

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

Введение.....	4
1. Постановка задачи	5
2. Анализ задачи.....	6
2.1. Анализ существующих решений.....	6
2.2. Анализ функциональности приложения	11
2.3. Анализ выбранных источников информации	11
2.4. Анализ модели данных.....	12
2.5. Анализ структуры приложения	19
3. Средства реализации.....	20
4. Требования к аппаратному и программному обеспечению	21
5. Интерфейс пользователя	22
5.1. Навигация страниц приложения.....	22
5.2. Главная страница Telegram-бота.....	23
5.3. Главная страница сервисов Telegram-бота.....	24
5.4. Главная страница настроек сервисов.....	24
5.5. Главная страница настроек частоты оповещений	25
5.6. Страница настроек рассылки новостей	26
5.7. Страница настроек сервиса Twitter	27
5.8. Страница настроек рассылки погоды	29
5.9. Страница модификаций пользовательских фильтров.....	30
5.10. Страница просмотра информации	30
6. Реализация	33
6.1. Структура приложения.....	33
6.2. Серверная часть приложения.....	34
6.3. Клиентская часть приложения.....	39
7. План тестирования	41
Заключение	45
Список литературы	46
Приложение 1. Физическая модель данных.....	47
Приложение 2. Листинг клиентской части приложения.....	49
Приложение 3. Листинг серверной части приложения.....	54

Введение

В современном мире быть в курсе всех последних новостей, когда весь день проходит очень быстро просто-напросто невозможно. Ситуация осложняется тем, что большинство новостных источников — радио, телевизор, новостные сайты для компьютеров не очень удобны для просмотра 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. Очень активно развиваются боты и каналы в социальных сетях и мессенджерах, поскольку их наличие в одном приложении делает программу универсальной. 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.

1. Постановка задачи

Разработать веб-приложение *KeepMePosted*, реализующее систему для сбора новостей, *твитов* и прогноза погоды как через открытое API, так и с помощью анализа RSS-каналов. Клиенты программы должны быть представлены в виде *Telegram* бота и веб-страницы, предоставляющие 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 городов 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*.

2. Анализ задачи

2.1. 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).

2.1.1. Telegram-бот @ONOMediaScopeBot

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

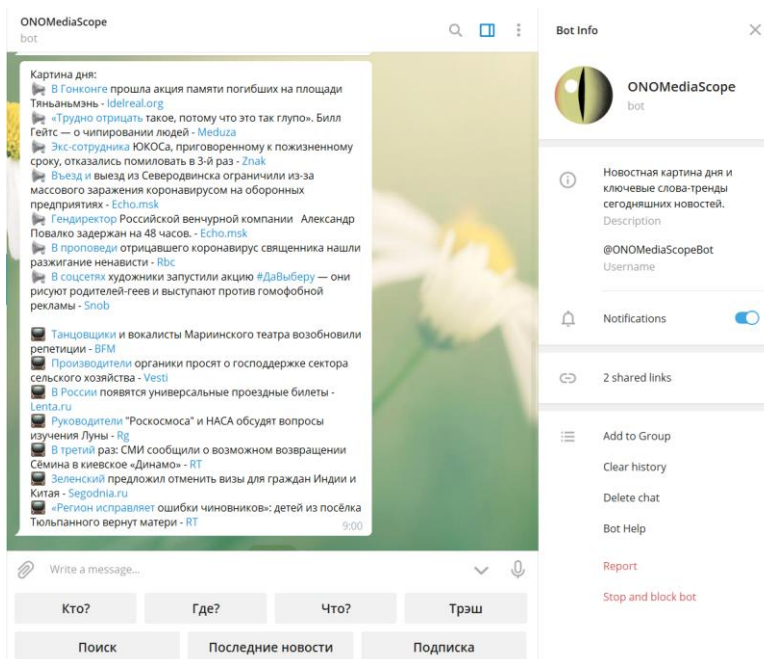


Рис. 2.1. Главное меню *Telegram*-бота @ONOMediaScopeBot

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 тех категорий, на которые подписан пользователь.

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

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

2.1.2. Сайт и мобильное приложение *mediametrics.ru*

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

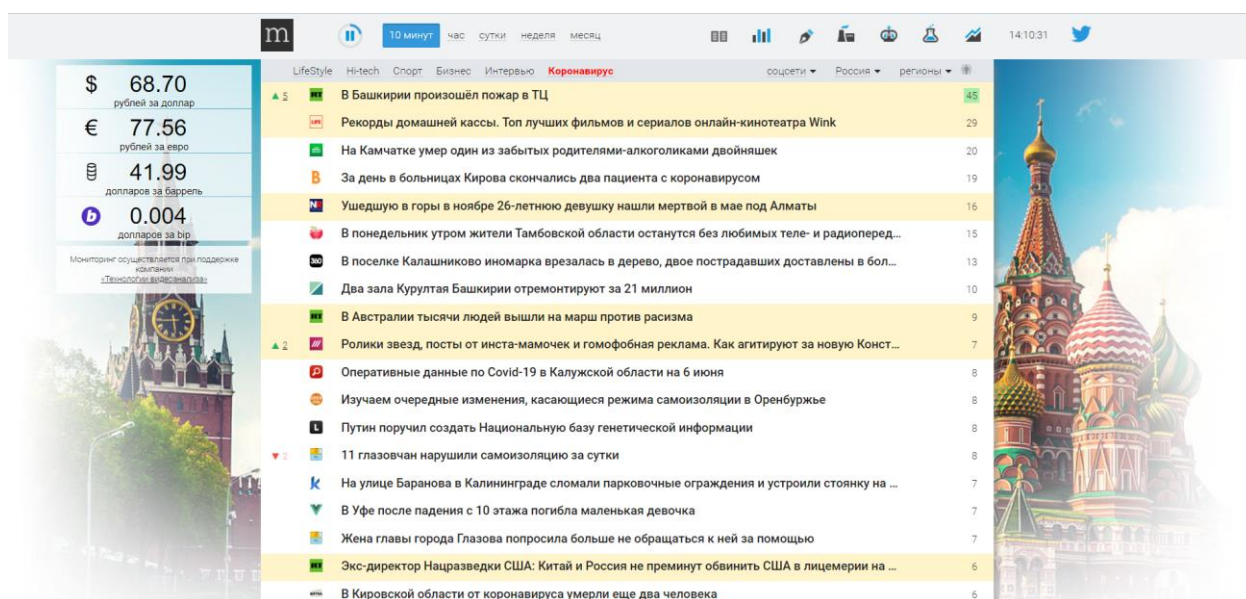


Рис. 2.2. Главная страница сервиса *mediametrics.ru*

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 *iOS*, которое представлено на рис. 2.3 и рис. 2.4.

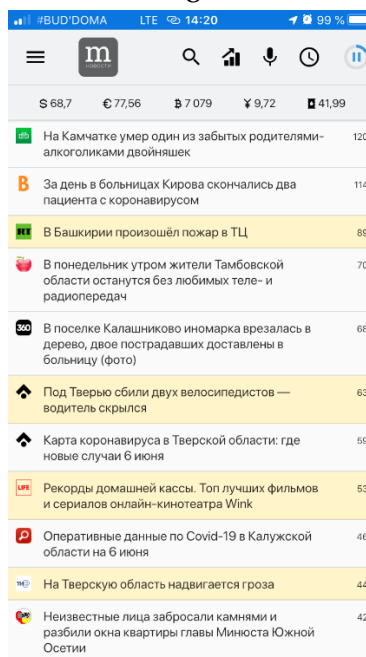


Рис. 2.3. Главная страница мобильного приложения *Mediametrics*

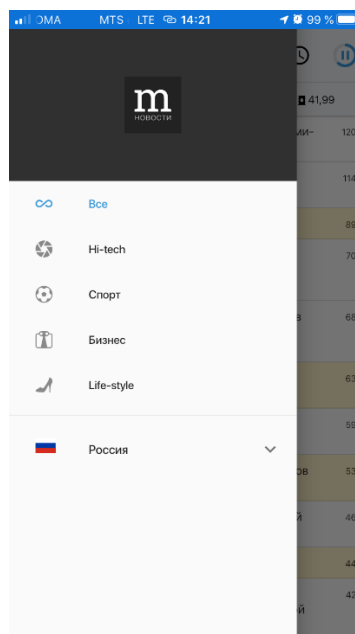


Рис. 2.4. Предопределенные темы мобильного приложения *Mediametrics*

Мобильное приложение реализует имеет следующие возможности:

- работа основного сайта в удобном интерфейсе;
- формирование рейтинга новостей в разрезе некоторого промежутка времени;
- интеграция с сервисом курса валют, имеется поиск по заданному слову.

Но приложение не поддерживает отправку оповещений.

2.1.3. Сервис Яндекс.Новости

На рис. 2.5 представлена главная страница портала *Яндекс.Новости*. Отображение не только популярных новостей, но и информации на основе ваших последних запросов в браузере являются отличительной чертой данного портала. 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.

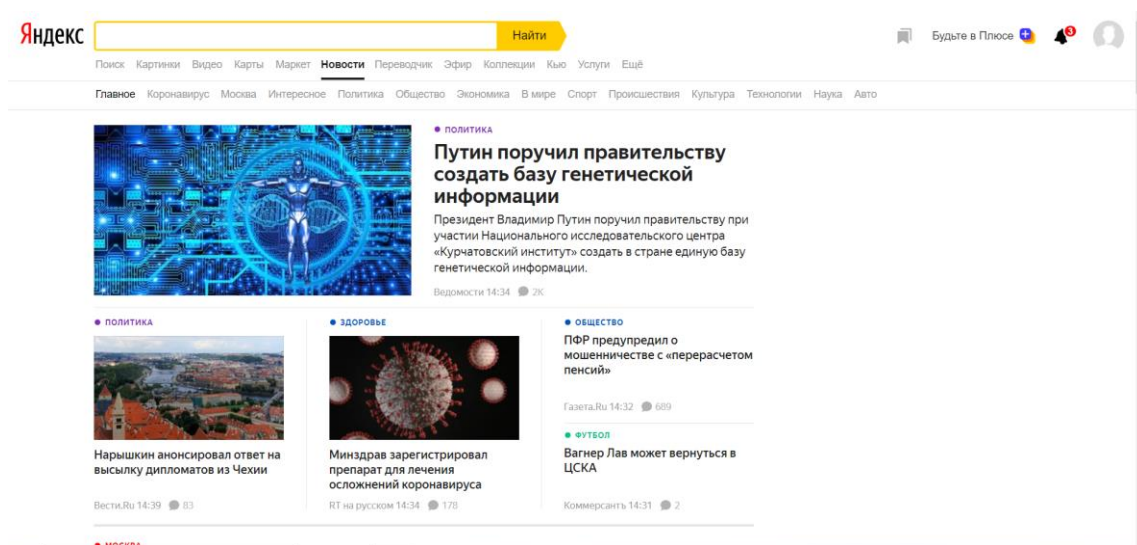


Рис. 2.5. Главная страница портала *Яндекс.Новости*

2.1.4. Сервис Яндекс.Дзен

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

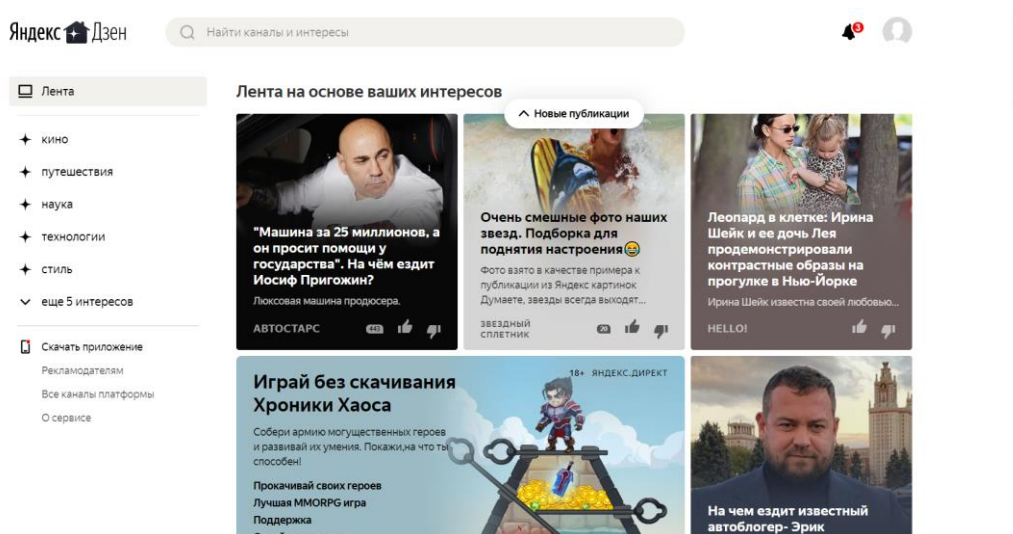


Рис. 2.6. Главная страница сервиса *Яндекс.Дзен*

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 *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.

2.1.5. Результаты анализа существующих решений

Результаты анализа представлены в таблице 2.1.

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 (<i>ВКонтакте</i> , <i>Telegram</i>)	+	—	—	—
Доступ на русском языке	+	+	+	+
Отбор новостей по нескольким категориям	+	—	—	—
Рейтинг популярных новостей в размере промежутка времени (день, неделя, месяц)	+	+	—	—
Наличие сторонних информационных ресурсов (погода, <i>Twitter</i> , курс валют)	—	+	+	—

Приложений подобного характера в данной бизнес-области было найдено не очень много. Большинство из них представляет собой сайты и мобильные приложения, показывающие только новости одной конкретной компании. Чтобы ознакомиться с новостями другого источника, пользователю необходимо будет перейти на его страницу, что не очень удобно. Либо это

приложения, которые не поддерживают подписку на определенные темы и не отправляют оповещения. При анализе существующих решений не было найдено решения, удовлетворяющего всем критериям.

2.2. 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 должен включать:

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

Сервис *твитов* соц. сети *Twitter* должен включать:

- ввод и модификацию отслеживаемых хештегов, пользователей;

Сервис погоды должен включать:

- 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;

2.3. 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 расширения *Spring Social* фреймворка *Spring*, можно легко интегрироваться и удобно пользоваться возможностями *Twitter* [2].

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*.

2.4. Анализ модели данных

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

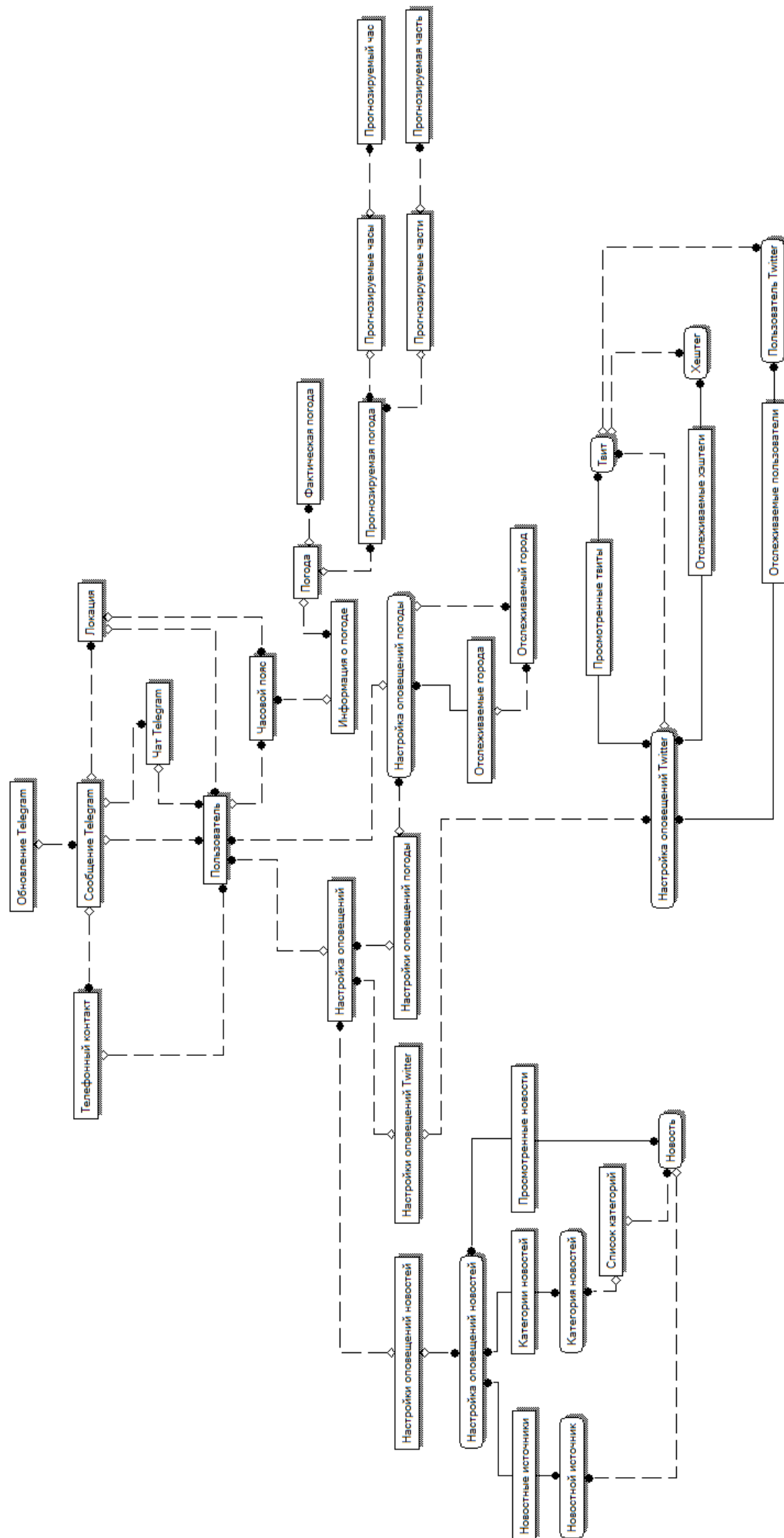


Рис. 2.7. Логическая модель данных уровня сущностей

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*, *ID_Информации_о_погоде*, *Время сервера в UTC*, *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_прогнозируемых_часов*, *ID_прогнозируемых_частей*.
12. *Прогнозируемые часы* – табличная часть, с первичными ключами *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, с первичными ключами *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, первичными 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, с первичными ключами *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, первичными 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_Пользователя, ID_Настройки_оповещений, Включены пользовательские настройки.*

23. *Просмотренные новости* – табличная часть, с первичными ключами *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, с первичными ключами *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, с первичными ключами *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, первичными 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, с первичными ключами *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, первичными 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*.

2.5. Анализ структуры приложения

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

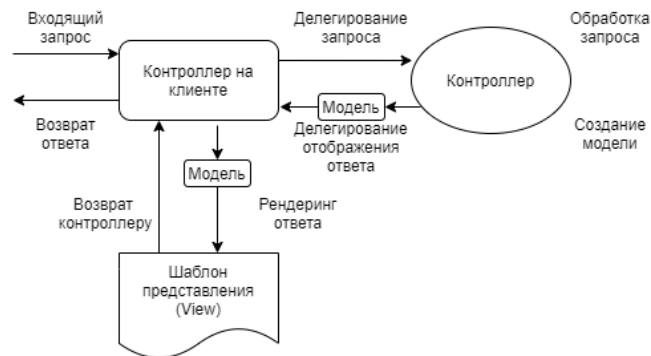


Рис. 2.8. Взаимодействие частей приложения

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

Модель (Model) объединяет данные приложения, все вместе они будут состоять из бинов *Java*, управляемых *Spring IoC* контейнером.

Шаблон представления (View) отвечает за отображение данных модели.

Контроллер (Controller) обрабатывает запрос пользователя, создает модель и передает ее для отображения в шаблон представления (View).

3. Средства реализации

Для разработки приложения выбраны следующие программные средства:

- интерактивная среда разработки – *IntelliJ IDEA 2020.1.2*;
- язык программирования – *Java* [8];
- 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* [1].

4. Требования к аппаратному и программному обеспечению

Требования к программному обеспечению сервера включают:

- операционную систему – *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.

5. Интерфейс пользователя

5.1. Навигация страниц приложения

Схема навигации главной страницы приложения для пользователя, использующего клиент *Telegram* представлена на рис. 5.1.

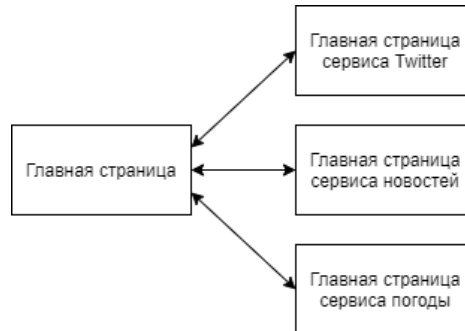


Рис. 5.1. Схема навигации главной страницы *Telegram*-бота

Все доступные сервисы *Telegram*-бота имеют одинаковую модель навигации, которая на рис. 5.2.

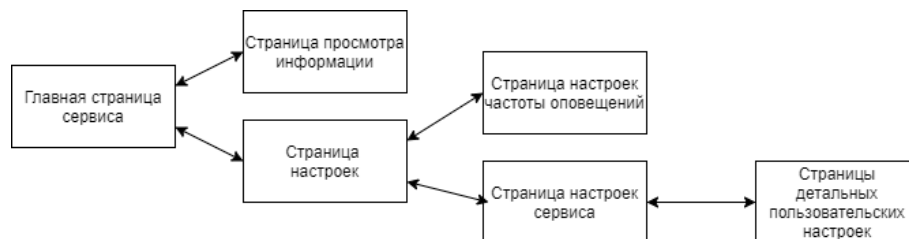


Рис. 5.2. Схема навигации сервисов *Telegram*-бота

Веб-сайт приложения в свою очередь имеет всего одну страницу с некоторым набором фильтров. Схема страницы представлена на рис. 5.3.

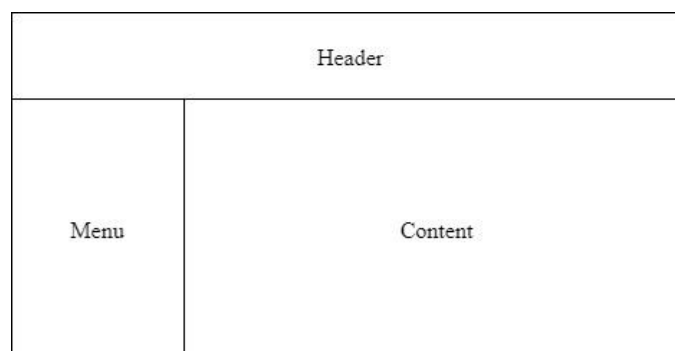


Рис. 5.3. Схема страницы

5.2. Главная страница Telegram-бота

При подключении к *Telegram*-боту, последний приветствует пользователя, показывается краткое описание того, что бот умеет делать (рис.5.4).

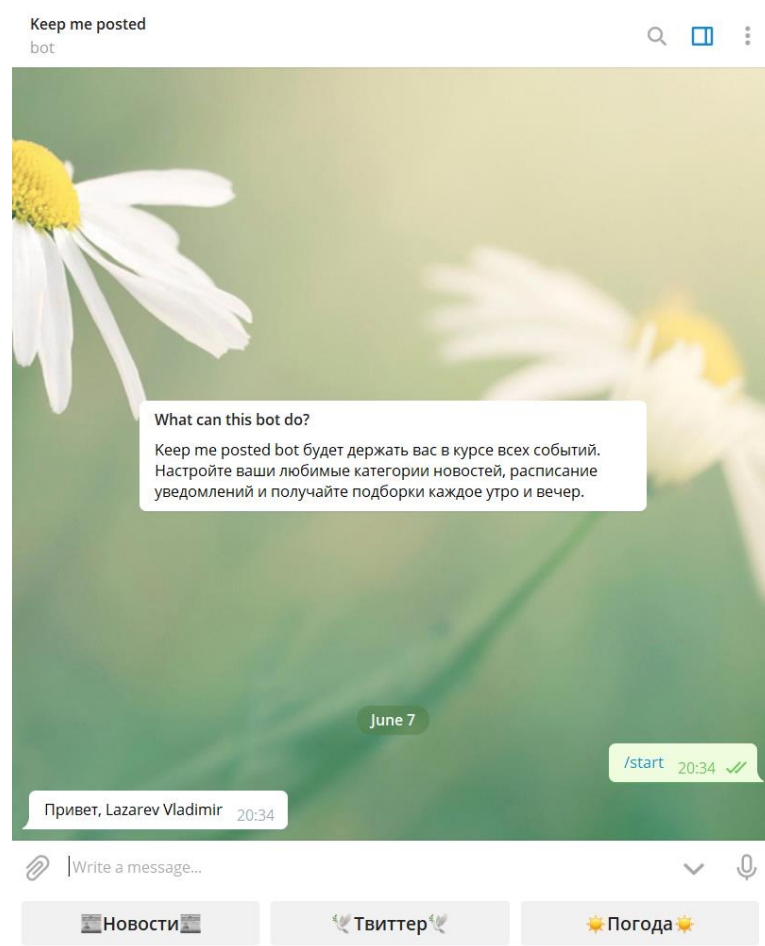


Рис. 5.4. Страница входа в приложение

Главная страница содержит следующие элементы управления:

- кнопка *Новости*;
- кнопка *Твиттер*;
- кнопка *Погода*.

При подключении бота пользователь автоматически регистрируется в системе по его уникальному идентификатору чата и профиля. По нажатию на любую из кнопок осуществляется переадресация на главную страницу выбранного пользователем сервиса.

5.3. Главная страница сервисов Telegram-бота

Главные страницы сервисов имеют одинаковую структуру. Образец внешнего вида представлен на рис. 5.5.

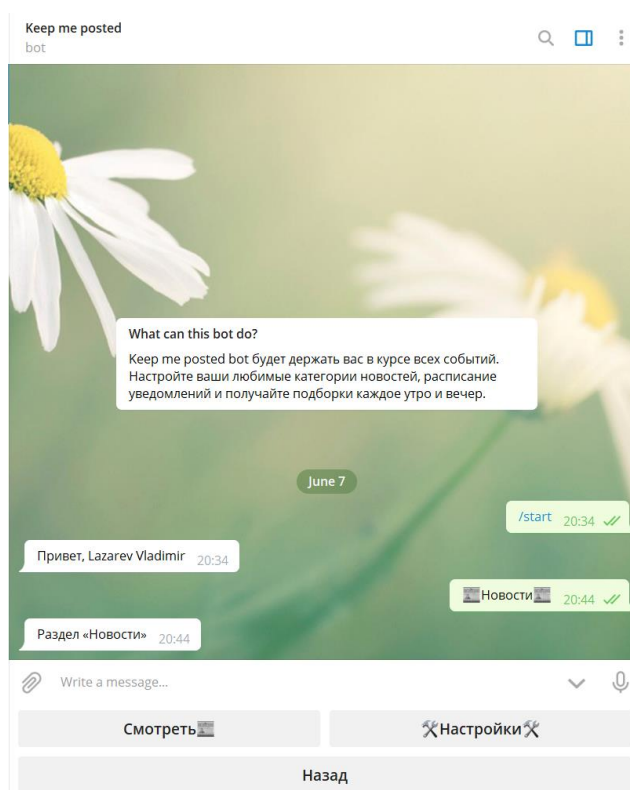


Рис. 5.5. Главная страница сервисов

Страница содержит следующие элементы управления:

- кнопка *Смотреть* отправляет пользователя на страницу просмотра информации по выбранному ранее сервису, то есть при выборе раздела *Новости* по нажатию на эту кнопку откроется страница просмотра новостей;
- кнопка *Настройки*, перенаправляющая на страницу настроек выбранного раздела;
- кнопка *Назад* служит для возврата на главную страницу приложения.

5.4. Главная страница настроек сервисов

При входе на главную страницу настроек пользователю покажется сообщение, в настройках какого раздела он находится (рис. 5.6). Данная страница имеет одинаковую структуру для всех сервисов.

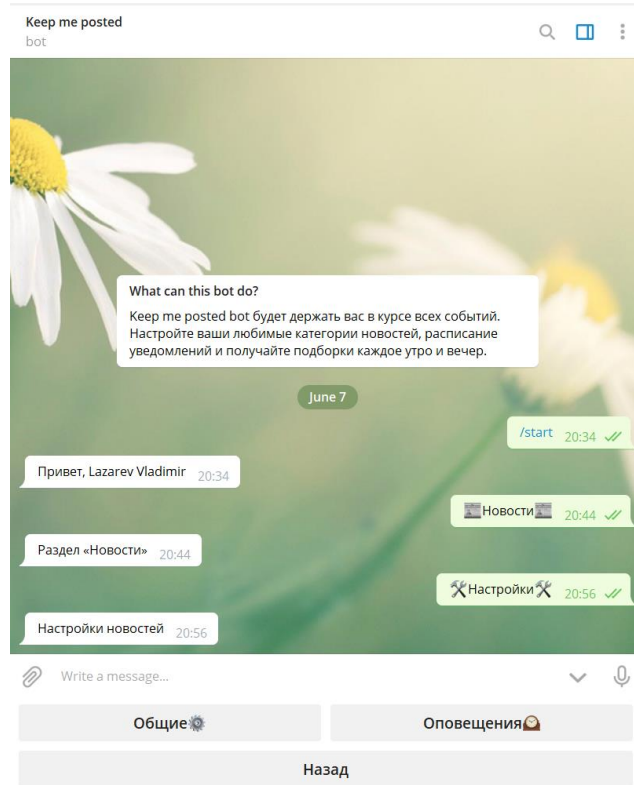


Рис. 5.6. Главная страница настроек сервисов

Страница содержит следующие элементы управления:

- кнопка *Общие*, переадресовывает на страницу детальных настроек выбранного раздела;
- кнопка *Оповещения* для отображения настроек частоты оповещений пользователя выбранным сервисом;
- кнопка *Назад* служит для возврата на главную страницу сервиса.

5.5. Главная страница настроек частоты оповещений

Страница предназначена для просмотра и выбора периода фоновых оповещений пользователя (рис. 5.7). Другими словами, осуществляется выбор, насколько часто необходимо присылать пользователю новую информацию. Данная страница имеет одинаковую структуру для всех сервисов.

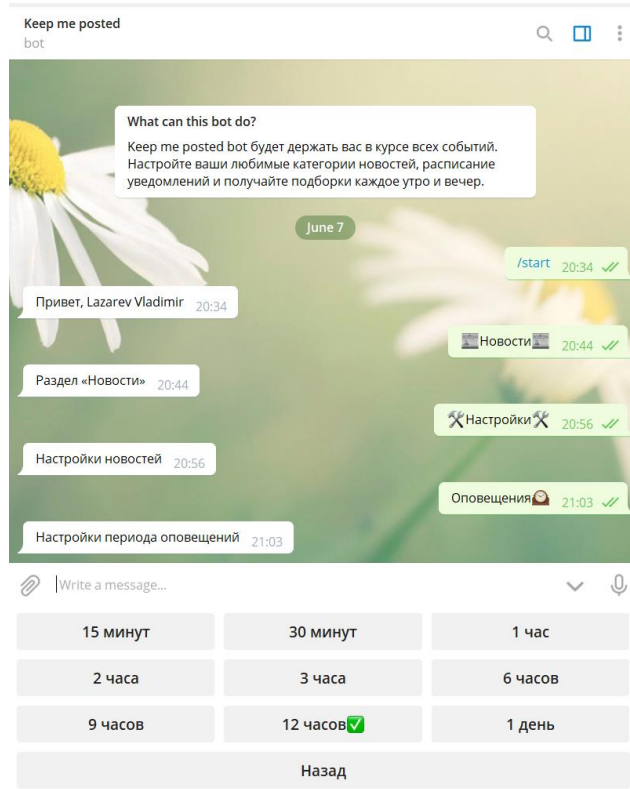


Рис. 5.7. Главная страница настроек оповещений сервисов

Страница содержит следующие элементы управления:

- кнопки с периодом времени, выбранный вариант выделяется галочкой, по умолчанию период устанавливается в *12 часов*;
- кнопка *Назад* служит для возврата на главную страницу настроек сервиса.

5.6. Страница настроек рассылки новостей

Страница предназначена для управления пользовательскими настройками, касающихся сервиса подбора новостей. Образец внешнего вида представлен на рис. 5.8.

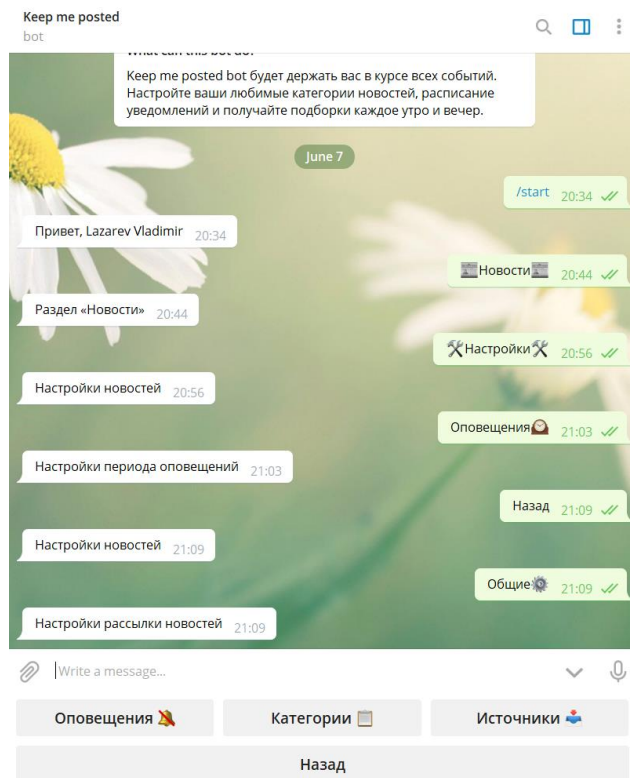


Рис. 5.8. Страница настроек сервиса рассылки новостей

Страница содержит следующие элементы управления:

- кнопка *Оповещения*, отражает текущее состояние активности оповещения: зачеркнутый колокольчик — оповещения отключены, включены в обратном случае;
- кнопка *Категории* переадресовывает на страницу просмотра, добавления и удаления отслеживаемых пользователем категорий;
- кнопка *Источники* переадресовывает на страницу просмотра, добавления и удаления запрещенных пользователем источников (данные из перечисленных информационных порталов не будут отображаться в подборке новостей и оповещениях);
- кнопка *Назад* служит для возврата на главную страницу настроек сервиса.

5.7. Страница настроек сервиса Twitter

Страница предназначена для управления пользовательскими настройками, касающихся сервиса *Twitter*. Образец внешнего вида представлен на рис. 5.9.

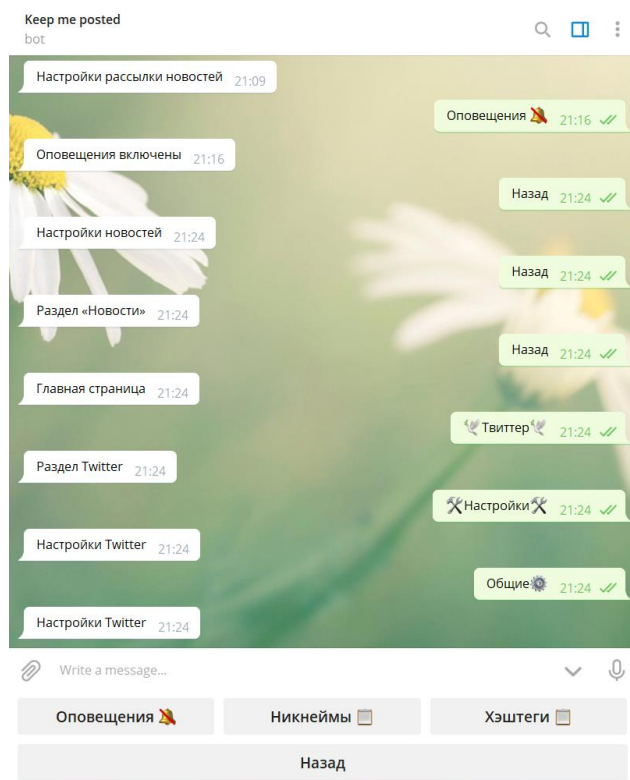


Рис. 5.9. Страница настроек сервиса Twitter

Страница содержит следующие элементы управления:

- кнопка *Оповещения*, отражает текущее состояние активности оповещения: зачеркнутый колокольчик — оповещения отключены, включены в обратном случае;
- кнопка *Никнеймы* переадресовывает на страницу просмотра, добавления и удаления отслеживаемых пользователей социальной сети *Twitter*;
- кнопка *Хэштеги* переадресовывает на страницу просмотра, добавления и удаления отслеживаемых пользователем хэштегов (будут отображаться только те *твиты*, содержащие как минимум один из указанных *хэштегов*);
- кнопка *Назад* служит для возврата на главную страницу настроек сервиса.

5.8. Страница настроек рассылки погоды

Страница предназначена для управления пользовательскими настройками, касающихся сервиса погоды. Образец внешнего вида представлен на рис. 5.10.

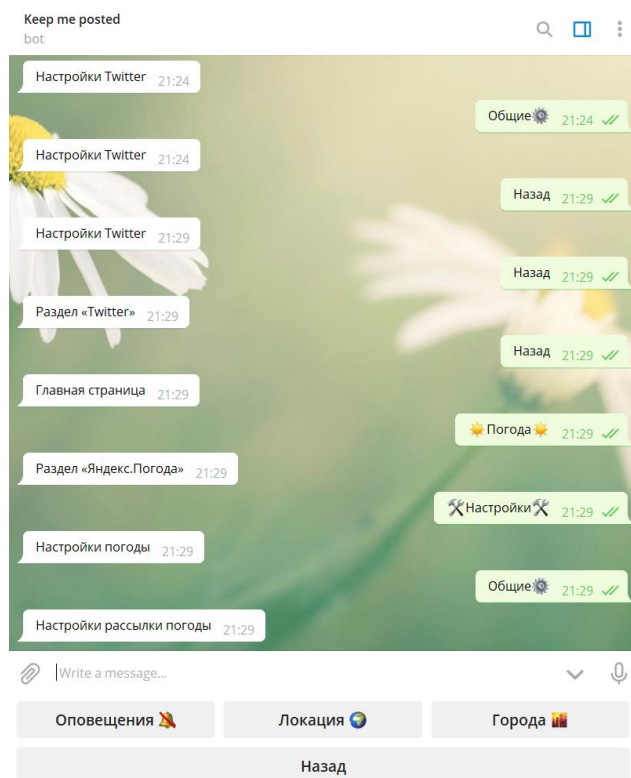


Рис. 5.10. Страница настроек сервиса погоды

Страница содержит следующие элементы управления:

- кнопка *Оповещения*, отражает текущее состояние активности оповещения: зачеркнутый колокольчик — оповещения отключены, включены в обратном случае;
- кнопка *Локация* из-за ограничений *Telegram* работает только на смартфонах — отправляется приложению текущая геолокация пользователя *Telegram*, через API *Яндекс.Геокодер* приложение получает наименование населенного пункта и добавляет его в список отслеживаемых городов;
- кнопка *Города* переадресовывает на страницу просмотра, добавления и удаления отслеживаемых пользователем городов;
- кнопка *Назад* служит для возврата на главную страницу настроек сервиса.

5.9. Страница модификаций пользовательских фильтров

Страница предназначена для просмотра, добавления и удаления каких-либо условий пользовательских фильтров. Образец внешнего вида представлен на рис. 5.11.

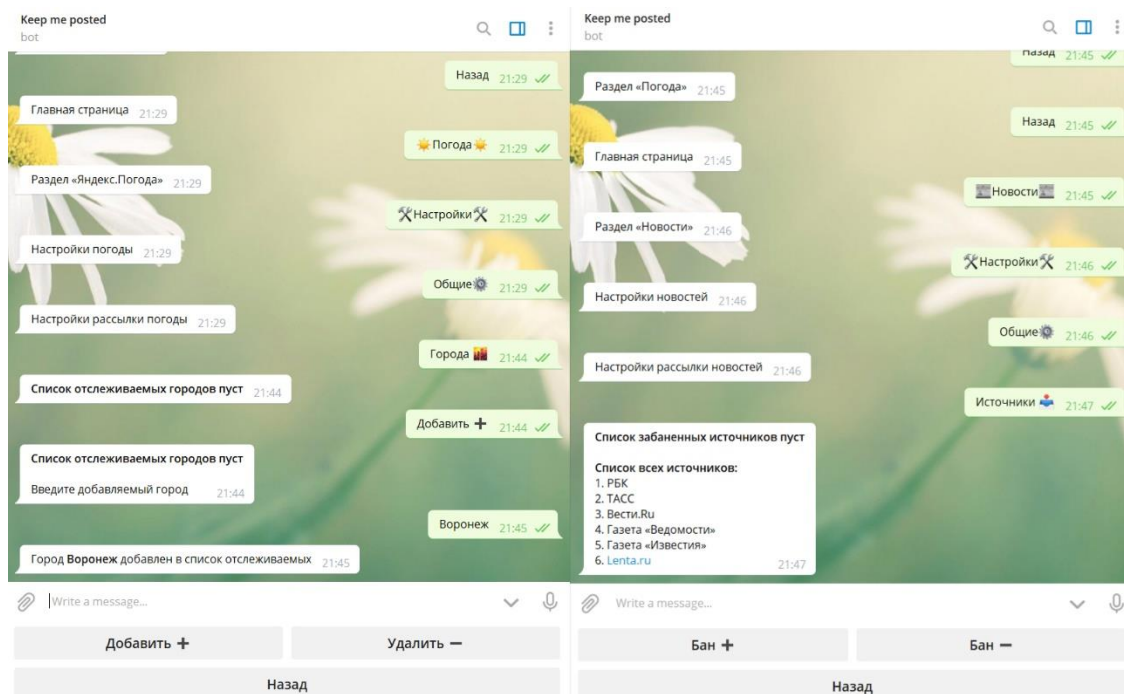


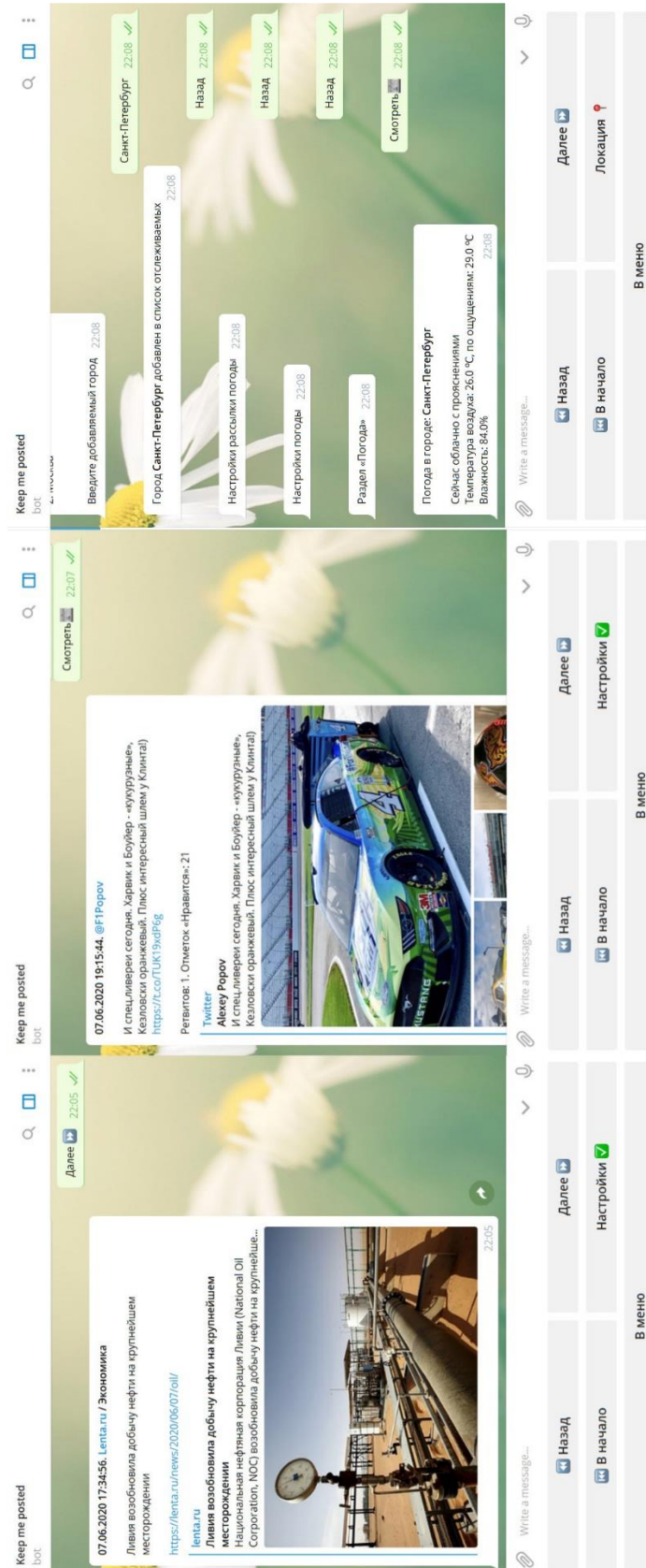
Рис. 5.11. Страница добавления и удаления фильтров

Страница содержит следующие элементы управления:

- по нажатию на кнопку *Добавить* (*Бан +*) пользователю показывается текущий список условий, предлагается ввести новый элемент в блоке отправки сообщения. Если такой элемент уже находится в списке, добавление отменяется, показывается соответствующее сообщение;
- кнопка *Удалить* (*Бан –*) работает обратно кнопке *Добавить* со всеми теми же проверками;
- кнопка *Назад* служит для возврата на главную страницу настроек сервиса.

5.10. Страница просмотра информации

Страница предназначена для просмотра контента выбранного ранее раздела. Образец внешнего вида представлен на рис. 5.12.

Рис. 5.12. Страницы просмотра новостей, *твитов*, погоды

Страница содержит следующие элементы управления:

- кнопка *Назад* служит для возврата на предыдущий просмотренный элемент, если такого не было, приложение покажет соответствующее сообщение;
- кнопка *Далее* для просмотра следующего элемента;
- кнопка *В начало* сбрасывает всю историю просмотренных элементов и показывает самый новый элемент раздела;
- кнопка *Настройки*, отражает текущее состояние активности пользовательских настроек раздела: красный крестик – настройки отключены (показывается вся «свежая» информация в базе), включены в обратном случае;
- кнопка *Локация* запрашивает текущую локацию пользователя *Telegram*, обрабатывает координаты и по ним формирует прогноз погоды;
- кнопка *В меню* возвращает на главную страницу выбранного ранее раздела.

6. Реализация

6.1. Структура приложения

Схема взаимодействия модулей приложения представлена на схеме компонентов на рис. 6.1, где *Telegram-бот* и *Веб-страница* – клиентские части приложения, отвечающие за отображение информации и взаимодействие с пользователем, а *Сервер* – серверная часть.



Рис. 6.1. Схема взаимодействия модулей приложения

Приложение построено на основе REST-архитектуры. *REST* (*Representational state transfer*) – это стиль архитектуры программного обеспечения для работы в сети Интернет, в котором передача состояния ресурсов осуществляется методом протокола HTTP. В таком случае действия над данными осуществляются с помощью методов:

- GET – для получения данных;
- POST – для добавления;
- PUT – для полной замены;
- PATCH – для частичного изменения полей в данных;
- DELETE – для удаления.

Данные отдаются ровно в том же самом виде, в котором и хранятся (JSON).

Связь с сервисом погоды и геокодирования реализуется с помощью *Геодекодера API Яндекс.Карт* и *API Яндекс.Погода*. Данные берутся из базы данных *Яндекс*. API данных сервисов позволяет производить определение

местоположения пользователя, получение информации о фактической погоде в населенном пункте и прогноз на неделю. Для полноценной работы нужно получить ключ приложения. С помощью уникального ключа компания *Яндекс* может отслеживать приложения, работающие с сервисом API и в случае надобности связаться с владельцем приложения [9].

6.2. Серверная часть приложения

Серверная часть приложения написана с использованием фреймворка *Spring Boot 2*, представляет собой классическую реализацию паттерна MVC. Структура паттерна представлена на рис. 6.2.

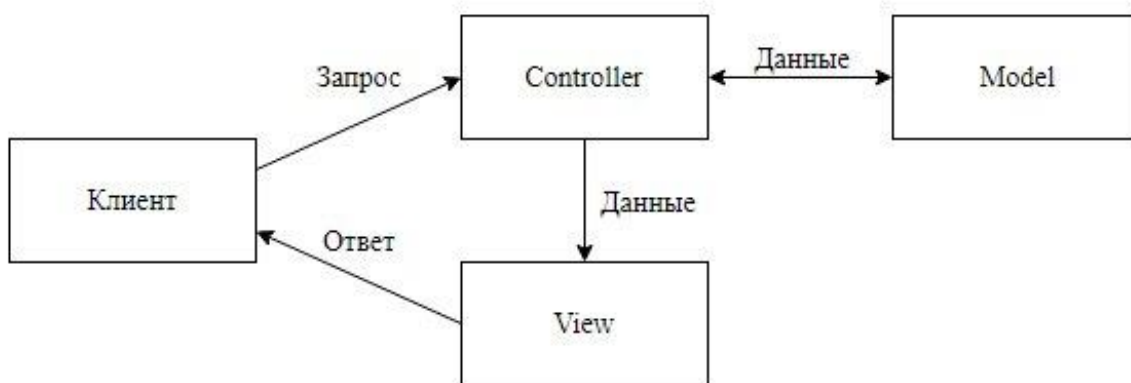


Рис. 6.2. Структура паттерна MVC

Данный паттерн подразумевает разделение данных приложения на три независимых компонента, таким образом позволяет добиться возможности независимых изменений частей приложения и повышения читабельности кода.

6.2.1. Модель данных

Физическая модель данных состоит из следующих таблиц:

- *news_category*;
- *news_item*;
- *news_item_category_list*;
- *news_settings*;
- *news_settings_news_categories*;
- *news_settings_news_sources*;

- *news_settings_viewed_news;*
- *news_source;*
- *notification_service_settings;*
- *notification_service_settings_news_settings;*
- *notification_service_settings_twitter_settings;*
- *notification_service_settings_weather_settings;*
- *telegram_chat;*
- *telegram_contact;*
- *telegram_location;*
- *telegram_message;*
- *telegram_update;*
- *tweet;*
- *twitter_hashtag;*
- *twitter_people;*
- *twitter_settings;*
- *twitter_settings_twitter_hashtags;*
- *twitter_settings_twitter_people;*
- *twitter_settings_viewed_tweets;*
- *users;*
- *weather;*
- *weather_city;*
- *weather_fact;*
- *weather_forecast;*
- *weather_forecast_hours;*
- *weather_forecast_parts;*
- *weather_forecasts;*
- *weather_hour;*
- *weather_info;*
- *weather_part;*

- *weather_settings*;
- *weather_settings_cities*;
- *weather_settings_viewed_cities*;
- *weather_tz_info*.

Физическая модель базы данных приложения представлена на рис. 6.3, рис. 6.4 и рис 6.5.

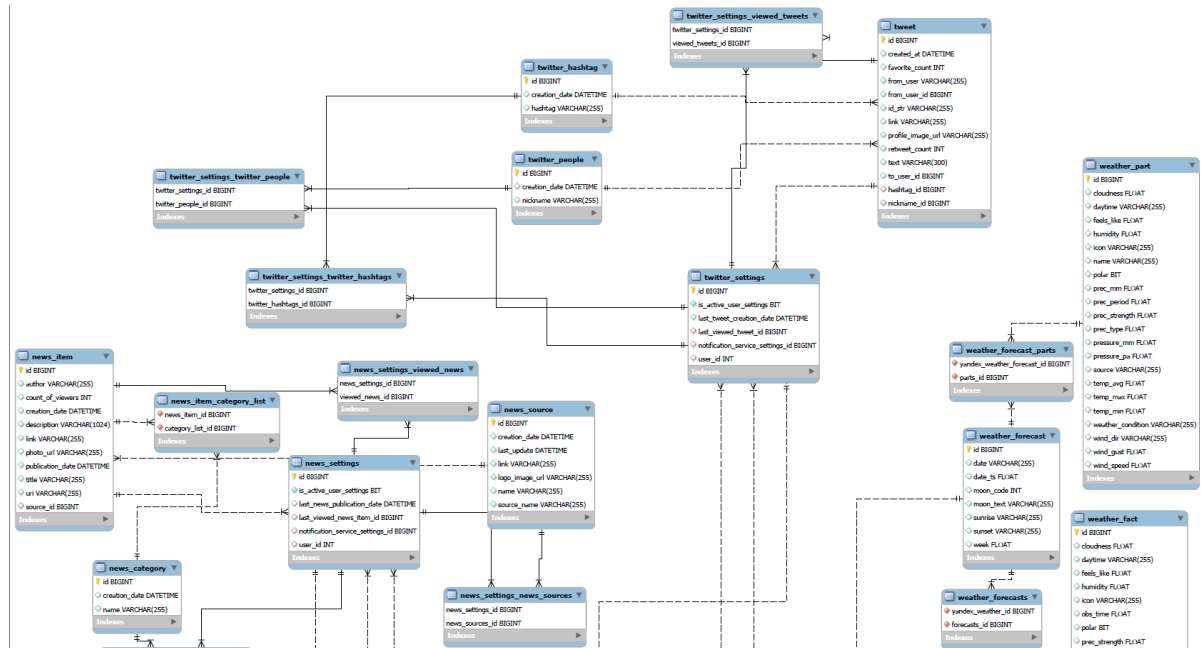


Рис. 6.3. Физическая модель данных (часть 1)

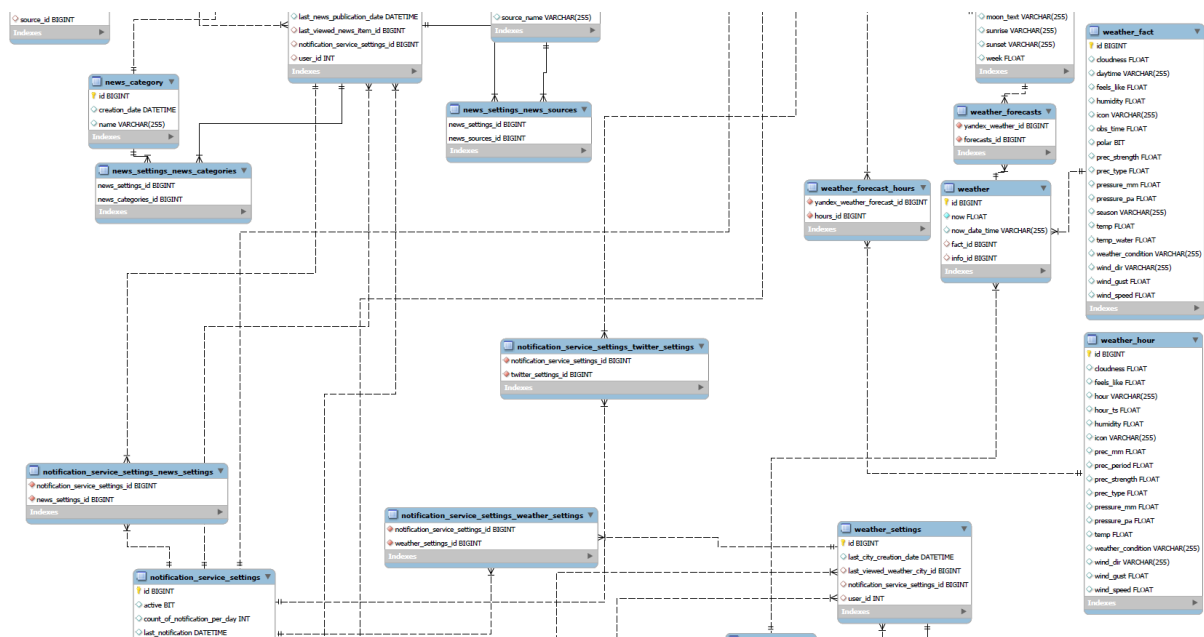


Рис. 6.4. Физическая модель данных (часть 2)

6.2.2. Реализованные классы

6.2.2. Реализованные классы

- `NewsController` — контроллер для отображения самых просматриваемых новостей в разрезе выбранного промежутка времени;

- `NewsController` – контроллер для отображения самых просматриваемых новостей в разрезе выбранного промежутка времени;
- `SendMessageController` – контроллер для ручного оповещения пользователей.

- `YandexGeoCoderService` – сервис для обработки координат в наименование населенного пункта и наоборот;
- `NewsService` – сервис, который в фоновом режиме с заданным интервалом опрашивает все новостные ресурсы и сохраняет новые данные;

- `YandexGeoCoderService` — сервис для обработки координат в наименование населенного пункта и наоборот;
- `NewsService` — сервис, который в фоновом режиме с заданным интервалом опрашивает все новостные ресурсы и сохраняет новые данные;
- `TwitterService` — сервис для получения новых *твитов* по заданным пользователями параметрами в фоновом режиме с заданной периодичностью;
- `NotificationService` — сервис, который рассылает уведомления пользователям в фоне, исходя из их настройки частоты оповещений;

- `YandexWeatherService` – сервис для получения фактической и прогнозируемой погоды;
- `TelegramUpdateService` – сервис, обрабатывающий все обновления и действия, которые происходят с *Telegram*-ботом.

Список классов, преобразующих объекты библиотеки *TelegramBots* от *Spring Boot* в объекты базы данных:

- `Mapper` – интерфейс маппера;
- `AbstractMapper` – абстрактный класс для маппера;
- `TelegramChatMapper` – конвертер класс *Chat* из библиотеки в класс *TelegramChat*;
- `TelegramContactMapper` – конвертер класс *Contact* из библиотеки в класс *TelegramContact*;
- `TelegramLocationMapper` – конвертер класс *Location* из библиотеки в класс *TelegramLocation*;
- `TelegramMessageMapper` – конвертер класс *Message* из библиотеки в класс *TelegramMessage*;
- `TelegramUpdateHandler` – конвертер класс *Update* из библиотеки в класс *TelegramUpdate*;
- `TelegramUserMapper` – конвертер класс *User* из библиотеки в класс *TelegramUser*.

Список классов-обработчиков для *Telegram*-бота:

- `TelegramMessageHandler` – интерфейс для классов-обработчиков;
- `TelegramHandler` – абстрактный класс, в котором инициализированы некоторые общие методы дочерних классов и перечень общих переменных;
- `NewsTelegramHandler` – класс-обработчик, отвечающий за работу и навигацию раздела *Новости*;
- `SettingsTelegramHandler` – класс-обработчик, отвечающий за настройку оповещений;

- `TwitterMessageHandler` – класс-обработчик, отвечающий за работу и навигацию раздела *Twitter*;
- `WeatherSettingsHandler` – класс-обработчик, отвечающий за работу и навигацию раздела *Погода*.

Листинг серверной части приложения приведен в Приложении 1.

6.3. Клиентская часть приложения

Клиентская часть приложения реализована с помощью фреймворка *Angular*, а также библиотеки для стилизации приложения *Angular Material*. Схема работы клиентской стороны приложения, написанной на *Angular* представлена на рис. 6.6 [5].

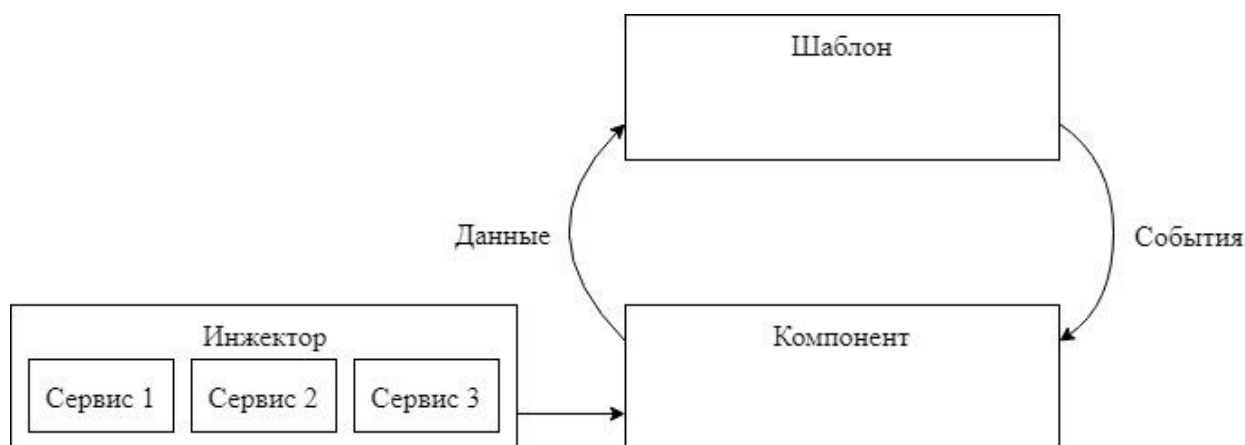


Рис. 6.6. Схема клиентского приложения на *Angular*

Веб-приложение, написанное с использованием фреймворка *Angular*, имеет структуру SPA (Single Page Application). Это означает, что фактического перехода между страницами не происходит, а только меняется содержимое DOM-структуры.

В терминологии *Angular* все реализованные классы можно разделить на компоненты и сервисы. Компоненты отвечают за отображение данных пользователю, а сервисы – за обмен данными между клиентом и сервером. Листинг клиентской части приложения приведен в Приложении 2.

Список реализованных компонент:

- `HeaderComponent` – компонент, отображающий шапку сайта;

- HomeComponent – компонент, отображающий главную страницу сайта;
- FooterComponent – компонент, отображающий подвал сайта;
- NewsDataComponent – компонент, содержащий таблицу с рейтингом новостей, количеством просмотров и ссылками на них;
- TimeSwitcherComponent – компонент, отвечающих за период времени, по которому фильтруются новости.

Список реализованных сервисов:

- NewsDataService – сервис, получающий страницы новостей, исходя из выбранного периода времени, размера таблицы и текущей страницы в таблице.

7. 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. «Добавление и удаление пользовательских настроек».

Цель теста. Проверка корректности выполнения функций добавления и удаления пользовательских настроек.

Описание теста. На странице добавления категорий новостей попытаться добавить элемент, который уже находится в списке. Попробовать удалить элемент, которого нет в списке.

Результат. Программа вместо добавления и удаления присылает пользователю сообщения (рис. 7.1).

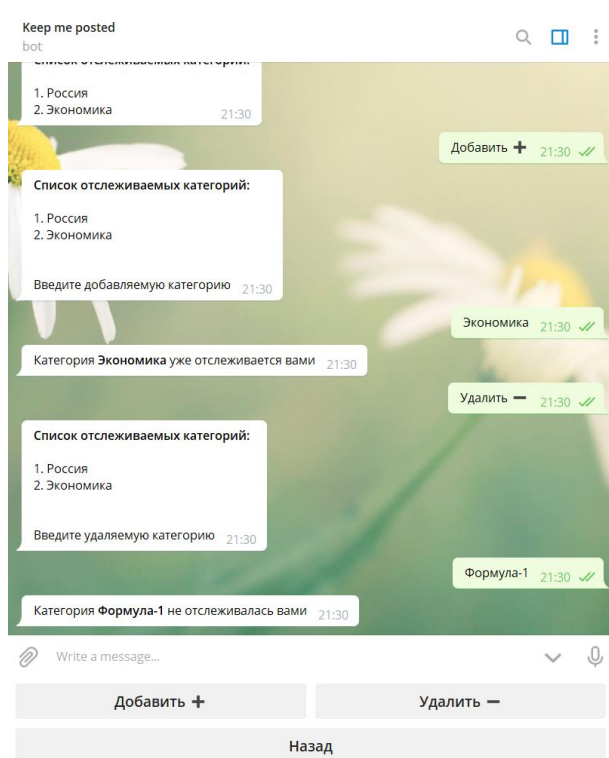


Рис. 7.1. Сообщения при добавлении и удалении категорий

Тест 2. «Проверка слайдера при просмотре новостей».

Цель теста. Проверка корректности работы просмотра новостей.

Описание теста. При просмотре новостей выбрать несколько следующих элементов, затем постепенно возвращаться на предыдущие. Когда пользователь находится на самой первой просмотренной новости, при попытке клика на кнопку *Назад* программа должна выдать сообщение. При включении и отключении пользовательских настроек, список просмотренных должен очищаться, а пользователю показываться самая последняя новость.

Результат. Приложение будет показывать предупреждения, после отключения пользовательских настроек отображается самая последняя новость по не отслеживаемой пользователем категории (рис. 7.2).

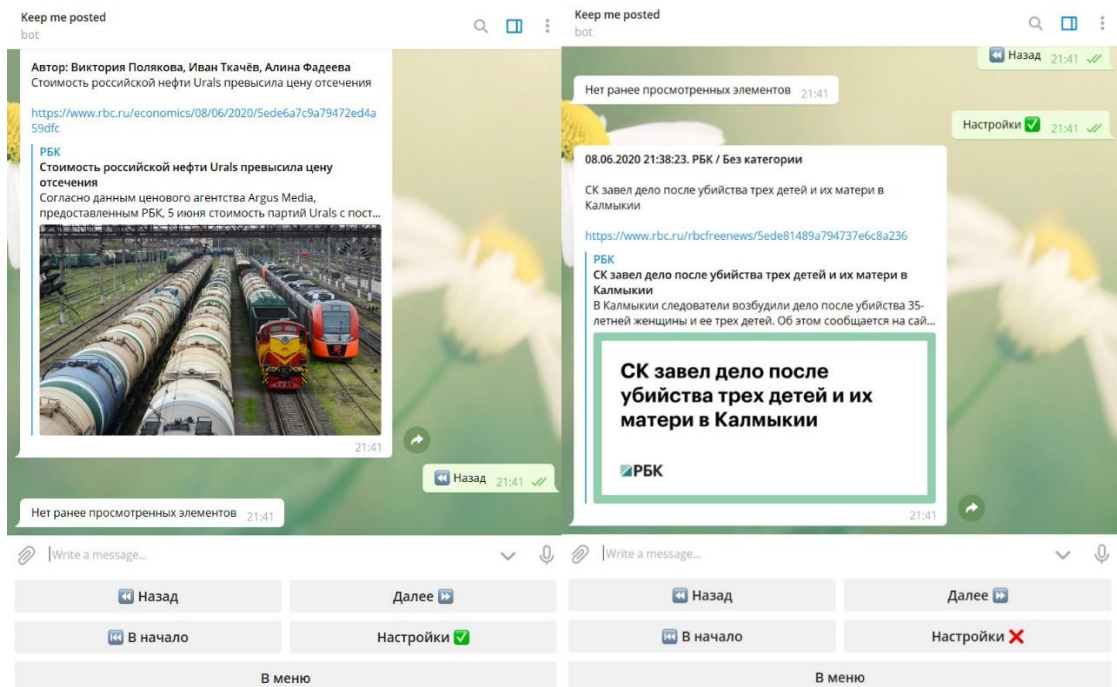


Рис. 7.2. Страница просмотра новостей и сообщения на ней

Тест 3. «Определение населенного пункта пользователя по его координатам».

Цель теста. Проверка корректности работы сервиса геодекодирования.

Описание теста. Поделиться геолокацией, когда город, в котором находится в списке отслеживаемых и нет.

Результат. Город по координатам успешно определен, сообщения о наличии города в списке отслеживаемых отображается, город добавляется во втором случае (рис. 7.3).

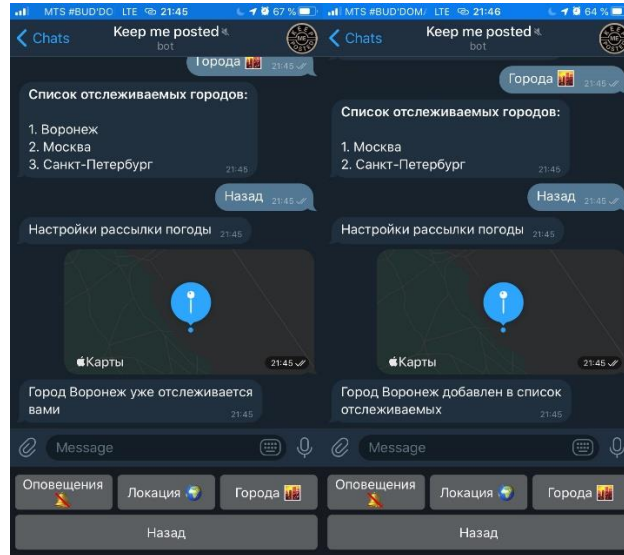


Рис. 7.3. Определение города по месторасположению пользователя

Тест 4. «Проверка сервиса уведомлений»

Цель теста. Проверка корректности сервиса отправки фоновых уведомлений.

Описание теста. Включить оповещения по сервису погоды, подождать заданный период, убедиться, что сообщения получены.

Результат. Получена информация обо всех отслеживаемых городах без действий со стороны пользователя (рис. 7.4).



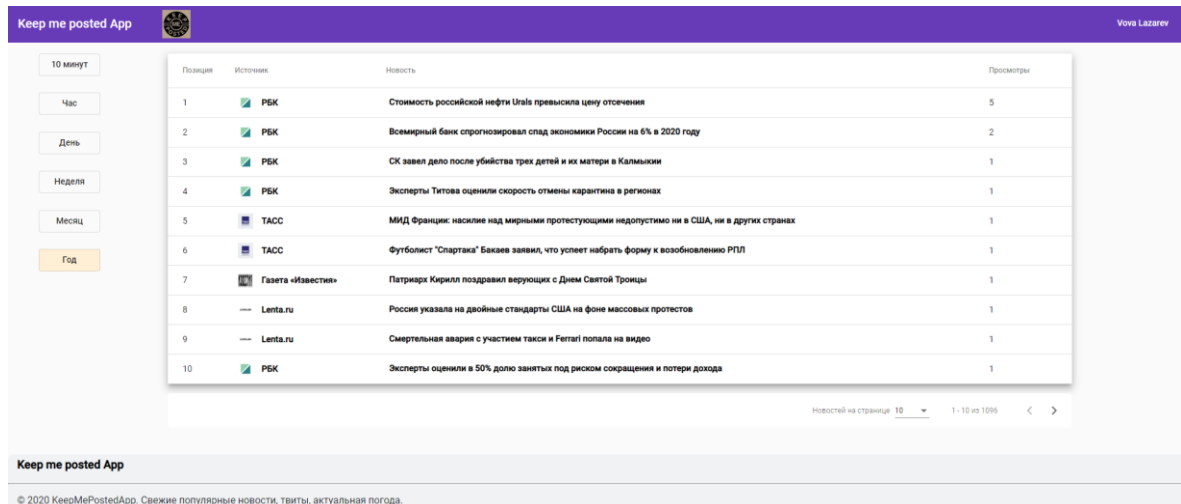
Рис. 7.4. Уведомления по отслеживаемым городам

Тест 5. «Проверка работы сайта».

Цель теста. Проверка корректности работы сайта, в частности работы всех кнопок и фонового обновления данных.

Описание теста. Новые данные должны попадать в таблицу без обновления страницы, данные – упорядочиваться по количеству просмотров.

Результат. Постановленные условия теста успешно выполняются (рис. 7.5).



Позиция	Источник	Новость	Просмотры
1	РЕК	Стоимость российской нефти Urals превысила цену отсечения	5
2	РЕК	Всемирный банк спрогнозировал спад экономики России на 6% в 2020 году	2
3	РЕК	СК завел дело после убийства трех детей и их матери в Калмыкии	1
4	РЕК	Эксперты Титова оценили скорость отмены карантина в регионах	1
5	TACC	МИД Франции: насилие над мирными протестующими недопустимо ни в США, ни в других странах	1
6	TACC	Футболист "Спартак" Бакаев заявил, что успеет набрать форму к возобновлению РПЛ	1
7	Газета «Известия»	Патриарх Кирилл поздравил верующих с Днем Святой Троицы	1
8	Lenta.ru	Россия указала на двойные стандарты США на фоне массовых протестов	1
9	Lenta.ru	Смертельная авария с участием такси и Ferrari попала на видео	1
10	РЕК	Эксперты оценили в 50% долю занятых под риском сокращения и потери дохода	1

Новости на странице: 10 | 1 - 10 из 1096

Keep me posted App

© 2020 KeepMePostedApp. Свежие популярные новости, твиты, актуальная погода.

Рис. 7.5. Веб-сайт приложения с обновляющимися данными

Заключение

Разработано веб-приложение *KeepMePosted*, представляющее собой новостной агрегатор, получающий информацию как через открытое API, так и посредством парсинга RSS-каналов.

Приложение рассчитано на два типа пользователей:

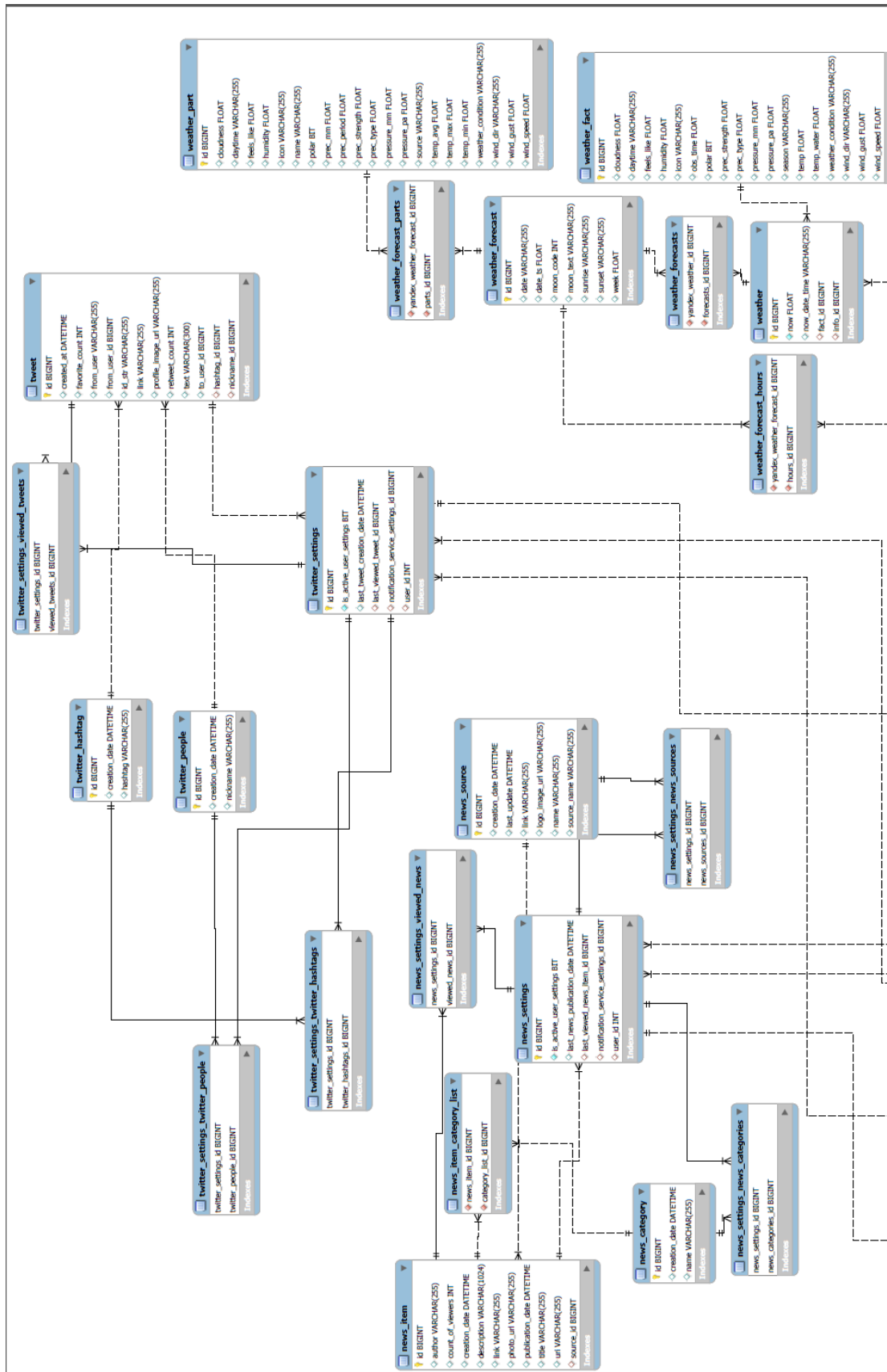
- пользователь, использующий клиент Telegram-бота (имеет функции детальной настройки, оповещений и т. д.);
- пользователь, использующий клиент веб-сайта (может просматривать самые просматриваемые новости в разрезе указанного периода времени).

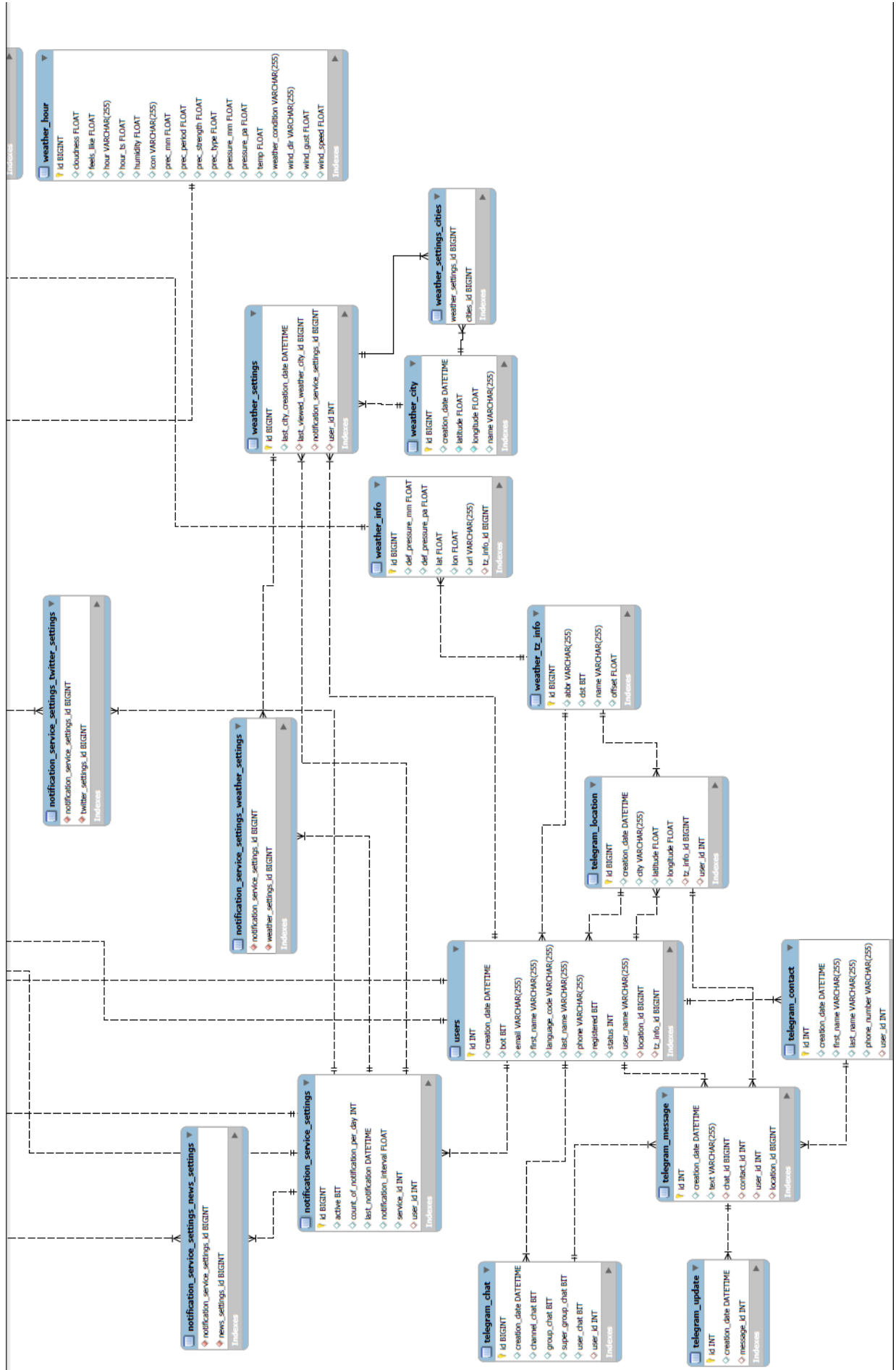
KeepMePosted представляет следующие возможности:

- просмотр самых свежих новостей;
- добавление и удаление категорий новостей, по которым пользователь будет получать обновления;
- блокировка и снятие блокировки с новостных ресурсов;
- просмотр самых новых *твитов* социальной сети Twitter;
- добавление и удаление отслеживаемых пользователей *Twitter* и *хэштегов*;
- просмотр 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;
- просмотр новостей, *твитов* и погоды применяя пользовательские настройки, так и без них;
- включение и отключение оповещений, настройка их частоты отдельно для каждого из сервиса.

Список литературы

1. Walls C. Spring in Action / C. Walls. – 5-е изд. – [s. l.] : Manning Publications, 2018. – 520 с.
2. Spring [Электронный ресурс] // Pivotal Software, Inc., 2020. URL: <https://spring.io/> (дата обращения 25.03.2020).
3. Cosmina I. Pro Spring 5 : An In-Depth Guide to the Spring Framework and Its Tools / I. Cosmina, R. Harrop, C. Schaefer, C. Ho – 5-е изд. – [s. l.] : Apress, 2017. – 849 с.
4. Хорстманн К. Java. Библиотека профессионала / К. Хорстман, Г. Корнелл – 10-е изд. – Санкт-Петербург : Альфа-книга, 2016. – Т. 2 : Расширенные средства программирования. – 976 с.
5. Angular [Электронный ресурс] // Google, 2010-2020. URL: <https://angular.io/> (дата обращения 15.05.2020).
6. Рефакторинг и Паттерны проектирования [Электронный ресурс] // URL: <https://refactoring.guru/ru> (дата обращения 02.05.2020).
7. Сьерра К. Изучаем Java / К. Сьерра, Б. Бейтс – 2-е изд. – Москва : Эксмо, 2013. – 717 с.
8. Шилдт Г. Java 8. Руководство для начинающих / Г. Шилдт – 6-е изд. – Москва : Вильямс, 2016. – 866 с.
9. API Яндекс.Погоды, Документация [Электронный ресурс] // URL: <https://yandex.ru/dev/weather/doc/dg/concepts/forecast-info-docpage/> (дата обращения 25.04.2020).





Приложение 2. Листинг клиентской части приложения

NewsDataService.ts

```
import {Injectable} from '@angular/core';
import {ApiService, urlDB} from '../api/api.service';
import {HttpClient, HttpHeaders} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class NewsDataService {

  newsOnPage;
  numberOfPage;
  typeOfTimePeriod;

  news$: any;

  constructor(private apiService: ApiService,
               private httpClient: HttpClient) {
    this.newsOnPage = 10;
    this.numberOfPage = 0;
    this.typeOfTimePeriod = '10min';

    this.getNewsFromDB(this.typeOfTimePeriod);
    setInterval(() => this.getNewsFromDBByCountNews(this.newsOnPage,
this.numberOfPage), 10000);
  }

  getNewsFromDB(typeOfTimePeriod: string): any {
    this.typeOfTimePeriod = typeOfTimePeriod;
    this.apiService.get('/news?numberOfPage=' + this.numberOfPage +
      '&newsOnPage=' + this.newsOnPage +
      '&typeOfTimePeriod=' + typeOfTimePeriod).subscribe(data => {
      this.news$ = data;
    });
  }

  getNewsFromDBByCountNews(newsOnPage: number, numberOfPage: number): any {
    this.newsOnPage = newsOnPage;
    this.numberOfPage = numberOfPage;

    this.apiService.get('/news?numberOfPage=' + numberOfPage +
      '&newsOnPage=' + newsOnPage +
      '&typeOfTimePeriod=' + this.typeOfTimePeriod).subscribe(data => {
      this.news$ = data;
    });
  }

  getNews(): any {
    return this.news$;
  }

  updateCountOfViews(id: number) {
    return this.httpClient.post(urlDB + '/news/add_count_of_views', id, {
      headers: new HttpHeaders().set('Content-type', 'application/json'),
    }).subscribe(this.getNewsFromDB(this.typeOfTimePeriod));
  }
}
```

NewsDataComponent.ts

```

import {Component, EventEmitter, OnInit, Output} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {Router} from '@angular/router';
import {ApiService} from '../api/api.service';
import {NewsDataService} from './news-data.service';
import {PageEvent} from '@angular/material/paginator';

export interface News {
  position: number;
  delta: number;
  logo: string;
  title: string;
  countOfViews: number;
}

@Component({
  selector: 'app-news-data',
  templateUrl: './news-data.component.html',
  styleUrls: ['./news-data.component.css']
})
export class NewsDataComponent implements OnInit {

  allNews: any;
  countOfNews: number;
  countOfPages: number;

  pageSize: number;
  pageIndex: number;

  displayedColumns: string[] = ['position', 'delta', 'logo', 'title',
'countOfViewers'];
  pageEvent: PageEvent;

  constructor(private httpClient: HttpClient,
               private router: Router,
               private apiService: ApiService,
               private newsDataService: NewsDataService) {
    this.pageSize = 10;
    this.pageIndex = 0;
  }

  ngOnInit(): void {
    setInterval(() => this.setAllNews(), 10);
  }

  setAllNews() {
    const data = this.newsDataService.getNews();
    if (data === undefined) {
      return;
    }

    if (data[0].length === 0) {
      this.pageIndex = 0;
      this.newsDataService.getNewsFromDBByCountNews(this.pageSize,
this.pageIndex);
      this.setAllNews();
    }
    this.allNews = data[0];
    this.countOfNews = data[1];
    this.countOfPages = data[2];
  }

  onClickNews(element: any): void {
    this.newsDataService.updateCountOfViews(element.id);
  }

```

```

    }

    UpdateCount(event: any, element: any): void {
        if (event.button === 1) {
            this.newsDataService.updateCountOfViews(element.id);
        }
    }

    setPageSizeOptions(setPageSizeOptionsInput: any): void {
        this.pageSize = setPageSizeOptionsInput.pageSize;
        this.pageIndex = setPageSizeOptionsInput.pageIndex;

        this.newsDataService.getNewsFromDBByCountNews(this.pageSize,
this.pageIndex);
        this.setAllNews();
        // if (this.allNews[0].length === 0) {
        //     this.pageIndex = 0;
        //     this.newsDataService.getNewsFromDBByCountNews(this.pageSize,
this.pageIndex);
        //     this.setAllNews();
        // }
    }
}

```

NewsDataComponent.html

```

<div class="newsTable">

    <!-- <p>{{timePeriodService.typeOfTimePeriod|async}}</p>-->

    <table mat-table [dataSource]="allNews" class="mat-elevation-z8">

        <ng-container class="position" matColumnDef="position">
            <th mat-header-cell *matHeaderCellDef>Позиция</th>
            <td mat-cell *matCellDef="let i=index"> {{pageIndex * pageSize + i +
1}} </td>
        </ng-container>

        <ng-container class="delta" matColumnDef="delta">
            <th mat-header-cell *matHeaderCellDef></th>
            <td mat-cell *matCellDef="let element"> {{element.delta}} </td>
        </ng-container>

        <ng-container matColumnDef="logo">
            <th mat-header-cell *matHeaderCellDef> Источник</th>
            <td class="logo" mat-cell *matCellDef="let element">
                <a [href]="element.uri" target="_blank"><img
[src]="element.source.logoImageUrl" alt="Фото источника"

(click)="OnClickNews(element)"

(auxclick)="UpdateCount($event, element)"></a>
                <a [href]="element.uri" (click)="OnClickNews(element)"
(auxclick)="UpdateCount($event, element)"
                target="_blank"> {{element.source.name}}</a>
            </td>
        </ng-container>

        <ng-container class="title" matColumnDef="title">
            <th mat-header-cell *matHeaderCellDef> Новость</th>
            <td mat-cell *matCellDef="let element">
                <a [href]="element.uri" (click)="OnClickNews(element)"
(auxclick)="UpdateCount($event, element)"
                target="_blank">{{element.title}}</a>
            </td>
        </ng-container>
    </table>

```

```

</ng-container>

<ng-container class="countOfViewers" matColumnDef="countOfViewers">
  <th mat-header-cell *matHeaderCellDef>Просмотры</th>
  <td mat-cell *matCellDef="let element"> {{element.countOfViewers}}
</td>
</ng-container>

<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
<tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>

<mat-paginator #paginator
  [length]=countOfNews
  [pageIndex]="0"
  [pageSize]="10"
  [pageSizeOptions]="[5, 10, 25, 100]"
  (page)="pageEvent = setPageSizeOptions($event)">
</mat-paginator>
</div>

<!--<mat-icon>expand_less</mat-icon>-->
<!--expand_more expand more icon <mat-icon>expand_more</mat-icon>-->

```

TimeSwitcher.ts

```

import {Component, Injectable, OnInit} from '@angular/core';
import {Subject} from 'rxjs';
import {NewsDataService} from '../news-data/news-data.service';

@Injectable({
  providedIn: 'root',
})
export class TimePeriodService {
  typeOfTimePeriod: Subject<string> = new Subject<string>();

  constructor(private newsDataService: NewsDataService) {
    this.typeOfTimePeriod.next('10min');
    this.typeOfTimePeriod.asObservable().subscribe((data) => {
      this.newsDataService.getNewsFromDB(data);
    });
  }
}

const LIST: string[] = ['10 минут', 'Час', 'День', 'Неделя', 'Месяц', 'Год'];

@Component({
  selector: 'app-time-switcher',
  templateUrl: './time-switcher.component.html',
  styleUrls: ['./time-switcher.component.css']
})
export class TimeSwitcherComponent implements OnInit {

  public list: string[] = LIST;
  public activeItem: string;

  constructor(private timePeriodService: TimePeriodService) {
    this.activeItem = this.list[0];
  }

  ngOnInit(): void {
  }
}

```

```

OnSelectedItem(item: string): void {
    this.activeItem = item;
    switch (item) {
        case '10 минут': {
            this.timePeriodService.typeOfTimePeriod.next('10min');
            break;
        }
        case 'Час': {
            this.timePeriodService.typeOfTimePeriod.next('1h');
            break;
        }
        case 'День': {
            this.timePeriodService.typeOfTimePeriod.next('1d');
            break;
        }
        case 'Неделя': {
            this.timePeriodService.typeOfTimePeriod.next('1w');
            break;
        }
        case 'Месяц': {
            this.timePeriodService.typeOfTimePeriod.next('1m');
            break;
        }
        case 'Год': {
            this.timePeriodService.typeOfTimePeriod.next('1y');
            break;
        }
    }
}
}
}

```

TimeSwitcher.html

```

<li *ngFor="let item of list"
    class="topBarItems" (click)="OnSelectedItem(item)"
    [class.active]="item===activeItem">
    <a mat-stroked-button>{{item}}</a>
</li>

```

Приложение 3. Листинг серверной части приложения

NewsController

```
package application.controller;

import application.data.model.news.NewsItem;
import application.data.repository.news.NewsItemRepository;
import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.http.HttpStatus;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.web.bind.annotation.*;

import java.lang.reflect.Array;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Date;
import java.util.Optional;
import java.util.Set;

@RestController
@FieldDefaults(level = AccessLevel.PRIVATE)
@RequiredArgsConstructor
@RequestMapping(value = "/news", produces = "application/json")
@CrossOrigin("*")
public class NewsController {

    @Autowired
    NewsItemRepository newsItemRepository;

    @GetMapping(params = {"numberOfPage", "newsOnPage", "typeOfTimePeriod"},
        produces = "application/json")
    public ArrayList<Object> allNews(@RequestParam("numberOfPage") int
        numberOfPage,
                                   @RequestParam("newsOnPage") int
        newsOnPage,
                                   @RequestParam("typeOfTimePeriod") String
        typeOfTimePeriod) {
        PageRequest page = PageRequest.of(numberOfPage, newsOnPage,
            Sort.by("countOfViewers")
                .and(Sort.by("publicationDate").descending())
                .and(Sort.by("creationDate").descending()));

        LocalDateTime ldt = getLocalDateTime(typeOfTimePeriod);
        Date outStart =
            Date.from(ldt.atZone(ZoneId.systemDefault()).toInstant());
        Date outEnd =
            Date.from(LocalDateTime.now().atZone(ZoneId.systemDefault()).toInstant());

        ArrayList result = new ArrayList<>();
        Page<NewsItem> queryResult =
            newsItemRepository.findByPublicationDateBetweenOrderByCountOfViewersDescPublicationDateDescCreationDateDesc(outStart,
                outEnd, page);
        result.add(queryResult.getContent());
    }
}
```

```

        result.add(queryResult.getTotalElements());
        result.add(queryResult.getTotalPages());

        return result;
    }

    @PostMapping(value = "/add_count_of_views", consumes =
"application/json")
    @ResponseStatus(HttpStatus.CHECKPOINT)
    public void updateCountOfViews(@RequestBody long id) {
        Optional<NewsItem> newsItemOptional =
newsItemRepository.findById(id);
        if (newsItemOptional.isPresent()) {
            NewsItem newsItem = newsItemOptional.get();
            newsItem.setCountOfViewers(newsItem.getCountOfViewers() + 1);
            newsItemRepository.save(newsItem);
        }
    }

    private LocalDateTime getLocalDateTime(@RequestParam("typeOfTimePeriod")
String typeOfTimePeriod) {
        LocalDateTime ldt = LocalDateTime.now();
        switch (typeOfTimePeriod) {
            case "10min": {
                ldt = ldt.minusMinutes(10);
                break;
            }
            case "1h": {
                ldt = ldt.minusHours(1);
                break;
            }
            case "1d": {
                ldt = ldt.minusDays(1);
                break;
            }
            case "1w": {
                ldt = ldt.minusWeeks(1);
                break;
            }
            case "1m": {
                ldt = ldt.minusMonths(1);
                break;
            }
            case "1y": {
                ldt = ldt.minusYears(1);
                break;
            }
        }
        return ldt;
    }
}

```

YandexGeoCoderService

```

package application.service.geocoder;

import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import lombok.*;
import lombok.experimental.FieldDefaults;

```

```

import lombok.extern.log4j.Log4j2;
import org.apache.commons.io.IOUtils;
import org.apache.http.NameValuePair;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)
@PropertySource("classpath:yandex.properties")
@Log4j2
public class YandexGeoCoderService {

    static String apiKey;
    static String defaultUrl;

    @Value("${yandex.geoCoder.apiKey}")
    public void setApiKey(String value) {
        apiKey = value;
    }

    @Value("${yandex.geoCoder.url}")
    public void setDefaultUrl(String value) {
        defaultUrl = value;
    }

    public String getCityByCoordinates(String coordinates) {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpGet request = createRequest(coordinates);

        try (CloseableHttpResponse response = httpClient.execute(request)) {
            int statusCode = response.getStatusLine().getStatusCode();
            if (statusCode == 200) {

                JsonObject geoObject = getJsonObject(response);

                return (geoObject == null) ? null :
geoObject.get("name").getAsString();
            } else {
                log.error("Сервис геокодирования не отвечает. Код ответа: " +
statusCode);
                return null;
            }
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    public HashMap<String, Float> getCoordinatesByCity(String city) {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpGet request = createRequest(city);

```



```

try (CloseableHttpResponse response = httpClient.execute(request)) {
    if (response.getStatusLine().getStatusCode() == 200) {
        JsonObject geoObject = getJSONObject(response);
        if (geoObject == null) {
            return null;
        }

        JsonObject point = geoObject.getAsJsonObject("Point");

        String pos = point.get("pos").getAsString();
        String[] posArray = pos.split(" ");

        HashMap<String, Float> resultMap = new HashMap<String,
Float>();

        resultMap.put("longitude", Float.valueOf(posArray[0]));
        resultMap.put("latitude", Float.valueOf(posArray[1]));

        return resultMap;
    } else {
        log.error("Сервис не отвечает");
        return null;
    }

} catch (IOException e) {
    e.printStackTrace();
    return null;
}

}

private HttpGet createRequest(String geocode) {
    StringBuilder requestUrl = new StringBuilder(defaultUrl);

    List<NameValuePair> urlParameters = getListUrlParameters(geocode);

    urlParameters.forEach(nameValuePair -> requestUrl
        .append(nameValuePair.getName())
        .append("=")
        .append(nameValuePair.getValue())
        .append("&"));

    requestUrl.deleteCharAt(requestUrl.length() - 1);

    return new HttpGet(requestUrl.toString());
}

private List<NameValuePair> getListUrlParameters(String geocode) {
    return new ArrayList<NameValuePair>() {{
        add(new BasicNameValuePair("geocode", geocode));
        add(new BasicNameValuePair("apikey", apiKey));
        add(new BasicNameValuePair("format", "json"));
        add(new BasicNameValuePair("kind", "locality"));
        add(new BasicNameValuePair("results", "1"));
    }};
}

private static JsonObject getJSONObject(CloseableHttpResponse response)
throws IOException {
    String str = new
String(IOUtils.toByteArray(response.getEntity().getContent()),
StandardCharsets.UTF_8);

    JsonParser parser = new JsonParser();
    JsonElement element = parser.parse(str);

```

```

        // Начинаем парсить
        JsonObject rootObject = element.getAsJsonObject();
        JsonObject responseObject = rootObject.getAsJsonObject("response");
        JsonObject geoObjectCollectionObject =
responseObject.getAsJsonObject("GeoObjectCollection");
        JsonArray featureMemberObject =
geoObjectCollectionObject.getAsJsonArray("featureMember");

        if (featureMemberObject.size() == 0) {
            log.error("По указанным координатам не найдено адреса");
            return null;
        } else if (featureMemberObject.size() > 1) {
            log.error("По указанным координатам вернули более 1 адреса");
        }

        JsonObject firstAddressBlock =
featureMemberObject.get(0).getAsJsonObject();

        return firstAddressBlock.getAsJsonObject("GeoObject");
    }
}

```

NewsService

```

package application.service.news;

import application.data.model.news.NewsCategory;
import application.data.model.news.NewsItem;
import application.data.model.news.NewsSource;
import application.data.repository.news.NewsCategoryRepository;
import application.data.repository.news.NewsItemRepository;
import application.data.repository.news.NewsSourceRepository;
import com.sun.syndication.feed.module.DCModuleImpl;
import com.sun.syndication.feed.synd.*;
import com.sun.syndication.io.FeedException;
import com.sun.syndication.io.SyndFeedInput;
import com.sun.syndication.io.XmlReader;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.apache.logging.log4j.core.util.KeyValuePair;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

import javax.swing.*;
import java.io.IOException;
import java.net.URL;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)

```

```

@PropertySource("classpath:news.properties")
@Log4j2
@EnableScheduling
@EnableAsync
public class NewsService {

    //region rss-links
    @Value("${news.rbk.rss}")
    String rbkRSSLink;

    @Value("${news.riatass.rss}")
    String riaTassRSSLink;

    @Value("${news.vestiru.rss}")
    String vestiRuRSSLink;

    @Value("${news.vedomosti.rss}")
    String vedomostiRSSLink;

    @Value("${news.izvestiya.rss}")
    String izvestiyaRSSLink;

    @Value("${news.lentaru.rss}")
    String lentaRuRSSLink;
    //endregion

    final HashMap<String, String> rbkCategories = new HashMap<String,
String>() {{
        put("rbcfreenews", "Без категории");
        put("economics", "Экономика");
        put("business", "Бизнес");
        put("opinions", "Мнения");
        put("finances", "Финансы");
        put("technology_and_media", "Технологии и медиа");
        put("politics", "Политика");
        put("society", "Общество");
        put("photoreport", "Фотоотчет");
        put("money", "Деньги");
        put("own_business", "Наш бизнес");
    }};

    @Autowired
    NewsSourceRepository newsSourceRepository;

    @Autowired
    NewsItemRepository newsItemRepository;

    @Autowired
    NewsCategoryRepository newsCategoryRepository;

    @Scheduled(fixedRate = 60000)
    @Async
    public void updateNews() throws IOException, FeedException {
        updateNewsSource(rbkRSSLink);
        updateNewsSource(riaTassRSSLink);
        updateNewsSource(vestiRuRSSLink);
        updateNewsSource(vedomostiRSSLink);
        updateNewsSource(izvestiyaRSSLink);
        updateNewsSource(lentaRuRSSLink);
    }

    @Async
    @Transactional(isolation = Isolation.SERIALIZABLE)
    void updateNewsSource(String rssLink) throws IOException, FeedException {

```

```

        SyndFeed feed = getSyndFeed(rssLink);
        boolean categoryFromHashMap = (rssLink.equals(rbkRSSLink) ||
rssLink.equals(izvestiyaRSSLink));

        String sourceName = feed.getDescription();
        NewsSource newsSource = findCreateNewsSource(feed, sourceName,
rssLink);
        final LocalDateTime[] lastUpdate = {null};

        List<SyndEntryImpl> entries = feed.getEntries();
        entries.forEach(entry -> {
            String uri = entry.getUri();
            NewsItem newsItem = newsItemRepository.findByUri(uri);

            if (newsItem == null) {
                createNewsItem(newsSource, entry, uri, categoryFromHashMap);
                lastUpdate[0] = LocalDateTime.now();
            }
        });

        if (lastUpdate[0] != null) {
            newsSource.setLastUpdate(lastUpdate[0]);
            newsSourceRepository.save(newsSource);
        }
    }

    private SyndFeed getSyndFeed(String rssLink) throws IOException,
FeedException {
        URL feedSource = new URL(rssLink);
        SyndFeedInput input = new SyndFeedInput();
        XmlReader reader = new XmlReader(feedSource);
        return input.build(reader);
    }

    private void createNewsItem(NewsSource newsSource, SyndEntryImpl entry,
String uri, boolean categoryFromHashMap) {
        String link = entry.getLink();

        NewsItem newsItem = new NewsItem();
        newsItem.setUri(uri);
        newsItem.setLink(link);
        newsItem.setSource(newsSource);
        newsItem.setTitle(entry.getTitle());

        setDescription(entry, newsItem);
        setAuthorPublicationDate(entry, newsItem);
        setPhotoUrl(entry, newsItem);

        if (categoryFromHashMap) {
            setNewsCategoriesFromHashMap(link, newsItem);
        } else {
            setNewsCategories(entry, newsItem);
        }

        newsItemRepository.save(newsItem);
    }

    private void setNewsCategories(SyndEntryImpl entry, NewsItem newsItem) {
        ArrayList<NewsCategory> newsCategories = new ArrayList<>();

        List<SyndCategoryImpl> categoryList = entry.getCategories();
        categoryList.forEach(category -> {
            String categoryName = category.getName();
            NewsCategory newsCategory = findCreateCategory(categoryName);

```

```

        newsCategories.add(newsCategory);
    });

    if (newsCategories.isEmpty()) {
        newsCategories.add(findCreateCategory("Без категории"));
    }

    newItem.setCategoryList(newsCategories);
}

private void setNewsCategoriesFromHashMap(String link, NewsItem newItem)
{
    ArrayList<NewsCategory> newsCategories = new ArrayList<>();

    rbkCategories.forEach((s, s2) -> {
        if (link.contains(s)) {
            newsCategories.add(findCreateCategory(s2));
        }
    });

    if (newsCategories.isEmpty()) {
        newsCategories.add(findCreateCategory("Без категории"));
    }

    newItem.setCategoryList(newsCategories);
}

private NewsCategory findCreateCategory(String categoryName) {
    NewsCategory newsCategory =
newsCategoryRepository.findByName(categoryName);
    if (newsCategory == null) {
        newsCategory = new NewsCategory();
        newsCategory.setName(categoryName);

        newsCategoryRepository.save(newsCategory);
    }
    return newsCategory;
}

private void setPhotoUrl(SyndEntryImpl entry, NewsItem newItem) {
    List<SyndEnclosureImpl> enclosureList = entry.getEnclosures();
    if (!enclosureList.isEmpty()) {
        newItem.setPhotoUrl(enclosureList.get(0).getUrl());
    }
}

private void setDescription(SyndEntryImpl entry, NewsItem newItem) {
    SyndContent description = entry.getDescription();
    if (description != null) {
        newItem.setDescription(description.getValue());
    }
}

private void setAuthorPublicationDate(SyndEntryImpl entry, NewsItem
newItem) {
    List<DCModuleImpl> dcModuleList = entry.getModules();
    if (!dcModuleList.isEmpty()) {
        DCModuleImpl firstDCModule = dcModuleList.get(0);

        newItem.setAuthor(firstDCModule.getCreator());
        newItem.setPublicationDate(firstDCModule.getDate());
    }
}

```

```

        private NewsSource findCreateNewsSource(SyndFeed feed, String sourceName,
String rssLink) {
            HashMap<String, String> sourceNames = new HashMap<String, String>()
            {{
                put(rbkRSSLink, "РБК");
                put(riaTassRSSLink, "ТАСС");
                put(vestiRuRSSLink, "Вести.Ру");
                put(vedomostiRSSLink, "Газета «Ведомости»");
                put(izvestiyaRSSLink, "Газета «Известия»");
                put(lentaRuRSSLink, "Lenta.ru");
            }};

            HashMap<String, String> photoUrls = new HashMap<String, String>() {{
                put(rbkRSSLink, "/assets/rbk_logo.jpg");
                put(riaTassRSSLink, "/assets/riaTass_logo.png");
                put(vestiRuRSSLink, "/assets/vestiRu_logo.jpg");
                put(vedomostiRSSLink, "/assets/vedomosti_logo.png");
                put(izvestiyaRSSLink, "/assets/izvestiya_logo.jpg");
                put(lentaRuRSSLink, "/assets/lentaRu_logo.png");
            }};

            return newsSourceRepository.findBySourceName(sourceName).orElseGet(()

-> {
                NewsSource newNewsSource = new NewsSource();
                newNewsSource.setSourceName(sourceName);
                newNewsSource.setName(sourceNames.get(rssLink));
                newNewsSource.setLastUpdate(LocalDate.now());
                newNewsSource.setLink(feed.getLink());

                SyndImage image = feed.getImage();
                if (image != null) {
                    newNewsSource.setLogoImageUrl(image.getUrl());
                }
                newNewsSource.setLogoImageUrl(photoUrls.get(rssLink));

                return newsSourceRepository.save(newNewsSource);
            });
        }
    }
}

```

TwitterService

```

package application.service.news;

import application.data.model.twitter.TwitterHashtag;
import application.data.model.twitter.TwitterPeople;
import application.data.model.twitter.Tweet;
import application.data.repository.twitter.TwitterHashtagRepository;
import application.data.repository.twitter.TwitterPeopleRepository;
import application.data.repository.twitter.TweetRepository;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.social.twitter.api.Twitter;

```

```

import org.springframework.social.twitter.api.impl.TwitterTemplate;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)
@PropertySource("classpath:application.properties")
@Log4j2
@EnableScheduling
@EnableAsync
public class TwitterService {

    @Value("${spring.social.twitter.appId}")
    String twitterAppId;

    @Value("${spring.social.twitter.appSecret}")
    String twitterAppSecret;

    Twitter twitter;

    @Autowired
    ModelMapper mapper;
    @Autowired
    TweetRepository tweetRepository;

    @Autowired
    TwitterHashtagRepository twitterHashtagRepository;
    @Autowired
    TwitterPeopleRepository twitterPeopleRepository;

    @Scheduled(fixedRate = 300000)
    @Async
    public void updateTweets() {
        twitter = new TwitterTemplate(twitterAppId, twitterAppSecret);

        updateTweetsByHashtag(twitter);
        updateTweetsByPeople(twitter);
    }

    @Async
    void updateTweetsByHashtag(Twitter twitter) {
        Iterable<TwitterHashtag> followingHashtags =
twitterHashtagRepository.findAll();

        followingHashtags.forEach(followingHashtag -> {
            String hashtag = followingHashtag.getHashtag();
            String searchedWord = hashtag.contains("#") ? hashtag : "#" +
hashtag;

            List<org.springframework.social.twitter.api.Tweet> tweets =
twitter.searchOperations().search(searchedWord)
                .getTweets();

            tweets.forEach(tweet -> {
                Optional<Tweet> tweetInDBOptional =
tweetRepository.findById(tweet.getId());
                Tweet customTweet;
                if (tweetInDBOptional.isPresent()) {
                    customTweet = tweetInDBOptional.get();
                    customTweet.setFavoriteCount(tweet.getFavoriteCount());

```

```

        customTweet.setRetweetCount(tweet.getRetweetCount());
        customTweet.setText(tweet.getText());
    } else {
        customTweet = mapper.map(tweet, Tweet.class);
        TwitterHashtag twitterHashtag =
twitterHashtagRepository.findByHashtag(hashtag);
        if (twitterHashtag == null) {
            twitterHashtag = new TwitterHashtag();
            twitterHashtag.setHashtag(hashtag);
            twitterHashtagRepository.save(twitterHashtag);
        }
        customTweet.setHashtag(twitterHashtag);
    }

    tweetRepository.save(customTweet);
});

});

}

@Async
@Transactional(isolation = Isolation.SERIALIZABLE)
void updateTweetsByPeople(Twitter twitter) {
    Iterable<TwitterPeople> followingPeoples =
twitterPeopleRepository.findAll();

    followingPeoples.forEach(followingPeople -> {
        String nickname = followingPeople.getNickname();
        String searchedWord = nickname.contains("@") ?
nickname.replace("@", "") : nickname;

        List<org.springframework.social.twitter.api.Tweet> tweets =
twitter.searchOperations()
            .search("from:" + searchedWord).getTweets();

        tweets.forEach(tweet -> {
            Optional<Tweet> tweetInDBOptional =
tweetRepository.findById(tweet.getId());
            Tweet customTweet;
            if (tweetInDBOptional.isPresent()) {
                customTweet = tweetInDBOptional.get();
                customTweet.setFavoriteCount(tweet.getFavoriteCount());
                customTweet.setRetweetCount(tweet.getRetweetCount());
                customTweet.setText(tweet.getText());
            } else {
                customTweet = mapper.map(tweet, Tweet.class);

                TwitterPeople twitterPeople =
twitterPeopleRepository.findByNickname(nickname);
                if (twitterPeople == null) {
                    twitterPeople = new TwitterPeople();
                    twitterPeople.setNickname(nickname);
                    twitterPeopleRepository.save(twitterPeople);
                }
                customTweet.setNickname(twitterPeople);
            }

            tweetRepository.save(customTweet);
        });
    });
}

}

```


NotificationService

```

package application.service.notification;

import application.data.model.service.NotificationServiceSettings;
import application.data.model.service.WebService;
import application.data.model.telegram.TelegramUser;
import
application.data.repository.service.NotificationServiceSettingsRepository;
import application.utils.handler.TelegramHandler;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

import static java.time.temporal.ChronoUnit.SECONDS;

@Service
@EnableScheduling
@EnableAsync
@FieldDefaults(level = AccessLevel.PRIVATE)
public class NotificationService {

    @Autowired
    TelegramHandler telegramHandler;
    @Autowired
    NotificationServiceSettingsRepository
notificationServiceSettingsRepository;

    // 5 минут
    final long updatePeriod = 300000;

    // 1 минута
    //    final long updatePeriod = 60000;
    //    final long updatePeriod = 5000;

    @Scheduled(fixedRate = updatePeriod)
    @Async
    public void checkNotification() {
        List<NotificationServiceSettings> notificationServiceSettingsList =
notificationServiceSettingsRepository.findAllByActiveIsTrueAndCountOfNotifica
tionPerDayGreaterThan(0);
        LocalDateTime currentDate = LocalDateTime.now();

        notificationServiceSettingsList.forEach(serviceSettings -> {
            LocalDateTime lastNotification =
serviceSettings.getLastNotification();

            long differenceBetweenNotifications =
SECONDS.between(lastNotification, currentDate);

            if (differenceBetweenNotifications >=
serviceSettings.getNotificationInterval()) {
                TelegramUser user = serviceSettings.getUser();
                Long chatId = Long.valueOf(user.getId());
            }
        });
    }
}

```

```

        WebService service = serviceSettings.getService();

        if (service == WebService.YandexWeather) {

telegramHandler.sendMessageForecastAboutFollowingCities(chatId, user,
false);

            } else if (service == WebService.NewsService) {
                telegramHandler.sendMessageLastNews(chatId, user,
true);
            } else if (service == WebService.TwitterService) {
                telegramHandler.sendMessageLastTweets(chatId, user,
true);
            }

            serviceSettings.setLastNotification(currentDate);
            notificationServiceSettingsRepository.save(serviceSettings);
        }
    });
}
}
}

```

YandexWeatherService

```

package application.service.weather;

import application.data.model.YandexWeather.*;
import application.data.repository.YandexWeather.*;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import lombok.*;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.apache.commons.io.IOUtils;
import org.apache.http.NameValuePair;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.*;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)
@PropertySource("classpath:yandex.properties")
@Log4j2
public class YandexWeatherService {

    @Autowired
    YandexWeatherRepository yandexWeatherRepository;
    @Autowired
    YandexWeatherFactRepository yandexWeatherFactRepository;
}

```

```

@Autowired
YandexWeatherForecastRepository yandexWeatherForecastRepository;
@Autowired
YandexWeatherHoursRepository yandexWeatherHoursRepository;
@Autowired
YandexWeatherInfoRepository yandexWeatherInfoRepository;
@Autowired
YandexWeatherPartsRepository yandexWeatherPartsRepository;
@Autowired
YandexWeatherTZInfoRepository yandexWeatherTZInfoRepository;

final HashMap<String, String> conditionMap = new HashMap<String,
String>() {
    {
        put("clear", "Ясно");
        put("partly-cloudy", "Малооблачно");
        put("cloudy", "Облачно с прояснениями");
        put("overcast", "Пасмурно");
        put("partly-cloudy-and-light-rain", "Небольшой дождь");
        put("cloudy-and-light-rain", "Небольшой дождь");
        put("overcast-and-light-rain", "Небольшой дождь");
        put("partly-cloudy-and-rain", "Дождь");
        put("cloudy-and-rain", "Дождь");
        put("overcast-and-rain", "Сильный дождь");
        put("overcast-thunderstorms-with-rain", "Сильный дождь, гроза");
        put("overcast-and-wet-snow", "Дождь со снегом");
        put("partly-cloudy-and-light-snow", "Небольшой снег");
        put("cloudy-and-light-snow", "Небольшой снег");
        put("overcast-and-light-snow", "Небольшой снег");
        put("partly-cloudy-and-snow", "Снег");
        put("cloudy-and-snow", "Снег");
        put("overcast-and-snow", "Снегопад");
    }
};
final HashMap<String, String> windDirectionMap = new HashMap<String,
String>() {{
    put("nw", "северо-западное");
    put("n", "северное");
    put("ne", "северо-восточное");
    put("e", "восточное");
    put("se", "юго-восточное");
    put("s", "южное");
    put("sw", "юго-западное");
    put("w", "западное");
    put("c", "штиль");
}};
final HashMap<Float, String> typePrecMap = new HashMap<Float, String>()
{{
    put(0F, "Без осадков");
    put(1F, "Дождь");
    put(2F, "Дождь со снегом");
    put(3F, "Снег");
}};
final HashMap<Float, String> strengthPrecMap = new HashMap<Float,
String>() {{
    put(0F, "Без осадков");
    put(0.25F, "Слабый дождь");
    put(0.5F, "Дождь");
    put(0.75F, "Сильный дождь");
    put(1F, "Сильный ливень");
}};
final HashMap<Float, String> cloudnessMap = new HashMap<Float, String>()
{{

```

```

        put(0F, "Ясно");
        put(0.25F, "Малооблачно");
        put(0.5F, "Облачно с прояснениями");
        put(0.75F, "Облачно с прояснениями");
        put(1F, "Пасмурно");
    });
    final HashMap<Float, String> moonCodeMap = new HashMap<Float, String>()
    {{
        put(0F, "Полнолуние");
        put(1F, "Убывающая Луна");
        put(2F, "Убывающая Луна");
        put(3F, "Убывающая Луна");
        put(4F, "Последняя четверть");
        put(5F, "Убывающая Луна");
        put(6F, "Убывающая Луна");
        put(7F, "Убывающая Луна");
        put(8F, "Новолуние");
        put(9F, "Растущая Луна");
        put(10F, "Растущая Луна");
        put(11F, "Растущая Луна");
        put(12F, "Первая четверть");
        put(13F, "Растущая Луна");
        put(14F, "Растущая Луна");
        put(15F, "Растущая Луна");
    });
    final HashMap<String, String> moonStatusMap = new HashMap<String,
String>() {{
        put("full-moon", "Полнолуние");
        put("decreasing-moon", "Убывающая Луна");
        put("last-quarter", "Последняя четверть");
        put("new-moon", "Новолуние");
        put("growing-moon", "Растущая Луна");
        put("first-quarter", "Первая четверть");
    });

    static String apiKey;
    static String defaultUrl;

    @Value("${yandex.weather.apiKey}")
    public void setApiKey(String value) {
        apiKey = value;
    }

    @Value("${yandex.weather.url}")
    public void setDefaultUrl(String value) {
        defaultUrl = value;
    }

    public YandexWeather getWeatherByCoordinates(String longitude, String
latitude) {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpGet request = createRequest(longitude, latitude);

        ObjectMapper mapper = new ObjectMapper();

        try (CloseableHttpResponse response = httpClient.execute(request)) {
            if (response.getStatusLine().getStatusCode() == 200) {
                String str = new
String(IOUtils.toByteArray(response.getEntity().getContent()),
StandardCharsets.UTF_8);

                // Подготовка всех данных
                YandexWeather yandexWeather = mapper.readValue(str,
YandexWeather.class);

```

```

        YandexWeatherInfo yandexWeatherInfo =
yandexWeather.getInfo();
        YandexWeatherTZInfo yandexWeatherTZInfo =
yandexWeatherInfo.getTzInfo();
        YandexWeatherFact yandexWeatherFact =
yandexWeather.getFact();
        List<YandexWeatherForecast> yandexWeatherForecasts =
yandexWeather.getForecasts();
        // Ибо parts надо парсить по-особенному
        if (yandexWeatherForecasts != null) {
            setPartsToForecasts(mapper, str, yandexWeatherForecasts);
        }

        return saveWeatherData(yandexWeather, yandexWeatherInfo,
yandexWeatherTZInfo, yandexWeatherFact,
            yandexWeatherForecasts);
    } else {
        log.error("Сервис не отвечает");
        return null;
    }

} catch (
    IOException e) {
    e.printStackTrace();
    return null;
}

}

@Transactional
YandexWeather saveWeatherData(YandexWeather yandexWeather,
YandexWeatherInfo yandexWeatherInfo,
    YandexWeatherTZInfo yandexWeatherTZInfo,
YandexWeatherFact yandexWeatherFact,
    List<YandexWeatherForecast>
yandexWeatherForecasts) {
    if (yandexWeatherTZInfo != null) {
        yandexWeatherTZInfoRepository.save(yandexWeatherTZInfo);
    }
    yandexWeatherInfoRepository.save(yandexWeatherInfo);
    yandexWeatherFactRepository.save(yandexWeatherFact);

    if (yandexWeatherForecasts != null) {
        yandexWeatherForecasts.forEach(yandexWeatherForecast -> {
            List<YandexWeatherParts> yandexWeatherParts =
yandexWeatherForecast.getParts();
            yandexWeatherParts.forEach(yandexWeatherPart -> {
                yandexWeatherPartsRepository.save(yandexWeatherPart);
            });

            List<YandexWeatherHours> yandexWeatherHours =
yandexWeatherForecast.getHours();
            yandexWeatherHours.forEach(yandexWeatherHour -> {
                yandexWeatherHoursRepository.save(yandexWeatherHour);
            });

            yandexWeatherForecastRepository.save(yandexWeatherForecast);
        });

    }

    return yandexWeatherRepository.save(yandexWeather);
}

private void setPartsToForecasts(ObjectMapper mapper, String str,

```

```

List<YandexWeatherForecast> yandexWeatherForecasts) {
    JsonParser parser = new JsonParser();
    JsonElement element = parser.parse(str);
    JsonObject yandexWeatherObject = element.getAsJsonObject();
    JSONArray forecasts =
yandexWeatherObject.get("forecasts").getAsJsonArray();

    for (int index = 0; index < yandexWeatherForecasts.size(); index++) {
        JsonObject forecastJsonObject =
forecasts.get(index).getAsJsonObject();
        JsonObject parts =
forecastJsonObject.get("parts").getAsJsonObject();

        List<YandexWeatherParts> yandexWeatherPartsList = new
ArrayList<>();

        forecastJsonObject.get("parts").getAsJsonObject().keySet().forEach(key -> {
            if (!key.contains("_short")) {
                String partJsonString =
parts.get(key).getAsJsonObject().toString();

                YandexWeatherParts yandexWeatherPart = null;
                try {
                    yandexWeatherPart = mapper.readValue(partJsonString,
YandexWeatherParts.class);
                    yandexWeatherPart.setName(key);
                    yandexWeatherPartsList.add(yandexWeatherPart);
                } catch (JsonProcessingException e) {
                    e.printStackTrace();
                }
            }
        });

        yandexWeatherForecasts.get(index).setParts(yandexWeatherPartsList);
    }

    private HttpGet createRequest(String longitude, String latitude) {
        StringBuilder requestUrl = new StringBuilder(defaultUrl);

        List<NameValuePair> urlParameters = new ArrayList<NameValuePair>() {{
            add(new BasicNameValuePair("lat", latitude));
            add(new BasicNameValuePair("lon", longitude));
//            add(new BasicNameValuePair("extra", "true"));
//            add(new BasicNameValuePair("limit", "1"));
        }};

        urlParameters.forEach(nameValuePair -> requestUrl
            .append(nameValuePair.getName())
            .append("=")
            .append(nameValuePair.getValue())
            .append("&"));

        HttpGet request = new HttpGet(requestUrl.toString());
        request.addHeader("X-Yandex-API-Key", apiKey);
        return request;
    }

    public String englishWeatherConditionToRussian(String condition) {
        return conditionMap.get(condition);
    }

    public String englishWindDirectionToRussian(String windDirection) {

```

```

        return windDirectionMap.get(windDirection);
    }

    public String floatTypePrecToRussian(float precType) {
        return typePrecMap.get(precType);
    }

    public String floatStrengthPrecToRussian(float precStrength) {
        return strengthPrecMap.get(precStrength);
    }

    public String floatCloudnessToRussian(float cloudness) {
        return cloudnessMap.get(cloudness);
    }

    public String floatMoonCodeToRussian(float moonCode) {
        return moonCodeMap.get(moonCode);
    }

    public String englishMoonStatusToRussian(String moonText) {
        return moonStatusMap.get(moonText);
    }
}

```

TelegramUpdateService

```

package application.service;

import application.data.model.telegram.*;
import application.data.repository.telegram.*;
import application.utils.mapper.*;
import lombok.AccessLevel;
import lombok.RequiredArgsConstructor;
import lombok.experimental.FieldDefaults;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.telegram.telegrambots.meta.api.objects.Chat;
import org.telegram.telegrambots.meta.api.objects.Location;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;

@Service
@Transactional
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
@RequiredArgsConstructor
public class TelegramUpdateService {

    TelegramChatRepository telegramChatRepository;
    TelegramMessageRepository telegramMessageRepository;
    TelegramUpdateRepository telegramUpdateRepository;
    TelegramUserRepository userRepository;
    TelegramContactRepository telegramContactRepository;
    TelegramLocationRepository telegramLocationRepository;

    TelegramUserMapper telegramUserMapper;
    TelegramChatMapper telegramChatMapper;
    TelegramContactMapper telegramContactMapper;
    TelegramLocationMapper telegramLocationMapper;
    TelegramMessageMapper telegramMessageMapper;
    TelegramUpdateMapper telegramUpdateMapper;

    // Турбо метод, записывающий все изменения, которые пришли по апдейту
    public TelegramUpdate save(Update update) {

```

```

    Message message = update.getMessage();
    boolean hasContact = message.hasContact();
    boolean hasLocation = message.hasLocation();

    // Находим персонажа или создаем его
    TelegramUser telegramUser = saveFindUser(message);

    // Находим или создаем чат
    TelegramChat telegramChat = saveFindChat(message, telegramUser);

    // Сохранение контакта
    TelegramContact telegramContact = hasContact ?
    saveFindContact(message, telegramUser) : null;

    // Сохранение локации
    TelegramLocation telegramLocation = hasLocation ?
    saveFindLocation(update, telegramUser) : null;

    // Запись истории сообщений
    TelegramMessage telegramMessage = saveTelegramMessage(message,
    telegramUser, telegramChat, telegramContact
    , telegramLocation);

    // Сохраняем все наши обновления
    return saveTelegramUpdate(update, telegramMessage);
}

private TelegramUser saveFindUser(Message message) {
    return userRepository.findById(message.getFrom().getId())
        .orElseGet(() -> {
            TelegramUser transformedUser =
    telegramUserMapper.toEntity(message.getFrom());
            transformedUser.setStatus(UserStatus.getInitialStatus());
            return userRepository.save(transformedUser);
        });
}

private TelegramChat saveFindChat(Message message, TelegramUser
telegramUser) {
    Chat chat = message.getChat();
    return telegramChatRepository.findById(chat.getId())
        .orElseGet(() -> {
            TelegramChat transformedChat =
    telegramChatMapper.toEntity(chat);
            transformedChat.setUser(telegramUser);

            // Пользователю сохраняем чат
            setUserChat(telegramUser, transformedChat);

            return telegramChatRepository.save(transformedChat);
        });
}

// private void setUserChat(TelegramUser telegramUser, TelegramChat
transformedChat) {
//     telegramUser.setChat(transformedChat);
//     userRepository.save(telegramUser);
// }

private TelegramContact saveFindContact(Message message, TelegramUser
telegramUser) {
    return telegramContactRepository.findById(telegramUser.getId())
        .orElseGet(() -> {
            TelegramContact transformedContact =

```



```

telegramContactMapper.toEntity(message.getContact());
    transformedContact.setUser(telegramUser);

    // Пользователю сохраняем номер телефона
    setUserPhone(telegramUser, transformedContact);

    return
telegramContactRepository.save(transformedContact);
    });
}

private void setUserPhone(TelegramUser telegramUser, TelegramContact
transformedContact) {
    telegramUser.setPhone(transformedContact.getPhoneNumber());
    userRepository.save(telegramUser);
}

private TelegramLocation saveFindLocation(Update update, TelegramUser
telegramUser) {
    Location location = update.getMessage().getLocation();
    float longitude = location.getLongitude();
    float latitude = location.getLatitude();

    return
telegramLocationRepository.findByLongitudeAndLatitude(longitude, latitude)
        .orElseGet(() -> {
            TelegramLocation transformedLocation =
telegramLocationMapper.toEntity(location);
            transformedLocation.setUser(telegramUser);

            telegramUser.setLocation(transformedLocation);
            userRepository.save(telegramUser);

            return
telegramLocationRepository.save(transformedLocation);
        });
}

private TelegramMessage saveTelegramMessage(Message message, TelegramUser
telegramUser, TelegramChat telegramChat,
                                           TelegramContact
telegramContact, TelegramLocation telegramLocation) {
    TelegramMessage telegramMessage =
telegramMessageMapper.toEntity(message);
    telegramMessage.setFrom(telegramUser);
    telegramMessage.setChat(telegramChat);
    telegramMessage.setContact(telegramContact);
    telegramMessage.setLocation(telegramLocation);
    return telegramMessageRepository.save(telegramMessage);
}

private TelegramUpdate saveTelegramUpdate(Update update, TelegramMessage
message) {
    TelegramUpdate telegramUpdate =
telegramUpdateMapper.toEntity(update);
    telegramUpdate.setMessage(message);
    return telegramUpdateRepository.save(telegramUpdate);
}
}

```

TelegramHandler

```
package application.utils.handler;

import application.data.model.YandexWeather.WeatherCity;
import application.data.model.YandexWeather.YandexWeather;
import application.data.model.YandexWeather.YandexWeatherFact;
import application.data.model.YandexWeather.YandexWeatherTZInfo;
import application.data.model.news.NewsCategory;
import application.data.model.news.NewsItem;
import application.data.model.service.*;
import application.data.model.telegram.*;
import application.data.model.twitter.Tweet;
import application.data.model.twitter.TwitterHashtag;
import application.data.model.twitter.TwitterPeople;
import application.data.repository.YandexWeather.WeatherCityRepository;
import application.data.repository.news.NewsItemRepository;
import application.data.repository.service.NewsSettingsRepository;
import
application.data.repository.service.NotificationServiceSettingsRepository;
import application.data.repository.service.TwitterSettingsRepository;
import application.data.repository.service.WeatherSettingsRepository;
import application.data.repository.telegram.TelegramChatRepository;
import application.data.repository.telegram.TelegramLocationRepository;
import application.data.repository.telegram.TelegramUserRepository;
import application.data.repository.twitter.TweetRepository;
import application.data.repository.twitter.TwitterHashtagRepository;
import application.data.repository.twitter.TwitterPeopleRepository;
import application.service.weather.YandexWeatherService;
import application.telegram.TelegramBot;
import application.telegram.TelegramKeyboards;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.text.SimpleDateFormat;
import java.time.ZoneId;
import java.util.*;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)
@Log4j2
@PropertySource("classpath:interface.properties")
@EnableAsync
public class TelegramHandler implements TelegramMessageHandler {

    @Autowired
    TelegramBot telegramBot;
    @Autowired
    public TelegramUserRepository userRepository;
    @Autowired
    public TelegramChatRepository telegramChatRepository;
    @Autowired
```

```

    public NotificationServiceSettingsRepository
notificationServiceSettingsRepository;
    @Autowired
    public TelegramKeyboards telegramKeyboards;
    @Autowired
    public WeatherSettingsRepository weatherSettingsRepository;
    @Autowired
    public YandexWeatherService yandexWeatherService;
    @Autowired
    public TelegramLocationRepository telegramLocationRepository;
    @Autowired
    public NewsSettingsRepository newsSettingsRepository;
    @Autowired
    public NewsItemRepository newsItemRepository;
    @Autowired
    public TwitterSettingsRepository twitterSettingsRepository;
    @Autowired
    public TwitterHashtagRepository twitterHashtagRepository;
    @Autowired
    public TwitterPeopleRepository twitterPeopleRepository;
    @Autowired
    public TweetRepository tweetRepository;
    @Autowired
    public WeatherCityRepository weatherCityRepository;

    //region Кнопки в приложении
    @Value("${telegram.START_COMMAND}")
    public String START_COMMAND;

    // Базовые кнопки
    @Value("${telegram.HELLO_BUTTON}")
    public String HELLO_BUTTON;
    @Value("${telegram.HELP_BUTTON}")
    public String HELP_BUTTON;

    // Кнопки регистрации
    @Value("${telegram.REGISTER_BUTTON}")
    public String REGISTER_BUTTON;
    @Value("${telegram.CANCEL_REGISTRATION_BUTTON}")
    public String CANCEL_REGISTRATION_BUTTON;
    @Value("${telegram.NEXT_BUTTON}")
    public String NEXT_BUTTON;
    @Value("${telegram.SHARE_PHONE_NUMBER}")
    public String SHARE_PHONE_NUMBER;
    @Value("${telegram.CONFIRM_EMAIL}")
    public String CONFIRM_EMAIL;

    // Кнопки настроек
    @Value("${telegram.SETTINGS_BUTTON}")
    public String SETTINGS_BUTTON;
    @Value("${telegram.NOTIFICATION_SETTINGS_BUTTON}")
    public String NOTIFICATION_SETTINGS_BUTTON;
    @Value("${telegram.BACK_BUTTON}")
    public String BACK_BUTTON;

    // Интеграция с погодой
    // @Value("${telegram.WEATHER_SETTINGS_BUTTON}")
    // public String WEATHER_SETTINGS_BUTTON;
    @Value("${telegram.ACTIVATE_WEATHER_BUTTON}")
    public String ACTIVATE_WEATHER_BUTTON;
    @Value("${telegram.DEACTIVATE_WEATHER_BUTTON}")
    public String DEACTIVATE_WEATHER_BUTTON;
    @Value("${telegram.SHARE_LOCATION_BUTTON}")
    public String SHARE_LOCATION_BUTTON;

```

```

@Value("${telegram.ADD_CITY_WEATHER_BUTTON}")
public String ADD_CITY_WEATHER_BUTTON;
@Value("${telegram.REMOVE_CITY_WEATHER_BUTTON}")
public String REMOVE_CITY_WEATHER_BUTTON;
@Value("${telegram.LIST_FOLLOWING_CITIES_BUTTON}")
public String LIST_FOLLOWING_CITIES_BUTTON;
@Value("${telegram.CANCEL_BUTTON}")
public String CANCEL_BUTTON;
@Value("${telegram.WEATHER_IN_CURRENT_LOCATION_BUTTON}")
public String WEATHER_IN_CURRENT_LOCATION_BUTTON;
// @Value("${telegram.SHOW_INFO_ABOUT_FOLLOWING_CITIES}")
// public String SHOW_INFO_ABOUT_FOLLOWING_CITIES;

//region Интервалы оповещений
@Value("${telegram.NOTIFICATION_15_MINUTES}")
public String NOTIFICATION_15_MINUTES;
@Value("${telegram.NOTIFICATION_30_MINUTES}")
public String NOTIFICATION_30_MINUTES;
@Value("${telegram.NOTIFICATION_1_HOUR}")
public String NOTIFICATION_1_HOUR;
@Value("${telegram.NOTIFICATION_2_HOURS}")
public String NOTIFICATION_2_HOURS;
@Value("${telegram.NOTIFICATION_3_HOURS}")
public String NOTIFICATION_3_HOURS;
@Value("${telegram.NOTIFICATION_6_HOURS}")
public String NOTIFICATION_6_HOURS;
@Value("${telegram.NOTIFICATION_9_HOURS}")
public String NOTIFICATION_9_HOURS;
@Value("${telegram.NOTIFICATION_12_HOURS}")
public String NOTIFICATION_12_HOURS;
@Value("${telegram.NOTIFICATION_24_HOURS}")
public String NOTIFICATION_24_HOURS;
//endregion

//region Интерпрация с новостями
// @Value("${telegram.SHOW_FOLLOWING_NEWS}")
// public String SHOW_FOLLOWING_NEWS;
// @Value("${telegram.SHOW_ALL_NEWS}")
// public String SHOW_ALL_NEWS;
@Value("${telegram.NEWS_SETTINGS_BUTTON}")
public String NEWS_SETTINGS_BUTTON;
@Value("${telegram.ACTIVATE_NEWS_BUTTON}")
public String ACTIVATE_NEWS_BUTTON;
@Value("${telegram.DEACTIVATE_NEWS_BUTTON}")
public String DEACTIVATE_NEWS_BUTTON;
@Value("${telegram.LIST_FOLLOWING_CATEGORIES_BUTTON}")
public String LIST_FOLLOWING_CATEGORIES_BUTTON;
@Value("${telegram.ADD_CATEGORY_NEWS_BUTTON}")
public String ADD_CATEGORY_NEWS_BUTTON;
@Value("${telegram.REMOVE_CATEGORY_NEWS_BUTTON}")
public String REMOVE_CATEGORY_NEWS_BUTTON;
@Value("${telegram.LIST_FOLLOWING_SOURCES_BUTTON}")
public String LIST_FOLLOWING_SOURCES_BUTTON;
@Value("${telegram.ADD_SOURCE_NEWS_BUTTON}")
public String ADD_SOURCE_NEWS_BUTTON;
@Value("${telegram.REMOVE_SOURCE_NEWS_BUTTON}")
public String REMOVE_SOURCE_NEWS_BUTTON;

@Value("${telegram.COMMON_ADD}")
public String COMMON_ADD;
@Value("${telegram.COMMON_DELETE}")
public String COMMON_DELETE;
@Value("${telegram.COMMON_BANNED_NEWS_SOURCES_ADD}")
public String COMMON_BANNED_NEWS_SOURCES_ADD;

```

```

@Value("${telegram.COMMON_BANNED_NEWS_SOURCES_DELETE}")
public String COMMON_BANNED_NEWS_SOURCES_DELETE;
//endregion

//region Интерпация с Twitter
@Value("${telegram.TWITTER_SETTINGS_BUTTON}")
public String TWITTER_SETTINGS_BUTTON;
@Value("${telegram.ACTIVATE_TWITTER_BUTTON}")
public String ACTIVATE_TWITTER_BUTTON;
@Value("${telegram.DEACTIVATE_TWITTER_BUTTON}")
public String DEACTIVATE_TWITTER_BUTTON;
@Value("${telegram.LIST_FOLLOWING_PEOPLES_BUTTON}")
public String LIST_FOLLOWING_PEOPLES_BUTTON;
@Value("${telegram.ADD_PEOPLES_BUTTON}")
public String ADD_PEOPLES_BUTTON;
@Value("${telegram.REMOVE_PEOPLES_BUTTON}")
public String REMOVE_PEOPLES_BUTTON;
@Value("${telegram.LIST_FOLLOWING_HASHTAGS_BUTTON}")
public String LIST_FOLLOWING_HASHTAGS_BUTTON;
@Value("${telegram.ADD_HASHTAG_BUTTON}")
public String ADD_HASHTAG_BUTTON;
@Value("${telegram.REMOVE_HASHTAG_BUTTON}")
public String REMOVE_HASHTAG_BUTTON;
// @Value("${telegram.SHOW_FOLLOWING_TWEETS}")
// public String SHOW_FOLLOWING_TWEETS;
// @Value("${telegram.SHOW_MOST_POPULAR_TWEETS}")
// public String SHOW_MOST_POPULAR_TWEETS;
//endregion

//region Новый интерфейс
@Value("${telegram.NEWS_BUTTON}")
public String NEWS_BUTTON;
@Value("${telegram.TWITTER_BUTTON}")
public String TWITTER_BUTTON;
@Value("${telegram.WEATHER_BUTTON}")
public String WEATHER_BUTTON;

@Value("${telegram.WATCH_BUTTON}")
public String WATCH_BUTTON;
@Value("${telegram.COMMON_SETTINGS_BUTTON}")
public String COMMON_SETTINGS_BUTTON;

@Value("${telegram.NEXT_ITEM_BUTTON}")
public String NEXT_ITEM_BUTTON;
@Value("${telegram.PREVIOUS_ITEM_BUTTON}")
public String PREVIOUS_ITEM_BUTTON;
@Value("${telegram.FIRST_ITEM_BUTTON}")
public String FIRST_ITEM_BUTTON;
@Value("${telegram.EXIT_WATCH_BUTTON}")
public String EXIT_WATCH_BUTTON;
@Value("${telegram.DEACTIVATE_PERSON_SETTINGS}")
public String DEACTIVATE_PERSON_SETTINGS;
@Value("${telegram.ACTIVATE_PERSON_SETTINGS}")
public String ACTIVATE_PERSON_SETTINGS;
//endregion
//endregion

@Override
public void handle(TelegramUpdate telegramUpdate, boolean hasText,
boolean hasContact, boolean hasLocation) {
}

//region Send messages
@Override

```

```

        public void sendMessageToUserByCustomMainKeyboard(Long chatId,
TelegramUser telegramUser, String text, UserStatus status) {
            ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getCustomReplyMainKeyboardMarkup(telegramUser);
            sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
        }

        @Override
        public void sendTextMessageReplyKeyboardMarkup(Long chatId, String text,
ReplyKeyboardMarkup replyKeyboardMarkup,
                                                    UserStatus status) {
            SendMessage sendMessage = makeSendMessage(chatId, text,
replyKeyboardMarkup);

            try {
                telegramBot.execute(sendMessage);

                if (status != null) {
                    TelegramUser user;
                    Optional<TelegramChat> chat =
telegramChatRepository.findById(chatId);

                    if (chat.isPresent()) {
                        user = chat.get().getUser();

                        user.setStatus(status);
                        userRepository.save(user);
                    } else {
                        log.error("Не найден чат с id: " + chatId);
                    }
                }
            } catch (TelegramApiException e) {
                log.error(e);
            }
        }

        @Override
        public void sendWeatherSettingsMessage(Long chatId, TelegramUser user,
String text, UserStatus status) {
            ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getWeatherSettingsKeyboard(user,
notificationServiceSettingsRepository);
            sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
        }

        @Override
        public void sendTextMessageForecastAboutFollowingCities(Long chatId,
TelegramUser telegramUser, boolean isUserLocation) {
            WeatherSettings weatherSettings =
weatherSettingsRepository.findById(telegramUser.getId());
            Set<WeatherCity> weatherCities = weatherSettings.getCities();

            List<String> textList = new ArrayList<>();
            for (int index = 0; index < weatherCities.size(); index++) {
                textList.add(getCityInfo(telegramUser, false, true));
            }
            textList.forEach(text -> sendTextMessageReplyKeyboardMarkup(chatId,
text, null, null));
        }

```

```

@Override
public void sendNewsSettingsMessage(Long chatId, TelegramUser user,
String text, UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getNewsSettingsKeyboard(user,
notificationServiceSettingsRepository);
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

@Override
public void saveServiceSettings(TelegramUser user, boolean active,
WebService webService) {
    NotificationServiceSettings notificationServiceSettings =
notificationServiceSettingsRepository.findByUserAndService(user, webService)
        .orElseGet(() -> {
            NotificationServiceSettings
newNotificationServiceSettings = new NotificationServiceSettings();
            newNotificationServiceSettings.setService(webService);
            newNotificationServiceSettings.setUser(user);
            return newNotificationServiceSettings;
        });

    notificationServiceSettings.setActive(active);

notificationServiceSettingsRepository.save(notificationServiceSettings);
}

@Override
public void sendTextMessageAddDeleteSomething(Long chatId, String text,
UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getAddDeleteSomethingKeyboardMarkup();
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

public NotificationServiceSettings saveFindServiceSettings(TelegramUser
user, WebService webService) {
    return
notificationServiceSettingsRepository.findByUserAndService(user, webService)
        .orElseGet(() -> {
            NotificationServiceSettings
newNotificationServiceSettings = new NotificationServiceSettings();
            newNotificationServiceSettings.setService(webService);
            newNotificationServiceSettings.setUser(user);

            return
notificationServiceSettingsRepository.save(newNotificationServiceSettings);
        });
}

@Override
public void sendTextMessageLastNews(Long chatId, TelegramUser
telegramUser, boolean isFollowingNews) {
    List<String> textList = new ArrayList<>();
    for (int index = 0; index < 3; index++) {
        textList.add(getNewsInfo(telegramUser, isFollowingNews));
    }

    textList.forEach(text -> sendTextMessageReplyKeyboardMarkup(chatId,
text, null, null));
}

```

```

@Override
public void sendTwitterSettingsMessage(Long chatId, TelegramUser user,
String text, UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getTwitterSettingsKeyboard(user,
notificationServiceSettingsRepository);
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

@Override
public void sendTextMessageLastTweets(Long chatId, TelegramUser
telegramUser, boolean isFollowingTweets) {
    List<String> textList = new ArrayList<>();
    for (int index = 0; index < 3; index++) {
        textList.add(getTweetsInfo(telegramUser, isFollowingTweets));
    }

    textList.forEach(text -> sendTextMessageReplyKeyboardMarkup(chatId,
text, null, null));
}

@Override
public void sendNewsTwitterMainPageKeyboard(Long chatId, TelegramUser
telegramUser, String text, UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getNewsTwitterMainPageKeyboard();
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

@Override
public void sendCommonSettingKeyboard(Long chatId, String text,
UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getCommonSettingsKeyboardMarkup();
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

@Override
public void sendCommonAddDeleteKeyboard(Long chatId, String text,
UserStatus status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getAddDeleteCommonKeyboard(status);
    sendTextMessageReplyKeyboardMarkup(chatId, text, replyKeyboardMarkup,
status);
}

@Override
public void sendNewsWatchKeyboard(Long chatId, TelegramUser user,
UserStatus status,
                                boolean isNextNews) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getNewsWatchKeyboard(user);
    sendTextMessageReplyKeyboardMarkup(chatId, getNewsInfo(user,
isNextNews), replyKeyboardMarkup, status);
}

@Override
public void sendTwitterWatchKeyboard(Long chatId, TelegramUser user,
UserStatus status,
                                boolean isNextNews) {
    ReplyKeyboardMarkup replyKeyboardMarkup =

```



```

telegramKeyboards.getNewsWatchKeyboard(user);
    sendTextMessageReplyKeyboardMarkup(chatId, getTweetsInfo(user,
isNextNews), replyKeyboardMarkup, status);
}

@Override
public void sendWeatherWatchKeyboard(Long chatId, TelegramUser user,
UserStatus status,
                                boolean isNextForecast, boolean
isUserLocation) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getNewsWatchKeyboard(user);
    sendTextMessageReplyKeyboardMarkup(chatId, getCityInfo(user,
isUserLocation, isNextForecast), replyKeyboardMarkup, status);
}

//endregion

//region Help methods
private SendMessage makeSendMessage(Long chatId, String text,
ReplyKeyboardMarkup replyKeyboardMarkup) {
    SendMessage sendMessage = new SendMessage();
    sendMessage.enableMarkdown(true);
    sendMessage.setChatId(chatId);
    sendMessage.setText(text);

    sendMessage.setReplyMarkup(replyKeyboardMarkup);
    return sendMessage;
}

private String getCityInfo(TelegramUser telegramUser, boolean
isUserLocation, boolean isNextForecast) {
    WeatherSettings weatherSettings =
weatherSettingsRepository.findByUserId(telegramUser.getId());
    StringBuilder result = new StringBuilder();

    if (weatherSettings == null) {
        isUserLocation = true;
    }

    if (isUserLocation) {
        TelegramLocation userLocation = telegramUser.getLocation();
        return
saveWeatherInfoAndDoMessageToUser(userLocation.getLongitude(),
                                userLocation.getLatitude(), telegramUser, true,
userLocation.getCity());
    }

    Set<WeatherCity> weatherCities = weatherSettings.getCities();
    if (weatherCities.isEmpty()) {
        return "Список отслеживаемых городов пуст.";
    }

    WeatherCity weatherCity;
    if (isNextForecast) {
        Date lastCreationDate =
weatherSettings.getLastCityCreationDate();
        if (lastCreationDate == null) {
            lastCreationDate = new Date();
        }

        weatherCity =
weatherCityRepository.findTop1ByCreationDateBeforeOrderByCreationDateDesc

```

```

(lastCreationDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime(
));
    } else {
        Object[] lastCities =
weatherSettings.getViewedCities().toArray();
        if (lastCities.length > 0) {
            weatherCity = (WeatherCity) lastCities[0];
        } else {
            return "Нет ранее просмотренных элементов";
        }
    }

    if (weatherCity == null) {
        return "Больше нет отслеживаемых городов";
    }

    float longitude = weatherCity.getLongitude();
    float latitude = weatherCity.getLatitude();
    String city = weatherCity.getName();

    result.append(saveWeatherInfoAndDoMessageToUser(longitude,
        latitude, telegramUser, false, city));

weatherSettings.setLastCityCreationDate(Date.from(weatherCity.getCreationDate
().atZone(ZoneId.systemDefault()).toInstant()));
    Set<WeatherCity> viewedCities = weatherSettings.getViewedCities();
    if (viewedCities == null) {
        viewedCities = new HashSet<>();
    }

    if (isNextForecast) {
        WeatherCity lastViewedWeatherCity =
weatherSettings.getLastViewedWeatherCity();
        if (lastViewedWeatherCity != null) {
            viewedCities.add(lastViewedWeatherCity);
        }
    } else {
        Object[] lastCities = viewedCities.toArray();
        if (lastCities.length > 1) {
            weatherSettings.setLastViewedWeatherCity((WeatherCity)
lastCities[1]);
        }
        viewedCities.clear();
        for (int index = 1; index <= lastCities.length - 1; index++) {
            viewedCities.add((WeatherCity) lastCities[index]);
        }
    }

    weatherSettings.setLastViewedWeatherCity(weatherCity);
    weatherCityRepository.save(weatherCity);
    weatherSettingsRepository.save(weatherSettings);

    return result.toString();
}

private String saveWeatherInfoAndDoMessageToUser(float longitude, float
latitude,
                                                    TelegramUser user,
boolean isUserLocation, String city) {
    String messageToUser;

    YandexWeather weather =
yandexWeatherService.getWeatherByCoordinates(Float.toString(longitude),

```

```

        Float.toString(latitude));

        if (weather == null) {
            messageToUser = "По вашему месторасположению не найдено
информации о погоде!";
        } else {
            if (isUserLocation) {
                saveUserTZ(user, weather);
            }

            YandexWeatherFact fact = weather.getFact();

            messageToUser = "Погода в городе: " + "*" + city + "*" +
"\r\n\r\n" +
                "Сейчас " +
yandexWeatherService.englishWeatherConditionToRussian(fact.getWeatherConditio
n()).toLowerCase()
                + "\r\n" +
                "Температура воздуха: " + fact.getTemp() + " °C, по
ощущениям: " + fact.getFeelsLike() + " °C" + "\r\n" +
                "Влажность: " + fact.getHumidity() + "%";
        }

        return messageToUser;
    }

    private String getNewsInfo(TelegramUser user, boolean isNextNews) {
        NewsSettings newsSettings =
newsSettingsRepository.findById(user.getId());
        StringBuilder result = new StringBuilder();
        List<NewsCategory> categoryList = new ArrayList<>();
        boolean isFollowingNews;

        if (newsSettings == null) {
            isFollowingNews = false;
        } else {
            categoryList = new ArrayList<>(newsSettings.getNewsCategories());
            isFollowingNews = newsSettings.isActiveUserSettings();
        }

        if (categoryList.isEmpty() && isFollowingNews) {
            result.append("Список отслеживаемых тем пуст");
        } else {
            if (getNewsAndMakeMessageFollowingNews(isNextNews, newsSettings,
result, categoryList, isFollowingNews)) {
                return "Нет ранее просмотренных элементов";
            }
        }
        return result.toString();
    }

    private boolean getNewsAndMakeMessageFollowingNews(boolean isNextNews,
NewsSettings newsSettings,
                                                                    StringBuilder result,
List<NewsCategory> categoryList,
                                                                    boolean
isFollowingNews) {
        NewsItem newsItem;
        if (isNextNews) {
            newsItem = getNextNewsItem(newsSettings, categoryList,
isFollowingNews);
        } else {
            Object[] lastItems = newsSettings.getViewedNews().toArray();

```

```

        if (lastItems.length > 0) {
            newItem = (NewItem) lastItems[0];
        } else {
            return true;
        }
    }

    makeNewsMessageAndCountViews(result, newItem, newsSettings,
isNextNews);
    return false;
}

private NewItem getNextNewItem(NewsSettings newsSettings,
List<NewsCategory> categoryList, boolean isFollowingNews) {
    NewItem newItem;
    Date lastPublicationDate = newsSettings.getLastNewsPublicationDate();
    if (lastPublicationDate == null) {
        lastPublicationDate = new Date();
    }

    if (isFollowingNews) {
        newItem =
newsItemRepository.findTop1ByCategoryListInAndPublicationDateBeforeOrderByPub
licationDateDescCreationDateDesc(
        categoryList, lastPublicationDate);
    } else {
        newItem =
newsItemRepository.findTop1ByIdIsNotNullAndPublicationDateBeforeOrderByPublic
ationDateDescCreationDateDesc(
        lastPublicationDate);
    }
    return newItem;
}

@Transactional
void makeNewsMessageAndCountViews(StringBuilder result, NewItem
newsItem, NewsSettings newsSettings, boolean isNextNews) {
    result.append(makeNewsMessage(newsItem));

    newsItem.setCountOfViewers(newsItem.getCountOfViewers() + 1);

newsSettings.setLastNewsPublicationDate(newsItem.getPublicationDate());
    Set<NewItem> viewedItems = newsSettings.getViewedNews();
    if (viewedItems == null) {
        viewedItems = new HashSet<>();
    }

    if (isNextNews) {
        NewItem lastViewedNewsItem =
newsSettings.getLastViewedNewsItem();
        if (lastViewedNewsItem != null) {
            viewedItems.add(lastViewedNewsItem);
        }
    } else {
        Object[] lastItems = viewedItems.toArray();
        if (lastItems.length > 1) {
            newsSettings.setLastViewedNewsItem((NewItem) lastItems[1]);
        }
        viewedItems.clear();
        for (int index = 1; index <= lastItems.length - 1; index++) {
            viewedItems.add((NewItem) lastItems[index]);
        }
        // He paбopaet

```

```

//          viewedItems.remove(itemToDelete);
    }

    newsSettings.setLastViewedNewsItem(newsItem);
    newsItemRepository.save(newsItem);
    newsSettingsRepository.save(newsSettings);
}

private String makeNewsMessage(NewsItem newsItem) {
    SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy
HH:mm:ss");
    String formattedDate = format.format(newsItem.getPublicationDate());
    String author = newsItem.getAuthor();
    List<NewsCategory> categoryList = newsItem.getCategoryList();
    StringBuilder categoriesSB = new StringBuilder();

    categoryList.forEach(category ->
categoriesSB.append(category.getName()).append(", "));
    String categories = categoriesSB.toString();
    if (!categoryList.isEmpty()) {
        categories = categories.substring(0, categories.length() - 2);
    }

    return "*" + formattedDate + ". " +
        newsItem.getSource().getName() + " / " + categories +
        "\r\n\r\n" +
        ((author == null) ? "*" : "Авtop: " + author + "*\r\n") +
        newsItem.getTitle() +
        "\r\n\r\n" +
        newsItem.getLink();
}

private String getTweetsInfo(TelegramUser user, boolean isNextTweets) {
    boolean isFollowingTweets;
    TwitterSettings twitterSettings =
twitterSettingsRepository.findByUserId(user.getId());
    StringBuilder result = new StringBuilder();

    List<TwitterHashtag> hashtags = new ArrayList<>();
    List<TwitterPeople> nicknames = new ArrayList<>();

    if (twitterSettings == null) {
        isFollowingTweets = false;
    } else {
        hashtags = new ArrayList<>(twitterSettings.getTwitterHashtags());
        nicknames = new ArrayList<>(twitterSettings.getTwitterPeople());

        if (hashtags.size() == 0 && nicknames.size() == 0) {
            return "Не заполнены настройки Twitter";
        }
        isFollowingTweets = twitterSettings.isActiveUserSettings();
    }

    Tweet tweet;
    if (isNextTweets) {
        tweet = getNextTweet(isFollowingTweets, twitterSettings,
hashtags, nicknames);
    } else {
        Object[] lastTweets =
twitterSettings.getViewedTweets().toArray();
        if (lastTweets.length > 0) {
            tweet = (Tweet) lastTweets[0];
        } else {
            return "Нет ранее просмотренных элементов";
        }
    }
}

```

```

        }
    }

    makeTwitterMessageAndUpdateViewHistory(isNextTweets, twitterSettings,
result, tweet);

    return result.toString();
}

private Tweet getNextTweet(boolean isFollowingTweets, TwitterSettings
twitterSettings, List<TwitterHashtag> hashtags, List<TwitterPeople>
nicknames) {
    Tweet tweet;
    Date lastCreatedAt = twitterSettings.getLastTweetCreationDate();
    if (lastCreatedAt == null) {
        lastCreatedAt = new Date();
    }

    if (isFollowingTweets) {
        tweet =
tweetRepository.findTop1ByCreatedAtBeforeAndNicknameInOrHashtagInOrderByCreat
edAtDesc(lastCreatedAt,
            nicknames, hashtags);
    } else {
        tweet =
tweetRepository.findTop1ByCreatedAtBeforeOrderByRetweetCountDescFavoriteCount
DescCreatedAtDesc(lastCreatedAt);
    }
    return tweet;
}

private void makeTwitterMessageAndUpdateViewHistory(boolean isNextTweets,
TwitterSettings twitterSettings, StringBuilder result, Tweet tweet) {
    result.append(makeTwitterMessage(tweet));
    twitterSettings.setLastTweetCreationDate(tweet.getCreatedAt());
    Set<Tweet> viewedTweets = twitterSettings.getViewedTweets();
    if (viewedTweets == null) {
        viewedTweets = new HashSet<>();
    }

    if (isNextTweets) {
        Tweet lastViewedTweet = twitterSettings.getLastViewedTweet();
        if (lastViewedTweet != null) {
            viewedTweets.add(lastViewedTweet);
        }
    } else {
        Object[] lastTweets = viewedTweets.toArray();
        if (lastTweets.length > 1) {
            twitterSettings.setLastViewedTweet((Tweet) lastTweets[1]);
        }
        viewedTweets.clear();
        for (int index = 1; index <= lastTweets.length - 1; index++) {
            viewedTweets.add((Tweet) lastTweets[index]);
        }
    }

    twitterSettings.setLastViewedTweet(tweet);
    tweetRepository.save(tweet);
    twitterSettingsRepository.save(twitterSettings);
}

private String makeTwitterMessage(Tweet tweet) {
    SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy
HH:mm:ss");

```

```

        String formattedDate = format.format(tweet.getCreatedAt());
        String author = "@" + tweet.getFromUser();
        TwitterHashtag twitterHashtag = tweet.getHashtag();
        String hashtag = (twitterHashtag == null) ? null : "#" +
twitterHashtag.getHashtag();

        return "*" + formattedDate + ". " + author + "*" + "\r\n\r\n"
            + tweet.getText() + "\r\n\r\n"
            + "Ретвитов: " + tweet.getRetweetCount().toString() + ". "
            + "Отметок «Нравится»: " +
tweet.getFavoriteCount().toString() + "\r\n"
            + ((hashtag == null) ? "" : hashtag);
    }

    @Transactional
    void saveUserTZ(TelegramUser user, YandexWeather weather) {
        YandexWeatherTZInfo tzInfo = weather.getInfo().getTzInfo();
        user.setTzInfo(tzInfo);

        TelegramLocation userLocation = user.getLocation();
        userLocation.setTzInfo(tzInfo);

        userRepository.save(user);
        telegramLocationRepository.save(userLocation);
    }
    //endregion
}

```

NewsTelegramHandler

```

package application.utils.handler;

import application.data.model.news.NewsCategory;
import application.data.model.news.NewsSource;
import application.data.model.service.NewsSettings;
import application.data.model.service.NotificationServiceSettings;
import application.data.model.service.WebService;
import application.data.model.telegram.TelegramMessage;
import application.data.model.telegram.TelegramUpdate;
import application.data.model.telegram.TelegramUser;
import application.data.model.telegram.UserStatus;
import application.data.repository.news.NewsCategoryRepository;
import application.data.repository.news.NewsSourceRepository;
import lombok.AccessLevel;
import lombok.Data;
import lombok.experimental.FieldDefaults;
import org.glassfish.grizzly.utils.ArraySet;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.repository.query.Param;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.telegram.telegrambots.meta.api.objects.User;

import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.concurrent.atomic.AtomicInteger;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)

```

```

@EnableAsync
public class NewsTelegramHandler extends TelegramHandler {

    @Autowired
    NewsCategoryRepository newsCategoryRepository;

    @Autowired
    NewsSourceRepository newsSourceRepository;

    final WebService webService = WebService.NewsService;

    @Override
    @Async
    public void handle(TelegramUpdate telegramUpdate, boolean hasText,
        boolean hasContact, boolean hasLocation) {
        if (!hasText) {
            return;
        }

        TelegramMessage telegramMessage = telegramUpdate.getMessage();
        Long chatId = telegramMessage.getChat().getId();
        TelegramUser telegramUser = telegramMessage.getFrom();
        UserStatus status = telegramUser.getStatus();
        String userAnswer = telegramMessage.getText();
        boolean addingDeletingStatus = (status == UserStatus.AddCategory ||
            status == UserStatus.RemoveCategory ||
                status == UserStatus.AddSource || status ==
            UserStatus.RemoveSource);

        boolean isNewsStatus = (status == UserStatus.NewsMainPage || status
            == UserStatus.NewsWatch);

        if (userAnswer.equals(BACK_BUTTON)) {
            backButtonHandler(chatId, telegramUser, status);
        } else if (userAnswer.equals(NEWS_BUTTON)) {
            sendNewsTwitterMainPageKeyboard(chatId, telegramUser, "Раздел
«Новости»", UserStatus.NewsMainPage);
        } else if (userAnswer.equals(ACTIVATE_NEWS_BUTTON) & status ==
            UserStatus.NewsSettings) {
            saveServiceSettings(telegramUser, true, webService);
            sendNewsSettingsMessage(chatId, telegramUser, "Оповещения
включены", null);
        } else if (userAnswer.equals(DEACTIVATE_NEWS_BUTTON) & status ==
            UserStatus.NewsSettings) {
            saveServiceSettings(telegramUser, false, webService);
            sendNewsSettingsMessage(chatId, telegramUser, "Оповещения
выключены", null);
        } else if (userAnswer.equals(LIST_FOLLOWING_CATEGORIES_BUTTON)) {
            String messageToUser = listNewsCategoriesToUser(telegramUser);
            sendCommonAddDeleteKeyboard(chatId, messageToUser,
            UserStatus.CategoriesList);
        } else if (userAnswer.equals(LIST_FOLLOWING_SOURCES_BUTTON)) {
            String messageToUser = listNewsSourcesToUser(telegramUser);
            sendCommonAddDeleteKeyboard(chatId, messageToUser,
            UserStatus.SourcesList);
        } else if (status == UserStatus.CategoriesList) {
            categoriesListHandler(chatId, telegramUser, userAnswer);
        } else if (status == UserStatus.SourcesList) {
            sourcesListHandler(chatId, telegramUser, userAnswer);
        } else if (userAnswer.equals(CANCEL_BUTTON) && addingDeletingStatus)
        {
            String messageToUser = getMessageToUser(status);
            sendNewsSettingsMessage(chatId, telegramUser, messageToUser,
            UserStatus.NewsSettings);
        }
    }
}

```



```

        } else if (addingDeletingStatus) {
            sendAddRemoveMessageToUser(chatId, telegramUser, userAnswer,
status);
        } else if (((userAnswer.equals(WATCH_BUTTON) ||
userAnswer.equals(NEXT_ITEM_BUTTON)) & isNewsStatus)) {
            sendNewsWatchKeyboard(chatId, telegramUser, UserStatus.NewsWatch,
true);
        } else if (userAnswer.equals(PREVIOUS_ITEM_BUTTON) & isNewsStatus) {
            sendNewsWatchKeyboard(chatId, telegramUser, UserStatus.NewsWatch,
false);
        } else if (userAnswer.equals(EXIT_WATCH_BUTTON) & isNewsStatus) {
            sendNewsTwitterMainPageKeyboard(chatId, telegramUser, "Раздел
«Новости»", UserStatus.NewsMainPage);
        } else if (userAnswer.equals(FIRST_ITEM_BUTTON) & isNewsStatus) {
            clearUserNewsHistory(telegramUser, chatId);
        } else if (userAnswer.equals(ACTIVATE_PERSON_SETTINGS) ||
userAnswer.equals(DEACTIVATE_PERSON_SETTINGS) & isNewsStatus) {
            changeActivityForNews(telegramUser, chatId);
        }
    }

    private String getMessageToUser(UserStatus status) {
        String messageToUser = "";
        if (status == UserStatus.AddCategory) {
            messageToUser = "Добавление категории отменено";
        } else if (status == UserStatus.RemoveCategory) {
            messageToUser = "Удаление категории отменено";
        } else if (status == UserStatus.AddSource) {
            messageToUser = "Добавление источника отменено";
        } else if (status == UserStatus.RemoveSource) {
            messageToUser = "Удаление источника отменено";
        }
        return messageToUser;
    }

    @Transactional
    void changeActivityForNews(TelegramUser telegramUser, Long chatId) {
        NewsSettings newsSettings =
newsSettingsRepository.findByUserId(telegramUser.getId());
        if (newsSettings == null) {
            return;
        }

        newsSettings.setActiveUserSettings(!newsSettings.isActiveUserSettings());
        newsSettingsRepository.save(newsSettings);
        clearUserNewsHistory(telegramUser, chatId);
    }

    private void clearUserNewsHistory(TelegramUser telegramUser, Long chatId)
    {
        NewsSettings newsSettings =
newsSettingsRepository.findByUserId(telegramUser.getId());
        if (newsSettings == null) {
            return;
        }

        newsSettings.setLastViewedNewsItem(null);
        newsSettings.setLastNewsPublicationDate(new Date());
        newsSettings.setViewedNews(new HashSet<>());
        newsSettingsRepository.save(newsSettings);

        sendNewsWatchKeyboard(chatId, telegramUser, UserStatus.NewsWatch,
true);
    }

```

```

    }

    private void backButtonHandler(Long chatId, TelegramUser telegramUser,
    UserStatus status) {
        if (status == UserStatus.NewsMainPage) {
            sendMessageToUserByCustomMainKeyboard(chatId, telegramUser,
            "Главная страница", UserStatus.MainPage);
        } else if (status == UserStatus.NewsSettings) {
            sendCommonSettingKeyboard(chatId, "Настройки новостей",
            UserStatus.NewsCommonSettings);
        } else if (status == UserStatus.CategoriesList || status ==
            UserStatus.SourcesList) {
            sendNewsSettingsMessage(chatId, telegramUser, "Настройки рассылки
            новостей", UserStatus.NewsSettings);
        }
    }

    private void sendAddRemoveMessageToUser(Long chatId, TelegramUser user,
    String userAnswer, UserStatus status) {
        String messageToUser;
        if (status == UserStatus.AddCategory) {
            messageToUser = addNewsCategoriesToUser(user, userAnswer)
                ? "Категория *" + userAnswer + "*" добавлена в список
                отслеживаемых"
                : "Категория *" + userAnswer + "*" уже отслеживается
                вами";
        } else if (status == UserStatus.RemoveCategory) {
            messageToUser = removeNewsCategoryToUser(user, userAnswer)
                ? "Категория *" + userAnswer + "*" удалена из списка
                отслеживаемых"
                : "Категория *" + userAnswer + "*" не отслеживалась вами";
        } else if (status == UserStatus.AddSource) {
            messageToUser = addNewsSourceToUser(user, userAnswer)
                ? "Источник *" + userAnswer + "*" добавлена в список
                забаненных"
                : "Источник *" + userAnswer + "*" уже забанен вами";
        } else if (status == UserStatus.RemoveSource) {
            messageToUser = removeNewsSourceToUser(user, userAnswer)
                ? "Источник *" + userAnswer + "*" удалена из списка
                забаненных"
                : "Источник *" + userAnswer + "*" не забанен вами";
        } else {
            return;
        }

        UserStatus nextStatus = (status == UserStatus.AddCategory || status
        == UserStatus.RemoveCategory) ? UserStatus.CategoriesList
            : UserStatus.SourcesList;

        sendCommonAddDeleteKeyboard(chatId, messageToUser, nextStatus);
    }

    //region Category Handlers

    @Transactional
    boolean addNewsCategoriesToUser(TelegramUser user, String keyword) {
        NotificationServiceSettings notificationServiceSettings =
        saveFindServiceSettings(user, webService);

        NewsSettings newsSettings =
        newsSettingsRepository.findById(user.getId());
        boolean needToCreateNewRule = (newsSettings == null ||
        keyword.equals(""));
        if (needToCreateNewRule) {

```

```

        saveNewsCategory(user, notificationServiceSettings, keyword);
    } else {
        Set<NewsCategory> categories = newsSettings.getNewsCategories();
        NewsCategory category =
newsCategoryRepository.findByName(keyword);
        needToCreateNewRule = (!categories.contains(category));

        if (needToCreateNewRule) {
            categories.add(category);
            newsSettings.setNewsCategories(categories);
            newsSettingsRepository.save(newsSettings);
        }
    }

    return needToCreateNewRule;
}

private void saveNewsCategory(TelegramUser user,
NotificationServiceSettings
notificationServiceSettings, String keyword) {
    NewsSettings newsSettings = new NewsSettings();
    newsSettings.setUser(user);

newsSettings.setNotificationServiceSettings(notificationServiceSettings);

    NewsCategory newsCategory =
newsCategoryRepository.findByName(keyword);
    Set<NewsCategory> newsCategories = new HashSet<>();
    newsCategories.add(newsCategory);

    newsSettings.setNewsCategories(newsCategories);
    newsSettingsRepository.save(newsSettings);
}

@Transactional
boolean removeNewsCategoryToUser(TelegramUser user, String keyword) {
    NewsSettings newsSettings =
newsSettingsRepository.findById(user.getId());

    if (newsSettings == null) {
        return false;
    }

    Set<NewsCategory> newsCategories = newsSettings.getNewsCategories();
    NewsCategory category = newsCategoryRepository.findByName(keyword);

    boolean needToDelete = (newsCategories.contains(category));
    if (needToDelete) {
        newsCategories.remove(category);
        newsSettings.setNewsCategories(newsCategories);
        newsSettingsRepository.save(newsSettings);
    }

    return needToDelete;
}

private void categoriesListHandler(Long chatId, TelegramUser
telegramUser, String userAnswer) {
    if (userAnswer.equals(COMMON_ADD)) {
        String listOfCategories = listNewsCategoriesToUser(telegramUser);
        String messageToUser = listOfCategories + "\r\n\r\n" + "Введите
добавляемую категорию";
        sendTextMessageAddDeleteSomething(chatId, messageToUser,
UserStatus.AddCategory);
    }
}

```

```

    } else if (userAnswer.equals(COMMON_DELETE)) {
        String listOfCategories = listNewsCategoriesToUser(telegramUser);
        String messageToUser = listOfCategories + "\r\n\r\n" + "Введите
удаляемую категорию";
        sendTextMessageAddDeleteSomething(chatId, messageToUser,
UserStatus.RemoveCategory);
    }
}

private String listNewsCategoriesToUser(TelegramUser telegramUser) {
    NewsSettings newsSettings =
newsSettingsRepository.findById(telegramUser.getId());
    if (newsSettings == null) {
        addNewsCategoriesToUser(telegramUser, "");
        return "*Список отслеживаемых категорий пуст*";
    }

    Set<NewsCategory> newsCategories = newsSettings.getNewsCategories();
    String headerMessage = !newsCategories.isEmpty() ? ("*Список
отслеживаемых категорий:* " + "\r\n\r\n")
        : "*Список отслеживаемых категорий пуст*";
    StringBuilder stringBuilder = new StringBuilder(headerMessage);
    AtomicInteger count = new AtomicInteger();

    newsCategories.forEach(newsCategory -> {
        count.getAndIncrement();
        stringBuilder.append(count).append("
").append(newsCategory.getName()).append("\r\n");
    });

    return stringBuilder.toString();
}

//endregion

//region Sources Handlers

@Transactional
boolean addNewsSourceToUser(TelegramUser user, String keyword) {
    NotificationServiceSettings notificationServiceSettings =
saveFindServiceSettings(user, webService);

    NewsSettings newsSettings =
newsSettingsRepository.findById(user.getId());
    boolean needToCreateNewRule = (newsSettings == null ||
keyword.equals(""));
    if (needToCreateNewRule) {
        saveNewsSource(user, notificationServiceSettings, keyword);
    } else {
        Set<NewsSource> sources = newsSettings.getNewsSources();
        NewsSource source = newsSourceRepository.findByName(keyword);
        needToCreateNewRule = (!sources.contains(source));

        if (needToCreateNewRule) {
            sources.add(source);
            newsSettings.setNewsSources(sources);
            newsSettingsRepository.save(newsSettings);
        }
    }

    return needToCreateNewRule;
}

private void saveNewsSource(TelegramUser user,

```

```

NotificationServiceSettings
    notificationServiceSettings, String keyword) {
    NewsSettings newsSettings = new NewsSettings();
    newsSettings.setUser(user);

newsSettings.setNotificationServiceSettings(notificationServiceSettings);

    NewsSource newsSource = newsSourceRepository.findByName(keyword);
    Set<NewsSource> newsSources = new HashSet<>();
    newsSources.add(newsSource);

    newsSettings.setNewsSources(newsSources);
    newsSettingsRepository.save(newsSettings);
}

@Transactional
boolean removeNewsSourceToUser(TelegramUser user, String keyword) {
    NewsSettings newsSettings =
newsSettingsRepository.findByUserId(user.getId());

    if (newsSettings == null) {
        return false;
    }

    Set<NewsSource> newsSources = newsSettings.getNewsSources();
    NewsSource source = newsSourceRepository.findByName(keyword);

    boolean needToDelete = (newsSources.contains(source));
    if (needToDelete) {
        newsSources.remove(source);
        newsSettings.setNewsSources(newsSources);
        newsSettingsRepository.save(newsSettings);
    }

    return needToDelete;
}

private void sourcesListHandler(Long chatId, TelegramUser telegramUser,
String userAnswer) {
    if (userAnswer.equals(COMMON_BANNED_NEWS_SOURCES_ADD)) {
        String listOfSources = listNewsSourcesToUser(telegramUser);
        String messageToUser = listOfSources + "\r\n\r\n" + "Введите
запрещаемый источник";
        sendTextMessageAddDeleteSomething(chatId, messageToUser,
UserStatus.AddSource);
    } else if (userAnswer.equals(COMMON_BANNED_NEWS_SOURCES_DELETE)) {
        String listOfSources = listNewsSourcesToUser(telegramUser);
        String messageToUser = listOfSources + "\r\n\r\n" + "Введите
разрешаемый источник";
        sendTextMessageAddDeleteSomething(chatId, messageToUser,
UserStatus.RemoveSource);
    }
}

private String listNewsSourcesToUser(TelegramUser telegramUser) {
    NewsSettings newsSettings =
newsSettingsRepository.findByUserId(telegramUser.getId());
    if (newsSettings == null) {
        addNewsSourceToUser(telegramUser, "");
        return "*Список забаненных источников пуст*";
    }

    Set<NewsSource> newsSources = newsSettings.getNewsSources();
    String headerMessage = !newsSources.isEmpty() ? ("*Список забаненных

```

```

источников:* " + "\r\n\r\n")
        : "*Список забаненных источников пуст*";
    StringBuilder stringBuilder = new StringBuilder(headerMessage);
    AtomicInteger count = new AtomicInteger();

    newsSources.forEach(newsSource -> {
        count.getAndIncrement();
        stringBuilder.append(count).append(".
    ").append(newsSource.getName()).append("\r\n");
    });

    Iterable<NewsSource> allNewsSources = newsSourceRepository.findAll();
    stringBuilder.append("\r\n\r\n*Список всех источников: *\r\n");
    count.set(0);
    allNewsSources.forEach(newsSource -> {
        count.getAndIncrement();
        stringBuilder.append(count).append(".
    ").append(newsSource.getName()).append("\r\n");
    });

    return stringBuilder.toString();
}

//endregion
}

```

SettingsTelegramHandler

```

package application.utils.handler;

import application.data.model.service.NotificationServiceSettings;
import application.data.model.service.WebService;
import application.data.model.telegram.TelegramMessage;
import application.data.model.telegram.TelegramUpdate;
import application.data.model.telegram.TelegramUser;
import application.data.model.telegram.UserStatus;
import lombok.AccessLevel;
import lombok.experimental.FieldDefaults;
import lombok.extern.log4j.Log4j2;
import org.springframework.scheduling.annotation.Async;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;

import java.util.HashMap;

@Component
@FieldDefaults(level = AccessLevel.PRIVATE)
@Log4j2
@EnableAsync
public class SettingsTelegramHandler extends TelegramHandler {

    @Override
    @Async
    public void handle(TelegramUpdate telegramUpdate, boolean hasText,
        boolean hasContact, boolean hasLocation) {
        TelegramMessage telegramMessage = telegramUpdate.getMessage();
        Long chatId = telegramMessage.getChat().getId();
        TelegramUser user = telegramMessage.getFrom();
        UserStatus status = user.getStatus();

        if (!hasText) {

```

```

        return;
    }

    String userAnswer = telegramMessage.getText();

    if (userAnswer.equals(SETTINGS_BUTTON)) {
        settingButtonHandler(chatId, status);
    } else if (userAnswer.equals(COMMON_SETTINGS_BUTTON)) {
        commonSettingHandler(chatId, user, status);
    } else if (userAnswer.equals(NOTIFICATION_SETTINGS_BUTTON)) {
        HashMap<UserStatus, UserStatus> userStatusToNext = new
        HashMap<UserStatus, UserStatus>() {{
            put(UserStatus.NewsCommonSettings,
                UserStatus.NewsNotificationSettings);
            put(UserStatus.TwitterCommonSettings,
                UserStatus.TwitterNotificationSettings);
            put(UserStatus.WeatherCommonSettings,
                UserStatus.WeatherNotificationSettings);
        }};
        sendNotificationSettingsKeyboard(chatId, user,
            userStatusToNext.get(status));
    } else if (userAnswer.equals(BACK_BUTTON)) {
        backButtonHandler(chatId, user, status);
    } else if (status == UserStatus.NewsNotificationSettings ||
        status == UserStatus.TwitterNotificationSettings ||
        status == UserStatus.WeatherNotificationSettings) {

        HashMap<UserStatus, WebService> userStatusToWebService = new
        HashMap<UserStatus, WebService>() {
            {
                put(UserStatus.NewsNotificationSettings,
                    WebService.NewsService);
                put(UserStatus.TwitterNotificationSettings,
                    WebService.TwitterService);
                put(UserStatus.WeatherNotificationSettings,
                    WebService.YandexWeather);
            }
        };

        updateUserNotificationInterval(user, userAnswer,
            userStatusToWebService.get(status));
        backButtonHandler(chatId, user, status);
    }
}

private void settingButtonHandler(Long chatId, UserStatus status) {
    HashMap<UserStatus, String> userStatusTextMessage = new
    HashMap<UserStatus, String>() {{
        put(UserStatus.NewsMainPage, "Настройки новостей");
        put(UserStatus.TwitterMainPage, "Настройки Twitter");
        put(UserStatus.WeatherMainPage, "Настройки погоды");
    }};

    HashMap<UserStatus, UserStatus> userStatusToNext = new
    HashMap<UserStatus, UserStatus>() {{
        put(UserStatus.NewsMainPage, UserStatus.NewsCommonSettings);
        put(UserStatus.TwitterMainPage,
            UserStatus.TwitterCommonSettings);
        put(UserStatus.WeatherMainPage,
            UserStatus.WeatherCommonSettings);
    }};

    sendCommonSettingKeyboard(chatId, userStatusTextMessage.get(status),

```

```

userStatusToNext.get(status));
    }

    private void commonSettingHandler(Long chatId, TelegramUser user,
UserStatus status) {
        if (status == UserStatus.NewsCommonSettings) {
            sendNewsSettingsMessage(chatId, user, "Настройки рассылки
новостей", UserStatus.NewsSettings);
        } else if (status == UserStatus.TwitterCommonSettings) {
            sendTwitterSettingsMessage(chatId, user, "Настройки Twitter",
UserStatus.TwitterSettings);
        } else if (status == UserStatus.WeatherCommonSettings) {
            sendWeatherSettingsMessage(chatId, user, "Настройки рассылки
погоды", UserStatus.WeatherSettings);
        }
    }

    private void backButtonHandler(Long chatId, TelegramUser user, UserStatus
status) {
        if (status == UserStatus.NewsCommonSettings) {
            sendNewsTwitterMainPageKeyboard(chatId, user, "Раздел «Новости»",
UserStatus.NewsMainPage);
        } else if (status == UserStatus.TwitterCommonSettings) {
            sendNewsTwitterMainPageKeyboard(chatId, user, "Раздел «Twitter»",
UserStatus.TwitterMainPage);
        } else if (status == UserStatus.WeatherCommonSettings) {
            sendNewsTwitterMainPageKeyboard(chatId, user, "Раздел «Порода»",
UserStatus.WeatherMainPage);
        } else if (status == UserStatus.NewsNotificationSettings) {
            sendCommonSettingKeyboard(chatId, "Настройки новостей",
UserStatus.NewsCommonSettings);
        } else if (status == UserStatus.TwitterNotificationSettings) {
            sendCommonSettingKeyboard(chatId, "Настройки Twitter",
UserStatus.TwitterCommonSettings);
        } else if (status == UserStatus.WeatherNotificationSettings) {
            sendCommonSettingKeyboard(chatId, "Настройки погоды",
UserStatus.WeatherCommonSettings);
        }
    }

    @Transactional
    void updateUserNotificationInterval(TelegramUser user, String text,
WebService webService) {
        NotificationServiceSettings notificationServiceSettings =
saveFindServiceSettings(user, webService);
        float secondsInDay = 86400f;
        HashMap<String, Float> intervalDescription = new HashMap<String,
Float>() {
            {
                put(NOTIFICATION_15_MINUTES, 900F);
                put(NOTIFICATION_30_MINUTES, 1800F);
                put(NOTIFICATION_1_HOUR, 3600F);
                put(NOTIFICATION_2_HOURS, 7200F);
                put(NOTIFICATION_3_HOURS, 10800F);
                put(NOTIFICATION_6_HOURS, 21600F);
                put(NOTIFICATION_9_HOURS, 32400F);
                put(NOTIFICATION_12_HOURS, 43200F);
                put(NOTIFICATION_24_HOURS, 86400F);
            }
        };

        try {
            float notificationInterval = intervalDescription.get(text);

```



```

notificationServiceSettings.setNotificationInterval(notificationInterval);
notificationServiceSettings.setCountOfNotificationPerDay((int)
(secondsInDay / notificationInterval));

notificationServiceSettingsRepository.save(notificationServiceSettings);
} catch (Exception e) {
    log.info("Пользователь " + user + " выбрал уже текущую настройку
оповещений.");
}

private void sendNotificationSettingsKeyboard(Long chatId, TelegramUser
telegramUser, UserStatus
status) {
    ReplyKeyboardMarkup replyKeyboardMarkup =
telegramKeyboards.getNotificationSettingsKeyboardMarkup(telegramUser);
    sendTextMessageReplyKeyboardMarkup(chatId, "Настройки периода
оповещений", replyKeyboardMarkup, status);
}

}

```