МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования

информационных систем

**Разработка новостного агрегатора KeepMePosted с клиентом Telegram**

Бакалаврская работа

Направление 02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль «Информационные системы и базы данных»

|  |  |  |  |
| --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м. н., проф | М.А. Артемов |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | В.А Лазарев |
| Руководитель | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м. н., проф | М.А. Артемов |

Воронеж 2020

# Аннотация

Данная работа посвящена проектированию и разработке новостного агрегатора *KeepMePosted* для предоставления доступа к новостям, «твитам» и погоде с различных интернет-ресурсов. Данное приложение было создано на базе фреймворков *Spring* и *Angular* и построено на микросервисной архитектуре.

В ходе работы рассмотрены основные этапы разработки веб-приложения, проведен анализ существующих решений, сформированы требования к проектируемой системе, описаны модели данных.

# Содержание

[Аннотация 2](#_Toc41665984)

[Содержание 3](#_Toc41665985)

[Введение 4](#_Toc41665986)

[1. Постановка задачи 5](#_Toc41665987)

[2. Анализ задачи 6](#_Toc41665988)

[2.1. Анализ существующих решений 6](#_Toc41665989)

[2.2. Анализ функциональности приложения 7](#_Toc41665990)

[2.3. Анализ структуры приложения 9](#_Toc41665991)

[2.4. Анализ модели данных 11](#_Toc41665992)

[2.5. Анализ пользовательского интерфейса 13](#_Toc41665993)

[3. Средства реализации 16](#_Toc41665994)

[4. Требования к аппаратному и программному обеспечению 17](#_Toc41665995)

[5. Интерфейс пользователя 18](#_Toc41665996)

[5.1. Навигация страниц приложения 18](#_Toc41665997)

[5.2. Вход и регистрация 20](#_Toc41665998)

[5.3. Страницы пользовательского интерфейса 21](#_Toc41665999)

[6. Реализация 26](#_Toc41666000)

[6.1. Структура приложения 26](#_Toc41666001)

[6.2. Модели данных 28](#_Toc41666002)

[6.3. Структура функциональных блоков 31](#_Toc41666003)

[7. План тестирования 34](#_Toc41666004)

[Заключение 37](#_Toc41666005)

[Список литературы 38](#_Toc41666006)

[Приложение 1. Листинг конфигурационных файлов проектов 39](#_Toc41666007)

[Приложение 2. Примеры коллекции данных 41](#_Toc41666008)

[Приложение 3. Листинг классов данных 43](#_Toc41666009)

# Введение

В современном мире темп городской жизни невероятно высок, и быть в курсе всех последних новостей, когда весь день проходит в спешке или «на ногах» просто-напросто невозможно. Ситуация осложняется тем, что большинство новостных источников — радио, телевизор, новостные сайты для компьютеров не очень удобны для поглощения контента, когда вы идете по улице, едете в метро или другом общественном транспорте.

С развитием смартфонов и приложений для них изучение последних новостей стало намного проще, но у каждого новостного ресурса появилось свое приложение, причем новостные порталы зачастую бывают узкоспециализированными, то есть пользователям приходится скачивать, устанавливать, настраивать и пользоваться не одним приложением, а сразу несколькими, что крайне неудобно.

Новостные агрегаторы позволяют людям существенно сократить время поиска необходимой информации, избежать засорения смартфона однотипными программами, а также дает возможность настраивать интересующие темы новостей, новостные порталы, откуда пользователь хочет получать обновления.

Но это все еще остается сторонним приложением. Заметно, как развивается такое направление как боты и каналы в социальных сетях и мессенджерах, поскольку в одном приложении с помощью различных ботов можно делать почти все, что угодно. Однако текущие боты имеют интеграцию с малым количеством сервисов, без возможности гибкой настройки и системы оповещений.

Таким образом, целесообразно создать приложение, которое позволит своевременно получать обновления по указанным пользователем темам, и может предоставлять более обширную информацию.

# Постановка задачи

Разработать веб-приложение *KeepMePosted*, реализующее систему для сбора новостей и прочей информации как через открытое API, так и с помощью анализа RSS-каналов. Клиенты программы должны быть представлены в виде *Telegram* бота и веб-страницы, предоставляющее следующие возможности:

* возможность просмотра самых свежих новостей;
* добавление и удаление категорий новостей, по которым пользователь будет получать обновления;
* возможность блокировать и снимать блокировку с новостных ресурсов;
* возможность просмотра самых новых *твитов* социальной сети Twitter;
* добавление и удаление отслеживаемых пользователей *Twitter* и хэштегов;
* возможность просмотра текущей погоды в указанных пользователем населенных пунктах;
* добавление текущей геолокации пользователя, преобразование ее в город, добавление в список отслеживаемых населенных пунктов;
* добавление и удаление городов для получения прогноза погоды;
* возможность просмотра новостей, *твитов* и погоды применяя пользовательские настройки, так и без них;
* включение и отключение оповещений, настройка их частоты отдельно для каждого из сервиса:

Для проектирования веб-интерфейса использовать библиотеку *Angular Material*, как современный UI-фреймворк.

# Анализ задачи

## **Анализ существующих решений**

В настоящее время существует большое количество решений, моделирующих работу новостного агрегатора. Для сравнительного анализа был введен следующий перечень критериев:

* наличие веб-страницы;
* возможность просмотра последних новостей;
* наличие бота в социальной сети или мессенджере (*ВКонтакте*, *Telegram*);
* отбор новостей по нескольким категориям;
* рейтинг популярных новостей в разрезе некоторого промежутка времени (день, неделя, месяц и т. д.);
* наличие сторонних информационных ресурсов (погода, *Twitter*, курс валют).

1. Telegram-бот @ONOMediaScopeBot

На рис. 2.1. показано главное меню *Telegram*-бота *@ONOMediaScopeBot*.

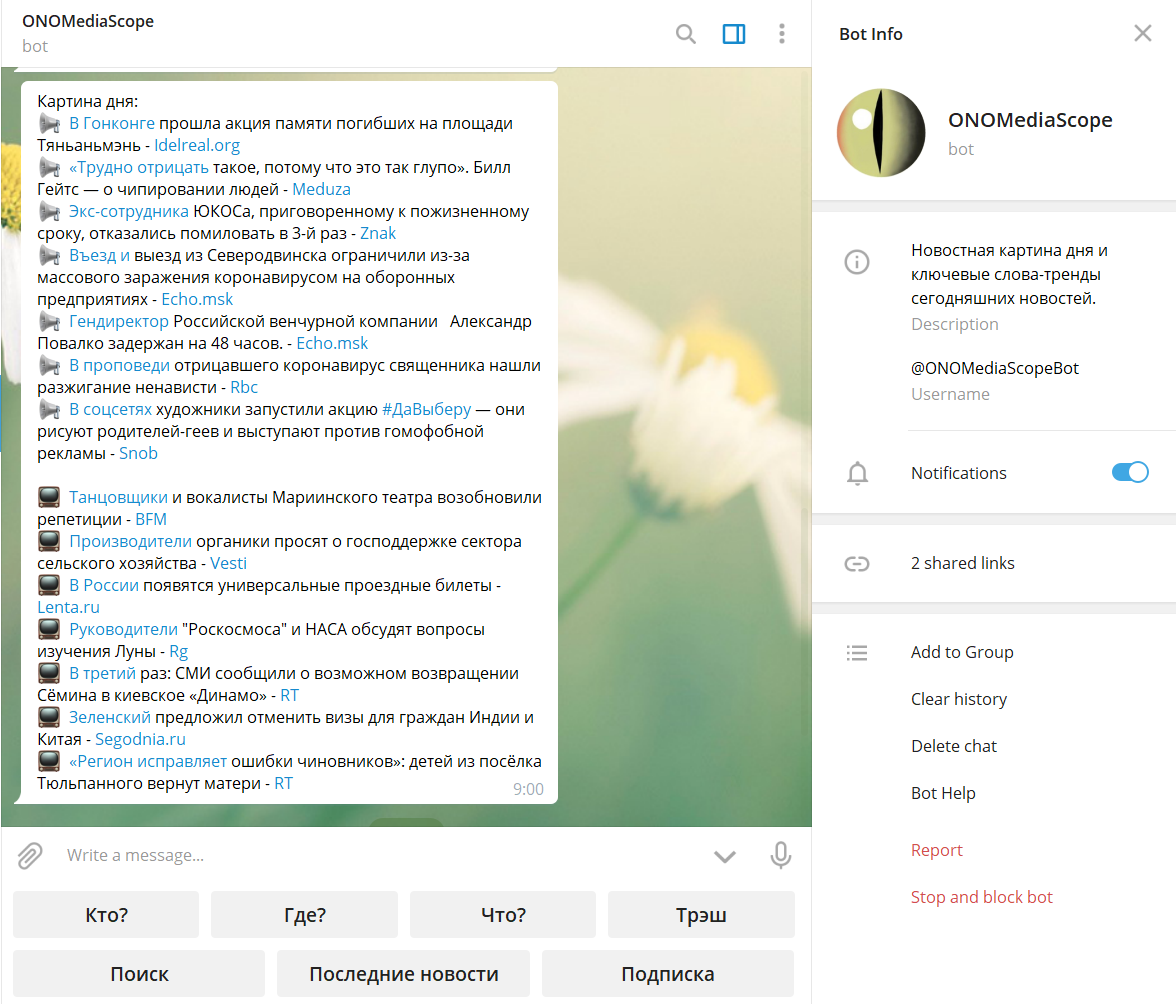


Рис. 2.1. Главное меню Telegram-бота @ONOMediaScopeBot

Для добавления отслеживаемой темы необходимо кликнуть на кнопку *Подписка* и ввести необходимую тему. Ровно такие же действия надо произвести, чтобы отписаться от категории. При этом просмотр новостей не будет учитывать того, на что пользователь подписан.

Кнопка *Последние новости* выводит 14 последних новостей одним сообщением, на экран смартфона такое сообщение не помещается, что является просчет со стороны дизайна пользовательского интерфейса.

Остальные кнопки никак не описаны, бот не выдает подсказок, что осложняет его использование. Веб-страницы, рейтинга новостей, интеграции с какими-то особенными сервисами данное приложение также не имеет.

1. Сайт и мобильное приложение mediametrics.ru

На рис. 2.2 представлена главная страница сервиса *mediametics.ru*.

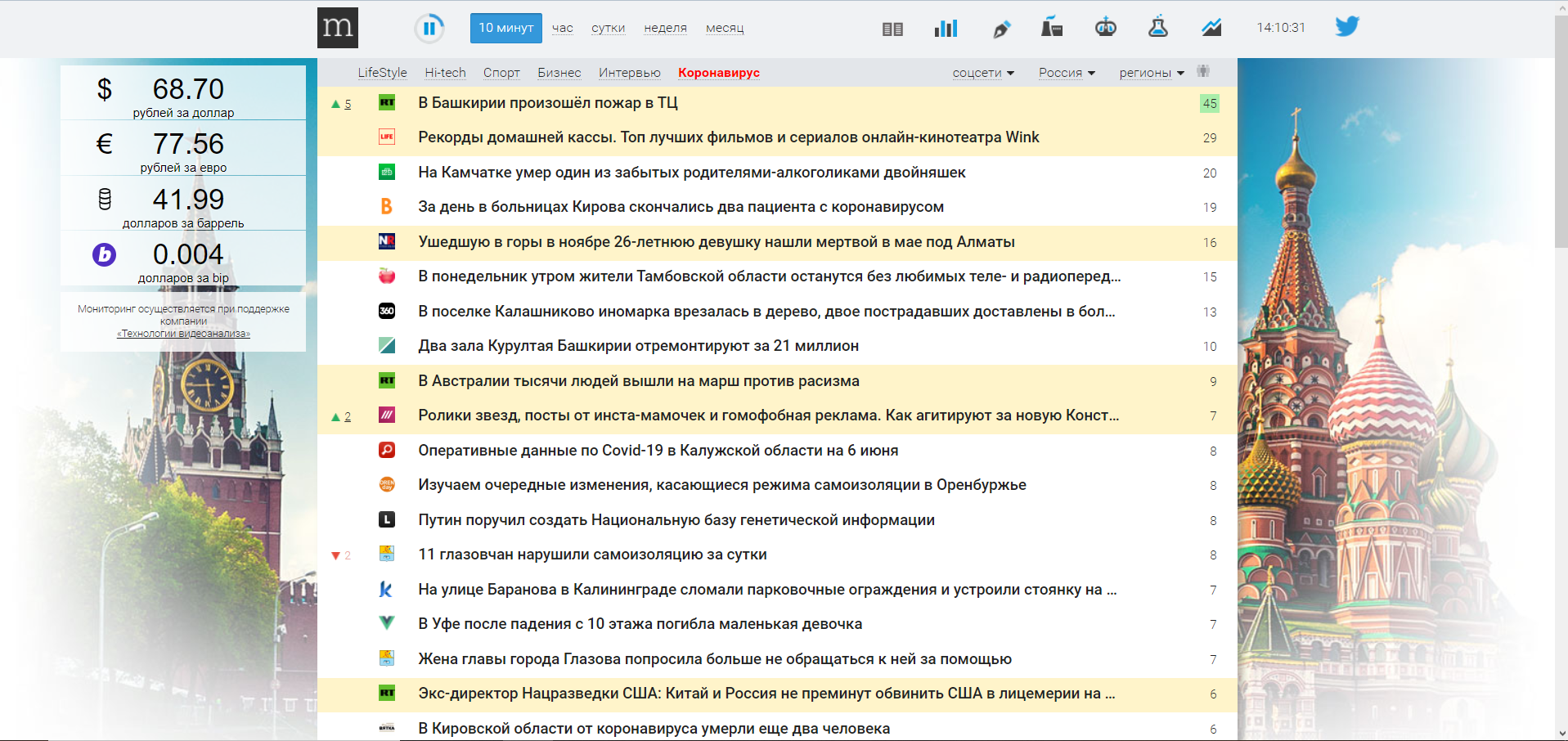


Рис. 2.2. Главная страница сервиса mediametrcis.ru

На веб-странице есть предопределенный перечень тем, по которым можно фильтровать новости, однако добавлять какие-то собственные категории нельзя.

Также, этот сервис имеет собственный канал *Telegram*, но настраивать отслеживаемые темы тоже нельзя. Существует мобильное официальное приложение *Mediametrics*, доступное для ОС *iOS*, которое представлено на рис. 2.3 и рис. 2.4.

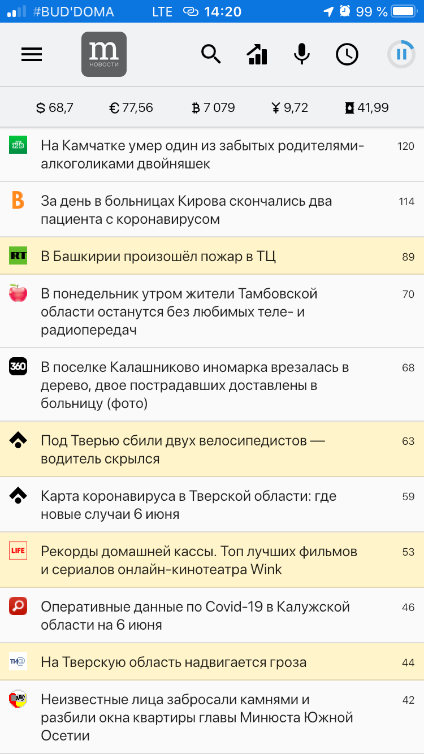


Рис. 2.3. Главная страница мобильного приложения Mediametric

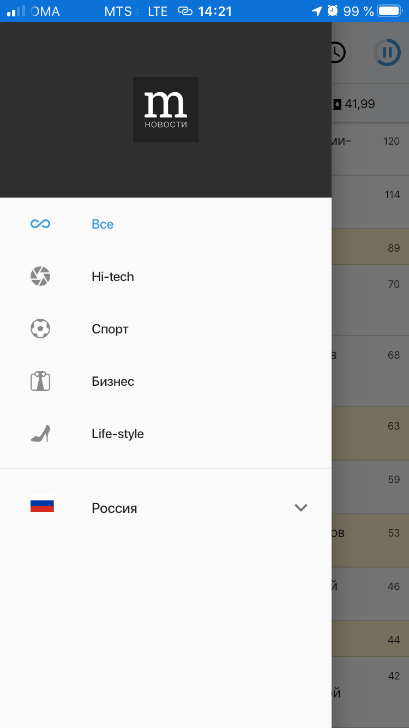


Рис. 2.4. Предопределенны темы мобильного приложения Mediametric

Мобильное приложение полностью отражает работу основного сайта, также имеется рейтинг новостей в разрезе некоторого промежутка времени, интеграция с сервисом курса валют, имеется поиск по заданному слову. Но приложение не поддерживает отправку оповещений.

1. Сервис Яндекс.Новости

На рис. 2.5 представлена главная страница портала *Яндекс.Новости*. Отличительной чертой данного портала является, что отображаются не только популярные новости, но и на основе ваших последних запросов в браузере. Однако, нет никаких рейтингов новостей, нет мобильного приложения и бота. Данный сервис очень удобен для потребления контента посредством ПК или ноутбука. Из минусов стоит выделить, что при загрузке страницы отображается реклама.

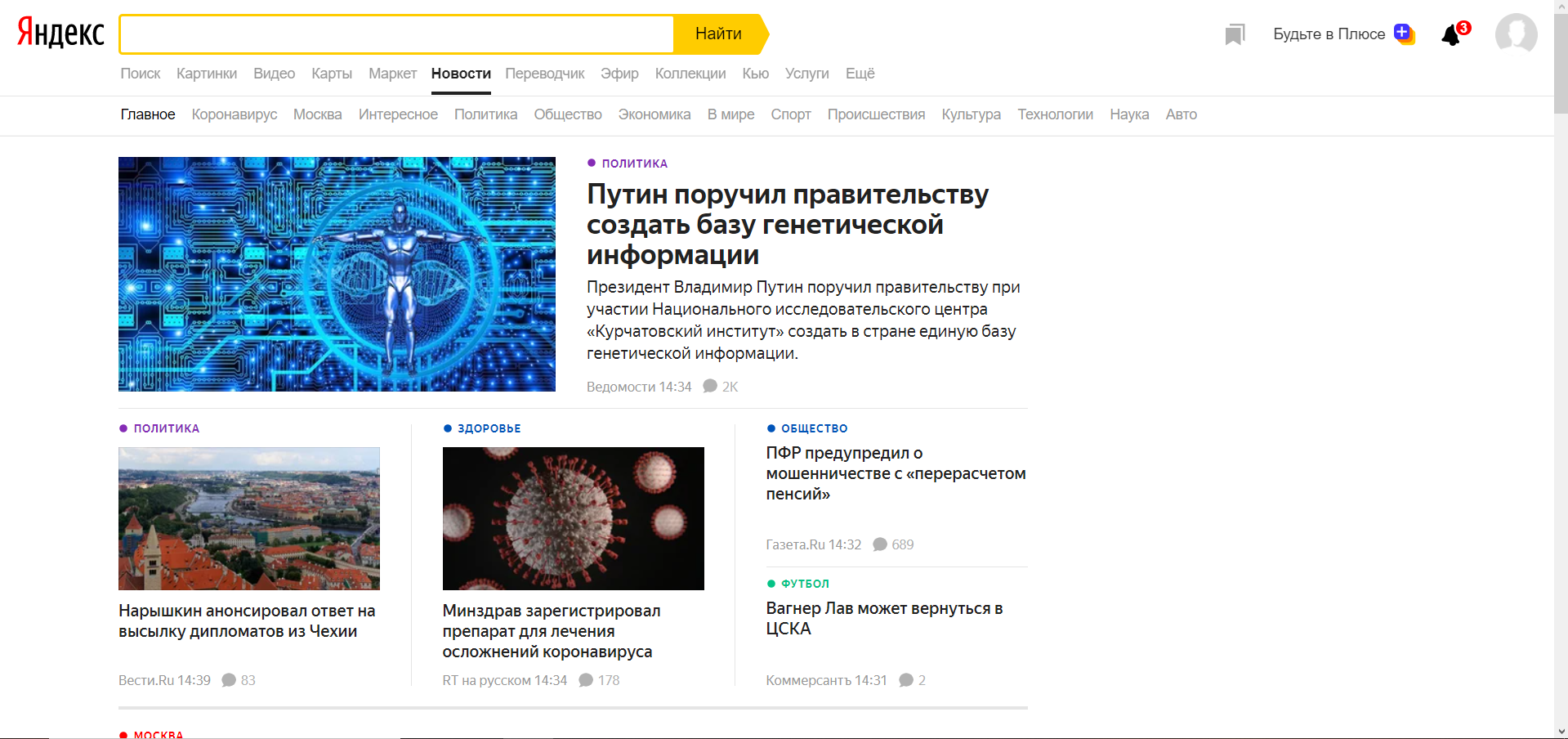


Рис. 2.5. Главная страница портала Яндекс.Новости

1. Сервис Яндекс.Дзен

На рис. 2.6 представлена главная страница сервиса *Яндекс.Дзен*.

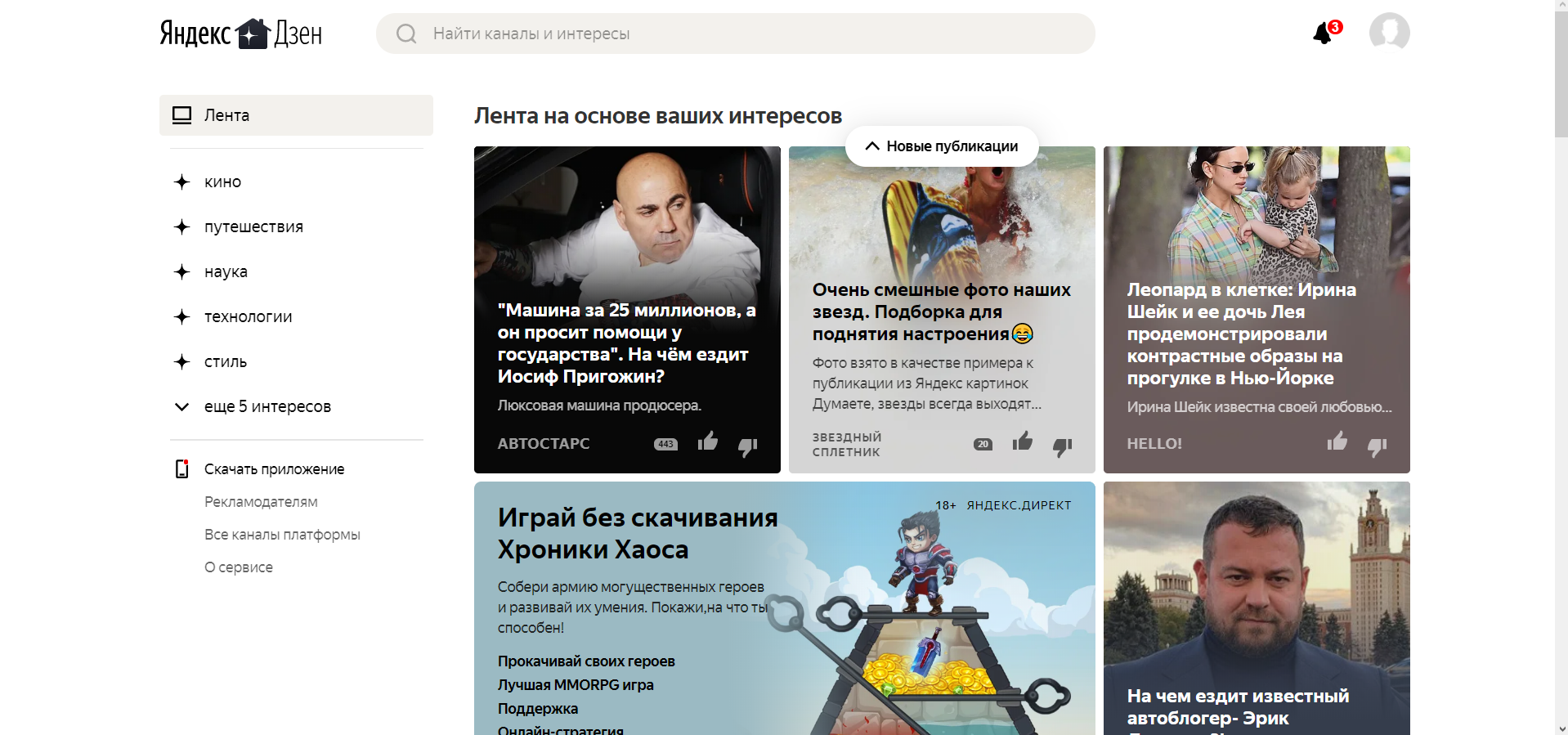


Рис. 2.6. Главная страница сервиса Яндекс.Дзен

Как и в случае с порталом *Яндекс.Новости*, *Яндекс.Дзен* отображает новости на основе ваших интересов и запросов в браузере. Но имеет отдельное приложение под ОС *iOS*. Однако в нем также нет гибкости в настройке отображаемых новостей, нет оповещений, интеграций со сторонними сервисами, но есть реклама, занимающая большую часть полезного пространства пользовательского интерфейса.

1. Результаты анализа существующих решений

Результаты анализа представлены в таблице 2.1.

Приложений подобного характера в данной бизнес-области было найдено не очень много. Большинство из них представляет собой собственные сайты и мобильные приложения, показывающие только собственные новости. Чтобы ознакомиться с остальными пользователю необходимо будет переходить на страницу другого информационного портала, что не очень удобно. Либо это приложения, которые не поддерживают подписку на определенные темы и не отправляют оповещения.

Таблица 2.1. Сравнительный анализ существующих решений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Характеристики | *@ONOMediaScopeBot* | *mediametrics.ru* | *Яндекс.Новости* | *Яндекс.Дзен* |
| Наличие веб-сайта | — | + | + | + |
| Просмотр последних новостей | + | + | + | + |
| Наличие бота в соц. сети или мессенджере (*ВКонтакте*, *Telegram*) | + | — | — | — |
| Доступ на русском языке | + | + | + | + |
| Отбор новостей по нескольким категориям | + | — | — | — |
| Рейтинг популярных новостей в размере промежутка времени (день, неделя, месяц) | + | + | — | — |
| Наличие сторонних информационных ресурсов (погода, *Twitter*, курс валют) | — | + | + | — |

При анализе существующих решений не было найдено существующего решения, удовлетворяющего всем критериям.

## **Анализ функциональности приложения**

На основе проведенного анализа выделена необходимая для реализации функциональность приложения:

* автоматическая регистрация пользователей *Telegram* на этапе подключения к боту;
* настройка интервала и активности оповещений для сервиса новостей, сервиса *Twitter* и сервиса погоды;
* возможность просмотра списка новостей, *твитов*, погоды в форме слайдера (доступен переход к предыдущей и к следующей записям), перемещение к самой «свежей» записи;
* возможность отключать настройки пользователя при просмотре информации.

Сервис новостей на клиенте *Telegram* должен включать:

* ввод и модификация отслеживаемых категорий новостей;
* ввод и модификация новостных ресурсов, с которых пользователь не будет получать информацию;

Сервис *твитов* соц. сети *Twitter* на клиенте *Telegram* должен включать:

* ввод и модификация отслеживаемых хештегов, пользователей;

Сервис погоды должен включать на клиенте *Telegram*:

* возможность добавления населенного пункта в список отслеживаемых с помощью функции *Поделиться текущей локацией* (актуально только для смартфонов);
* ввод и удаление отслеживаемых городов по их наименованию;

## **Анализ выбранных источников информации**

В качестве источников новостей в приложении *KeepMePosted* будут использоваться следующие новостные ресурсы:

* *РБК*;
* *ТАСС*;
* *Вести.Ru*;
* *Газета «Ведомости»*;
* *Газета «Известия»*;
* *Lenta.ru*.

Данные ресурсы были выбраны так как являются наиболее популярными новостными порталами. Все эти источники не обладают открытым API, однако есть RSS-рассылка. Для получения интересующей нас информации будем использовать фреймворк *ROME*, который позволяет удобно и быстро парсить содержимое RSS-каналов в пригодную для обработки структуру данных.

Для поиска информации в социальной сети *Twitter* существует специальная API для разработчиков приложений. С помощью библиотеки в фреймворке *Spring*, можно легко интегрироваться и удобно пользоваться возможностями *Twitter*.

Для рассылки прогноза погоды используется сервис *Яндекс.Погода*, а для определения населенного пункта по локации пользователя используется *Яндекс.Геокодер*. Оба сервиса предоставляют доступ по API, возвращают ответ в формате JSON, который парсится в пригодную для обработки структуру данных с помощью фреймворка *Jackson*.

## **Анализ модели данных**

Для корректной работы с данными необходимо реализовать реляционную базу данных. Предполагаемая логическая модель данных уровня сущностей приведена на рис. 2.7.

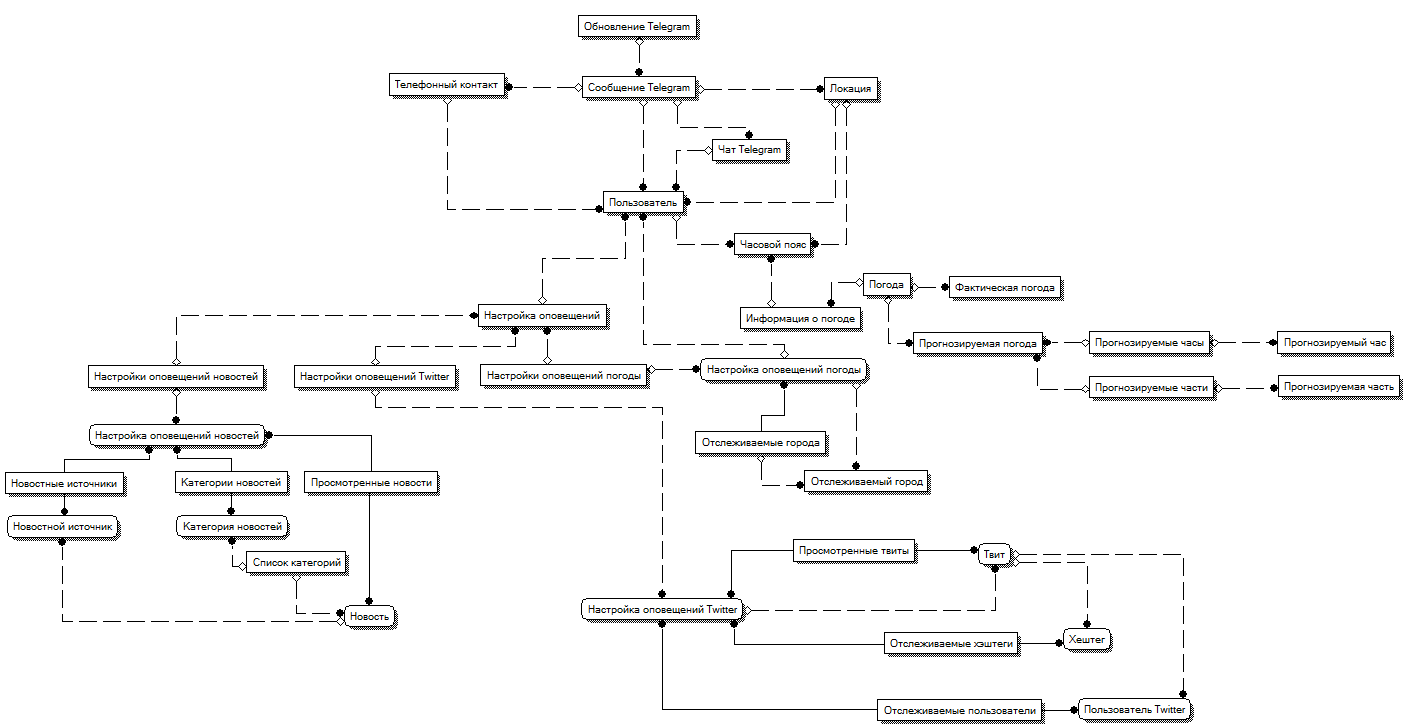


Рис. 2.7. Логическая модель данных

Предполагаемые сущности, которые следует спроектировать:

1. *Обновление* – сущность, хранящая все полученные от бота обновления *Telegram*. Имеет атрибут *Дата создания*, *ID\_сообщения\_Telegram*.
2. *Телефонный контакт* – таблица, хранящая контактные телефоны пользователей, которые используют *Telegram*-бот. Имеет атрибуты *Дата создания*, *Имя*, *Фамилия*, *Телефонный номер*, *ID\_Пользователя*.
3. *Сообщение Telegram* – содержит информацию обо всех сообщениях, присланных ботом. Имеет следующие атрибуты: *Дата создания*, *Текст сообщения*, *ID\_Чата\_Telegram*, *ID\_Телефонного\_контакта*, *ID\_Пользователя*, *ID\_Локации*.
4. *Локация* – сущность, которая хранит локации всех пользователей, которые используют *Telegram*-бот. Имеет атрибуты *Дата создания*, *Город* (определяется с помощью сервиса *Яндекс.Геокодер* при создании записи), *Широта*, *Долгота*, *ID\_Часового\_Пояса*, *ID\_Пользователя*.
5. *Чат Telegram*– таблица, хранящая информацию о каждом чате бота с пользователем *Telegram*. В качестве атрибутов выступают *Дата создания*, *Чат канала*, *Чат группы*, *Чат супергруппы*, *Чат с пользователем*, *ID\_Пользователя*.
6. *Пользователь*– таблица со всеми пользователями *Telegram*-бота. Имеет следующие атрибуты: *Дата создания*, *Это бот*, *E-mail*, *Имя*, *Фамилия*, *Язык*, *Телефонный номер*, *Зарегистрирован*, *Статус*, *Никнейм*, *ID\_Локации*, *ID\_Часового\_Пояса*.
7. *Часовой пояс* – справочник со всеми часовыми поясами, заполняется по мере выполнения запросов в сервис погоды. Имеет атрибуты *Аббревиатура*, *Признак летнего времени*, *Наименование*, *Часовой пояс в секундах от UTC*.
8. *Информация о погоде* –хранит в себе побочную информацию при запросе в сервис *Яндекс.Погода*. Имеет атрибуты *Широта*, *Долгота*, *Норма давления мм*, *Норма давления па*, *URL*, *ID\_Часового пояса*.
9. *Погода* – основная сущность при запросе погоды. Имеет атрибуты *Время сервера в unixtime*, *Время сервера в UTC*, *ID\_Информации\_о\_погоде*, *ID\_Фактической\_Погоды*, *ID\_Прогнозируемой\_погоды*.
10. *Фактическая погода* – таблица, хранящая фактическую информацию о погоде на определенный момент времени. Имеет атрибуты *Температура*, *Ощущаемая температура*, *Температура воды*, *Код иконки*, *Код расшифровки погодного состояния*, *Скорость ветра*, *Скорость порывов ветра*, *Направление ветра*, *Давление мм*, *Давление па*, *Влажность*, *Время суток*, *Это полярный день*, *Наименование сезона*, *Время замера погодных условий в unixtime*, *Тип Осадков*, *Сила осадков*, *Облачность*.
11. *Прогнозируемая погода* – таблица, хранящая прогнозируемую информацию о погоде на определенный момент времени. Имеет следующие атрибуты: *Дата прогноза*, *Дата прогноза в unixtime*, *Порядковый номер недели*, *Время восхода Солнца*, *Время заката Солнца*, *Код фазы Луны*, *Текстовый код для фазы Луны*, *ID\_прогнозирумых\_часов*, *ID\_прогнозируемых\_частей*.
12. *Прогнозируемые часы* – табличная часть, первичным ключами которой выступают *ID\_Прогнозируемой\_погоды* и *ID\_Прогнозируемого\_часа*.
13. *Прогнозируемый час*– сущность, хранящая прогноз на определенный час. Имеет атрибуты *Значение часа*, *Время прогноза в unixtime*, *Температура*, *Ощущаемая температура*, *Код иконки*, *Код расшифровки погодного описания*, *Скорость ветра*, *Скорость порывов ветра*, *Направление ветра*, *Давление мм*, *Давление па*, *Влажность*, *Прогнозируемое количество осадков*, *Прогнозируемый период осадков*, *Тип осадков*, *Сила осадков*, *Облачность*.
14. *Прогнозируемые части* – табличная часть, первичным ключами которой выступают *ID\_Прогнозируемой\_погоды* и *ID\_Прогнозируемой\_части*.
15. *Прогнозируемая часть*– таблица, хранящая детальную информацию о погоде на каждый час. Имеет атрибуты *Наименование*, *Источник*, *Минимальная температура*, *Максимальная температура*, *Средняя температура*, *Ощущаемая температура*, *Код иконки*, *Код расшифровки погодного описания*, *Время суток*, *Это полярный день*, *Скорость ветра*, *Направление ветра*, *Давление мм*, *Давление па*, *Влажность*, *Прогнозируемое количество осадков*, *Прогнозируемый период осадков*, *Тип осадков*, *Сила осадков*, *Облачность*.
16. *Настройка оповещений* – основная таблица для сервиса оповещения пользователей. Имеет следующие атрибуты: *Оповещения включены*, *Количество оповещений в день*, *Последнее оповещение*, *Интервал оповещений*, *ID\_Сервиса*, *ID\_Пользователя*.
17. *Настройки оповещений погоды* – табличная часть, первичным ключами которой выступают *ID\_Настройки\_Оповещений* и *ID\_Настройки\_Оповещений\_Погоды*.
18. *Настройка оповещений погоды* – таблица, хранящая настройки для оповещения пользователя *Telegram* для сервиса погоды. Имеет атрибуты *Дата создания последнего просмотренного города*, *ID\_Последнего\_Просмотренного\_Города*, *ID\_Настройки оповещений*, *ID\_Пользователя*.
19. *Отслеживаемые города*– табличная часть, первичным ключами которой выступают *ID\_Настройки\_Оповещений\_Погоды* и *ID\_Отслеживаемого города*.
20. *Отслеживаемый город*– сущность, хранящая информацию о городах. Имеет атрибуты *Наименование*, *Широта*, *Долгота*, *Дата создания*.
21. *Настройки оповещений новостей*– табличная часть, первичным ключами которой выступают *ID\_Настройки\_Оповещений* и *ID\_Настройки\_Оповещений\_Новостей*.
22. *Настройка оповещений новостей*– таблица, хранящая настройки для оповещения пользователя *Telegram* для сервиса новостей. Имеет атрибуты *Дата создания последней просмотренной новости*, *ID\_Последней\_Просмотренной\_Новости*, *ID\_Настройки\_оповещений*, *ID\_Пользователя*, *Включены пользовательские настройки*.
23. *Просмотренные новости*– табличная часть, первичным ключами которой выступают *ID\_Новости* и *ID\_Настройки\_Оповещений\_Новостей*.
24. *Новость*– основная таблица для сервиса новостей. Имеет атрибуты *Автор*, *Количество просмотров*, *Дата создания*, *Описание*, *Ссылка*, *Ссылка на фото*, *Дата публикации*, *Заголовок*, *URL*, *ID\_Источника*.
25. *Новостные источники*– табличная часть, первичным ключами которой выступают *ID\_Новостного\_Источника* и *ID\_Настройки\_Оповещений\_Новостей*.
26. *Новостной источник*– сущность, хранящая все источники новостей, с которых поступает информация. Содержит атрибуты *Дата создания*, *Дата последнего обновления*, *Ссылка*, *Ссылка на логотип*, *Наименование*, *Наименование источника*.
27. *Категории новостей*– табличная часть, первичным ключами которой выступают *ID\_Категории\_новостей* и *ID\_Настройки\_Оповещений\_Новостей*.
28. *Категория новостей*– сущность, хранящая все категории новостей, которые есть в базе данных. Имеет следующие атрибуты: *Дата создания*, *Наименование*.
29. *Список категорий*– табличная часть, первичным ключами которой выступают *ID\_Категории\_новостей* и *ID\_Новости*.
30. *Настройки оповещений Twitter*– табличная часть, первичным ключами которой выступают *ID\_Настройки\_Оповещений* и *ID\_Настройки\_Оповещений\_Twitter*.
31. *Настройка оповещений Twitter*– таблица, хранящая настройки для оповещения пользователя *Telegram* для сервиса *Twitter*. Имеет атрибуты *Дата создания последнего просмотренного твита*, *ID\_Последнего\_Просмотренного\_Твита*, *ID\_Настройки\_оповещений*, *ID\_Пользователя*, *Включены пользовательские настройки*.
32. *Просмотренные твиты* – табличная часть, первичным ключами которой выступают *ID\_Твита* и *ID\_Настройки\_Оповещений\_Twitter*.
33. *Твит*– основная таблица для сервиса *Twitter*. Имеет атрибуты *Дата создания*, *Количество лайков*, *Отправитель*, *ID\_Отправителя*, *ID\_строкой*, *Ссылка*, *Ссылка на изображение профиля*, *Количество ретвитов*, *Текст*, *Получатель*, *ID\_Хэштега*, *ID\_Пользователя\_Twitter*.
34. *Отслеживаемые хэштеги*– табличная часть, первичным ключами которой выступают *ID\_Хэштега* и *ID\_Настройки\_Оповещений\_Twitter*.
35. *Хэштег*– сущность, хранящая все отслеживаемые хэштеги *Twitter*, по которым пользователи получают информацию. Содержит атрибуты *Дата создания*, *Хэштег*.
36. *Отслеживаемые пользователи*– табличная часть, первичным ключами которой выступают *ID\_Пользователя\_Twitter* и *ID\_Настройки\_Оповещений\_Twitter*.
37. *Пользователь Twitter*– сущность, хранящая всех отслеживаемых пользователей *Twitter*, по которым пользователи получают информацию. Содержит атрибуты *Дата создания*, *Никнейм*.

## **Анализ структуры приложения**

В приложении будет использоваться MVC-подход к разработке  
(Model-View-Controller).

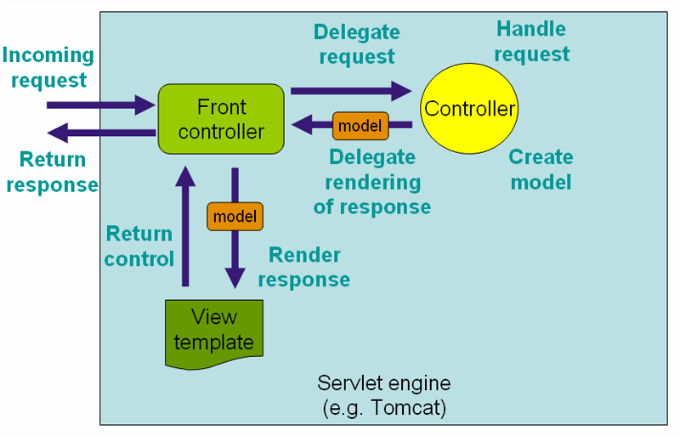


Рис. 2.8. Взаимодействие частей приложения

Это означает, что логика ввода, бизнес-логика и логика пользовательского интерфейса разделены, но обеспечена свободная связь между ними.

Модель объединяет данные приложения, все вместе они будут состоять из POJO (Бины *Java*, управляемые *Spring IoC* контейнером).

Вид отвечает за отображение данных модели.

Контроллер обрабатывает запрос пользователя, создает модель и передает ее для отображения в вид.

# Средства реализации

Для разработки приложения выбраны следующие программные средства:

* интерактивная среда разработки – *IntelliJ IDEA 2020.1.2*;
* язык программирования – *Java* [2];
* система контроля версий – *Git*;
* набор инструментов разработки – *JDK 1.8.1*;
* фреймворк сервера – *Spring Boot Framework*;
* язык разработки веб-интерфейса – *TypeScript*;
* фреймворк веб-интерфейса –*Angular*;
* СУБД – *MySQL*;

Фреймворк *Spring Boot* был выбран в качестве основного инструмента разработки, так как он является легким фреймворком для построения веб-приложений на языке *Java*. *MySQL* используется в качестве СУБД, потому что является одной из самых передовых реляционных систем управления базами данных.

# Требования к аппаратному и программному обеспечению

Требования к программному обеспечению сервера включают:

* операционная система – *Unix* или *Windows 10* и выше.

Требования к аппаратному обеспечению сервера:

* объем свободной оперативной памяти – не менее 4 ГБ;
* объем свободного дискового пространства – не менее 2 ГБ.

Для обеспечения работы на стороне клиента необходимо:

* установленный официальный клиент *Telegram* (в случае работы с *Telegram*-ботом);
* один из браузеров на основе *Gecko 30* и выше, *Chromium 57* и выше, *WebKit 2* и выше.

# Интерфейс пользователя

## Навигация страниц приложения

Схема навигации страниц приложения для пользователя с ролью «Владелец питомцев» представлена на рис. 5.1.

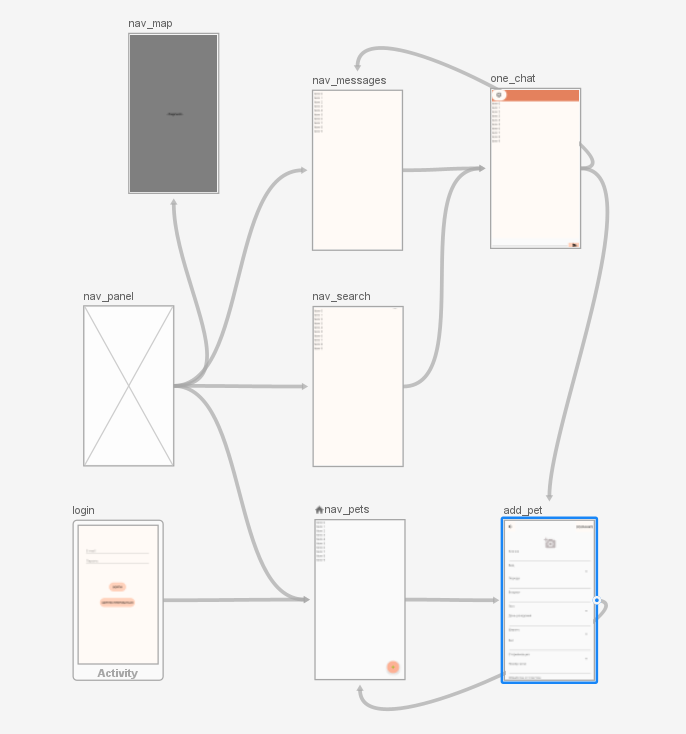


Рис. 5.1. Схема навигации для пользователя с ролью «Владелец питомцев»

Схема навигации страниц приложения для пользователя с ролью «Ветеринар» представлена на рис. 5.2.

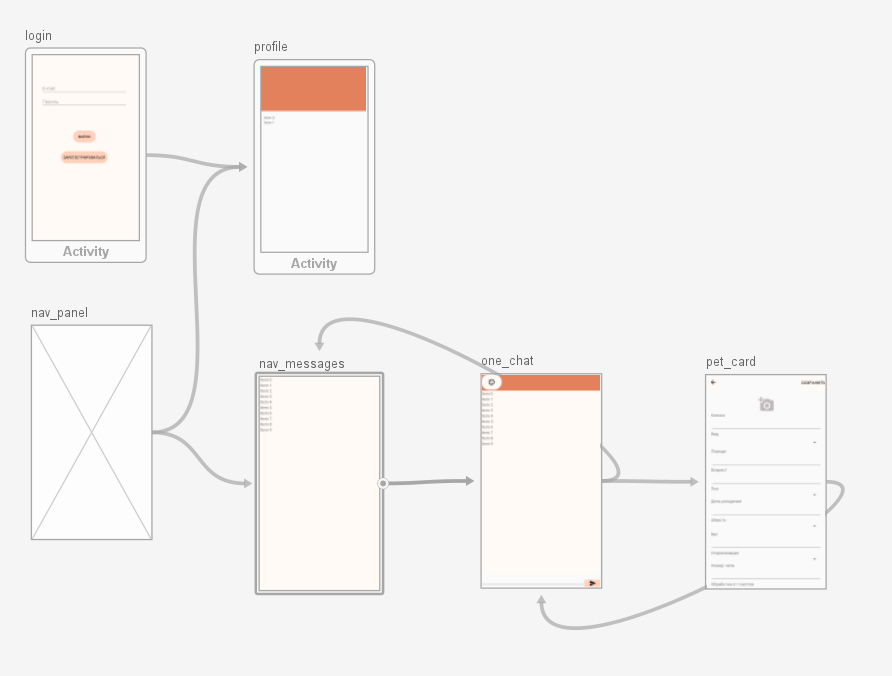


Рис. 5.2. Схема навигации для пользователя с ролью «Ветеринар».

Схема навигации страниц во время регистрации пользователя представлена на рис. 5.3.

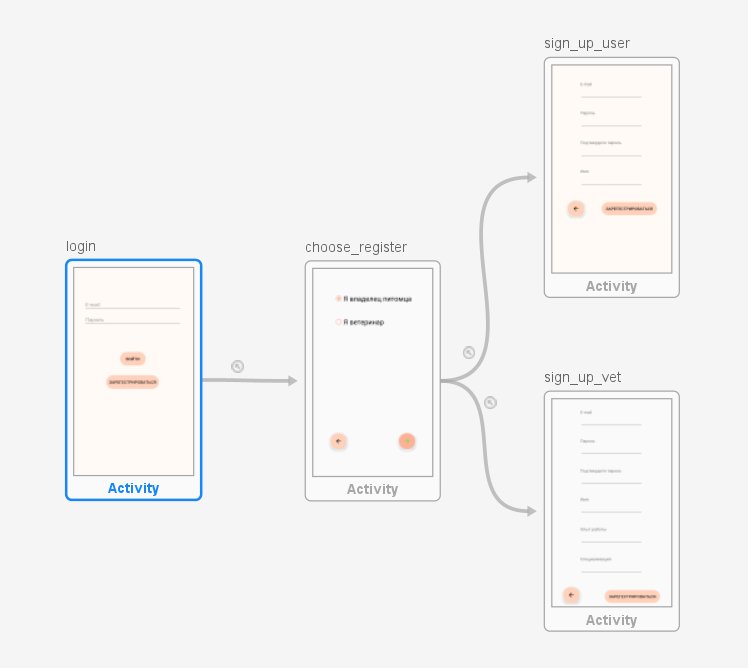


Рис. 5.3. Схема навигации приложения при регистрации пользователя

## Вход и регистрация

Страница входа в приложение содержит поля для ввода электронной почты и пароля, кнопку входа и кнопку перехода к регистрации (рис.5.4).



Рис. 5.4. Страница входа в приложение

Страницы регистрации содержат поля для ввода электронной почты, имени, пароля, подтверждения пароля. Для ветеринара также требуется указать опыт работы и специализацию. Они представлены на рис. 5.5.

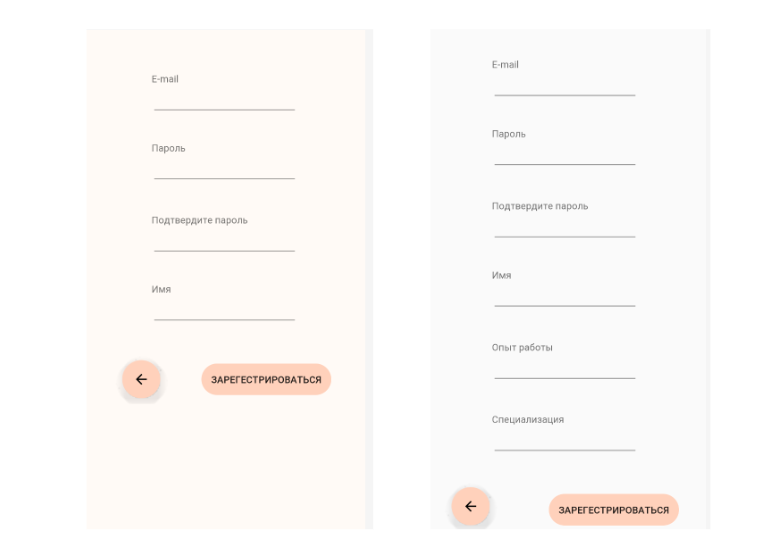


Рис. 5.5. Страницы регистрации

## Страницы пользовательского интерфейса

Любые переходы производятся через навигационную панель (рис 5.6). На ней расположены активные вкладки «Мои питомцы», «Сообщения», «Поиск» и «Клиники поблизости», а также кнопка выхода. При нажатии на фотографию открывается встроенная галерея. После выбора фотографии она отображается на навигационной панели как фото профиля.

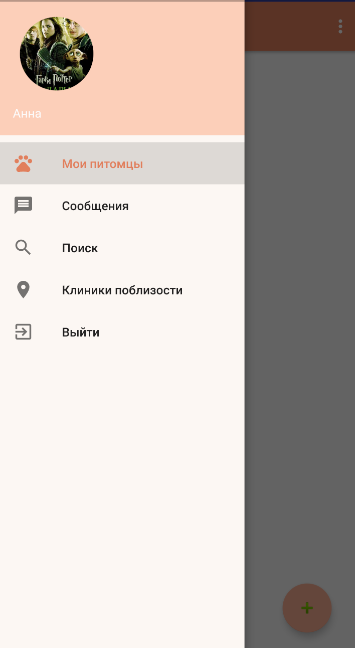


Рис. 5.6. Навигационная панель пользователя

Основной страницей является список питомцев (рис 5.7). При нажатии на любой элемент списка, а также при нажатии на кнопку «+» (добавить питомца), отрывается окно редактирования карты питомца. На странице питомца возможен ввод и редактирование различных данных о питомце, а также, добавление фотографии (рис 5.8).

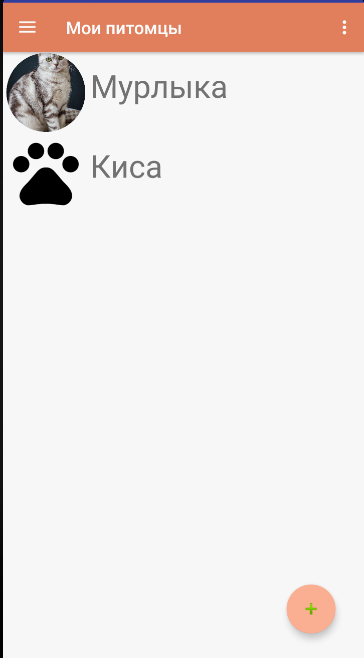


Рис. 5.7. Список питомцев

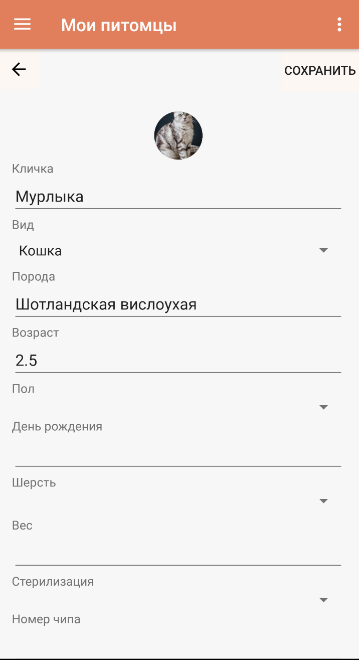


Рис. 5.8. Просмотр карты питомца

На вкладке «сообщения» отображается список всех диалогов (рис. 5.9). При нажатии на любой элемент списка, открывается окно самого диалога (рис. 5.10). На странице присутствует поле для ввода текста, кнопка отправления, кнопка перехода к просмотру карты питомца. Также, при нажатии на имя пользователя открывается его профиль (если это ветеринар).

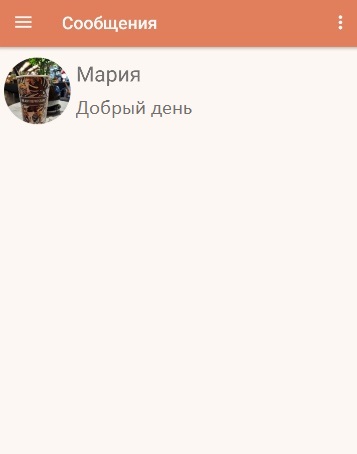


Рис. 5.9. Список диалогов

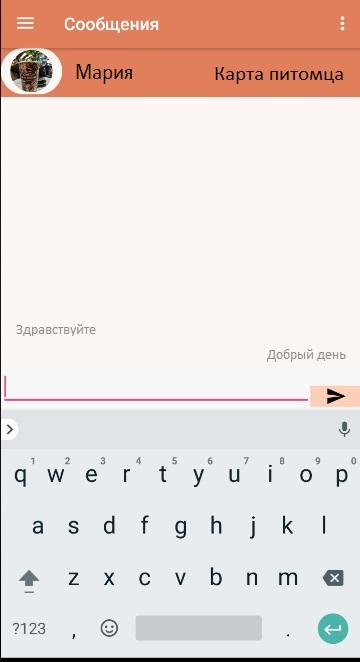


Рис. 5.10. Диалог

На странице поиска присутствуют критерии поиска, а именно некоторые специализации врачей. Если не выбрана ни одна специализация, по умолчанию показаны все зарегистрированные в приложении ветеринарные врачи. Страница поиска представлена на рис. 5.11 — 5.12.

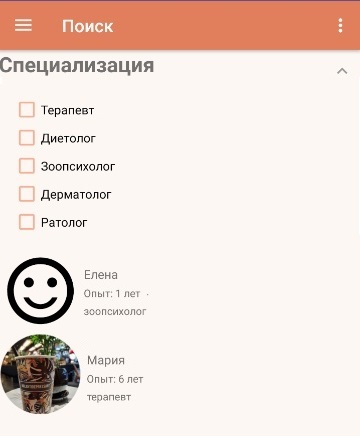


Рис. 5.11. Список всех ветеринаров

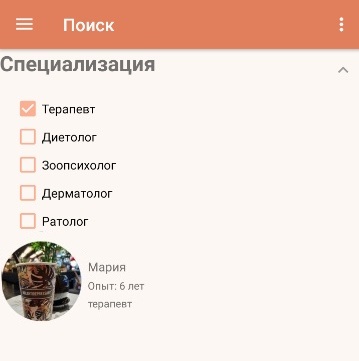


Рис. 5.12. Список терапевтов

На вкладке «клиники поблизости» расположена карта. По умолчанию определяется местоположение пользователя (синий маркер) и места поблизости, имеющие маркер «ветеринарная клиника». Также доступен фильтр, где кроме ветеринарной клиники можно указать такие места как «ветеринарная аптека» и «зоомагазин» (товары для животных). Страницы представлены на рис. 5.13 — 5.14.

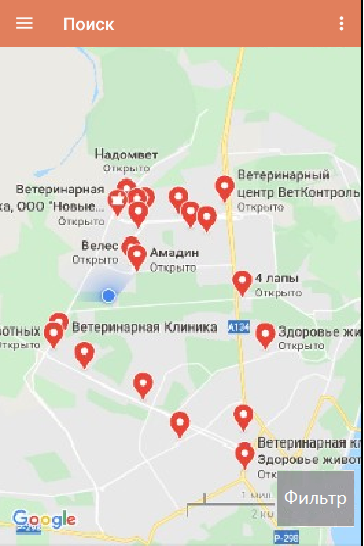


Рис. 5.13. Карта местности

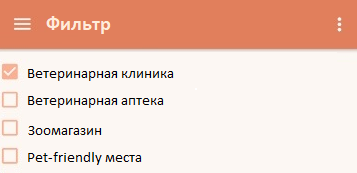


Рис. 5.14. Фильтр мест

# Реализация

## Структура приложения

Структура приложения представлена на схеме компонентов на рис. 6.1.

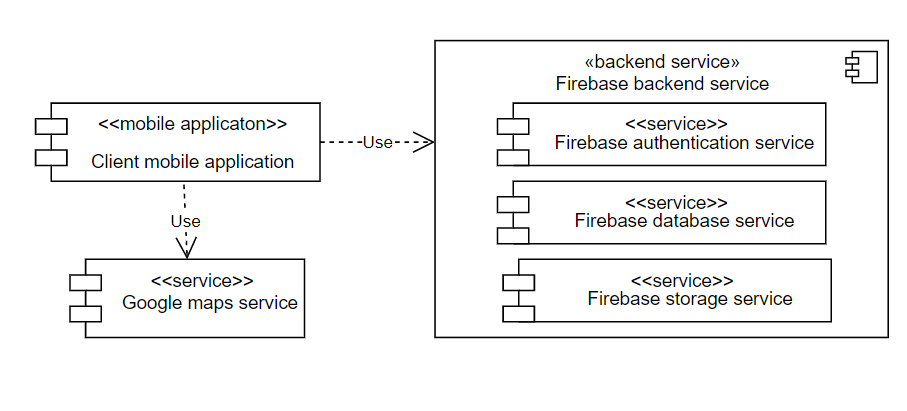


Рис. 6.1. Структура приложения

Для управления пользователями, а также хранения данных используется Firebase (система Backend-as-service). Вместе с хранилищем Firebase также предоставляет пользовательскую аутентификацию, и поэтому все данные передаются через защищенное соединение SSL.

**Firebase Authentication** используется для регистрации и хранения пользователей авторизованными в системе. Регистрация осуществляется через адрес электронной почты и пароль. Каждому пользователю присваивается индивидуальный идентификационный номер [4].

Для хранения данных пользователей используется NoSql система **Firebase Realtime Database**. Данные хранятся в формате Json. Данная система позволяет обновлять данные на стороне мобильного приложения, как только они обновляются на сервере.

**Firebase Storage** позволяет организовывать загрузку и хранение файлов. Возможно хранение изображений, аудио, видео или другого пользовательского контента. Хранилище Firebase поддерживается облачным хранилищем Google, мощным, простым и экономичным средством хранения объектов.

Связь с картографическим сервисом реализуется с помощью **Google Maps API** и **Google Places API**. Данные берутся из базы данных Google. API данных сервисов позволяет производить просмотр карты, определение местоположения пользователя, поиск мест и получение подробной информации о месте. Для полноценной работы нужно получить ключ приложения. С помощью уникального ключа компания Google может отслеживать приложения работающие с сервисом API и в случае надобности связаться с владельцем приложения [5].

Диаграмма компонентов мобильного приложения представлена на рис. 6.3.

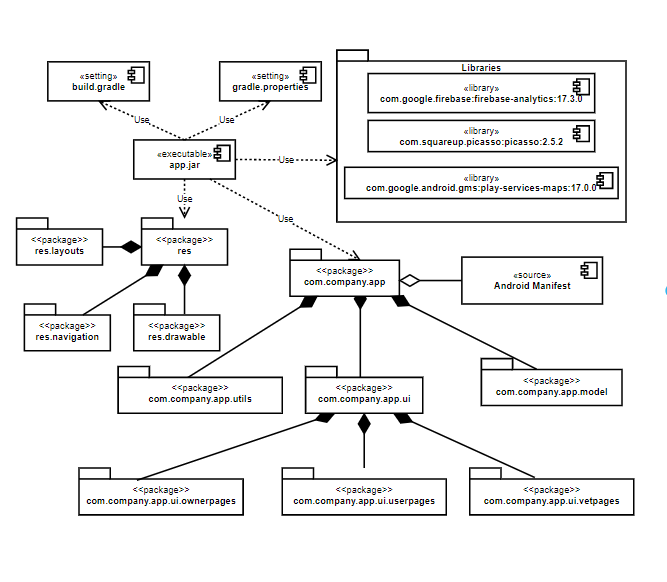


Рис. 6.3. Диаграмма компонентов мобильного приложения

**gradle.properties** — файл, содержащий параметры конфигурации приложения.

**build.gradle** — файл, содержащий информацию о структуре проекта и различных деталях конфигурации, которые используются для автоматической сборки проекта. Листинг конфигурационных файлов проекта приведен в приложении 1.

**Firebase-analytics** — библиотека для взаимодействия c системой Firebase.

**Picasso** — библиотека, содержащая набор функций, для отображения фотографий [6].

Пакеты **com.company.app.ui** содержат классы, реализующие логику поведения и взаимодействия отображаемых страниц пользователя.

Пакет **com.company.app.model** содержит классы данных, соответствующие сущностям базы данных.

Пакет **com.company.app.utils** содержит различные адаптеры списков и различные вспомогательные классы.

Пакеты **res** содержат макеты страниц пользователя, описание навигационной структуры, а также различные шаблоны компонентов страниц.

Пакет **com.company.app.model** содержит классы данных, соответствующие сущностям базы данных.

Файл **Android Manifest** является точкой входа в приложение и содержит конфигурационные данные о порядке запуска страниц, необходимых правах доступа для приложения, метаданные приложения для взаимодействия со сторонними системами [7].

## Модели данных

Данные в Firebase Realtime Database, в отличие от реляционных БД, хранятся в коллекциях, состоящих из объектов (документов) в формате JSON, которые не имеют заранее определенного формата и схемы. На основе анализа модели данных спроектирована база данных, состоящая коллекций.

Примеры данных коллекции представлены в приложении 2.

Описание структуры содержится в табл. 6.1 — 6.6.

Таблица 6.1. Описание полей сущности «Пользователь»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| id | Object ID | Уникальный номер, который генерируется автоматически. |
| name | String | Имя пользователя |
| email | String | Адрес электронной почты |
| profileImage | String | Ссылка на фото профиля |
| chats | Map | Список диалогов |

Таблица 6.2. Описание дополнительных полей сущности «Владелец питомца»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| animals | Map | Список питомцев |

Таблица 6.3. Описание дополнительных полей сущности «Ветеринар»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| specialization | String | Специализация |
| experience | Double | Опыт работы |

Таблица 6.4. Описание полей сущности «Питомец»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| id | Object ID | Уникальный номер, который генерируется автоматически. |
| name | String | Кличка питомца |
| kind | String | Вид |
| breed | String | Порода |
| age | Double | Возраст |
| sex | String | Пол |
| birthday | Date | Дата рождения |
| wool | String | Длина шерсти |
| weight | Double | Вес |
| sterilization | Boolean | Стерилизация (есть/нет) |
| chipNumber | String | Номер чипа (если есть) |
| degelmintization | Date | Дата последней обработки питомца от глистов |
| commonVaccination | Date | Дата последней общей вакцинации |
| fleaVaccination | Date | Дата последней обработки питомца от паразитов |
| rabieVaccination | Date | Дата последней прививки от бешенства |
| profileImage | String | Ссылка на фото профиля |
| info | String | Дополнительные комментарии владельца |

Таблица 6.5. Описание полей сущности «Диалог»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| id | Object ID | Уникальный номер, который генерируется автоматически. |
| userId | String | Идентификатор оппонента |
| cntUnread | String | Количество непрочитанных сообщений |
| listMessages | Map | Список сообщений |

Таблица 6.6. Описание полей логической сущности «Сообщение»

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| id | Object ID | Уникальный номер, который генерируется автоматически. |
| sender | String | Идентификатор отправителя |
| dateTime | Date | Дата и время отправки |
| text | String | Текст сообщения |
| read | Boolean | Маркер чтения |

В соответствии с моделями данных реализованы классы данных, структура и взаимосвязи между ними представлены на диаграмме классов (рис. 6.4). Листинг классов данных приведен в приложении 3.

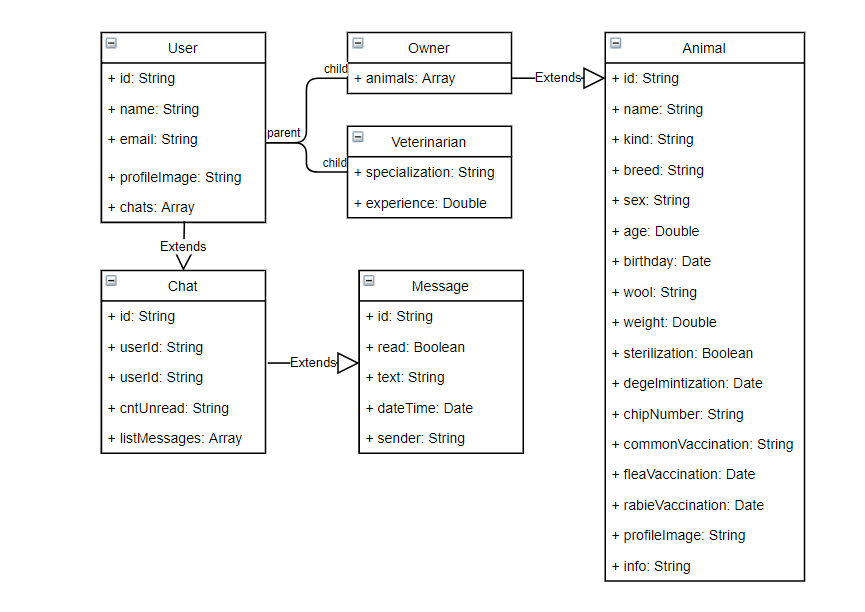


Рис. 6.4. Диаграмма классов модели данных

## Структура функциональных блоков

Для взаимодействия с системой Firebase требуется зарегистрировать приложение в Firebase Console, а также добавить соответствующие конфигурации и библиотеки. После этого доступно использование объеков классов FirebaseUser, FirebaseDatabase, FirebaseAuthentication и FirebaseStorage. Структура класса, реализующего связь с системой Firebase (табл. 6.7).

Таблица 6.7. Методы класса FirebaseUtils

|  |  |
| --- | --- |
| **Имя метода** | **Назначение** |
| static String getCurrentUserId() | Получение id текущего авторизованного пользователя |
| static FirebaseDatabase getDatabase() | Получение ссылки в корень базы данных |
| static StorageReference getUserImage() | Получение ссылки на объект фото профиля текущего пользователя |
| static StorageReference getUserImage(String userId) | Получение ссылки на объект фото профиля по id пользователя |
| static DatabaseReference getUserReference() | Получение ссылки на текущего пользователя приложения в базе данных |
| static DatabaseReference getPetReference(String id) | Получение ссылки в БД по id питомца |
| static void saveUserToDatabase(String userId, Object value) | Создание нового пользователя в БД |
| static void createPet(AnimalDataModel pet) | Создание нового питомца в БД |
| static void updatePet(AnimalDataModel pet) | Редактирование питомца в БД |
| static DatabaseReference getPetsReference() | Получение ссылки на всех питомцев пользователя |
| static DatabaseReference getChatReference(String chatId) | Получение ссылки на диалог по id |
| static Query getMessages(String chatId) | Получение ссылки на сообщения диалога по id |
| static void saveMessageToDatabase(MessageDataModel message, String vetId) | Сохранение сообщения в базе данных |
| static Query searchVeterinarians(String specialization) | Поиск ветеринаров по специализации |
| static DatabaseReference searchVeterinariansById(String id) | Поиск ветеринара по id |

Мобильное приложение реализуется с помощью классов, наследующихся от Activity. В зависимости от того, когда должно выполниться действие, существует ряд методов (табл. 6.8).

Таблица 6.8. Методы класса, наследованного от Activity

|  |  |
| --- | --- |
| **Имя метода** | **Назначение** |
| **void** onCreate(Bundle savedInstanceState) | Метод вызывается при создании объекта класса |
| **void** onStart(Bundle savedInstanceState) | Метод вызывается при старте Activity |
| **void** onResume(Bundle savedInstanceState) | Метод вызывается каждый раз, когда ничего не происходит |
| **void** onStop(Bundle savedInstanceState) | Метод вызывается при остановке Activity |

Для навигации через боковую панель, внутри Activity используются классы Fragment. Так же, как и в классе Activity, в этом классе существует метод onCreate(), который вызывается при создании нового объекта. Кроме этого, существует метод onCreateView(), который будет вызван при непосредственном отображении части страницы. Использование Fragment позволяет заменить часть контента на странице, не перерисовывая ее полностью и сохранить состояние части объектов.

При первом запуске по умолчанию запускается LoginActivity. Далее идет проверка, авторизован ли пользователь в системе. Если нет, то отображается страница входа. Если пользователь авторизован, то запускается соответствующая Activity (в зависимости от его роли).

Для отображения списков включающих шаблоны или активные элементы требуется сам шаблон элемента списка и класс Adapter. Методы этого класса представлены в таблице 6.9.

Таблица 6.9. Методы класса, наследованного от Adapter

|  |  |
| --- | --- |
| **Имя метода** | **Назначение** |
| **int getItemCount()** | Количество элементов списка |
| RecyclerView.ViewHolder **onCreateViewHolder**(ViewGroup parent, **int** viewType) | Метод вызывается при создании элемента списка |
| **void onBindViewHolder**(RecyclerView.ViewHolder holder, **int** position | Метод вызывается при динамической генерации элемента списка |

Для класса адаптера используется внутренний класс, наследующийся от ViewHolder и реализующий интерфейс View.OnClickListener [2]. Для этого класса обязательно определяется метод конструктор и метод onClick(), который срабатывает при нажатии на элемент списка.

# План тестирования

Тестирование разработанного приложения состоит из следующих этапов:

* + - 1. Проверка соответствия пользовательского интерфейса поставленным задачам.
      2. Проверка корректности обработки всех действий, которые потенциально может выполнить пользователь.
      3. Проверка корректности взаимодействий реального времени.

***Тест 1.*** Тест авторизации при входе в приложение.

***Цель.*** Проверка корректности работы авторизации.

***Порядок выполнения***. При входе в приложение, ввести ошибочные данные в форму авторизации. Нажать кнопку «Войти».

***Результат.*** Сообщение об ошибке (рис. 7.1).



Рис. 7.1. Ошибка входа

***Тест 2.*** Тест добавления питомца.

***Цель.*** Проверка корректности работы добавления и просмотра карты питомца.

***Порядок выполнения.*** В списке питомцев нажать кнопку «+». Откроется окно добавления (рис. 7.2). Ввести данные о питомце и нажать «сохранить».

***Результат.*** Страница со списком питомцев будет обновлена, и новый питомец будет отображаться на ней рис. 7.3.

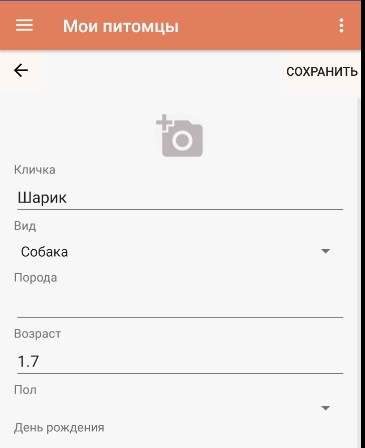


Рис. 7.2. Окно добавления

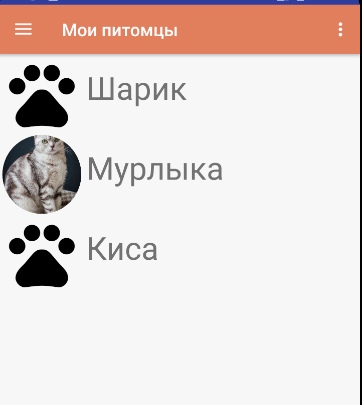


Рис. 7.3. Питомец добавлен

***Тест 3*.** Тест приема сообщений.

***Цель***. Проверка корректности работы получения сообщений.

***Порядок выполнения.*** В консоли Firebase вручную ввести сообщение.

***Результат.*** Сообщение появится в диалоге.

***Тест 4*.** Тест отправки сообщений.

***Цель***. Проверка корректности работы отправки сообщений.

***Порядок выполнения.*** На вкладке «сообщения» выбрать диалог. Ввести в поле текст и нажать кнопку отправления. Ввести данные о питомце и нажать «сохранить».

***Результат.*** Сообщение появится в консоли Firebase.

***Тест 5*.** Тест поиска ветеринаров.

***Цель***. Проверка корректности работы поиска людей по критериям.

***Порядок выполнения.*** На вкладке «поиск» выбрать несколько специализаций.

***Результат.*** В списке представлены ветеринары выбранных специализаций.

# Заключение

В ходе работы разработано мобильное приложение для взаимодействия владельцев питомцев и ветеринаров, предоставляющее следующие возможности:

* регистрация пользователей:
* регистрация владельцев животных;
* регистрация ветеринаров;
* добавление и просмотр карты питомца;
* ввод и модификация личных данных пользователя (ветеринара или владельца питомца);
* загрузка фотографий;
* возможность поиска ветеринаров определенной специализации;
* возможность переписки владельцев животных и ветеринаров;
* доступ к карте питомца для ветеринара;
* поиск ветеринарных клиник в городе;
* поиск мест в определенном городе, в которых разрешено находится с животными.

# Список литературы

1. Фримен, Эрик Паттерны проектирования / Эрик Фримен и др.  – М. Питер, 2015.  – 656 c.
2. Хорстман К. Java. Библиотека профессионала. Том 1. Основы / К. Хорстман, Г. Корнелл. — М. : Вильямс, 2014. — 864 с.
3. Леоненков А. В. Самоучитель UML 2 / А. В. Леоненков. – Санкт-Петербург : БХВ-Петербург, 2007. – 576 с.
4. Firebase documentation. Read and write data on Andoid. URL: https://firebase.google.com/docs/database/android/structure-data?hl=ru
5. Goggle Maps platform. [Maps SDK for Android](https://developers.google.com/maps/documentation/android-sdk/intro). URL: https://developers.google.com/maps/documentation/android-sdk/start
6. Best practices от Google по разработке Android приложений. URL: https://habr.com/ru/post/264255/
7. Build anything on Android. URL: https://developer.android.com
8. Эффективная Java / Д. Блох. – М. Лори, 2014. – 294 с.

# Приложение 1. Листинг конфигурационных файлов проектов

*build.gradle*

apply plugin: 'com.android.application'

apply plugin: 'com.google.gms.google-services'

android {

compileSdkVersion 29

buildToolsVersion "29.0.3"

defaultConfig {

applicationId "com.example.app"

minSdkVersion 24

targetSdkVersion 29

versionCode 1

versionName "1.0"

vectorDrawables.useSupportLibrary = true

testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"

}

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'

}

}

compileOptions {

sourceCompatibility = 1.8

targetCompatibility = 1.8

}

}

dependencies {

implementation fileTree(dir: 'libs', include: ['\*.jar'])

implementation 'androidx.appcompat:appcompat:1.0.2'

implementation 'com.google.android.material:material:1.0.0'

implementation 'androidx.annotation:annotation:1.1.0'

implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

implementation 'androidx.lifecycle:lifecycle-extensions:2.0.0'

implementation 'com.google.firebase:firebase-storage:19.1.1'

implementation 'androidx.legacy:legacy-support-v4:1.0.0'

implementation 'androidx.navigation:navigation-fragment:2.0.0'

implementation 'androidx.navigation:navigation-ui:2.0.0'

testImplementation 'junit:junit:4.12'

androidTestImplementation 'androidx.test.ext:junit:1.1.1'

androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

implementation 'com.google.firebase:firebase-analytics:17.3.0'

implementation 'com.firebaseui:firebase-ui-auth:6.2.0'

implementation 'com.google.firebase:firebase-auth:19.3.0'

implementation 'com.google.android.gms:play-services-auth:18.0.0'

implementation 'com.google.firebase:firebase-database:19.2.1'

implementation 'com.squareup.picasso:picasso:2.5.2'

implementation 'com.robertlevonyan.view:MaterialExpansionPanel:2.0.8'

implementation 'com.google.android.gms:play-services-base:17.1.0'

implementation 'com.google.android.gms:play-services-maps:17.0.0'

implementation 'com.google.android.gms:play-services-location:17.0.0'

implementation 'com.google.android.libraries.places:places:2.2.0'

}

*gradle.properties*

distributionBase=GRADLE\_USER\_HOME

distributionPath=wrapper/dists

zipStoreBase=GRADLE\_USER\_HOME

zipStorePath=wrapper/dists

distributionUrl=https\://services.gradle.org/distributions/gradle-5.6.4-all.zip

org.gradle.jvmargs=-Xmx1536m

http://www.gradle.org/docs/current/userguide/multi\_project\_builds.html#sec:decoupled\_projects

android.enableJetifier=true

*AndroidManifest.xml*

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.app">

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity

android:name=".activities.vetpages.VetPageActivity"

android:label="@string/title\_activity\_vet\_page"

android:theme="@style/AppTheme.NoActionBar"></activity>

<activity android:name=".activities.login.SignUpVeterinarianActivity" />

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="@string/google\_map\_api\_key" />

<activity android:name=".activities.login.ChooseRegisterActivity" />

<activity

android:name=".activities.UserPageActivity"

android:label="@string/title\_activity\_user\_page"

android:theme="@style/AppTheme.NoActionBar" />

<activity android:name=".activities.login.SignUpUserActivity" />

<activity

android:name=".activities.login.LoginActivity"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

# Приложение 2. Примеры коллекции данных

{

"users" : {

"dz5RloQiT5cutLZCbNhjgO46paW2" : {

"animals" : {

"-M5S1koaTSZC-BPnfbli" : {

"age" : 2.5,

"breed" : "вислоухая",

"chipNumber" : "",

"id" : "-M5S1koaTSZC-BPnfbli",

"kind" : "Кошка",

"name" : "Сара",

"profileImage" : "кошка.jpg",

"sex" : "девочка",

"sterilization" : "нет",

"wool" : "короткая"

},

"-M5SQ0-bE\_G-DV3VMc4c" : {

"age" : 5,

"breed" : "обычная",

"kind" : "Кошка",

"name" : "Киса"

}

},

"chats" : {

"q22" : {

"cntUnread" : 1,

"listMessages" : {

"-M6G2CIrhD5BLgmSqLT0" : {

"dateTime" : "01/05/2020 16:23:31",

"read" : false,

"sender" : "dz5RloQiT5cutLZCbNhjgO46paW2",

"text" : "Привет"

},

"-M6G2EBNLOj7iTs8jrwz" : {

"dateTime" : "01/05/2020 16:23:38",

"read" : false,

"sender" : "dz5RloQiT5cutLZCbNhjgO46paW2",

"text" : "Как дела?"

}

},

"userId" : "q22"

}

},

"email" : "a@c.com",

"name" : "A",

"profileImage" : "59384.jpg"

},

"veterinarians" : {

"2" : {

"experience" : 1,

"name" : "Мария"

},

"q22" : {

"name" : "Иван",

"profileImage" : "20200202\_150053.jpg",

"chats" : {

"dz5RloQiT5cutLZCbNhjgO46paW2" : {

"listMessages" : {

"-M6G2CIt3qaYUfLSpVDa" : {

"dateTime" : "01/05/2020 16:23:31",

"read" : false,

"sender" : "dz5RloQiT5cutLZCbNhjgO46paW2",

"text" : "Привет"

},

"-M6G2EBOmz-lXdyaGAtj" : {

"dateTime" : "01/05/2020 16:23:38",

"read" : false,

"sender" : "dz5RloQiT5cutLZCbNhjgO46paW2",

"text" : "Как дела?"

},

}

}

}

}

}

}

# Приложение 3. Листинг классов данных

*UserDataModel*.

public class UserDataModel implements Serializable {

protected String id;

protected String name;

protected String email;

protected String profileImage;

protected Map<String, ChatDataModel> chats;

public User() {

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getProfileImage() {

return profileImage;

}

public void setProfileImage(String profileImage) {

this.profileImage = profileImage;

}

public Map<String, ChatDataModel> getChats() {

return chats;

}

public void setChats(Map<String, ChatDataModel> chats) {

this.chats = chats;

}

}

*OwnerDataModel*

public class OwnerDataModel extends UserDataModel implements Serializable {

private Map<String,AnimalDataModel> animals;

public Map<String, AnimalDataModel> getAnimals() {

return animals;

}

public void setAnimals(Map<String, AnimalDataModel> animals) {

this.animals = animals;

}

public UserDataModel(){

}

}

*VeterinarianDataModel*

public class VeterinarianDataModel extends UserDataModel implements Serializable {

private String specialization;

private Integer experience;

public VeterinarianDataModel() {

}

public String getSpecialization() {

return specialization;

}

public void setSpecialization(String specialization) {

this.specialization = specialization;

}

public Integer getExperience() {

return experience;

}

public void setExperience(Integer experience) {

this.experience = experience;

}

}

*AnimalDataModel*

public class AnimalDataModel implements Serializable {

private String id;

private String name;

private String kind;

private String breed;

private Double age;

private String sex;

private Date birthday;

private String wool;

private Double weight;

private String sterilization;

private String chipNumber;

private Date degelmintization;

private Date fleaVaccination;

private Date commonVaccination;

private Date rabieVaccination;

private String info;

private String profileImage;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getKind() {

return kind;

}

public void setKind(String kind) {

this.kind = kind;

}

public String getBreed() {

return breed;

}

public void setBreed(String breed) {

this.breed = breed;

}

public Double getAge() {

return age;

}

public void setAge(Double age) {

this.age = age;

}

public AnimalDataModel(@Nullable String name,@Nullable String kind,

@Nullable String breed,@Nullable Double age) {

this.name = name;

this.kind = kind;

this.breed = breed;

this.age = age;

}

public AnimalDataModel(){

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

public String getWool() {

return wool;

}

public void setWool(String wool) {

this.wool = wool;

}

public Double getWeight() {

return weight;

}

public void setWeight(Double weight) {

this.weight = weight;

}

public String getSterilization() {

return sterilization;

}

public void setSterilization(String sterilization) {

this.sterilization = sterilization;

}

public String getChipNumber() {

return chipNumber;

}

public void setChipNumber(String chipNumber) {

this.chipNumber = chipNumber;

}

public Date getDegelmintization() {

return degelmintization;

}

public void setDegelmintization(Date degelmintization) {

this.degelmintization = degelmintization;

}

public Date getFleaVaccination() {

return fleaVaccination;

}

public void setFleaVaccination(Date fleaVaccination) {

this.fleaVaccination = fleaVaccination;

}

public Date getCommonVaccination() {

return commonVaccination;

}

public void setCommonVaccination(Date commonVaccination) {

this.commonVaccination = commonVaccination;

}

public Date getRabieVaccination() {

return rabieVaccination;

}

public void setRabieVaccination(Date rabieVaccination) {

this.rabieVaccination = rabieVaccination;

}

public String getInfo() {

return info;

}

public void setInfo(String info) {

this.info = info;

}

public String getProfileImage() {

return profileImage;

}

public void setProfileImage(String profileImage) {

this.profileImage = profileImage;

}

}

*ChatDataModel*

public class ChatDataModel implements Serializable {

private String id;

private String userId;

private Map<String ,MessageDataModel> listMessages;

private Long cntUnread;

public String getUserId() {

return userId;

}

public void setUserId(String userId) {

this.userId = userId;

}

public Long getCntUnread() {

return cntUnread;

}

public void setCntUnread(Long cntUnread) {

this.cntUnread = cntUnread;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public Map<String, MessageDataModel> getListMessages() {

return listMessages;

}

public void setListMessages(Map<String, MessageDataModel> listMessages) {

this.listMessages = listMessages;

}

public ChatDataModel() {

}

}

*MessageDataModel*

public class MessageDataModel implements Serializable {

private String id;

private String sender;

private String dateTime;

private String text;

private Boolean read;

public static Comparator<MessageDataModel> compareByDate = new Comparator<MessageDataModel>() {

@Override

public int compare(MessageDataModel o1, MessageDataModel o2) {

return o1.getDateTime().compareTo( o2.getDateTime() );

}

};

public MessageDataModel() {

}

public void setRead(Boolean read) {

this.read = read;

}

public Boolean getRead() {

return read;

}

public static Comparator<MessageDataModel> getCompareByDate() {

return compareByDate;

}

public static void setCompareByDate(Comparator<MessageDataModel> compareByDate) {

MessageDataModel.compareByDate = compareByDate;

}

public MessageDataModel(@NonNull String sender, @NonNull String dateTime, @NonNull String text) {

this.sender = sender;

this.dateTime = dateTime;

this.text = text;

this.read = false;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getSender() {

return sender;

}

public void setSender(String sender) {

this.sender = sender;

}

public String getDateTime() {

return dateTime;

}

public void setDateTime(String dateTime) {

this.dateTime =dateTime;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

}