Частное образовательное учреждение высшего образования «Казанский инновационный университет имени В.Г. Тимирясова (ИЭУП)»

КОЛЛЕДЖ

Курсовой проект

по МДК. 02.01 Разработка, внедрение и адаптация программного обеспечения отраслевой направленности

на тему:

Разработка, внедрение и адаптация программного обеспечения для Анализа рынка труда

Выполнил:
обучающийся гр. К1081/9
В. С. Кузнецов

_____

Руководитель:

_____

Казань 2021

СОДЕРЖАНИЕ

Введение

“Современный рынок труда ежегодно растёт и меняется.” — с такой мыслью год тому назад, в 2020-м году начинался отчёт автора по учебной практике по схожей теме. В этой работе описан процесс разработки программного обеспечения для анализа рынка труда, а именно рынок труда в сфере информационных технологий.

Данная работа является продолжением и развитием выше указанного проекта.

Цель работы — разработка программного обеспечения для анализа вакансий базы данных одного из самых крупных сайтов по поиску вакансий и сотрудников — HeadHunter. Автора заинтересован в исследовании самых конкурентноспособных навыков соискателей.

Задачи работы:

1) рассмотреть методологию разработки программного обеспечения в рамках цели исследования;
2) изучение методологии анализа рынка труда и набора квалификаций;
3) разработка программного обеспечения;
4) применение разработанного ПО и дальнейший анализ.

Актуальность работы заключается в конкретизации требуемых от соискателей навыков и компетенций. В дальнейшем, полученный список навыков, может стать основой для улучшения образовательных программ заведений среднего и высшего образования. Периодические анализы рынка труда помогут студентам сформировать представление об актуальных требованиях рынка труда, а сфере образования улучшить качество образовательных программ.

Объектом исследования является рынок труда информационных технологий.

Предметом исследования выступает выборка требуемых квалификаций соискателей.

1. Методология разработки

1.1. Языковая средства, системы и технологии программирования.

Перед тем, как приступить к написанию любого приложения, важно выбрать платформу (рабочую и целевую), стек (англ. stack — стопка) технологий, иначе говоря, набор инструментов, применяемых в совокупности для достижения тех или иных программных целей.

Так, основными критериями в выборе подходящего инстумента стали:

- мультиплатформенность — возможность исполнять программу на большинстве современных платформ;
- производительность — ввиду нужды в обработке больших объемов информации.
- ассинхронный ввод-вывод — для совершения множества параллельных запросов к API сайта HeadHunter

По итогу обзора имеющихся на рынке технологий, выбор пал между **Node.js** (язык **JavaScript**) и **Python**.

За последние пару десятков лет, язык Python однозначно стал своего рода лингва франка для компьютерных вычислений в научных исследованиях, ввиду богатой стандартной библиотеки, однако, было решено использовать именно Node.js (далее — Node) из-за предрасположенности платформы к ассинхронному вводу-выводу и вычислениям.

1.1.1. JavaScript

JavaScript — C-подобный язык программирования с утиной (неявной динамической) типизацией.

JavaScript появился в 1995-м году как разработка корпорации Netscape Communications, лидирующей в те годы в области разработки веб-браузеров,

использующийся в качестве простого скриптового языка для придания веб-страницам возможности динамически реагировать на действия пользователя.

Уже с самого появления языка стало ясно, что для поддержки совместимости с разными браузерами необходима стандартизация языка, однако, ввиду спешки, было решено проводить стандартизацию не в W3C (Консорциум Всемирной паутины), известной продолжительным процессом стандартизации, а также не предрасположенной для стандартизации спецификаций ПО независимых поставщиков (vendor-independent), а в ECMA International, позиционирующую себя, как организацию стандартизации, лояльную и ориентируемую на бизнес. Помимо этого, стандарты ECMA International принимаются и могут быть рассмотрены Международной Организацией по Стандартизации (ISO).

Уже в 1996-м году при содействии Netscape, Sun Microsystems и Microsoft, и ряда других компаний, на заседании 21-22 ноября состоялась встреча технического коммитета 39 (TC39), ответственного за стандартизацию языка JavaScript.

Первый стандарт языка, названный ECMAScript, был предложен ISO уже в 1997-м году, однако, в документации стандарта были найдены неточности, и спустя год, в 1998-м году ISO приняло вторую редакцию ECMAScript.

Таким образом, JavaScript стал международным стандартом веба, использующимся и по сей день.

В 1999-м году вышла третья редакция стандарта языка, но четвёртая редакция, запланированная на 2003-й год, была отменена ввиду разногласий между компаниями-вендорами.

И так, практически до 2015-го года стандарт языка не менялся.

Между тем, в 2008-м году компания Google совместно со своим браузером Google Chrome, выпустили и движок JavaScript **V8**, совершивший революцию в быстродействии исполнения JavaScript благодаря JIT-

компиляции (англ. just-in-time, компиляция «на лету») и внутренним оптимизациям.

В контексте Веба, JavaScript считается достаточно медленным языком, однако, если убрать из него самую времязатратную составляющую — инструменты для манипуляции **DOM (HTML, CSS)**, благодаря оптимизациям V8 (JIT-компиляция [2, с. 33]) и асинхронной природе Node.js [2, с. 51], данный язык становится достаточно быстрым не только для требуемых для проекта задач, но и в целом обгоняя такие крупные языки, как Python, R и прочие, традиционно используемые в статическом анализе языки.

1.1.2. Node.js

Node — это платформа и среда исполнения JavaScript-кода и работы с системой ввода-вывода. Раньше Node.js использовался исключительно как решение для сервера, однако теперь Node можно установить практически на любой машине для самых различных целей [2, с. 16].

Основным аргументом к использованию Node были асинхронный ввод-вывод [2, с. 51] и быстродействие. Изначально, разработчик Node.js, Райан Даль (Ryan Dahl) позаимствовал данный подход из проекта веб-сервера Nginx, ставшего в результате самым популярным серверным решением для бизнеса. Данный подход позволяет Nginx значительно обгонять веб-сервер Apache по количеству обрабатываемых запросов и соединений.

1.1.3. TypeScript

TypeScript — это подмножество языка JavaScript, типизированное расширение [1, с. 18], язык программирования со статической сильной типизацией, разработанный в Microsoft, с открытым кодом, в качестве замены JavaScript на этапе разработки.

Преимущества TypeScript:

- TypeScript использует современный синтаксис последних версий ECMAScript [1, с. 237];
- TypeScript компилируется в JavaScript выбранной версии [1, с. 260];
- TypeScript привносит богатую систему типов [1, с. 44];
- TypeScript позволяет выбирать нужные опции проверки [1, с. 25].
- Все программы JavaScript поддерживаются в TypeScript, но не наоборот [1, с. 19];
- Помогает IDE быстро использовать технологию IntelliSense (технология автодополнения кода) [1, с. 12].

JavaScript — объектно-ориентированный язык со слабой “утиной” типизацией [1, с. 35], где всякая сущность является объектом. В отличии от других объектно-ориентированных языков программирования, JavaScript использует прототипное наследование. Под этим имеется ввиду, что вместо наследования классов от классов, объекты наследуется от других объектов.

Для автоматического дополнения JavaScript, компилятор (сервер IntelliSense) должен проанализировать AST (абстрактное синтаксическое дерево) объекта, и контекста, окружающего данный объект, его внутренние десятки прототипов, что является слишком производительной и медленной операцией. Это значит, что посмотрев на функцию, нельзя сказать ничего о её аргументах, т.к. нет представления о том, какие поля и особенности данные аргументы должны иметь. Данное ограничение с одной стороны позволяет проводить быстрое прототипирование информационных систем и кодовой базы, однако, на крупных проектах, данный фактор скорее является проблемой, т.к. разработчикам нужно согласовывать и узнавать у других разработчиков, какие аргументы имелись ввиду в каждом отдельном случае, что крайне замедляет процесс разработки программного обеспечения.

Для решения данной проблемы, TypeScript имеет собственную систему конкретно записанных типов из стандартной библиотеки. TypeScript-компилятор предоставляет собственную систему статических типов и анализа для обработки технологией IntelliSense языка JavaScript и TypeScript, предоставляя быстрое и отзывчивое решение.

1.1.4. Yarn

Node.js имеет собственную систему контроля библиотек (пакетов) — NPM (node package manager). Однако, существуют альтернативы. В данном проекте использовся yarn — проект пакетного менеджера от Facebook.

Главным отличием yarn от npm является процесс установки. npm устанавливает библиотеки последовательно, а yarn — асинхронно. Более того, yarn позволяет использовать режим "flat" (англ. плоский), трансформируя установку древа зависимостей, с возможностью наличия нескольких версий одних пакетов, в подобие плоского графа, с наличием только одной версии для каждого пакета. Таким образом, с использованием данного инструмента, установка библиотек для работы программы будет устанавливаться **быстрее**.

1.2. Методы анализа рынка труда в России

1.2.1. Историческая справка

Первые биржи труда в России стали появляться ещё во времена Российской Империи, практически сразу после отмены крепостного права. Возникновение наёмной рабочей силы своевременно дало почву для появления и развития первых коммерческих бирж труда. Ещё в конце XIX века начинали появляться общественные движения, выступающие за создание некоммерческих бирж труда, однако, благотворительные биржи труда оказались неконкурентоспособными по сравнению с коммерческими. [8]

В советские годы, правительство начало бороться с безработицей путём централизации бирж труда. Биржи труда были преобразованы в “Органы учета и распределения рабочей силы”, на которые была возложена ответственность не только к распределению кадров, но и ведению статистики текучести кадров. [8]

Накоплен огромный фактологический массив данных. Так, в библиографии по рынку труда, составленной А. Исаевым в 1925 г., указано более 120 работ, посвященных вопросам организации и реорганизации бирж труда в 20-е годы, и более 100 работ об их деятельности. [8]

Уже к концу 1918 г. частные конторы, посреднические бюро при профсоюзах, Союзы безработных для трудоустройства солдат были ликвидированы. [8]

С первыми шагами индустриализации рынок труда перестает напоминать капиталистический. Началось сворачивание НЭПа и переход к плановой экономике, при которой надобность в традиционных биржах труда полностью отпала. Их заменили отделы кадров и бюро по трудоустройству. [8]

1.2.2. Современность

В современных условиях открытого рынка труда, на смену как государственным, так и коммерческим оффлайн биржам трудоустройста, устройству по знакомству родственников и знакомых, пришли онлайн платформы, аккумулирующие огромные массивы информации, в частности вакансии работодателей и резюме соискателей.

На момент 2021-го года, поиск работы в России наиболее представлен популярной одноименной платформой компании HeadHunter, в базу которой на данный момент входят порядка 1 млн. вакансий, 50 млн. резюме соискателей, 1.5 млн компаний (включая ИП и самозанятых) со всей России, СНГ и даже далекого зарубежья. Платформа еженедельно обрабатывает около 1 млн приглашений на собеседования.

Сложно себе представить, что имея такие большие массивы данных, и обладая популярнейшей площадкой в России и СНГ для нахождения персонала и работодателя, компания HeadHunter бы остановилась только на этом. Действительно, начиная с 2014-го года компания регулярно публикует в открытом доступе собственные исследования и еженедельные обзоры рынка труда, предоставляя как современным, так и будущим исследователям бесценный материал для дальнейших изучения. Совершенствуются и внедряются новые более совершенные методы подборки персонала и вакансий на основе машинного обучения.

Помимо частных компаний, проблемами и развитием рынка труда в России не в меньшей мере занимается государство, а именно Министерство труда. Ежегодно публикуются отчёты и доклады о проделанной работе министерства, проведённых мероприятиях, установленных в прошлом целей развития, результаты и выводы проделанной политики, и устанавливаются новые цели для развития и решения проблем рынка труда.

2. Реализация

2.1. Способы поиска вакансий

Платформа HeadHunter предоставляет открытый доступ ко множеству методов собственного внутреннего API [3], в том числе и к методам поиска открытых вакансий.

API работает по протоколу HTTPS, что гарантирует безопасность при передаче данных, доступных только в формате JSON.

2.1.1. Ограничения API

Однако, предоставленное платформой HeadHunter API поиска вакансий имеет ограничения по глубине поиска, не позволяя получить более **2000** элементов [3].

Помимо этого, получение вакансий в виде списка не предоставляет полной информации о вакансии, т.е. в список попадают сокращённые версии, в которых не содержатся списки ключевых навыков — основного предмета исследования.

Информация полной вакансии получается по отдельному запросу на каждую определённую вакансию. Данное ограничение, как и нужда, заставляет задуматься о **кэшировании** (сохранении) результатов запросов, для уменьшения количества лишних запросов к минимуму, снятия нагрузки с сети, а также сокращения интервалов (времени) ожидания для повторного получения полной информации о вакансии с 100-200 мс до 1-10 мс на запрос. Более того, если в дальнейшем API введет ограничение на количество запросов в секунду, данная мера предотвратит или уменьшит нужду в балансировке (введении строгой очередности) запросов.

2.1.2. Методы обхода ограничений API

Методы поиска API HeadHunter могут возвращать кластеризованные результаты, сгруппированные по следующим признакам:

- Профобласть;
- Специализация;
- Регион;
- Метро;
- График работы;
- Опыт работы;
- Зарплата;
- Исключения;

Каждый кластер содержит в себе ссылку на поисковую выдачу с некоторым фильтром и предполагает использование с целью сузить количество найденных вакансий.

Таким образом, используя кластеризованную выдачу, планируется разбивать имеющиеся запросы на более специализированные.

2.1.3. Алгоритм поиска вакансий

Исходя из поставленных выше проблем при поиске и методов их обхода, составлен следующий алгоритм для поиска вакансий в условиях ограничения по глубине поиска в 2 тыс.:

1. Сначала происходит инициализирующий запрос, имеющий в себе информацию о регионе поиска, тексте запроса (ключевому слову в названии вакансии) и парамтр для получение кластеризованной выдачи;
2. Полученные кластеры по регионам разделяются на такие, в которых меньше или равно 2 тыс. вакансий, и на такие, в которых больше:
    a. В первой категории кластеров происходит стандартный поиск по всем 20 страницам выдачи размером 100 вакансий на страницу, а полученные списки вакансий в итоге объединяются;

b. Во второй категории, каждый кластер разбивается по линиям метро.
    i. Однако, если на линии метро расположено больше 2 тыс. вакансий, кластер разбивается по станциям метро;
    ii. По полученным кластерам, имеющим количество вакансий меньше или равное 2 тыс., проводится стандартный поиск, как в пункте 2.a;
c. Полученные списки сокращённых вакансий объединяются;

Таким образом получаются списки сокращённых вакансий. Для их получения, в среднем, требуется количество запросов, равное числу общего числа вакансий, разделённому на число в интервале от 100 (ограничению по глубине поиска на 1 страницу поисковой выдачи) до 110. Так, для 55,000 вакансий, количество запросов будет находится на интервале между 500 и 550.

На самом деле, настоящее количество запросов зависит от реального распределения вакансий по кластерам и субкластерам, полученным в результате дробления крупного кластера на меньшие ветви. Так, во время активации программы на момент написания данной работы, на 54931 полученных вакансий по запросу “программист”, получены 504 ссылки для запросов.

В ходе разработки программы по выше указанному алгоритму, были найдены новые ограничения, а конкретно, их рамки. Так, при потоковой отправке однотипных запросов к API, примерно после 2 тыс. запросов API начинает отправлять ошибку, в целях закрытия сессии.

Более того, выясненно, что подобное поведение API можно обойти, если отправлять запросы не в виде потока, а группируя их на чанки (англ. chunk — кусок) произвольных размеров. В результате продолжительных экспериментов, был вычислен оптимальный размер чанка в пределах числа 50. Т.е. в каждом чанке будет содержатся 50 ссылок для запросов.

Основная идея в разделении общей совокупности ссылок на чанки состоит в том, что пока совершаются запросы по ссылкам одного

произвольного чанка, и пока возвращаются ответы по данным запросам, проходит необходимая естественная задержка, а именно около 2-х сек. на чанк. Таким образом, группируя запросы, программа не перегружает сессию и API лишними запросами, и API не отправляет ошибку в целях остановить поток запросов.

Так, имея на данный момент 504 ссылки, мы делим общий массив на 50 (число, полученное выше опытным путем), и получаем 11 чанков.

Ожидаемое время работы алгоритма в таком случае, будет равно общему времени обработки каждого чанка, примерно равное количеству чанков, умноженному на среднее время обработки одного чанка, на данный момент равное 2 сек.

Однако данный алгоритм обхода ограничений также имеет свои проблемы. А именно: итоговое количество полученных вакансий не совпадает с изначальным ожиданием, в процессе теряется около 10-20% вакансий. Так, для 54934 вакансий ожидания, по окончанию работы алгоритма, возвращается 44103 вакансий, и потери равняются 10831 (около 19.71% от ожидания).

Данная проблема не критична для результатов исследования, т.к. общие статистические показатели будут вычисляться не на основе обработки отдельных вакансий, но на соотношениях уже подготовленных кластеров.

На основе полученных вакансий будут анализироваться **ключевые навыки**. Для получения информации о ключевых навыках требуется получить полные версии вакансий, доступные у каждой сокращённой вакансии по отдельному запросу.

Согласно указанным выше ограничениям API, это значит, что для получения полных вакансий требуется совершить количество запросов, равное количеству вакансий. Так, для 44103 вакансий, требуется сделать 44103 новых запросов.

Для получения этой информации, помимо описанного ранее способа разделения общего числа запросов на чанки, для этой задачи необходимо организовать работу кэширования полученных данных.

Составлен следующий алгоритм получения полных версий вакансий:

1. Получаем массив сокращенных вакансий;
2. Преобразуем полученный массив, из каждой вакансии получаем ссылку на полную версию;
3. Разделяем полученные ссылки на чанки по 100 ссылок;
4. На каждый чанк совершается группа запросов;
5. Полученные полные вакансии объединяются в список.
6. Полученный список полных вакансий может занимать слишком много места в оперативной памяти, потому, для оптимизации, список преобразуется, выделяются только необходимые для дальнейшего анализа поля.

2.2. План ПО

Построенная в результате проектирования система приложения структурно состоит из трёх состовляющих:

- Модуль CLI — текстовый интерфейс пользователя, доступный из командной строки;
- Модуль ядра — функциональная составляющая программы, занимается получением вакансии из сети;
- Модуль утилит — вспомогательные функции.

Каждый модуль из себя представляет набор файлов, содержащих набор функций. Эти функции преобразуют данные и/или применяют к ним функции из других модулей. Далее описан каждый модуль в отдельности и его внутренние подмодули:

2.2.1. Интерфейс пользователя

Модуль CLI (англ. Command Line Interface — Интерфейс Командной строки) или же TUI (англ. Text User Interface — Текстовый Интерфейс Пользователя) предоставляет инструменты для общения, вызова методов функционального ядра логики приложения. В своей основе CLI построен на JavaScript библиотеке "Commander" [4] (см. Приложение Б, листинг 1.1). Данная библиотека упрощает написание собственного CLI и в сущности является "оберткой" над встроенными средствами Node.js по работе с аргументами коммандной строки.

Модуль CLI предоставляет доступ к одной функции, инициализирующей объект сущности для манипуляций с коммандной строкой. Инициализируются комманды, доступные из интерфейса коммандной строки.

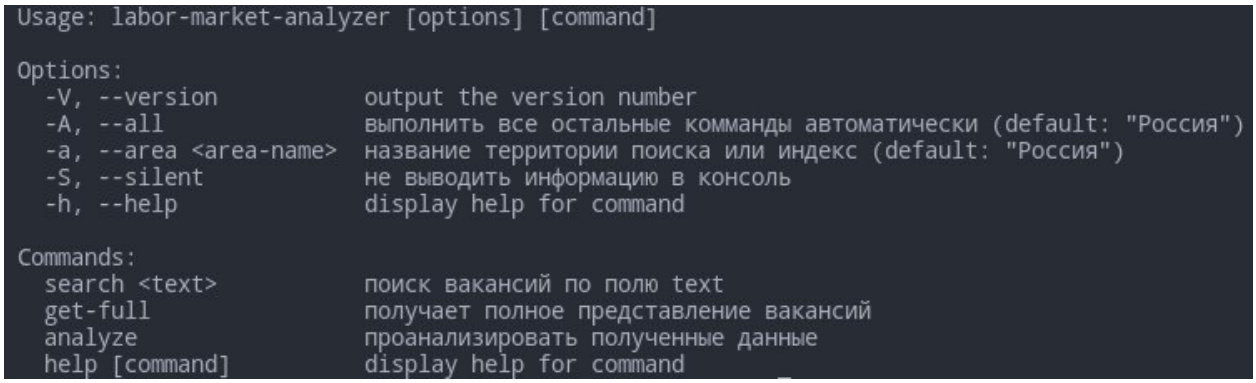

Рисунок 1 - Интерфейс пользователя в коммандной строке

2.2.2. Ядро

Модуль ядра объединяет в себе крупных два модуля и предоставляет интерфейс к ним:

- Модуль запросов — включает в себя все функции для взаимодействия с сетью и получения вакансий в частности;
- Модуль анализа — анализирует полученные вакансии и кластеры;

Ядро спроектировано таким образом, что предоставляет единый интерфейс коммуникации, и при потребности, может быть использован другими интерфейсами. Так, можно заменить интерфейс пользователя с коммандной строки на графический или веб-интерфейс (API).

Далее следует подробное описание внутренней работы данных модулей.

2.2.3. Модуль запросов

Ключевую роль в сборе информации из сети занимают специализированные запросы, занимающие наибольшую часть времени исполнения приложения. Язык JavaScript предоставляет Fetch API — гибкий и мощный инструмент для получения ресурсов из сети.

Для ускорения работы запросов используется:

- Не блокирующий ввод-вывод (асинхронные запросы);
- Кэширование запросов.

Первое включено в систему Node.js. и называется Promise API и Fetch API, описанные ранее. Асинхронное взаимодействие Fetch и Promise реализовано использованием статического метода Promise.all(), данный метод возвращает массив значений от всех промисов.

В Node.js нет встроенной реализации Fetch API, реализованной во всех современных браузерах, но имеется ряд библиотек, реализующих интерфейсы по стандарту Fetch API из крайних стандартов JavaScript. В данном проекте использовалось библиотека "node-fetch" [5], наиболее популярная и поддерживаемая реализация Fetch API для Node.js.

2.2.4. Запросы с кэшированием

Как уже было описано ранее, в процессе работы программы запланировано исполнение и ожидание тысяч запросов в сеть для получения информации о полных версиях вакансий из сети. Это значит, что при каждом новом запуске программы, придётся ожидать новых запросов, уже совершенных при предыдущем поиске. Чтобы предотвратить это, единственным решением является кэширование результатов.

Для решения этой проблемы была написана собственная мини-библиотека, выполняющей функции отправки запросов, их кэширования и чтения из кэша уже совершённых запросов.

Библиотека объединяет в себе модули кэширования и хеширования.

2.2.4.1. Кэширование запросов

Под кэшированием подразумевается временное сохранение результатов уже совершённых когда-то запросов в локальную память. При этом, ожидание уже полученных (кэшированных) ответов на порядки ниже, чем если ожидать новые запросы. Так, среднее время ожидания одного запроса равно около 200-300 мс, когда чтение информации из файла, расположенного в локальной памяти компьютера составляет около 20 мс.

Современные браузеры предоставляют собственную систему кэширования запросов и страниц, однако, в Node.js встроенная реализация также отсутствует. Был проведен поиск подходящей библиотеки для решения поставленной задачи, и не найдя таковой, было решено написать собственную реализацию.

Алгоритм написанного простого кэщирования следующий:

1. Хешировать строку запроса;
2. Проверить на существование директории "cache":
    a. Если существует — продолжить работу;

b. Если отсутствует — создать такую директорию;

3. Проверить на наличие в директории "cache" на наличие файла с названием. содержащим хеш, полученный в пункте 1;
    a. Если существует — прочитать из данного файла;
    b. Если отсутствует — сделать новый запрос и сохранить его результат в директории "cache" с названием хэша из пункта 1.
4. Вернуть результат

2.2.4.2. Хеширование

Модуль хеширования предоставляет доступ к функции хеширования по алгоритму MD5. Алгоритм хеширования MD5 был выбран из-за более высокой скорости, ввиду меньшего количество итераций по сравнению с SHA.

Криптографически, алгоритм слабее алгоритмов SHA, однако, в данной задаче нет требований по обеспечению безопасности, т.к. пользователю не требуется вводить или хранить собственные персональные данные.

Алгоритм, по сравнению с другими более совершенными и современными криптографическими алгоритмами вычисления хеш-сумм, имеет проблему появления коллизий, однако, на малой выборке, используемой в данной программе (не более 50 тысяч сообщений), шанс появления коллизий незначителен и стремится к нулю.

2.2.5. Модуль утилит

Модуль утилит состоит из ряда вспомогательных функций, выделенных в отдельный модуль для структуризации часто используемого программного кода. Структурно, модуль объединяет следующие подмодули:

- Модуль форматирования
- Модуль пагинации
- Модуль “подсказок”

2.2.5.1. Форматирование

Написание собственных объектов структур хранения информации предполагает также и написание методов для преобразования данных объектов с целью объединить или разделить хранимую такими объектами информацию в новые сущности для дальнейшего использования.

Примером этого может послужить преобразование строки, содержащей дату в формате ISO 8601 (например, 2012-09-27) в объект, содержащий поля года 2012, месяца 09 и дня 27. Таким же образом, такой объект может быть преобразован в удобную строку и транспортирован.

Похожим образом в программе выглядит форматирование объекта пользовательского типа запроса в строку формата URL, для дальнейшего использования для совершения запросов в сеть.

Также модуль форматирования предоставляет доступ к функции для преобразования массива кластеров в объект с полями в виде названий этих кластеров. Таким образом, упрощая доступ к данным этого кластера, давая возможность обращаться не по индексу кластера в массиве, а по его названию.

2.2.5.2. Пагинация

Пагинация в данном случае — это разветвление одного запроса на несколько запросов для разных страниц. Модуль пагинации предоставляет одну функцию, принимающую в качестве аргумента массив субкластеров, имеющих поля URL и количества вакансий, относящихся к данному адресу.

2.2.5.3. Подсказки

Модуль подсказок требуется для преобразования слов естественного языка в набор кодов или идентификаторов, использующихся в работе API HeadHunter, Например, если хочется сузить поисковую выдачу на

определённом регионе, API требует использования в качестве значения региона его код, указанный в системе HeadHunter, а не его естественное имя. Так, чтобы получить все вакансии с России, в качестве аргумента “area” требуется передать значение 113.

Модуль подсказок предоставляет доступ к функции, использующей в качестве аргумента слово естественного языка, затем она обращается к специальному методу API HeadHunter для получения идентификатора региона, и возвращает идентификатор в качестве ответа.

2.2.6. Подготовка к анализу

Анализ полученных данных происходит в две стадии:

- Обработка информационных кластеров;
- Обработка вакансий.

Ввиду большого количества обрабатываемой информации, в целях ускорения анализа, все данные упрощаются:

- Кластеры форматируются для удобного доступа к каждому подкластеру информации;
- Вакансии преобразуются, выделяются только важные для исследования поля информации.

Полученные полные версии вакансий помимо полезной для исследования информации предоставляют также и второстепенные, не относящиеся к теме данные. Для исследования статистических показателей на базе полученных вакансий критичны следующие поля данных:

- Ключевые навыки — наборы компетенций, которые работодатель ожидает от претендентов на вакансию;
- Наличие теста в вакансии — иногда работодатель желает испытать соискателей на решение определенного тестового задания, по окончанию которого, можно составить общее представление о компетенции;

- Обязательность выполнения теста — тестовые задания могут иметь необязательный характер, даже если соискатель не сможет выполнить задачу, он всё ещё может претендовать на место в компании;
- Обязательность наличия ответного письма — часто работодатель желает нанять не только компетентного сотрудника, но и заинтересованного в конкретно данной вакансии, помимо этого, данное условие может быть использовано в целях сокращения слишком большой выборки соискателей на более меньшую;
- Принимаемость неполных резюме — данное условие может свидетельствовать о том, что работодатель желает фильтровать такие резюме, в которых не представлена важная информация о соискателе, например, опыт работы или фотография, наличие образование и т.д.;
- Принимаемость на временное трудоустройство — так можно узнать, нацелен ли работотель на продолжительное сотрудничество, либо, на решение конкретной проблемы тут и сейчас.

Выше описанные данные и их анализ в наибольшей степени приоритетны, по сравнению с текстовым содержанием вакансий и прочими данными, не предоставляющей в открытом виде информацию, по которой можно сформировать полезные статистические выводы о состоянии рынка труда.

2.2.7. Модуль анализа

Модуль анализа предоставляет доступ к одной функции, принимающей в качестве аргументов подготовленные и отобранные по выше описанным критериям вакансии и кластеры, полученные ранее.

Порядок анализа получченных данных следующий:

1. Анализ вакансий;
2. Анализ кластеров;
3. Объединение полученной информации.

2.2.7.1. Анализ вакансий

Анализ вакансий — наиболее важная часть данного исследования. Только проанализировав все полученные выше описанными методами можно получить информацию о требуемых современному рынку труда компетенциях.

Составлен следующий алгоритм взвешнвания и оценки ключевых навыков:

1. Полученные вакансии фильтруются с целью оставить только такие, где присутствуют ключевые навыки;
2. Отобранные вакансии преобразуются с целью получить поля ключевых навыков в едином списке;
3. Получив все ключевые навыки, происходит взвешиввание, с целью получить массив ключ-значений, где ключ — уникальный навык, а значение — количество его повторений в изначальном списке;
4. Полученный список преобразуется, на каждую пару ключ-значения вызывается функция для вычисления отношения количества повторений определённого навыка к количеству всех вакансий;
5. Взвешенный список фильтруется, чтобы оставить только такие навыки, отношение которых к общему числу вакансий больше или равно 1%;
6. Взвешенный список сортируется по убыванию.

Помимо компетенций, анализу подвергнуты и другие данные вакансий, перечисленные в пункте 2.2.5. Для данных булевых параметров происходит подсчёт вакансий с наличием этих параметров в истинных значениях, а далее происходит подсчёт отношений количеств к общему числу вакансий.

2.2.7.2. Анализ кластеров

Полученные кластеры можно условно разделить на две категории: вариационные и интервальные. Кластеры трудоустройства, рабочего графика и индустрии относятся к вариационным, т.к. не требуют лишних преобразований, и имеют только информацию о количестве и типе субкластеров. Например, кластер рабочего графика предоставляет субкластеры “Полный день”, “Удалённая работа”, “Гибкий график”, “Сменный график”, “Вахтовый метод”, у каждого из которых единственная полезная информация — это количество вакансий на каждую категорию.

Кластеры, относящиеся к вариационным, проходят общую обработку, где вычисляется только отношение определённого субкластера по количеству входящих в него вакансий ко всем найденным вакансиям.

К интервальным кластерам относятся кластеры уровня дохода и опыта работы, т.к. предоставляют информацию в виде интервалов. Например, “от 85000 руб”, “от 160000 руб” и т.д.. Т.е. можно представить данные кластера в виде интервалов “85000-160000”, для обработки которых понадобятся дополнительные преобразования. Например, в каждый n-ный субкластер кластера уровня дохода входят все последующие. Так, в субкластер “от 85000 руб” будут входить “от 160000 руб”, “от 235000” и последующие. Это значит, что для получения точного количества вакансий на каждом интервале нужно из количества вакансий в субкластере вычитать количество вакансий в следующем субкластере.

Субкластеры кластера уровня дохода изначально отсортированы по убыванию количества относящихся к ним вакансий. Однако можно заметить, что количество вакансий в первом субкластере меньше, чем общее количество найденных вакансий с указанной заработной платой. Это значит, что при анализе в самое начало списка требуется добавить ещё один субкластер, отражающий не входящие ни в один субкластер вакансии. Для данного

кластера выставим условную заработную плату в размере 48151 руб., средней предлагаемой в России по данным сайта HeadHunter на момент 2021-го года

Таким образом, полученные данные по кластеру уровня дохода преобразуются в удобную для подсчёта показателей интервального вариационного ряда.

2.3. Результаты анализа

В процессе обшего анализа были составлены таблицы по следующим регионам: Россия, Беларусь, Украина, Москва, Санкт-Петербург, Республика Татарстан (см Приложение А, таблицы 2.1-2.6).

Дальнейший анализ кластеров зарплат будет происходить на основе данных таблицы 2.1, т.е. только по России.

2.3.1. Кластеры

По обработанному кластеру уровня дохода (см Приложение А, таблица 2.1) получена следующая таблица (для умещения всех полезных данных для расчётов в одной таблице, использована экспоненциальная запись чисел):

Таблица 2.7 - вариационный ряд на основе кластера зарплат

интервалы		$x_{ср}$	f	$x_{ср} \times f$	S	$(x-x_{ср}) \times f$	$(x-x_{ср})^2 \times f$
13000	85000	66575,5	8387	410963000	8387	683945770,6	5,57746E+13
85000	160000	122500	7698	943005000	16085	61955977,25	4,98642E+11
160000	235000	197500	3729	736477500	19814	249662809,9	1,67153E+13
235000	310000	272500	1734	472515000	21548	246144210,9	3,49406E+13
310000	385000	347500	582	202245000	22130	126265877	2,73936E+13
385000	470000	427500	417	178267500	22547	123828850	3,67712E+13
Итого:			22547	2943473000		1491803496	1,72094E+14

2.3.1.1. Показатели вариации:

Для оценки полученной информации требуется вычислить показатели вариации, а именно: среднюю взвешанную, моду, медиану, размах, дисперсию и среднеквадратическое отклонение.

Находим среднюю взвешанную:

$$\tilde{x} = \frac{\sum x_i \cdot f_i}{\sum f_i} = \frac{2943473000}{22547} = 130548.321$$

Находим моду построенного интервального ряда по формуле:

$$M_o = x_0 + h\frac{f_2 - f_1}{(f_2 - f_1) + (f_2 - f_3)},$$

где x_0 — начало модального интервала;

h — величина интервала;

f_2 — частота, соответствующая модальному интервалу;

f_1 — предмодальная частота;

f_3 — послемодальная частота.

В качестве начала модального интервала выбираем 13000, так как именно на этот интервал приходится наибольшее количество вакансий.

$$M_o = 13000 + 72000\frac{8387 - 0}{(8387 - 0) + (8387 - 7698)} = 79534.156$$

Таким образом, наиболее часто встречающееся значение ряда равняется 79534.156.

Находим медиану вариации:

$$M_e = x_0 + \frac{h}{f_{me}}\left(\frac{\sum f_i}{2} - S_{me-1}\right),$$

$$M_e = 85000 + \frac{75000}{7698}\left(\frac{22547}{2} - 8387\right) = 113122.564$$

Таким образом, статистически, половина вакансий будут иметь заработную плату меньше по величине 113122.564, а другая половина — больше.

Затем, вычисляем абсолютные показатели:

Размах вариации:

$$R = x_{max} - x_{min} = 470000 - 13000 = 457000$$

Дисперсия:

$$D = \frac{\sum (x_i - \tilde{x})^2}{\sum f_i} = \frac{1{,}72094\text{E} + 14}{22547} = 7632677433{,}981$$

Несмещённая оценка дисперсии:

$$S^2 = \frac{\sum (x_i - \tilde{x})^2}{\sum f_i - 1} = \frac{1{,}72094\text{E} + 14}{22546} = 7633015971.967$$

Среднеквадратическое отклонение от средней величины:

$$\sigma = \sqrt{D} = 87365.196$$

Каждое значение ряда отличается от среднего значения 130548.321 в среднем на 87365.196.

Оценка среднеквадратического отклонения:

$$s = \sqrt{S^2} = 87367.133$$

Каждое значение ряда отличается от среднего значения 130548.321 в среднем на 87365.196.

На основе полученных данных составлена следующая гистограмма:

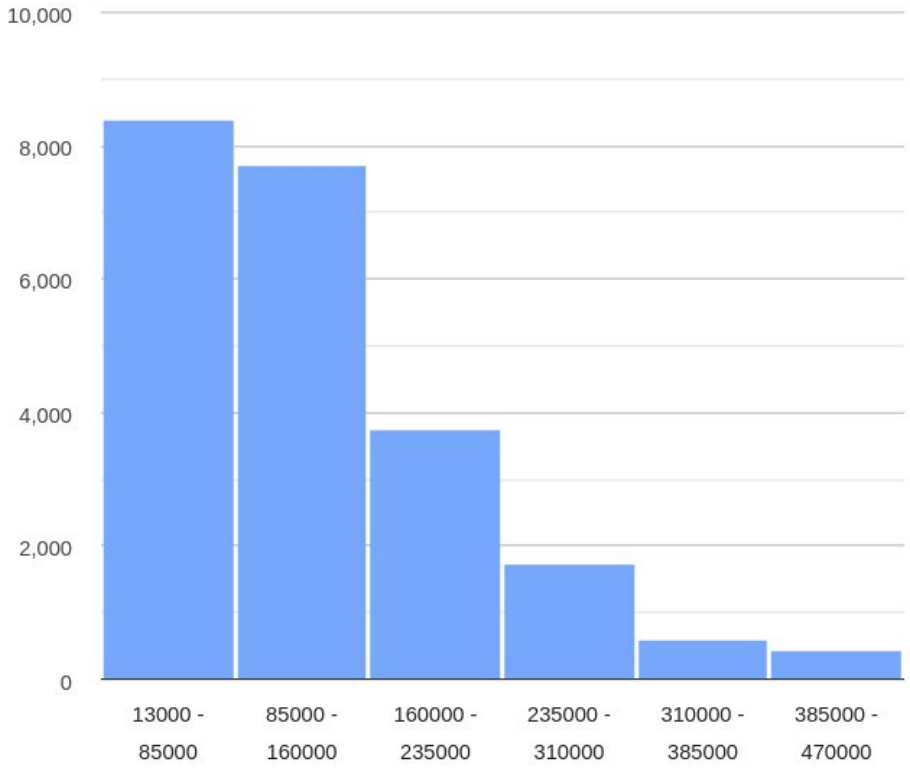

Рисунок 2 — Гистограмма распределения зарплаты

2.3.2. Ключевые навыки

В процессе обработки 44787 вакансий выяснено, что около 99.98% вакансий имеют в своём описании ключевые навыки, что равно 44733 вакансиям. Т.е. практически все работодатели и составители вакансий добавляют ключевые навыки, что говорит об эффективности их добавления при поиске персонала. При анализе этих вакансий получен список из 211180 навыков, 9093 из которых являются уникальными, и 105 из которых имеют процент использования больше 1%, а остальные, в рамках исследования, будут считаться незначительными. Помимо этого, это исключает анализ большого числа навыков с ошибками в названии.

Из этих 105 навыков выбраны первые 30 и по ним составлена таблица 2.8 (см Приложение А).

Далее идёт список список этих навыков с группированием по категориям, описанием, и выводами

Общие технологии:

- SQL — инструмент, предназначенный для организации, управления, выборки и обработки информации, содержащейся в базе данных [7, с. 30];
- Git — система контроля версий и необходимый инструмент для командной разработки [8, с. 13];
- Linux — семейство операционных систем, используемых в качестве серверов;
- Объектно-ориентированное программирование — преобладающий подход написания ПО.

Языки программирования:

- JavaScript (HTML, CSS) — основной стек технологий для веб-разработки;

- Java — основная технология практически для всех типов сетевых приложений и всеобщий стандарт разработки и распространения встроенных и мобильных приложений, игр, веб-контента и корпоративного программного обеспечения;
- Python — высокоуровневый язык программирования общего назначения. Этот язык является первым и наиболее популярным для преподавания в высших учебных заведениях США;
- PHP — распространённый язык программирования общего назначения;
- C# — язык программирования общего назначения платформы .NET, поддерживаемый Microsoft;
- C++ — изначально, надмножество языка C, поддерживающее парадигму ООП, а теперь — самый язык программирования после C;
- 1C — встроенный язык программирования комплекса ПО российской компании 1C.

Базы данных (в порядке убывания популярности):

- PostgreSQL;
- MySQL;
- MS SQL.

Общие навыки:

- Английский язык — знание английского языка входит в 10 самых востребованных навыков на рынке IT, т.к. разработчику постоянно требуется пользоваться технической документацией, часто написанной только на английском языке.
- Работа в команде — разработка программного обеспечения подразумевает работу в команде и разделение ответственности над отдельными модулями программы, а конфликты в коллективе могут намного замедлить процесс разработки, чего требуется избегать;
- Управление проектами;

- Разработка технических заданий;
- Бизнес-анализ;
- Грамотная речь;
- Ведение переговоров;

Помимо этого, в список навыков входит платформа контроля управления и разработки проектов “Atlassian Jira”

Библиотеки и фреймворки:

- Spring Framework — универсальный фреймворк Java для серверной разработки;
- .NET Framework — общеязыковая платформа Microsoft, использующая в качестве исполнения общий язык (своего рода байткод) Common Language Runtime, в который могут компилироваться языки C#, Visual Basic Managed C++, F#;
- jQuery — JavaScript-библиотека для упрощения манипуляций объектами DOM на веб-страницах HTML;
- React — JavaScript-библиотека для создания пользовательских интерфейсов.

В конце списка из 30 ключевых навыков распологаются навыки разработки для Android и IOS

2.3.2. Заработная плата

Среди полученных 44787 вакансий только 22982 имеют указанную заработную плату, что составляет 51.31% от всех вакансий.

Есть много причин, по которым работодатель не указывает конкретные цифры заработной платы, и, предполагается, самая частая из них — конкуренция за специалиста.

Обычно работодатель может заинтересовать новый персонал либо высокой зарплатой, либо хорошими условиями работы, либо интересными задачами. Как можно понять из анализа полученных данных (см Приложение А, таблицы 2.1-2.6), помимо высокой зарплаты, работодатель в большей половине случаев пользуется и тактикой поиска при помощи скрытой, часто средней или выше по рынку зарплаты, тем самым интересуя специалиста не только доходом, но и самой работой, условиями.

Помимо этого можно выделить и такой показатель, как “открытость” рынка труда. Так, в одних регионах процент вакансий с указанной зарплатой может быть высоким, т.е. средний работодатель меньше пробует заинтересовать специалиста только условиями, указывая конкретную цену. Другие же регионы могут использовать только условия. Так, при анализе данных (см Приложение А, таблицы 2.1-2.6), можно увидеть, что в России данный показатель составляет 51%, в Москве (не учитывая Московскую Область) и Украине процент опускается до 30, достигая 15-20% в Беларуси. Таким образом, анализируя количество вакансий, можно предположить, что в регионах с низким показателем указанных выше соответственно и нужда в мотивированных и качественных специалистах.

2.3.3. Сравнение рынков России, Украины и Беларуси

Почему так происходит? В первую очередь стоит посмотреть на устройство данной сферы рынка труда в двух последних странах. Если проанализировать поля ключевых навыков этих двух стран, в сравнении с Россией или регионами Российской Федерации, они прежде всего отличаются значительным превалированием навыка “английский язык”. В России и её регионах процент упоминания данного навыка составляет порядка 5%, а в Украине и Беларуси он превалирует со значением 15-20%.

Из этого можно сделать вывод, что в большинстве своем, украинский и белорусский рынки прежде всего направлены на работу с иностранными заказчиками, или при их крупном сотрудничестве и поддержке.

Полученные выводы, однако, нельзя считать исключительно верными, т.к. услуги по поиску вакансий в Украине и Беларуси предоставляет не исключительно компания HeadHunter, однако, по информации данного ресурса, эти выводы правдивы, или имеют объективную оценку.

2.3.4. Опыт работы

Желательный/обязательный опыт работы также различается для всех регионов. В Республике Татарстан, и в среднем по России, как и в других странах (Украина, Беларусь), значение категории “1-3 года” выше категории “3-6 лет” на 15-30% , когда в Санкт-Петербурге и Москве эти категории отличаются всего на 5% (см. Приложение А, таблицы 1.1-1.6). Это явно может говорить о молодости индустрии, меньшей нужде в более опытных специалистах. Таким образом крупные центры аккумулируют в себе не только в среднем больше людей, но и привлекают более опытных специалистов, не нашедших подобный спрос на свой опыт вне этих центров.

Заключение

В ходе выполнения данной работы было разработано кроссплатформенное программное обеспечение для анализа базы данных вакансий сайта HeadHunter в сфере информационных технологий.

Были проанализированы около 43-х тысяч вакансий в России, 2.2 тысячи на Украине, 1.7 тысячи в Белоруссии, на основе чего выведены списки навыков, и статистическая информация по кластерам заработной платы, опыта работы, сферы деятельности, графика работы, и т.д..

На основе анализа 44787 вакансии (около 80% от всех вакансий на рынке IT), был выделен следующий список навыков, требуемых работодателям от соискателей на рынке IT (в скобках указано количество вакансий с упоминанием данных навыков):

- SQL (5981)
- Git (5870)
- JavaScript (4958)
- Java (3360)
- Linux (3231)
- Английский язык (3130)
- Python (2911)
- PostgreSQL (2824)
- HTML (2798)
- 1C (2714)

Все эти навыки, по результату исследования, являются основными и наиболее предпочтительными для изучения с целью устроиться на работу в секторе IT. Таким образом, для удовлетворения спроса настоящего рынка труда, именно на эти навыки, помимо общей теоретической подготовки, должны быть нацелены образовательные программы обучения специалистов.

Список используемых источников

1. Дэн Вандеркам. Эффективный TypeScript: 62 способа улучшить код. — СПб.: Питер, 2020. — 288 с.: ил. — (Серия «Бестселлеры O’Reilly»).
2. Пауэрс Ш. Изучаем Node. Переходим на сторону сервера. 2-е изд., доп. и перераб. — СПб.: Питер, 2017. — 304 с.: ил. — (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02941-4
3. Документация API HeadHunter [Электронный ресурс]: https://github.com/hhru/api
4. Commander [Электронный ресурс]: Режим доступа: https://github.com/tj/commander.js
5. node-fetch [Электронный ресурс]: Режим доступа: https://github.com/node-fetch/node-fetch
6. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппелъ, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. —М.: ООО "И.Д. Вильяме", 2015. — 960 с. : ил. — Парал. тит. англ. ISBN 978-5-8459-1654-9 (рус.)
7. Москалева Ю.П., Сейдаметова З.С. Программирование: введение в Git: учебное пособие. — Симферополь: ГАУ РК «Медиацентр им. И. Гаспринского», 2018. – 114 с. ISBN 978-5-906959-45-4
8. АТАЯН И.М. Биржи труда в 20-е годы: опыт государственного трудового посредничества с. 117-121 // Социологические исследования. 2000. №4.

Приложение А — Результаты Исследования

Таблица 2.1 - Статистика по России 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	22547	40
Ниже 85 тыс. руб	8387	19
От 85 тыс. руб.	14160	32
От 160 тыс. руб.	6462	15
От 235 тыс. руб.	2733	6
От 310 тыс. руб.	999	2
От 385 тыс. руб.	417	1
Требование по стажу		
Стаж		
1 год	4003	10
1-3 года	18807	49
3-6 лет	13800	36
Более 6 лет	1647	4
Рабочий график		
График		
Полный день	28793	75
Удалённая работа	6310	16
Гибкий график	2656	7
Сменный график	420	1
Вахтовый метод	78	<1
Всего:	55945	100
Ключевые навыки (10 основных)		
Навык		
SQL	5874	13.3
Git	5795	13.2
JavaScript	4848	11
Java	3300	7.5
Linux	3152	7.2
Английский язык	3069	7
Python	2910	6.6
PostgreSQL	2741	6.2
HTML	2733	6.2
1C	2700	6.2

Таблица 2.2 - Статистика по Белоруссии 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	601	36
Ниже 70 тыс. руб	198	12
От 70 тыс. руб.	160	25
От 130 тыс. руб.	109	15
От 195 тыс. руб.	58	8
От 260 тыс. руб.	46	5
От 320 тыс. руб.	30	2
Требование по стажу		
Стаж		
1 год	366	10
1-3 года	1676	49
3-6 лет	1209	35
Более 6 лет	127	3
Рабочий график		
График		
Полный день	2348	69
Удалённая работа	557	16
Гибкий график	457	13
Сменный график	15	0.4
Вахтовый метод	1	~0
Всего:	3378	100
Ключевые навыки (10 основных)		
Навык		
Английский язык	341	21
JavaScript	279	17.2
Git	247	15.2
Java	170	10.5
CSS	153	9.4
HTML	150	9.2
SQL	144	8.9
C#	118	7.3
Python	108	6.7
ООП	107	6.6

Таблица 2.3 - Статистика по Украине 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	386	17
Ниже 95 тыс. руб	114	5
От 95 тыс. руб.	272	13
От 180 тыс. руб.	179	8
От 265 тыс. руб.	80	4
От 350 тыс. руб.	36	2
От 440 тыс. руб.	13	1
Требование по стажу		
Стаж		
1 год	270	12
1-3 года	1020	47
3-6 лет	834	38
Более 6 лет	52	2
Рабочий график		
График		
Полный день	1434	69
Удалённая работа	495	23
Гибкий график	244	11
Сменный график	3	~0
Вахтовый метод	0	0
Всего:	2176	100
Ключевые навыки (10 основных)		
Навык		
Английский язык	521	21.1
JavaScript	279	12.9
Git	279	12.9
SQL	251	11.6
Java	243	11.2
Project Management	191	8.8
HTML	189	8.7
CSS	177	8.2
Python	173	8
C#	151	7

Таблица 2.4 - Статистика по Москве 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	6714	37
Ниже 90 тыс. руб	957	5
От 90 тыс. руб.	5757	32
От 170 тыс. руб.	3171	18
От 245 тыс. руб.	1507	8
От 325 тыс. руб.	558	3
От 400 тыс. руб.	198	1
Требование по стажу		
Стаж		
1 год	1645	7
1-3 года	9442	43
3-6 лет	9225	42
Более 6 лет	1185	5
Рабочий график		
График		
Полный день	15900	73
Удалённая работа	9442	21
Гибкий график	985	4
Сменный график	81	0.3
Вахтовый метод	7	~0
Всего:	21497	100
Ключевые навыки (10 основных)		
Навык		
SQL	2693	15.2
Git	2103	11.9
JavaScript	1551	8.7
Linux	1470	8.3
Python	1462	8.2
Java	1443	8.1
Английский язык	1399	7.9
PostgreSQL	1264	7.1
MS SQL	995	5.6
Atlassian Jira	933	5.3

Таблица 2.5 - Статистика по Санкт-Петербургу 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	6714	37
Ниже 85 тыс. руб	821	20
От 85 тыс. руб.	1207	61
От 160тыс. руб.	600	32
От 230 тыс. руб.	422	16
От 305 тыс. руб.	128	5
От 380 тыс. руб.	89	2
Требование по стажу		
Стаж		
1 год	668	7
1-3 года	3969	45
3-6 лет	3670	44
Более 6 лет	377	9
Рабочий график		
График		
Полный день	5986	68
Удалённая работа	1699	19
Гибкий график	923	10
Сменный график	73	0.8
Вахтовый метод	3	~0
Всего:	8684	100
Ключевые навыки (10 основных)		
Навык		
Git	565	14.1
JavaScript	544	13.6
SQL	450	11.2
Linux	433	10.8
Java	333	8.3
CSS	300	7.5
Английский язык	290	7.2
Python	284	7.1
HTML	282	7
PostgreSQL	271	6.8

Таблица 2.6 - Статистика по Республике Татарстан 2021

Название категории	Количество вакансий	Соотношение к вакансиям (%)
Заработная плата		
ЗП указана	825	44
Ниже 85 тыс. руб	292	16
От 85 тыс. руб.	533	28
От 160тыс. руб.	284	15
От 230 тыс. руб.	126	7
От 305 тыс. руб.	72	4
От 380 тыс. руб.	32	2
Требование по стажу		
Стаж		
1 год	260	14
1-3 года	1018	54
3-6 лет	554	30
Более 6 лет	40	2
Рабочий график		
График		
Полный день	1190	64
Удалённая работа	515	28
Гибкий график	110	6
Сменный график	51	0.3
Вахтовый метод	6	~0
Всего:	1872	100
Ключевые навыки (10 основных)		
Навык		
Git	286	15.3
SQL	240	12.8
JavaScript	225	12
Английский язык	172	9.2
HTML	142	7.6
Linux	141	7.5
PostgreSQL	138	7.4
Java	137	7.3
CSS	120	6.4
MS SQL	113	6

Таблица 2.8 - Ключевые навыки по России

№	Название навыка	Количество вакансий (ед.)
1	SQL	5874
2	Git	5795
3	JavaScript	4848
4	Java	3300
5	Linux	3152
6	Английский язык	3069
7	Python	2910
8	PostgreSQL	2741
9	HTML	2733
10	1C	2700
11	CSS	2591
12	ООП	2521
13	MySQL	2438
14	PHP	2368
15	MS SQL	2331
16	C#	2116
17	Работа в команде	1960
18	Управление проектами	1812
19	Atlassian Jira	1713
20	Разработка технических заданий	1673
21	C++	1586
22	Spring Framework	1229
23	jQuery	1092
24	React	1068
25	Бизнес-анализ	1055
26	Грамотная речь	1037
27	Android разработка	1013
28	Ведение переговоров	932
29	IOS разработка	925
30	.NET Framework	889

Приложение Б — Код приложения. CLI.

Листинг 1.1. Модуль CLI

```
import { Command } from "commander";
import { getArea, getFromLog } from "./utils";
import { API } from "./types/api/module";
import { getFull, search, prepare } from "./core/index.js";
import { analyze } from "./core/analyze";

const getCLI = () => {
  const cli = new Command();

  cli.name("node-hh-parser").version("1.0.0");

  // инициализация полей (опций)
  cli
    .option(
      "-A, --all",
      "выполнить все остальные комманды автоматически",
      "Россия"
    )
    .option(
      "-a, --area <area-name>",
      "название территории поиска или индекс",
      "Россия"
    )
    .option("-S, --silent", "не выводить информацию в консоль");

  // инициализация команд (операций)
  cli
    .command("search <text>")
    .description("поиск вакансий по полю text")
    .action(async (text: string) => {
      const area = await getArea(cli.opts().area);

      const raw_query: API.Query = {
        text: text,
        area: area,
        clusters: true,
      };

      const data = search({ ...raw_query, text, area });

      if (cli.opts().all) {
        await data
          .then((vacancies) => getFull(vacancies))
          .then((full_vacancies) => prepare(full_vacancies))
          .then((prepared_vacancies) => {
            const clusters: API.FormattedClusters = getFromLog(
              "data",
```

```
            "clusters.json"
          );

          return analyze(prepared_vacancies, clusters);
        });
    }
  });

 cli
  .command("get-full")
  .description("получает полное представление вакансий")
  .action(() => {
    const vacancies: API.Vacancy[] = getFromLog("data", "vacancies.json");

    getFull(vacancies);
  });

 cli
  .command("analyze")
  .description("проанализировать полученные данные")
  .action(() => {
    const prepared_vacancies: API.FullVacancy[] = getFromLog(
      "data",
      "prepared_vacancies.json"
    );
    const clusters: API.FormattedClusters = getFromLog(
      "data",
      "clusters.json"
    );

    analyze(prepared_vacancies, clusters);
  });

 return cli;
};

export default getCLI;
```

Листинг 1.2. CLI — запуск CLI

```
import getCLI from "./cli";

const cli = getCLI();

cli.parse(process.argv);
```

Приложение В — Код приложения. Ядро.

Листинг 3.3. — Функции и методы модуля ядра

```
import { API } from "../types/api/module.js";
import { buildQueryURL, formatClusters, saveToFile } from "../utils";
import {
  getFullVacancies,
  getURLsFromClusters,
  getVacancies,
  getVacanciesInfo,
} from "./requests.js";

export const search = async (query: API.Query) => {
  const query_url = buildQueryURL({
    ...query,
    per_page: 0,
    page: 0,
  });

  const response: API.Response = await getVacanciesInfo(query_url);
  console.log("всего по данному запросу найдено:", response.found, "вакансий");

  const clusters: API.FormattedClusters = formatClusters(response.clusters);
  saveToFile(clusters, "data", "clusters.json");

  console.log("парсинг сокращённых вакансий");

  const urls = await getURLsFromClusters(clusters);
  console.log(
    "количество запросов для получения сокращённых вакансий:",
    urls.length
  );

  const vacancies: API.Vacancy[] = await getVacancies(urls);

  console.log(vacancies.length);
  saveToFile(vacancies, "data", "vacancies.json");

  return vacancies;
};

export const getFull = async (vacancies: API.Vacancy[]) => {
  console.log("парсинг полных вакансий");

  const full_vacancies_urls = vacancies
    .filter((vacancy) => vacancy?.url != undefined)
    .map((vacancy) => {
      try {
        return vacancy.url;
```

```
    } catch (error) {
      console.log(vacancy);
    }
    return vacancy.url;
  });

 const full_vacancies = await getFullVacancies(full_vacancies_urls);

 console.log("получено полных вакансий:", full_vacancies.length);

 return full_vacancies;
};

export const prepare = async (full_vacancies: API.FullVacancy[]) => {
 // нам важны поля key_skills
 const prepared_vacancies: API.PreparedVacancy[] = full_vacancies.map(
  (vac: API.FullVacancy) => {
   return {
    key_skills: vac.key_skills,
    response_letter_required: vac.response_letter_required,
    has_test: vac.has_test,
    test: vac.test,
    salary: vac.salary,
    allow_messages: vac.allow_messages,
    accept_incomplete_resumes: vac.accept_incomplete_resumes,
    accept_temporary: vac.accept_temporary,
   };
  }
 );

 saveToFile(prepared_vacancies, "data", "prepared_vacancies.json");

 return prepared_vacancies;
};
```

Листинг 3.3.1.А — Запросы.

```
import { Spinner } from "cli-spinner";
import { chunk, partition } from "lodash-es";
import fetch from "node-fetch";
import { API } from "../types/api/module";
import { fetchCache, formatClusters, paginateClusters } from "../utils";

const hh_headers = {
 "User-Agent": "labor-market-analyzer (vadim.kuz02@gmail.com)",
};

export const getVacanciesInfo = async (url: string): Promise<API.Response> => {
 const data: API.Response = await fetch(
  url.match(/[_\.!~*'()-]/) && url.match(/%[0-9a-f]{2}/i)
```

```
    ? url
    : encodeURI(url),
   {
    headers: hh_headers,
   }
  ).then((res) => res.json());

  return data;
};

export const getVacancies = async (urls: string[]) => {
  const chunk_size = 50;
  const chunked_urls = chunk(urls, chunk_size);

  console.log("количество чанков:", chunked_urls.length);
  console.log("размер чанка:", chunk_size);

  const vacancies: API.Vacancy[] = [];

  const spinner = new Spinner("подготовка... %s");
  spinner.setSpinnerString("|/-\\");
  spinner.start();

  let i = 1;
  for (const chunk of chunked_urls) {
    spinner.setSpinnerTitle(`${i}/${chunked_urls.length} %s`);
    vacancies.push(...(await getVacanciesFromURLs(chunk)));
    i++;
  }
  console.log("");
  spinner.stop();

  return vacancies;
};

export const getFullVacancies = async (
  urls: string[]
): Promise<API.FullVacancy[]> => {
  const chunked_urls = chunk(urls, 100);
  const full_vacancies: API.FullVacancy[] = [];

  console.log("количество чанков:", chunked_urls.length);

  let i = 1;
  for (const chunk of chunked_urls) {
    process.stdout.write(`${i}/${chunked_urls.length}\r`);
    i++;
    full_vacancies.push(...(await getFullVacanciesFromURLs(chunk)));
  }
```

```
  return full_vacancies;
};

export const getURLsFromClusters = async (clusters: API.FormattedClusters) => {
  const [small_area_clusters, big_area_clusters] = partition(
    clusters?.area?.items ?? clusters.metro?.items,
    (cluster) => cluster.count <= 2000
  );

  const branched_big_cluster = await branchVacanciesFromDeepCluster(
    big_area_clusters
  );

  const paginated_urls_from_big_clusters = paginateClusters(
    branched_big_cluster
  );

  const paginated_urls_from_small_clusters = paginateClusters(
    small_area_clusters
  );

  return [
    ...paginated_urls_from_big_clusters,
    ...paginated_urls_from_small_clusters,
  ];
};

export const branchVacanciesFromDeepCluster = async (
  cluster_items: API.ClusterItem[]
): Promise<API.ParseItem[]> => {
  const urls = cluster_items.map((item) => item.url);

  const clusters: Promise<API.Cluster[]>[] = urls.map((url) =>
    fetch(url, {
      headers: hh_headers,
    })
      .then((res) => res.json() as Promise<API.Response>)
      .then((res) => res.clusters)
  );

  const parse_items = ([] as API.ParseItem[]).concat(
    ...(await Promise.all(clusters)).map((clusters) => {
      const formatted_clusters = formatClusters(clusters);
      const urls: any[] = [];
      if (formatted_clusters.metro !== undefined) {
        formatted_clusters.metro.items.forEach(async (item) => {
          urls.push({
            count: item.count,
```

```
          url: item.url,
          name: item.metro_line?.area.name + " " + item.name,
        });
      });
    }
    return urls;
  })
 );

 return parse_items;
};

const branchMetroCluster = async (item: API.MetroClusterItem) => {
 const stations = await fetch(item.url, {
  headers: hh_headers,
 })
  .then((res) => res.json() as Promise<API.Response>)
  .then((res) => formatClusters(res.clusters))
  .then((clusters) => clusters.metro?.items ?? []);

 return stations;
};

export const getVacanciesFromURLs = async (
 urls: string[]
): Promise<API.Vacancy[]> => {
 const data: Promise<API.Response>[] = urls.map((url) =>
  fetch(url, {
   headers: hh_headers,
  }).then((res) => res.json())
 );

 // дождаться резолва промисов, получить их поля items
 const vacancies: API.Vacancy[] = ([] as API.Vacancy[]).concat(
  ...(await Promise.all(data)).map((page) => {
   return page.items;
  })
 );

 return vacancies;
};

export const getFullVacanciesFromURLs = async (
 urls: string[]
): Promise<API.FullVacancy[]> => {
 const data: Promise<API.FullVacancy>[] = urls.map((url) =>
  fetchCache(url, { headers: hh_headers })
 );
```

```
  const full_vacancies: API.FullVacancy[] = await Promise.all(data);

  return full_vacancies;
};
```

Листинг 3.3.1.F — Кэширование запросов

```
import { existsSync, mkdirSync } from "fs";
import fetch, { RequestInit } from "node-fetch";
import { addToCache, getFromCache, resolvePathForCache } from "./cache.js";

export const fetchCache = async (
  url: string,
  init?: RequestInit
): Promise<any> => {
  const cache_dir_path = "./data/cache";
  if (!existsSync(cache_dir_path)) {
    mkdirSync(cache_dir_path, { recursive: true });
  }

  const cache_file_path: string = resolvePathForCache(url, cache_dir_path);

  if (existsSync(cache_file_path)) {
    // read from cache
    const data: any = getFromCache(cache_file_path);

    if (data === undefined) {
      const data: any = await fetch(url, init).then((res) => res.json());

      // add data to cache
      addToCache(cache_file_path, data);

      return data;
    }

    return data;
  } else {
    // make new fetch and get json
    const data: any = await fetch(url, init).then((res) => res.json());

    // add data to cache
    addToCache(cache_file_path, data);

    return data;
  }
};
```