**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**РАЗРАБОТКА TELEGRAM БОТА С ПАРСЕРОМ САЙТА ПОГОДЫ**

КУРСОВОЙ ПРОЕКТ

по дисциплине «Программирование»

Пояснительная записка

09.03.03 58.29.21 015 ПЗ

Руководитель ст. пр. Н.А. Архипов

Нормоконтpолер ст. пр. Н.А. Архипов

А.А. Фищев  
Д.В. Васяев

Студент гр. РИВ-120938у

**Екатеринбург 2023**

[ВВЕДЕНИЕ 4](#_Toc135338950)

[1 Описание технологии 6](#_Toc135338951)

[1.1. История развития 7](#_Toc135338952)

[1.2 HTML 12](#_Toc135338953)

[1.3 HTML 5 13](#_Toc135338954)

[1.3 XML 14](#_Toc135338955)

[1.4 XHTML 15](#_Toc135338956)

[2 Технологии, применяемые при написании парсеров 16](#_Toc135338957)

[2.1 Алгоритмы применяемые в парсерах 18](#_Toc135338958)

[2.2 Языки применяемы в парсерах (сравнение трех ЯП) 20](#_Toc135338959)

[2.3 Востребованность навыка написания парсеров на рынке труда 22](#_Toc135338960)

[3 Разработка парсера 24](#_Toc135338961)

[3.1 Постановка задачи 25](#_Toc135338962)

[3.2 Описание решения поставленной задачи 26](#_Toc135338963)

[3.3 Реализация парсера (описание кода реализованного парсера) 27](#_Toc135338964)

[3.4 Результат работы парсера 34](#_Toc135338965)

[ЗАКЛЮЧЕНИЕ 37](#_Toc135338966)

**ОПРЕДЕЛЕНИЯ**

Мессенджер - это программа для мгновенного обмена текстовыми сообщениями, аудиозаписями, фотографиями и другими мультимедиа. Приложения устанавливаются на компьютер, смартфон, планшет и работают через интернет.

Фреймворки - программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Spring Boot - Это отдельный модуль, который упрощает настройку фреймворка Spring и ускоряет запуск проектов. Он может автоматически сконфигурировать приложение и создать веб-сервер для его запуска. Большинство новых приложений Spring создаётся с помощью Spring Boot.

LomBok - Это библиотека Java, в которой предоставляется ряд аннотаций, направленных на исключение именно того кода Java, о котором известно, что он часто становится повторяющимся и/или шаблонным. Проект Lombok включается прямо в процесс сборки. Затем Lombok автоматически сгенерирует для Java байт-код, который вставляет в файлы .class, необходимые для реализации желаемого поведения, в зависимости от используемых вами аннотаций. Следовательно, каждая аннотация, предлагаемая в проекте Lombok, позволяет частично обойтись без написания методов и логики, без которых вы хотели бы обойтись. Речь о конструкторах, равенствах и функциях хеш-кода. Так вы сможете сэкономить массу времени и сосредоточиться на бизнес-логике вашего проекта. Кроме того, вы сможете держать базу кода сравнительно компактной, чистой, удобной для чтения и поддержки.

Fasterxml.Jackson - Jackson в основном известен как библиотека, которая конвертирует строки JSON и простые объекты Java (англ POJO — Plain Old Java Object).

ВВЕДЕНИЕ

Прежде всего это процесс сбора данных с последующей их обработкой и анализом. К этому способу прибегают, когда предстоит обработать большой массив информации, с которым сложно справиться вручную. Понятно, что программу, которая занимается парсингом, называют — парсер.

С помощью парсеров можно делать много полезных задач:

Цены. Актуальная задача для интернет-магазинов. Например, с помощью парсинга вы можете регулярно отслеживать цены конкурентов по тем товарам, которые продаются у вас. Или актуализировать цены на своем сайте в соответствии с ценами поставщика (если у него есть свой сайт).

Товарные позиции: названия, артикулы, описания, характеристики и фото. Например, если у вашего поставщика есть сайт с каталогом, но нет выгрузки для вашего магазина, вы можете спарсить все нужные позиции, а не добавлять их вручную. Это экономит время. Метаданные: SEO-специалисты могут парсить содержимое тегов title, description и другие метаданные.

Анализ сайта. Так можно быстро находить страницы с ошибкой 404, редиректы, неработающие ссылки и т. д. Есть еще серый парсинг. Сюда относится скачивание контента конкурентов или сайтов целиком. Или сбор контактных данных с агрегаторов и сервисов по типу Яндекс.Карт или 2Гис (для спам-рассылок и звонков). Но мы будем говорить только о белом парсинге, из-за которого у вас не будет проблем.

Десктопные и облачные парсеры

Облачные парсеры ‒ основное преимущество облачных парсеров — не нужно ничего скачивать и устанавливать на компьютер. Вся работа производится «в облаке», а вы только скачиваете результаты работы алгоритмов. У таких парсеров может быть веб-интерфейс и/или API (полезно, если вы хотите автоматизировать парсинг данных и делать его регулярно).

Большинство десктопных парсеров разработаны под Windows — на macOS их необходимо запускать с виртуальных машин. Также некоторые парсеры имеют портативные версии — можно запускать с флешки или внешнего накопителя.

1. Описание технологии

Что такое парсинг?

Глагол “to parse” в дословном переводе не означает ничего плохого. Делать грамматический разбор или структурировать — действия полезные и нужные. На языке всех, кто работает с данными на сайтах, это слово имеет свой оттенок.

Парсить — собирать и систематизировать информацию, размещенную на определенных сайтах, с помощью специальных программ, автоматизирующих процесс. Если вы когда-либо задавались вопросом, что такое парсер сайта, то вот он ответ. Это программные продукты, основной функцией которых является получение необходимых данных, соответствующих заданным параметрам.

* 1. История развития

В настоящее время все процессы, где применяется синтаксический анализ, используют парсеры — программы для проведение визуального или программно-автоматизированного синтаксического и лексического анализа или разбора какого-либо документа с целью извлечения из него необходимых данных. Это и различные автоматизированные переводчики с одного языка на другой, и трансляторы языков программирования, которые формируют программный код на машинно-ориентированный язык, это и язык SQL-запросов и тому подобные применения. В этой статье мы в хронологическом порядке перечислим важнейшие этапы развития парсеров как инструмента анализа данных.

1960

Выпущена спецификация ALGOL 60, в которой впервые описан язык с блочной структурой. Комитету ALGOL хорошо известно, что никто не знает, как парсить такой язык. Но они считают, что, если они детально описали язык с блочной структурой, то для него будет изобретен анализатор/парсер. Рискованный подход, который впоследствии окупился.

1961

Нед Айронс выпускает свой парсер ALGOL. Фактически, алгоритм Айронса является первым подобным анализатором, который был описан в печати. Парсер Неда является лево-рекурсивным (форма рекурсивного спуска) парсером . В отличие от современного рекурсивного спуска, алгоритм Айронса носит общий характер и является синтаксически-управляемым. «Общий» означает, что они могут разобрать что написано в БНФ (форма Бэкуса — Наура — формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории). «Синтаксически-управляемый» (декларативный) означает, что анализатор фактически создается из БНФ — парсер не нужно создавать самому.

1961

Почти одновременно, появляются самокодируемые (т.е. с возможностью внесения изменений в исходный код) подходы к реализации леворекурсивных алгоритмов. Сейчас их рассматривают как рекурсивный спуск. В течение следующих лет самокодируемые подходы становятся все более популярными для леворекурсивных анализаторов, чем синтаксически-управляемые алгоритмы. Важную роль сыграли следующие три фактора: в 1960-х память и CPU очень ограничены. Ручное кодирование (hand-coding) окупается даже тогда, когда выигрыш от его применения мал. Чистый леворекурсивный анализ — очень слабый метод синтаксического анализа. Ручному кодированию часто приходится преодолевать эти ограничения. Это утверждение верно как в 1961, так и в наши дни. Леворекурсивный анализ хорошо работает в сочетании с ручным кодированием — они дополняют друг друга.

1965

Дональд Кнут (Don Knuth) изобретает алгоритм LR. Ученый в первую очередь заинтересован в математической стороне задачи. Кнут описывает алгоритм анализа, но его подход считается непрактичным.

1968

Джей Эрли (Jay Earley) изобретает алгоритм, названный в его честь. Как и алгоритм Irons, алгоритм Эрли является синтаксически-управляемым и носит общий характер. В отличие от алгоритма Irons, не использует метод поиска с возвратом. Основная идея Эрли состоит в том, чтобы отслеживать этапы работы алгоритма в таблицах. Алгоритм Эрли заманчив, но имеет три основных недостатка:

Во-первых, есть ошибка в обработке правил нулевой длины.

Во-вторых, при право-сторонней рекурсии необходимо применять алгоритм дважды. В-третьих, чтобы создать таблицы, необходимо вести учёт системных ресурсов, что, по меркам аппаратных средств 1968 года, является довольно сложной задачей.

1969

Фрэнк ДеРемер (Frank DeRemer) описывает новый вариант LR Кнута. Для алгоритма LALR ДеРемера необходим только стек и таблица состояний, размер которой можно легко изменить.

1972

Ахо (Aho) и Алманн (Ullmann) описали простой способ исправить ошибку правила нулевой длины в оригинальном алгоритме Эрли. К сожалению, это исправление требует даже больше системных ресурсов, чем алгоритм Эрли.

1975

Белл Лэбс (Bell Labs) преобразует свой компилятор языка C из самокодируемого рекурсивного спуска в алгоритм LALR ДеРемера.

1977

Выходит, первая «Книга дракона». Этот, вскоре ставший классическим, учебник назван так из-за обложки, на которой изображен рыцарь, бросающий вызов дракону. На копье рыцаря выгравированы буквы «LALR». Впредь, говорить легкомысленно LALR — значит запятнать герб теории синтаксического анализа.

1979

Bell Labs выпустила новую версию UNIX — седьмую. V7 включает в себя безусловно наиболее полный, полезный и доступный инструментарий для разработки компиляторов. Центральное место занимает инструментарий Yacc, генератор синтаксических анализаторов на основе LALR. С небольшим трудом Yacc парсит свой собственный язык ввода, а также язык основного компилятора V7 — переносимого компилятора языка C. Кажется, что после двух десятилетий исследований, проблема синтаксического анализа наконец решается.

1987

Ларри Уолл (Larry Wall) представляет Perl 1. Perl позволяет решать более сложные задачи, чем отличается от уже существующих языков. Ларри активно использует LALR — насколько известно автору, чаще, чем кто-либо до или после него.

1991

Джуп Лео (Joop Leo) обнаруживает способ ускорить правосторонние рекурсии в алгоритме Эрли. Алгоритм Лео является линейным практически для любой: как однозначной, так и неоднозначной грамматики, представляющей практический интерес. Аппаратное обеспечение в 1991 на шесть порядков быстрее, чем в 1968 году, так что вопрос учёта системных ресурсов стал не столь важен. Однако, если дело касается скорости, то выигрывает алгоритм Эрли. Алгоритм Лео оказался важным открытием, но его практическая реализация появится лишь через 20 лет.

1990-е

Алгоритм Эрли забыт. Все приверженцы LALR довольны, не так ли? Нет. Пользователи LALR делают неприятные открытия. В то время как LALR автоматически генерирует свои анализаторы, их отладка настолько трудна, что проще написать парсер самому. После отладки их LALR анализаторы при правильных входных данных работают быстро. Но почти все, что они говорят пользователям о некорректных входных данных — лишь сообщение о неверном формате без указания дополнительной информации. По словам Ларри, LALR «быстр, но глуп».

2000

Ларри Уолл принимает решение о радикальном переопределении Perl — издании Perl 6. Он даже не рассматривает вопрос о повторном использовании LALR.

2002

Эйкок (Aycock) и Хоспул (Horspool) опубликовали результаты своих опытов по созданию быстрого, практичного парсера Эрли. Однако при этом не использовалось улучшение, предложенное Джупом Лео — они, кажется, не знают о нем. Их собственный метод ускорения ограничен в своих показателях и сложности, которые он несет с собой, они могут быть даже контрпродуктивными при оценке времени. Но в этой работе ценным оказалось решение ошибки правила нулевой длины. И на этот раз оно не требует никакого дополнительного учета системных ресурсов.

2006

GNU объявляет, что был переписан парсер компилятора GCC. В течение трех десятилетий, компиляторы языка C промышленного флагмана использовали LALR в качестве парсера — доказательство утверждения, что LALR и серьезный алгоритм эквивалентны. Теперь GNU заменяет LALR технологией, которую тот заменил четверть века назад: рекурсивным спуском.

С 2000 и до сегодняшнего дня

С отступлением от LALR приходит крах престижа теории синтаксического анализа. Спустя полтора века, мы пришли к тому, с чего начали. Если вы возьмете оригинальный алгоритм Ned Irons 1961 года, измените имена и даты и переведете код из смеси ассемблера и ALGOL в Haskell, то вы бы легко смогли выпустить его в наши дни и представить как новый и революционный подход.

1.2 HTML

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора.

Элементы HTML являются строительными блоками HTML страниц. С помощью HTML разные конструкции, изображения и другие объекты, такие как интерактивная веб-форма, могут быть встроены в отображаемую страницу. HTML предоставляет средства для создания заголовков, абзацев, списков, ссылок, цитат и других элементов. Элементы HTML выделяются тегами, записанными с использованием угловых скобок. Такие теги, как <img /> и <input />, напрямую вводят контент на страницу. Другие теги, такие как <p>, окружают и оформляют текст внутри себя и могут включать другие теги в качестве подэлементов.

Браузеры не отображают HTML-теги, но используют их для интерпретации содержимого страницы. Язык гипертекстовой разметки HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986—1991 годах в стенах ЦЕРНа в Женеве в Швейцарии[3]. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности SGML путём определения небольшого набора структурных и семантических элементов — дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже.

1.3 HTML 5

HTML5 (англ. HyperText Markup Language, version 5) — язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта (англ. HTML Living Standard). Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

В HTML5 реализовано множество новых синтаксических особенностей. Например, элементы <video>, <audio> и <canvas>, а также возможность использования SVG и математических формул. Эти новшества разработаны для упрощения создания и управления графическими и мультимедийными объектами в сети без необходимости использования сторонних API и плагинов. Другие новые элементы, такие как <section>, <article>, <header> и <nav>, разработаны для того, чтобы обогащать семантическое содержимое документа (страницы). Новые атрибуты были введены с той же целью, хотя ряд элементов и атрибутов был удалён. Некоторые элементы, например <a>, <menu> и <cite>, были изменены, переопределены или стандартизированы. API и DOM стали основными частями спецификации HTML5. HTML5 также определяет некоторые особенности обработки ошибок вёрстки, поэтому синтаксические ошибки должны рассматриваться одинаково всеми совместимыми браузерами.

HTML и HTML5: в чем разница

Одно из главных различий — размещение мультимедиа (аудио и видео на страницах). Раньше это было возможно только с помощью большого HTML-кода и дополнительных программ. В HTML5 для этого нужен только один тег.

1.3 XML

XML - это язык разметки подобный HTML. Расшифровывается как (англ. Extensible Markup Language - Расширяемый Язык Разметки) и является рекомендацией сообщества W3C в качестве языка разметки общего назначения (W3C recommended). В отличии от остальных языков разметки, XML сам по себе не определён (это означает, что вы должны сами определять используемые теги). Основной целью XML является передача данных между разными системами (даже концептуально разными), такими как интернет.

Все составляющие части документа обобщаются в пролог и корневой элемент. Корневой элемент — обязательная часть документа, составляющая всю его суть (пролог, вообще говоря, может отсутствовать). Корневой элемент может включать (а может не включать) вложенные в него элементы, символьные данные и комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее. Пролог может включать объявления, инструкции обработки, комментарии. Его следует начинать с объявления XML, хотя в определённой ситуации допускается отсутствие этого объявления.

Элементы документа должны быть правильно вложены: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался.

Символьные данные могут встречаться внутри элементов как непосредственно так и в специальных секциях «CDATA». Объявления, инструкции обработки и элементы могут иметь связанные с ними атрибуты. Атрибуты используются для связывания с логической единицей текста пар имя-значение.

1.4 XHTML

Все составляющие части документа обобщаются в пролог и корневой элемент. Корневой элемент — обязательная часть документа, составляющая всю его суть (пролог, вообще говоря, может отсутствовать). Корневой элемент может включать (а может не включать) вложенные в него элементы, символьные данные и комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее. Пролог может включать объявления, инструкции обработки, комментарии. Его следует начинать с объявления XML, хотя в определённой ситуации допускается отсутствие этого объявления.

Элементы документа должны быть правильно вложены: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался. Символьные данные могут встречаться внутри элементов как непосредственно так и в специальных секциях «CDATA». Объявления, инструкции обработки и элементы могут иметь связанные с ними атрибуты. Атрибуты используются для связывания с логической единицей текста пар имя-значение.

2 Технологии, применяемые при написании парсеров

-Import.io предлагает разработчику легко формировать собственные пакеты данных: нужно только импортировать информацию с определенной веб-страницы и экспортировать ее в CSV. Можно извлекать тысячи веб-страниц за считанные минуты, не написав ни строчки кода, и создавать тысячи API согласно вашим требованиям.  
 -Webhose.io обеспечивает прямой доступ в реальном времени к структурированным данным, полученным в результате парсинга тысяч онлайн источников. Этот парсер способен собирать веб-данные на более чем 240 языках и сохранять результаты в различных форматах, включая XML, JSON и RSS.

-CloudScrape способен парсить информацию с любого веб-сайта и не требует загрузки дополнительных приложений, как и Webhose. Редактор самостоятельно устанавливает своих поисковых роботов и извлекает данные в режиме реального времени. Пользователь может сохранить собранные данные в облаке, например, Google Drive и Box.net, или экспортировать данные в форматах CSV или JSON.

-Scrapinghub – это облачный инструмент парсинга данных, который помогает выбирать и собирать необходимые данные для любых целей. Scrapinghub использует Crawlera, умный прокси-ротатор, оснащенный механизмами, способными обходить защиты от ботов. Сервис способен справляться с огромными по объему информации и защищенными от роботов сайтами.

-[ParseHub](https://www.parsehub.com/)может парсить один или много сайтов с поддержкой JavaScript, AJAX, сеансов, cookie и редиректов. Приложение использует технологию самообучения и способно распознать самые сложные документы в сети, затем генерирует выходной файл в том формате, который нужен пользователю.

VisualScraper – это еще одно ПО для парсинга больших объемов информации из сети. VisualScraper извлекает данные с нескольких веб-страниц и синтезирует результаты в режиме реального времени. Кроме того, данные можно экспортировать в форматы CSV, XML, JSON и SQL.

-[Spinn3r](http://docs.spinn3r.com/)позволяет парсить данные из блогов, новостных лент, новостных каналов RSS и Atom, социальных сетей. Spinn3r имеет «обновляемый» API, который делает 95 процентов работы по индексации. Это предполагает усовершенствованную защиту от спама и повышенный уровень безопасности данных.  
 -[80legs](http://80legs.com/)– это мощный и гибкий веб-инструмент парсинга сайтов, который можно очень точно подстроить под потребности пользователя. Сервис справляется с поразительно огромными объемами данных и имеет функцию немедленного извлечения. Клиентами 80legs являются такие гиганты как MailChimp и PayPal.

-[Scraper](http://scraper.com/)– это расширение для Chrome с ограниченными функциями парсинга данных, но оно полезно для онлайновых исследований и экспортирования данных в Google Spreadsheets. Этот инструмент предназначен и для новичков, и для экспертов, которые могут легко скопировать данные в буфер обмена или хранилище в виде электронных таблиц, используя OAuth.

-[OutWit Hub](https://www.outwit.com/products/hub/) – это дополнение Firefox с десятками функций извлечения данных. Этот инструмент может автоматически просматривать страницы и хранить извлеченную информацию в подходящем для пользователя формате. OutWit Hub предлагает простой интерфейс для извлечения малых или больших объемов данных по необходимости. OutWit позволяет «вытягивать» любые веб-страницы прямо из браузера и даже создавать в панели настроек автоматические агенты для извлечения данных и сохранения их в нужном формате. Это один из самых простых бесплатных веб-инструментов по сбору данных, не требующих специальных знаний в написании кодов.

2.1 Алгоритмы применяемые в парсерах

-AST(Abstract Syntax Tree)/АСД(Абстрактное синтаксическое дерево) (выходная структура данных) — Структура объектов, представляющая иерархию нетерминальных сущностей грамматики разбираемого потока и составляющих их терминалов. Например, алгебраический поток (1 + 2) + 3 можно представить в виде ВЫРАЖЕНИЕ(ВЫРАЖЕНИЕ(ЧИСЛО(1) ОПЕРАТОР(+) ЧИСЛО(2)) ОПЕРАТОР(+) ЧИСЛО(3)). Как правило, потом это дерево как-то обрабатывается клиентом парсера для получения результатов (например, подсчета ответа данного выражения)

-CFG(Context-free grammar)/КСГ(Контекстно-свободная грамматика) — вид наиболее распространенной грамматики, используемый для описания входящего потока символов для парсера (не только для этого, разумеется). Характеризуется тем, что использование её правил не зависит от контекста (что не исключает того, что она в некотором роде задает себе контекст сама, например правило для вызова функции не будет иметь значения, если находится внутри фрагмента потока, описываемого правилом комментария). Состоит из правил продукции, заданных для терминальных и не терминальных символов.

-BNF(Backus-Naur Form, Backus normal form)/БНФ(Бэкуса-Наура форма) — форма, в которой одни синтаксические категории последовательно определяются через другие. Форма представления КСГ, часто используемая непосредственно для задания входа парсеру. Характеризуется тем, что определяемым является всегда ОДИН нетерминальный символ. Классической является форма записи вида:

<определяемый символ> ::= <посл.1> | <посл.2> | . . . | <посл.n>

Так же существует ряд разновидностей, таких как ABNF(AugmentedBNF), EBNF(ExtendedBNF) и др. В общем, эти формы несколько расширяют синтаксис обычной записи BNF.

-LL(k), LR(k), SLR,... — виды алгоритмов парсеров. В этой статье мы не будем подробно на них останавливаться, если кого-то заинтересовало, внизу я дам несколько ссылок на материал, из которого можно о них узнать. Однако остановимся подробнее на другом аспекте, на грамматиках парсеров. Грамматика LL/LR групп парсеров является BNF, это верно. Но верно также, что не всякая грамматика BNF является также LL(k) или LR(k). Да и вообще, что значит буква k в записи LL/LR(k)? Она означает, что для разбора грамматики требуется заглянуть вперед максимум на k терминальных символов по потоку. То есть для разбора (0) грамматики требуется знать только текущий символ. Для (1) — требуется знать текущий и 1 следующий символ. Для (2) — текущий и 2 следующих и т.д. Немного подробнее о выборе/составлении BNF для конкретного парсера поговорим ниже.

-PEG(Parsing expression grammar)/РВ-грамматика — грамматика, созданная для распознавания строк в языке.

2.2 Языки применяемы в парсерах (сравнение трех ЯП)

Мы сравнили следующие языки программирования: Golang (используя платформу Diggernaut), Perl, PHP5, Python 2.7, Python + Scrapy, Ruby.

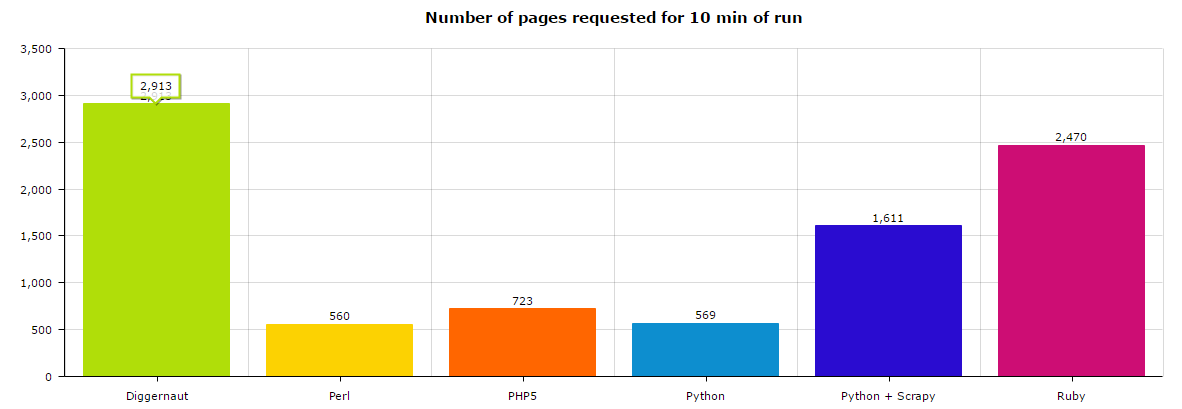


Рисунок 1 – количество страниц, запрошенных за 10 минут работы.

Мы видим, что у нас есть три лидера: Golang забрал почти 3 тысячи страниц, Ruby — примерно 2.5 тысячи, а Python + Scrapy — около 1.5 тысяч. Другие языки на их фоне выглядят действительно медленными.

Однако, если мы посмотрим на график использования CPU, мы увидим немного другую картину:

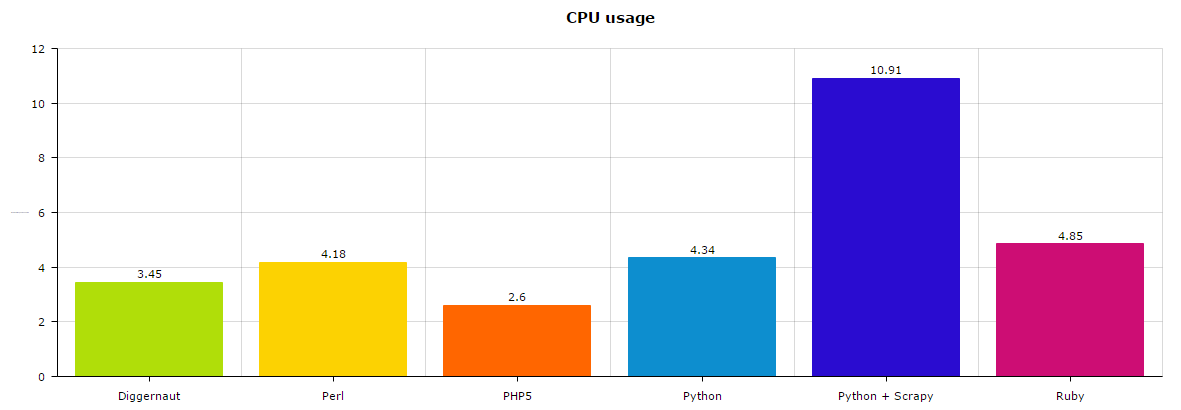


Рисунок 2 – использование ЦП.

Первое место здесь уходит к PHP5, использующему только лишь 2.5% CPU, за ним идет Golang с 3.5% и третьим замыкает гонку Perl приблизительно с 4%. Другие языки отстают совсем не намного, за исключением Python + Scrapy — 11% это слегка много

И последний испытываемый параметр — использование RAM:

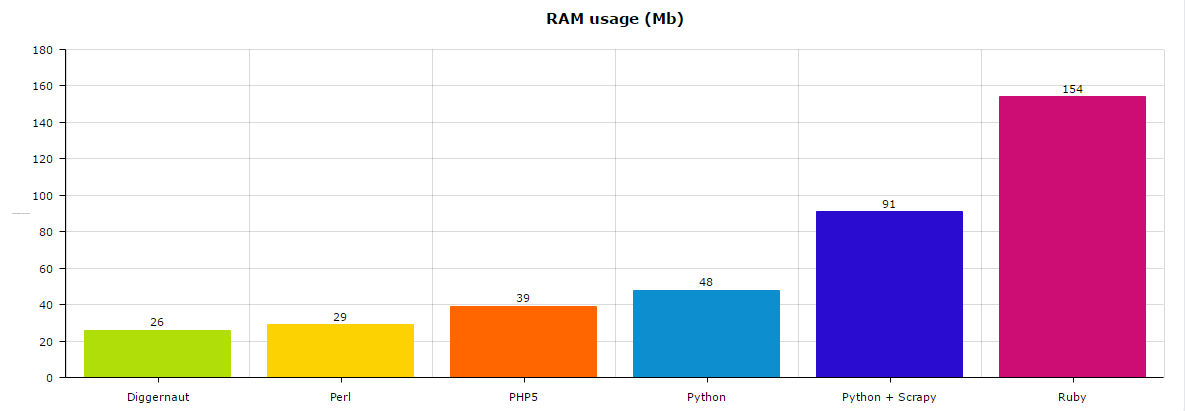


Рисунок 3 – Использование ОЗУ(Мб)

Победителем в этом измерении является Golang с 26Mb, за ним идет Perl с 29Mb, и замыкает тройку PHP5 с 39Mb. Ruby здесь показал себя полным аутсайдером со 154Mb.

Чтобы обобщить все тесты, мы оценим каждое испытание по 100-бальной системе (лучший результат получит 100 баллов, худший 0, остальные соответственно своим результатам) и затем возьмем среднее арифметическое по всем 3 испытаниям.

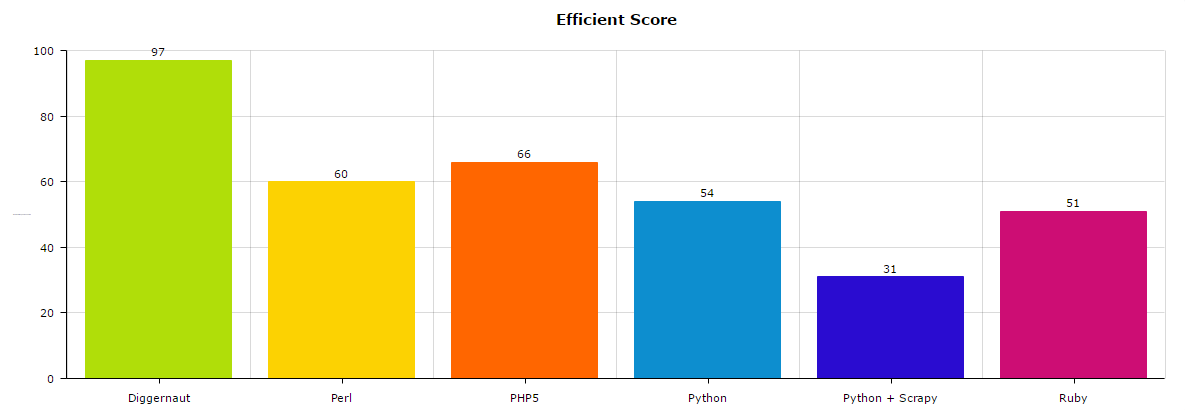


Рисунок 4 – Таблица результатов

Golang является чистым победителем в этом испытании. Именно поэтому он был выбран в качестве бакэнда для платформы Diggernaut.

2.3 Востребованность навыка написания парсеров на рынке труда

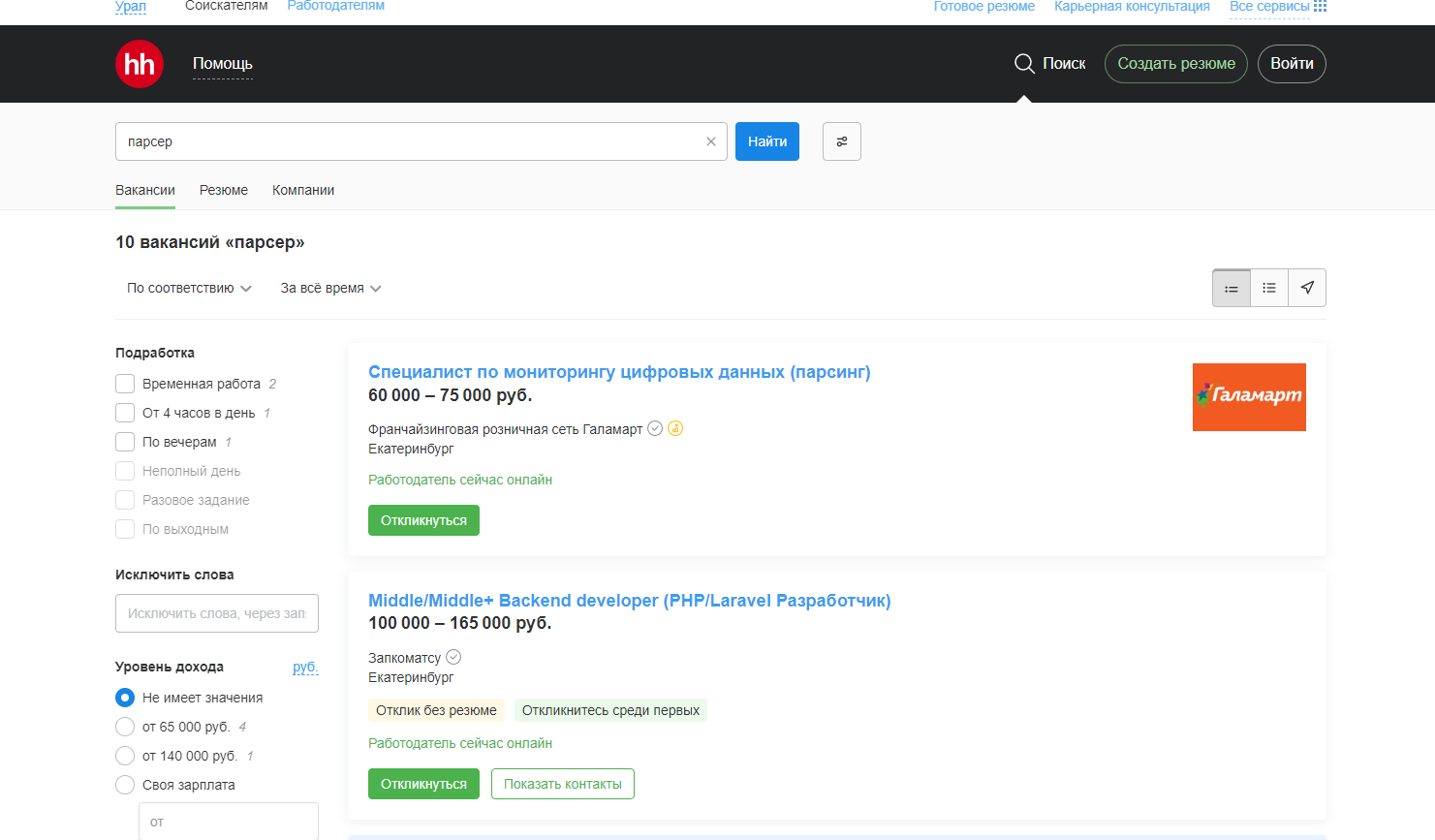
Востребованность навыка мы определили по поиску вакансий на крупнейшей российской компании интернет-рекрутмента - hh.ru. 

Рисунок 5 – Скриншот со страницы поиска вакансий на hh.ru.

На рисунке №5 происходил поиск по ключевому слову “парсер” нашлось 10 активных вакансий.

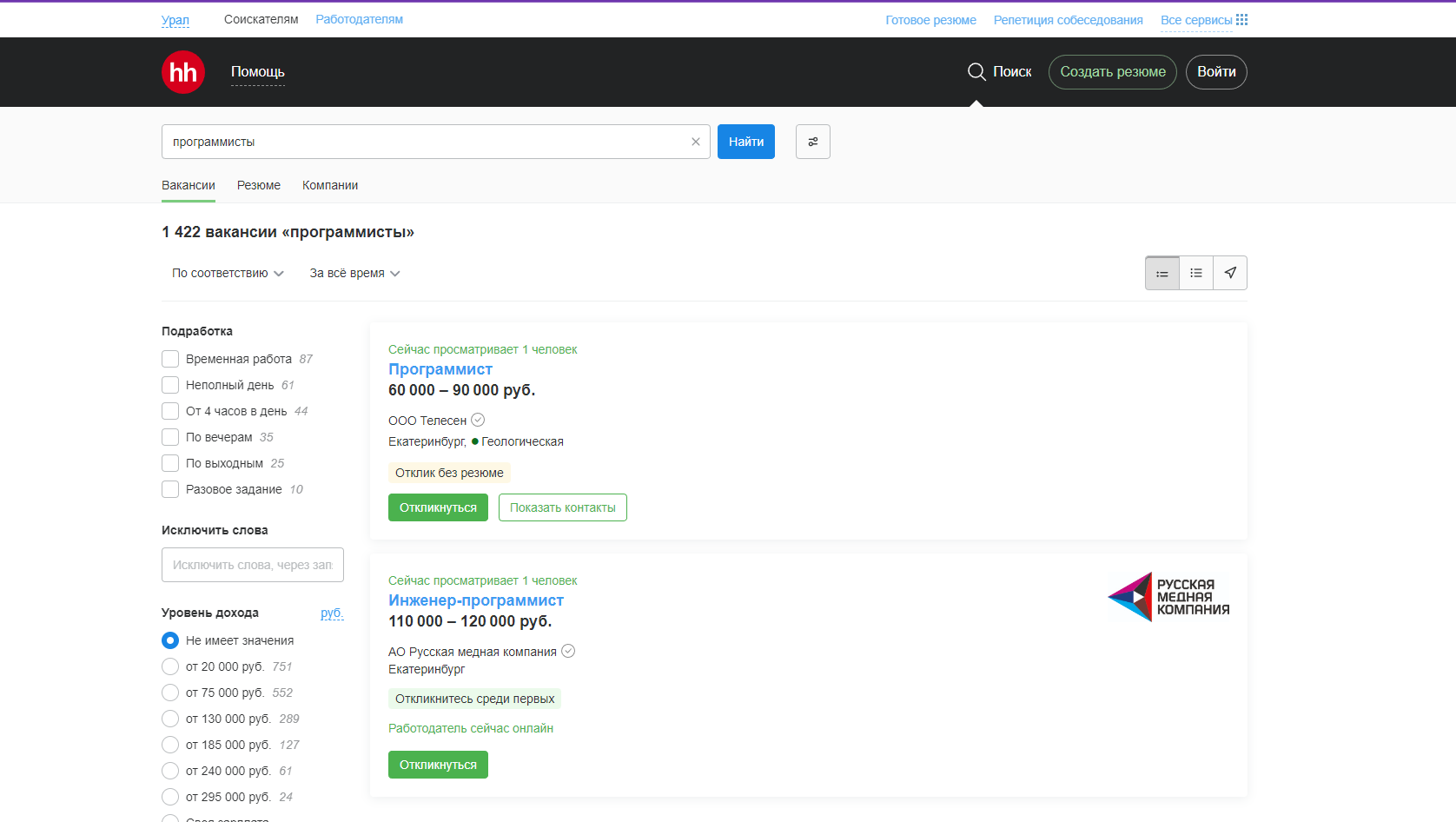


Рисунок 6 – Скриншот со страницы поиска вакансий на hh.ru

На рисунке №6 происходил поиск по ключевому слову “программисты” нашлось 1422 активных вакансии. Из которых, учитываются программисты 1С и стажёры, которые уменьшают количество рабочих мест в несколько раз.

Из полученных данных, мы можем сделать вывод, что навык парсинга уже включается в базовые навыки программиста.

3 Разработка парсера

В этой курсовой работе мы использовали телеграмм-бота для сбора информации с сайта прогноза погоды.

Заходим в телеграмм и ищем там @BotFather. После команды /start мы ввели название телеграмм-бота

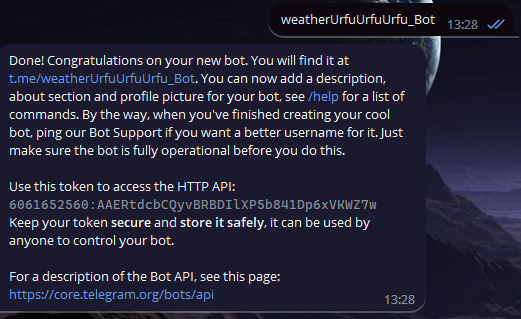


Рисунок 7 – Получение токена для телеграмм-бота.

После этого BotFather сгенерирует нам уникальный токен с помощью которого мы получим доступ к нему. Данный токен лучше держать в секрете и никому не показывать, потому что узнав его можно получить доступ к этому боту.

3.1 Постановка задачи

Для курсовой работы была выбрана задача научиться разрабатывать парсер на платформе телеграмм. Мессенджер облегчает работу с конечным пользователем, предлагает узнать прогноз погоды, а так же знакомит с погодой на всю неделю в вашем регионе по городу, индексу и отправленной геолокации. Подписавшись на бота, он должен уведомлять пользователя о погоде утром на весь день.

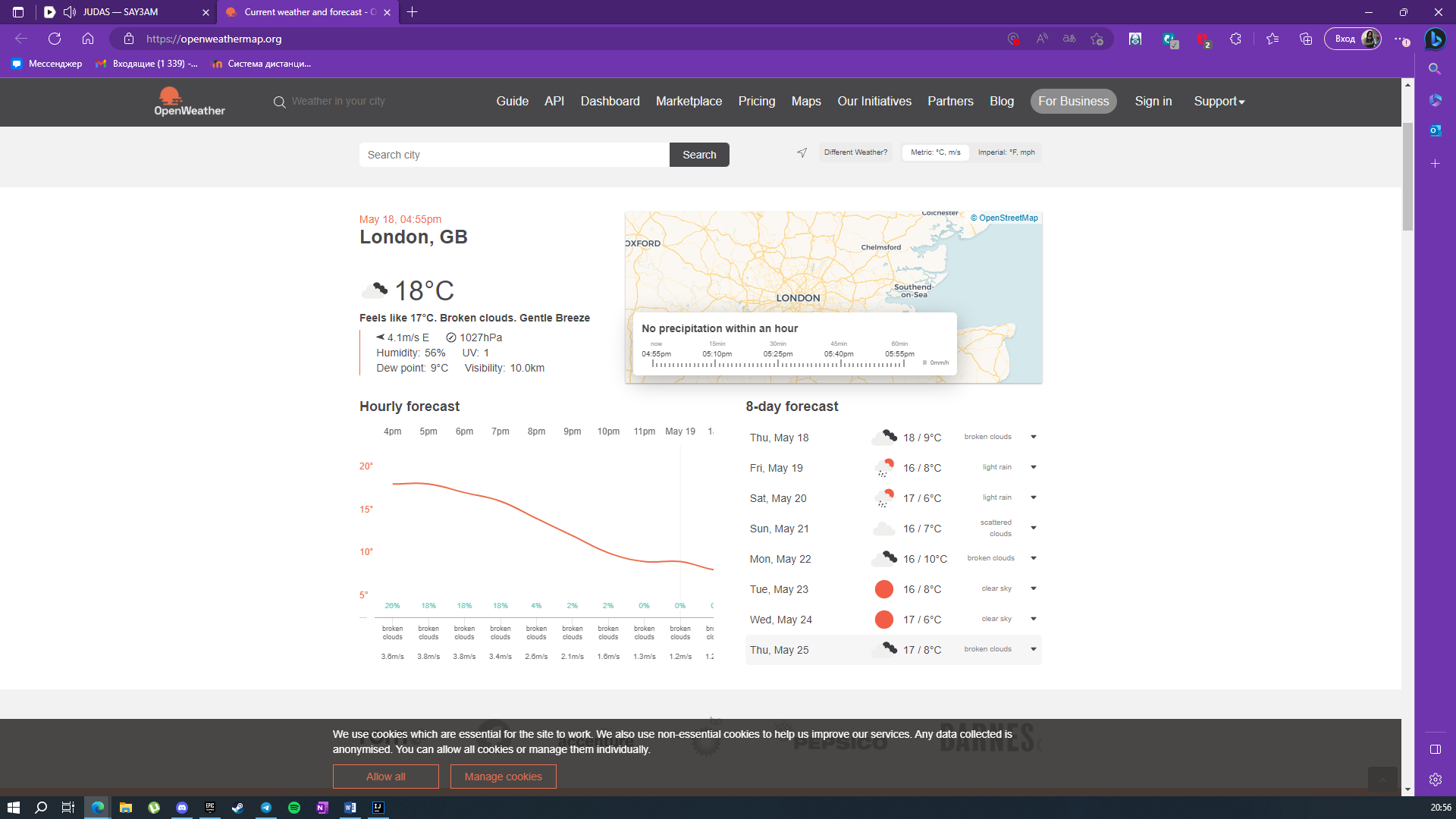


Рисунок 8 – Демонстрация сайта погоды.

3.2 Описание решения поставленной задачи

Для решения поставленной задачи мы изучили документацию telegram API. Познакомились с фреймворками Spring Boot и Lombok, и Fasterxml.Jackson.

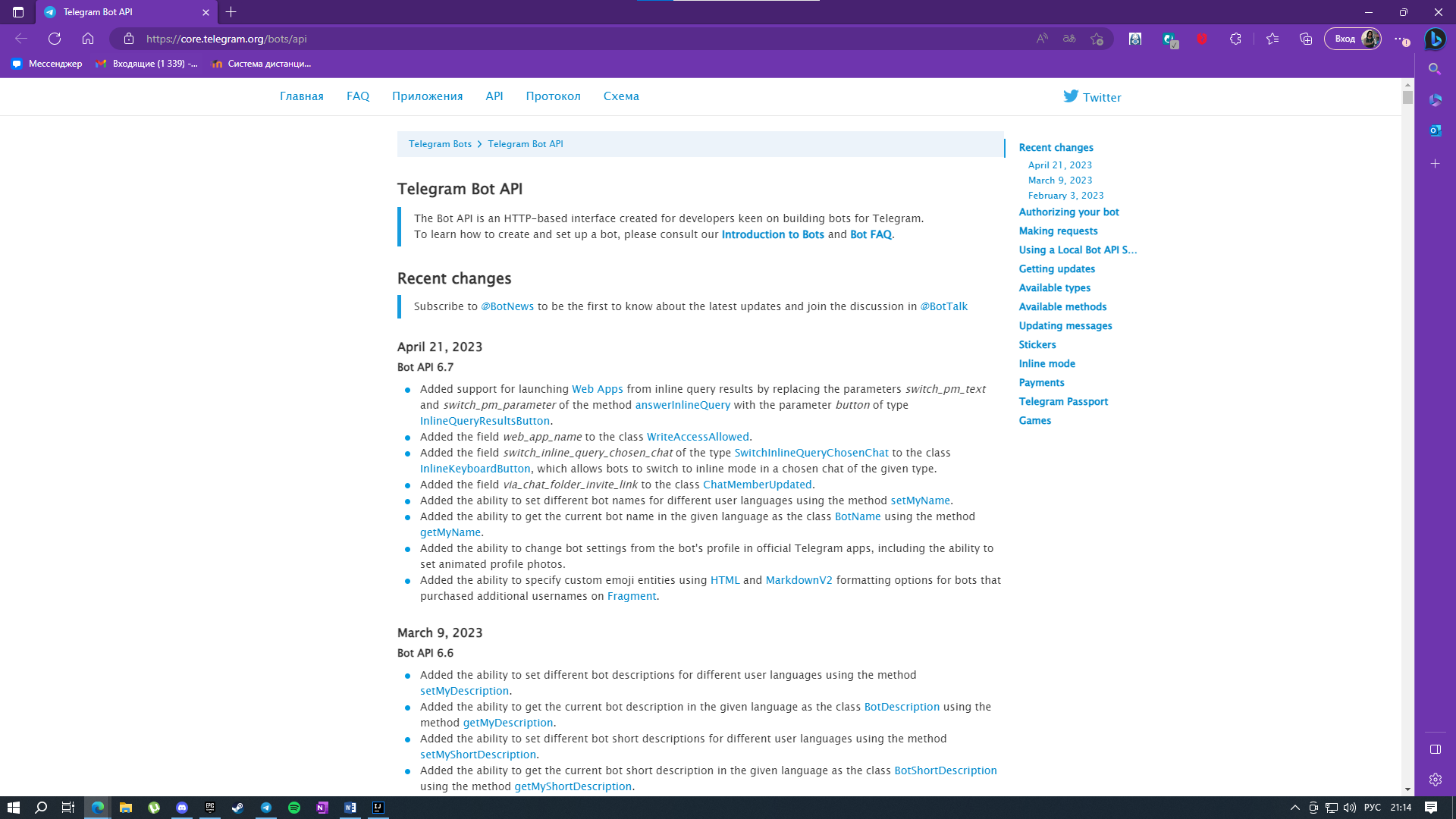


Рисунок 9 – telegram API.

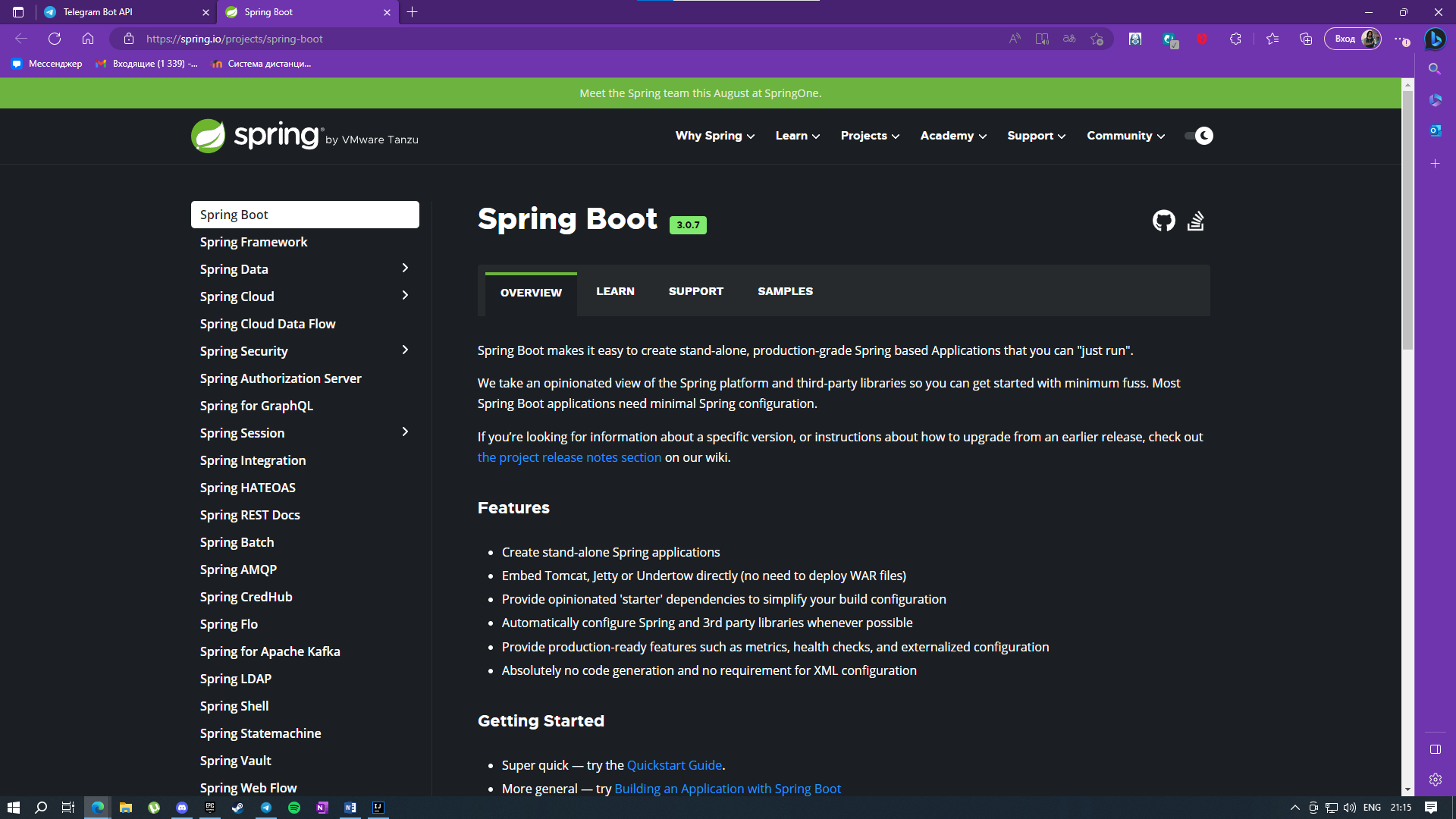


Рисунок 10 - Spring Boot.

3.3 Реализация парсера (описание кода реализованного парсера)

package com.fishev.vasyaev.service;  
  
import com.fishev.vasyaev.config.Keyboard;  
import com.fishev.vasyaev.config.User;  
import com.fishev.vasyaev.enums.BotState;  
import lombok.extern.slf4j.Slf4j;  
import com.fishev.vasyaev.config.BotConfig;  
import com.fishev.vasyaev.exception.PlaceNotFoundException;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Component;  
import org.telegram.telegrambots.bots.TelegramLongPollingBot;  
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;  
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;  
import org.telegram.telegrambots.meta.api.objects.CallbackQuery;  
import org.telegram.telegrambots.meta.api.objects.Message;  
import org.telegram.telegrambots.meta.api.objects.Update;  
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;  
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;  
  
  
import java.io.IOException;  
import java.net.URISyntaxException;  
import java.util.ArrayList;  
import java.util.Map;  
  
@Component  
@Slf4j  
public class Bot extends TelegramLongPollingBot {  
  
 final BotConfig config;  
 Map<Long, User> users;  
 @Autowired  
 WeatherRequest weatherRequest;  
 @Autowired  
 Serializer serializer;  
 @Autowired  
 Keyboard keyboard;  
  
 public Bot(BotConfig config) {  
 this.config = config;  
 }  
  
 @Override  
 public String getBotUsername() {  
 return config.getBotUserName();  
 }  
  
 @Override  
 public String getBotToken() {  
 return config.getToken();  
 }  
  
 @Override  
 public void onUpdateReceived(Update update) {  
 Long chatId;  
  
 if (users == null) {  
 users = serializer.deserialize();  
 }  
 if (update.hasMessage()) {  
 chatId = update.getMessage().getChatId();  
 } else if (update.hasCallbackQuery()) {  
 chatId = update.getCallbackQuery().getFrom().getId();  
 } else {  
 return;  
 }  
 if (users.get(chatId) == null) {  
 users.put(chatId, new User());  
 }  
 if ((update.hasMessage() && update.getMessage().hasText()) || update.hasCallbackQuery()) {  
 readUserCommand(update, chatId);  
 }  
 if (update.hasMessage()) {  
 processBotState(update.getMessage(), chatId);  
 } else if (update.hasCallbackQuery()) {  
 processBotState(update.getCallbackQuery(), chatId);  
 }  
 serializer.serialize(users);  
 }  
  
 private void readUserCommand(Update update, Long chatId) {  
  
 String userCommand;  
  
 if (update.hasMessage() && update.getMessage().hasText()) {  
 userCommand = update.getMessage().getText();  
 } else if (update.hasCallbackQuery()) {  
 userCommand = update.getCallbackQuery().getData();  
 } else {  
 return;  
 }  
  
 if (users.get(chatId).getBotState() != BotState.*DEFAULT* &&  
 (userCommand.equals("/current") || userCommand.equals("/future"))) {  
 sendTextMessageToUser(chatId, "сначала отправь город, индекс или GPS координаты");  
 return;  
 }  
 switch (userCommand) {  
 case "/settings":  
 users.get(chatId).clear();  
 users.get(chatId).setBotState(BotState.*CHANGE\_SETTINGS*);  
 break;  
 case "/current":  
 users.get(chatId).setBotState(BotState.*CURRENT\_FORECAST*);  
 break;  
 case "/future":  
 users.get(chatId).setBotState(BotState.*FUTURE\_FORECAST*);  
 break;  
 case "/start":  
 users.get(chatId).clear();  
 users.get(chatId).setBotState(BotState.*BOT\_START*);  
 break;  
 case "/subscribe":  
 users.get(chatId).setBotState(BotState.*BOT\_SUBSCRIBE*);  
 break;  
 case "/unsubscribe":  
 users.get(chatId).setBotState(BotState.*BOT\_UNSUBSCRIBE*);  
 break;  
 }  
 }  
  
 public void sendTextMessageToUser(Long chatId, String text) {  
 SendMessage sendMessage = new SendMessage();  
 sendMessage.setChatId(chatId.toString());  
 sendMessage.setText(text);  
 try {  
 execute(sendMessage);  
 } catch (TelegramApiException e) {  
 *log*.error(e.toString(), e);  
 }  
 }  
  
 private void processBotState(Message message, Long chatId) {  
 switch (users.get(chatId).getBotState()) {  
 case *DEFAULT*:  
 sendTextMessageToUser(chatId, "Не понимаю, выбери команду из меню \uD83D\uDE43");  
 break;  
 case *CHANGE\_SETTINGS*:  
 sendTextMessageToUser(chatId, "Отправь название города на русском или английском языке, " +  
 "6 цифр индекса города РФ или геолокацию");  
 users.get(chatId).setBotState(BotState.*READ\_SETTINGS*);  
 break;  
 case *READ\_SETTINGS*:  
 readSettings(message, chatId);  
 break;  
 case *CURRENT\_FORECAST*:  
 requestCurrentForecast(chatId);  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 break;  
 case *FUTURE\_FORECAST*:  
 requestFutureForecast(chatId, null);  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 break;  
 case *BOT\_START*:  
 sendTextMessageToUser(chatId, "Привет, я умею в погоду \uD83D\uDE42 \nВведи город, индекс или GPS координаты");  
 users.get(chatId).setBotState(BotState.*READ\_SETTINGS*);  
 break;  
 case *BOT\_SUBSCRIBE*:  
// sendTextMessageToUser(chatId, "Вы успешно подписались \uD83D\uDE42");  
 subscriptionActivity(message, chatId);  
 users.get(chatId).setBotState(BotState.*CHANGE\_SETTINGS*);  
 break;  
  
 case *BOT\_UNSUBSCRIBE*:  
// sendTextMessageToUser(chatId, "Вы так же успешно отписались \uD83D\uDE42");  
 subscriptionDeactivity(message, chatId);  
 users.get(chatId).setBotState(BotState.*CHANGE\_SETTINGS*);  
 break;  
  
  
 }  
  
 }  
  
 private void processBotState(CallbackQuery callbackQuery, Long chatId) {  
 switch (users.get(chatId).getBotState()) {  
 case *CHANGE\_SETTINGS*:  
 sendTextMessageToUser(chatId, "Отправь название города на русском или английском языке, " +  
 "6 цифр индекса города РФ или геолокацию");  
 users.get(chatId).setBotState(BotState.*READ\_SETTINGS*);  
 break;  
 case *CURRENT\_FORECAST*:  
 requestCurrentForecast(chatId);  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 break;  
 case *FUTURE\_FORECAST*:  
 requestFutureForecast(chatId, null);  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 break;  
 case *DEFAULT*:  
 requestFutureForecast(chatId, callbackQuery);  
 }  
 }  
  
 private void subscriptionActivity(Message message, Long chatId) {  
 try {  
 if (!users.get(chatId).getIsSubscriptions()) { // Если false то подписываемся на рассылку  
 if (users.get(chatId).getIsCity()) {  
 sendTextMessageToUser(chatId, "Вы успешно подписались ");  
 users.get(chatId).setSubscriptions(); // и меняем значение isSubscriptions на TRUE c помощью метода setSubscriptions  
 foo();  
 } else {  
 sendTextMessageToUser(chatId, "Перед тем как подписаться на рассылку, выбери город ");  
 }  
  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void subscriptionDeactivity(Message message, Long chatId) {  
 try {  
 if (users.get(chatId).getIsSubscriptions()) { // Если true то может позволить отписаться от рассылки  
 sendTextMessageToUser(chatId, "Вы также успешно отписались ");  
 users.get(chatId).setSubscriptions(); // и меняем значение isSubscriptions на FALSE c помощью метода setSubscriptions  
  
 } else {  
 sendTextMessageToUser(chatId, "Перед тем как отписаться от рассылки, выбери город, да, даже это я предусмотрел, негодник");  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Scheduled(fixedRate = 5000L)  
 public void foo() {  
 System.*out*.println("прошло 5 секунд с момента подписки ");  
  
 }  
  
  
 private void readSettings(Message message, Long chatId) {  
  
 if (message.hasLocation()) {  
 users.get(chatId).setLocation(message.getLocation());  
 sendMessageToUserWithKeyboard(chatId, users.get(chatId).getSettings(), keyboard.afterSettingsKeyboard());  
 } else if (message.hasText()) {  
 String userInput = message.getText();  
 if (userInput.equals("/current") || userInput.equals("/future") || userInput.equals("/subscribe") || userInput.equals("/unsubscribe")) {  
 return;  
 }  
 if (isIndex(userInput)) {  
 users.get(chatId).setIndex(Integer.*parseInt*(userInput));  
 sendMessageToUserWithKeyboard(chatId, users.get(chatId).getSettings(), keyboard.afterSettingsKeyboard());  
 } else if (isCity(userInput)) {  
 users.get(chatId).setCity(userInput);  
 sendMessageToUserWithKeyboard(chatId, users.get(chatId).getSettings(), keyboard.afterSettingsKeyboard());  
 } else {  
 sendTextMessageToUser(chatId, "Что-то не похоже на индекс или название города \uD83E\uDD13");  
 }  
 }  
 }  
  
 private void sendMessageToUserWithKeyboard(Long chatId, String text, InlineKeyboardMarkup keyboard) {  
 SendMessage sendMessage = new SendMessage();  
 sendMessage.setChatId(chatId.toString());  
 sendMessage.setText(text);  
 sendMessage.setReplyMarkup(keyboard);  
 try {  
 execute(sendMessage);  
 } catch (TelegramApiException e) {  
 *log*.error(e.toString(), e);  
 }  
 }  
  
  
 private void requestCurrentForecast(Long chatId) {  
  
 String messageText = null;  
 try {  
 if (users.get(chatId).getCity() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getCity(), "weather", null);  
 } else if (users.get(chatId).getIndex() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getIndex(), "weather", null);  
 } else if (users.get(chatId).getLocation() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getLocation(), "weather", null);  
 }  
 } catch (PlaceNotFoundException e) {  
 messageText = "Такой город по названию или индексу не найден, измени настройки";  
 } catch (IOException | URISyntaxException e) {  
 messageText = "Произошла критическая ошибка на при запросе прогноза погоды от сервера. Попробуйте позже";  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 *log*.error(e.toString(), e);  
 }  
  
 sendTextMessageToUser(chatId, messageText);  
 }  
  
 private void requestFutureForecast(Long chatId, CallbackQuery callbackQuery) {  
  
 String messageText = null;  
 String userChoice;  
 if (callbackQuery == null) {  
 userChoice = null;  
 } else {  
 userChoice = callbackQuery.getData();  
 }  
  
 try {  
 if (users.get(chatId).getCity() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getCity(), "forecast", userChoice);  
 } else if (users.get(chatId).getIndex() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getIndex(), "forecast", userChoice);  
 } else if (users.get(chatId).getLocation() != null) {  
 messageText = weatherRequest.getForecast(users.get(chatId).getLocation(), "forecast", userChoice);  
 }  
 } catch (PlaceNotFoundException e) {  
 messageText = "Такой город по названию или индексу не найден, измени настройки";  
 } catch (IOException | URISyntaxException e) {  
 messageText = "Произошла критическая ошибка на при запросе прогноза погоды от сервера. Попробуйте позже";  
 users.get(chatId).setBotState(BotState.*DEFAULT*);  
 *log*.error(e.toString(), e);  
 }  
  
 ArrayList<String> dates = weatherRequest.getForecastDates();  
  
 if (callbackQuery != null) {  
 editMessage(chatId, messageText, keyboard.futureForecastKeyboard(dates, userChoice),  
 callbackQuery.getMessage().getMessageId());  
 } else  
 sendMessageToUserWithKeyboard(chatId, messageText, keyboard.futureForecastKeyboard(dates, null));  
 }  
  
 private void editMessage(Long chatId, String text, InlineKeyboardMarkup keyboard, Integer messageId) {  
 EditMessageText editMessageText = new EditMessageText();  
 editMessageText.setChatId(chatId.toString());  
 editMessageText.setText(text);  
 editMessageText.setMessageId(messageId);  
 editMessageText.setReplyMarkup(keyboard);  
 try {  
 execute(editMessageText);  
 } catch (TelegramApiException e) {  
 *log*.error(e.toString());  
 }  
 }  
  
 private boolean isIndex(String userInput) {  
 for (char c : userInput.toCharArray()) {  
 if (!Character.*isDigit*(c)) {  
 return false;  
 }  
 }  
 int index;  
 try {  
 index = Integer.*parseInt*(userInput);  
 } catch (NumberFormatException e) {  
 return false;  
 }  
 return Integer.*toString*(index).length() == 6;  
 }  
  
 private boolean isCity(String userInput) {  
 for (char c : userInput.toCharArray()) {  
 if (Character.*isDigit*(c)) {  
 return false;  
 }  
 }  
 for (char c : userInput.toCharArray()) {  
 if (!Character.*isAlphabetic*(c) && c != ' ' && c != '-' && c != '\'') {  
 return false;  
 }  
 }  
 return userInput.length() > 1;  
 }  
}

Реализация основного класса Bot в Idea, а всё остальное Вы может ознакомиться в следующем пункте.

3.4 Результат работы парсера

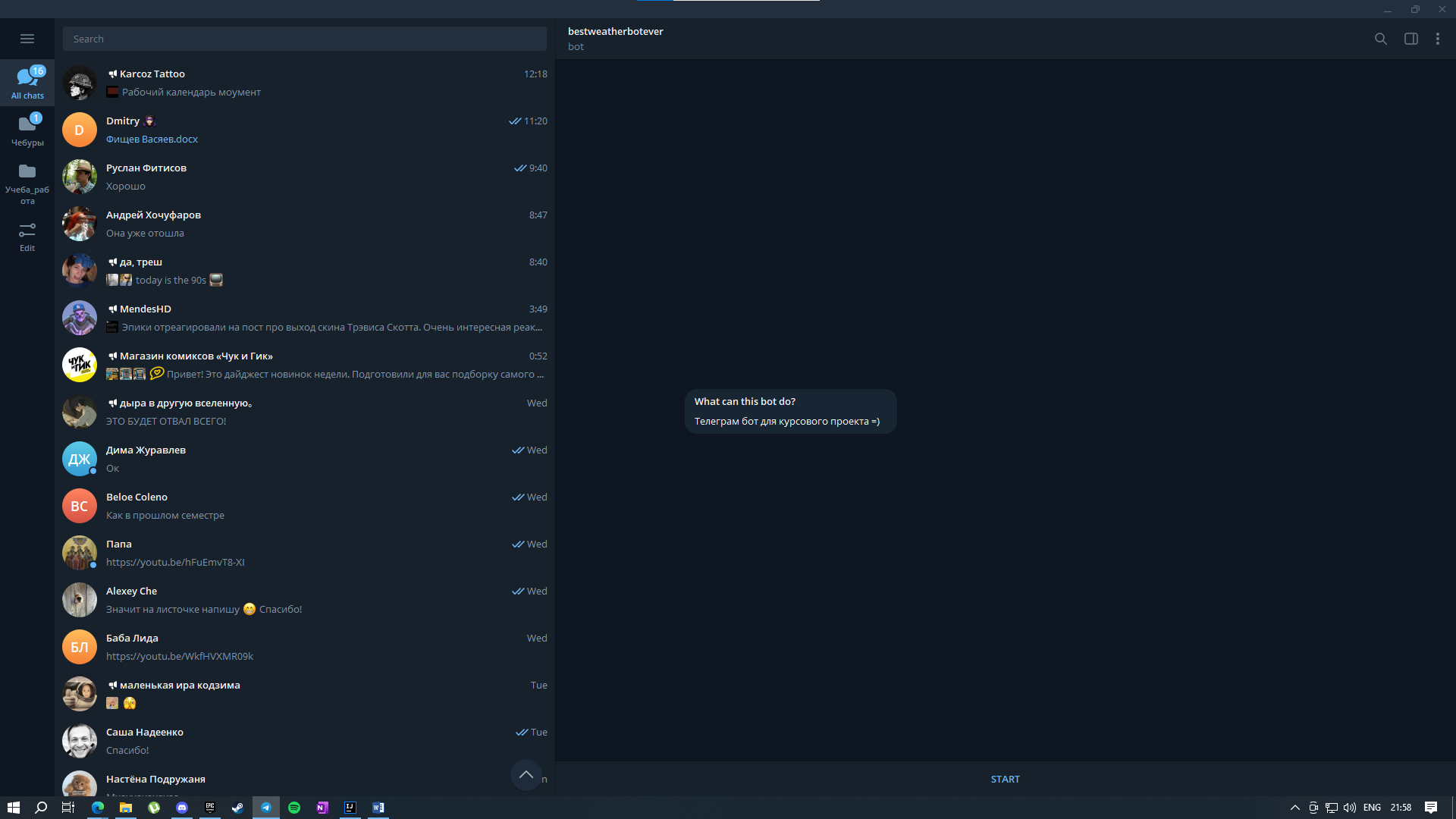


Рисунок 11 – telegram-bot.

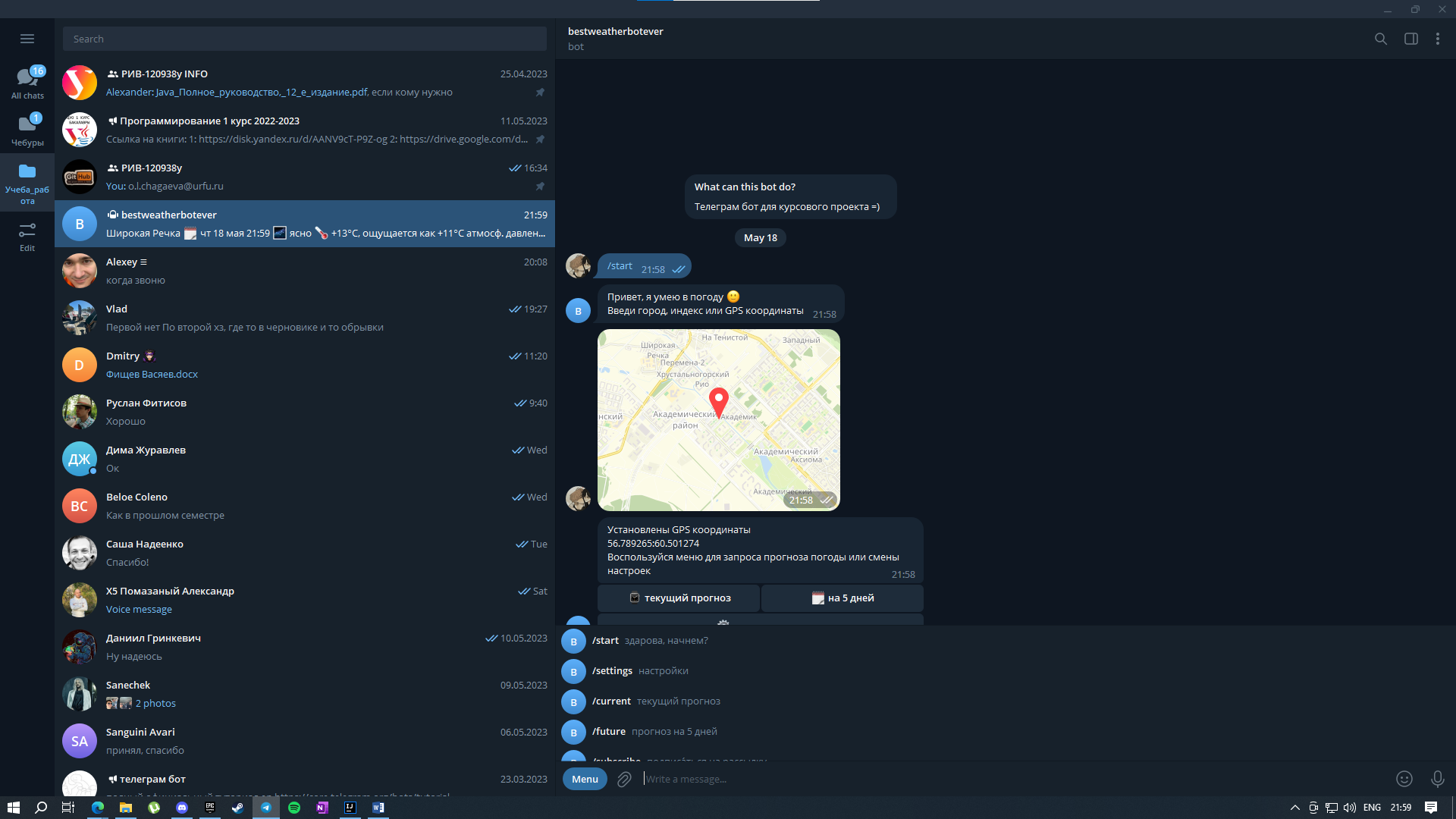


Рисунок 12 – список команд в telegram-bot.

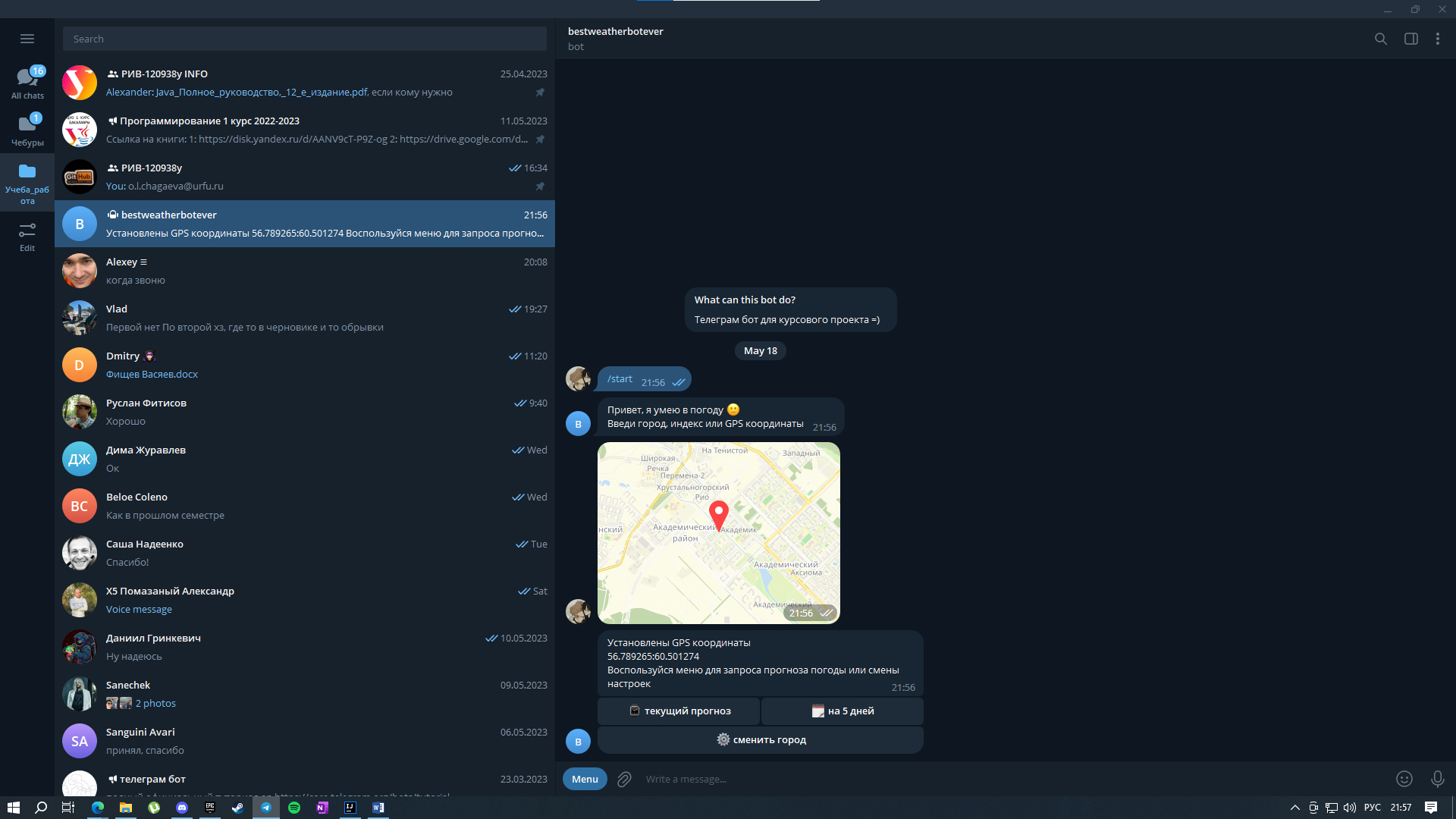


Рисунок 13 – Выбор погоды по геолокации.

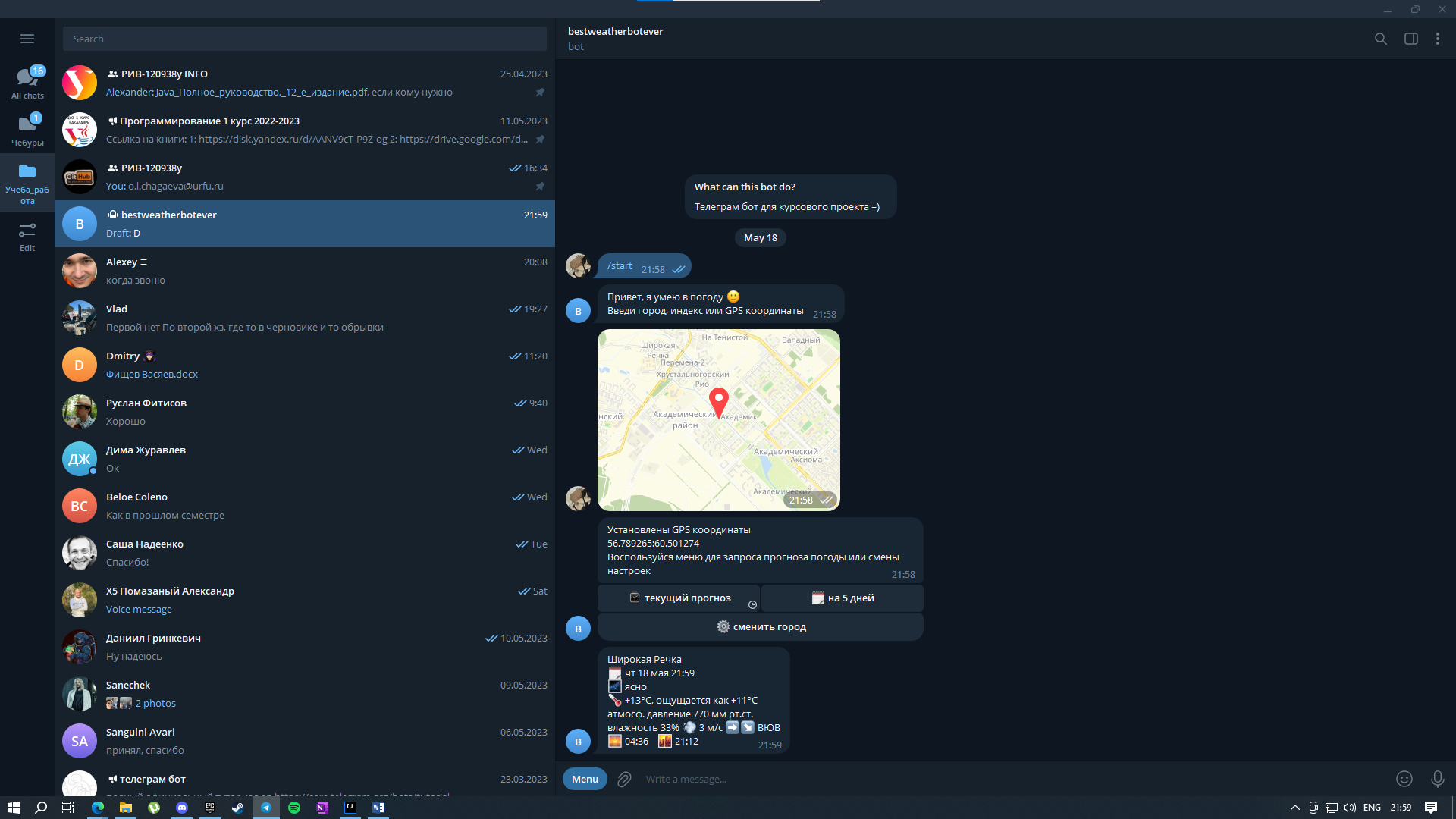


Рисунок 14 – Текущий прогноз погоды.

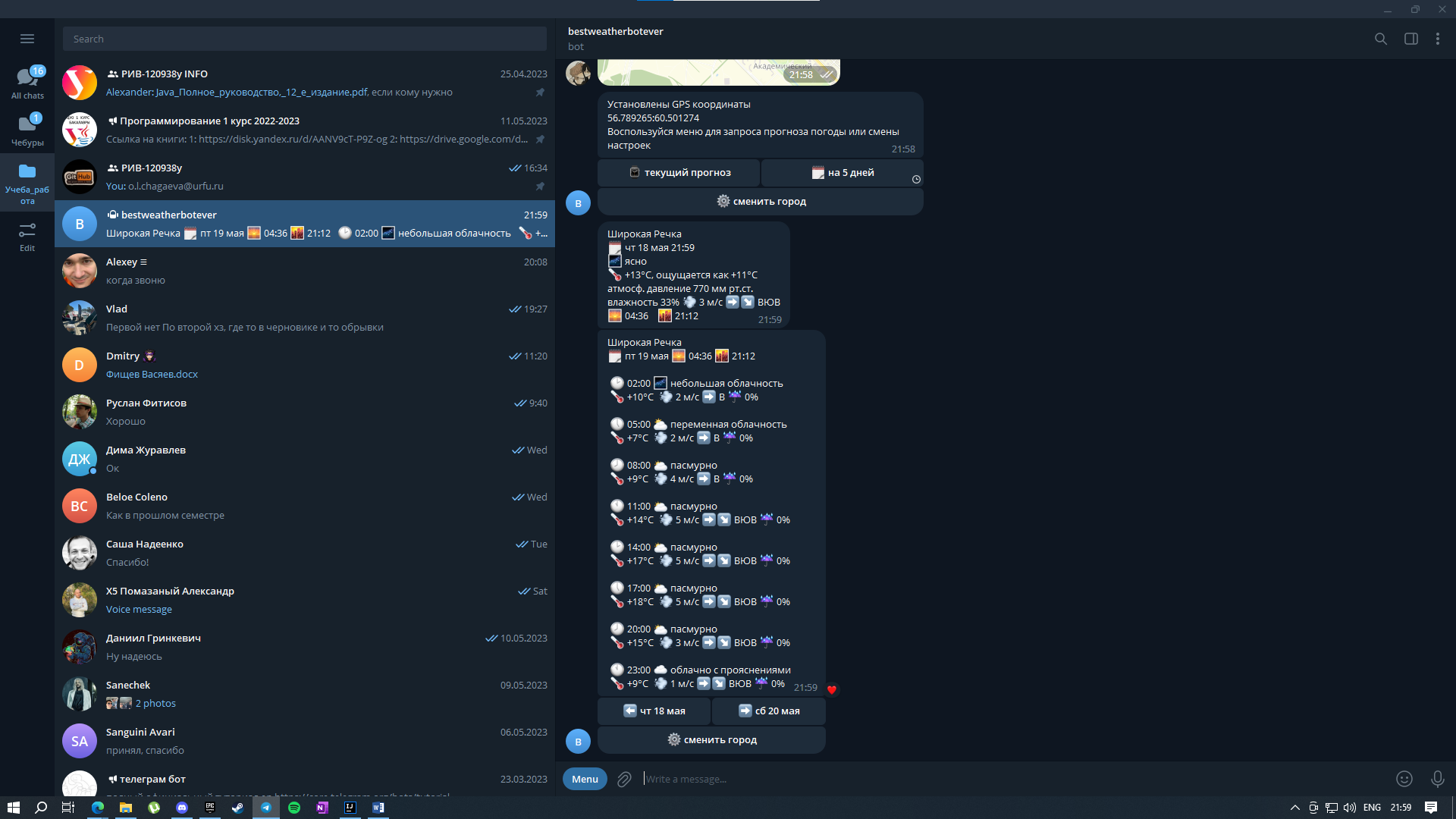


Рисунок 15 – Прогноз на 5 дней.

[GitHub - papakarlorabotaet/WeatherTelegramBot](https://github.com/papakarlorabotaet/WeatherTelegramBot)

<https://t.me/weatherUrfuUrfuUrfu_Bot>



Рисунок 16 – 1-ый QR-code ссылка на репозиторий GitHub,   
 2-ой QR-code ссылка на сам telegram-bot.

ЗАКЛЮЧЕНИЕ

В курсовой работе приобрели навыки разработки парсера на платформе telegram. Познакомились c фреймворками. Научили бота собирать данные с сайта с погодой и преобразовывать их в сообщения, обрабатывать входящие сообщения от конечного пользователя, в которых указаны индекс, город и геолокация.