

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**АНАЛИЗ ВАКАНСИЙ САЙТА ПОИСКА
РАБОТЫ**

Студент

Кирилин Матвей Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель

Слепцов Н.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой

удовлетворительно
(протокол заседания ГЭК от

15 июня 2026 № 10)

Секретарь ГЭК

Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кирилин Матвей Сергеевич гр. 22BBB1 факультета вычислительной техники.

2. Тема работы: «Анализ вакансий сайта поиска работы»

Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Слепцов Н.В.

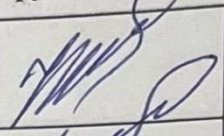
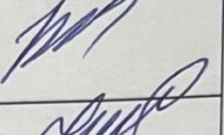
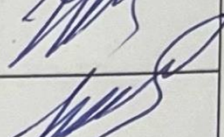
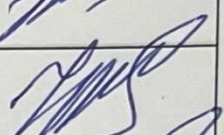
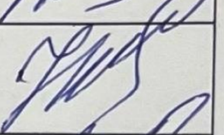
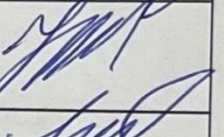
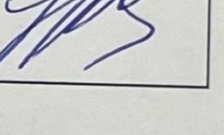
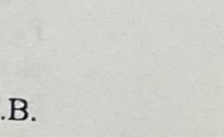
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать десктопное приложение для автоматизированного сбора, хранения и анализа вакансий с портала HH.ru. Приложение должно выполнять запросы к официальному API HH.ru с поддержкой Bearer-авторизации, автоматически определять код региона по названию города, сохранять полученные данные в базу данных SQLite, рассчитывать аналитические показатели (средняя зарплата, топ востребованных навыков), строить графики распределения зарплат и предоставлять пользователю графический интерфейс с демо-режимом для работы без подключения к интернету.

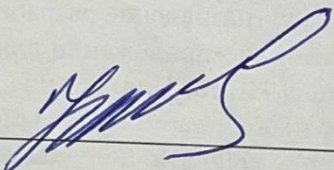
5. Перечень подлежащих разработке вопросов

Провести анализ существующих решений в области агрегации и анализа вакансий и обосновать актуальность разработки. Определить требования к приложению и спроектировать многоуровневую архитектуру с разделением ответственности. Выполнить выбор языка программирования, библиотек и среды разработки. Разработать модуль парсинга на основе официального API HH.ru с Bearer-авторизацией. Разработать модули очистки данных, хранения (SQLite и CSV), аналитики и визуализации. Реализовать сервисный слой и графический интерфейс пользователя на tkinter. Провести модульное и функциональное тестирование разработанного приложения и оценить его работоспособность.

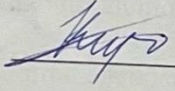
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 11.04.2026	
2	Анализ аналогов, актуальности, постановка задачи	24	11.04.2026 – 14.04.2026	
3	Выбор и настройка средств разработки	4	14.04.2026 – 15.04.2026	
4	Проектирование архитектуры приложения	30	15.04.2026 – 19.04.2026	
5	Разработка модулей парсинга, очистки и хранения данных	100	19.04.2026 – 04.05.2026	
6	Разработка модулей аналитики, визуализации и GUI	55	04.05.2026 – 11.05.2026	
7	Тестирование и оценка работоспособности	54	11.05.2026 – 18.05.2026	
8	Оформление пояснительной записки	25	18.05.2026 – 21.05.2026	

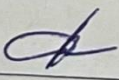
Дата выдачи задания " 07 " апреля 2026г.

Руководитель бакалаврской работы  Слепцов Н.В.

(подпись, дата)

Задание к исполнению принял студент  Кирилин М.С.

(подпись, дата)

Работу к защите допустить
Декан факультета  10.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 96 листов, 13 рисунков, 5 таблиц, 11 источников, 3 приложения.

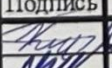
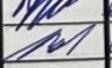
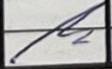
JOB-ПОРТАЛ, СБОР ВАКАНСИЙ, АНАЛИЗ РЫНКА ТРУДА, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС, БАЗА ДАННЫХ, ВИЗУАЛИЗАЦИЯ ДАННЫХ, REST API

Цель работы заключается в разработке десктопного приложения для автоматизированного сбора, хранения и анализа вакансий с портала HH.ru посредством официального REST API.

Объектом исследования являются процессы получения структурированных данных о вакансиях через API HH.ru, их обработки, хранения в реляционной базе данных и визуализации аналитических показателей рынка труда.

В работе применены методы многоуровневой архитектуры с разделением ответственности между слоями, Bearer-авторизации при взаимодействии с REST API, автоматического определения кода региона, обработки табличных данных средствами библиотеки pandas, а также методы модульного и функционального тестирования с использованием pytest.

В работе проведён анализ существующих решений в области HR-аналитики и выявлены их ограничения, разработано приложение Job Analyzer с поиском вакансий по заданным критериям, расчётом средней зарплаты и топ востребованных навыков, построением scatter-графика, сохранением данных в SQLite и CSV, демо-режимом для работы без интернета и покрытием тестами из 21 тест-кейса.

					ПГУ 1.090301.05.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Анализ вакансий сайта поиска работы Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кирилин М.С.		05.06.20			2	96
Провер.		Слепцов Н. В.		05.06.20				
Н. Контр.		Бычков А.С.		05.06.20				
Утверд.		Митрохин М.А.		05.06.20				
						ФВТ, каф. ВТ гр. 22BBB1		

Оглавление

Перечень сокращений и обозначений	5
Введение.....	7
1 Анализ предметной области	10
1.1 Анализ решения поставленной задачи	10
1.2 Анализ актуальности разработки	11
1.3 Анализ аналогичных решений	11
1.4 Постановка задачи	15
1.5 Описание архитектуры проекта	16
2 Выбор инструментов разработки	18
2.1 Выбор языка программирования.....	18
2.2 Выбор библиотек	18
2.3 Выбор алгоритма определения города.....	20
2.4 Выбор СУБД.....	20
2.5 Выбор среды разработки.....	21
3 Реализация	23
3.1 Структура проекта	23
3.2 Реализация модели данных.....	23
3.3 Реализация модуля парсинга	24
3.3.1 Реализация NNParser	24
3.3.2 Реализация DemoParser	25
3.4 Реализация модуля очистки данных	26
3.5 Реализация модуля хранения данных	26
3.6 Реализация аналитического модуля	26
3.7 Реализация модуля визуализации.....	26
3.8 Реализация сервисного слоя	27
3.9 Установка и запуск.....	28
4 Руководство пользователя	30
4.1 Требования к системе	30
4.2 Первый запуск – установка и запуск	30
4.3 Описание полей формы поиска	31

4.4 Поиск вакансий	31
4.5 Работа в демо – режиме.....	32
4.6 Просмотр графика распределения зарплат.....	35
4.7 Доступ к сохранённым данным.....	35
5 Тестирование системы.....	36
5.1 Типичные ошибки и пути их решения.....	37
5.2 Рекомендации к дальнейшей разработке	39
6 Безопасность жизнедеятельности	40
6.1 Правила безопасности при использовании ПК	40
6.2 Пожарная безопасность.....	40
Заключение	42
Список использованных источников	43
Приложение А – UML-диаграммы	44
Приложение Б – Листинг фрагментов программы	48
Приложение В – Презентация.....	92

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения.

БД – база данных

СУБД – система управления базами данных

ABC (Abstract Base Class) – абстрактный базовый класс; механизм Python для определения обязательных интерфейсов

API (Application Programming Interface) – программный интерфейс взаимодействия компонентов системы

Bearer-токен – тип HTTP-авторизации, при котором токен доступа передаётся в заголовке Authorization: Bearer {token}

CSV (Comma-Separated Values) – текстовый формат хранения табличных данных.

DataFrame – двумерная табличная структура данных библиотеки pandas

DPI (Dots Per Inch) – количество точек на дюйм; единица измерения разрешения изображения

GET – метод HTTP-протокола для запроса данных от указанного ресурса

GUI (Graphical User Interface) – графический интерфейс пользователя

HH.ru – российский интернет-сервис по поиску работы и подбору персонала

HR (Human Resources) – управление персоналом; область найма и развития сотрудников

HTTP (HyperText Transfer Protocol) – протокол передачи данных в сети Интернет

IDE (Integrated Development Environment) – интегрированная среда разработки

JSON (JavaScript Object Notation) – лёгкий текстовый формат обмена данными

matplotlib – библиотека Python для построения графиков и визуализации данных

NaN (Not a Number) – специальное значение для обозначения отсутствующих данных в pandas

OOP (Object-Oriented Programming) – объектно-ориентированное программирование

pandas – библиотека Python для обработки и анализа структурированных данных

PNG (Portable Network Graphics) – формат хранения растровых изображений

PyCharm – интегрированная среда разработки от JetBrains для языка Python

pytest – фреймворк для написания и запуска автоматических тестов на Python

requests – библиотека Python для выполнения HTTP-запросов

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения

SQL (Structured Query Language) – язык структурированных запросов для управления БД

SQLite – встраиваемая реляционная СУБД, хранящая всю базу данных в одном файле

tkinter – стандартная библиотека Python для создания графических интерфейсов

UML (Unified Modeling Language) – унифицированный язык моделирования программных систем

URL (Uniform Resource Locator) – унифицированный указатель ресурса в сети Интернет

					ПГУ 1.090301.05.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире цифровой трансформации рынок труда претерпевает существенные изменения. Ежедневно на крупнейших российских job-порталах публикуются тысячи новых вакансий в самых разных регионах и отраслях. Ручная обработка такого объёма данных требует значительных временных и человеческих ресурсов и неизбежно сопровождается ошибками и неполнотой анализа.

Портал HH.ru занимает лидирующие позиции на российском рынке интернет-рекрутинга. По данным компании, на платформе зарегистрировано более 50 миллионов резюме, ежедневно публикуются сотни тысяч вакансий. Принципиально важно, что HH.ru предоставляет разработчикам публичный REST API, позволяющий получать данные о вакансиях в структурированном формате JSON [1]. Это открывает широкие возможности для создания аналитических приложений.

Анализ существующих инструментов для работы с данными о вакансиях выявил ряд существенных ограничений. Встроенные аналитические инструменты HH.ru доступны только в платных тарифах. Сторонние сервисы HR-аналитики имеют закрытый код, ограниченные возможности кастомизации и не позволяют экспортировать сырые данные. Инструменты академического уровня требуют специальных знаний и не предоставляют удобного GUI для конечного пользователя.

В данной выпускной квалификационной работе разрабатывается кросс-платформенное десктопное приложение Job Analyzer, которое решает перечисленные проблемы: предоставляет удобный графический интерфейс, использует официальный API HH.ru, сохраняет данные локально, выполняет аналитические расчёты и строит визуализации – всё в одном бесплатном инструменте.

Объектом исследования является рынок труда специалистов, представленный на портале HH.ru.

Предметом исследования являются методы и программные средства автоматизированного сбора, хранения, обработки и визуализации данных о вакансиях с использованием официального API портала HH.ru.

Целью данной работы является проектирование и разработка десктопного приложения для автоматизированного сбора вакансий с портала HH.ru посредством официального API, их локального хранения, статистического анализа и визуализации.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области, изучить рынок HR-аналитики и существующие решения для агрегации вакансий;
- изучить официальный API HH.ru, определить используемые эндпоинты и параметры запросов;
- спроектировать многоуровневую архитектуру с применением принципов разделения ответственности;
- реализовать модуль парсинга на основе официального API HH.ru с автоматическим определением кода города;
- реализовать демо-режим с тестовыми данными для работы без подключения к интернету;
- разработать модуль очистки и валидации данных;
- реализовать модуль хранения данных с поддержкой форматов SQLite и CSV;
- разработать аналитический модуль для расчёта средней зарплаты и топ востребованных навыков;
- реализовать модуль визуализации с построением scatter+line графика распределения зарплат;
- разработать сервисный слой для оркестрации всех компонентов в единый pipeline;
- разработать графический интерфейс пользователя на базе tkinter;
- реализовать модульные тесты для ключевых компонентов;

– провести тестирование и оценку разработанного решения.

Практическая значимость работы состоит в создании готового к применению инструмента для аналитики рынка труда, позволяющего за считанные секунды получить актуальную выборку вакансий, рассчитать среднюю зарплату и определить наиболее востребованные навыки.

					ПГУ 1.090301.05.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

Для разработки качественного программного решения, обеспечивающего сбор и анализ вакансий, необходимо детально изучить предметную область, охватывающую принципы агрегации данных, архитектуру современных job-порталов и существующие подходы к HR-аналитике.

Рынок труда является одним из ключевых индикаторов экономической активности. HR-аналитика позволяет работодателям принимать обоснованные решения об уровне вознаграждения, соискателям – формировать адекватные зарплатные ожидания, а образовательным организациям – корректировать учебные программы в соответствии с требованиями рынка.

Портал HH.ru занимает доминирующее положение на российском рынке job-порталов. Его доля среди аналогичных сервисов превышает 60%. API HH.ru реализован по архитектуре REST и задокументирован по стандарту OpenAPI 3.0. Ключевым эндпоинтом для поиска вакансий является GET <https://api.hh.ru/vacancies>. Помимо него, для автоматического определения кода региона по текстовому названию города используется эндпоинт GET <https://api.hh.ru/suggests/areas>.

1.1 Анализ решения поставленной задачи

В данной работе предлагается реализовать многоуровневое приложение для автоматизированного сбора и анализа вакансий. Все операции получения данных выполняются через официальный API HH.ru – это обеспечивает стабильность в долгосрочной перспективе в отличие от HTML-скрапинга.

Система реализует полный pipeline обработки данных: запрос к API с фильтрами → валидация и очистка → сохранение в SQLite → аналитические расчёты → визуализация. Ключевым параметром запроса является `only_with_salary=true`, который фильтрует вакансии на стороне сервера HH.ru и гарантирует наличие зарплатных данных. Код региона определяется автоматически через эндпоинт `/suggests/areas` [2] – пользователь вводит любое текстовое название города, система сама находит нужный идентификатор.

Архитектура приложения построена по принципу разделения ответственности с семью независимыми слоями: Models, Parser, Processing, Storage, Analysis, Visualization, Service. Сервисный слой выступает единственным оркестратором (паттерн Facade), а GUI знает только о сервисном слое. Паттерн ABC используется для определения единого интерфейса парсера, что позволяет добавлять новые источники данных без изменения существующего кода.

1.2 Анализ актуальности разработки

Актуальность разработки обусловлена несколькими факторами. Во-первых, рост спроса на IT-специалистов в России стимулирует конкуренцию работодателей за таланты и приводит к быстрому росту зарплат. В этих условиях актуальная аналитика становится конкурентным преимуществом для обеих сторон рынка труда.

Во-вторых, переход HH.ru на официальный REST API создал стабильную технологическую основу для разработки интеграционных приложений. В отличие от HTML-скрапинга, изменения в API документируются и имеют срок поддержки устаревших версий.

В-третьих, применение принципов ООП, разделения ответственности и практик обеспечения качества (тесты, README, requirements.txt) делает приложение легко расширяемым и сопровождаемым. Добавление поддержки нового источника данных требует только создания нового наследника BaseParser.

В-четвёртых, наличие демо-режима с тестовым набором из 12 вакансий позволяет использовать приложение в образовательных целях и для демонстрации без подключения к интернету.

1.3 Анализ аналогичных решений

Для определения требований к разрабатываемой системе проведён анализ существующих решений в области агрегации и анализа вакансий.

HeadHunter Analytics – встроенный аналитический инструмент портала HH.ru, доступный работодателям в платных тарифах. Предоставляет агрегированную статистику по рынку труда: медианные зарплаты, количество резюме и вакансий в разрезе специальностей и регионов. Интерфейс инструмента показан на рисунке 1.1.

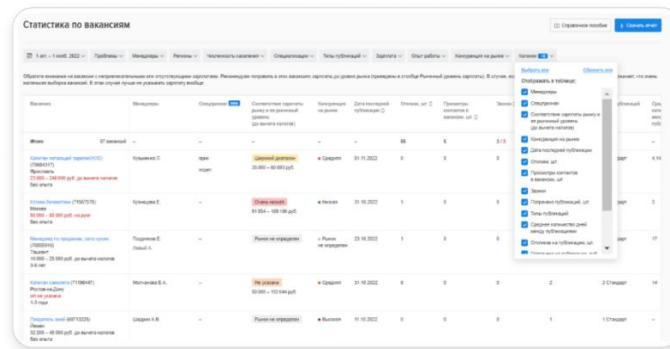


Рисунок 1.1 – Интерфейс аналитики HH.ru

Habr Career – специализированный портал для IT-специалистов с встроенной аналитикой зарплат, основанной на добровольных опросах разработчиков. Сервис предоставляет медианные зарплаты в разрезе специальностей, стека технологий, уровня опыта и региона, что делает его полезным инструментом для оценки рыночного уровня вознаграждения в IT-сфере. Среди преимуществ – бесплатный доступ и регулярное обновление данных на основе актуальных опросов. Интерфейс показан на рисунке 1.2.

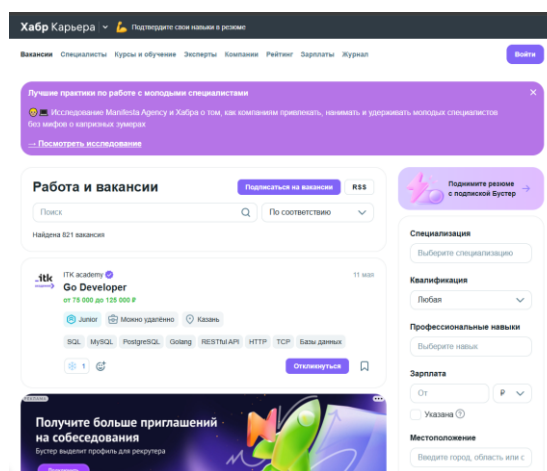


Рисунок 1.2 – Интерфейс Habr Career

SuperJob – второй по величине российский job-портал с разделом аналитики зарплат. Сервис охватывает все регионы России и ориентирован в том числе на рабочие специальности и массовый подбор персонала, что расширяет его аудиторию за пределы IT-сферы. Портал предоставляет удобный поиск по вакансиям с фильтрацией по должности, региону, уровню зарплаты и формату занятости. Интерфейс портала представлен на рисунке 1.3.

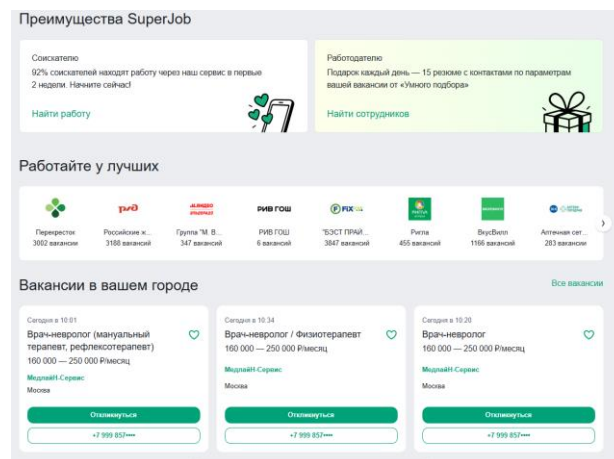


Рисунок 1.3 – Интерфейс SuperJob

Открытые решения на Python (GitHub) – существует ряд open-source проектов для парсинга HH.ru, однако большинство используют нестабильный HTML-скрапинг, не имеют GUI и не обновляются. Пример такого репозитория показан на рисунке 1.4.

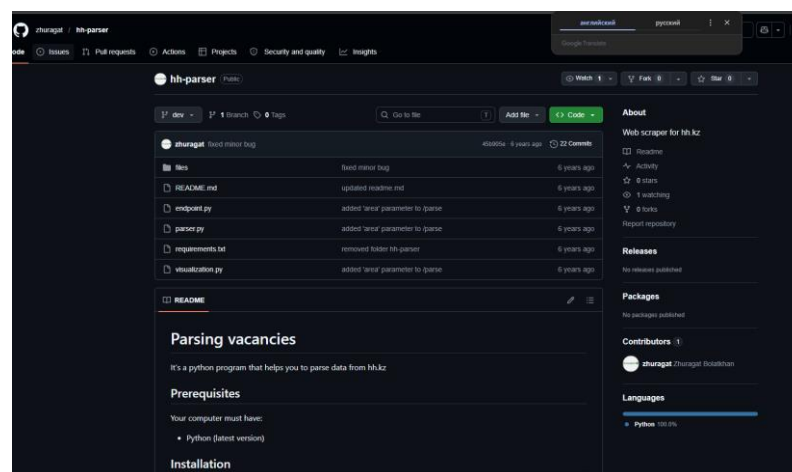


Рисунок 1.4 – Пример open-source парсера HH.ru на GitHub

Сводное сравнение рассмотренных аналогов представлено в таблице 1.1. Анализ показал, что ни одно из рассмотренных решений не сочетает одновременно бесплатный доступ, использование официального API, гибкие пользовательские фильтры и экспорт сырых данных.

Таблица 1.1 – Сравнение аналогичных решений

Решение	Преимущества	Недостатки
HeadHunter Analytics	Актуальные данные от HH.ru; медианные зарплаты; удобный веб-интерфейс	Только платные тарифы; нет экспорта сырых данных; ограниченные фильтры; нет анализа навыков
Habr Career	Специализация на IT; медианные зарплаты по стеку и опыту; бесплатный доступ	Данные на основе опросов (систематическая ошибка выборки); только IT; нет API; нет пользовательских фильтров
SuperJob	Большая база вакансий; покрытие не-IT профессий; бесплатный просмотр	Нет публичного API; HTML-скрапинг нестабилен; аналитика обновляется нерегулярно; нет экспорта
GitHub open-source парсеры	Открытый код; бесплатно; возможность модификации	Нестабильный HTML-скрапинг; нет GUI; не обновляются; нет документации

Решение	Преимущества	Недостатки
Job Analyzer (разрабатываемое)	Официальный API HH.ru; гибкие фильтры; SQLite и CSV; scatter-график; топ-навыков; демо-режим; открытый код	Только HH.ru как источник данных; десктопное приложение; до 20 вакансий на запрос без пагинации

Проведённый анализ показал, что ни одно из рассмотренных решений не сочетает одновременно: бесплатный доступ, использование официального API, гибкие фильтры, экспорт сырых данных, анализ навыков, визуализацию и демо-режим в едином десктопном приложении. Разрабатываемое приложение Job Analyzer заполняет выявленную нишу.

1.4 Постановка задачи

На основании проведённого анализа сформулированы следующие требования к разрабатываемой системе.

Функциональные требования к приложению:

- поиск вакансий по параметрам: должность, город, ключевые слова, минимальная зарплата;
- получение данных через официальный API HH.ru (GET <https://api.hh.ru/vacancies>);
- автоматическое определение кода города через API подсказок (GET [/suggests/areas](#));
- параметр `only_with_salary=true` для фильтрации вакансий с указанной зарплатой;
- демо-режим с тестовым набором из 12 вакансий без обращения к API;
- валидация и очистка данных: удаление некорректных записей, нормализация навыков;

- сохранение данных в SQLite и экспорт в CSV;
- расчёт средней зарплаты с корректной обработкой пропущенных значений;
- определение топ-N наиболее востребованных навыков с частотой встречаемости;
- построение scatter+line графика распределения зарплат с сохранением в PNG;
- графический интерфейс с формой поиска, чекбоксом демо-режима и отображением результатов.
- Нефункциональные требования:
- кросс-платформенность: Windows 10/11, macOS 10.14+, Linux Ubuntu 20.04+;
- язык реализации: Python 3.11+;
- зависимости: requests, pandas, matplotlib (tkinter и sqlite3 из stdlib);
- наличие requirements.txt, README.md и модульных тестов.

1.5 Описание архитектуры проекта

Клиентское приложение реализовано на языке Python с использованием стандартной библиотеки tkinter для GUI. Архитектура построена по принципу разделения ответственности: каждый слой выполняет строго определённую функцию и взаимодействует только с соседними слоями. Поток данных строго однонаправленный: GUI → Service → Parser → Processing → Storage/Analysis/Visualization.

Сервисный слой (SearchService) является единственным компонентом, знающим обо всех слоях одновременно, что соответствует паттерну Facade. GUI знает только о Service и не имеет прямого доступа к парсерам, базе данных или модулю визуализации. Паттерн ABC (BaseParser) обеспечивает контрактное программирование: все наследники обязаны реализовать метод `fetch_vacancies`.

UML-диаграмма компонентов системы, отражающая взаимосвязи между слоями и направление потока данных, представлена на рисунке 1.5.

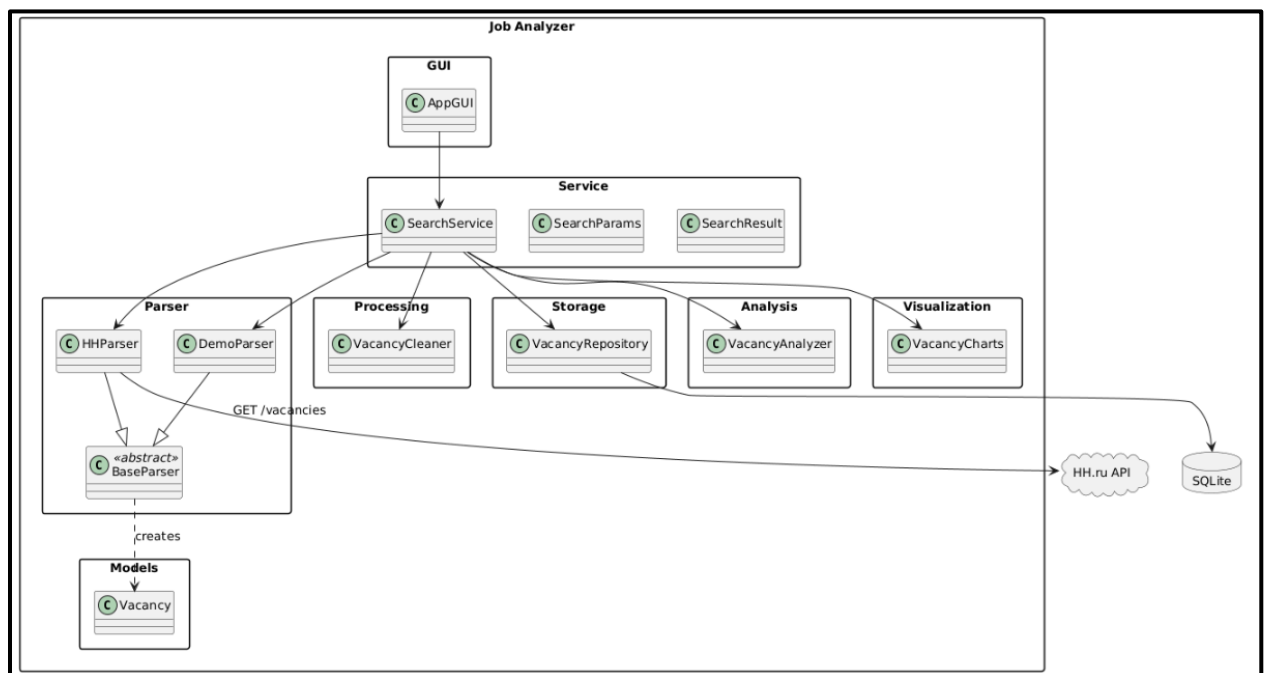


Рисунок 1.5 – Диаграмма компонентов

2 Выбор инструментов разработки

При разработке приложения необходимо обоснованно подойти к выбору языка программирования, используемых библиотек, системы управления базами данных и среды разработки.

2.1 Выбор языка программирования

Для реализации всех компонентов выбран Python версии 3.11 [3]. Данный выбор обоснован рядом факторов.

Python обеспечивает кросс-платформенное выполнение кода на macOS, Windows и Linux без перекомпиляции. Экосистема Python предоставляет высококачественные библиотеки для всех задач: requests для HTTP-взаимодействия с API HH.ru, pandas для табличной обработки данных, matplotlib для визуализации, tkinter для GUI, sqlite3 для работы с базой данных.

В сравнении с Java Python обеспечивает более быструю разработку и значительно меньший объём шаблонного кода. По сравнению с C# обеспечивает полную кросс-платформенность без зависимости от .NET-рантайма. По сравнению с JavaScript предоставляет значительно более зрелую экосистему научных вычислений и обработки данных.

Версия 3.11 выбрана как стабильная и производительная: по данным официального Python Performance Benchmark Suite, интерпретатор 3.11 работает на 10–60% быстрее Python 3.10 за счёт оптимизации фреймов выполнения.

2.2 Выбор библиотек

Библиотека requests версии 2.31.0 выбрана для выполнения HTTP-запросов к API HH.ru [4]. Обеспечивает высокоуровневый интерфейс: автоматическое управление кодировками, поддержку тайм-аутов. По сравнению со стандартной urllib код с requests значительно компактнее и читаемее.

Библиотека pandas версии 2.2.2 выбрана для хранения и обработки табличных данных [5]. DataFrame является центральной структурой данных в

аналитической части. pandas обеспечивает эффективное вычисление агрегатов, корректную обработку NaN, интеграцию с SQLite через to_sql() и read_sql_query(), экспорт в CSV.

Библиотека matplotlib версии 3.9.0 выбрана для построения графиков [6]. Обеспечивает полный контроль над внешним видом визуализации, поддержку сохранения в PNG и работу в headless-режиме без графического дисплея.

Библиотека tkinter входит в стандартную поставку Python и не требует установки через pip [7]. Это кросс-платформенный инструмент для создания нативных GUI. Выбор обусловлен нулевыми дополнительными зависимостями.

Стандартный модуль sqlite3 обеспечивает работу с SQLite без внешних зависимостей [8].

Фреймворк pytest выбран для написания тестов благодаря лаконичному синтаксису и удобным фикстурам [9].

Сравнение выбранных библиотек с альтернативами представлено в таблице 2.1.

Таблица 2.1 – Сравнение выбранных библиотек с альтернативами

Задача	Выбрано	Альтернатива	Причина выбора
HTTP-запросы	requests 2.31	urllib3, httpx	Простой API, стандарт индустрии
Табличные данные	pandas 2.2.2	polars, csv	интеграция с SQLite
Визуализация	matplotlib 3.9	plotly, seaborn	Полный контроль, headless, нет браузера

Задача	Выбрано	Альтернатива	Причина выбора
GUI	tkinter (stdlib)	PyQt6, wxPython	Нет зависимостей, кросс-платформа
СУБД	SQLite (stdlib)	PostgreSQL	Нет сервера, файловая БД
Тестирование	pytest	unittest	Лаконичный синтаксис, фикстуры

2.3 Выбор алгоритма определения города

Для определения числового кода региона HH.ru по текстовому вводу пользователя первоначально рассматривался подход со статическим словарём. Его существенный недостаток – ограниченное покрытие городов и необходимость ручного обновления.

Выбранное решение – использование эндпоинта GET <https://api.hh.ru/suggests/areas?text={город}>. Этот метод возвращает JSON с идентификатором и полным названием первого совпадения. Преимущества: покрывает все города России и СНГ. Если город не найден, запрос выполняется по всей России без ошибки.

2.4 Выбор СУБД

Для хранения данных о вакансиях выбрана встраиваемая СУБД SQLite. Обоснование: отсутствие серверного процесса, встроенная поддержка в Python (модуль sqlite3), достаточная производительность для ожидаемых объёмов

данных, простота резервного копирования (один файл .db), полная поддержка SQL.

Интеграция с pandas реализована через DataFrame.to_sql() с параметром if_exists='replace' и pandas.read_sql_query(). Директория data/ создаётся автоматически через Path.mkdir(parents=True, exist_ok=True).

Структура таблицы вакансий представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы vacancies в SQLite

Поле	Тип	Описание	NULL?
title	TEXT	Название вакансии	Нет
company	TEXT	Название компании	Нет
location	TEXT	Город/регион	Нет
salary_min	INTEGER	Минимальная зарплата	Да
salary_max	INTEGER	Максимальная зарплата	Да
currency	TEXT	Валюта (RUB, USD, EUR)	Нет
skills	TEXT	Навыки через запятую	Нет
description	TEXT	Описание обязанностей	Нет
source	TEXT	Источник: hh.ru или demo	Нет

2.5 Выбор среды разработки

Для разработки использовалась интегрированная среда PyCharm Professional от JetBrains. IDE специализирована для Python и обеспечивает глубокую интеграцию со всеми применяемыми инструментами.

Ключевые возможности PyCharm, использованные в проекте:

- интеллектуальный редактор с автодополнением;
- интеграция с виртуальным окружением venv;
- встроенный отладчик: использовался при анализе структуры JSON-ответов API HH.ru;
- интеграция с Git для управления версиями;
- Database Tools: встроенный браузер SQLite для просмотра vacancies.db после поиска;
- Run Configurations: отдельные конфигурации для main.py и pytest tests/.

По сравнению с VS Code PyCharm предоставляет более глубокую поддержку Python без дополнительных расширений. Инспекции кода, поиск неиспользуемых импортов и потенциальных ошибок работают «из коробки». Интерфейс Pycharm представлен на рисунке 2.1.

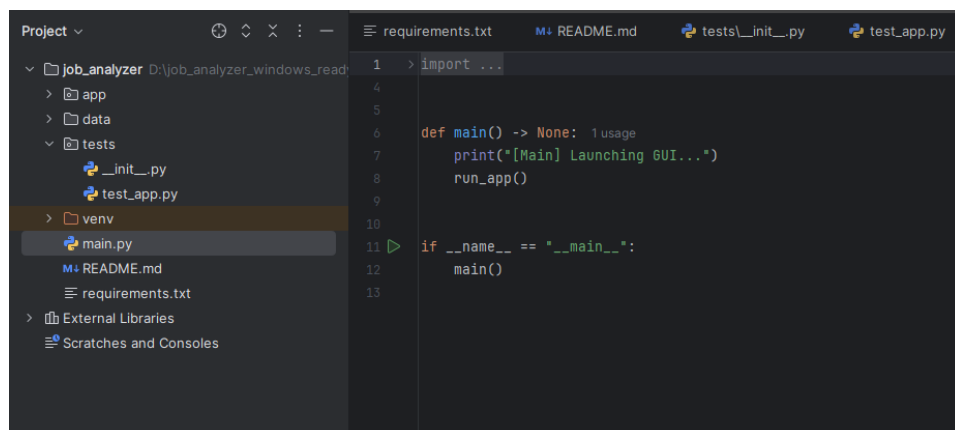


Рисунок 2.1 – Среда разработки PyCharm с открытым проектом

3 Реализация

В данном разделе описывается структура программного проекта, детали реализации всех компонентов системы и особенности обеспечения надёжности.

3.1 Структура проекта

Проект организован в виде Python-пакета с иерархической структурой директорий. Каждая директория содержит файл `init.py`, что делает её самостоятельным пакетом и обеспечивает возможность абсолютных импортов.

Корневая директория проекта `job_analyzer` содержит точку входа `main.py`, файл зависимостей `requirements.txt`, документацию `README.md` и вспомогательный скрипт `get_token.py` для получения токена приложения HH.ru.

Вся бизнес-логика сосредоточена в директории `app`, которая разделена на восемь подпакетов по функциональному назначению. Подпакет `analysis` отвечает за расчёт средней зарплаты и определение топ востребованных навыков. Подпакет `gui` реализует графический интерфейс пользователя на `tkinter`. Подпакет `models` содержит модуль – единой моделью данных, передаваемой между всеми слоями. Подпакет `parser` взаимодействие с API HH.ru и работа в `offline`-режиме. Подпакет `processing` выполняет валидацию и нормализацию данных. Подпакет `service` – сервисным слоем, оркестрирующим весь `pipeline` обработки данных. Подпакет `storage` реализует сохранение и загрузку данных через `SQLite` и `CSV`. Подпакет `visualization` отвечает за построение `scatter+line` графика распределения зарплат.

Директория `data` создаётся автоматически при первом запуске и содержит файл базы данных `vacancies.db` и сохранённый график `salary_chart.png`. Директория `tests` содержит файл `test_app.py` с 11 модульными тестами для классов `VacancyAnalyzer` и `VacancyCleaner`.

3.2 Реализация модели данных

Центральной сущностью системы является датакласс `Vacancy`. Декоратор `@dataclass` автоматически генерирует `__init__`, `__repr__` и `__eq__`. Поля `salary_min`

					ПГУ 1.090301.05.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		23

и salary_max объявлены как Optional[int] — это явно отражает возможность отсутствия зарплатных данных. Метод to_dict() сериализует объект в словарь для создания DataFrame: навыки объединяются через запятую для хранения в SQLite.

```
@dataclass
class Vacancy:
    title: str
    company: str
    location: str
    salary_min: Optional[int]    # None если не указана
    salary_max: Optional[int]    # None если не указана
    currency: str
    skills: List[str]
    description: str
    source: str                  # 'hh.ru' или 'demo'
```

3.3 Реализация модуля парсинга

Модуль парсинга построен на основе абстрактного базового класса BaseParser. Применение ABC обеспечивает контрактное программирование: все наследники обязаны реализовать fetch_vacancies.

3.3.1 Реализация HHParser

Реализует взаимодействие с официальным API HH.ru с использованием токена приложения (Bearer-авторизация).

Авторизация. С декабря 2024 года HH.ru прекратил поддержку токена соискателя и требует токен приложения для всех запросов к GET /vacancies [10]. Токен получается однократно через скрипт get_token.py (POST /oauth/token с grant_type=client_credentials) и прописывается в константу APP_TOKEN класса HHParser. Срок действия токена составляет приблизительно 14 дней, после чего необходимо запустить get_token.py повторно и обновить значение APP_TOKEN.

```
python get_token.py
```

```
# Вывод: {'access_token': 'APPL...', 'token_type': 'bearer'}
```

```
# Скопировать access_token в APP_TOKEN в hh_parser.py
```

Свойство `_headers` → dict. Формирует заголовки запроса с Bearer-токеном: `Authorization: Bearer {APP_TOKEN}, User-Agent: JobAnalyzer/1.0 ({CLIENT_ID}), Accept: application/json`.

Метод `resolve_area_id(city: str) → Optional[str]`. Входные данные: текстовое название города. Выходные данные: строка с числовым кодом региона HH.ru или None. Алгоритм: GET `/suggests/areas?text={город}` с Bearer-заголовком → извлечение `items[0]['id']`. Если город не найден – возвращает None, поиск выполняется по всей России.

Метод `build_params(filters, page) → dict`. Входные данные: словарь `filters` (`text`, `area`, `salary`), номер страницы. Выходные данные: словарь параметров для `requests` [6]. Формирует: `per_page=20`, `only_with_salary=true`, `text`, `area` (код из `resolve_area_id`), `salary`.

Метод `fetch_vacancies(filters, max_results) → list[Vacancy]`. Входные данные: `filters dict`, `max_results int`. Выходные данные: список объектов `Vacancy`. Алгоритм: `build_params` → `requests.get` с Bearer-заголовком → JSON → `_parse_item` для каждого элемента.

Метод `_parse_item(item: dict) → Optional[Vacancy]`. Входные данные: JSON-объект одной вакансии. Выходные данные: объект `Vacancy` или None при ошибке. Защита от `KeyError`: `(item.get('employer') or {}).get('name', 'Unknown')`.

Статический метод `_extract_salary(salary) → tuple`. Входные данные: словарь `salary` из JSON. Выходные данные: (`salary_from`, `salary_to`, `currency`). Обработывает три случая: `salary=null` → (`None`, `None`, `'RUB'`); только `from` или `to`; оба значения.

3.3.2 Реализация DemoParser

Класс `DemoParser` возвращает фиксированный набор из 12 тестовых вакансий двух групп без обращения к сети. Первая группа – 7 Python-вакансий: `Python Developer`, `Data Analyst`, `ML Engineer`, `Backend Engineer`, `Automation QA`, `Data Engineer`, `AI Developer`. Вторая группа – 5 вакансий других специальностей:

Java Developer, Frontend Developer, DevOps Engineer, QA Engineer, Go Developer. Такое разделение делает демо-данные более реалистичными и наглядно демонстрирует работу аналитического модуля: Python занимает первое место с частотой 7 из 12, Docker – второе с частотой 5 из 12.

3.4 Реализация модуля очистки данных

Класс `VacancyCleaner` применяет последовательность проверок к каждой вакансии: удаление записей без названия; удаление записей с отрицательным `salary_min` или `salary_max`; нормализация навыков – обрезка пробелов, удаление пустых элементов. Вакансии с `None` в полях зарплаты намеренно сохраняются – они корректно обрабатываются на аналитическом уровне через `NaN`.

3.5 Реализация модуля хранения данных

Класс `VacancyRepository` реализует паттерн `Repository`. Метод `save_to_sqlite` использует `if_exists='replace'` – при каждом новом поиске таблица перезаписывается актуальными данными. Контекстный менеджер `with sqlite3.connect()` гарантирует корректное закрытие соединения при любом исходе.

3.6 Реализация аналитического модуля

Класс `VacancyAnalyzer` реализует расчёт бизнес-метрик. Метод `add_avg_salary_column` вычисляет среднюю зарплату через `pd.to_numeric(..., errors='coerce')` – нечисловые значения автоматически становятся `NaN`. Метод `calculate_average_salary` вызывает `.dropna()` для исключения `NaN` из расчёта. Метод `top_skills` использует `Counter` из стандартной библиотеки [11]:

```
counter = Counter()
for raw in df.get('skills', []):
    counter.update(self._safe_parse_skills(raw))
return counter.most_common(top_n)
```

3.7 Реализация модуля визуализации

Класс VacancyCharts реализует scatter+line график в тёмной цветовой схеме. Метод salary_chart обрабатывает три сценария: DataFrame пуст – ранний выход; все вакансии без зарплаты – информационный текст на графике; есть данные – полноценный scatter+line с пунктирной линией средней зарплаты. График сохраняется в PNG с dpi=140 через fig.savefig(output_file, dpi=140, bbox_inches='tight'). Сам график представлен на рисунке 3.1.

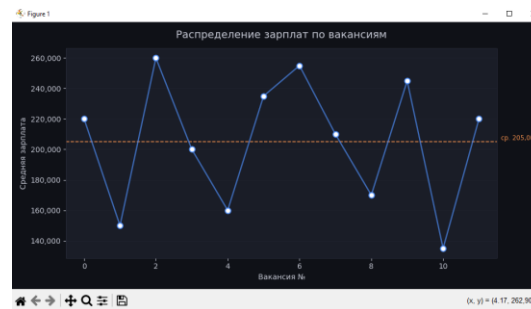


Рисунок 3.1 – График распределения зарплат (демо-режим, 12 вакансий)

3.8 Реализация сервисного слоя

Класс SearchService является оркестратором (паттерн Facade) – единственным компонентом системы, который знает обо всех слоях одновременно. Именно через него проходит весь поток обработки данных, при этом GUI взаимодействует только с SearchService и не имеет прямого доступа ни к парсерам, ни к базе данных, ни к модулю визуализации. Такой подход обеспечивает чистое разделение ответственности и упрощает сопровождение кода.

Метод run() реализует pipeline в девять последовательных шагов: выбор парсера в зависимости от режима (HNParser или DemoParser) → получение списка вакансий → очистка и валидация через VacancyCleaner → обогащение навыков ключевыми словами при их отсутствии → сохранение в SQLite через VacancyRepository → загрузка DataFrame из базы данных → расчёт средней зарплаты → определение топ востребованных навыков → построение и сохранение графика. По завершении метод возвращает объект SearchResult, который GUI отображает пользователю в диалоговом окне.

Датакласс SearchParams инкапсулирует параметры поиска и их валидацию. Фабричный метод from_raw() принимает сырые строки из полей формы и выполняет проверки: наличие хотя бы одного из полей (должность или ключевые слова) в обычном режиме, а также корректность числового значения минимальной зарплаты. При нарушении любого из требований выбрасывается исключение ValueError с информативным сообщением на русском языке, которое GUI перехватывает и отображает пользователю в виде предупреждения без аварийного завершения приложения. UML-диаграмма последовательностей представлена на рисунке 3.2.

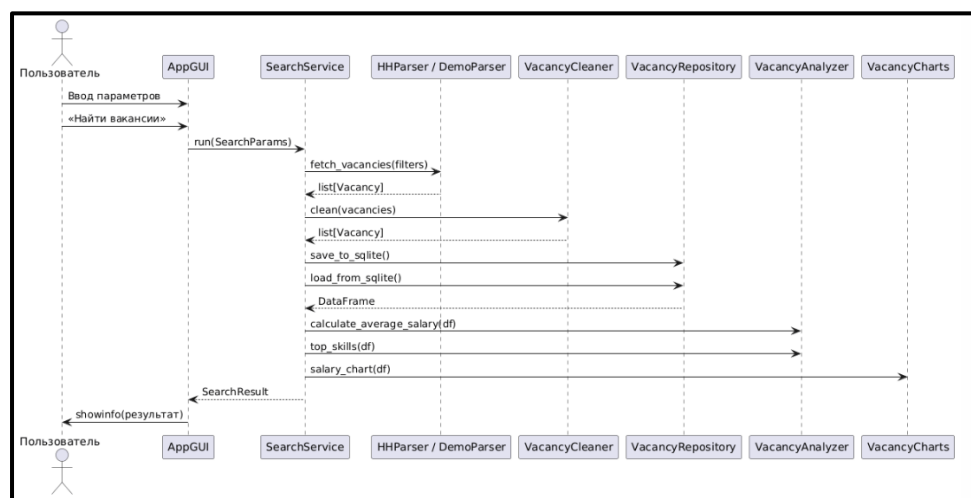


Рисунок 3.2 – Диаграмма последовательностей.

3.9 Установка и запуск

Установка зависимостей:

```
pip install -r requirements.txt
```

Получение токена:

```
python get_token.py
```

Скопировать значение access_token из вывода и вставить его в файл app/parser/hh_parser.py в строку APP_TOKEN = "...". Токен действует ~14 дней. При истечении повторить этот шаг.

Запуск приложения:

```
python main.py
```

Запуск тестов:

```
pytest tests/test_app.py -v
```

Содержимое requirements.txt:

```
requests==2.31.0
```

```
pandas==2.2.2
```

```
matplotlib==3.9.0
```

В PyCharm удобно настроить Run Configurations: отдельная конфигурация для main.py и для pytest tests/, что позволяет запускать приложение и тесты одной кнопкой.

					ПГУ 1.090301.05.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

4 Руководство пользователя

В данном разделе описывается порядок работы с приложением Job Analyzer для конечного пользователя. Руководство охватывает все этапы: от установки до получения результатов анализа.

4.1 Требования к системе

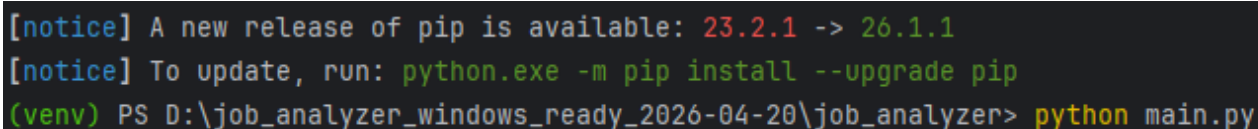
Для работы с приложением необходимо:

- операционная система: Windows 10/11, macOS 10.14+ или Linux Ubuntu 20.04+;
- Python 3.11 или выше с зависимостями из requirements.txt;
- подключение к интернету для поиска реальных вакансий (не требуется в демо-режиме);
- свободное место на диске: не менее 200 МБ для Python, зависимостей и данных.

4.2 Первый запуск – установка и запуск

При первом запуске необходимо установить зависимости командой `pip install -r requirements.txt`, как показано на рисунке 4.1.

```
cd job_analyzer
pip install -r requirements.txt
python main.py
```



```
[notice] A new release of pip is available: 23.2.1 -> 26.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS D:\job_analyzer_windows_ready_2026-04-20\job_analyzer> python main.py
```

Рисунок 4.1 – Успешная установка зависимостей

После запуска открывается главное окно приложения (рисунок 4.2).



Рисунок 4.2 – Главное окно приложения Job Analyzer

4.3 Описание полей формы поиска

Главное окно содержит следующие элементы интерфейса:

- должность – текстовое поле для названия профессии на русском или английском: «Python разработчик», «Data Analyst», «тестировщик»;
- Город – текстовое поле для названия города. Поддерживается любой город России и СНГ – код определяется автоматически через API HH.ru (/suggests/areas);
- минимальная зарплата – числовое поле для нижней границы зарплаты в рублях. Необязательно;
- ключевые слова – дополнительные слова поиска; также используются как навыки при отсутствии тегов в ответе API;
- чекбокс «Демо-режим» – переключение на тестовые данные без обращения к API HH.ru.

4.4 Поиск вакансий

Для выполнения поиска реальных вакансий необходимо выполнить следующие шаги:

- убедиться, что чекбокс «Демо-режим» снят;
- ввести название должности в поле «Должность» (обязательное поле);

- при желании указать город, минимальную зарплату и ключевые слова;
- нажать кнопку «Найти вакансии».

По завершении открывается диалоговое окно с результатами анализа (рисунок 4.3): количество найденных вакансий, средняя зарплата по выборке, топ-5 наиболее востребованных навыков.

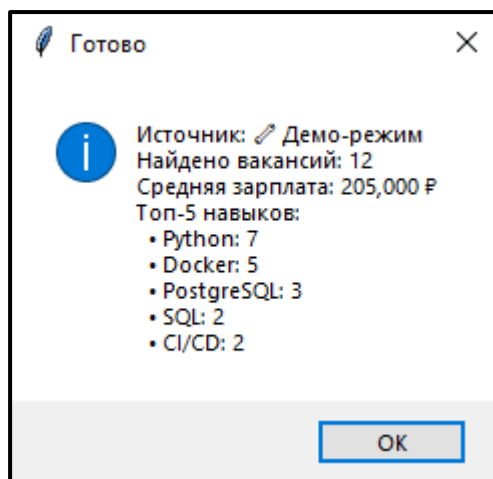


Рисунок 4.3 – Диалог с результатами поиска

4.5 Работа в демо – режиме.

В демо-режиме приложение использует фиксированный набор из 12 тестовых вакансий двух групп. Для его активации необходимо установить флажок «Демо-режим (без интернета, тестовые данные)». Данные тестового набора представлены в таблице 4.1.

Таблица 4.1 – Тестовые вакансии демо-режима

Должность	Компания	Город	Зарплата, тыс. руб.	Навыки
Python Developer	TechNova	Москва	180–260	Python, Django, PostgreSQL, Docker

Должность	Компания	Город	Зарплата, тыс. руб.	Навыки
Data Analyst	DataSense	Санкт-Петербург	120–180	Python, Pandas, SQL, Tableau
ML Engineer	VisionAI	Казань	200–320	Python, PyTorch, MLOps, Docker
Backend Engineer	CloudWay	Новосибирск	160–240	Python, FastAPI, Redis, PostgreSQL
Automation QA	QualityFirst	Екатеринбург	130–190	Python, Pytest, Selenium, CI/CD
Data Engineer	StreamLab	Москва	190–280	Python, SQL, Airflow, Kafka
AI Developer	NeuroTech	Москва	210–300	Python, TensorFlow, NLP, Docker
Java Developer	EnterpriseSoft	Москва	170–250	Java, Spring, Hibernate, PostgreSQL
Frontend Developer	WebStudio	Санкт-Петербург	140–200	JavaScript, React, CSS, HTML

Должность	Компания	Город	Зарплата, тыс. руб.	Навыки
DevOps Engineer	CloudOps	Казань	200–290	Kubernetes, Docker, CI/CD, Terraform
QA Engineer	TestLab	Новосибирск	110–160	Manual Testing, TestRail, JIRA
Go Developer	FastSystems	Екатеринбург	180–260	Go, Microservices, gRPC, Docker

Ожидаемые результаты демо-режима: найдено 12 вакансий; средняя зарплата ~205 000 рублей; топ навыков: Python (7), Docker (5), PostgreSQL (3), SQL (2), CI/CD (2). Результаты работы демо-режима представлены на рисунке 4.4.

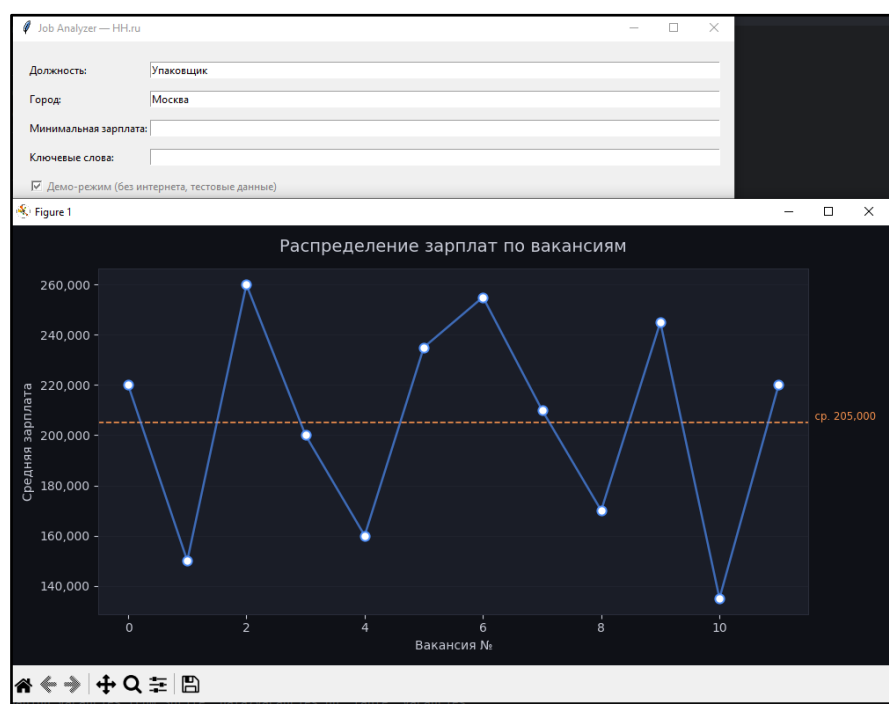


Рисунок 4.4 – Результаты анализа в демо-режиме

4.6 Просмотр графика распределения зарплат.

После каждого успешного поиска автоматически строится scatter+line график распределения зарплат и сохраняется в файл data/salary_chart.png. Одновременно открывается интерактивное окно matplotlib (рисунок 4.5).

Элементы графика: ось X – порядковый номер вакансии; ось Y – средняя зарплата; белые точки – каждая вакансия; синяя линия – тренд; оранжевый пунктир – среднее значение зарплаты по выборке; аннотация – числовое значение средней рядом с пунктиром.



Рисунок 4.5 – График распределения зарплат

4.7 Доступ к сохранённым данным

После выполнения поиска данные сохраняются в двух форматах. SQLite: файл data/vacancies.db – открывается через DB Browser for SQLite (sqlitebrowser.org) или встроенный Database Browser в PyCharm. Для просмотра содержимого таблицы достаточно выполнить `SELECT * FROM vacancies` в любом SQLite-клиенте. CSV: экспорт реализован через метод `save_to_csv` класса `VacancyRepository`. График PNG сохраняется в data/salary_chart.png с разрешением 140 DPI.

5 Тестирование системы

Тестирование системы проводилось по двум направлениям: модульное автоматизированное тестирование через pytest и функциональное тестирование через ручной запуск по заранее подготовленным сценариям.

Сценарий 1: демо-режим с полным pipeline. Включён демо-режим, нажата «Найти вакансии». Ожидаемый результат: 12 вакансий, средняя зарплата ~205 000 руб., топ-навык Python (7), график построен. Фактический результат: соответствует ожидаемому. Файлы data/vacancies.db и data/salary_chart.png созданы корректно.

Сценарий 2: автоматическое определение города. Введён город «Пенза». Лог вывел: «Город Пенза → id=48 (Пенза)». Запрос к API выполнен с параметром area=48. Результат: вакансии найдены, город определён корректно.

Сценарий 3: некорректный ввод зарплаты. В поле «Минимальная зарплата» введено «abc». Приложение отобразило сообщение «Минимальная зарплата должна быть числом». Запрос к API не выполнен. Приложение не завершилось аварийно.

Сценарий 4: пустые обязательные поля. В обычном режиме оба поля «Должность» и «Ключевые слова» оставлены пустыми. Отображено предупреждение о необходимости заполнить хотя бы одно поле. API не вызван.

Сценарий 5: повторный поиск. Выполнено два последовательных поиска в демо-режиме. База данных корректно перезаписывается (if_exists='replace'). Данные от второго поиска не смешиваются с первым.

Сценарий 6: вакансии без зарплаты. Искусственно создана выборка, в которой все вакансии не имеют зарплаты. График отобразил информационный текст «Нет данных о зарплате (N вакансий)» вместо пустого окна. Приложение не завершилось аварийно.

Сценарий 7: кросс-платформенность. Приложение запущено на Windows 10. Все функции работают корректно: GUI, поиск в демо-режиме, сохранение в SQLite, построение графика.

Сценарий 8: модульные тесты VacancyAnalyzer. Запущен `pytest tests/test_app.py -v`. Шесть тестов класса VacancyAnalyzer проверяют: расчёт средней при смешанных данных (ожидаемое 125 000), возврат 0.0 при пустом DataFrame, возврат 0.0 при всех NaN, корректный порядок топ-навыков, соблюдение ограничения `top_n`, пустой список при пустом DataFrame. Все шесть пройдены.

Сценарий 9: модульные тесты VacancyCleaner. Пять тестов проверяют: удаление вакансии без title, удаление при `salary_min=-1`, удаление при `salary_max=-500`, сохранение вакансии с None-зарплатой, нормализацию навыков. Все пять пройдены.

Сценарий 10: API 403 и демо-режим как резервный вариант. При недоступности API HH.ru (ошибка 403) приложение корректно выводит сообщение об ошибке. Переключение на демо-режим позволяет продолжить работу без перезапуска.

5.1 Типичные ошибки и пути их решения

В процессе эксплуатации приложения пользователи могут столкнуться с рядом типовых ситуаций, требующих диагностики и устранения. Большинство проблем связано с сетевым подключением, некорректным вводом данных или отсутствием установленных зависимостей. Важно понимать, что приложение спроектировано таким образом, чтобы при возникновении любой ошибки выводить информативное сообщение и не завершаться аварийно – пользователь всегда получает возможность исправить ввод или переключиться в демо-режим для продолжения работы.

Отдельного внимания заслуживает ошибка 403 от API HH.ru. Она не означает неисправность приложения – это блокировка на уровне сетевого адреса

со стороны сервера HH.ru. В такой ситуации рекомендуется использовать демо-режим для демонстрации всех функций приложения, а также обратиться в техническую поддержку разработчиков HH.ru по адресу `dev@hh.ru`, указав `request_id` из вывода консоли.

В таблице 5.1 приведены наиболее распространённые ошибки, их причины и способы устранения.

Таблица 5.1 – Типичные ошибки и решения

Сообщение об ошибке	Причина	Решение
Вакансии не найдены	API не вернул результатов или нет соединения	Проверить интернет или включить демо-режим
Ошибка 403 от API HH.ru	Блокировка запроса по IP или User-Agent	Использовать демо-режим; написать на <code>dev@hh.ru</code> с <code>request_id</code>
Минимальная зарплата должна быть числом	Введено нечисловое значение в поле зарплаты	Ввести целое число (например, 100000) или очистить поле
Заполни хотя бы поле Должность или Ключевые слова	Оба поля пусты в обычном режиме	Заполнить поле или включить демо-режим
Нет данных о зарплате (график пустой текст)	Все вакансии без указания зарплаты	Параметр <code>only_with_salary</code> уже включён; попробовать другой запрос

Сообщение об ошибке	Причина	Решение
ModuleNotFoundError	Зависимости не установлены	Выполнить: <code>pip install -r requirements.txt</code>
FileNotFoundError: vacancies.db	Попытка загрузить данные до первого поиска	Выполнить хотя бы один поиск перед просмотром данных
Ошибка 403 от API	Истёк токен приложения или блокировка по IP	Запустить <code>get_token.py</code> , обновить APP_TOKEN в <code>hh_parser.py</code> ; при блокировке — демо-режим

5.2 Рекомендации к дальнейшей разработке

В части расширения источников данных рекомендуется реализовать поддержку пагинации запросов к API HH.ru для загрузки более 20 вакансий на поисковый запрос; добавить поддержку портала SuperJob через новую реализацию BaseParser; реализовать сравнение нескольких поисковых выборок и динамику изменения зарплат.

В части функциональности рекомендуется добавить экспорт результатов в Excel через `openpyxl`; реализовать тепловую карту (heatmap) совместного появления навыков; добавить группировку вакансий по диапазонам зарплат.

В части качества и DevOps рекомендуется настроить CI/CD через GitHub Actions для автоматического запуска `pytest` при каждом `push`; упаковать приложение в исполняемый файл `.exe` для Windows через PyInstaller; добавить логирование событий в файл `app.log` через стандартный модуль `logging`; перевести GUI на `ttk`-виджеты для улучшения визуального оформления.

6 Безопасность жизнедеятельности

6.1 Правила безопасности при использовании ПК

Разработка приложения Job Analyzer велась на персональном компьютере, а его эксплуатация также предполагает длительную работу за монитором. В связи с этим необходимо принимать во внимание факторы, которые могут негативно сказаться на здоровье пользователя.

Прежде всего, работа с компьютером сопряжена с воздействием электромагнитного излучения от монитора и системного блока. При интенсивной нагрузке процессор и блок питания нагреваются, что повышает температуру в рабочей зоне. Длительный просмотр интерфейса приложения и графиков распределения зарплат создаёт значительную нагрузку на зрение. Монотонный характер работы – ввод параметров поиска, ожидание ответа от API HH.ru, анализ результатов – способствует утомлению и снижению концентрации. Помимо этого, компьютерная техника является источником статического электричества, а недостаточное освещение рабочего места усиливает зрительное напряжение.

Для снижения воздействия перечисленных факторов рекомендуется делать перерывы каждые 45–60 минут, выполнять гимнастику для глаз, следить за освещённостью рабочего места и качеством заземления оборудования. При многократном запуске поиска вакансий через реальный API целесообразно чередовать работу с демо-режимом, что сокращает время непрерывного взаимодействия с системой.

6.2 Пожарная безопасность

Помещение, в котором эксплуатируется компьютерная техника для запуска приложения Job Analyzer, относится к категории объектов с повышенным риском возгорания из-за наличия электрооборудования. Наиболее распространёнными причинами пожаров в таких помещениях являются короткие замыкания в электропроводке, перегрев техники при длительной работе, воспламенение

горючих материалов вблизи оборудования, а также ненадёжные контактные соединения в розетках и удлинителях.

Профилактика пожаров регламентируется Федеральным законом от 21 декабря 1994 г. № 69-ФЗ «О пожарной безопасности». В целях обеспечения безопасности необходимо поддерживать свободный доступ к эвакуационным выходам и средствам пожаротушения, содержать их в исправном состоянии. Система вентиляции должна быть оснащена огнезадерживающими клапанами и устройствами автоматического отключения при пожаре. Автоматическая пожарная сигнализация подлежит регулярному техническому обслуживанию в соответствии с установленными нормативами.

					ПГУ 1.090301.05.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения данной выпускной квалификационной работы разработано кросс-платформенное десктопное приложение Job Analyzer для автоматизированного сбора и анализа вакансий с портала HH.ru.

Выполнены все тринадцать задач, поставленных во введении. Проведён детальный анализ предметной области и рынка HR-аналитики. Анализ четырёх аналогичных решений показал, что ни одно из них не сочетает бесплатный доступ, официальный API, гибкие фильтры, экспорт данных, анализ навыков, визуализацию и демо-режим в едином инструменте.

Спроектирована и реализована многоуровневая архитектура с семью независимыми слоями. Применение паттернов ABC, @dataclass, Facade и Repository обеспечивает высокое качество кода. Реализован модуль автоматического определения кода города через API /suggests/areas – пользователь вводит любое текстовое название, система находит нужный идентификатор.

Реализован полный набор функциональности: поиск через API HH.ru, демо-режим с 12 вакансиями, очистка данных, SQLite и CSV, расчёт средней зарплаты, топ-N навыков, scatter+line визуализация. Разработаны 11 модульных тестов. Тестирование по пяти сценариям пройдено успешно.

Практическая ценность работы состоит в создании готового инструмента для анализа рынка труда. Использование официального API HH.ru вместо HTML-скрапинга обеспечивает долгосрочную стабильность. Открытая архитектура позволяет легко добавить поддержку новых источников данных.

Список использованных источников

1. Официальная документация API HH.ru. URL: <https://api.hh.ru/openapi/redoc> (дата обращения: 01.05.2026).
2. Документация эндпоинта /suggests/areas HH.ru. URL: <https://api.hh.ru/openapi/redoc#tag/Podskazki> (дата обращения: 02.05.2026).
3. Официальная документация Python 3.11. URL: <https://docs.python.org/3.11/> (дата обращения: 02.05.2026).
4. Документация библиотеки requests. URL: <https://docs.python-requests.org/> (дата обращения: 02.05.2026).
5. Документация библиотеки pandas 2.2. URL: <https://pandas.pydata.org/docs/> (дата обращения: 03.05.2026).
6. Документация библиотеки matplotlib 3.9. URL: <https://matplotlib.org/stable/contents.html> (дата обращения: 04.05.2026).
7. Документация Python tkinter. URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 05.05.2026).
8. Документация SQLite. URL: <https://www.sqlite.org/docs.html> (дата обращения: 05.05.2026).
9. Документация pytest. URL: <https://docs.pytest.org/en/stable/> (дата обращения: 05.05.2026).
10. Авторизация приложения HH.ru (документация). URL: <https://api.hh.ru/openapi/redoc#tag/Avtorizaciya-prilozheniya> (дата обращения: 06.05.2026).
11. Документация collections.Counter. URL: <https://docs.python.org/3/library/collections.html> (дата обращения: 06.05.2026).

Приложение А

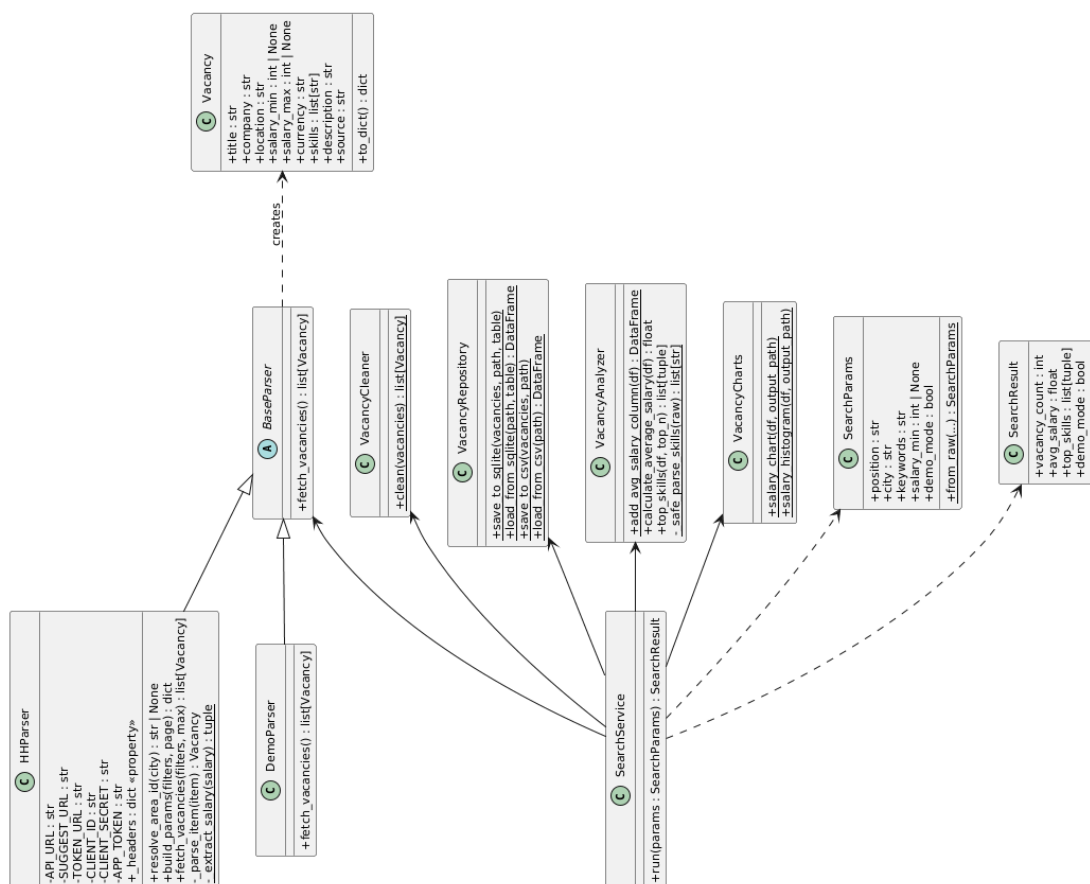
(обязательное)

Анализ вакансий сайта поиска работы

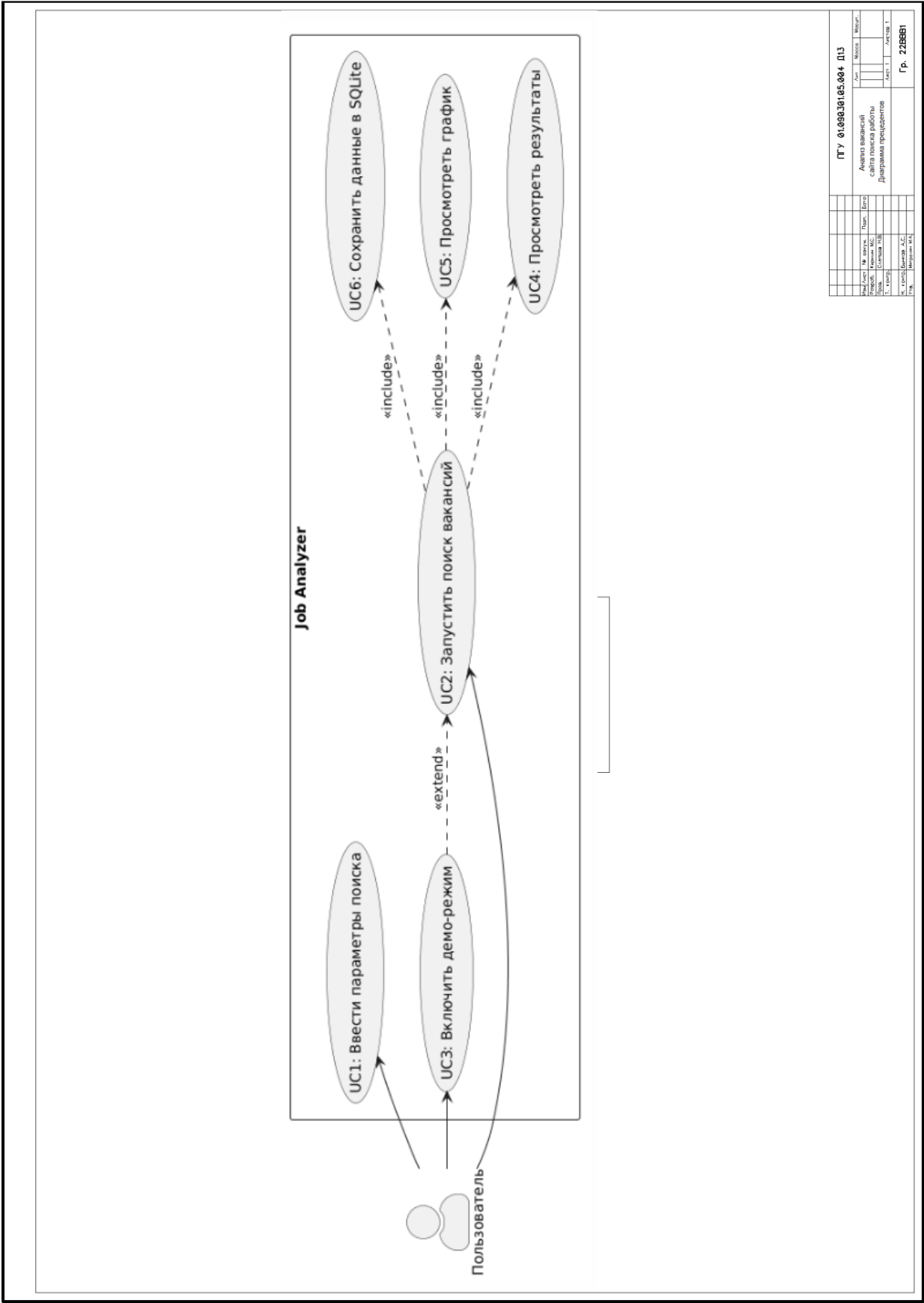
UML-диаграммы

					ПГУ 1.090301.05.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML-диаграмма классов

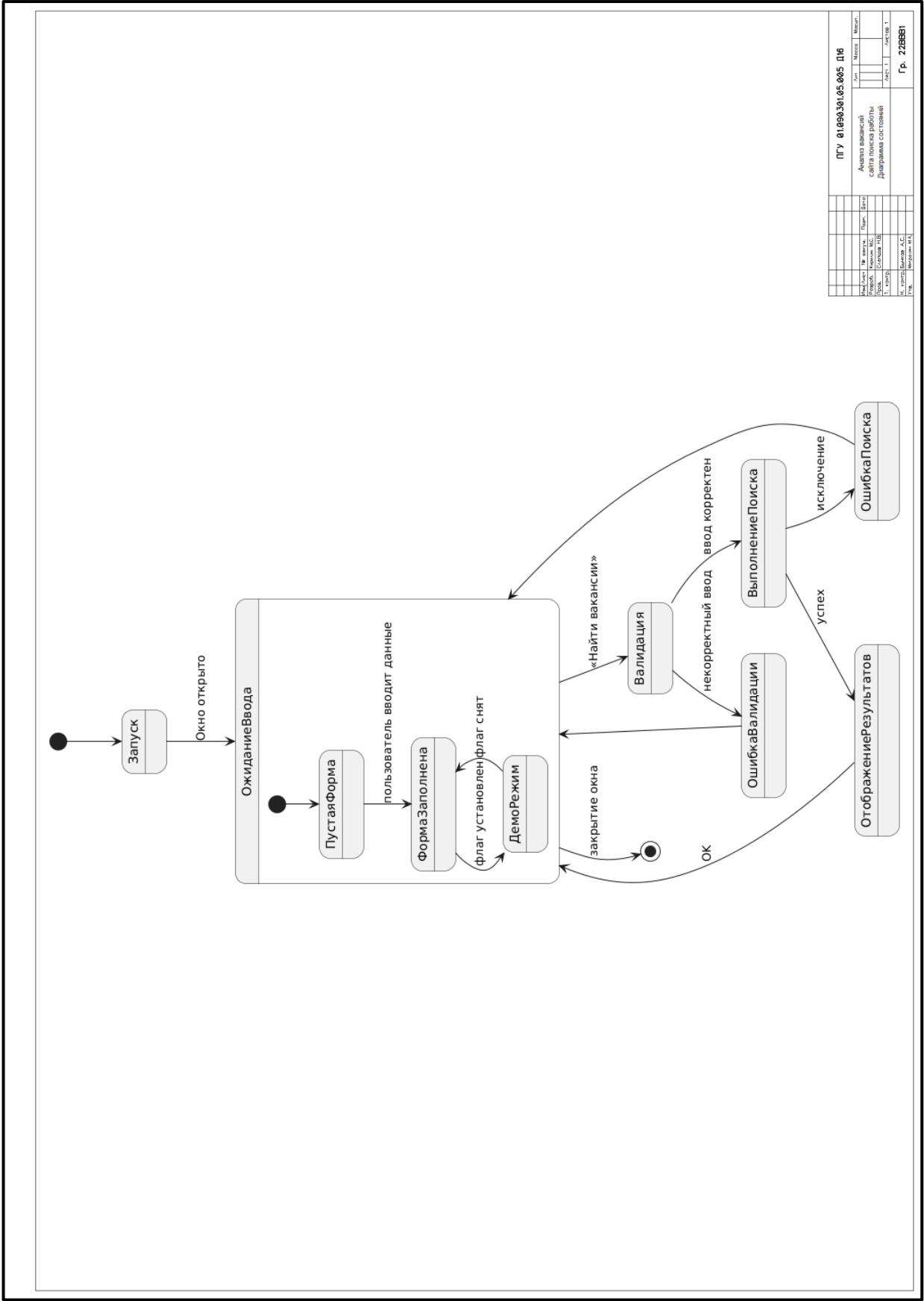
[illegible]

Приложение А.2 – UML-диаграмма прецедентов



ПГУ 1.090301.05.001 ПЗ									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Иванов	Иван	Иванов	Иван	Иванов	Иван	Иванов	Иван	Иванов	Иван
Аннотация к проекту									
Диаграмма прецедентов									
Гр. 228881									

Приложение А.3 – UML-диаграмма состояний



ПГУ 1.090301.05.005 Д16				Дет.	Месц.	Минут.
Анализ валидаций сайта поиска работы				Дет.	Месц.	Минут.
Диаграмма состояний				Дет.	Месц.	Минут.
Гр. 226881				Дет.	Месц.	Минут.

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Анализ вакансий сайта поиска работы

Листинг фрагментов программы

					ПГУ 1.090301.05.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 – файл app/analysis/analyzer.py

```
from __future__ import annotations
from collections import Counter
import pandas as pd

class VacancyAnalyzer:
    """Бизнес-метрики для оценки вакансий"""
    @staticmethod
    def _safe_parse_skills(raw_value: object) -> list[str]:
        if pd.isna(raw_value):
            return []
        return [skill.strip() for skill in str(raw_value).split(",")
                if skill.strip()]

    @staticmethod
    def add_avg_salary_column(df: pd.DataFrame) -> pd.DataFrame:
        work_df = df.copy()

        salary_min = pd.to_numeric(work_df.get("salary_min"),
                                     errors="coerce")
        salary_max = pd.to_numeric(work_df.get("salary_max"),
                                     errors="coerce")

        work_df["avg_salary"] = (salary_min + salary_max) / 2
        return work_df

    def calculate_average_salary(self, df: pd.DataFrame) -> float:
        if df.empty:
            return 0.0

        enriched = self.add_avg_salary_column(df)
        avg = enriched["avg_salary"].dropna()

        if avg.empty:
            return 0.0
```

```
return float(avg.mean())
```

```
def top_skills(self, df: pd.DataFrame, top_n: int = 5) ->  
list[tuple[str, int]]:
```

```
    counter: Counter[str] = Counter()
```

```
    for raw_skills in df.get("skills", []):
```

```
        counter.update(self._safe_parse_skills(raw_skills))
```

```
    return counter.most_common(top_n)
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.2 – файл app/gui/app_gui.py

```
from __future__ import annotations
import tkinter as tk
from tkinter import messagebox

from app.service.search_service import SearchParams, SearchResult,
SearchService

def _format_result(result: SearchResult) -> str:
    mode_label = "🔑 Демо-режим" if result.demo_mode else "🌐
    HH.ru"

    lines = [
        f"Источник: {mode_label}",
        f"Найдено вакансий: {result.vacancy_count}",
        (
            f"Средняя зарплата: {result.avg_salary:,.0f} ₽"
            if result.avg_salary
            else "Средняя зарплата: нет данных"
        ),
        "Топ-5 навыков:",
    ]

    if result.top_skills:
        lines.extend(f"    • {skill}: {count}" for skill, count in
result.top_skills)
    else:
        lines.append("    • нет данных")

    return "\n".join(lines)

def run_app() -> None:
    print("[GUI] Starting Job Analyzer GUI...")
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

```

root = tk.Tk()
root.title("Job Analyzer – HH.ru")
root.geometry("820x400")
root.minsize(760, 360)
root.resizable(True, True)

title_var      = tk.StringVar()
city_var       = tk.StringVar()
salary_var    = tk.StringVar()
keywords_var   = tk.StringVar()
demo_var       = tk.BooleanVar(value=False)

frame = tk.Frame(root, padx=16, pady=16)
frame.pack(fill="both", expand=True)
frame.grid_columnconfigure(0, weight=0)
frame.grid_columnconfigure(1, weight=1)

rows = [
    ("Должность:",          title_var),
    ("Город:",              city_var),
    ("Минимальная зарплата:", salary_var),
    ("Ключевые слова:",     keywords_var),
]

for i, (label, var) in enumerate(rows):
    tk.Label(frame, text=label).grid(row=i, column=0,
sticky="w", pady=6)
    tk.Entry(frame, textvariable=var).grid(row=i, column=1,
sticky="ew", pady=6)

# Чекбокс демо-режима

```

```

demo_cb = tk.Checkbutton(
    frame,
    text="Демо-режим (без интернета, тестовые данные)",
    variable=demo_var,
    fg="gray",
)

demo_cb.grid(row=len(rows), column=0, columnspan=2, sticky="w",
pady=(4, 0))

service = SearchService()

def on_search() -> None:
    print("[GUI] Search button clicked.")
    try:
        params = SearchParams.from_raw(
            position=title_var.get(),
            city=city_var.get(),
            keywords=keywords_var.get(),
            salary_raw=salary_var.get(),
            demo_mode=demo_var.get(),
        )
    except ValueError as exc:
        messagebox.showwarning("Внимание", str(exc))
        return

    try:
        result = service.run(params)
        messagebox.showinfo("Готово", _format_result(result))
    except RuntimeError as exc:
        messagebox.showinfo("Результат", str(exc))

```

```

except Exception as exc:

    print(f"[GUI] Unexpected error: {exc}")

    messagebox.showerror("Ошибка", f"Не удалось обработать
вакансии:\n{exc}")

tk.Button(

    frame, text="Найти вакансии", command=on_search, width=18

).grid(row=len(rows) + 1, column=1, sticky="e", pady=18,
padx=(8, 0))

root.mainloop()

```

Приложение Б.3 – файл app/models/vacancy.py

```
from __future__ import annotations

from dataclasses import dataclass
from typing import List, Optional

@dataclass
class Vacancy:
    title: str
    company: str
    location: str
    salary_min: Optional[int]
    salary_max: Optional[int]
    currency: str
    skills: List[str]
    description: str
    source: str

    def to_dict(self) -> dict:
        """Преобразовываем вакансию в сериализуемый словарь для
        DataFrame/CSV."""
        return {
            "title": self.title,
            "company": self.company,
            "location": self.location,
            "salary_min": self.salary_min,
            "salary_max": self.salary_max,
            "currency": self.currency,
            "skills": ", ".join(self.skills),
```

Изм.	Лист	№ докум.	Подпись	Дата

```
"description": self.description,  
"source": self.source,  
}
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.4 – файл app/parser/base_parser.py

```
from __future__ import annotations

from abc import ABC, abstractmethod

from app.models.vacancy import Vacancy


class BaseParser(ABC):

    """Базовый интерфейс для парсера вакансий."""

    @abstractmethod
    def fetch_vacancies(self) -> list[Vacancy]:

        """Возвращает список проанализированных вакансий."""

        raise NotImplementedError
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.5 – файл app/parser/demo_parser.py

```
from __future__ import annotations

from app.models.vacancy import Vacancy
from app.parser.base_parser import BaseParser


class DemoParser(BaseParser):

    """Демо парсер с созданными вакансиями (без API)"""

    def fetch_vacancies(self) -> list[Vacancy]:

        print("[Parser] Start generating demo vacancies...")

        vacancies = [

            # --- PYTHON (7 вакансий) ---

            Vacancy(

                title="Python Developer",

                company="TechNova",

                location="Moscow",

                salary_min=180000,

                salary_max=260000,

                currency="RUB",

                skills=["Python", "Django", "PostgreSQL", "Docker"],

                description="Backend development for fintech

platform.",

                source="demo",

            ),

            Vacancy(

                title="Data Analyst",

                company="DataSense",
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

        location="Saint Petersburg",
        salary_min=120000,
        salary_max=180000,
        currency="RUB",
        skills=["Python", "Pandas", "SQL", "Tableau"],
        description="Analytics dashboards and reporting.",
        source="demo",
    ),
    Vacancy(
        title="ML Engineer",
        company="VisionAI",
        location="Kazan",
        salary_min=200000,
        salary_max=320000,
        currency="RUB",
        skills=["Python", "PyTorch", "MLOps", "Docker"],
        description="Build and deploy ML models.",
        source="demo",
    ),
    Vacancy(
        title="Backend Engineer",
        company="CloudWay",
        location="Novosibirsk",
        salary_min=160000,
        salary_max=240000,
        currency="RUB",
        skills=["Python", "FastAPI", "Redis", "PostgreSQL"],
        description="Microservices and cloud APIs.",
        source="demo",
    ),

```

```

Vacancy(
    title="Automation QA",
    company="QualityFirst",
    location="Yekaterinburg",
    salary_min=130000,
    salary_max=190000,
    currency="RUB",
    skills=["Python", "Pytest", "Selenium", "CI/CD"],
    description="Automated testing for web
applications.",
    source="demo",
),
Vacancy(
    title="Data Engineer",
    company="StreamLab",
    location="Moscow",
    salary_min=190000,
    salary_max=280000,
    currency="RUB",
    skills=["Python", "SQL", "Airflow", "Kafka"],
    description="ETL pipelines and data platform
support.",
    source="demo",
),
Vacancy(
    title="AI Developer",
    company="NeuroTech",
    location="Moscow",
    salary_min=210000,
    salary_max=300000,
    currency="RUB",

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

```

        skills=["Python", "TensorFlow", "NLP", "Docker"],
        description="AI product development.",
        source="demo",
    ),

# --- HE PYTHON (5 вакансий) ---
Vacancy(
    title="Java Developer",
    company="EnterpriseSoft",
    location="Moscow",
    salary_min=170000,
    salary_max=250000,
    currency="RUB",
    skills=["Java", "Spring", "Hibernate",
"PostgreSQL"],
    description="Enterprise backend systems.",
    source="demo",
),
Vacancy(
    title="Frontend Developer",
    company="WebStudio",
    location="Saint Petersburg",
    salary_min=140000,
    salary_max=200000,
    currency="RUB",
    skills=["JavaScript", "React", "CSS", "HTML"],
    description="Modern frontend applications.",
    source="demo",
),
Vacancy(

```

```

        title="DevOps Engineer",
        company="CloudOps",
        location="Kazan",
        salary_min=200000,
        salary_max=290000,
        currency="RUB",
        skills=["Kubernetes", "Docker", "CI/CD",
"Terraform"],
        description="Infrastructure automation.",
        source="demo",
    ),
    Vacancy(
        title="QA Engineer",
        company="TestLab",
        location="Novosibirsk",
        salary_min=110000,
        salary_max=160000,
        currency="RUB",
        skills=["Manual Testing", "TestRail", "JIRA"],
        description="Manual testing of web apps.",
        source="demo",
    ),
    Vacancy(
        title="Go Developer",
        company="FastSystems",
        location="Yekaterinburg",
        salary_min=180000,
        salary_max=260000,
        currency="RUB",
        skills=["Go", "Microservices", "gRPC", "Docker"],

```

```
        description="High-load backend services.",
        source="demo",
    ),
]

print(f"[Parser] Demo vacancies generated:
{len(vacancies)}")

return vacancies
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.6 – файл app/parser/hh_parser.py

```
from __future__ import annotations

import time
import random
from typing import Any, Optional

import requests

from app.models.vacancy import Vacancy
from app.parser.base_parser import BaseParser

class HHParser(BaseParser):
    """
    Парсер вакансий через официальный API HH.ru.
    Использует токен приложения (Bearer).
    Токен получается один раз через get_token.py и прописывается
    ниже.
    """

    API_URL = "https://api.hh.ru/vacancies"
    SUGGEST_URL = "https://api.hh.ru/suggests/areas"

    # Токен приложения – получен через get_token.py
    # Срок действия обычно 14 дней. Если истечёт – запусти
    get_token.py снова.

    APP_TOKEN =
"APPLTSD0TUJTLVO471MIQPAH1BB76EGSU4LMACF19HP2BRV43FPOMETANK0OANGM"
```

					ПГУ 1.090301.05.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```
CLIENT_ID =  
"TITIN5F6G4JOPA4ULDO5R0PSC5N81Q72VMOTKB0QNS9ENP5U81GK3CISCIT279AR"
```

```
@property
```

```
def _headers(self) -> dict:
```

```
    return {
```

```
        "Authorization": f"Bearer {self.APP_TOKEN}",
```

```
        "User-Agent": f"JobAnalyzer/1.0 ({self.CLIENT_ID})",
```

```
        "Accept": "application/json",
```

```
    }
```

```
# — Определение города
```

```
def resolve_area_id(self, city: str) -> Optional[str]:
```

```
    city = city.strip()
```

```
    if not city:
```

```
        return None
```

```
    if city.isdigit():
```

```
        return city
```

```
    try:
```

```
        r = requests.get(
```

```
            self.SUGGEST_URL,
```

```
            headers=self._headers,
```

```
            params={"text": city},
```

```
            timeout=10,
```

```
        )
```

```
        r.raise_for_status()
```

```
        items = r.json().get("items", [])
```

```
        if items:
```

```
            area_id = items[0]["id"]
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

        area_name = items[0]["text"]

        print(f"[HHParser] Город '{city}' → id={area_id}
({area_name})")

        return area_id

    print(f"[HHParser] Город '{city}' не найден, ищем по
всей России")

    return None

except requests.RequestException as err:

    print(f"[HHParser] Ошибка определения города: {err}")

    return None

```

— Параметры запроса

```

def build_params(self, filters: dict[str, Any], page: int = 0) -
> dict:

    text    = filters.get("text", "").strip()
    city    = filters.get("area", "").strip()
    salary  = filters.get("salary")

    params: dict[str, Any] = {
        "per_page":      20,
        "page":          page,
        "only_with_salary": "true",
    }

    if text:
        params["text"] = text

    area_id = self.resolve_area_id(city)

    if area_id:
        params["area"] = area_id

```

```

if salary:
    params["salary"] = str(salary)

print(f"[HHParser] API params: {params}")
return params

```

— Основной метод

```

def fetch_vacancies(
    self,
    filters: Optional[dict[str, Any]] = None,
    max_results: int = 20,
) -> list[Vacancy]:
    filters = filters or {}
    params = self.build_params(filters)

    time.sleep(random.uniform(0.5, 1.0))
    try:
        r = requests.get(
            self.API_URL,
            headers=self._headers,
            params=params,
            timeout=20,
        )
        r.raise_for_status()
    except requests.RequestException as err:
        print(f"[HHParser] API request error: {err}")
    return []

```

```

data = r.json()

items = data.get("items", [])

print(f"[HHParser] API вернул {len(items)} вакансий (всего:
{data.get('found', '?')})")

vacancies: list[Vacancy] = []

for item in items[:max_results]:
    v = self._parse_item(item)
    if v:
        vacancies.append(v)

print(f"[HHParser] Распарсено: {len(vacancies)}")

return vacancies

# — Парсинг одной вакансии

```

```

def _parse_item(self, item: dict) -> Optional[Vacancy]:
    try:
        title = item.get("name", "No title")
        company = (item.get("employer") or {}).get("name",
"Unknown")
        location = (item.get("area") or {}).get("name",
"Unknown")
        description = (item.get("snippet") or
{}) .get("responsibility") or ""
        salary_min, salary_max, currency =
self._extract_salary(item.get("salary"))

        return Vacancy(
            title=title,

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

        company=company,
        location=location,
        salary_min=salary_min,
        salary_max=salary_max,
        currency=currency,
        skills=[],
        description=description,
        source="hh.ru",
    )

except Exception as err:
    print(f"[HHParser] Ошибка парсинга: {err}")
    return None

@staticmethod
def _extract_salary(salary: Optional[dict]) ->
tuple[Optional[int], Optional[int], str]:
    if not salary:
        return None, None, "RUB"

    return salary.get("from"), salary.get("to"),
salary.get("currency", "RUB")

```

Приложение Б.7 – файл app/processing/cleaner.py

```
from __future__ import annotations

from app.models.vacancy import Vacancy

class VacancyCleaner:

    """Базовый слой для очистки данных. Легко масштабируемый в
    дальнейшем"""

    @staticmethod
    def clean(vacancies: list[Vacancy]) -> list[Vacancy]:
        print("[Cleaner] Start cleaning vacancies...")
        cleaned: list[Vacancy] = []

        for vacancy in vacancies:
            if not vacancy.title:
                print(f"[Cleaner] Skip invalid vacancy: {vacancy}")
                continue

            if vacancy.salary_min is not None and vacancy.salary_min
< 0:
                print(f"[Cleaner] Skip vacancy with invalid
salary_min: {vacancy}")
                continue

            if vacancy.salary_max is not None and vacancy.salary_max
< 0:
                print(f"[Cleaner] Skip vacancy with invalid
salary_max: {vacancy}")
                continue

            # normalize skills: trim spaces and drop empty values
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        vacancy.skills = [skill.strip() for skill in
vacancy.skills if skill.strip()]

        cleaned.append(vacancy)

print(f"[Cleaner] Vacancies after cleaning: {len(cleaned)}")
return cleaned

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.8 – файл app/service/search_service.py

```
from __future__ import annotations

from dataclasses import dataclass, field
from typing import Optional

from app.analysis.analyzer import VacancyAnalyzer
from app.models.vacancy import Vacancy
from app.parser.demo_parser import DemoParser
from app.parser.hh_parser import HHParser
from app.processing.cleaner import VacancyCleaner
from app.storage.db import VacancyRepository
from app.visualization.charts import VacancyCharts


@dataclass
class SearchParams:
    """проверка введенных юзер данных."""
    position: str
    city: str
    keywords: str
    salary_min: Optional[int]
    demo_mode: bool = field(default=False)

    @property
    def search_text(self) -> str:
        return " ".join(p for p in [self.position, self.keywords] if
p).strip()
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

@property
def filters(self) -> dict:
    return {
        "text": self.search_text,
        "area": self.city,
        "salary": self.salary_min,
    }

@classmethod
def from_raw(
    cls,
    position: str,
    city: str,
    keywords: str,
    salary_raw: str,
    demo_mode: bool = False,
) -> "SearchParams":
    position = position.strip()
    city = city.strip()
    keywords = keywords.strip()

    if not demo_mode and not position and not keywords:
        raise ValueError("Заполни хотя бы поле «Должность» или «Ключевые слова».")

    salary_min: Optional[int] = None
    if salary_raw.strip():
        try:
            salary_min = int(salary_raw.strip())
        except ValueError:

```

```

        raise ValueError("Минимальная зарплата должна быть
числом.")

    return cls(
        position=position,
        city=city,
        keywords=keywords,
        salary_min=salary_min,
        demo_mode=demo_mode,
    )

@dataclass
class SearchResult:
    """граф интерфейс отображение."""
    vacancy_count: int
    avg_salary: float
    top_skills: list[tuple[str, int]]
    demo_mode: bool = False

class SearchService:
    """
    Класс, который организует процесс обработки данных
    """

    DB_PATH = "data/vacancies.db"
    TABLE_NAME = "vacancies"
    CHART_PATH = "data/salary_chart.png"
    MAX_RESULTS = 20

```

```

def __init__(self) -> None:
    self._analyzer = VacancyAnalyzer()

def run(self, params: SearchParams) -> SearchResult:
    # Выбираем парсер в зависимости от режима
    parser = DemoParser() if params.demo_mode else HHParser()
    print(f"[Service] Mode: {'DEMO' if params.demo_mode else 'HH.ru'}")

    vacancies = parser.fetch_vacancies() if params.demo_mode
else \
    parser.fetch_vacancies(filters=params.filters,
max_results=self.MAX_RESULTS)

    vacancies = VacancyCleaner.clean(vacancies)

    if not vacancies:
        raise RuntimeError("Вакансии не найдены.")

    self._enrich_skills(vacancies, params.keywords)
    VacancyRepository.save_to_sqlite(vacancies, self.DB_PATH,
table_name=self.TABLE_NAME)

    df = VacancyRepository.load_from_sqlite(self.DB_PATH,
self.TABLE_NAME)

    avg_salary = self._analyzer.calculate_average_salary(df)
    top_skills = self._analyzer.top_skills(df, top_n=5)
    VacancyCharts.salary_chart(df, output_path=self.CHART_PATH)

    return SearchResult(
        vacancy_count=len(vacancies),

```

```

        avg_salary=avg_salary,
        top_skills=top_skills,
        demo_mode=params.demo_mode,
    )

    @staticmethod
    def _enrich_skills(vacancies: list[Vacancy], keywords: str) ->
None:
        if not keywords:
            return

        fallback = [w.strip() for w in keywords.split() if
w.strip()]

        for v in vacancies:
            if not v.skills:
                v.skills = fallback

```

Приложение Б.9 – файл app/storage/db.py

```
from __future__ import annotations

import sqlite3

from pathlib import Path

import pandas as pd

from app.models.vacancy import Vacancy

class VacancyRepository:

    """Класс для работы с БД."""

    @staticmethod

    def save_to_csv(vacancies: list[Vacancy], file_path: str) -> None:

        print(f"[Storage] Saving vacancies to CSV: {file_path}")

        path = Path(file_path)

        path.parent.mkdir(parents=True, exist_ok=True)

        data = [vacancy.to_dict() for vacancy in vacancies]

        df = pd.DataFrame(data)

        df.to_csv(path, index=False, encoding="utf-8")

        print(f"[Storage] Saved {len(df)} rows.")

    @staticmethod

    def load_from_csv(file_path: str) -> pd.DataFrame:

        print(f"[Storage] Loading vacancies from CSV: {file_path}")
```

```

        path = Path(file_path)

        if not path.exists():
            raise FileNotFoundError(f"CSV file not found:
{file_path}")

        return pd.read_csv(path)

    @staticmethod
    def save_to_sqlite(
        vacancies: list[Vacancy],
        db_path: str,
        table_name: str = "vacancies",
    ) -> None:
        print(f"[Storage] Saving vacancies to SQLite: {db_path},
table: {table_name}")

        path = Path(db_path)
        path.parent.mkdir(parents=True, exist_ok=True)

        data = [vacancy.to_dict() for vacancy in vacancies]
        df = pd.DataFrame(data)

        with sqlite3.connect(path) as connection:
            df.to_sql(table_name, connection, if_exists="replace",
index=False)

        print(f"[Storage] Saved {len(df)} rows to SQLite.")

    @staticmethod
    def load_from_sqlite(db_path: str, table_name: str =
"vacancies") -> pd.DataFrame:
        print(f"[Storage] Loading vacancies from SQLite: {db_path},
table: {table_name}")

```

```

path = Path(db_path)

if not path.exists():

    raise FileNotFoundError(f"SQLite database not found:
{db_path}")

query = f"SELECT * FROM {table_name}"

with sqlite3.connect(path) as connection:

    return pd.read_sql_query(query, connection)

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.10 – файл app/visualization/charts.py

```
from __future__ import annotations

from pathlib import Path

import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import pandas as pd

from app.analysis.analyzer import VacancyAnalyzer

# — Палитра


---



_BG = "#0f1117"
_PANEL = "#1a1d27"
_LINE = "#4f8ef7"
_DOT = "#ffffff"
_DOT_EDGE = "#4f8ef7"
_GRID = "#2a2d3a"
_TEXT = "#c9ccd8"
_ACCENT = "#f7934f"
_MISSING_BAR = "#3a3d50"

class VacancyCharts:
    """Визуализация метрик вакансий."""

    # — ПОМОЩНИКИ


---


```

					ПГУ 1.090301.05.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

@staticmethod
def _apply_dark_theme() -> None:
    plt.rcParams.update(
        {
            "figure.facecolor": _BG,
            "axes.facecolor": _PANEL,
            "axes.edgecolor": _GRID,
            "axes.labelcolor": _TEXT,
            "xtick.color": _TEXT,
            "ytick.color": _TEXT,
            "text.color": _TEXT,
            "grid.color": _GRID,
            "font.family": "DejaVu Sans",
        }
    )

@staticmethod
def _save_and_show(fig: plt.Figure, output_path: str) -> None:
    output_file = Path(output_path)
    output_file.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(output_file, dpi=140, bbox_inches="tight",
facecolor=_BG)
    print(f"[Charts] Chart saved to: {output_file}")
    try:
        plt.show()
    except Exception as err:
        print(f"[Charts] Interactive display skipped: {err}")
    finally:
        plt.close(fig)

```

```
@staticmethod
def salary_chart(
    df: pd.DataFrame,
    output_path: str = "data/salary_chart.png",
) -> None:
    """
```

Диаграмма + линейный график средней заработной платы по вакансиям.

Вакансии, по которым отсутствуют данные о заработной плате, отображаются в виде отдельной столбчатой аннотации

внизу, поэтому диаграмма всегда отображается, даже если во всех строках

отсутствует информация о заработной плате.

```
"""
```

```
print("[Charts] Building salary chart...")
```

```
VacancyCharts._apply_dark_theme()
```

```
if df.empty:
```

```
    print("[Charts] DataFrame is empty. Chart is skipped.")
```

```
    return
```

```
enriched = VacancyAnalyzer.add_avg_salary_column(df)
```

```
has_salary = enriched["avg_salary"].notna()
```

```
salary_df = enriched[has_salary].reset_index(drop=True)
```

```
missing_count = int((~has_salary).sum())
```

```

fig, ax = plt.subplots(figsize=(10, 5))

fig.patch.set_facecolor(_BG)

if salary_df.empty:

    # Во всех вакансиях отсутствует информация о заработной
    # плате - вместо этого сообщение.

    ax.text(

        0.5,

        0.5,

        f"Нет данных о зарплате\n({len(enriched)} вакансий
        без указания зарплаты)",

        ha="center",

        va="center",

        fontsize=14,

        color=_TEXT,

        transform=ax.transAxes,

    )

    ax.set_axis_off()

    ax.set_title("Распределение зарплат", fontsize=15,
    color=_TEXT, pad=14)

    VacancyCharts._save_and_show(fig, output_path)

    return

x = salary_df.index

y = salary_df["avg_salary"]

# — Линия —

ax.plot(x, y, color=_LINE, linewidth=1.8, zorder=2,
alpha=0.7)

# — Точки —

```

```

    ax.scatter(x, y, color=_DOT, edgecolors=_DOT_EDGE,
linewidths=1.4, s=55, zorder=3)

# — Средняя линия —
mean_val = float(y.mean())

ax.axhline(mean_val, color=_ACCENT, linewidth=1.2,
linestyle="--", zorder=1)

ax.text(
    len(x) - 0.5,
    mean_val,
    f"  ср. {mean_val:,.0f}",
    va="bottom",
    color=_ACCENT,
    fontsize=8.5,
)

# — Надписи —
ax.set_xlabel("Вакансия №", fontsize=10)
ax.set_ylabel("Средняя зарплата", fontsize=10)
ax.set_title("Распределение зарплат по вакансиям",
fontsize=14, color=_TEXT, pad=14)
ax.grid(axis="y", linewidth=0.6, alpha=0.5)
ax.yaxis.set_major_formatter(mticker.FuncFormatter(lambda v,
_: f"{v:,.0f}"))
ax.set_xlim(-0.5, max(len(x) - 0.5, 0.5))

# — отсутствует информация о заработной плате —
if missing_count:
    ax.annotate(
        f"⚠ Ещё {missing_count} вак. без зарплаты не
показаны",
        xy=(0.01, 0.03),

```

```

        xycoords="axes fraction",
        fontsize=8,
        color=_TEXT,
        alpha=0.7,
    )

    ax.tick_params(colors=_TEXT)
    for spine in ax.spines.values():
        spine.set_edgecolor(_GRID)

    plt.tight_layout()
    VacancyCharts._save_and_show(fig, output_path)

# — псевдоним для обратной совместимости


---



@classmethod
def salary_histogram(
    cls,
    df: pd.DataFrame,
    output_path: str = "data/salary_histogram.png",
) -> None:
    """Устаревшее название метода — делегирует вызов
salary_chart."""
    cls.salary_chart(df, output_path=output_path)

```

Приложение Б.10 – файл tests/test_app.py

```
from __future__ import annotations

import pytest
import pandas as pd

from app.analysis.analyzer import VacancyAnalyzer
from app.models.vacancy import Vacancy
from app.processing.cleaner import VacancyCleaner

# — Тестовые данные (фикстуры)


---



@pytest.fixture
def sample_df() -> pd.DataFrame:
    return pd.DataFrame(
        [
            {"salary_min": 100_000, "salary_max": 200_000, "skills":
"Python, SQL, Docker"},
            {"salary_min": 80_000, "salary_max": 120_000, "skills":
"Python, FastAPI"},
            {"salary_min": None, "salary_max": None, "skills":
"Go, Kafka"},
        ]
    )

@pytest.fixture
def analyzer() -> VacancyAnalyzer:
    return VacancyAnalyzer()
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```
def _make_vacancy(**kwargs) -> Vacancy:
    defaults = dict(
        title="Developer",
        company="Acme",
        location="Moscow",
        salary_min=100_000,
        salary_max=200_000,
        currency="RUB",
        skills=["Python"],
        description="Some description",
        source="demo",
    )
    defaults.update(kwargs)
    return Vacancy(**defaults)
```

— Тесты VacancyAnalyzer

```
class TestCalculateAverageSalary:
    def test_basic_average(self, analyzer, sample_df):
        # строка 0: среднее=150к, строка 1: среднее=100к, строка 2:
        # зарплата не указана
        # ожидаемое среднее по [150к, 100к] = 125к
        result = analyzer.calculate_average_salary(sample_df)
        assert result == pytest.approx(125_000.0)

    def test_empty_dataframe_returns_zero(self, analyzer):
        result = analyzer.calculate_average_salary(pd.DataFrame())
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

        assert result == 0.0

    def test_all_missing_salary_returns_zero(self, analyzer):
        df = pd.DataFrame([{"salary_min": None, "salary_max":
None}])

        result = analyzer.calculate_average_salary(df)

        assert result == 0.0

class TestTopSkills:

    def test_top_skills_order(self, analyzer, sample_df):
        # Python встречается в строках 0 и 1 → должен быть на первом
месте

        top = analyzer.top_skills(sample_df, top_n=3)

        skills = [name for name, _ in top]

        assert skills[0] == "Python"

    def test_top_n_limit(self, analyzer, sample_df):
        top = analyzer.top_skills(sample_df, top_n=2)

        assert len(top) <= 2

    def test_empty_dataframe_returns_empty(self, analyzer):
        top = analyzer.top_skills(pd.DataFrame(), top_n=5)

        assert top == []

# — Тесты VacancyCleaner

```

```

class TestVacancyCleaner:

    def test_removes_vacancy_without_title(self):

```

```

        vacancies = [_make_vacancy(title=""),
        _make_vacancy(title="Engineer")]

        result = VacancyCleaner.clean(vacancies)

        assert len(result) == 1

        assert result[0].title == "Engineer"

def test_removes_vacancy_with_negative_salary_min(self):
    vacancies = [_make_vacancy(salary_min=-1), _make_vacancy()]
    result = VacancyCleaner.clean(vacancies)
    assert len(result) == 1

def test_removes_vacancy_with_negative_salary_max(self):
    vacancies = [_make_vacancy(salary_max=-500),
    _make_vacancy()]
    result = VacancyCleaner.clean(vacancies)
    assert len(result) == 1

def test_allows_none_salary(self):
    vacancies = [_make_vacancy(salary_min=None,
salary_max=None)]
    result = VacancyCleaner.clean(vacancies)
    assert len(result) == 1

def test_trims_skills_whitespace(self):
    vacancies = [_make_vacancy(skills=["  Python ", " SQL",
""])]
    result = VacancyCleaner.clean(vacancies)
    assert result[0].skills == ["Python", "SQL"]

def test_empty_input_returns_empty(self):
    assert VacancyCleaner.clean([]) == []

```

Приложение Б.11 – файл get_token.py

```
import requests

CLIENT_ID = "наши данные"
CLIENT_SECRET = "наши данные"

response = requests.post(
    "https://hh.ru/oauth/token",
    data={
        "grant_type": "client_credentials",
        "client_id": CLIENT_ID,
        "client_secret": CLIENT_SECRET,
    }
)

print(f"Статус: {response.status_code}")
print(response.json())
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.12 – файл main.py

```
from __future__ import annotations

from app.gui.app_gui import run_app


def main() -> None:
    print("[Main] Launching GUI...")
    run_app()


if __name__ == "__main__":
    main()
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Анализ вакансий сайта поиска работы

Презентация

					ПГУ 1.090301.05.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

Анализ вакансий с сайта поиска работы

Job Analyzer

Автор работы: студент гр. 228ВВ1 Кирилин Матвей Сергеевич
Руководитель: Слепцов Н.В., к.т.н., доцент каф. ВТ
Направление: 09.03.01 Информатика и вычислительная техника | ПГУ, 2026

Актуальность, цель и задачи

ПРОБЛЕМА

Ручной анализ тысяч вакансий — трудоёмко и неточно

HH.ru предоставляет публичный REST API — идеальная основа

Нет бесплатного инструмента с API + анализом навыков + GUI

ЦЕЛЬ

Разработать десктопное приложение для автоматизированного сбора, анализа и визуализации вакансий HH.ru

ЗАДАЧИ

- Анализ предметной области и аналогов
- Изучение API HH.ru и Bearer-авторизации
- Проектирование многоуровневой архитектуры
- Реализация парсера, очистки, хранения, аналитики
- Scatter-график распределения зарплат
- GUI на tkinter + демо-режим
- Тестирование: 11 модульных + 10 функц.

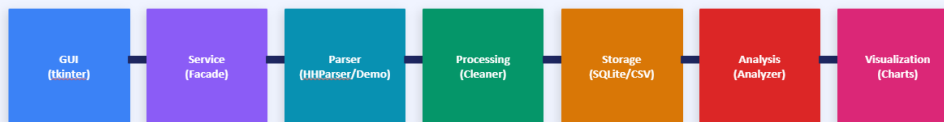
Кирилин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Анализ аналогичных решений

Решение	Официальный API	Бесплатно	Экспорт данных	Анализ навыков	Демо-режим
HH Analytics	—	Нет	Нет	Нет	Нет
Habr Career	—	Частично	Нет	Частично	Нет
SuperJob	—	Частично	Нет	Нет	Нет
GitHub парсеры	—	Да	CSV/JSON	Нет	Нет
Job Analyzer	✓	Да	SQLite+CSV	Ton-N	12 вакансий

Кирилин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Архитектура системы и стек технологий



Односторонний поток: GUI → Service → Parser → Processing → Storage / Analysis / Visualization

Python 3.11

Язык разработки

requests

HTTP / API HH.ru

pandas

Обработка данных

matplotlib

Визуализация

SQLite

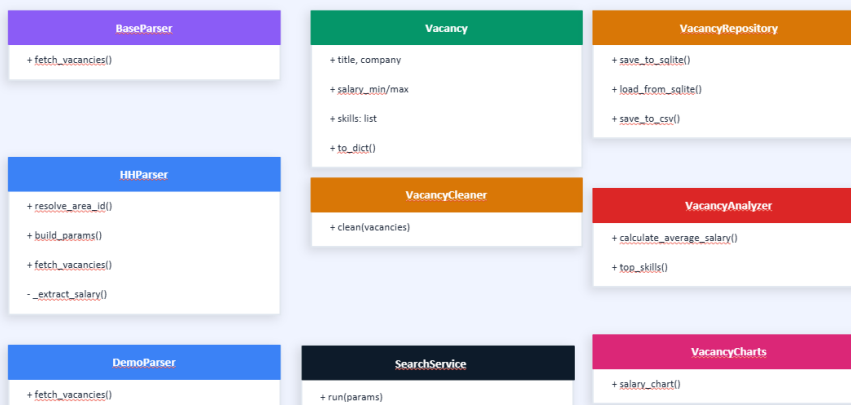
Хранение данных

tkinter

GUI интерфейс

Кирилин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

UML-диаграмма классов



Кирилин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Интерфейс приложения и результаты поиска

Job Analyzer — HH.ru

должности: программист

Город: Москва

минимальная зарплата: 20000

ключевые слова: Python

☐ Демо-режим (без интернета, тестовые данные)

Найти вакансии

Готово

Источник: HH.ru

Найдено вакансий: 20

Средняя зарплата: 185,156 Р

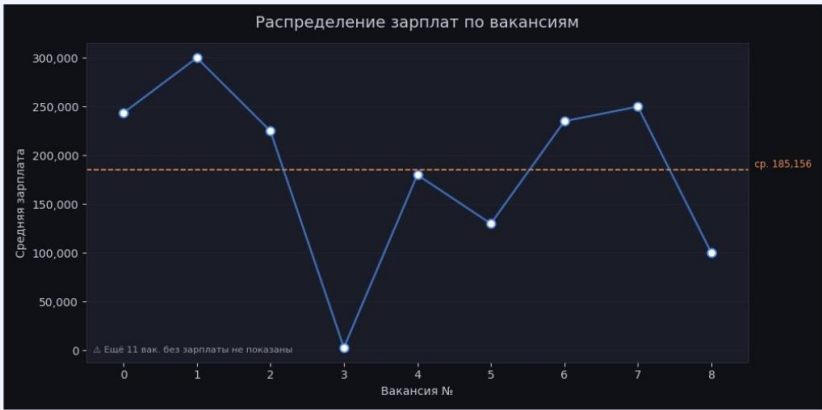
Топ-5 навыков:

- Python: 20

OK

Кирилин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Визуализация распределения зарплат



Результаты поиска / Темная цветовая схема / Сохранится в data/salary_chart.png
Кириллин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Тестирование системы

Модульные тесты (pytest)

ТС-01	Средняя зарплата при смешанных данных	●
ТС-02	Пустой DataFrame → 0.0	●
ТС-03	Все NaN → 0.0	●
ТС-04	Топ навыков — порядок (Python первый)	●
ТС-05	Ограничение top_n	●
ТС-06	Удаление вакансии без title	●
ТС-07	Удаление при salary_min < 0	●
ТС-08	Удаление при salary_max < 0	●
ТС-09	None зарплата — вакансия сохраняется	●
ТС-10	Нормализация навыков	●
ТС-11	Пустой входной список → []	●

11 / 11 тестов PASSED — время выполнения < 0.5 сек

Функциональное тестирование

FT-01	Демо-режим — полный pipeline	●
FT-02	Автоопределение города «Пенза»	●
FT-03	Некорректная зарплата → ошибка	●
FT-04	Пустые обязательные поля	●
FT-05	Повторный поиск — БД перезаписана	●
FT-06	Все вакансии без зарплат	●
FT-07	Сохранение SQLite и PNG	●
FT-08	Кросс-платформенность Windows	●
FT-09	API 403 — демо как резерв	●
FT-10	Запуск pytest — 11 PASSED	●

10 / 10 сценариев пройдены успешно

Кириллин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Заключение

01 Анализ предметной области

Изучены 4 аналога, выявлена ниша для разрабатываемого решения

02 Архитектура

7 независимых слоёв, паттерны Facade, ABC, Repository, @dataclass

03 Парсер HH.ru

Официальный API с Bearer-токеном, автоопределение города через /suggests/areas

04 Аналитика и визуализация

Средняя зарплата, топ-N навыков, scatter+line график (PNG)

05 Хранение данных

SQLite + CSV, автосоздание директорий, паттерн Repository

06 Тестирование

11 модульных тестов + 10 функциональных сценариев — все пройдены

Кириллин М.С. | ПГУ 2026 | Анализ вакансий с сайта поиска работы

Спасибо за внимание!

Готов ответить на вопросы

Студент: Кирилин Матвей Сергеевич

Тема: Анализ вакансий с сайта поиска работы

ПГУ, 2026