell`	Метод, возвращающий ячеейку
`func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat`	Высота заголовка секции

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					