

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Мобильное приложение поддержки взаимодействия

членов творческого коллектива

Студент

(подпись, дата)

Бардин Владислав Алексеевич

(ФИО полностью)

Руководитель

(подпись, дата)

Макарычев П.П.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Макарычев П.П.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГАК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2020

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Заведующий кафедрой

_____ П.П. Макарычев

«___» _____ 2020г.

ЗАДАНИЕ
на выпускную работу бакалавра

1. Студент гр. 16ВП1 факультета ВТ направления 09.03.04
Бардин Владислав Алексеевич
2. Руководитель работы Макарычев П.П.
3. Время проектирования с «08» 04 2020 по «28» 05 2020
4. Тема проекта: Мобильное приложение поддержки взаимодействия членов творческого коллектива
Тема утверждена приказом ПГУ № _____ от «___» _____ 2020г.
5. Техническое задание на квалификационную работу (назначение, технические требования)
1) Разработать приложение для мобильных устройств под управлением операционной системы iOS для предоставления членам творческого коллектива возможность обмениваться сообщениями, вести общий календарь событий и организовать структурированное файловое Хранилище
2) Среда выполнения: операционная система iOS 13.0 и выше Языковые средства разработки: Swift, UML Инструментальные средства разработки: Xcode, iOS Simulator, Microsoft Project, Realm Studio Технология разработки: RUP, ООП
3) Основные функции мобильного приложения: а) обмен сообщениями между пользователями; б) обмен аудиосообщениями между пользователями; в) организация файлового хранилища; г) добавление файлов в хранилище; д) просмотр календаря событий; е) добавление событий в календарь;

6. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области
 - 2) Анализ функциональных требований
 - 3) Проектирование
 - 4) Реализация
 - 5) Контроль качества ПО:
 - Тестирование
 - Анализ качества кода
 - 6) План разработки и оценка бюджета
-
-
-

7. Календарный график по выполнению работы

Наименование этапов работы	Объем работы (%)	Срок выполнения	Подпись руководителя, консультанта
1) Анализ предметной области и требований	10	8.04.2020 – 13.04.2020	
2) Проектирование	25	13.04.2020 – 25.04.2020	
3) Реализация	30	25.04.2020 – 10.05.2020	
4) Контроль качества	20	10.05.2020 – 20.05.2020	
5) План разработки, бюджет	5	20.05.2020 – 23.05.2020	
6) Оформление пояснительной записки	10	23.05.2020 – 28.05.2020	

Дата выдачи задания «25» _____ декабря _____ 2019г.

Руководитель бакалаврской работы _____

Задание к исполнению принял «__» _____ 2020г.

Студент _____

Реферат

Пояснительная записка содержит 118 листов, 51 рисунок, 25 использованных источников и 20 таблиц.

ООП, SWIFT, XCODE, UML, REALM, FIREBASE, MVC, iOS.

Объектом разработки – программные средства поддержки взаимодействия членов коллектива.

Целью выпускной квалификационной работы является разработка мобильного приложения на платформе iOS, предоставляющего удобный и быстрый способ обмена различной информацией между участниками.

Технология разработки – ООП, MVC.

В результате проведенной работы создано мобильное приложение, позволяющее пользователям обмениваться сообщениями и аудиосообщениями в чате, просматривать и организовывать общее хранилище различных файлов, а также вести календарь событий и мероприятий.

Разработка программного обеспечения проводилась на языке программирования Swift 5 в среде Xcode в операционной системе macOS Catalina. В качестве основного хранилища данных использована объектная база данных realm, а в качестве дополнительного хранилища для различных файлов использовано облачное хранилище Firebase.

					ПГУ 09.03.04-08БР151.04 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Бардин В.А.			«Мобильное приложение поддержки взаимодействия членов творческого коллектива» Пояснительная записка	Лит.	Лист	Листов
Провер.		Макарычев П.П.					3	118
						Гр. 16ВП1		
Н. Контр.		Такташкин Д.В.						
Утв.								

Содержание

Введение.....	5
1 Постановка задачи и анализ требований	6
1.1 Постановка задачи	6
1.2 Анализ предметной области.....	7
1.3 Анализ программного обеспечения аналогичного назначения	9
1.4 Анализ функциональных требований	15
1.5 Анализ нефункциональных требований	19
2 Проектирование мобильного приложения	20
2.1 Выбор технологий и инструментальных средств разработки	20
2.2 Планирование разработки и оценка бюджета	22
2.3 Проектирование пользовательского интерфейса	27
2.4 Проектирование архитектуры приложения	32
2.4.1 Проектирование классов модели.....	33
2.4.2 Проектирование классов представлений.....	34
2.4.3 Проектирование классов контроллеров.....	35
3 Реализация мобильного приложения	39
3.1 Реализация структур данных.....	39
3.2 Реализация пользовательского интерфейса	39
3.3 Развертывание приложения.....	44
3.4 Сравнение разработанного приложения с прототипом.....	46
4 Тестирование мобильного приложения.....	48
4.1 Выбор режима тестирования приложения.....	48
4.2 Функциональное тестирование	49
4.3 Структурное тестирование	50
4.4 Автоматизированное тестирование	54
4.4.1 Unit-тестирование	55
4.4.2 UI-тестирование	56
4.5 Метрики кода	59
Заключение	62
Список использованных источников	63
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ	67
ПРИЛОЖЕНИЕ Б РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ.....	112

Введение

В ходе совместной учёбы, работы или любой другой групповой деятельности у участников возникает необходимость обмениваться различной информацией, такой как текстовые сообщения, время посещения мероприятий, документы, файлы. Например, в ходе обучения в университете студентам группы необходимо распространять информацию о времени посещения занятий, мероприятий, связанных с учебной и внеучебной деятельностью, отправлять документы и различные файлы с лабораторными заданиями, вопросами к зачётам и экзаменам и т.п. Для данных целей студенты часто используют социальную сеть «ВКонтакте» или приложение «Telegram». Однако данные варианты больше сфокусированы на обмене текстовой информацией, при обмене файлами не поддерживается структуризация хранилища, а ведение какого-либо расписания возможно только с помощью стороннего или дополнительного программного обеспечения, что приводит к использованию нескольких различных приложений для организации полноценного взаимодействия членов коллектива, таких как «Google Календарь» и «Google Диск».

Таким образом, целесообразна разработка программного обеспечения, предоставляющего функции обмена сообщениями, создания файлового хранилища с возможностью структурирования по каталогам, ведения общего календаря для отслеживания важных событий и мероприятий в одном приложении.

Целью бакалаврской работы является разработка мобильного приложения, реализующего указанные ранее функции и упрощающего взаимодействие группы людей путем объединения ключевых функций различных приложений, необходимых для организации удобного обмена информацией.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Разрабатываемое мобильное приложение предназначено для удобного обмена информацией между группой людей. Главной задачей приложения является объединение в одном месте таких функций как обмен сообщениями в общем чате пользователей, добавление файлов в хранилище с возможностью просмотра содержимого, структурирование файлов в хранилище по каталогам, ведение календаря событий.

Приложение должно функционировать на устройствах, работающих на операционной системе iOS 13.0 и выше.

Обмен информацией между устройствами происходит путем передачи данных через Интернет, однако приложение должно обеспечивать доступ к уже скаченным сообщениям, файлам и событиям календаря.

В приложении должно быть предусмотрена безопасная регистрация и авторизация пользователей. Пользователи в приложении должны быть разделены на 2 группы:

- обычный пользователь;
- администратор.

Только администратор должен иметь возможность управлять файловым хранилищем и календарем событий, а также выделять любые сообщения как избранные.

Приложение должно обладать удобным интерфейсом для отправки сообщений. Наряду с обычными сообщениями, приложение должно обеспечивать возможность отправки аудиосообщения, а также преобразование данного аудиосообщения в текст, если такое преобразование возможно.

При добавлении события в календарь должна быть возможность задать циклическое повторение события для ведения расписания занятий в школе или университете.

Приложение должно предоставлять возможность загружать фотографию пользователя для отображения в чате и личном кабинете.

1.2 Анализ предметной области

Исходя из поставленных задач можно построить концептуальную модель данных предметной области (рисунок 1), содержащую следующие классы, имеющие наибольшее значение для решения поставленной задачи:

- пользователь;
- сообщение;
- каталог;
- событие;
- аудиосообщение;
- файл.

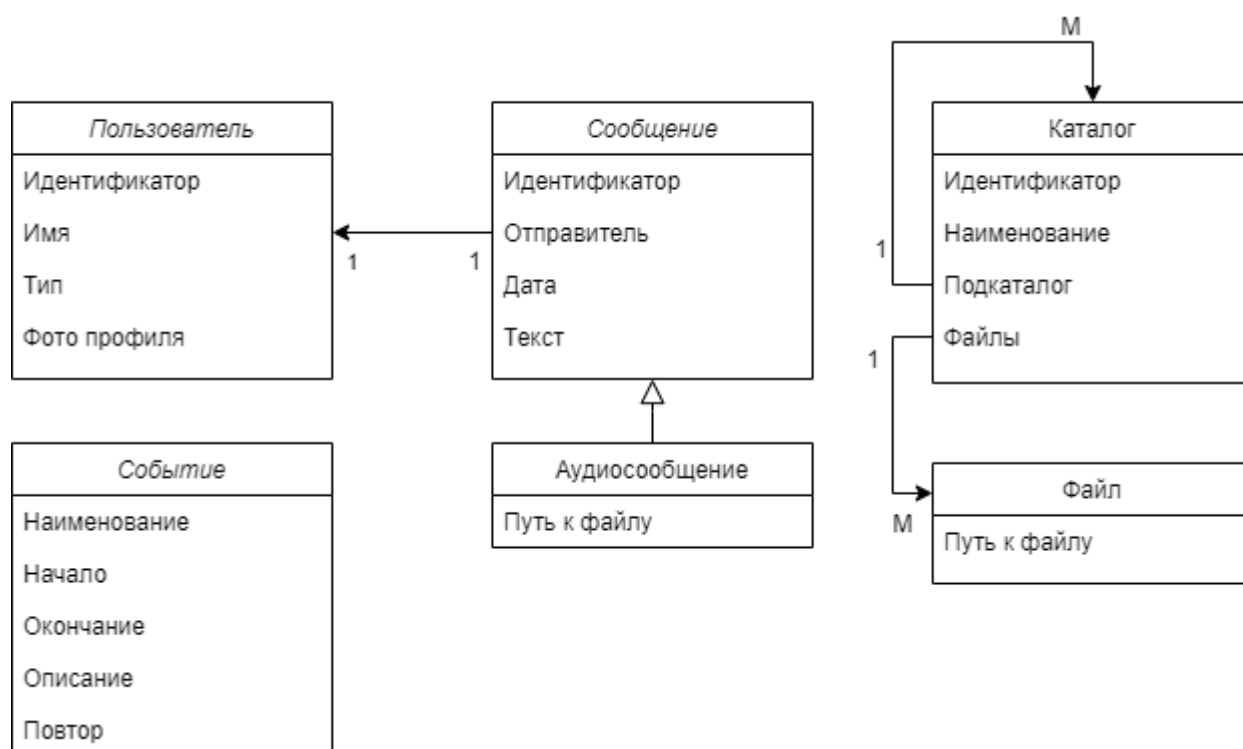


Рисунок 1 – Концептуальная модель

Описание классов концептуальной модели, а также их свойств приведено в таблице 1.

Таблица 1 – Описание концептуальной модели

Наименование класса	Назначение	Свойства
Сообщение	Класс предназначенный для хранения информации об отправляемом пользователем сообщении	Идентификатор – уникальный идентификатор сообщения; Отправитель – пользователь, отправивший сообщение; Дата – дата отправки сообщения; Текст – текст отправленного сообщения.
Аудиосообщение	Для хранения информации об аудиосообщении	Путь к файлу – путь, по которому хранится аудиофайл;
Пользователь	Класс для хранения основной информации о пользователе	Идентификатор – уникальный идентификатор пользователя (логин); Имя – отображаемое имя пользователя (в чате); Тип – тип пользователя (обычный/администратор); Фото пользователя – путь к фото пользователя для отображения в чате и личном кабинете.
Событие	Класс, предназначенный для хранения информации о предстоящих событиях и мероприятиях, добавляемых в календарь	Начало – начальная дата события; Окончание – конечная дата события; Наименование – название события; Описание – текстовое описание события; Повтор – определяет цикл повторения события (день, неделя, две недели, месяц, год).

Продолжение таблицы 1

Наименование класса	Назначение	Свойства
Категория	Класс, хранящий информацию о каталогах, добавляемых в файловое хранилище	Идентификатор – уникальный идентификатор каталога; Наименование – название каталога; Подкаталог – список подкаталогов; Файлы – список файлов, относящихся к каталогу.
Файлы	Класс, предназначенный для хранения пути к файлу	Путь к файлу – путь для скачивания в облачном хранилище

Поскольку класс аудиосообщение наследует все свойства класса сообщение, то текст аудиосообщения будет записываться в поле «Текст».

1.3 Анализ программного обеспечения аналогичного назначения

Проведем сравнение программ аналогичного назначения методом анализа иерархий.

Исходя из поставленной задачи выделим следующие критерии для сравнения программ-аналогов:

- организация работы с файловым хранилищем;
- функциональность чата пользователей;
- организация календаря событий;
- UX/UI-дизайн [1].

Существует множество различных приложений, которые можно сравнивать по данным критериям, однако многие из них имеют одинаковый функционал, поскольку относятся к одним и тем же группам ПО. Выделим три группы программных продуктов – социальные сети, мессенджеры и корпоративные мессенджеры. Нет смысла выбирать более одного продукта из данных групп, поскольку, например, мессенджеры «WhatsApp» и «Telegram»

по сравниваемым критериям будут аналогичны, однако сильно различаться с программными продуктами из группы социальных сетей.

Таким образом, сравним наиболее известные программные продукты для каждой из групп («Вконтакте», «Telegram», «Slack») по выделенным критериям с помощью метода анализа иерархий [2].

Построим матрицы попарных сравнений по качественной шкале, с последующим преобразованием в баллы. Обозначения баллов:

- равно, безразлично (1 балл);
- немного лучше (3 балла);
- лучше (5 баллов);
- значительно лучше (7 баллов);
- принципиально лучше (9 баллов).

Также возможны промежуточные баллы – 2, 4, 6 и 8.

Составляем матрицу a_{ij} – отношения критерия i к критерию j (таблица 2).

Таблица 2 – Матрица попарных сравнений критериев

	Организация работы с файловым хранилищем	Функциональность чата пользователей	Организация календаря событий	UX/UI-дизайн
Организация работы с файловым хранилищем	1	4	6	3
Функциональность чата пользователей	1/7	1	7	5
Организация календаря событий	1/6	1/7	1	1/7
UX/UI-дизайн	1/3	1/5	7	1

Составляем аналогичные матрицы сравнения программ-аналогов по каждому критерию. Матрицы по критериям организации работы с файлами и функциональности чата представлены на таблицах 3 и 4.

Таблица 3 – Матрица сравнения альтернатив по критерию организации работы с файловым хранилищем

Организация работы с файловым хранилищем	«Вконтакте»	«Telegram»	«Slack»
«Вконтакте»	1	5	7
«Telegram»	1/5	1	5
«Slack»	1/7	1/5	1

Таблица 4 – Матрица сравнения альтернатив по критерию функциональности чата пользователей

Функциональность чата пользователей	«Вконтакте»	«Telegram»	«Slack»
«Вконтакте»	1	3	7
«Telegram»	1/3	1	7
«Slack»	1/7	1/7	1

Матрицы по критериям организации календаря событий и UX/UI-дизайн представлены на таблицах 5 и 6.

Таблица 5 – Матрица сравнения альтернатив по критерию организации календаря событий

Организация календаря событий	«Вконтакте»	«Telegram»	«Slack»
«Вконтакте»	1	2	1/6
«Telegram»	1/2	1	1/6
«Slack»	6	6	1

Таблица 6 – Матрица сравнения альтернатив по критерию UX/UI-дизайн

UX/UI-дизайн	«Вконтакте»	«Telegram»	«Slack»
«Вконтакте»	1	1/4	7
«Telegram»	4	1	9
«Slack»	1/7	1/9	1

Проведем нормировку матрицы попарных сравнений критериев, вычислив суммарное значение столбцов и разделив баллы критериев на полученное значение, затем рассчитаем весовой столбец, путем вычисления среднего значения полученных нормированных показателей. Результаты расчетов представлены в таблице 7.

Таблица 7 – Весовые доли критериев

	Вес в долях	Вес в процентах
Организация работы с файловым хранилищем	0,493	49,3%
Функциональность чата пользователей	0,289	28,9%
Организация календаря событий	0,048	4,8%
UX/UI-дизайн	0,17	17%

Исходя из полученных данных, наиболее весомыми критериями являются организация работы с файловым хранилищем и функциональность чата пользователей. Менее важным является критерий UX/UI-дизайна, а наименее важным критерий организации календаря событий. Аналогичным образом получим векторы весовых коэффициентов программ-аналогов с точки зрения их соответствия отдельным критериям. Для критериев организации работы с файлами и функциональности чата векторы коэффициентов приведены в таблицах 8 – 9.

Таблица 8 – Весовые доли программ-аналогов по критерию организации файлового хранилища

Организация работы с файловым хранилищем	Вес в долях	Вес в процентах
«Вконтакте»	0,696	69,6%
«Telegram»	0,232	23,2%
«Slack»	0,072	7,2%

Таблица 9 – Весовые доли программ-аналогов по критерию функциональности чата пользователей

Функциональность чата пользователей	Вес в долях	Вес в процентах
«Вконтакте»	0,623	62,3%
«Telegram»	0,311	31,1%
«Slack»	0,066	6,6%

Для критериев организации календаря событий и UX/UI-дизайн векторы весовых коэффициентов приведены в таблицах 10 – 11.

Таблица 10 – Весовые доли программ-аналогов по критерию организации календаря событий

Организация календаря событий	Вес в долях	Вес в процентах
«Вконтакте»	0,16	16%
«Telegram»	0,1	10%
«Slack»	0,74	74%

Таблица 11 – Весовые доли программ-аналогов по критерию UX/UI-дизайна

UX/UI-дизайн	Вес в долях	Вес в процентах
«Вконтакте»	0,263	26,3%
«Telegram»	0,681	68,1%
«Slack»	0,056	5,6%

Сведем полученные результаты в одну таблицу для удобства восприятия (таблица 12).

Таблица 12 – Матрица весов альтернатив по каждому критерию

	Организация работы с файловым хранилищем	Функциональность чата пользователей	Организация календаря событий	UX/UI- дизайн
«Вконтакте»	0,696	0,623	0,16	0,263
«Telegram»	0,232	0,311	0,1	0,681
«Slack»	0,072	0,066	0,74	0,056

Чтобы получить лучший вариант по всем критериям умножим матрицу из таблицы 12 на матрицу из таблицы 7. Результат умножения приведен в таблице 13.

Таблица 13 – Веса альтернативных приложений

	Вес в долях	Вес в %
«Вконтакте»	0,58	58%
«Telegram»	0,32	32%
«Slack»	0,1	10%

Результат сравнения показывает, что лучшим приложением по заданным критериям является социальная сеть «Вконтакте» с весовой долей в 58 процентов. Также данная сеть является самой распространённой и имеет наибольшую пользовательскую базу в России [3, 4], что делает этот вариант наиболее оптимальным для выбора в качестве прототипа.

1.4 Анализ функциональных требований

В соответствии с поставленной задачей и анализом аналогичного программного обеспечения были выделены следующие функции, которые должны быть реализованы в мобильном приложении:

- регистрация пользователя;
- отправка сообщения/аудиосообщения;
- перевод аудиосообщения в текст;
- добавление документов в файловое хранилище;
- структурирование файлового хранилища по каталогам;
- просмотр календаря событий по датам;
- добавление событий в календарь и возможность добавления события с заданным циклом (день/неделя/2 недели/месяц/год);
- добавление фотографии пользователя для его удобной идентификации другими пользователями.

На рисунке 2 представлена диаграмма вариантов использования, полученная в соответствии с предъявленными требованиями.

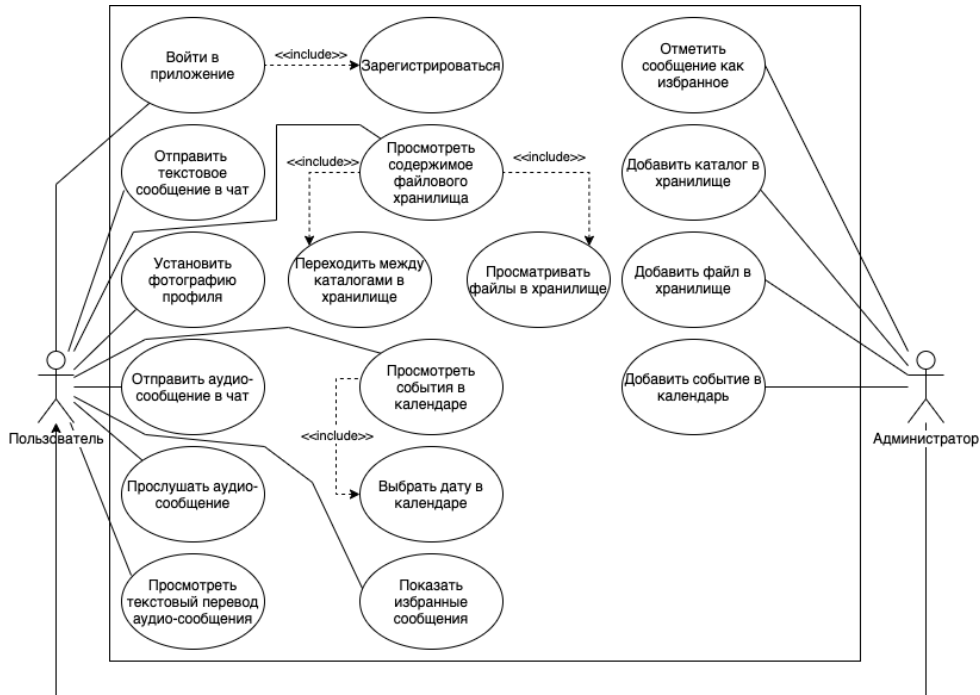


Рисунок 2 – Диаграмма вариантов использования

На диаграмме представлены два вида участников системы – пользователь и администратор, который также обладает функциями пользователя. К пользователю относятся прецеденты основной функциональности системы, такие как отправка аудио/текстовых сообщений, просмотр файлового хранилища и календаря и другие. Роль администратора системы заключается в заполнении файлового хранилища и календаря, а также отметка сообщений как избранных.

Опишем процесс обработки данных подробнее и приведём спецификацию для вариантов использования «Отправить аудиосообщение в чат», «Добавить файл в хранилище» и «Добавить событие в календарь», которые затрагивают главные функции приложения (рисунки 3 – 5).

Прецедент: Отправить аудиосообщение в чат
ID: 1
Краткое описание: Пользователь отправляет аудиосообщение
Главные актёры: Пользователь
Второстепенные актёры: Нет
Предусловия: <ol style="list-style-type: none"> 1. Пользователь авторизовался в приложении 2. Пользователь разрешил системе использовать микрофон
Основной поток: <ol style="list-style-type: none"> 1. Пользователь нажимает кнопку записи аудиосообщения 2. Система записывает сообщение пользователя 3. Пользователь нажимает кнопку закончить запись 4. Система отправляет информацию об аудиосообщении в БД 5. Система отправляет аудиофайл в облачное хранилище Firebase 6. Система обновляет чат пользователей, добавляя новое сообщение 7. Система отправляет аудиофайл в систему распознавания речи 8. Если удалось получить текст сообщения <ol style="list-style-type: none"> 8.1 Система добавляет к аудиосообщению текст в БД 8.2 Система обновляет интерфейс для отображения текста аудиосообщения
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 3 – Спецификация варианта использования «Отправить аудиосообщение в чат»

Прецедент: Добавить файл в хранилище
ID: 2
Краткое описание: Администратор системы добавляет файл в хранилище
Главные актёры: Администратор
Второстепенные актеры: Нет
Предусловия: <ol style="list-style-type: none"> 1. Администратор авторизовался в приложении 2. Администратор выбрал необходимый каталог в хранилище
Основной поток: <ol style="list-style-type: none"> 1. Администратор нажал кнопку добавления файла 2. Администратор выбирает файл на своем устройстве для добавления в систему 3. Система добавляет выбранный файл в облачное хранилище Firebase 4. Система добавляет информацию о файле в БД 5. Система обновляет интерфейс пользователя для отображения нового файла в каталоге
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 4 – Спецификация варианта использования «Добавить файл в хранилище»

Прецедент: Добавить событие в календарь
ID: 3
Краткое описание: Администратор добавляет событие в календарь
Главные актёры: Администратор
Второстепенные актеры: Нет
Предусловия: Администратор авторизовался в приложении
Основной поток: <ol style="list-style-type: none"> 1. Администратор нажимает кнопку добавления события 2. Система отображает форму добавления события 3. Администратор заполняет необходимые поля описания события 4. Администратор добавляет событие 5. Система добавляет данные события в БД 6. Система возвращает пользователя в календарь 7. Система обновляет интерфейс пользователя
Постусловия: Нет
Альтернативные потоки: Нет

Рисунок 5 – Спецификация варианта использования «Добавить событие в календарь»

Для уточнения взаимодействия между объектами применим диаграмму деятельности (рисунок 6) для прецедента «Отправить аудиосообщение в чат». Данный вариант затрагивает наибольшее количество объектов и связей между ними.

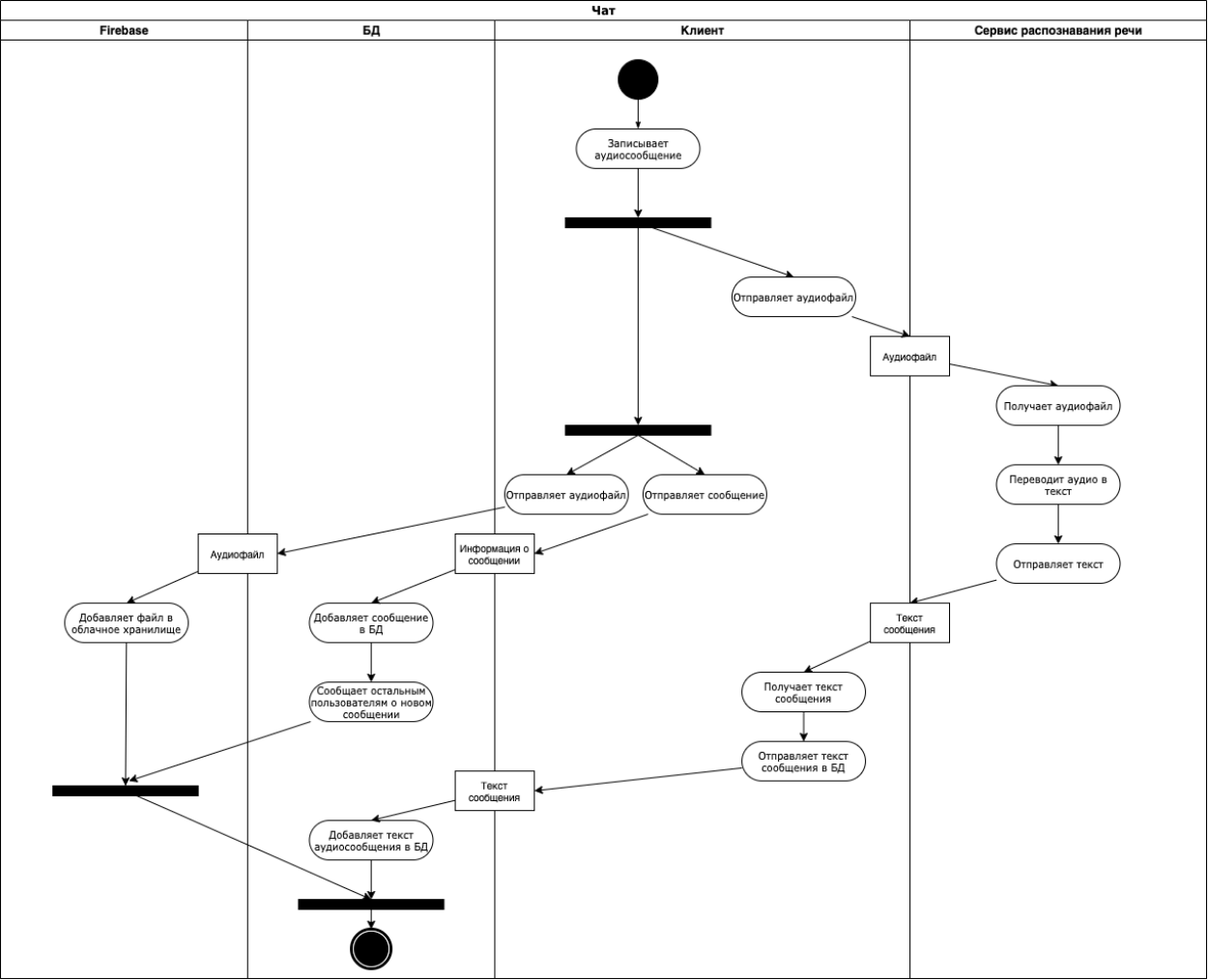


Рисунок 6 – Диаграмма деятельности для варианта использования «Отправить аудиосообщение в чат»

Исходя из представленной диаграммы деятельности можно добавить трех участников системы: облачное хранилище Firebase, БД и сервис распознавания речи. Можно заметить, что от клиентского приложения требуется параллельная обработка данных. Также вытекают требования к базе данных, которая должна уведомлять всех пользователей системы о добавлении новых данных.

Логично, что диаграммы деятельности для других основных вариантов использования системы «Добавить событие в календарь», «Добавить файл в

хранилище» не сильно отличаются от представленной на рисунке 6. Оба прецедента не используют сервис распознавания речи, но похожим образом используют БД и Firebase (в случае добавления файла).

1.5 Анализ нефункциональных требований

К разрабатываемому приложению можно предъявить следующие нефункциональные требования:

- клиентское приложение должно иметь удобный и современный интерфейс чата пользователей;
- должен быть интуитивно понятный интерфейс календаря для удобного отслеживания текущих и будущих событий;
- необходимо, чтобы приложение имело защищенный механизм регистрации и авторизации;
- приложение должно поддерживаться на мобильных устройствах с версией операционной системы iOS 13.0 и выше;
- приложение должно быть адаптировано под все размеры мобильных дисплеев;
- должно интуитивно понятно отображаться состояние файлов/аудиосообщений как скаченных/доступных для скачивания.

2 Проектирование мобильного приложения

2.1 Выбор технологий и инструментальных средств разработки

При реализации приложения использовалась методология разработки Rational Unified Process (RUP). Поскольку данная методология основывается на итеративном подходе, она позволяет:

- учитывать изменяющиеся требования к проекту;
- интегрировать отдельные функции постепенно, позволяя им проходить цикл разработки, проверки и внедрения в проект обособлено, что снижает риски и стоимость производства;
- при наращивании функциональности повторно использовать выделенные типовые решения, что сокращает разработку;
- позволяет сделать ранний выпуск продукта с уменьшенной функциональностью, для занятия ниши на рынке.

Приложение разрабатывается для устройств с операционной системой iOS 13.0 и выше, поэтому выбор языка программирования (ЯП) осуществляется между Objective-C и Swift. Для данного проекта был выбран ЯП Swift, поскольку Objective-C весьма сложен для изучения и понимания, чем Swift [5], который для написания приложения требует меньше кода, удобен для чтения, имеет больше возможностей, а также поддерживает полноценное взаимодействие с кодом на Objective-C, что может быть удобно, если необходимо использовать библиотеки, написанные на этом языке.

Выбор среды разработки является почти безальтернативным, поскольку любой проект для iOS необходимо собирать в среде Xcode, хотя разработку можно вести и в различных текстовых редакторах таких как Atom, SublimeText, VS Code и других, но для удобства была выбрана среда Xcode, чтобы не производить разработку приложения, разработку интерфейсов и сборку приложения, используя несколько программных продуктов.

Для запуска и тестирования приложения были использованы как эмуляторы устройств с операционной системой iOS, предоставляемые средой

разработки Xcode, так и физические устройства iPhone 7 и iPhone 11 Pro, которые в полной мере позволяют оценить приложение с точки зрения пользовательского опыта (UX).

Наиболее популярным решением для баз данных на языке Swift являются SQLite, Core Data и Realm Database. Среди перечисленных вариантов была выбрана база данных Realm Database, поскольку её развёртывание происходит намного быстрее и проще. Realm Database – это бесплатная и с открытым исходным кодом объектная база данных. С помощью данной БД можно без особых усилий связать интерфейс и данные, что позволяет создавать отзывчивый и современный интерфейс для мобильного приложения [6]. Преимущества и причины выбора данной базы, следующие:

- простая установка;
- простое использование;
- скорость работы;
- кроссплатформенность;
- масштабируемость;
- хорошая документация и поддержка разработчиков.

Главным преимуществом использования Realm является простота и быстрота интеграции базы данных в мобильное приложение, что позволяет сконцентрироваться на разработке функциональности приложения [6].

Несмотря на то, что Realm Database бесплатна, для того чтобы синхронизировать данные между устройствами понадобится использовать Realm Object Server, который доступен только по подписке. Для проверки работы приложения можно воспользоваться бесплатной месячной версией, однако разработка приложения с использованием сервера синхронизации и без отличается лишь способом создания объекта базы данных и авторизацией пользователей, а код для добавления, обновления и удаления данных ничем не отличается в обоих случаях [7].

Так как разработчики Realm не рекомендуют хранить файлы большого размера в базе, то для преодоления этого ограничения хранение файлов и

фотографий пользователей будет осуществляться с помощью облачного файлового хранилища Firebase, а в Realm будет храниться путь к файлу в хранилище.

2.2 Планирование разработки и оценка бюджета

Чтобы определить трудоемкость и стоимость проекта разработки мобильного приложения, а также обосновать его целесообразность необходимо провести его планирование. Для этого воспользуемся инструментальным средством MS Project. Данное программное обеспечение позволяет провести расчет бюджета проекта, составить календарный план, сетевой график, график ресурсов, диаграммы использования ресурсов и другие графики и схемы [8].

В таблице ниже представлен список задач проекта, необходимых для его реализации, также приведены длительности каждой задачи и последовательность их выполнения.

Таблица 14 – Список задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало реализации проекта	–	–
2	Программирование	9 дней	–
3	Разработка модуля файлового хранилища	6 дней	–
4	Постановка задачи	2 дня	1
5	Разработка структуры базы данных	2 дня	4
6	Разработка интерфейса	2 дня	4
7	Привязка интерфейса к базе данных	2 дня	5;6
8	Разработка модуля файлового хранилища завершена	–	7
9	Разработка модуля календаря	8 дней	–
10	Постановка задачи	2 дня	1
11	Разработка структуры базы данных	2 дня	10
12	Разработка интерфейса	4 дня	10
13	Привязка интерфейса к базе данных	2 дня	11;12

Продолжение таблицы 14

14	Разработка модуля календаря завершена	–	13
15	Разработка модуля чата	9 дней	–
16	Постановка задачи	2 дня	1
17	Разработка структуры базы данных	4 дня	16
18	Разработка интерфейса	6 дней	16
19	Привязка интерфейса к базе данных	2 дня	17;18
20	Разработка модуля чата завершена	–	19
21	Программирование завершено	–	8;14;20
22	Отладка	19 дней	
23	Оплата облачного хранилища	1 день	21
24	Отладка программного комплекса	4 дня	23
25	Тестирование и исправление ошибок	14 дней	24
26	Составление программной документации	3 дня	24
27	Отладка завершена	–	25;26
28	Конец проекта	–	27

В таблице 15 приведен список затрачиваемых ресурсов. В качестве программистов рассматриваются студенты начальных курсов университета, из чего вытекает низкая стандартная ставка.

Таблица 15 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист1	Трудовой	20 000,00 Р/мес	0,00 Р/ч
Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист2	Трудовой	15 000,00 Р/мес	0,00 Р/ч
Программист1	Трудовой	15 000,00 Р/мес	0,00 Р/ч
Интернет	Затраты	–	–
Облачное хранилище	Затраты	–	–

Используя MS Project получим таблицу (рисунок 7) из приведенных выше задач с распределением доступных ресурсов.

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Пн 13.04.20	Пн 13.04.20		
2	Программирование	9 дней	Пн 13.04.20	Чт 23.04.20		Интернет[1 000,00 Р]
3	Разработка модуля файлового хранилища	6 дней	Пн 13.04.20	Пн 20.04.20		
4	Постановка задачи	2 дней	Пн 13.04.20	Вт 14.04.20	1	Программист1[50%];Программист2
5	Разработка структуры базы данных	2 дней	Ср 15.04.20	Чт 16.04.20	4	Программист2[50%]
6	Разработка интерфейса	2 дней	Ср 15.04.20	Чт 16.04.20	4	Программист1[50%];Программист2[50%]
7	Привязка интерфейса к базе данных	2 дней	Пт 17.04.20	Пн 20.04.20	5,6	Программист2
8	Разработка модуля файлового хранилища завершена	0 дней	Пн 20.04.20	Пн 20.04.20	7	
9	Разработка модуля календаря	8 дней	Пн 13.04.20	Ср 22.04.20		
10	Постановка задачи	2 дней	Пн 13.04.20	Вт 14.04.20	1	Программист1[50%];Программист3
11	Разработка структуры базы данных	2 дней	Ср 15.04.20	Чт 16.04.20	10	Программист3[50%]
12	Разработка интерфейса	4 дней	Ср 15.04.20	Пн 20.04.20	10	Программист2[50%];Программист1[50%]
13	Привязка интерфейса к базе данных	2 дней	Вт 21.04.20	Ср 22.04.20	11,12	Программист3
14	Разработка модуля календаря завершена	0 дней	Ср 22.04.20	Ср 22.04.20	13	
15	Разработка модуля чата	9 дней	Пн 13.04.20	Чт 23.04.20		
16	Постановка задачи	1 день	Пн 13.04.20	Пн 13.04.20	1	Программист1
17	Разработка структуры базы данных	4 дней	Вт 14.04.20	Пт 17.04.20	16	Программист1[50%]
18	Разработка интерфейса	6 дней	Вт 14.04.20	Вт 21.04.20	16	Программист1[50%]
19	Привязка интерфейса к базе данных	2 дней	Ср 22.04.20	Чт 23.04.20	17,18	Программист1
20	Разработка модуля чата завершена	0 дней	Чт 23.04.20	Чт 23.04.20	19	
21	Программирование завершено	0 дней	Чт 23.04.20	Чт 23.04.20	8,14,20	
22	Отладка	19 дней	Пт 24.04.20	Ср 20.05.20		Интернет[1 000,00 Р]
23	Оплата облачного хранилища	1 день	Пт 24.04.20	Пт 24.04.20	21	Программист1;Облачное хранилище[2 000,00 Р]
24	Отладка программного комплекса	4 дней	Чт 30.04.20	Чт 30.04.20	23	Программист1;Программист2;Программист3
25	Тестирование и исправление ошибок	14 дней	Пт 01.05.20	Ср 20.05.20	24	Программист2;Программист3
26	Составление программной документации	3 дней	Чт 01.05.20	Вт 05.05.20	24	Программист2;Программист3
27	Отладка завершена	0 дней	Ср 20.05.20	Ср 20.05.20	25,26	
28	Конец проекта	0 дней	Ср 20.05.20	Ср 20.05.20	27	

Рисунок 7 – Перечень задач проекта

Создание такой таблицы в MS Project позволяет получить автоматически созданные графики и таблицы, демонстрирующие распределение ресурсов, перегрузку программистов, начало и окончание проекта по датам, а самое главное расчёт бюджета проекта [8].

Диаграмма Ганта (рисунок 8) наглядно демонстрирует порядок выполнения задач и доли распределения трудовых ресурсов по задачам.

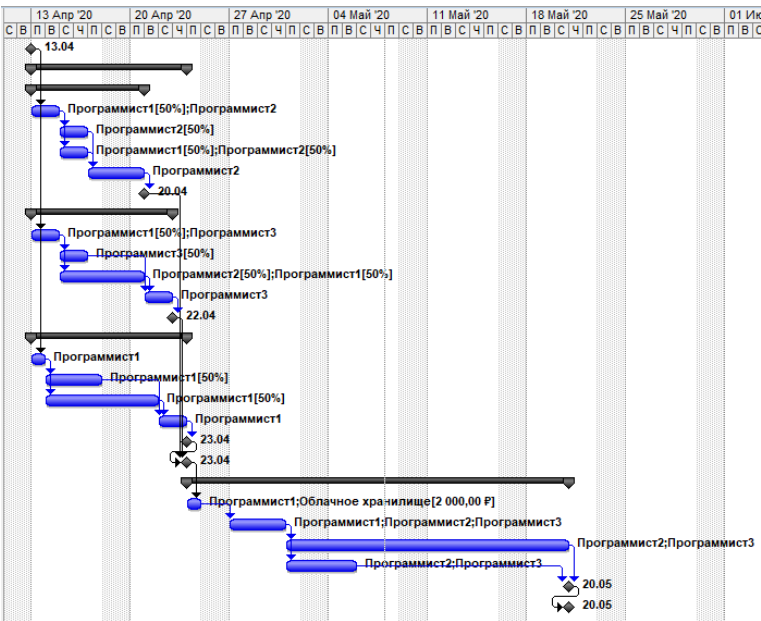


Рисунок 8 – Диаграмма Ганта

На диаграмме хорошо заметна высокая параллельность задач разработки проекта. Это достигается полной независимостью разрабатываемых модулей.

Средства MS Project также позволяют отслеживать перегрузку или недогрузку трудовых ресурсов, что на начальных этапах помогает скорректировать некоторые параметры проекта [8]. На рисунках 9 – 11 приведены графики трудовых ресурсов.

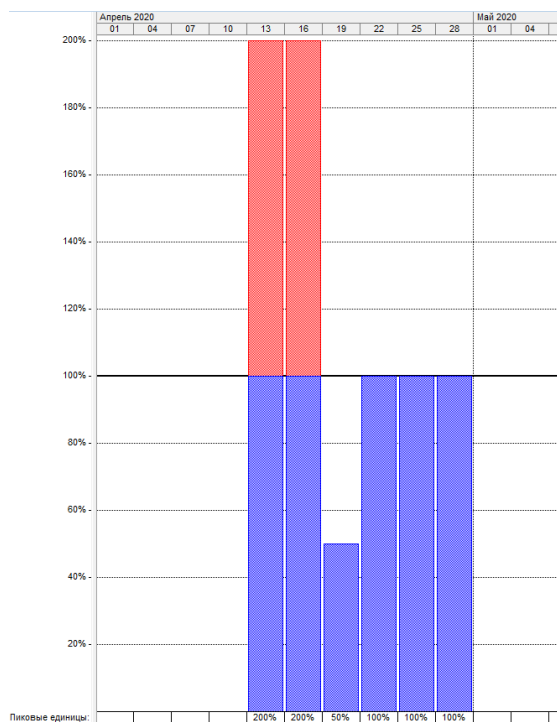


Рисунок 9 – График загруженности ресурса «Программист 1»

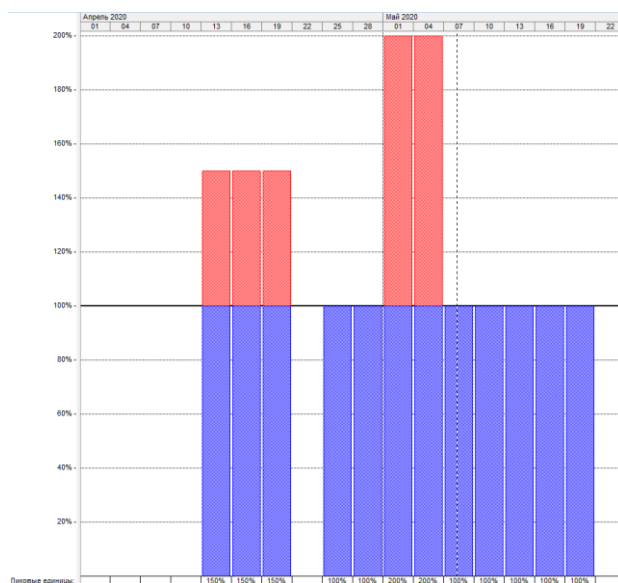


Рисунок 10 – График загруженности ресурса «Программист 2»

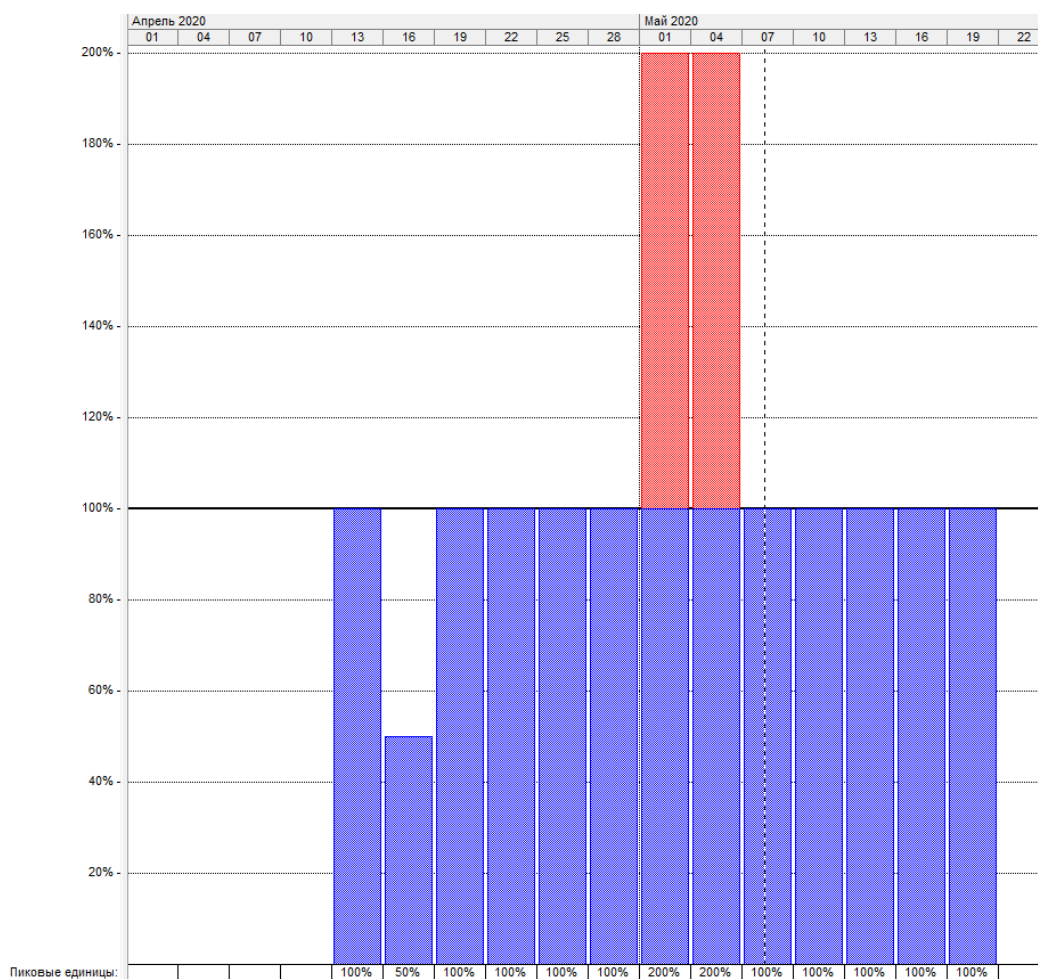


Рисунок 11 –График загрузки ресурса «Программист 3»

По приведенным графикам загрузки можно сделать вывод, что все программисты имеют небольшую перегруженность в ходе реализации проекта. Исходя из этих данных, мы можем принять различные решения:

- нанять еще одного программиста, чтобы разгрузить остальных;
- сделать решаемые задачи последовательными, что сильно увеличит длительность проекта;
- увеличить бюджет проекта, чтобы оплачивать сверхурочные часы работы.

Рассчитаем также бюджет проекта с помощью MS Project и получим итоговый отчет о проекте (рисунок 12).

Даты			
Начало:	Пн 13.04.20	Окончание:	Ср 20.05.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	28 дней	Оставшаяся:	28 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	28 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	560 ч	Оставшиеся:	560 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	560 ч	Процент завершения:	0%
Затраты			
Текущий план:	60 750,00 Р	Оставшиеся:	60 750,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	60 750,00 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	28	Трудовые ресурсы:	2
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	3
Завершенные задачи:	0	Материальные ресурсы:	0
Итого задач:	28	Итого ресурсов:	5

Рисунок 12 – Отчет по проекту

По отчету получаем информацию, что длительность реализации проекта 28 дней, а предполагаемые затраты 60 750 руб.

2.3 Проектирование пользовательского интерфейса

На основе анализа функциональных требований и ПО аналогичного назначения выделим следующие части интерфейса приложения:

- окно авторизации пользователя;
- окно регистрации пользователя;
- чат пользователей;
- личный кабинет;
- файловое хранилище;
- календарь событий;
- окно добавления события.

Разработанные прототипы для меню авторизации и регистрации представлены на рисунке 13.

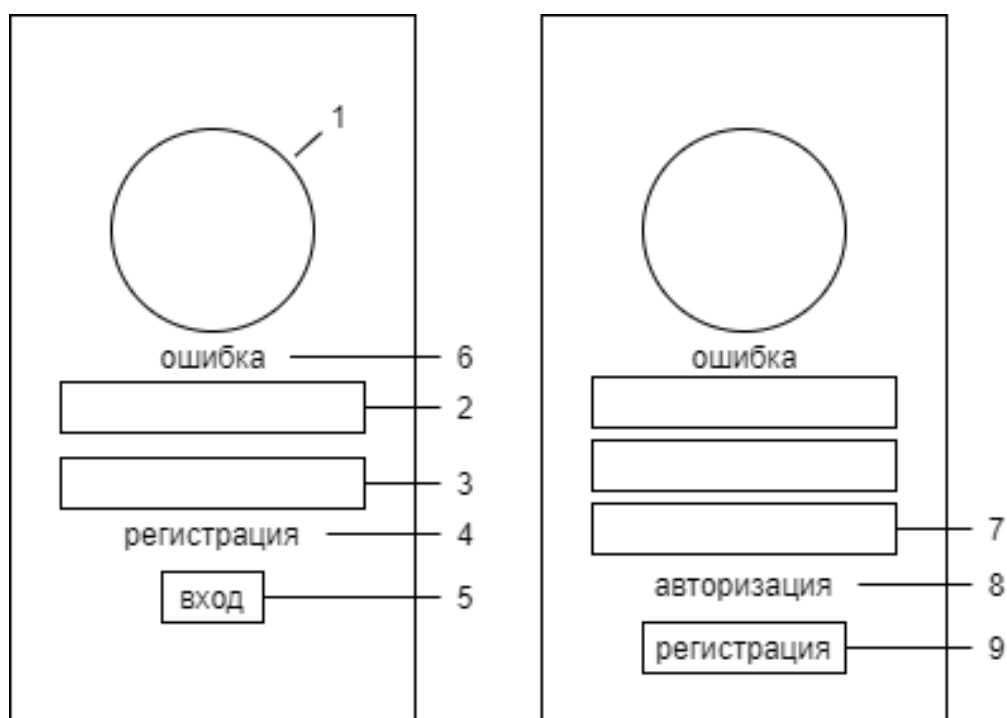


Рисунок 13 – Прототип меню авторизации и регистрации

На рисунке цифрами отмечено:

- 1 – логотип приложения;
- 2 – поле ввода логина пользователя;
- 3 – поле ввода пароля;
- 4 – кнопка смены меню на окно регистрации;
- 5 – кнопка входа в приложение;
- 6 – поле вывода сообщения об ошибке;
- 7 – поле ввода отображаемого в чате имени пользователя;
- 8 – кнопка возврата в меню авторизации пользователя;
- 9 – кнопка регистрации.

Не отмеченные элементы интерфейса окна регистрации выполняют те же функции, что и элементы меню авторизации.

После входа в систему пользователя попадает в чат, прототип которого приведен на рисунке 14.

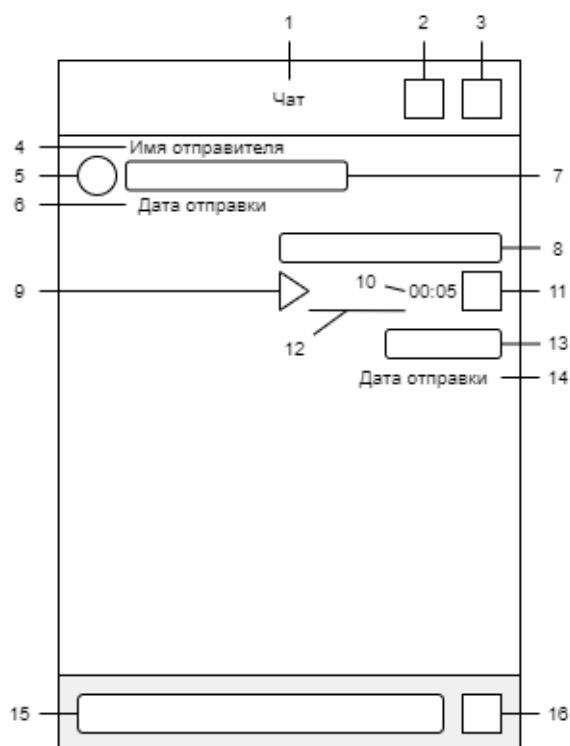


Рисунок 14 – Прототип интерфейса чата

На этом прототипе цифрами отмечено:

- 1 – наименование окна;
- 2 – кнопка выбора избранных сообщений;
- 3 – кнопка перехода в личный кабинет пользователя;
- 4 – отображаемое имя отправителя;
- 5 – фото пользователя;
- 6 – дата отправки;
- 7 – поле, содержащее текст сообщения отправителя;
- 8 – поле, содержащее текст сообщения авторизованного пользователя;
- 9 – кнопка воспроизведения аудиосообщения;
- 10 – длительность аудиосообщения;
- 11 – кнопка открытия текстового представления аудиосообщения;
- 12 – полоса для отслеживания проигрываемого аудиосообщения;
- 13 – текст аудиосообщения;
- 14 – дата отправки;
- 15 – поле для ввода сообщения;
- 16 – кнопка отправки сообщения.

Для прототипа чата необходимо отметить, что элемент под номером 10 также должен изменяться при воспроизведении аудиосообщения и показывает, сколько осталось до конца сообщения. Если не удалось получить осмысленный текст для аудиосообщения тогда кнопка под номером 11 не должна появляться. Для кнопки 16 должно быть предусмотрено 2 состояния:

- если пользователь не ввел текст в поле, тогда нажатие кнопки инициализирует запись аудиосообщения;
- иначе пользователь отправляет текстовое сообщение.

Прототип для интерфейса личного кабинета пользователя представлен на рисунке 15.

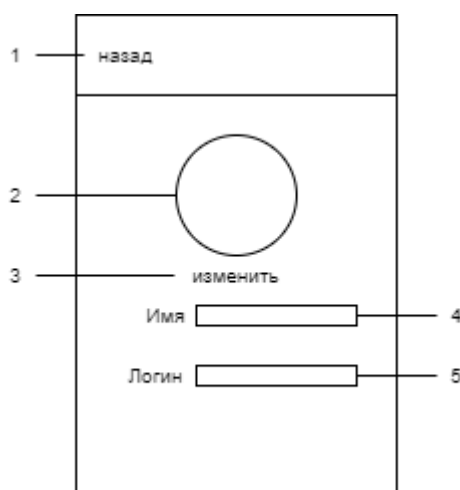


Рисунок 15 – Прототип интерфейса личного кабинета

Цифрами отмечено:

- 1 – кнопка возврата в чат пользователей;
- 2 – фото пользователя;
- 3 – кнопка загрузки другого фото;
- 4 – поле с отображаемым именем пользователя;
- 5 – поле с логином пользователя.

Представим прототип для интерфейса файлового хранилища на рисунке 16.

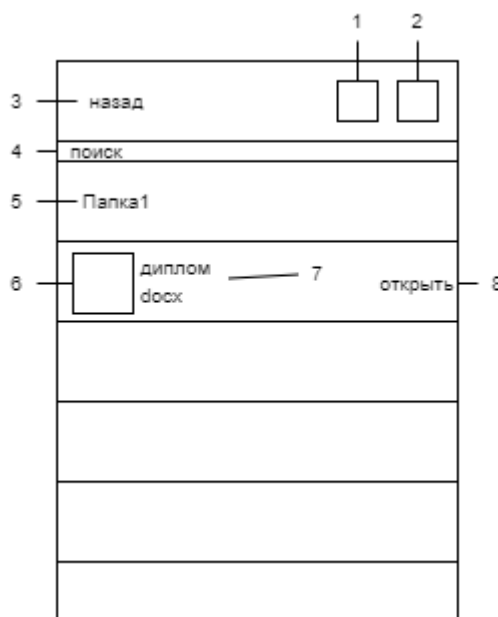


Рисунок 16 – Прототип интерфейса файлового хранилища

На прототипе хранилища цифрами отмечено:

- 1 – кнопка для добавления каталога в текущий;
- 2 – кнопка добавления файла в текущий каталог;
- 3 – кнопка возвращения в чат;
- 4 – поиск каталога;
- 5 – ячейка с названием каталога;
- 6 – миниатюрное изображения файла;
- 7 – название документа;
- 8 – кнопка для открытия файла.

Стоит отметить, что кнопки 1 и 2 будут отображаться только для пользователей, имеющих привилегии добавления файлов и каталогов, т.е. администраторов.

Для реализации календаря событий планируется использование библиотеки, представляющей интерфейс календаря, приближенного к системному для операционной системы iOS. Однако приведем прототип окна добавления события в календарь (рисунок 17).

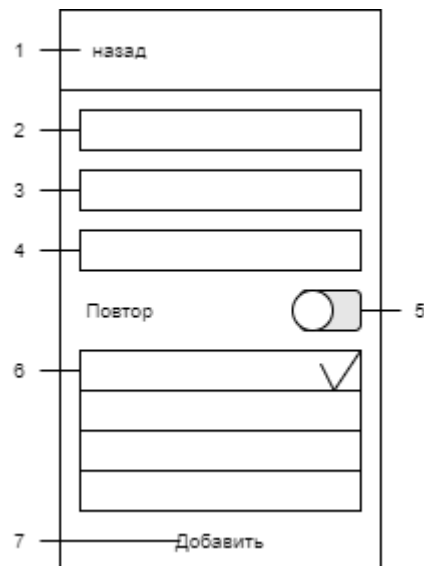


Рисунок 17 – Прототип интерфейса окна добавления события в календарь

На рисунке цифрами отмечено:

- 1 – кнопка возврата в чат;
- 2 – поле названия события;
- 3 – поле ввода даты и времени начала события;
- 4 – поле ввода даты и времени окончания события;
- 5 – переключатель повторяющихся событий;
- 6 – кнопки вариантов периодичности события;
- 7 – кнопка добавить событие.

Необходимо уточнить, что доступ к этому окну также могут получить только администраторы, а кнопки под номером 6 появляются только при включенном переключателе 5.

2.4 Проектирование архитектуры приложения

Приложение реализуется с помощью архитектурного шаблона Model View Controller (MVC), поэтому разрабатываемые классы разделим на 3 группы:

- классы модели;
- классы контроллеров;
- классы представлений.

2.4.1 Проектирование классов модели

Поскольку приложение разрабатывается с помощью объектной базы данных, то к классам модели будут относиться классы сущностей БД. Диаграмма классов модели представлена на рисунке 18.

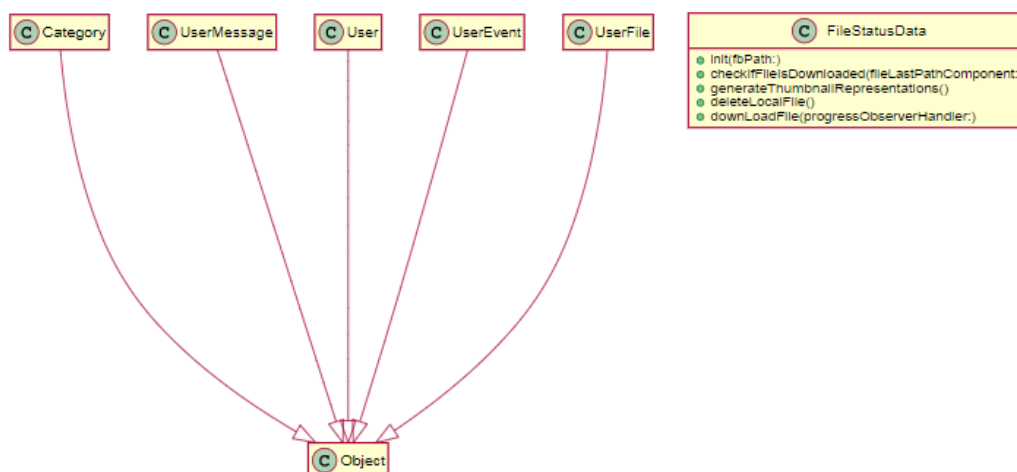


Рисунок 18 – Диаграмма классов модели

На диаграмме выше представлены классы базы данных и класс для работы с файлами. Классы, относящиеся к БД, а именно **Category**, **UserMessage**, **User**, **UserEvent** и **UserFile** соответствуют классам концептуальной модели предметной области и предназначены для хранения информации о пользователях, их сообщениях, о событиях, добавляемых в календарь, и о каталогах с файлами. Данные классы наследуются от класса **Object**, который является обязательным базовым классом для всех классов сущностей Realm Database [6]. Класс **FileStatusData** предназначен для работы с файлами, которые администратор добавляет в файловое хранилище. Данный класс отвечает за загрузку файла, его удаление, генерацию миниатюрного изображения файла и управление состоянием файлов в хранилище. Состояния файла могут быть следующими:

- доступен для загрузки;
- загружается;
- загрузка на паузе;
- загружен.

2.4.2 Проектирование классов представлений

При разработке интерфейсов приложений в среде Xcode используется программа Interface Builder, позволяющая создавать интерфейс с помощью перетаскивания готовых элементов на экран пользователя [9]. Этих элементов достаточно для разработки любых приложений, поэтому часто нет необходимости создавать свои классы для представлений. Однако в данном случае, помимо стандартных элементов, были также разработаны представления для различных ячеек:

- ячейки для текстовых сообщений пользователей;
- ячейки для аудиосообщений пользователей;
- ячейки для отображения документов в файловом хранилище.

Диаграмма классов для классов представлений приведена на рисунке 19.

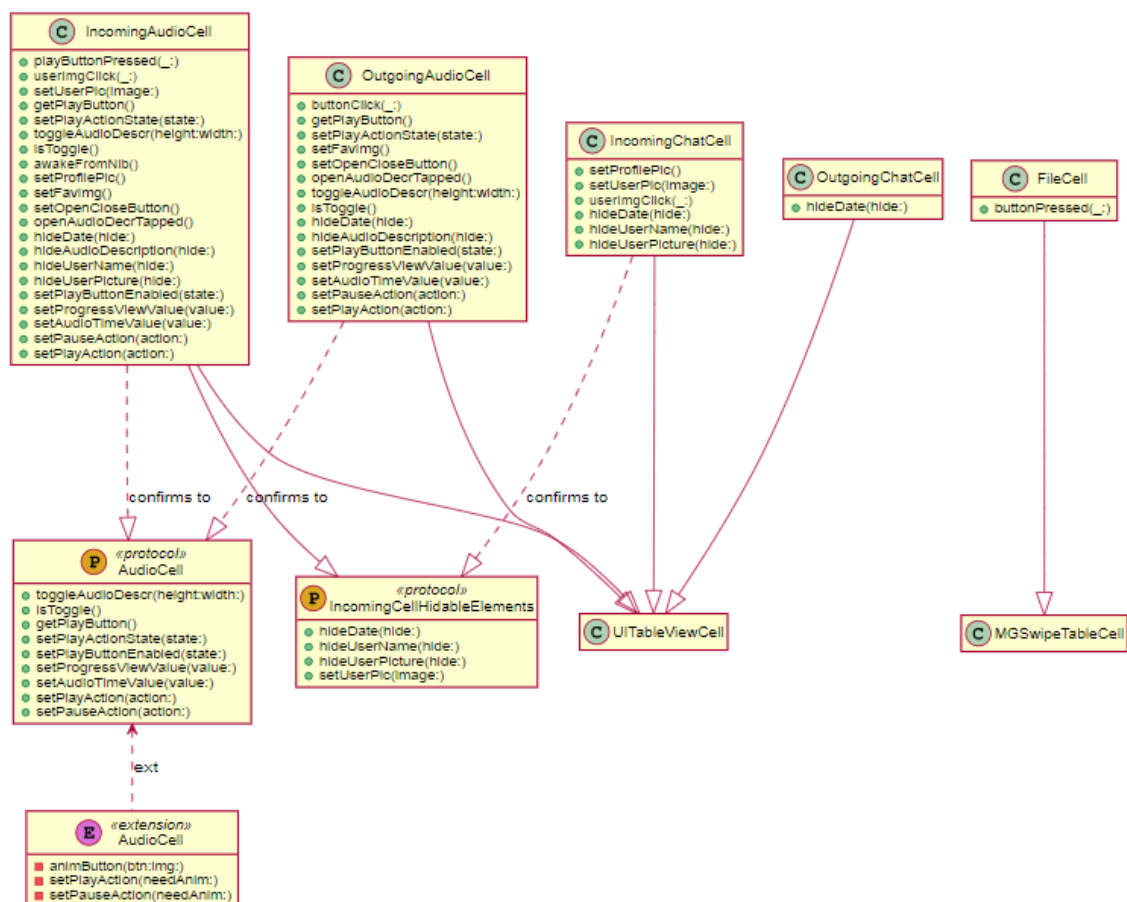


Рисунок 19 – Диаграмма классов представлений

Классы `IncomingAudioCell` и `OutgoingAudioCell` предназначены для отображения входящих и исходящих аудиосообщений. Данные классы имеют общие элементы управления, функции взаимодействия которых вынесены в протокол (в других языках программирования – интерфейс) `AudioCell`, который определяет функции скрывания/открывания текстового описания аудиосообщения, обработки нажатия кнопок `start/pause` и обработка отображения оставшегося времени проигрывания аудиосообщения. Интерфейс `AudioCell` расширен (на диаграмме «extension» `AudioCell`) функциями, анимирующими нажатия кнопок `start` и `pause`.

Все классы представленных ячеек наследуются от класса `UITableViewCell`, являющимся стандартным для любых ячеек в ЯП Swift.

Классы входящих ячеек `IncomingAudioCell` и `IncomingChatCell` имеют общий интерфейс `IncomingCellHidableElements`, который определяет функции для скрывания/показа общих для всех входящих ячеек элементов, таких как дата отправки, имя пользователя и фото пользователя.

Класс `FileCell` управляет отображением файловых ячеек в хранилище и наследуется от класса `MGSwipeTableCell`, который позволяет реализовать добавление действий путем свайпа (движением пальца влево/вправо удерживая элемент интерфейса) ячейки. В данном случае это применяется для возможности удаления файла с устройства пользователя.

2.4.3 Проектирование классов контроллеров

Приложение разбито на 3 самостоятельные части, поэтому необходимо реализовать контроллер для чата пользователей, файлового хранилища и календаря событий, а также еще два контроллера для обработки авторизации в систему и для личного кабинета пользователя. Классы контроллеров представлены на рисунке 20.

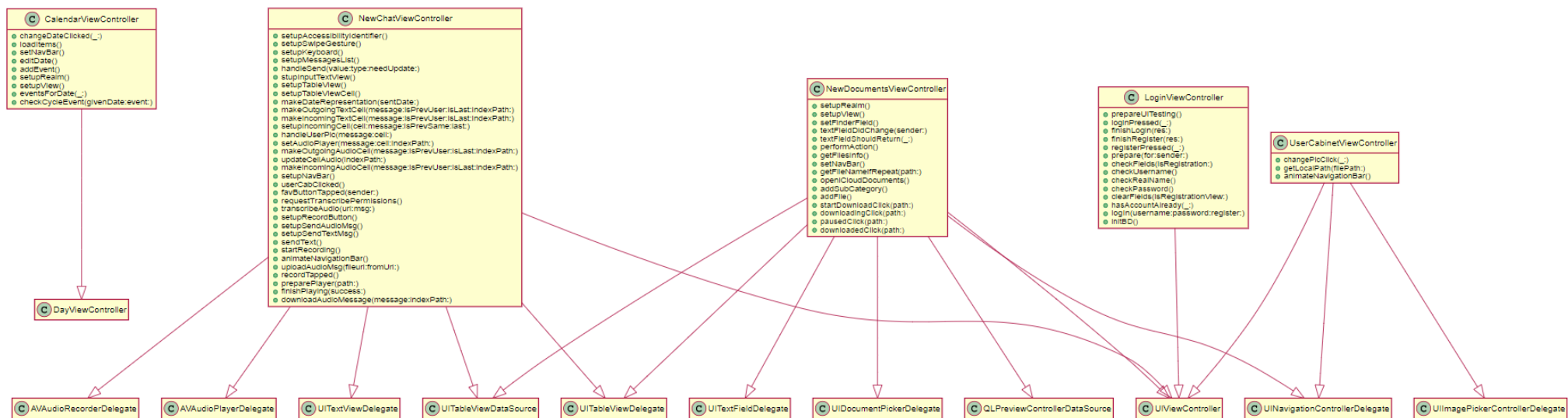


Рисунок 20 – Диаграмма классов контроллеров

Все контроллеры приложения, кроме контроллера календаря, наследуются от стандартного класса `UIViewController`, общего для любого контроллера в Swift [10]. Контроллер чата `NewChatViewController` выполняет большое количество функций по настройке приложения как до начала работы, так и после. Например, перед началом функционирования чата, контроллер должен:

- запросить доступ к микрофону пользователя;
- запросить разрешение на отправку аудиосообщений для отправки в сервис распознавания речи;
- установить для распознавания жесты свайпа вправо/влево;

Контроллер чата отвечает за добавление сообщений в БД, добавление аудиосообщений в облачное хранилище `Firebase`, отправку аудиофайла в сервис распознавания речи, обработку отправки сообщений, формирование ячеек для сообщений в чате, проигрывание и записывание аудиосообщений, запуск переходов в файловое хранилище, календарь событий и личный кабинет пользователя. Для работы с аудиосообщениями, а именно записью и проигрыванием контроллеру необходимо реализовать интерфейс `AVAudioRecorderDelegate` и `AVAudioPlayerDelegate`, для работы с таблицами `UITableViewDataSource` и `UITableViewDelegate` [10], а реализация `UITextViewDelegate` необходима для обработки поля ввода сообщений.

Контроллер файлового хранилища отвечает за организацию поиска по каталогам, добавление каталогов и файлов, обработку загрузки и удаления файлов, а также определение наличия файла на локальном устройстве. Перед началом отображения контроллеру необходимо определить обладает ли пользователь правами администратора и настроить интерфейс в зависимости от наличия прав. Данный контроллер также реализует множество интерфейсов, необходимых для корректной работы, назначения некоторых уже были описаны в контроллере чата. Контроллер хранилища реализует `UIDocumentPickerDelegate` для получения пути к выбранному администратором файла [10] для загрузки в `Firebase`, реализация

QLPreviewControllerDataSource позволяет получать миниатюрные изображения документов и файлов, UITextFieldDelegate необходим для корректной обработки поиска каталогов.

LoginViewController реализует логику авторизации и регистрации пользователей. Контроллер одновременно обрабатывает представления окна входа и окна регистрации, он также отвечает за инициализацию связи с БД.

UserCabinetViewController ответственен за отображение данных пользователя (фото, логин, отображаемое имя), а также за загрузку фотографии, выбранной пользователем, для чего реализован UIImagePickerControllerDelegate, с помощью которого можно получить выбранную фотографию из альбома пользователя.

Контроллер календаря CalendarViewController наследуется от DayViewController, класса из загружаемой библиотеки, которая предоставляет готовый интерфейс календаря. В классе CalendarViewController определены функции:

- обработка смены даты в календаре;
- обработка добавления события в календарь;
- декларативное создание формы добавления события в календарь;
- обработка циклических событий;
- правка сообщения в чат о добавлении нового события.

3 Реализация мобильного приложения

3.1 Реализация структур данных

Поскольку при реализации приложения использовалась объектная база данных Realm, то структуры данных модели представляют собой классы. На рисунке 21 приведены поля и связи классов структуры данных.

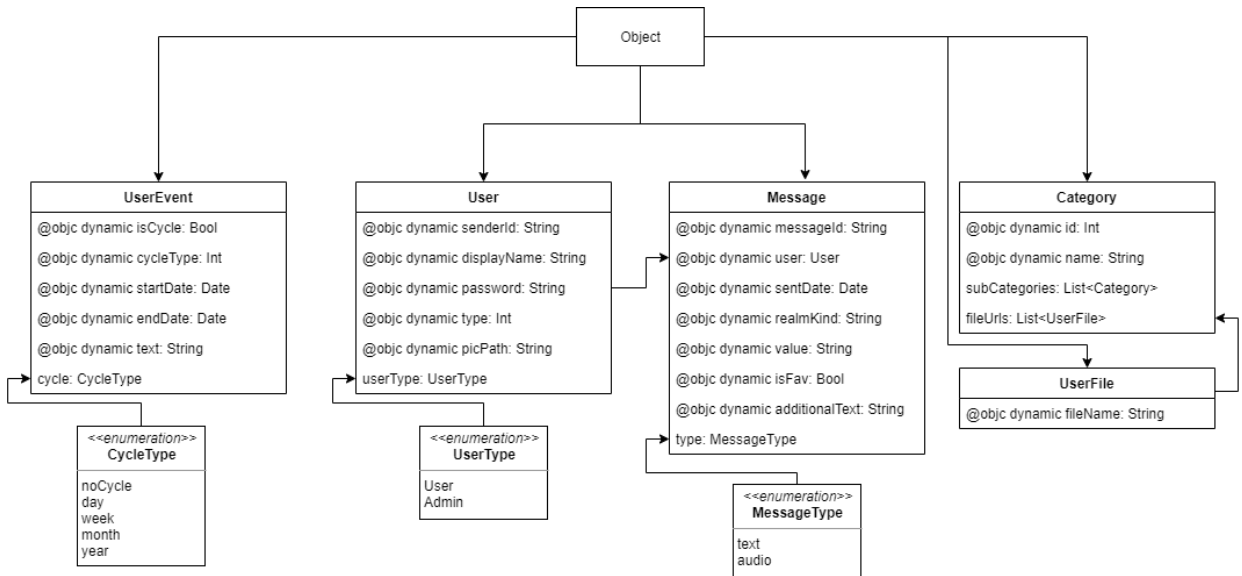


Рисунок 21 – Классы структур данных

Все классы наследуются от общего класса Object, который позволяет библиотеке RealmDatabase работать с ними [6]. Поля, отмеченные атрибутами @objc dynamic, будут добавляться в БД, а остальные необходимы для удобной работы с этими классами. Например, поле cycle в классе UserEvent имеет тип перечисления CycleType, но поскольку в БД нельзя хранить собственные типы данных, то в базе хранится поле cycleType типа Int, а в приложении мы используем поле cycle, которое автоматически переводит cycleType в тип перечисления CycleType. Поле user класса Message реализует связь «один-к-одному», а поля класса subCategories и fileUrls реализуют связь «один-ко-многим».

3.2 Реализация пользовательского интерфейса

Для реализации пользовательского интерфейса был использован встроенный в среду разработки Xcode графический редактор, позволяющий

использовать элементы интерфейса путем перетаскивания их из списка на форму, представляющую собой экран телефона. Совокупность всех форм и определенных переходов между ними показана на рисунке 22.

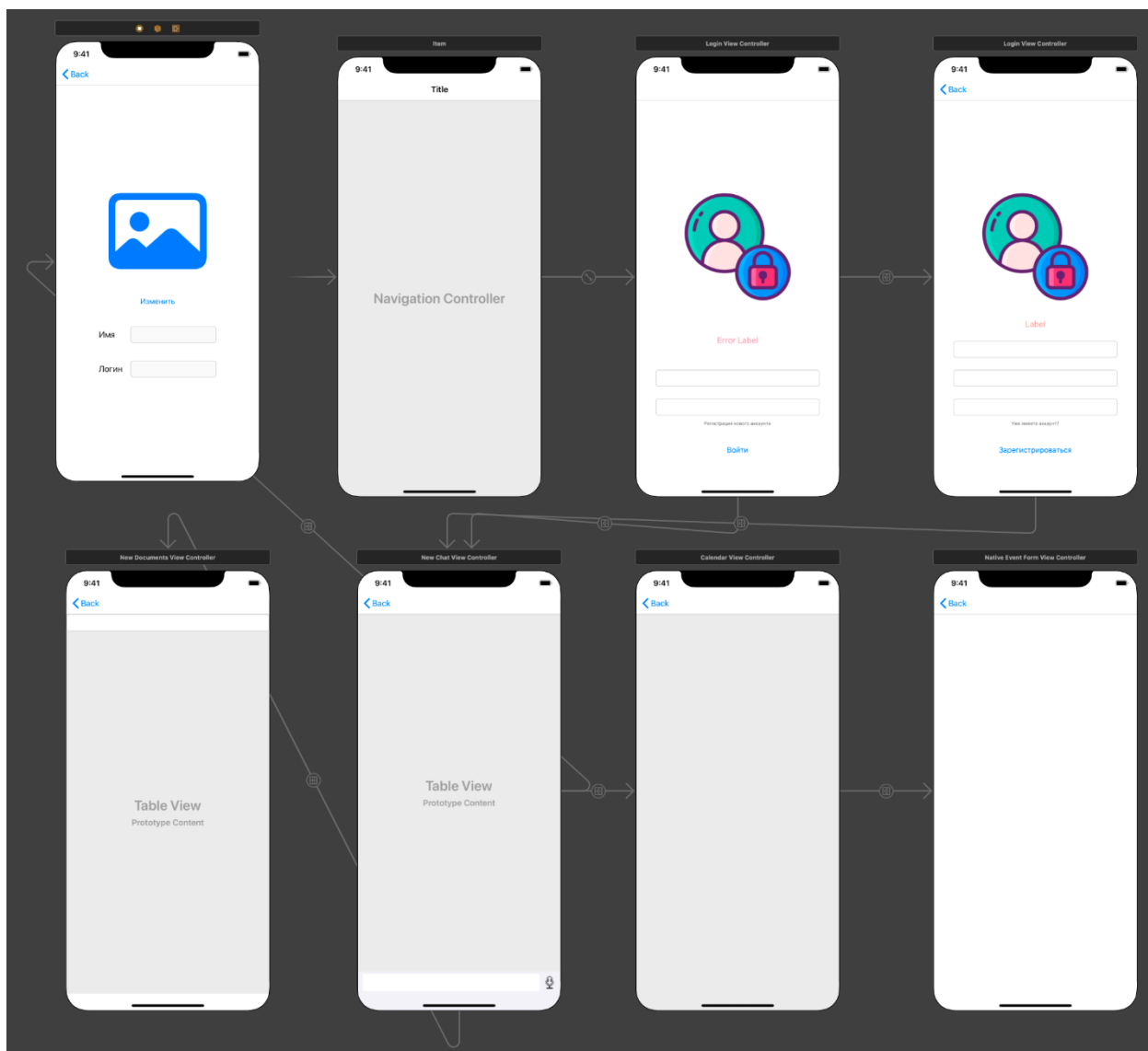


Рисунок 22 – Экран и переходы

Помимо использования графического редактора, также программным методом добавляются некоторые элементы интерфейса, например, кнопки, которые зависят от типа пользователя. Также календарь событий и форма добавления событий реализованы программно.

Поскольку на большинстве экранов главным элементом является таблица, то были реализованы различные ячейки для этих таблиц. На рисунках 23 – 25 показаны ячейки, используемые для файлового хранилища и для входящих сообщений чата.

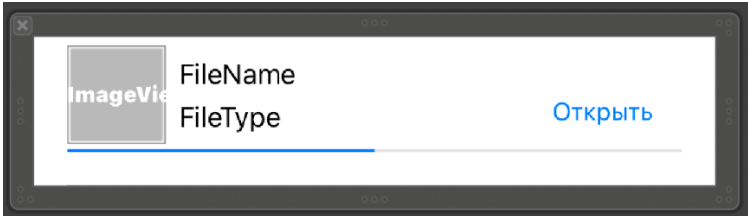


Рисунок 23 – Ячейка таблицы файлового хранилища

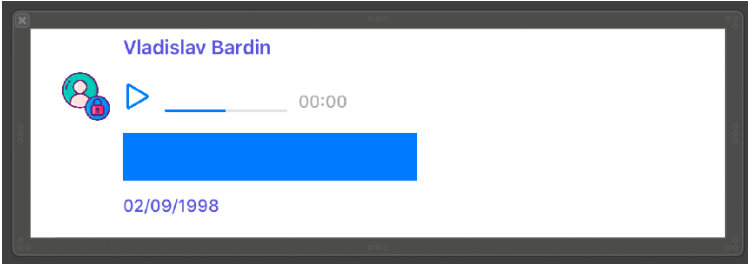


Рисунок 24 – Ячейка входящего аудиосообщения

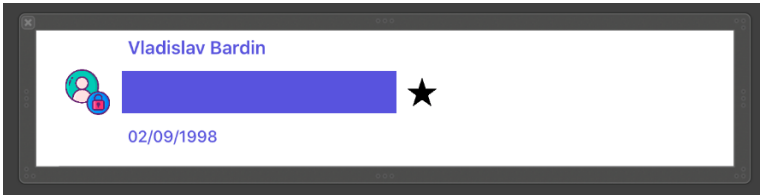


Рисунок 25 – Ячейка входящего текстового сообщения

На рисунках 26 – 27 показаны ячейки, используемые для исходящих сообщений в чате пользователей.

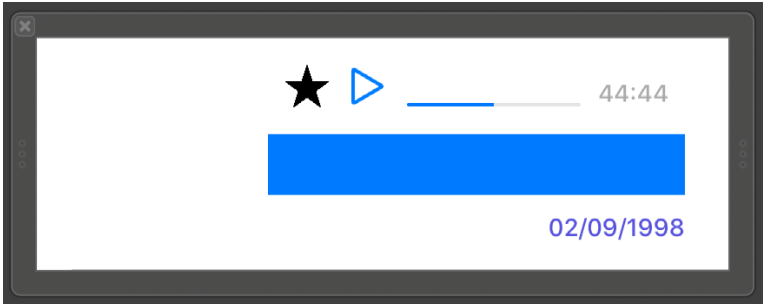


Рисунок 26 – Ячейка исходящего аудиосообщения

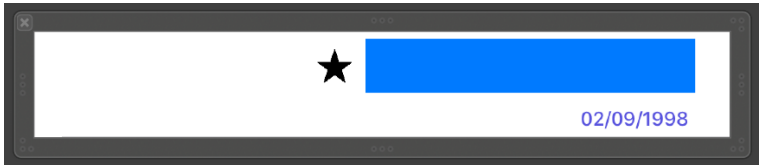


Рисунок 27 – Ячейка исходящего текстового сообщения

Некоторые элементы ячеек редактируются программно. Например, скругляются углы прямоугольника для сообщений, некоторые элементы скрываются/показываются в зависимости от расположения, формат даты для сообщений может изменяться в зависимости от того, насколько давно было отправлено сообщение, некоторые кнопки могут менять иконку в зависимости от состояния, например, аудиосообщения (загружено/проигрывает/готово к проигрыванию).

Финальные версии форм интерфейса для чата пользователей и личного кабинета показаны на рисунках 28 и 29.

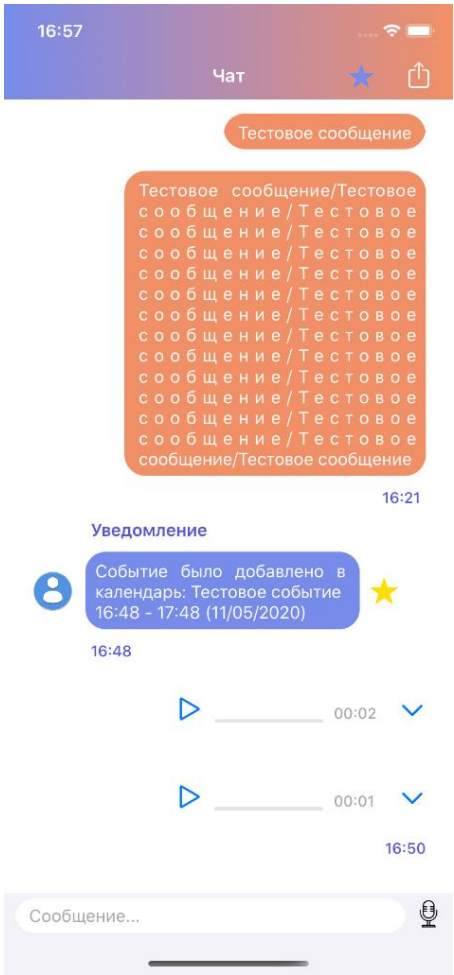


Рисунок 28 – Чат пользователей

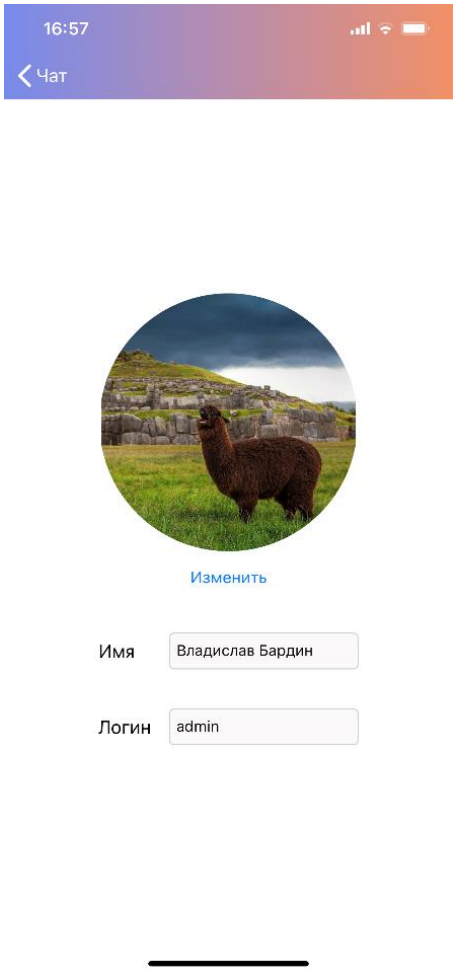


Рисунок 29 – Личный кабинет пользователя

Финальные версии форм интерфейса для файлового хранилища, формы добавления события в календарь и самого календаря показаны на рисунках 30 – 32.

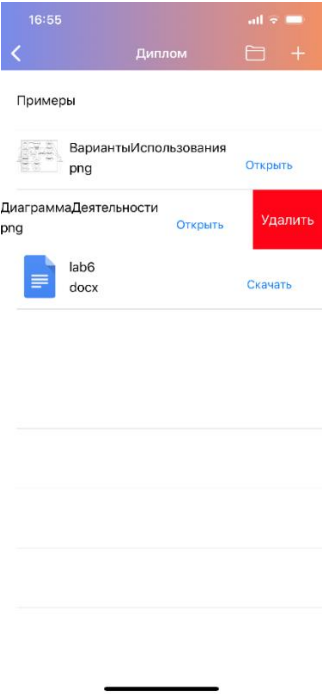


Рисунок 30 – Файловое хранилище

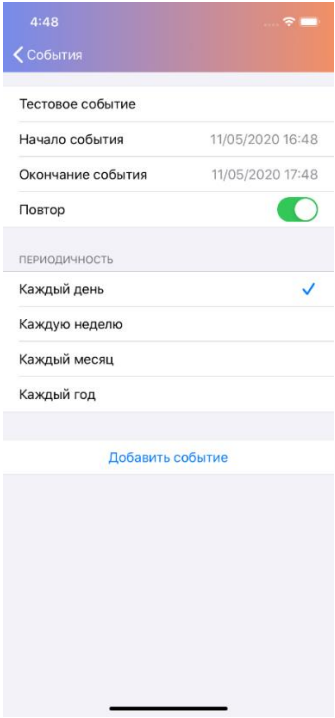


Рисунок 31 – Добавление события

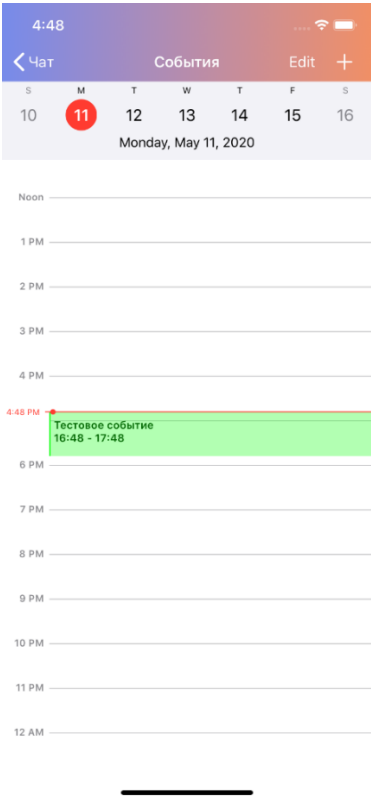


Рисунок 32 – Календарь событий

На приведенных рисунках показана реализация спроектированного ранее интерфейса.

3.3 Развертывание приложения

Разработка мобильного приложения осуществлялась на языке программирования Swift в среде Xcode. Листинг классов приведен в приложении А.

На рисунке 33 приведена диаграмма компонентов, показывающая структуру приложения, включая исходные коды, файлы интерфейса и библиотеки.

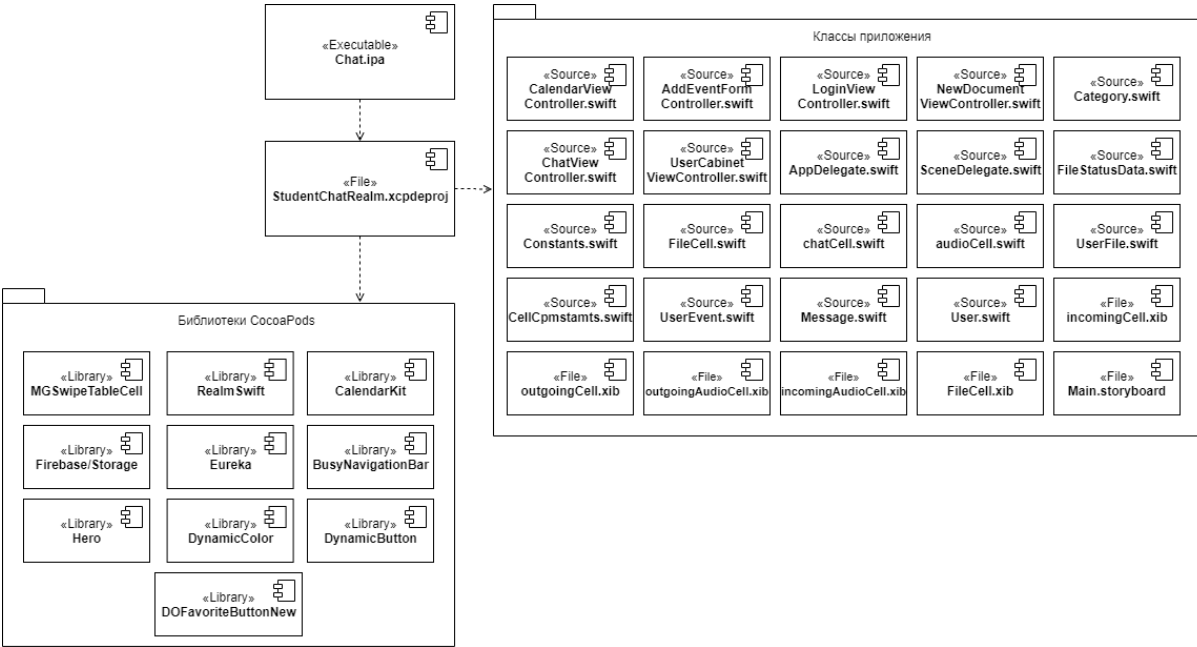


Рисунок 33 – Диаграмма компонентов

Главный файл проекта зависит от библиотек, устанавливаемых с помощью CocoaPods (Система управления пакетами), и от разработанных классов приложения, включающих в себя классы модели, контроллеров, представлений, классы с различными константами, классы для обработки некоторых состояний приложения при его сворачивании, открытии, закрытии, неожиданном завершении и.т.д. Исходные коды всех контроллеров приведены в приложении А.

В таблице 16 опишем назначение использованных библиотек.

Таблица 16 – Назначение библиотек

Наименование библиотеки	Назначение
MGSwipeTable	Добавление ячейкам файлов в хранилище возможность удаления свайпом влево
RealmSwift	Библиотека для работы с объектной базой данных RealmDatabase
CalendarKit	Предоставляет интерфейс, близкий к системному в iOS
Firebase/Storage	Библиотека для работы с облачным хранилищем Firebase
Eureka	Библиотека для декларативного способа создания форм. Использовалась для создания формы добавления события в календарь
BusyNavigationBar	Анимация верхней части приложения при загрузке/отправке аудиосообщений
Hero	Библиотека для работы с анимацией
DynamicColor	Библиотека для работы с различными цветами
DynamicButton	Анимированная кнопка, использовалась для открытия/закрытия текста аудиосообщения
DOFavoriteButtonNew	Анимация для кнопки избранных сообщений

На рисунке 34 представлена диаграмма развертывания, показывающая распределение аппаратных и программных компонентов по узлам и их связи.

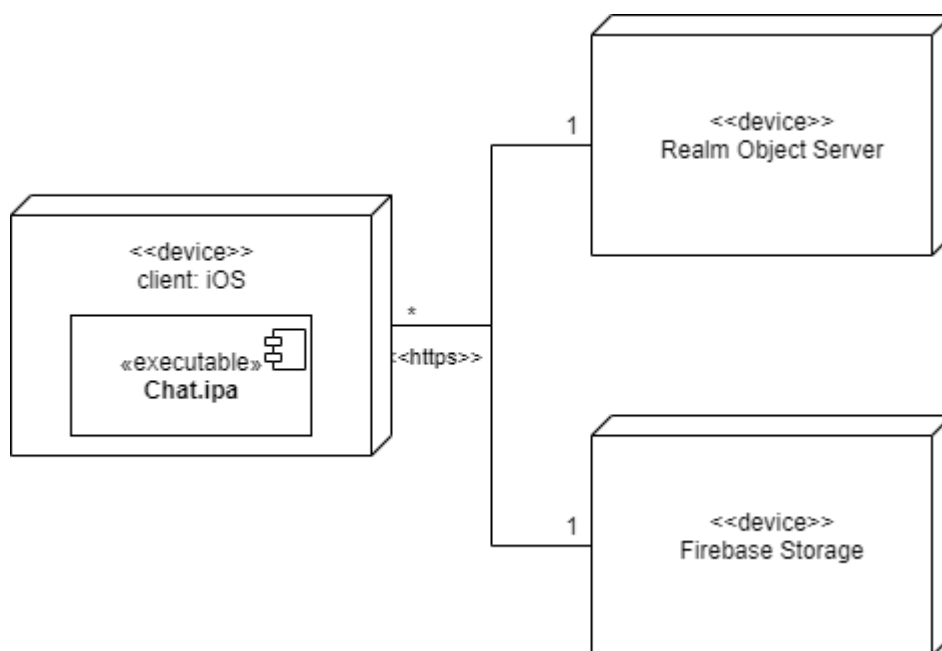


Рисунок 34 – Диаграмма развертывания

На рисунке показано, что клиентская часть приложения работает на мобильных устройствах под управлением операционной системы iOS. Приложение связывается по защищенному каналу с Realm Object Server, который выполняет синхронизацию базы данных, а также безопасную авторизацию пользователей и с Firebase Storage для обмена файлами с хранилищем.

3.4 Сравнение разработанного приложения с прототипом

В пункте 1.3 было получено, что социальная сеть «ВКонтакте» является наилучшим аналогичным программным обеспечением по группе выделенных критериев. Сравним разработанное приложение с мобильным приложением «ВКонтакте» по следующим признакам:

- организация работы с файловым хранилищем;
- функциональность чата пользователей;
- организация календаря событий;
- UX/UI-дизайн.

Было выявлено, что ведущими признаками являются организация работы с файловым хранилищем и функциональность чата пользователей,

поэтому при разработке мобильного приложения больше всего внимания было уделено именно этим критериям.

По результатам проведенного анализа, «Вконтакте» имеет наивысшие весовые доли по ведущим критериям, а именно критерии организации работы с файлами и функциональность чата. Однако, во «Вконтакте» позволяет лишь отправлять файлы и затем просматривать их в общем списке, структурированном только по типам файлов (картинки, документы, аудиозаписи), а в разработанном приложении поддерживается разбиение хранилища на каталоги и подкаталоги, что делает использование общего хранилища файлов несомненно удобнее, особенно при обмене большим количеством файлов.

Что касается функциональности чата, разработанное приложение выгодно отличается от «Вконтакте» возможностью перевода аудиосообщений в текст. Данная функция безусловно является востребованной, поскольку такие компании как «Яндекс» и «Mail.ru» добавляют эту возможность в свои приложения «Яндекс.Мессенджер» и «icq new» [11, 12].

По остальным критериям можно выделить, что во «Вконтакте» не поддерживается ведение календаря событий в отличие от созданного приложения, которое предоставляет возможность вести календарь и добавлять обычные или циклические события. По параметру UX/UI-дизайн «Вконтакте» лучше, поскольку имеет больше плавных анимаций, а пользовательский опыт работы с интерфейсом корректировался в течение многих лет с помощью большой базы пользователей, однако данный параметр не является ведущим.

Таким образом разработанное приложение лучше прототипа по трем из четырех критериев, два из которых являются ведущими.

4 Тестирование мобильного приложения

4.1 Выбор режима тестирования приложения

В рамках выпускной работы основными применяемыми методологиями тестирования обычно являются функциональное, структурное и модульное тестирование.

Для проверки выполняемости функциональных требований проводят функциональное тестирование. Существует множество методов функционального тестирования, такие как классы эквивалентности, метод граничных значений, метод функциональных диаграмм, парное тестирование. Данные методы часто применяются для тестирования работы приложения при обработке ввода пользователем каких-либо данных и позволяют находить ошибки, связанные с неверной обработкой нестандартных значений [13, 14, 15, 16]. В данном проекте имеется 3 формы, обрабатывающие пользовательский ввод:

- форма обработки авторизации;
- форма ввода сообщения;
- форма создания события в календаре.

Для всех этих форм используются встроенные поля ввода, которые сами осуществляют обработку неверного ввода, либо предоставляют значения по умолчанию [10], что позволяет полностью исключить возможность ввода неверных значений, исходя из этого применение приведенным методов функционального тестирования является нецелесообразным, к тому же реализация данных методов тестирования требует намного больше трудозатрат, чем обычное ручное тестирование, а также совсем не приспособлено к возможным изменениям в приложении. Однако в любом случае необходимо провести тестирование приложения, поэтому в качестве функционального тестирования было выбрано ручное тестирование, а для решения проблемы с изменениями в приложении будем использовать

модульное тестирование. Также будет проведено структурное тестирование функции добавления события в календарь.

4.2 Функциональное тестирование

Сформируем план тестирования на основе вариантов использования и представим результаты в таблице 17.

Таблица 17 – План тестирования

Вариант использования	Тест	Результаты
Войти в приложение	Вход в приложение с логином «admin» и паролем «123»	Тест пройден. Пользователь переведен на окно с чатом (рисунок Б.1)
Отправить текстовое сообщение в чат	Отправка сообщения в чат с текстом «Тестовое сообщение»	Тест пройден. Сообщение отправилось в чат и появилось справа как исходящее и слева как входящее у других пользователей (рисунок Б.2)
Установить фотографию профиля	Установка пользовательской фотографии профиля	Тест пройден. У пользователя установлена его фотография, у других пользователей она отображается в чате (рисунок Б.2 и Б.3)
Отправить аудиосообщение в чат	Запись и отправка аудиосообщения в чат	Тест пройден. Сообщение записано и отправлено в чат (рисунок Б.4)
Прослушать аудиосообщение	Прослушивание аудиосообщения и проверка корректного отображения времени проигрываемого сообщения	Тест пройден. При проигрывании слышно записанное ранее сообщение и корректно отображается время (рисунок Б.5)

Продолжение таблицы 17

Просмотреть текстовый перевод аудиосообщения	Развертывание скрытого текстового описания аудиосообщения	Тест пройден. Система успешно перевела аудиосообщение в тест и отображает его в чате (рисунок Б.6)
Добавить файл в хранилище	Добавление картинки в файловое хранилище	Тест пройден. Картинка успешно добавлена в хранилище (рисунок Б.7)
Добавить событие в календарь	Добавление повторяемого события в календарь	Тест пройден. Событие успешно добавлено и отображается каждый день в календаре. В чат добавлено сообщение о новом событии (рисунки Б.8, Б.9)
Сделать сообщение избранным	Пометка сообщения в чате как избранного	Тест пройден. К сообщению в чате добавлена звездочка, что означает его отображение как избранного (рисунок Б.10)
Выбрать только избранные сообщения	Установка фильтра сообщений на избранные	Тест пройден. В чате показаны только избранные сообщения (рисунок Б.11)

Приведенные тесты демонстрируют полное соответствие функциональным требованиям и работоспособность приложения. Все рисунки с результатами приведены в приложении Б.

4.3 Структурное тестирование

Проведём тестирование функции добавления события в базу данных (листинг 1) методом белого ящика [17], чтобы учесть внутренние механизмы функции. При тестировании рассчитаем некоторые критерии покрытия кода, такие как покрытие операторов, покрытие решений и покрытие маршрутов.

Листинг 1 – Функция добавления события в календарь

```

@IBAction func showAlert() {

    let alertController = UIAlertController(title:
"OnCellSelection", message: "Button Row Action", preferredStyle:
.alert)
    let defaultAction = UIAlertAction(title: "OK", style:
.default, handler: nil)
    alertController.addAction(defaultAction)
    //present(alertController, animated: true)

    let startRow: DateTimeInlineRow! = self.form.rowBy(tag:
"Starts")
    let endRow: DateTimeInlineRow! = self.form.rowBy(tag: "Ends")
    let eventName: TextRow! = self.form.rowBy(tag: "EventName")

    let cycle: SwitchRow! = self.form.rowBy(tag: "switchRowTag")

    var selectedCycle = ""

    if (cycle.value > 0){

        let cycleOption: SelectableSection<ListCheckRow<String>>!
= (self.form.sectionBy(tag: "cycleOptions") as!
SelectableSection<ListCheckRow<String>>))

        selectedCycle =
cycleOption.selectedRow()!.selectableValue!
    }

    if let eventNameValue = eventName.value {

        let newEvent = UserEvent()
        newEvent.startDate = startRow.value!
        newEvent.endDate = endRow.value!
        newEvent.text =
"\ (eventNameValue)\n\ (newEvent.startDate.format(with: "HH:mm")) -
\ (newEvent.endDate.format(with: "HH:mm")) "
        newEvent.isCycle = cycle.value!

        switch selectedCycle {
            case "day":
                newEvent.cycleType =
UserEvent.CycleType.day.rawValue
            case "week":
                newEvent.cycleType =
UserEvent.CycleType.week.rawValue
            case "month":
                newEvent.cycleType =
UserEvent.CycleType.month.rawValue
            case "year":
                newEvent.cycleType =
UserEvent.CycleType.year.rawValue
            default:
                newEvent.cycleType =
UserEvent.CycleType.noCycle.rawValue
        }
    }
}

```

```

        let message = UserMessage()
        message.messageId = UUID().uuidString
        message.user = realm.objects(User.self).filter("senderId =
'Notifier'")[0]
        message.sentDate = Date()
        message.realmKind = "text"
        message.value = "Событие было добавлено в календарь:
\\(newEvent.text) \\(newEvent.startDate.format(with: "dd/MM/YYYY"))"

        try! realm.write {
            realm.add(message)
        }

        try! realm.write {
            realm.add(newEvent)
        }

        self.navigationController?.popViewController(animated:
true)
    } else {
        let alertController = UIAlertController(title: "Ошибка",
message: "Введите название события", preferredStyle: .alert)
        let defaultAction = UIAlertAction(title: "OK", style:
.default, handler: nil)
        alertController.addAction(defaultAction)
        present(alertController, animated: true)
    }
}

```

На рисунке ниже представлена блок-схема для листинга 1.

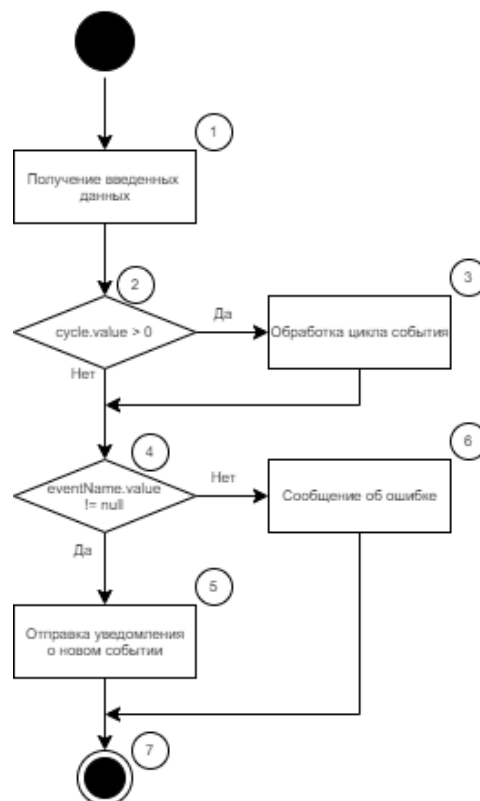


Рисунок 35 – Блок-схема программного кода

Согласно блок-схеме, создадим управляющий граф алгоритма (рисунок 36).

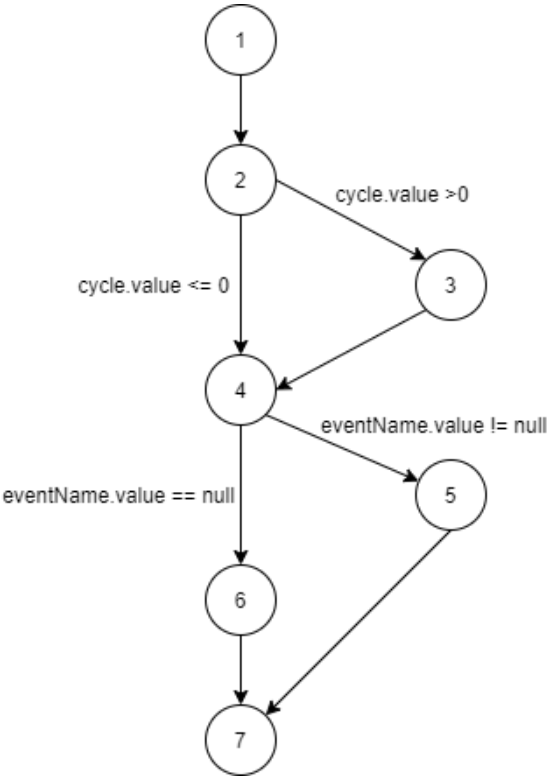


Рисунок 36 – Управляющий граф

Построим тесты проверки покрытия операторов, решений и маршрутов на основании управляющего графа и блок-схемы.

Пусть «A = cycle.value» и «B = eventName.value». Результаты приведены в таблицах 18 – 20.

Таблица 18 – Покрытие операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A > 0 B == null	Сообщение об ошибке	Сообщение об ошибке	Успешно
A > 0 B != null	Данные добавлены в БД	Данные добавлены в БД	Успешно

Таблица 19 – Покрытие решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A > 0 B == null	Сообщение об ошибке	Сообщение об ошибке	Успешно
A > 0 B != null	Данные добавлены в БД	Данные добавлены в БД	Успешно
A < 0 B != null	Данные добавлены в БД	Данные добавлены в БД	Успешно

Таблица 20 – Покрытие маршрутов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A > 0 B == null	Сообщение об ошибке	Сообщение об ошибке	Успешно
A > 0 B != null	Данные добавлены в БД	Данные добавлены в БД	Успешно
A < 0 B != null	Данные добавлены в БД	Данные добавлены в БД	Успешно
A < 0 B == null	Сообщение об ошибке	Сообщение об ошибке	Успешно

Таким образом для покрытия всех операторов необходимо провести всего 2 теста, для покрытия всех решений 3 теста, а для покрытия всех маршрутов 4 теста. Используя эти данные, можно эффективнее создавать автоматические тесты.

4.4 Автоматизированное тестирование

Автоматизированное тестирование позволяет при активной разработке быстро находить участки кода, вызывающее ошибки. С помощью модульных тестов можно автоматически проверять работоспособность классов модели, представлений и контроллеров при внесении любых малейших изменений.

Протестируем некоторые классы модели с помощью unit-тестирования [18], а также создадим автоматические тесты для нескольких важных элементов интерфейса с помощью ui-тестирования [19] в среде Xcode.

4.4.1 Unit-тестирование

Создадим тесты для проверки работы функций конвертации числового значения типа пользователя или события в тип перечисления для классов `User` и `UserEvent`, а также тест для проверки работы класса модели для работы с файлами `FileStatusData`. Класс с написанными тестами приведен в листинге 2.

Листинг 2 – Класс unit-тестов

```
import XCTest
@testable import StudentChatRealm

class StudentChatRealmTests: XCTestCase {

    func testUserTypeConvert() throws {
        let user = User()
        user.type = 1
        user.displayName = "Test"
        user.password = "123"
        user.senderId = "1"

        XCTAssertTrue(user.userType == .Admin)
    }

    func testWrongUserTypeConvert() throws {
        let user = User()
        user.type = -2
        user.displayName = "Test"
        user.password = "123"
        user.senderId = "1"

        XCTAssertTrue(user.userType == .User)
    }

    func testEventCycleConvert() throws {
        let event = UserEvent()
        event.isCycle = true
        event.cycleType = 2

        XCTAssertTrue(event.cycle == .month)
    }

    func testWrongEventCycleConvert() throws {
        let event = UserEvent()
        event.isCycle = true
        event.cycleType = -12

        XCTAssertTrue(event.cycle == .noCycle)
    }

    func testFileInfo() throws {
        let fileInfo = FileStatusData(fbPath: "default.png")

        XCTAssertTrue(fileInfo.localPath == nil)
    }
}
```

```
XCTAssertTrue(fileInfo.state == .startDownload)
}
}
```

Для тестирования класса User были написаны функции testUserTypeConvert и testWrongUserTypeConvert, которые проверяют конвертацию числа в тип перечисления. В первом случае проверяется число 1, которое обозначает тип администратора, а во втором случае проверяется число минус 2, которое не определено в перечислении.

Для класса UserEvent были написаны аналогичные тесты.

Для класса FileStatusData проводится проверка статуса файла, не находящегося на устройстве, но присутствующего в облачном хранилище Firebase.

Результаты тестов приведены на рисунке ниже.

```
Test Suite 'Selected tests' started at 2020-05-12 19:41:23.919
Test Suite 'StudentChatRealmTests.xctest' started at 2020-05-12 19:41:23.920
Test Suite 'StudentChatRealmTests' started at 2020-05-12 19:41:23.920
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testEventCycleConvert]' started.
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testEventCycleConvert]' passed (0.001 seconds).
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testFileInfo]' started.
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testFileInfo]' passed (0.001 seconds).
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testUserTypeConvert]' started.
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testUserTypeConvert]' passed (0.000 seconds).
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testWrongEventCycleConvert]' started.
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testWrongEventCycleConvert]' passed (0.000 seconds).
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testWrongUserTypeConvert]' started.
Test Case '-[StudentChatRealmTests.StudentChatRealmTests testWrongUserTypeConvert]' passed (0.000 seconds).
Test Suite 'StudentChatRealmTests' passed at 2020-05-12 19:41:23.923.
Executed 5 tests, with 0 failures (0 unexpected) in 0.002 (0.003) seconds
Test Suite 'StudentChatRealmTests.xctest' passed at 2020-05-12 19:41:23.923.
Executed 5 tests, with 0 failures (0 unexpected) in 0.002 (0.004) seconds
Test Suite 'Selected tests' passed at 2020-05-12 19:41:23.924.
Executed 5 tests, with 0 failures (0 unexpected) in 0.002 (0.005) seconds
```

Рисунок 37 – Результаты тестирования

Все тесты пройдены без ошибок.

4.4.2 UI-тестирование

Тестирование интерфейса позволяет в автоматическом режиме проверить правильность изменения элементов интерфейса в зависимости от действий пользователя.

Сами тесты можно задать как интерактивно, взаимодействуя с эмулятором, на котором запущено приложение, так и программно [20]. Все действия в эмуляторе преобразуются в код, однако данная система не всегда

корректно обрабатывает желаемые действия, поэтому зададим тесты вручную. Полученные тесты приведены в листинге 3.

Листинг 3 – Класс, выполняющий тестирование интерфейса

```
import XCTest

class StudentChatRealmUITests: XCTestCase {

    var app: XCUIApplication!

    override func setUp() {
        super.setUp()
        continueAfterFailure = false
        app = XCUIApplication()
        app.launchArguments.append("--uitesting")
    }

    func testSendMessage() {
        app.launch()
        XCTAssertTrue(app.otherElements["chatView"].exists)
        let sendAudioBtn = app.buttons["sendAudio"].firstMatch
        XCTAssertTrue(sendAudioBtn.exists)

        let inputElem = app.textViews["input"].firstMatch
        inputElem.tap()
        inputElem.typeText("test msg")

        XCTAssertTrue(inputElem.value as! String == "test msg")
        XCTAssertTrue(!sendAudioBtn.exists)

        let sendTextBtn = app.buttons["sendText"].firstMatch

        XCTAssertTrue(sendTextBtn.exists)

        let tableChat = app.tables["chat"].firstMatch
        let tableCount = tableChat.cells.count

        sendTextBtn.tap()

        XCTAssertTrue(tableChat.cells.count > tableCount)
        XCTAssertTrue(sendTextBtn.exists)

        tableChat.tap()

        XCTAssertTrue(!sendTextBtn.exists)
        XCTAssertTrue(sendAudioBtn.exists)
    }

    func testSwipeDocumentView() {
        app.launch()
        XCTAssertTrue(app.otherElements["chatView"].exists)

        app.swipeRight()
        XCTAssertTrue(app.otherElements["documentView"].waitForExistence(timeout: 3))
        XCTAssertTrue(!app.otherElements["chatView"].exists)
    }
}
```

```

func testSwipeCalendarView() {
app.launch()

XCTAssertTrue(app.otherElements["chatView"].exists)

app.swipeLeft()

XCTAssertTrue(app.otherElements["calendarView"].waitForExistence(timeout: 3))
XCTAssertTrue(!app.otherElements["chatView"].exists)
}
func testLaunchPerformance() {
if #available(macOS 10.15, iOS 13.0, tvOS 13.0, *) {
// This measures how long it takes to launch your application.
measure(metrics: [XCTOSSignpostMetric.applicationLaunch]) {
XCUIApplication().launch()
}
}
}
}
}

```

Функция `testSendMessage` тестирует реакции интерфейса на отправку сообщения и выполняет проверки существования окна чата, существования кнопки отправки аудиосообщения, изменения кнопки отправки на текстовое сообщение при нажатии на клавиатуру, изменения количества ячеек в таблице чата при отправке сообщения и обратное изменение кнопки отправки текстового сообщения на аудиосообщение.

Функции `testSwipeDocumentView` и `testSwipeCalendarView` проверяют переходы из чата пользователей путем свайпа вправо и влево в окно файлового хранилища и календаря соответственно.

Результаты тестирования на рисунке 38.

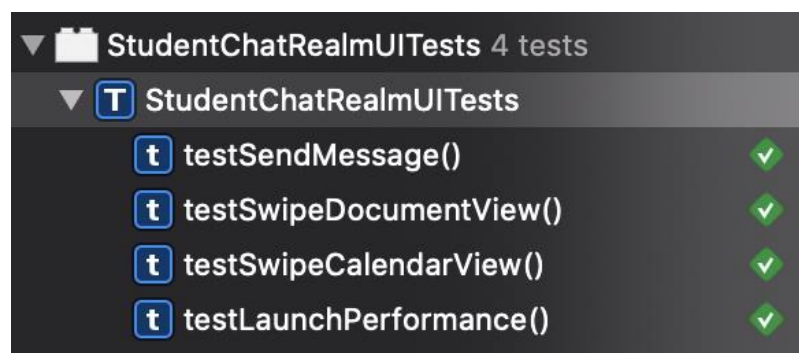


Рисунок 38 – Результаты тестирования интерфейса

Все тесты успешно прошли проверку.

4.5 Метрики кода

Различные метрики кода позволяют сделать вывод о сложности и качестве программного кода, однако расчет данных метрик вручную для каждой функции является очень трудоемким процессом, а полученные числовые значения не всегда могут отображать реальную сложность программы, поскольку могут не учитывать особенности языка программирования Swift [8], так как некоторые метрики были разработаны еще в 70х годах.

Однако можно воспользоваться условно бесплатным сервисом Codebeat, анализирующим программный код, выложенный на GitHub. Данный сервис может рассчитать сложность кода для программного продукта и предоставить информацию для каждой функции программы. На рисунке ниже показана оценка проекта.

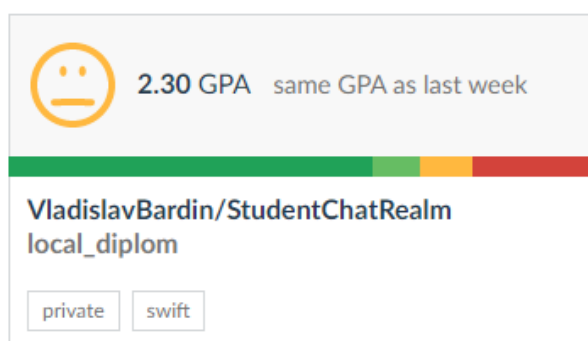


Рисунок 39 – Оценка проекта в сервисе Codebeat

На рисунке зелеными линиями показан процент классов с низкой сложностью и малым количеством проблем в коде, а желтым и красным более проблемные компоненты. Оценка GPA 2.30 из 4.00, что является средним показателем.

Наиболее сложным классом по оценке Codebeat является класс NewChatViewController, который является самым объемным классом, отвечающим за чат пользователей. На рисунке ниже приведен отчет по данному классу.

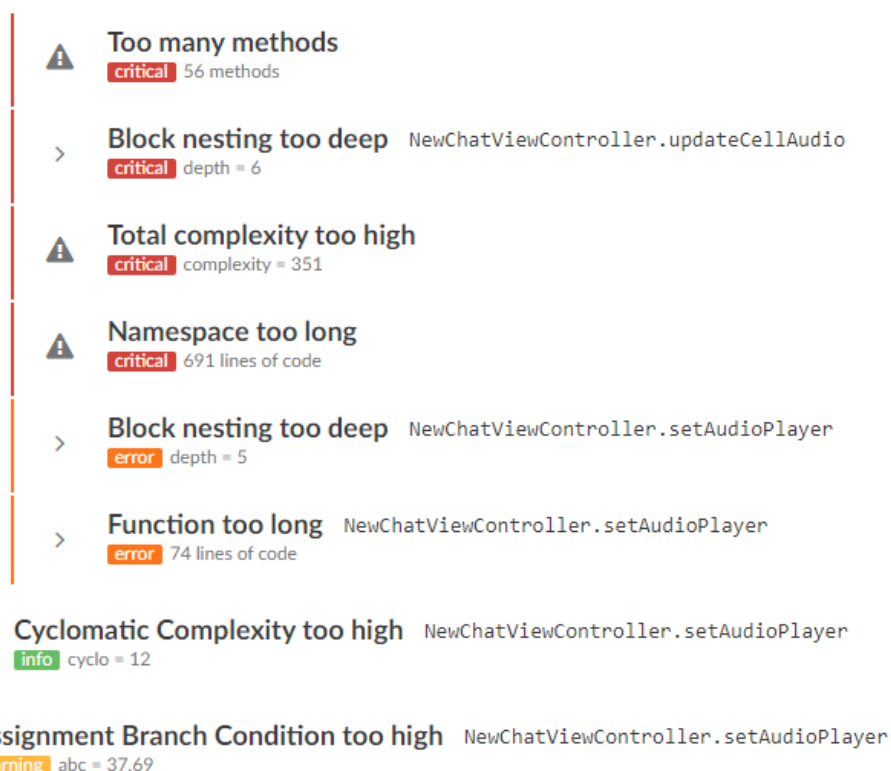


Рисунок 40 – Отчет сложности по классу `NewChatViewController`

Основными ошибками является слишком большое количество методов (56 методов), слишком большая вложенность для функций `updateCellAudio` и `setAudioPlayer` (максимальный уровень вложенности 6), большая общая сложность, много строк кода (691 строка) и длинная функция `setAudioPlayer` (74 строки). Также одна из ошибок – высокая цикломатическая сложность функции `setAudioPlayer`. На рисунке 40 видно, что цикломатическая сложность данной функции равна 12. Посчитаем данную метрику вручную, чтобы убедиться в правильности работы сервиса.

Поскольку функция длинная, ее листинг приведен в приложении А. Так как функция имеет одну точку входа и одну точку выхода, то цикломатическая сложность эквивалентна числу точек ветвления [21], т.е. операторов `if`. Исходя из этого, цикломатическая сложность данного программного кода равна 12.

В отчете также рассчитана метрика ABC для той же функции установки плеера для проигрывания аудиосообщения и указана ее сложность 37,69. ABC метрика основана на вычислении количества присвоений (A), вызовов

функций (В) и логических проверок (С) [22]. Просчитаем вручную данную метрику для функции `setAudioPlayer`.

Таким образом:

$$A = 11$$

$$B = 34$$

$$C = 12$$

Для оценки сложности программы необходимо вычислить квадратный корень из суммы квадратов А, В, и С. Вычислив, получаем оценку сложности функции равную 37,69. Воспользовавшись методическими рекомендациями разработчиков ПО [23], реализующего подсчет ABC метрики для программного кода, полученное значение метрики позволяет сделать вывод о достаточно высокой сложности функции, что требует дальнейшего ее упрощения и рефакторинга [24]. По приведенной шкале идеальное значение лежит в промежутке от 0 до 10, хорошее значение от 11 до 20, требующее рефакторинг от 21 до 40, от 41 до 60 может быть оправдано, а все что выше 61 говорит о серьезных проблемах в программном коде, которые должны быть обязательно устранены. Автор шкалы в отдельной заметке отмечает, что сложность кода в промежутке от 20 до 60 может быть, как и обоснована, так и означать, что метод срочно нуждается в упрощении [25].

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильно приложение для устройств под управлением операционной системы iOS для поддержки взаимодействия членов творческого коллектива.

Была поставлена задача и проведен анализ предметной области. Были проанализированы программы аналогичного назначения и по полученным результатам были разработаны и проанализированы функциональные и нефункциональные требования к проектируемому приложению.

Был произведен и обоснован выбор технологий разработки, было проведено планирование разработки и оценка бюджета, по результатам которых получили, что длительность реализации проекта составляет 28 дней, а затраты по проекту составят 60 750 рублей. Также был спроектирован пользовательский интерфейс и архитектура приложения.

Для реализации мобильного приложения использовался язык программирования Swift 5 и среда программирования Xcode.

Был выбран и обоснован режим тестирования приложения. Было проведено ручное функциональное тестирование, показавшее корректную работу и полное соответствие приложения функциональным требованиям. Также было проведено структурное тестирование и автоматизированное тестирование классов модели приложения и интерфейса средствами среды разработки Xcode. Автоматически была рассчитана сложность кода с помощью сервиса Codebeat, а результаты некоторых метрик были проверены вручную.

Таким образом, результатом выполнения выпускной квалификационной работы является разработанное мобильное приложение, отвечающее поставленной задаче и требованиям.

Список использованных источников

1. Статья «UX/UI-дизайн: что это такое?» // Онлайн-университет Skillbox [Электронный ресурс]. URL: https://skillbox.ru/media/design/ux_ui_dizayn_chno_eto_takoe/ (дата обращения: 20.05.2020).
2. Статья о методе анализа иерархий // Сайт по теме теории и практики оценки экономических величин [Электронный ресурс]. URL: <http://vamocenka.ru/metod-analiza-ierarxij-procedura-primeneniya/> (дата обращения: 19.05.2020).
3. Статья на Wikipedia о ВКонтакте // Электронная энциклопедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/ВКонтакте> (дата обращения: 13.05.2020).
4. Исследование аудитории Telegram за 2019 год» // Сайт с аналитикой аудитории и каналов в Telegram [Электронный ресурс]. URL: <https://tgstat.ru/research> (дата обращения: 13.05.2020).
5. Статья сравнения языков программирования Swift и Objective-C // Образовательный портал GeekBrains [Электронный ресурс]. URL: https://geekbrains.ru/posts/swift_vs_obj_c (дата обращения: 13.05.2020).
6. Статья «База данных Realm для iOS» // Образовательный портал СвифтЛаб [Электронный ресурс]. URL: <https://swiftlab.ru/2018/12/21/realm-for-ios/> (дата обращения: 13.05.2020).
7. Статья «Начало работы с Realm Mobile Platform для iOS» // Информационный ресурс [Электронный ресурс]. URL: <https://code.tutsplus.com/ru/tutorials/getting-started-with-realm-mobile-platform-for-ios--cms-29080> (дата обращения: 13.05.2020).
8. Статья о планировании стоимости в MS Project // Информационный ресурс [Электронный ресурс]. URL: <http://www.taurion.ru/project/13/2> (дата обращения: 13.05.2020).

9. Статья на Wikipedia о приложении Interface Builder // Электронная энциклопедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Interface_Builder (дата обращения: 13.05.2020).
10. Усов В. Swift. Основы разработки приложений под iOS, iPadOS и macOS / Усов В., Тульцева К. – СПб.: Питер, 2020. – 496 с.
11. Целевая страница об «Яндекс.Мессенджер» // Сайт с информацией о мессенджере [Электронный ресурс]. URL: <https://yandex.ru/messenger> (дата обращения: 07.06.2020).
12. Целевая страница об «icq new» // Сайт с информацией о мессенджере [Электронный ресурс]. URL: <https://icq.com/> (дата обращения: 07.06.2020).
13. Статья о технике анализа классов эквивалентности // Сайт с записями о тестировании программных продуктов и методах тестирования [Электронный ресурс]. URL: <https://qaevolution.ru/testovaya-dokumentaciya/test-dizajn/tehnika-analiza-klassov-ekvivalentnosti/> (дата обращения: 09.06.2020).
14. Статья о технике анализа граничных значений // Сайт с записями о тестировании программных продуктов и методах тестирования [Электронный ресурс]. URL: <https://qaevolution.ru/testovaya-dokumentaciya/test-dizajn/tehnika-analiza-granichnyx-znachenij/> (дата обращения: 09.06.2020).
15. Статья о методе функциональных диаграмм // Сайт с общедоступной информацией для студентов разных предметных областей [Электронный ресурс]. URL: https://studopedia.ru/8_201409_metod-funktsionalnih-diagramm.html (дата обращения: 09.06.2020).
16. Перевод статьи о парном тестировании // Сайт со статьями про тестирование [Электронный ресурс]. URL: <https://www.software-testing.ru/library/around-testing/processes/2286-pair-testing> (дата обращения: 09.06.2020).

17. Статья о тестирование методом белого ящика // Центр подготовки IT-специалистов QALight [Электронный ресурс]. URL: <https://qalight.com.ua/baza-znaniy/white-black-grey-box-testirovanie/> (дата обращения: 09.06.2020).

18. Статья на Wikipedia о модульном тестировании // Электронная энциклопедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Модульное_тестирование (дата обращения: 09.06.2020).

19. Статья «Тестирование UI (пользовательского интерфейса)» // Сайт компании WOXAPP, занимающейся разработкой мобильных приложений [Электронный ресурс]. URL: <https://woxapp.com/ru/our-blog/testing-the-ui-user-interface/> (дата обращения: 09.06.2020).

20. Справка по unit и ui тестировании в iOS // Сайт сообщества iOS разработчиков на языке Swift [Электронный ресурс]. URL: <https://swiftbook.ru/post/tutorials/unit-testing-ui-testing-tutorial/> (дата обращения: 09.06.2020).

21. Статья на Wikipedia о цикломатической сложности // Электронная энциклопедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Цикломатическая_сложность (дата обращения: 09.06.2020).

22. Статья на Wikipedia о ABC метрике сложности кода // Электронная энциклопедия [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/ABC_Software_Metric (дата обращения: 09.06.2020).

23. Заметка о расчете ABC метрики и её интерпретации // Информационная страница проекта на github, реализующего подсчет ABC метрики [Электронный ресурс]. URL: <https://dx42.github.io/gmetrics/metrics/AbcMetric.html> (дата обращения: 09.06.2020).

24. Статья на Wikipedia о рефакторинге // Электронная энциклопедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Рефакторинг> (дата обращения: 09.06.2020).

25. Заметка об интерпретации результата метрики ABC // Сайт с блогом пользователей [Электронный ресурс]. URL: <http://jakescruggs.blogspot.com/2008/08/whats-good-flog-score.html> (дата обращения: 09.06.2020).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг A.1 – ChatViewController.swift

```
import Foundation
import UIKit
import RealmSwift
import AVFoundation
import Firebase
import Speech
import BusyNavigationBar
import DOFavoriteButtonNew
import Hero

class NewChatViewController: UIViewController,
    UITableViewDataSource,
    UITableViewDelegate,
    UITextViewDelegate,
    AVAudioRecorderDelegate,
    AVAudioPlayerDelegate {

    @IBOutlet var tableView: UITableView!
    @IBOutlet var inputTextView: UITextView!

    @IBOutlet var keyboardHeightLayoutConstraint: NSLayoutConstraint!

    var notificationToken: NotificationToken? = nil

    var messageList : Results<UserMessage>?

    @IBOutlet var sendButton: UIButton!
    var recordingSession: AVAudioSession!
    var audioRecorder: AVAudioRecorder!
    var audioPlayer: AVAudioPlayer!

    override func viewDidLoad() {
        super.viewDidLoad()

        view.accessibilityIdentifier = "chatView"

        setupAccessibilityIdentifier()

        setupKeyboard()

        setupMessagesList()

        stupInputTextView()

        setupTableView()

        setupNavBar()

        setupRecordButton()

        requestTranscribePermissions()

        setupSwipeGesture()
    }

    deinit {
        NotificationCenter.default.removeObserver(self)
    }
}
```

```

        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)
            navigationItem.hidesBackButton = true
            navigationController?.setNavigationBarHidden(false, animated:
animated)
        }

        override func viewDidAppear(_ animated: Bool) {
            super.viewDidAppear(animated)
        }

        func setupAccessibilityIdentifier(){
            inputTextView.accessibilityIdentifier = "input"
            tableView.accessibilityIdentifier = "chat"
            sendButton.accessibilityIdentifier = "sendAudio"
        }

        // MARK: Swipe

        func setupSwipeGesture(){
            let swipeRight: UISwipeGestureRecognizer =
UISwipeGestureRecognizer(target: self, action: #selector(swipeRightHandler))
            swipeRight.direction = .right
            self.view.addGestureRecognizer(swipeRight)

            let swipeLeft: UISwipeGestureRecognizer =
UISwipeGestureRecognizer(target: self, action: #selector(swipeLeftHandler))
            swipeLeft.direction = .left
            self.view.addGestureRecognizer(swipeLeft)
        }

        @objc
        func swipeRightHandler(){
            self.navigationController?.hero.navigationAnimationType =
.slide(direction: .right)
            self.performSegue(withIdentifier: "documentViewSegue", sender: nil)
        }

        @objc
        func swipeLeftHandler(){
            self.navigationController?.hero.navigationAnimationType =
.slide(direction: .left)
            self.performSegue(withIdentifier: "calendarViewSegue", sender: nil)
        }

        // MARK: Keyboard

        func setupKeyboard(){
            NotificationCenter.default.addObserver(self,
                                                    selector:
#selector(self.keyboardNotification(notification:)),
                                                    name:
UIResponder.keyboardWillChangeFrameNotification,
                                                    object: nil)

            let tap: UITapGestureRecognizer = UITapGestureRecognizer(target:
self, action: #selector(UIViewController.dismissKeyboard))
            tap.cancelsTouchesInView = false
            tableView.addGestureRecognizer(tap)

```

```

    }

    @objc func keyboardNotification(notification: NSNotification) {
        if let userInfo = notification.userInfo {
            let endFrame = (userInfo[UIResponder.keyboardFrameEndUserInfoKey]
as? NSValue)?.cgRectValue
            let endFrameY = endFrame?.origin.y ?? 0
            let duration:TimeInterval =
(userInfo[UIResponder.keyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
            let animationCurveRawNSN =
userInfo[UIResponder.keyboardAnimationCurveUserInfoKey] as? NSNumber
            let animationCurveRaw = animationCurveRawNSN?.intValue ??
UIView.AnimationOptions.curveEaseInOut.rawValue
            let animationCurve:UIView.AnimationOptions =
UIView.AnimationOptions(rawValue: animationCurveRaw)
            if endFrameY >= UIScreen.main.bounds.size.height {
                self.keyboardHeightLayoutConstraint?.constant = 0.0
            } else {
                if let size = endFrame?.size.height {
                    self.keyboardHeightLayoutConstraint?.constant = -size +
view.safeAreaInsets.bottom
                } else {
                    self.keyboardHeightLayoutConstraint?.constant = 0.0
                }
            }

            UIView.animate(withDuration: duration,
                           delay: TimeInterval(0),
                           options: animationCurve,
                           animations: { self.view.layoutIfNeeded()}) { (res) in
                self.tableView.scrollToLast()
            }
        }
    }

    // MARK: MessageList

    func setupMessagesList(){
        messageList = realm.objects(UserMessage.self)

        notificationToken = messageList!.observe { [weak self] (changes:
RealmCollectionChange) in
            self!.tableView.reloadData()
            self!.tableView.scrollToLast()
        }

        self.tableView.scrollToLast()
    }

    @discardableResult
    func handleSend(value: String, type: UserMessage.MessageType, needUpdate:
Bool = true) -> UserMessage{

        if (needUpdate){
            DispatchQueue.global(qos: .background).async {
                DispatchQueue.main.async {
                    self.inputTextView.text = ""
                }
            }
        }
    }

```

```

    }
}

let message = UserMessage()
message.messageId = UUID().uuidString
message.user = currentUser!
message.sentDate = Date()
message.realmKind = type.rawValue
message.value = value

try! realm.write {
    realm.add(message)
}

return message
}

// MARK: InputTextView

func setupInputTextView(){
    inputTextView.delegate = self
    inputTextView.textContainerInset = UIEdgeInsets(top: 6, left: 8,
bottom: 6, right: 16)
    inputTextView.text = "Сообщение..."
    inputTextView.textColor = UIColor.lightGray
}

func textViewDidEndEditing(_ textView: UITextView) {
    if inputTextView.text.isEmpty {
        inputTextView.text = "Сообщение..."
        inputTextView.textColor = UIColor.lightGray

        if let image = UIImage(named: "mic") {

            sendButton.accessibilityIdentifier = "sendAudio"

            UIView.animate(withDuration: 0.1,
animations: {
                self.sendButton.transform = CGAffineTransform(scaleX:
0.6, y: 0.6)

                self.sendButton.alpha = 0.0
            },
completion: { _ in
                UIView.animate(withDuration: 0.1) {
                    self.sendButton.setImage(image, for: .normal)
                    self.sendButton.alpha = 1.0
                    self.sendButton.transform =
CGAffineTransform.identity
                }
            })

            setupSendAudioMsg()
        }
    }
}

func textView(_ textView: UITextView, shouldChangeTextIn range: NSRange,
replacementText text: String) -> Bool {
    if (text == "\n") {

```

```

        textView.resignFirstResponder()
        handleSend(value: inputTextView.text, type: .text)
    }
    return true
}

func textViewDidBeginEditing(_ textView: UITextView) {
    if inputTextView.textColor == UIColor.lightGray {
        inputTextView.text = nil
        inputTextView.textColor = UIColor.black

        if let image = UIImage(named: "send") {

            sendButton.accessibilityIdentifier = "sendText"

            UIView.animate(withDuration: 0.1,
                animations: {
                    self.sendButton.transform = CGAffineTransform(scaleX:
0.6, y: 0.6)

                    self.sendButton.alpha = 0.0
                },
                completion: { _ in
                    UIView.animate(withDuration: 0.1) {
                        self.sendButton.setImage(image, for: .normal)
                        self.sendButton.alpha = 1.0
                        self.sendButton.transform =
CGAffineTransform.identity
                    }
                })

            setupSendTextMsg()

        }
    }
}

// MARK: TableView

func setupTableView() {
    tableView.dataSource = self
    tableView.delegate = self
    tableView.separatorStyle = .none

    tableView.contentInset.bottom = 8
    tableView.contentInset.top = 8

    setupTableViewCell()
}

func setupTableViewCell() {
    let incomingCell = UINib(nibName: "incomingCell", bundle: nil)
    tableView.register(incomingCell, forCellReuseIdentifier:
"incomingCell")

    let outgoingCell = UINib(nibName: "outgoingCell", bundle: nil)
    tableView.register(outgoingCell, forCellReuseIdentifier:
"outgoingCell")

    let outgoingAudioCell = UINib(nibName: "outgoingAudioCell", bundle:
nil)

```

```

        tableView.register(outgoingAudioCell, forCellReuseIdentifier:
"outgoingAudioCell")

        let incomingAudioCell = UINib(nibName: "incomingAudioCell", bundle:
nil)
        tableView.register(incomingAudioCell, forCellReuseIdentifier:
"incomingAudioCell")
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return messageList?.count ?? 0
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let message = messageList![indexPath.row]

        var isPrevUser = false

        if indexPath.row == 0 {
            isPrevUser = false
        } else {
            if messageList![indexPath.row - 1].sender.senderId ==
message.sender.senderId {
                isPrevUser = true
            }
        }

        var isLast = false

        if messageList?.count > indexPath.row + 1 {
            if messageList?[indexPath.row + 1].sender.senderId !=
messageList?[indexPath.row].sender.senderId {
                isLast = true
            }
        } else {
            isLast = true
        }

        if message.sender.senderId == currentUser?.senderId {

            switch message.type {
                case .text:
                    return makeOutgoingTextCell(message: message, isPrevUser:
isPrevUser, isLast: isLast, indexPath: indexPath)
                case .audio:
                    return makeOutgoingAudioCell(message: message,
isPrevUser: isPrevUser, isLast: isLast, indexPath: indexPath)
            }
        } else {
            switch message.type {
                case .text:
                    return makeIncomingTextCell(message: message, isPrevUser:
isPrevUser, isLast: isLast, indexPath: indexPath)
                case .audio:
                    return makeIncomingAudioCell(message: message,
isPrevUser: isPrevUser, isLast: isLast, indexPath: indexPath)
            }
        }
    }

```

```

}

func makeDateRepresentation(sentDate: Date) -> String{
    let currentDate = Date()

    var result: String = ""

    if (currentDate.isSameDay(date: sentDate)){
        result = sentDate.format(with: "HH:mm")
    } else {
        if (currentDate.months(from: sentDate) == 0){
            if (currentDate.weeks(from: sentDate) == 0){
                result = sentDate.format(with: "E, HH:mm")
            } else {
                result = sentDate.format(with: "MMM d, HH:mm")
            }
        } else {
            if (currentDate.years(from: sentDate) == 0){
                result = sentDate.format(with: "MMMM d")
            } else {
                result = sentDate.format(with: "MMM yyyy")
            }
        }
    }

    return result
}

func makeOutgoingTextCell(message: UserMessage, isPrevUser: Bool, isLast:
Bool, indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"outgoingCell", for: indexPath) as! OutgoingChatCell

    cell.dateLabel.text = makeDateRepresentation(sentDate:
message.sentDate)

    var fontSize: CGFloat = 16

    if message.value.containsOnlyEmoji {
        cell.message?.backgroundColor = .clear
        cell.message.textContainerInset = UIEdgeInsets(top: 8, left: 0,
bottom: 8, right: 4)
        fontSize = 36
    } else {
        cell.message.backgroundColor = softOrangeColor
        cell.message.textContainerInset = UIEdgeInsets(top: 8, left: 8,
bottom: 8, right: 4)
    }

    cell.message.font = UIFont.systemFont(ofSize: fontSize)

    let rect = message.value.estimateFrameForText(fontSize: fontSize,
width: (tableView.frame.width - 32)/1.5)

    cell.messageViewHeight.constant = rect.height + 24
    cell.messageWidth.constant = rect.width + 10 + 8 + 8

    cell.message.text = message.value

    if message.isFav {

```

```

        cell.favImg.isHidden = false
    } else {
        cell.favImg.isHidden = true
    }

    if isLast {
        cell.hideDate(hide: false)
    } else {
        cell.hideDate(hide: true)
    }

    return cell
}

func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
shouldReceive touch: UITouch) -> Bool {
    return touch.view == gestureRecognizer.view
}

var transitionUser: User?

func makeIncomingTextCell(message: UserMessage, isPrevUser: Bool, isLast:
Bool, indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"incomingCell", for: indexPath) as! IncomingChatCell

    cell.imgClickFunc = {

        let viewController: UserCabinetViewController =
UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: "userInfo") as!
UserCabinetViewController

        viewController.user = message.user

        self.navigationController?.pushViewController(viewController,
animated: true)
    }

    var fontSize: CGFloat = 16

    if message.value.containsOnlyEmoji {
        cell.messageField?.backgroundColor = .clear
        cell.messageField.textContainerInset = UIEdgeInsets(top: 8, left:
0, bottom: 8, right: 4)
        fontSize = 36
    } else {
        cell.messageField?.backgroundColor = softBlueColor
        cell.messageField.textContainerInset = UIEdgeInsets(top: 8, left:
4, bottom: 8, right: 8)
    }

    cell.messageField?.font = UIFont.systemFont(ofSize: fontSize)

    let rect = message.value.estimateFrameForText(fontSize: fontSize,
width: (tableView.frame.width - 32)/1.5)

    cell.messageViewHeight.constant = rect.height + 24
    cell.messageWidth.constant = rect.width + 10 + 8 + 8

```

```

        cell.messageField.text = message.value

        cell.dateLabel.text = makeDateRepresentation(sentDate:
message.sentDate)

        cell.userNameLabel.text = message.user?.displayName

        if message.isFav {
            cell.favImg.isHidden = false
        } else {
            cell.favImg.isHidden = true
        }

        setupIncomingCell(cell: cell, message: message, isPrevSame:
isPrevUser, last: isLast)

        return cell
    }

    func setupIncomingCell(cell: IncomingCellHidableElements, message:
UserMessage, isPrevSame: Bool, last: Bool) {
        if (isPrevSame) {
            cell.hideUserPicture(hide: true)
            cell.hideUserName(hide: true)
        } else {
            handleUserPic(message: message, cell: cell)
            cell.hideUserPicture(hide: false)
            cell.hideUserName(hide: false)
        }

        if last {
            cell.hideDate(hide: false)
        } else {
            cell.hideDate(hide: true)
        }
    }

    func handleUserPic(message: UserMessage, cell:
IncomingCellHidableElements){
        if let user = message.user {
            if user.picPath != "" {
                let url = FileStatusData.getLocalPath(filePath: user.picPath)

                if FileManager.default.fileExists(atPath: url.path) {
                    cell.setUserPic(image: UIImage(contentsOfFile:
url.path)!)
                } else {
                    let fileRef = Storage.storage().reference(withPath:
user.picPath)

                    // Download to the local filesystem
                    _ = fileRef.write(toFile: url) { url, error in
                        if error != nil {
                            print("ERROR DOWNLOADING")
                        } else {
                            self.tableView.reloadData()
                        }
                    }
                }
            }
        } else {

```

```

        cell.setUserPic(image: UIImage(named: "default.png")!)
    }
}

var playingCell: IndexPath?

func setAudioPlayer(message: UserMessage, cell: AudioCell, indexPath:
IndexPath){
    if checkAudioMessage(message: message) {
        cell.setPlayButtonEnabled(state: true)

        let url = self.getLocalPathForAudioMes(mes: message)

        if (audioPlayer != nil){
            if (audioPlayer.url != url){
                do {
                    let tempAudioPlayer = try AVAudioPlayer(contentsOf:
url)

                    cell.setProgressViewValue(value: 0.0)
                    cell.setAudioTimeValue(value:
getTimeRepresentation(time: tempAudioPlayer.duration -
tempAudioPlayer.currentTime))
                    cell.setPlayAction(needAnim: false)
                } catch {
                    if let err = error as Error? {
                        print("AVAudioPlayer error:
\\(err.localizedDescription)")
                        audioPlayer = nil
                    }
                }
            } else {
                if (audioPlayer.isPlaying) {
                    cell.setPauseAction(needAnim: false)
                } else {
                    cell.setPlayAction(needAnim: false)
                }
            }
        } else {
            do {
                let tempAudioPlayer = try AVAudioPlayer(contentsOf: url)
                cell.setAudioTimeValue(value: getTimeRepresentation(time:
tempAudioPlayer.duration - tempAudioPlayer.currentTime))
                cell.setProgressViewValue(value: 0.0)
                cell.setPlayAction(needAnim: false)
            } catch {
                if let err = error as Error? {
                    print("AVAudioPlayer error:
\\(err.localizedDescription)")
                    audioPlayer = nil
                }
            }
        }

        cell.setPlayAction {

            if (self.audioPlayer != nil){
                if (self.audioPlayer.url == url){
                    if (self.isAudioPlayerPaused){
                        self.audioPlayer.play()
                    }
                }
            }
        }
    }
}

```

```

        self.playingCell = indexPath
        self.isAudioPlayerPaused = false
    }
} else {
    self.isAudioPlayerPaused = false

    if (self.audioPlayer.isPlaying){
        self.audioPlayer.stop()
    }

    self.audioPlayer = nil
    usleep(50000)
    self.tableView.reloadData()
    usleep(50000)

    self.preparePlayer(path:
self.getLocalPathForAudioMes(mes: message).absoluteString)
        self.playingCell = indexPath
    }
} else {
    self.preparePlayer(path:
self.getLocalPathForAudioMes(mes: message).absoluteString)
        self.playingCell = indexPath
    }

    self.updateCellAudio(indexPath: indexPath)
}

cell.onPauseAction {
    if self.audioPlayer != nil {
        if self.audioPlayer.isPlaying {
            self.audioPlayer.pause()
            self.isAudioPlayerPaused = true
            self.playingCell = nil
        }
    }
}

} else {
    cell.setPlayButtonEnabled(state: false)
    cell.setPlayAction { print("clicked") }
    self.playingCell = nil
    downloadAudioMessage(message: message, indexPath: indexPath)
}
}

func makeOutgoingAudioCell(message: UserMessage, isPrevUser: Bool,
isLast: Bool, indexPath: IndexPath) -> OutgoingAudioCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"outgoingAudioCell", for: indexPath) as! OutgoingAudioCell

    cell.dateLabel.text = makeDateRepresentation(sentDate:
message.sentDate)

    cell.audioDescriptionText.text = message.additionalText

    if message.isFav {
        cell.favImg.isHidden = false
    } else {

```

```

        cell.favImg.isHidden = true
    }

    cell.finishToggle = { self.tableView.performBatchUpdates(nil,
completion: nil) }

    if message.additionalText != "" {
        cell.dynamicButton.isHidden = false
    } else {
        cell.dynamicButton.isHidden = true
    }

    if isLast {
        cell.hideDate(hide: false)
    } else {
        cell.hideDate(hide: true)
    }

    setAudioPlayer(message: message, cell: cell, indexPath: indexPath)

    return cell
}

func updateCellAudio(indexPath: IndexPath){
    DispatchQueue.global(qos: .background).async {
        if (self.audioPlayer != nil){

            var prevProgress: Float = 0.0

            while (self.audioPlayer.isPlaying &&
!self.isAudioPlayerPaused) {
                DispatchQueue.main.async {
                    let audioCell = self.tableView.cellForRow(at:
indexPath) as? AudioCell
                    if let cell = audioCell {
                        let progress = Float(self.audioPlayer.currentTime
/ self.audioPlayer.duration)
                        if (prevProgress - progress > 0.9) {
                            cell.setProgressViewValue(value: 0.0)
                        } else {
                            cell.setProgressViewValue(value: progress)
                            cell.setAudioTimeValue(value:
self.getTimeRepresentation(time: self.audioPlayer.duration -
self.audioPlayer.currentTime))
                        }

                        prevProgress = progress
                    }
                }
                usleep(10000)
                if (self.audioPlayer == nil){
                    self.playingCell = nil
                    DispatchQueue.main.async {
                        self.tableView.reloadData()
                    }
                    break
                }
            }
        }
    }
}

```

```

    }
}

func makeIncomingAudioCell(message: UserMessage, isPrevUser: Bool,
isLast: Bool, indexPath: IndexPath) -> IncomingAudioCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"incomingAudioCell", for: indexPath) as! IncomingAudioCell

    cell.imgClickFunc = {

        let viewController: UserCabinetViewController =
UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: "userInfo") as!
UserCabinetViewController

        viewController.user = message.user

        self.navigationController?.pushViewController(viewController,
animated: true)
    }

    cell.dateLabel.text = makeDateRepresentation(sentDate:
message.sentDate)

    cell.audioDescriptionText.text = message.additionalText
    cell.userNameLabel.text = message.user?.displayName

    cell.finishToggle = { self.tableView.performBatchUpdates(nil,
completion: nil) }

    if message.additionalText != "" {
        cell.dynamicButton.isHidden = false
    } else {
        cell.dynamicButton.isHidden = true
    }

    if message.isFav {
        cell.favImg.isHidden = false
    } else {
        cell.favImg.isHidden = true
    }

    setupIncomingCell(cell: cell, message: message, isPrevSame:
isPrevUser, last: isLast)

    setAudioPlayer(message: message, cell: cell, indexPath: indexPath)

    return cell
}

// MARK: NavigationBar

func setupNavBar(){
    if let navigationBar = self.navigationController?.navigationBar {
        let gradient = CAGradientLayer()
        var bounds = navigationBar.bounds
        bounds.size.height +=
UIApplication.shared.statusBarFrame.size.height
        gradient.frame = bounds
    }
}

```

```

        gradient.colors = [softBlueColor.cgColor,
softOrangeColor.cgColor]
        gradient.startPoint = CGPoint(x: 0, y: 0)
        gradient.endPoint = CGPoint(x: 1, y: 0)

        if let image = getImageFrom(gradientLayer: gradient) {
            navigationBar.setBackgroundImage(image, for:
UIBarMetrics.default)
        }

        navigationBar.titleTextAttributes =
[NSAttributedString.Key.foregroundColor: UIColor.white]
        navigationBar.tintColor = .white
        navigationBar.barStyle = .black

        navigationItem.title = "Үат"

        let faveButton = DOFavoriteButtonNew(frame: CGRect(x: 0, y:0,
width: 45, height: 45), image: UIImage(named: "favStar.png"))
        faveButton.addTarget(self, action:
#selector(favButtonTapped(sender:)), for: UIControl.Event.touchUpInside)

        faveButton.imageColorOff = softBlueColor

        let favItem = UIBarButtonItem(customView: faveButton)

        let userCabinetBtn = UIBarButtonItem(barButtonItemSystemItem:
.action, target: self, action: #selector(userCabClicked))

        self.navigationItem.setRightBarButtonItems([userCabinetBtn,
favItem], animated: false)
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        if let destinationVC = segue.destination as?
UserCabinetViewController {

            destinationVC.user = currentUser
        }
    }

    @objc func userCabClicked(){

        if (currentUser?.picPath != ""){
            let destinationURL = FileStatusData.getLocalPath(filePath:
currentUser!.picPath)

            if !FileManager.default.fileExists(atPath: destinationURL.path){

                let fileRef = Storage.storage().reference(withPath:
currentUser!.picPath)

                _ = fileRef.write(toFile: destinationURL) { url, error in
                    if error != nil {
                        print("ERROR DOWNLOADING")
                    } else {
                        self.performSegue(withIdentifier: "showUser", sender:
nil)

```

```

        }
    }
    } else {
        self.performSegue(withIdentifier: "showUser", sender: nil)
    }
} else {
    self.performSegue(withIdentifier: "showUser", sender: nil)
}
}

}

@objc func favButtonTapped(sender: DOFavoriteButtonNew) {
    if let indexPath = self.playingCell {
        let audioCell = self.tableView.cellForRow(at: indexPath) as?
AudioCell
        audioCell?.getPlayButton().sendActions(for: .touchUpInside)
    }

    if sender.isSelected {
        // deselect
        sender.deselect()
        messageList = realm.objects(UserMessage.self)
        self.tableView.reloadData()
    } else {
        // select with animation
        sender.select()
        messageList = realm.objects(UserMessage.self).filter("isFav =
true")

        self.tableView.reloadData()
    }
}

func getImageFrom(layer: CAGradientLayer) -> UIImage? {
    var gradientImage: UIImage?
    UIGraphicsBeginImageContext(layer.frame.size)
    if let context = UIGraphicsGetCurrentContext() {
        layer.render(in: context)
        gradientImage =
UIGraphicsGetImageFromCurrentImageContext()?.resizableImage(withCapInsets:
UIEdgeInsets.zero, resizingMode: .stretch)
    }
    UIGraphicsEndImageContext()
    return gradientImage
}

// MARK: Speech

func requestTranscribePermissions() {
    SFSpeechRecognizer.requestAuthorization { authStatus in
        DispatchQueue.main.async {
            if authStatus == .authorized {
                print("Good to go!")
            } else {
                print("Transcription permission was declined.")
            }
        }
    }
}
}

```

```

func transcribeAudio(url: URL, msg: UserMessage) {
    // create a new recognizer and point it at our audio
    let recognizer = SFSpeechRecognizer()
    let request = SFSpeechURLRecognitionRequest(url: url)

    // start recognition!
    recognizer?.recognitionTask(with: request) { (result, error) in
        // abort if we didn't get any transcription back
        guard let result = result else {
            print("There was an error: \(error!)")
            return
        }

        // if we got the final transcription back, print it
        if result.isFinal {
            // pull out the best transcription...
            try! realm.write {
                msg.additionalText =
result.bestTranscription.formattedString
            }
        }
    }

    // MARK: Recording Setup

    func setupRecordButton(){

        recordingSession = AVAudioSession.sharedInstance()

        do {
            try recordingSession.setCategory(.playAndRecord, options:
.defaultToSpeaker)
            try recordingSession.setActive(true)
            recordingSession.requestRecordPermission() { [unowned self]
allowed in
                DispatchQueue.main.async {
                    if allowed {
                        self.setupSendAudioMsg()
                    } else {
                        // failed to record!
                    }
                }
            }
        } catch {
            // failed to record!
        }
    }

    func setupSendAudioMsg(){
        sendButton.removeTarget(self, action: #selector(sendText), for:
.touchDown)
        sendButton.addTarget(self, action: #selector(recordTapped), for:
.touchDown)
    }

    func setupSendTextMsg(){
        sendButton.removeTarget(self, action: #selector(recordTapped), for:
.touchDown)

```

```

        sendButton.addTarget(self, action: #selector(sendText), for:
        .touchDown)
    }

    @objc func sendText() {
        handleSend(value: inputTextView.text, type: .text)
    }

    func startRecording() {
        let audioId = "\(UUID().uuidString).m4a"
        let audioFilename =
getDocumentsDirectory().appendingPathComponent(audioId)

        let settings = [
            AVFormatIDKey: Int(kAudioFormatMPEG4AAC),
            AVSampleRateKey: 12000,
            AVNumberOfChannelsKey: 1,
            AVEncoderAudioQualityKey: AVAudioQuality.high.rawValue
        ]

        do {
            audioRecorder = try AVAudioRecorder(url: audioFilename, settings:
settings)
            audioRecorder.delegate = self
            audioRecorder.prepareToRecord()
            audioRecorder.record()

            self.inputTextView.text = String(format: "%02d:%02d", 0, 0)
            self.inputTextView.textColor = .red
            self.inputTextView.isEditable = false

            DispatchQueue.global(qos: .background).async {
                if (self.audioRecorder != nil){

                    while (self.audioRecorder.isRecording) {
                        let playinTime = self.getTimeRepresentation(time:
self.audioRecorder.currentTime)

                        DispatchQueue.main.async {
                            self.inputTextView.text = playinTime
                        }
                        usleep(1000)
                    }
                }
            }

        } catch {
            finishRecording(success: false)
        }
    }

    func getDocumentsDirectory() -> URL {
        let paths = FileManager.default.urls(for: .cachesDirectory, in:
.userDomainMask)
        return paths[0]
    }

    func finishRecording(success: Bool) {
        audioRecorder.stop()
    }

```

```

DispatchQueue.global(qos: .background).async {
    DispatchQueue.main.async {
        self.inputTextView.isEditable = true
        self.inputTextView.text = ""
        self.textViewDidEndEditing(self.inputTextView)
    }
}

if success {
    animateNavigationBar()
    uploadAudioMsg(fileurl: audioRecorder.url, fromUrl:
audioRecorder.url)
}
sleep(1)
audioRecorder = nil
}

func animateNavigationBar(){
    let options = BusyNavigationBarOptions()

    options.animationType = .stripes
    options.color = UIColor.white
    options.alpha = 0.8
    options.barWidth = 20
    options.gapWidth = 30
    options.speed = 1
    options.transparentMaskEnabled = true
    self.navigationController?.navigationBar.start(options)
}

func uploadAudioMsg(fileurl: URL, fromUrl: URL){
    let store = Storage.storage()
    let storeRef = store.reference()

    _ = storeRef.child(fileurl.lastPathComponent).putFile(from: fromUrl,
metadata: nil) { (metadata, error) in
        self.navigationController?.navigationBar.stop()
        guard let metadata = metadata else {
            print("Error occurred: \(error!)")
            return
        }
        let msg = self.handleSend(value: metadata.path!, type: .audio,
needUpdate: false)
        self.transcribeAudio(url: fileurl, msg: msg)
    }
}

@objc func recordTapped() {
    if audioRecorder == nil {
        startRecording()
    } else {
        finishRecording(success: true)
    }
}

func audioRecorderDidFinishRecording(_ recorder: AVAudioRecorder,
successfully flag: Bool) {
    if !flag {
        finishRecording(success: false)
    }
}

```

```

}

// MARK: Audio player

var isAudioPlayerPaused = false

func preparePlayer(path: String) {
    do {
        let url2 = URL(string: path)
        if let url = url2 {
            audioPlayer = try AVAudioPlayer(contentsOf: url)
            audioPlayer.delegate = self
            audioPlayer.volume = 1.0
            audioPlayer.prepareToPlay()
            audioPlayer.play()
        }
    } catch {
        if let err = error as Error? {
            print("AVAudioPlayer error: \(err.localizedDescription)")
            audioPlayer = nil
        }
    }
}

func finishPlaying(success: Bool) {
    audioPlayer.stop()
    audioPlayer = nil

    if success {
        print("Success played")
    } else {
        print("Error played")
    }
}

func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully
flag: Bool) {
    finishPlaying(success: flag)
}

func getLocalPathForAudioMes(mes: UserMessage) -> URL {
    let path = FileManager.default.urls(for: .cachesDirectory, in:
.userDomainMask)[0]
    let firebaseUrl = URL(fileURLWithPath: mes.value)
    let dest = path.appendingPathComponent(firebaseUrl.lastPathComponent)

    return dest
}

func checkAudioMessage(message: UserMessage) -> Bool {
    if message.type == .audio {
        let dest = getLocalPathForAudioMes(mes: message)

        if (FileManager.default.fileExists(atPath: dest.path)) {
            return true
        } else {
            return false
        }
    }
}

```

```

        return false
    }

    func downloadAudioMessage(message: UserMessage, indexPath: IndexPath) {
        let path = FileManager.default.urls(for: .cachesDirectory, in:
.userDomainMask)[0]
        let firebaseUrl = URL(fileURLWithPath: message.value)
        let dest = path.appendingPathComponent(firebaseUrl.lastPathComponent)

        let fileRef = Storage.storage().reference(withPath: message.value)

        _ = fileRef.write(toFile: dest) { url, error in
            if error != nil {
                print("Error downloading audio mes")
            } else {
                self.tableView.reloadRows(at: [indexPath], with: .automatic)
            }
        }
    }

    // MARK: Other

    func getTimeRepresentation(time: TimeInterval) -> String{
        let min = Int(time / 60)

        let sec = Int(time.truncatingRemainder(dividingBy: 60))
        let playinTime = String(format: "%02d:%02d", min, sec)

        return playinTime
    }

    // MARK: Context Menu Setup

    func tableView(_ tableView: UITableView, contextMenuConfigurationForRowAt
indexPath: IndexPath, point: CGPoint) -> UIContextMenuConfiguration? {

        let configuration = UIContextMenuConfiguration(identifier: nil,
previewProvider: nil){ action in

            let message = self.messageList![indexPath.row]

            let favOption: UIAction?

            if (message.isFav){
                favOption = UIAction(title: "Удалить из избранного", image:
UIImage(named: "favStar.png")?.maskWithColor(color: .gray), identifier: nil)
            }
            {_ in
                DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
                    try! realm.write {
                        message.isFav = false
                    }
                }
            }
        } else {
            favOption = UIAction(title: "Добавить в избранное", image:
UIImage(named: "favStar.png")?.maskWithColor(color: UIColor(hexString:
"FFDF00"))), identifier: nil) {_ in
                DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
                    try! realm.write {
                        message.isFav = true
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}

return UIMenu(title: "Избранное", image: nil, identifier: nil,
children: [favOption!])
}

return configuration
}
}

extension UITableView {
    func scrollToNextItem() {
        let contentOffset = CGFloat(floor(self.contentOffset.x +
self.bounds.size.width))
        self.moveToFrame(contentOffset: contentOffset)
    }

    func scrollToPreviousItem() {
        let contentOffset = CGFloat(floor(self.contentOffset.x -
self.bounds.size.width))
        self.moveToFrame(contentOffset: contentOffset)
    }

    func scrollToLast() {
        guard numberOfSections > 0 else {
            return
        }

        let lastSection = numberOfSections - 1

        guard numberOfRows(inSection: lastSection) > 0 else {
            return
        }

        let lastItemIndexPath = IndexPath(item: numberOfRows(inSection:
lastSection) - 1,
                                           section: lastSection)

        scrollToRow(at: lastItemIndexPath, at: .bottom, animated: true)
    }

    func moveToFrame(contentOffset : CGFloat) {
        self.setContentOffset(CGPoint(x: contentOffset, y:
self.contentOffset.y), animated: true)
    }
}

extension String {
    func height(withConstrainedWidth width: CGFloat, font: UIFont) -> CGFloat
    {
        let constraintRect = CGSize(width: width, height:
.greatestFiniteMagnitude)
        let options =
NSStringDrawingOptions.usesFontLeading.union(.usesLineFragmentOrigin)
        let boundingBox = self.boundingBox(with: constraintRect, options:
options, attributes: [.font : font], context: nil)
    }
}

```

```

        return ceil(boundingBox.height)
    }

    func width(withConstrainedHeight height: CGFloat, font: UIFont) ->
    CGFloat {
        let constraintRect = CGSize(width: .greatestFiniteMagnitude, height:
height)
        let options =
NSStringDrawingOptions.usesFontLeading.union(.usesLineFragmentOrigin)
        let boundingBox = self.boundingRect(with: constraintRect, options:
options, attributes: [.font : font], context: nil)

        return ceil(boundingBox.width)
    }

    func estimateFrameForText(fontSize: CGFloat, width: CGFloat) -> CGRect {

        let size = CGSize(width: width, height: 1000)
        let options =
NSStringDrawingOptions.usesFontLeading.union(.usesLineFragmentOrigin)

        return NSString(string: self).boundingRect(with: size, options:
options, attributes: [NSAttributedString.Key.font: UIFont.systemFont(ofSize:
fontSize)], context: nil)
    }
}

extension UIImage {

    func maskWithColor(color: UIColor) -> UIImage? {
        let maskImage = cgImage!

        let width = size.width
        let height = size.height
        let bounds = CGRect(x: 0, y: 0, width: width, height: height)

        let colorSpace = CGColorSpaceCreateDeviceRGB()
        let bitmapInfo = CGBitmapInfo(rawValue:
CGImageAlphaInfo.premultipliedLast.rawValue)
        let context = CGContext(data: nil, width: Int(width), height:
Int(height), bitsPerComponent: 8, bytesPerRow: 0, space: colorSpace,
bitmapInfo: bitmapInfo.rawValue)!

        context.clip(to: bounds, mask: maskImage)
        context.setFillColor(color.cgColor)
        context.fill(bounds)

        if let cgImage = context.makeImage() {
            let coloredImage = UIImage(cgImage: cgImage)
            return coloredImage
        } else {
            return nil
        }
    }
}

extension Character {
    /// A simple emoji is one scalar and presented to the user as an Emoji
    var isSimpleEmoji: Bool {

```

```

        guard let firstProperties = unicodeScalars.first?.properties else {
            return false
        }
        return unicodeScalars.count == 1 &&
            (firstProperties.isEmojiPresentation ||
             firstProperties.generalCategory == .otherSymbol)
    }

    /// Checks if the scalars will be merged into an emoji
    var isCombinedIntoEmoji: Bool {
        return (unicodeScalars.count > 1 &&
                unicodeScalars.contains { $0.properties.isJoinControl ||
                $0.properties.isVariationSelector })
            || unicodeScalars.allSatisfy({ $0.properties.isEmojiPresentation
        })
    }

    var isEmoji: Bool {
        return isSimpleEmoji || isCombinedIntoEmoji
    }
}

extension String {
    var isSingleEmoji: Bool {
        return count == 1 && containsEmoji
    }

    var containsEmoji: Bool {
        return contains { $0.isEmoji }
    }

    var containsOnlyEmoji: Bool {
        return !isEmpty && !contains { !$0.isEmoji }
    }

    var emojiString: String {
        return emojis.map { String($0) }.reduce("", +)
    }

    var emojis: [Character] {
        return filter { $0.isEmoji }
    }

    var emojiScalars: [UnicodeScalar] {
        return filter{ $0.isEmoji }.flatMap { $0.unicodeScalars }
    }
}

```

Листинг A.2 – LoginViewController.swift

```

import Foundation
import RealmSwift
import UIKit

var realm: Realm = try! Realm()
var currentUser : User?
class LoginViewController : UIViewController {
    @IBOutlet weak var userName: UITextField!
    @IBOutlet weak var password: UITextField!

```

```

@IBOutlet weak var realName: UITextField!
@IBOutlet weak var errorLabel: UILabel!

var enterSender : User?

override func viewDidLoad() {
    super.viewDidLoad()
}

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

    hideKeyboardWhenTappedAround()

    clearFields(isRegistrationView: self.restorationIdentifier ==
"registrationView" ? true : false)

    navigationController?.setNavigationBarHidden(true, animated:
animated)
}

@IBAction func loginPressed(_ sender: Any) {
    errorLabel.isHidden = true

    if checkFields(isRegistration: false) {

        login(username: userName.text!, password: password.text!,
register: false)
    }
}

func finishLogin(res: Bool) {
    if res {
        let user = realm.objects(User.self).filter("senderId =
'\(self.userName.text!)'")
        self.enterSender = user[0]

        self.performSegue(withIdentifier: "loginSegue", sender: self)
    } else {
        errorLabel.text = "Неверный логин или пароль"
        errorLabel.isHidden = false
    }
}

func finishRegister(res: Bool) {
    if res {
        let newUser = User()
        newUser.senderId = self.userName.text!
        newUser.displayName = self.realName.text!
        newUser.password = self.password.text!

        try! realm.write {
            realm.add(newUser)
            self.enterSender = newUser

            self.performSegue(withIdentifier: "registerSegue", sender:
self)
        }
    } else {
        errorLabel.text = "Пользователь с таким логином уже существует"
    }
}

```

```

        errorLabel.isHidden = false
    }
}

@IBAction func registerPressed(_ sender: Any) {
    errorLabel.isHidden = true

    if checkFields(isRegistration: true) {
        logIn(username: userName.text!, password: password.text!,
register: true)
    }
}

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destinationVC = segue.destination as? NewChatViewController {
        if let user = enterSender {
            currentUser = user
        }
    }
}

func checkFields(isRegistration : Bool) -> Bool {
    let res1 = checkUsername()

    var res2 = true
    if isRegistration {
        res2 = checkRealName()
    }

    let res3 = checkPassword()
    return res1 && res2 && res3
}

func checkUsername() -> Bool {
    if userName.text != "" {
        userName.backgroundColor = UIColor.clear
        return true
    } else {
        userName.backgroundColor = UIColor.red.withAlphaComponent(0.5)
        userName.placeholder = "Заполните поле (Логин)"
        return false
    }
}

func checkRealName() -> Bool {
    if realName.text != "" {
        realName.backgroundColor = UIColor.clear
        return true
    } else {
        realName.backgroundColor = UIColor.red.withAlphaComponent(0.5)
        realName.placeholder = "Заполните поле (Отображаемое имя)"
        return false
    }
}

```

```

func checkPassword() -> Bool {

    if password.text != "" {
        password.backgroundColor = UIColor.clear
        return true
    } else {
        password.backgroundColor = UIColor.red.withAlphaComponent(0.5)
        password.placeholder = "Заполните поле (Пароль)"
        return false
    }
}

func clearFields(isRegistrationView : Bool) {
    userName.text = ""
    userName.placeholder = "Логин"

    if (isRegistrationView) {
        realName.text = ""
        realName.backgroundColor = UIColor.clear
        realName.placeholder = "Отображаемое имя"
    }

    password.text = ""
    password.placeholder = "Пароль"

    errorLabel.isHidden = true

    userName.backgroundColor = UIColor.clear
    password.backgroundColor = UIColor.clear
}

@IBAction func hasAccountAlready(_ sender: Any) {

    self.navigationController?.popViewController(animated: true)

}

func logIn(username: String, password: String, register: Bool) {
    print("Log in as user: \(username) with register: \(register)")

    if register {
        let user = realm.objects(User.self).filter("senderId =
'\(self.userName.text!)'")
        if user.count > 0 {
            finishRegister(res: false)
        } else {
            finishRegister(res: true)
        }
    }

    } else {
        let user = realm.objects(User.self).filter("senderId =
'\(self.userName.text!)'")

        if user.count > 0 {
            if user[0].password == password {
                finishLogin(res: true)
            } else {
                finishLogin(res: false)
            }
        }
    }
}

```

```

        } else {
            finishLogin(res: false)
        }
    }
}
}

```

Листинг A.3 – NewDocumentViewController.swift

```

import Foundation
import UIKit
import QuickLook
import Firebase
import MobileCoreServices
import RealmSwift
import QuickLook
import QuickLookThumbnails
import MGSwipeTableCell

class NewDocumentsViewController : UIViewController,
    UIDocumentPickerDelegate,
    UINavigationControllerDelegate,
    QLPreviewControllerDataSource,
    UITableViewDelegate,
    UITableViewDataSource,
    UITextFieldDelegate{

    var categoryId = 1
    @IBOutlet var tableView: UITableView!
    @IBOutlet var finderInput: UITextField!
    @IBOutlet var finderHeight: NSLayoutConstraint!
    var currentCategory: Results<Category>?
    var allFiles: Results<UserFile>?
    var filteredCategory: Results<Category>?
    var isFiltered = false
    var filesInfoDict: [Int : FileStatusData] = [:]
    var notificationToken: NotificationToken? = nil

    override func viewDidLoad() {
        super.viewDidLoad()

        view.accessibilityIdentifier = "documentView"

        setupRealm()
    }

    override func viewWillAppear(_ animated: Bool) {
        self.navigationController?.hero.navigationAnimationType =
        .slide(direction: .left)
    }

    override func viewWillAppear(_ animated: Bool) {
        setNavBar()
    }

    func setupRealm(){
        self.setupView()
    }
}

```

```

func setupView() {
    finderInput.placeholder = "Введите название категории"
    finderInput.delegate = self
    finderInput.addTarget(self, action:
#selector(textFieldDidChange(sender:)), for: .editingChanged)

    tableView.rowHeight = 77
    tableView.delegate = self
    tableView.dataSource = self

    let fileCell = UINib(nibName: "FileCell", bundle: nil)
    tableView.register(fileCell, forCellReuseIdentifier: "fileCell")

    let initCategoryObj = Category()
    initCategoryObj.id = 1
    initCategoryObj.name = "Хранилище"

    // Updating category with id = 1
    try! realm.write {
        realm.create(Category.self, value: ["id": 1], update: .modified)
    }

    currentCategory = realm.objects(Category.self).filter("id ==
\\(categoryId)")

    navigationItem.title = currentCategory?[0].name

    allFiles = realm.objects(UserFile.self)

    setFinderField()

    FileStatusData.gotPreview = { self.tableView.reloadData() }
    getFilesInfo()

    notificationToken = currentCategory!.observe { [weak self] (changes:
RealmCollectionChange) in
        self?.getFilesInfo()
        self?.tableView.reloadData()
    }
}

func setFinderField() {
    if currentCategory?[0].id != 1 {
        finderInput.isHidden = true
        finderHeight.constant = 0
    }
}

@objc func textFieldDidChange(sender: UITextField) {
    performAction()
}

func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    textField.resignFirstResponder()
    performAction()
    return true
}

func performAction() {

```

```

        if let filter = finderInput.text {
            if filter != "" {
                filteredCategory = realm.objects(Category.self).filter("name
CONTAINS '\(filter)'")
                isFiltered = true
                self.tableView.reloadData()
            } else {
                isFiltered = false
                self.tableView.reloadData()
            }
        } else {
            isFiltered = false
            self.tableView.reloadData()
        }
    }

    func getFilesInfo(){
        if let fileUrls = currentCategory?[0].fileUrls {
            for i in 0..

```

```

        return tempName
    }

    public func documentPicker(_ controller: UIDocumentPickerViewController,
didPickDocumentsAt urls: [URL]) {
        guard let myURL = urls.first else {
            return
        }
        print("import result : \(myURL)")

        let store = Storage.storage()
        let storeRef = store.reference()

        let newFileName = getFileNameIfRepeat(path: myURL)

        let uploadUserProfileTask = storeRef.child(newFileName).putFile(from:
myURL, metadata: nil) { (metadata, error) in
            guard let metadata = metadata else {
                print("Error occurred: \(error!)")
                return
            }
            try! realm.write {
                let userFile = UserFile()
                userFile.fileName = metadata.path!
                realm.add(userFile)

                self.currentCategory?[0].fileUrls.append(userFile)
                self.GetFilesInfo()
                self.tableView.reloadData()
            }
        }
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        var result = 0

        if (isFiltered) {
            result = filteredCategory?.count ?? 0
        } else {
            let subCatCount = currentCategory?[0].subCategories.count ?? 0
            let filesCount = currentCategory?[0].fileUrls.count ?? 0

            result = subCatCount + filesCount
        }
        return result
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        var cell = UITableViewCell(style: UITableViewCell.CellStyle.default,
reuseIdentifier: "Cell")

        if isFiltered {
            cell.textLabel?.text = filteredCategory?[indexPath.row].name ??
"Нет категории"
        } else {
            let subCatCount = currentCategory?[0].subCategories.count ?? 0

```

```

        if (indexPath.row <= subCatCount - 1) {
            cell.textLabel?.text =
currentCategory?[0].subCategories[indexPath.row].name ?? "null"
        } else {
            let reuseIdentifier = "fileCell"
            if let cell2 = tableView.dequeueReusableCell(withIdentifier:
reuseIdentifier, for: indexPath) as? FileCell {

                let fileInfo = filesInfoDict[indexPath.row - subCatCount]

                cell2.fileTypeLabel.text =
fileInfo?.fileUrl.pathExtension
                cell2.fileNameLabel.text =
fileInfo?.fileUrl.deletingPathExtension().lastPathComponent

                cell2.path = indexPath

                switch (fileInfo!.state){
                    case .downloading:
                        cell2.progressBar.isHidden = false
                        cell2.progressBar.progress =
Float(fileInfo!.percent / 100)

cell2.progressBar.setProgress(cell2.progressBar.progress, animated: true)
                        cell2.actionButton.setTitle("Пауза", for:
.normal)

                        cell2.buttonAction = downloadingClick
                        break
                    case .startDownload:
                        cell2.progressBar.isHidden = true
                        cell2.actionButton.setTitle("Скачать", for:
.normal)

                        cell2.buttonAction = startDownloadClick
                        break
                    case .downloaded:
                        cell2.progressBar.isHidden = true
                        cell2.actionButton.setTitle("Открыть", for:
.normal)

                        cell2.buttonAction = downloadedClick

                        //configure right buttons

                        let swipeBtn = MGSwipeButton(title: "Удалить",
backgroundColor: .red) {
                            (sender: MGSwipeTableCell!) -> Bool in
                                fileInfo?.deleteLocalFile()
                                self.tableView.reloadRows(at: [indexPath],
with: .automatic)

                                return true
                            }

                        cell2.rightButtons = [swipeBtn]

                        cell2.rightSwipeSettings.transition = .drag
                        break
                    case .paused:
                        cell2.progressBar.isHidden = false
                        cell2.actionButton.setTitle("Возобновить", for:
.normal)

```

```

        cell2.buttonAction = pausedClick
    }

    if let image = fileInfo?.previewImage {
        cell2.filePreviewImage.image = UIImage(cgImage:
image)
    } else {
        cell2.filePreviewImage.image = UIImage(named:
"docIcon")
    }

    return cell2
}

cell.textLabel?.text = "no file found"

}
}

return cell
}

func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    var chosenId = 0

    if !isFiltered {
        let subCatCount = currentCategory?[0].subCategories.count ?? 0
        if indexPath.row > subCatCount - 1 {
            return
        }
        chosenId = currentCategory![0].subCategories[indexPath.row].id
    } else {
        chosenId = filteredCategory?[indexPath.row].id ?? 0
    }

    let viewControllerStoryboardId = "documentsController"
    let storyboardName = "Main"
    let storyboard = UIStoryboard(name: storyboardName, bundle:
Bundle.main)
    let controller = storyboard.instantiateViewController(withIdentifier:
viewControllerStoryboardId) as! NewDocumentsViewController

    controller.categoryId = chosenId

    self.navigationController?.pushViewController(controller, animated:
true)

    tableView.reloadData()
}

func documentPickerWasCancelled(_ controller:
UIDocumentPickerViewController) {
    print("view was cancelled")
    dismiss(animated: true, completion: nil)
}

func openiCloudDocuments() {

```

```

        let importMenu = UIDocumentPickerViewController(documentTypes:
[String("public.data")], in: .import)
        importMenu.delegate = self
        importMenu.modalPresentationStyle = .formSheet
        self.present(importMenu, animated: true, completion: nil)
    }

    @objc func addSubCategory() {
        var text = ""
        let alert = UIAlertController(style: .alert, title: "Добавление
категории")
        let config: TextField.Config = { textField in
            textField.becomeFirstResponder()
            textField.textColor = .black
            textField.placeholder = "Название категории"
            textField.leftViewPadding = 12
            textField.borderWidth = 1
            textField.cornerRadius = 8
            textField.borderColor = UIColor.lightGray.withAlphaComponent(0.5)
            textField.backgroundColor = nil
            textField.keyboardAppearance = .default
            textField.keyboardType = .default
            textField.isSecureTextEntry = false
            textField.returnKeyType = .done
            textField.action { textField in
                text = textField.text ?? ""
            }
        }
        alert.addOneTextField(configuration: config)
        alert.addAction(title: "Добавить", style: .default) { action in

            if text != "" {
                try! realm.write {
                    let subCategory = Category()
                    subCategory.id = (self.getMaxId() ?? 0) + 1
                    subCategory.name = text

                    realm.add(subCategory)

self.currentCategory?[0].subCategories.append(subCategory)

                    self.tableView.reloadData()
                }
            }
        }
        alert.addAction(title: "Отмена", style: .cancel)
        alert.show()
    }

    @objc func addFile() {
        openCloudDocuments()
    }

    func getMaxId() -> Int? {
        let categories = realm.objects(Category.self)
        return categories.max(ofProperty: "id") as Int?
    }

```

```

func numberOfPreviewItems(in controller: QLPreviewController) -> Int {
    return 1
}

func previewController(_ controller: QLPreviewController, previewItemAt
index: Int) -> QLPreviewItem {
    let fileInfo = filesInfoDict[previewIndex]
    return NSURL(fileURLWithPath: fileInfo!.localPath!)
}

var previewIndex = 0

func startDownloadClick(path: IndexPath?)
{
    let subCatCount = currentCategory?[0].subCategories.count ?? 0

    let fileInfo = filesInfoDict[path!.row - subCatCount]

    self.tableView.reloadRows(at: [path!], with: .none)

    fileInfo?.downloadFile(progressObserverHandler: {
        self.tableView.reloadRows(at: [path!], with: .none)
    })
}

func downloadingClick(path: IndexPath?)
{
    let subCatCount = currentCategory?[0].subCategories.count ?? 0

    let fileInfo = filesInfoDict[path!.row - subCatCount]

    fileInfo?.downloadTask?.pause()
    fileInfo?.state = .paused
    self.tableView.reloadRows(at: [path!], with: .none)
}

func pausedClick(path: IndexPath?)
{
    let subCatCount = currentCategory?[0].subCategories.count ?? 0

    let fileInfo = filesInfoDict[path!.row - subCatCount]

    fileInfo?.downloadTask?.resume()
    fileInfo?.state = .downloading
    self.tableView.reloadRows(at: [path!], with: .none)
}

func downloadedClick(path: IndexPath?)
{
    let subCatCount = currentCategory?[0].subCategories.count ?? 0

    let fileInfo = filesInfoDict[path!.row - subCatCount]

    previewIndex = path!.row - subCatCount

    if let localPath = fileInfo?.localPath {
        if QLPreviewController.canPreview(NSURL(fileURLWithPath:
localPath)) {

```

```

        let quickLookController = QLPreviewController()
        quickLookController.dataSource = self
        navigationController?.pushViewController(quickLookController,
animated: true)
    }
}
}
}
}

```

Листинг A.4 – CalendarViewController.swift

```

import Foundation
import UIKit
import CalendarKit
import DateToolsSwift
import RealmSwift

class CalendarViewController : DayViewController {
    var storage : Results<UserEvent>?
    var notificationToken: NotificationToken? = nil
    var chosenDate: Date = Date()

    @IBAction func changeDateClicked(_ sender: Any) {
        let alert = UIAlertController(style: .actionSheet, title: "Select
date")
        alert.addDatePicker(mode: .date, date: chosenDate) { date in
            self.chosenDate = date
        }
        let changeAction = UIAlertAction(title: "Выбрать", style: .default) {
            _ in
                self.dayView.state?.move(to: self.chosenDate)
            }
        let cancelAction = UIAlertAction(title: "Отмена", style: .cancel) { _
in
            print("Cancel")
        }

        alert.addAction(changeAction)
        alert.addAction(cancelAction)
        alert.show()
    }

    enum SelectedStyle {
        case Dark
        case Light
    }

    func loadItems(){
        storage = realm.objects(UserEvent.self)
        reloadData()
    }

    override func viewWillAppear(_ animated: Bool) {
        setNavBar()
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}

```

```

        view.accessibilityIdentifier = "calendarView"
        setupRealm()
    }

    override func viewDidLoad(_ animated: Bool) {
        self.navigationController?.hero.navigationAnimationType =
.slide(direction: .right)
    }

    func setNavBar() {
        var buttons = [UIBarButtonItem]()
        if (currentUser?.userType == .Admin) {
            let addEventItem = UIBarButtonItem(barButtonSystemItem: .add,
target: self, action: #selector(addEvent))
            buttons.append(addEventItem)
        }

        let editDateItem = UIBarButtonItem(barButtonSystemItem: .edit,
target: self, action: #selector(editDate))
        buttons.append(editDateItem)

        self.navigationItem.setRightBarButtonItems(buttons, animated: true)
    }

    @objc func editDate() {
        let alert = UIAlertController(style: .actionSheet, title: "Select
date")
        alert.addDatePicker(mode: .date, date: chosenDate) { date in
            self.chosenDate = date
        }
        let changeAction = UIAlertAction(title: "Выбрать", style: .default) {
            self.dayView.state?.move(to: self.chosenDate)
        }

        let cancelAction = UIAlertAction(title: "Отмена", style: .cancel) { _
            print("Cancel")
        }

        alert.addAction(changeAction)
        alert.addAction(cancelAction)
        alert.show()
    }

    @objc func addEvent() {
        self.performSegue(withIdentifier: "addEventSegue", sender: nil)
    }

    func setupRealm() {
        self.setupView()
    }

    func setupView() {
        loadItems()
        notificationToken = storage!.observe { [weak self] (changes:
RealmCollectionChange) in
            self?.reloadData()
        }
        title = "События"
    }

```

```

        dayView.scrollToFirstEvent = true
        reloadData()
    }

    override func eventsForDate(_ date: Date) -> [EventDescriptor] {

        var events = [Event]()

        let dateStorage = storage?.filter("startDate BETWEEN %@", date,
date.adding(.day, value: 1))

        if let mystorage = dateStorage {
            for model in mystorage {

                let event = Event()
                event.startDate = model.startDate
                event.endDate = model.endDate
                event.text = model.text
                event.color = UIColor.green
                events.append(event)
            }
        }

        let cycleStorage = storage?.filter("isCycle == true")

        if let mystorage = cycleStorage {
            for model in mystorage {

                let dates = checkCycleEvent(givenDate: date, event: model)

                if let startDate = dates.0 {
                    if let endDate = dates.1 {

                        let event = Event()

                        event.startDate = startDate
                        event.endDate = endDate

                        event.text = model.text
                        event.color = UIColor.green
                        events.append(event)
                    }
                }
            }
        }

        return events
    }

    func checkCycleEvent(givenDate: Date, event: UserEvent) -> (Date?, Date?)
{

    var dateComponent: Calendar.Component = .day

    if (event.isCycle){
        switch event.cycle {
            case .day:
                dateComponent = .day
            case .week:
                dateComponent = .weekOfYear
        }
    }
}

```

```

        case .month:
            dateComponent = .month
        case .year:
            dateComponent = .year
        case .noCycle:
            return (nil, nil)
        }
    }

    var startDate = event.startDate
    var endDate = event.endDate

    while (!givenDate.isEarlier(than: startDate)){
        startDate = startDate.adding(dateComponent, value: 1)
        endDate = endDate.adding(dateComponent, value: 1)

        if startDate.isSameDay(date: givenDate) {
            return (startDate, endDate)
        }
    }

    return (nil, nil)
}

var colors = [UIColor.blue,
              UIColor.yellow,
              UIColor.green,
              UIColor.red]

var currentStyle = SelectedStyle.Light

// MARK: EventDataSource

private func textColorForEventInDarkTheme(baseColor: UIColor) -> UIColor
{
    var h: CGFloat = 0, s: CGFloat = 0, b: CGFloat = 0, a: CGFloat = 0
    baseColor.getHue(&h, saturation: &s, brightness: &b, alpha: &a)
    return UIColor(hue: h, saturation: s * 0.3, brightness: b, alpha: a)
}

// MARK: DayViewDelegate

override func dayViewDidSelectEventView(_ eventView: EventView) {
    guard let descriptor = eventView.descriptor as? Event else {
        return
    }
    print("Event has been selected: \(descriptor) \(String(describing:
descriptor.userInfo))")
}

override func dayViewDidLongPressEventView(_ eventView: EventView) {
    guard let descriptor = eventView.descriptor as? Event else {
        return
    }
    print("Event has been longPressed: \(descriptor) \(String(describing:
descriptor.userInfo))")
}

```

```

    override func dayView(dayView: DayView, willMoveTo date: Date) {
        print("DayView = \(dayView) will move to: \(date)")
    }

    override func dayView(dayView: DayView, didMoveTo date: Date) {
        print("DayView = \(dayView) did move to: \(date)")
    }

    override func dayView(dayView: DayView, didLongPressTimelineAt date:
Date) {
        print("Did long press timeline at date \(date)")
    }
}

```

Листинг A.5 – UserCabinetViewController.swift

```

import Foundation
import UIKit
import Firebase
import BusyNavigationBar

class UserCabinetViewController: UIViewController,
UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    var imagePicker = UIImagePickerController()
    @IBOutlet var userPic: UIImageView!
    @IBOutlet var displayName: UITextField!
    @IBOutlet var login: UITextField!
    @IBOutlet var changeBtn: UIButton!

    var user: User?

    override func viewDidLoad() {
        displayName.text = user?.displayName
        login.text = user?.senderId
        userPic.layer.borderWidth = 1
        userPic.layer.borderColor = UIColor.white.cgColor
        userPic.layer.masksToBounds = false
        userPic.layer.cornerRadius = userPic.frame.height/2
        userPic.clipsToBounds = true

        if (user?.senderId != currentUser?.senderId){
            changeBtn.isHidden = true
        }

        if (user!.picPath != ""){
            let destinationURL = FileStatusData.getLocalPath(filePath:
user!.picPath)
            if FileManager.default.fileExists(atPath: destinationURL.path){
                userPic.image = UIImage(contentsOfFile: destinationURL.path)
            }
        }
        hideKeyboardWhenTappedAround()
    }

    @IBAction func changePicClick(_ sender: Any) {
        if UIImagePickerController.isSourceTypeAvailable(.savedPhotosAlbum){
            print("Button capture")
            imagePicker.delegate = self

```

```

        imagePicker.sourceType = .savedPhotosAlbum
        imagePicker.allowsEditing = false
        present(imagePicker, animated: true, completion: nil)
    }
}

func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
    picker.dismiss(animated: true, completion: nil)
}

func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any])
{
    picker.dismiss(animated: true, completion: nil)
    guard let image = info[.originalImage] as? UIImage else {
        fatalError("Expected a dictionary containing an image, but was
provided the following: \(info)")
    }
    guard let nsUrl = info[.imageURL] as? NSURL else {
        fatalError("Expected a dictionary containing an url, but was
provided the following: \(info)")
    }

    let store = Storage.storage()
    let storeRef = store.reference()
    animateNavigationBar()
    let url = URL(fileURLWithPath: nsUrl.path!)

    if (user?.picPath != "") {
        storeRef.child(user!.picPath).delete(completion: nil)
        let destinationURL = FileStatusData.getLocalPath(filePath:
user!.picPath)
        if FileManager.default.fileExists(atPath: destinationURL.path) {
            try? FileManager.default.removeItem(at: destinationURL)
        }
    }

    _ = storeRef.child(url.lastPathComponent).putFile(from: url,
metadata: nil) { (metadata, error) in
        guard let metadata = metadata else {
            print("Error occurred: \(error!)")
            return
        }
    }

    let destinationURL = FileStatusData.getLocalPath(filePath:
metadata.path!)
    try? FileManager.default.removeItem(at: destinationURL)
    try? FileManager.default.copyItem(at: url, to: destinationURL)
    try! realm.write {
        self.user?.picPath = metadata.path!
        self.userPic.image = image
    }
    self.navigationController?.navigationBar.stop()
}

func getLocalPath(filePath: String) -> URL {

    let url = NSURL(fileURLWithPath: filePath)

```

```

        let documentsPath = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask)[0]
        let destinationURL =
documentsPath.appendingPathComponent(url.lastPathComponent!)

        return destinationURL
    }

    func animateNavigationBar() {
        let options = BusyNavigationBarOptions()
        options.animationType = .stripes
        options.color = UIColor.white
        options.alpha = 0.8
        options.barWidth = 20
        options.gapWidth = 30
        options.speed = 1
        options.transparentMaskEnabled = true
        // Start animation
        self.navigationController?.navigationBar.start(options)
    }
}

```

Листинг A.6 – AddEventFormController.swift

```

import Foundation
import Eureka
import RealmSwift

class NativeEventFormViewController : FormViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        initializeForm()
    }

    private func initializeForm() {
        form +++
            TextRow("EventName").cellSetup { cell, row in
                cell.textField.placeholder = "Название события"
            }
            <<< DateTimeInlineRow("Starts") {
                $0.title = "Начало события"
                let date = Date()
                $0.value = date
                $0.minimumDate = date
                let formatter = DateFormatter()
                formatter.dateFormat = "dd/MM/YYYY HH:mm"
                $0.dateFormatter = formatter
            }
            .onChange { [weak self] row in
                let endRow: DateTimeInlineRow! = self?.form.rowBy(tag:
"Ends")

                endRow.minimumDate = row.value
                var startDate = row.value
                startDate?.hour = 0
                startDate?.minute = 0
                let timeInterval =
row.value?.timeIntervalSince(startDate!)
                endRow.maximumDate =
row.value?.addingTimeInterval(TimeInterval(24*60*60 - timeInterval! - 60))

```

```

        endRow.value = row.value?.addingTimeInterval(60*60)
        endRow.minimumDate = endRow.value
        endRow.updateCell()
    }
    .onExpandInlineRow { cell, row, _ in
        let color = cell.detailTextLabel?.textColor
        row.onCollapseInlineRow { cell, _, _ in
            cell.detailTextLabel?.textColor = color
        }
        cell.detailTextLabel?.textColor = cell.tintColor
    }
<<< DateTimeInlineRow("Ends") {
    $0.title = "Окончание события"
    let date = Date().addingTimeInterval(TimeInterval(60 * 60))
    $0.value = date
    $0.minimumDate = date
    let formatter = DateFormatter()
    formatter.dateFormat = "dd/MM/YYYY HH:mm"
    $0.dateFormatter = formatter

    var startDate = date
    startDate.hour = 0
    startDate.minute = 0
    let timeInterval = date.timeIntervalSince(startDate)

    $0.maximumDate =
date.addingTimeInterval(TimeInterval(24*60*60 - timeInterval - 60))
    }
    .onExpandInlineRow { cell, row, _ in
        let color = cell.detailTextLabel?.textColor
        row.onCollapseInlineRow { cell, _, _ in
            cell.detailTextLabel?.textColor = color
        }
        cell.detailTextLabel?.textColor = cell.tintColor
    }
<<< SwitchRow("switchRowTag") {
    $0.title = "Повтор"
    $0.value = false
}

+++ SelectableSection<ListCheckRow<String>>("Периодичность",
selectionType: .singleSelection(enableDeselection: false)) {
    $0.tag = "cycleOptions"

    $0.hidden = Condition.function(["switchRowTag"], { form in
        return !((form.rowBy(tag: "switchRowTag") as?
SwitchRow)?.value ?? false)
    })
}
<<< ListCheckRow<String> {

    $0.title = "Каждый день"
    $0.selectableValue = "day"
    $0.value = "Каждый день"
}
<<< ListCheckRow<String> {

    $0.title = "Каждую неделю"
    $0.selectableValue = "week"
    $0.value = nil
}

```

```

    }
    <<< ListCheckRow<String> {

        $0.title = "Каждый месяц"
        $0.selectableValue = "month"
        $0.value = nil
    }
    <<< ListCheckRow<String> {

        $0.title = "Каждый год"
        $0.selectableValue = "year"
        $0.value = nil
    }
+++ Section()
    <<< ButtonRow() { (row: ButtonRow) -> Void in
        row.title = "Добавить событие"
    }
    .onCellSelection { [weak self] (cell, row) in
        self?.showAlert()
    }
}

@IBAction func showAlert() {
    let alertController = UIAlertController(title: "OnCellSelection",
message: "Button Row Action", preferredStyle: .alert)
    let defaultAction = UIAlertAction(title: "OK", style: .default,
handler: nil)
    alertController.addAction(defaultAction)
    let startRow: DateTimeInlineRow! = self.form.rowBy(tag: "Starts")
    let endRow: DateTimeInlineRow! = self.form.rowBy(tag: "Ends")
    let eventName: TextRow! = self.form.rowBy(tag: "EventName")
    let cycle: SwitchRow! = self.form.rowBy(tag: "switchRowTag")
    var selectedCycle = ""

    if (cycle.value!){
        let cycleOption: SelectableSection<ListCheckRow<String>>! =
(self.form.sectionBy(tag: "cycleOptions") as!
SelectableSection<ListCheckRow<String>>)
        selectedCycle = cycleOption.selectedRow()!.selectableValue!
    }

    if let eventNameValue = eventName.value {
        let newEvent = UserEvent()
        newEvent.startDate = startRow.value!
        newEvent.endDate = endRow.value!
        newEvent.text =
"\(eventNameValue)\n\((newEvent.startDate.format(with: "HH:mm")) -
\((newEvent.endDate.format(with: "HH:mm")))"
        newEvent.isCycle = cycle.value!
        if (cycle.value!){
            switch selectedCycle {
                case "day":
                    newEvent.cycleType = UserEvent.CycleType.day.rawValue
                case "week":
                    newEvent.cycleType =
UserEvent.CycleType.week.rawValue
                case "month":
                    newEvent.cycleType =
UserEvent.CycleType.month.rawValue
                case "year":

```

```

        newEvent.cycleType =
UserEvent.CycleType.year.rawValue
        default:
        newEvent.cycleType =
UserEvent.CycleType.noCycle.rawValue
    }
}

let message = UserMessage()
message.messageId = UUID().uuidString
message.user = realm.objects(User.self).filter("senderId =
'Notifier'")[0]
message.sentDate = Date()
message.realmKind = "text"
message.value = "Событие было добавлено в календарь:
\(newEvent.text) \(newEvent.startDate.format(with: "dd/MM/YYYY"))"

    try! realm.write {
        realm.add(message)
    }
    try! realm.write {
        realm.add(newEvent)
    }
    self.navigationController?.popViewController(animated: true)
} else {
    let alertController = UIAlertController(title: "Ошибка", message:
"Введите название события", preferredStyle: .alert)
    let defaultAction = UIAlertAction(title: "OK", style: .default,
handler: nil)
    alertController.addAction(defaultAction)
    present(alertController, animated: true)
}
}
}

```

ПРИЛОЖЕНИЕ Б
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

17:22



admin

Регистрация нового аккаунта

Войти

Рисунок Б.1 – Вход в приложение

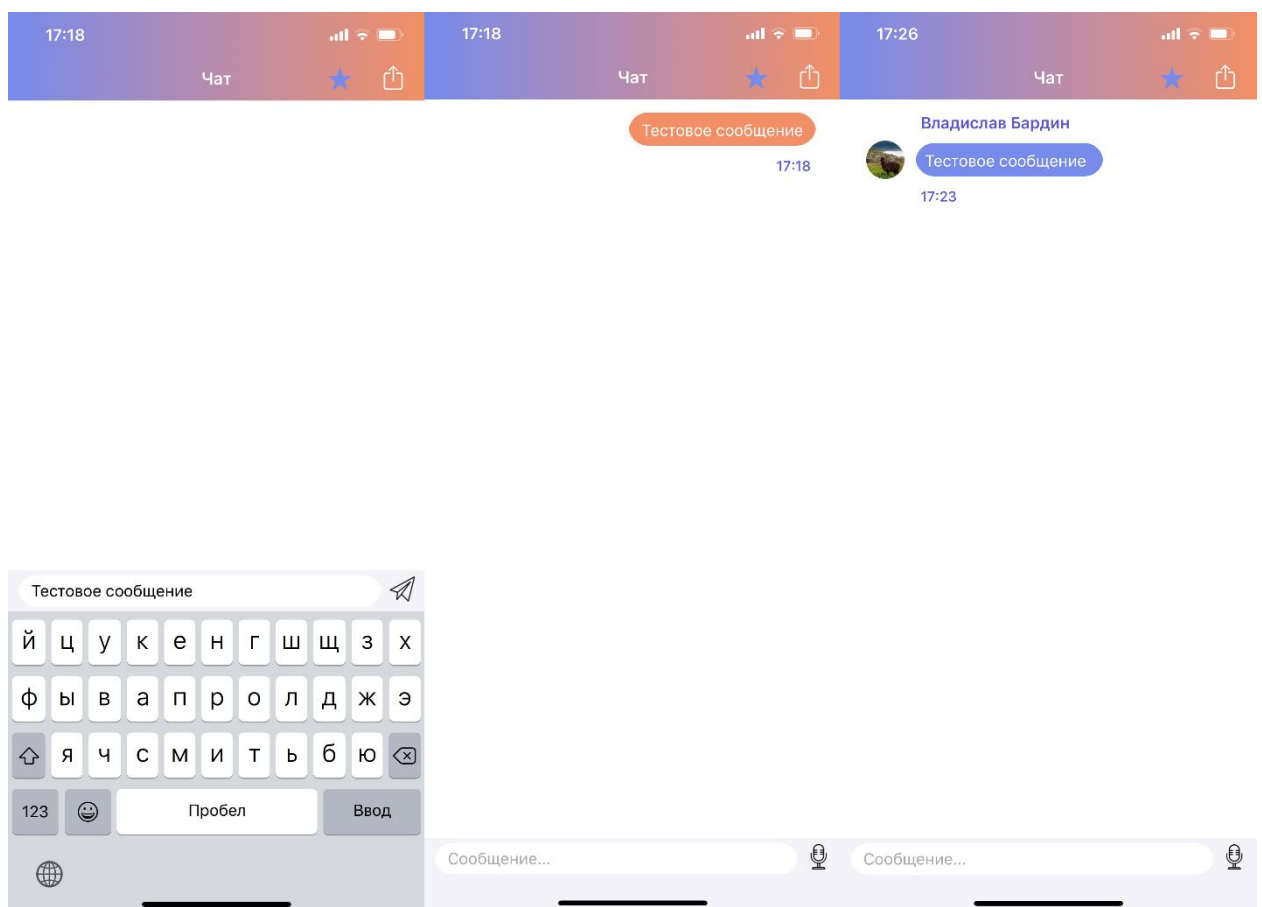


Рисунок Б.2 – Отправка текстового сообщения (отображение исходящего и входящего сообщения)

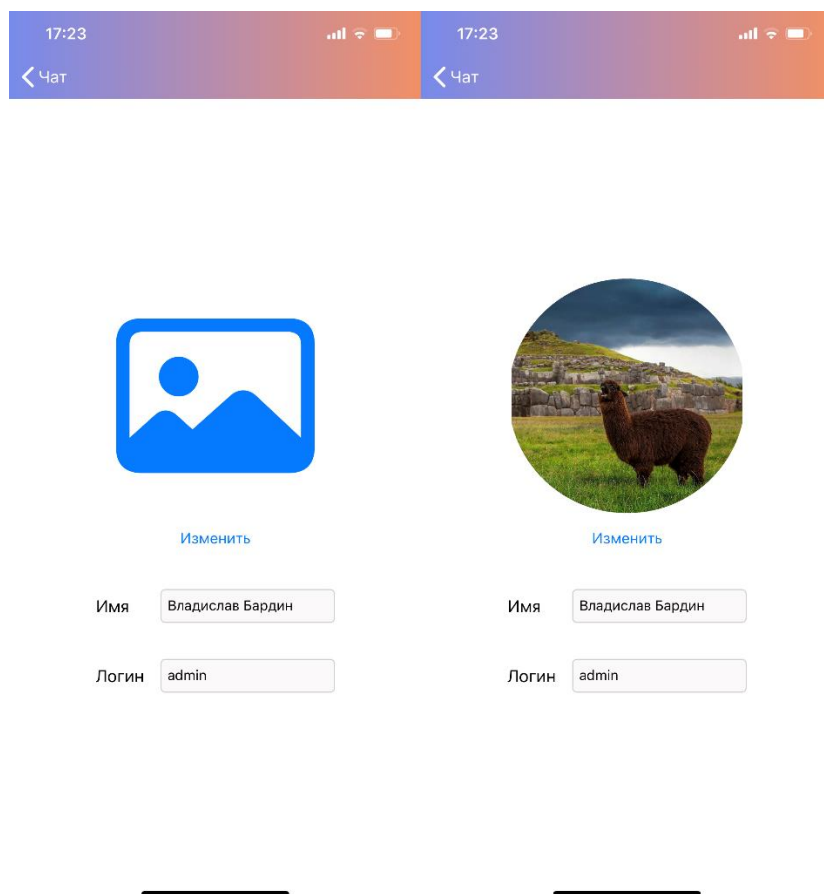


Рисунок Б.3 – Изменение фотографии пользователя

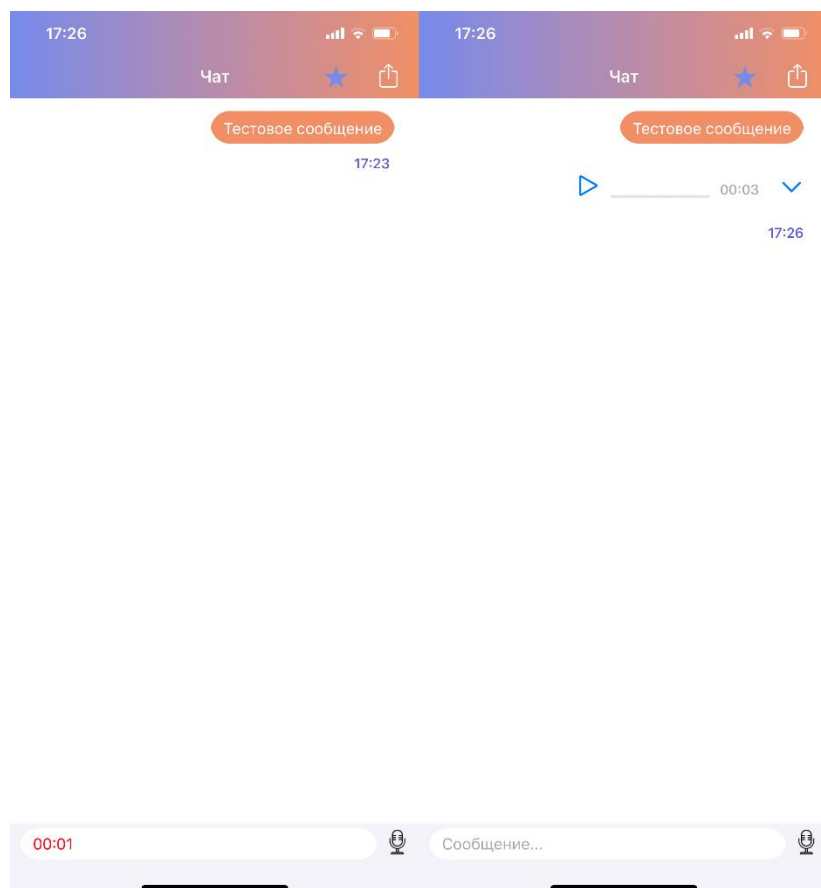


Рисунок Б.4 – Запись и отправка аудиосообщения

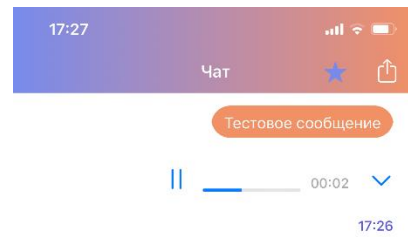


Рисунок Б.5 – Проигрывание аудиосообщения

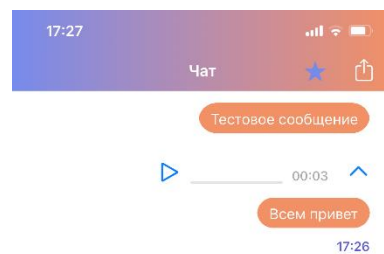


Рисунок Б.6 – Просмотр текстового содержания аудиосообщения

The screenshot displays a mobile application interface with two main sections: a settings panel on the left and a calendar/timeline view on the right.

Left Panel (Settings):

- Header:** 17:27, signal strength, Wi-Fi, and battery icons.
- Navigation:** < События
- Event Name:** Повторяющееся событие
- Start Time:** 12/05/2020 17:27
- End Time:** 12/05/2020 18:27
- Repeat:** Повтор (toggle switch is ON)
- Periodicity:** ПЕРИОДИЧНОСТЬ
 - Каждый день (selected with a blue checkmark)
 - Кажую неделю
 - Каждый месяц
 - Каждый год
- Footer:** Добавить событие

Right Panel (Calendar/Calendar View):

- Header:** 17:28, signal strength, Wi-Fi, and battery icons.
- Navigation:** < Чат, События, Править, +
- Calendar Grid:** Shows days of the week (ПН, ВТ, СР, ЧТ, ПТ, СБ, ВС) and dates (11, 12, 13, 14, 15, 16, 17). The date 12 is highlighted in red, and 13 is highlighted in black.
- Event Details:** вторник, 12 мая 2020 г. and среда, 13 мая 2020 г.

Timeline View (Bottom):

- Left Timeline:** Shows a repeating event "Повторяющееся событие 17:27 - 18:27" on May 12, 2020, from 17:28 to 18:00.
- Right Timeline:** Shows a repeating event "Повторяющееся событие 17:27 - 18:27" on May 13, 2020, from 17:00 to 18:00.

Рисунок Б.8 – Добавление события в календарь и просмотр в разные даты

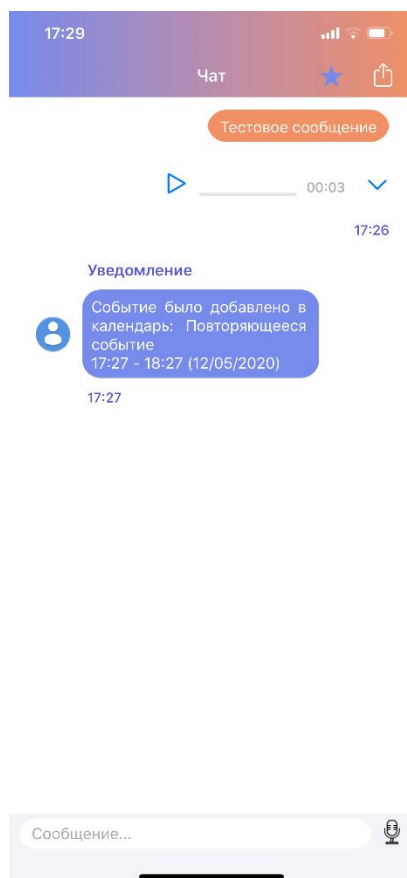


Рисунок Б.9 – Уведомление в чате о добавлении события

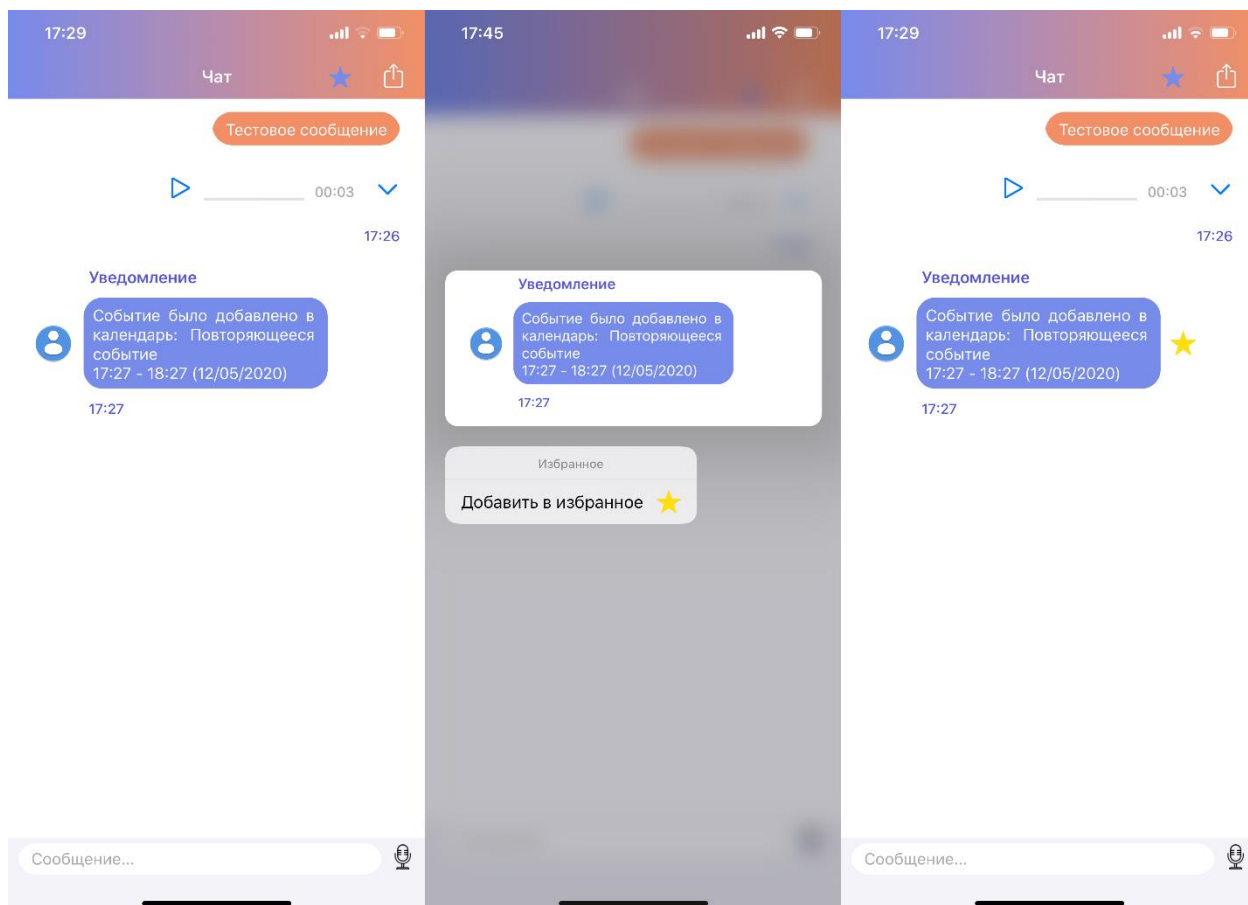


Рисунок Б.10 – Пометка сообщения избранным

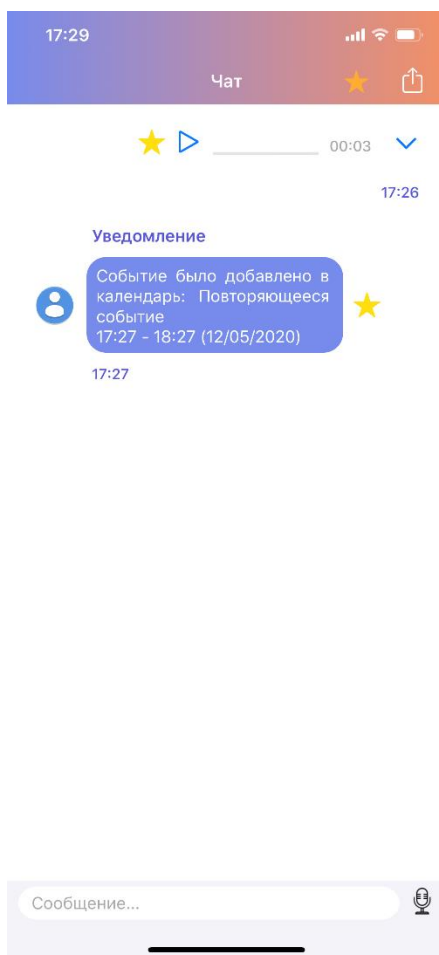


Рисунок Б.11 – Фильтр избранных сообщений