Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. каф. ЭВМ

\_\_\_\_\_\_\_\_\_\_ Д. И. Самаль

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

на тему

ПРОГРАММНЫЙ МОДУЛЬ АНАЛИЗА ДАННЫХ ПРОФЕССИОНАЛЬНЫХ СОЦИАЛЬНЫХ СЕТЕЙ

БГУИР ДП 1 – 40 02 01 01 003 ПЗ

Студент А. В. Белов

Руководитель М. М. Лукашевич

Консультанты:

от кафедры ЭВМ М. М. Лукашевич

по экономической части К. Р. Литвинович

Нормоконтролер А. С. Сидорович

Рецензент

МИНСК 2016

РЕФЕРАТ

Дипломный проект представлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата А1. Пояснительная записка: 93 страницы, 11 рисунков, 9 таблиц, 27 литературных источников, 4 приложения.

Ключевые слова: анализ данных, профессиональные социальные сети, data mining, Ruby, Ruby on Rails, Facebook, LinkedIn, Github, Jupyter.

Объектом исследования и разработки является получение и анализ информации, предоставляемой профессиональными социальными сетями.

Целью проекта является разработка программного модуля, позволяющего проводить получать и анализировать информацию, доступную в социальной сети Facebook, профессиональной социальной сети LinkedIn, а также сервисе совместной работы Github и обладающего возможностями для визуализации полученных результатов исследования.

При разработке использовались среда разработки Atom, язык программирования Ruby, фреймворк Ruby on Rails, база данных PostgreSQL. Для организации интерактивных вычислений применялась система Jupyter. В работе были задействованы библиотеки Koala и Octokit, предоставляющие удобные для разработчика средства взаимодействия с программными интерфейсами социальной сети Facebook и сервиса Github соответственно.

В процессе испытаний разработанного программного модуля были обработаны данные студентов кафедры ЭВМ в профессиональных социальных сетях, получена представляющая интерес информация о распределении выпускников по странам, количественному составу штатов предприятий, ценимых студентами навыков и используемых в разработке языков программирования и технологий.

Предполагается возможность практического применения разработанного программного модуля с целью получения университетом и кафедрами данных о профессиональных предпочтениях студентов для последующего повышения качества образовательных услуг и актуализации учебных планов.

Проведенное технико-экономическое обоснование показывает, что разработанный программный модуль можно считать экономически эффективным.

Дипломный проект полностью завершен, дальнейшее улучшение возможно путем увеличения количества поддерживаемых источников получения информации, а также добавления на базе созданного ядра новых способов анализа и визуализации информации.

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: нет.

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Д. И. Самаль

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2016 г.

ЗАДАНИЕ

по дипломному проекту студента

Белова Александра Владимировича

**1** Тема проекта: «Программный модуль анализа данных профессиональных социальных сетей» – утверждена приказом по университету от 22 февраля 2016 г. № 371-с.

**2** Срок сдачи студентом законченного проекта: 1 июня 2016 г.

**3** Исходные данные к проекту:

**3.1** Операционная система: Linux.

**3.2** Язык программирования: Ruby.

**3.3** Программная платформа: Ruby on Rails.

**3.4** Среда разработки: Atom.

**4** Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Технико-экономическое обоснование разработки и использования программного продукта. Заключение. Список литературы. Приложения.

**5** Перечень графического материала (с точным указанием обязательных чертежей):

**5.1** Вводный плакат. Плакат.

**5.2** Программный модуль анализа данных. Схема структурная.

**5.3** Программный модуль анализа данных. Диаграмма классов.

**5.4** Программный модуль анализа данных. Модель данных.

**5.5** Программный модуль анализа данных. Диаграмма последовательности.

**5.6** Заключительный плакат. Плакат.

**6** Содержание задания по экономической части: «Технико-экономическое обоснование разработки и использования программного продукта».

ЗАДАНИЕ ВЫДАЛ К. Р. Литвинович

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  дипломного проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Подбор и изучение литературы | 10 | 01.02 – 15.02 |  |
| Структурное проектирование | 10 | 15.02 – 29.02 |  |
| Функциональное проектирование | 20 | 29.02 – 15.03 |  |
| Разработка программных модулей | 20 | 15.03 – 15.04 |  |
| Программа и методика испытаний | 10 | 15.04 – 15.05 |  |
| Расчет экономической эффективности | 10 | 01.04 – 01.05 |  |
| Оформление пояснительной записки | 10 | 01.05 – 30.05 |  |

Дата выдачи задания: 1 февраля 2016 г.

Руководитель М. М. Лукашевич

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc451465349)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc451465350)

[1.1 Обзор существующих социальных сетей 8](#_Toc451465351)

[1.2 Обзор аналогов 13](#_Toc451465352)

[1.3 Выбор используемых технологий 14](#_Toc451465353)

[1.4 Выводы 15](#_Toc451465354)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 16](#_Toc451465355)

[2.1 Структура программного средства 16](#_Toc451465356)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 20](#_Toc451465357)

[3.1 Описание работы фреймворка Ruby on Rails 20](#_Toc451465358)

[3.2 Описание модели данных 23](#_Toc451465359)

[3.3 Описание структуры и взаимодействия между классами 31](#_Toc451465360)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 48](#_Toc451465361)

[4.1 Алгоритм получения данных Github 48](#_Toc451465362)

[4.2 Алгоритм получения данных LinkedIn 51](#_Toc451465363)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 59](#_Toc451465364)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 64](#_Toc451465365)

[6.1 Установка системы на удаленном сервере 64](#_Toc451465366)

[6.2 Установка системы локально 66](#_Toc451465367)

[6.3 Описание основных функций программного модуля 67](#_Toc451465368)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА 69](#_Toc451465369)

[7.1 Характеристика программного продукта 69](#_Toc451465370)

[7.2 Расчет сметы затрат, цены и прибыли 69](#_Toc451465371)

[7.3 Расчет экономического эффекта у пользователя 78](#_Toc451465372)

[ЗАКЛЮЧЕНИЕ 83](#_Toc451465373)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 84](#_Toc451465374)

[ПРИЛОЖЕНИЕ А 86](#_Toc451465375)

[ПРИЛОЖЕНИЕ Б 91](#_Toc451465376)

[ПРИЛОЖЕНИЕ В 92](#_Toc451465377)

[ПРИЛОЖЕНИЕ Г 93](#_Toc451465378)

# ВВЕДЕНИЕ

Социальные сети давно стали частью повседневной жизни для немалой части человечества. В частности, социальная Facebook, по последним данным, имеет ежемесячную аудиторию более полутора миллиардов человек, что не намного меньше общего числа пользователей Интернета на планете Земля. В основном, подобного рода ресурсы используются для построения социальных отношений между людьми, разделяющими общие интересы, увлечения, имеющих, в некотором смысле, схожий жизненный опыт.

В процессе мощного роста и постоянного развития на протяжении последних десяти лет социальные сети начали оказывать влияние на совершенно различные сферы деятельности человека. Примером можно считать области электронной коммерции, образования, профессиональных отношений, которые, с приходом к социальным сетям широкой популярности, претерпели значительные изменения.

К примеру, появлений социальной сети Linkedin позволило изменить процесс поиска персонала в сторону значительного его упрощения. Около полумиллиарда человек имеют профили с указанием мест работы, используемых технологий, наград и сертификатов. Соответственно, представители отдела кадров заинтересованных организаций имеют возможность найти наиболее подходящего на определенную роль кандидата и составить первичное представление о человеке, не обращаясь при этом к нему напрямую. Что, в свою очередь, позволяет увеличить количество рассматриваемых кандидатов до величин, еще совсем недавно считавшихся невозможными в принципе.

Еще одним следствием развития таких сервисов, как социальные сети, является увеличения значимости анализа данных. Объем доступной информации с течением времени экспоненциально возрастает, как и доступные вычислительные мощности, однако выделение полезных данных было и остается достаточно непростой задачей. Следует заметить, что существует заметное число инструментов и методов для обработки данных определенных типов. Однако выбор и обоснование средств для решения конкретной задачи является нетривиальным вопросом, и от этого выбора в значительной степени зависит результативность получившегося программного продукта.

Для расширения возможностей по работе с данными, ведущие игроки рынка социальных сетей предоставляет возможность получения некоторой части информации путем HTTP запросов определенной формы. Однако и это не решает проблему целиком. Практика показывает, что весьма значительная часть данных при данном способе работы недоступна, либо доступна с ограничениями. Причем это именно наиболее полезные данные о пользователе, как, например, его местоположение, места работы и обучения.

Веб-версии социальных сетей содержат некоторые инструменты для анализа данных о пользователях, обычно, в пределах заведомо ограниченной социальной группы. Однако рядовой пользователь использует различные сервисы для различных категорий задач. Например, фотографии загружаются в Instagram и Flickr, для общения используется Facebook и Вконтакте, в рамках профессиональных отношений используются Linkedin и Github, для коротких высказываний используется Twitter. Причем данные пользователя оказываются равномерно распределены между большим числом сервисов и для качественного решения задачи анализа данных необходимо сопоставить информацию из профилей, принадлежащих одному пользователю в различных социальных сетях.

Социальные сети являются прекрасным источником данных для разнообразных научных и практических исследований. Особую значимость в последнее время приобретает использование мониторинга сообщений в социальных сетях для оценки тональности отзывов о товарах, что позволяет ведущим мировым компаниям выстраивать свои взаимоотношения с пользователями на основании самых свежих тенденций.

Также одним из перспективных вариантов использования результатов анализа для учреждений образования является возможность корректировать свои учебные программы и рекламные кампании на основании данных о профессиональных достижениях и области деятельности своих выпускников.

В качестве примера грамотного использования данных, полученных из социальных сетей, можно привести опыт Стэнфордского университета, в котором существует специальная группа, состоящая из представителей факультетов, докторантов и студентов, занимающаяся исключительно аналитикой социальных сетей в применении к большому количеству областей, в том числе, к сферам школьного и университетского образования.

Задачей разрабатываемого программного модуля является облегчение сбора данных из профессиональных социальных сетей и дальнейший их анализ по определенным критериям. Получение необходимых данных реализуется частично путем HTTP запросов, определенных конкретными социальными сетями, и частично путем автоматического взаимодействия с веб-сайтами данных сетей. Второй способ, в соответствии с пользовательскими соглашениями сервисов, не является в полной степени законным, однако значительную часть информации иначе никак получить невозможно.

В качестве конкретной задачи данного программного модуля следует рассматривать сбор определенной информации о студентах специальности ВМСиС, в том числе, уже завершивших университетское обучение, на основании принадлежащих им профилей в профессиональных социальных сетях. Например, такой информацией являются текущее местоположение пользователя, перечень используемых их технологий, места работы, оценка вклада в открытое программное обеспечение. Перечисленные данные имеют первостепенное значение для оценки качества обучения и актуальности учебных планов специальности.

# **1** ОБЗОР ЛИТЕРАТУРЫ

Социальные сети уже давно и прочно вошли в повседневную жизнь людей во всем мире. Непосредственно перед проектирование программного модуля для анализа данных имеет смысл оценить текущее состояние сферы социальных сетей, выяснить положительные и отрицательные стороны для каждой отдельной сети и проанализировать их профессиональную направленность.

## **1.1** Обзор существующих социальных сетей

Анализ сравнительных характеристик основных социальных сетей (см. таблицу 1.1), доступных на настоящий момент, показал, что по целесообразности анализа информации лидируют социальные сети Linkedin и Github.

Linkedin является социальной сетью для поиска и установления деловых контактов, доступной на глобальном рынке. Актуальной данной социальной сети не вызывает сомнений, поскольку она, фактически, не имеет аналогов в своей области. Одним из наиболее заметных отличий от остальных перечисленных социальных сетей является способ установления контактов между пользователями. Linkedin требует предварительного знакомства с контактами. Также пользователь может быть рекомендован через общие контакты. Следует считать, что данные ограничения позволяют достичь большей точности в построении социального окружения пользователя. Кроме того, в результате ориентированности Linkedin на профессиональных пользователей, в нем является рекомендуемой для заполнения информация о местах работы, образования пользователя и изученных технологиях из конкретной профессиональной области. Благодаря этому, с помощью Linkedin становится возможным обработать данные, которые крайне тяжело получить любым иным способом.

Помимо использования социальной сети Linkedin профессионалами в различных областях для установления рабочих отношений, одобряется также использование контактов специалистами по подбору персонала в процессе поиска подходящих сотрудников. Для них доступны дополнительные инструменты для работы со списком контактов, однако раскрытие всего спектра возможностей требует платной подписки.

Общая же схема монетизации такова. Существует тарифный план для людей, находящихся в процессе поиска работы, стоимостью 30 долларов США в месяц. Он позволяет находиться в верхних строчках поисковой выдачи при поиске кандидатов специалистами отдела кадров. Для руководителей доступен план стоимостью 60 долларов США в месяц с расширенными возможностями по управлению профилями организаций. Специалистам отдела продаж предлагается приобрести тарифный план стоимостью 80 долларов в месяц, а специалистам отдела кадров - стоимость

Таблица 1.1 – Сравнительная характеристика социальных сетей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Facebook | Twitter | Google+ | Myspace |
| facebook.com | twitter.com | plus.google.com | myspace.com |
| 1 | 2 | 3 | 4 | 5 |
| Приблизительная месячная аудитория, пользователей | 1.59 миллиарда | 332 миллиона | 418 миллионов | 75.9 миллионов |
| Наличие мобильного приложения | Android, iOS, Nokia, Palm, Blackberry, Windows, Sidekick, Sony | Android, iOS, Blackberry, Windows | Android, iOS | iOS |
| Интеграция со сторонними сайтами | Есть | Есть | Есть | Есть |
| Обмен фотографиями | Есть | Есть | Есть | Есть |
| Обмен видео | Есть | Нет | Есть | Нет |
| Обмен сообщениями | Есть | Нет | Есть | Есть, через стороннее приложение |
| Ориентированность на профессиональную аудиторию | Нет | Нет | Нет | Нет |
| Выручка компании за 2015 год, долларов США | 17.928 миллиардов | 2.21 миллиарда | 74.5 миллиарда – общая выручка Google | 109 миллионов |
| Число работников | 12691 | 3638 | 61814 | 200 |
| Год запуска | 2004 | 2006 | 2011 | 2003 |

*Продолжение таблицы 1.1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Linkedin | Github | Вконтакте | Одноклассники |
| linkedin.com | github.com | vk.com | ok.ru |
| 1 | 6 | 7 | 8 | 9 |
| Приблизительная месячная аудитория, пользователей | 400 миллионов | 10 миллионов | 314 миллионов | 200 миллионов |
| Наличие мобильного приложения | Android, iOS | Android, iOS | Android, iOS | Android, iOS |
| Интеграция со сторонними сайтами | Есть | Нет | Есть | Есть |
| Обмен фотографиями | Нет | Нет | Есть | Есть |
| Обмен видео | Нет | Нет | Есть | Есть |
| Обмен сообщениями | Есть | Нет | Есть | Есть |
| Ориентированность на профессиональную аудиторию | Да | Да | Нет | Нет |
| Выручка компании за 2015 год, долларов США | 2.99 миллиарда | Нет данных | 90 миллионов | Нет данных |
| Число работников | 8735 | 467 | 500 | 100 |
| Год запуска | 2003 | 2008 | 2006 | 2006 |

120 долларов в месяц, причем для последнего тарифа доступны расширенные возможности по поиску кандидатов и установлению контактов в обход общепринятых ограничений.

Для разработчиков доступен API (программный интерфейс), однако политика сервиса в этом отношении вызывает некоторые вопросы. Так, вместо планомерного расширения доступных разработчикам возможностей, происходит их сокращение [1]. Для выполнения полного спектра действий и доступа к информации пользователей без ограничений предлагается заключать с компанией партнерское соглашение, которое, благодаря ценовой политике, доступно только крупным организация, к примеру, Samsung, WeChat и Evernote.

В результате при проектировании программного модуля анализа данных, для Linkedin следует ожидать использования методов сбора информации, не являющихся в полной степени законными с точки зрения пользовательского соглашения. А именно, так как доступ к данным через API ограничен, следует воспользоваться возможностью анализа содержимого веб-страниц и автоматизации действий пользователя, в результате чего становится возможным получение всех необходимых сведений о пользователе.

Кроме сервиса Likendin, который позиционируется именно как социальная сеть, следует рассматривать в качестве источника данных также Github - крупнейший сервис для хранения и совместной работы над IT-проектами. Хотя рассмотрение средства совместной разработки в качестве социальной сети может показаться весьма неожиданным допущением, данный подход не нов и имеет своих сторонников [2]. Из таблицы 1.1 можно заметить, что у Github отсутствует типичная для социальных сетей функциональность обмена видеофайлами, фотографиями и сообщениями. Однако присутствует возможность установления профессиональных контактов и оценки результатов работы других пользователей. Установление контактов и оценка труда других пользователей без преувеличения являются одними из основных причин взрывного роста социальных сетей как класса клиент-серверных приложений.

Github имеет одно из лучших API среди всех представленных сервисов, причем именно здесь оно наиболее дружелюбно к независимым разработчикам. Без авторизации позволяется выполнить 60 запросов к системе за час, с авторизацией с помощью существующего аккаунта число запросов возрастает до 5000, чего вполне достаточно для получения необходимых данных, особенно при наличии возможности задать извлечение информации по определенному расписанию. Все эти возможности доступны бесплатно. При заключении дополнительных соглашений на платной основе возможно дальнейшее увеличение лимитов, однако, как уже было сказано, для большинства применений достаточно базового набора возможностей.

Github является одной из основных площадок для работы над свободным программным обеспечением и опыт показывает, что значительная часть студентов сталкивается с необходимостью регистрации в данной системе либо в процессе обучения в университете, либо с началом трудовой деятельности по специальности. Актуальность использования данной площадки обеспечивает наличие недоступных нигде более данных о реально используемых человеком в рабочем процессе технологий, а также процентное соотношение по ним. Также становится возможным оценить вклад человека в движение свободного программного обеспечения на основании оценок других пользователей. В применении к данному конкретному сервису используются бинарные оценки для проектов - так называемые Stars, соответствующие числу заинтересованных в разработке проекта людей.

Помимо сервисов Github и Linkedin, которые являются профессионально-ориентированными, следует включить в рассмотрение так социальную сеть Facebook. Facebook является общепризнанным лидером сферы социальных сетей, при сборе данных о конкретном человеке крайне высока вероятность обнаружить его аккаунт на данной площадке. Несмотря на то, что Facebook не способен предоставить специфическую рабочую информацию о пользователе, что является следствием общего характера предоставляемых социальной сетью возможностей, следует заметить, что в плане получения информации о местоположении пользователей данных подобной полноты не может предоставить не одна другая социальная сеть из рассмотренных. Кроме местоположения, Facebook также следует использовать для получения списка пользователей определенной социальной группы, например, студентов конкретной специальности, что становится возможным благодаря установившемуся в процессе исторического развития широкому применению возможностей групповых коммуникаций. Однако следует заметить, что данные о списке пользователей следует дублировать, если это возможно, сведениями из иных официальных источников для повышения точности результатов.

В плане удобства работы разработчика с информацией, размещенной в социальной сети, Facebook находится на середине условной шкалы между Linkedin и Github. С помощью API возможно получения перечня членов определенного сообщества и ссылок на их профили, однако из самих профилей доступна только самая общая информация, вроде имени и фамилии пользователя. Ничего больше без особого разрешения пользователя через API получить нельзя. Однако с помощью автоматизации работы с веб-ресурсом, описанной применительно к Github, становится возможным получение информации о текущем местоположении пользователя с точностью до города.

О не рассмотренных социальных сетях из списка следует сказать, что их полезность для анализа вызывает справедливые сомнения, так как данные либо дублируются, либо не имеют непосредственного отношения к профессиональной сфере, либо извлечение полезной информации представляет значительную трудность. Кроме того, использование региональных социальных сетей не является в общем случае предпочтительным - основная масса пользователей имеет и периодически использует аккаунты сразу в нескольких сервисах подобного плана, причем как минимум часть из них будут являться глобальными, что делает их более предпочтительными в качестве источника данных для работы модуля анализа.

Следует также отметить наличие у всех сервисов схожего назначения приложений для мобильных платформ. Современная реальность такова, что все увеличивающаяся часть пользователей использует именно мобильный доступ к социальным сетям [3].

## **1.2** Обзор аналогов

Несмотря на тот факт, что анализ данных профессиональных социальных сетей обычно является задачей, решаемой в случае каждого конкретного продукта индивидуально, можно указать ряд аналогов, обладающих возможностями, схожими в деталях в возможностями разрабатываемого программного модуля. Следует заметить, что анализ данных профессиональных социальных сетей является частным случаем анализа данных социальных сетей общего назначения, а программный аналоги программного модуля следует рассматривать в составе готовых продуктов.

Brandwatch – сервис социальной аналитики, позиционируется как решение для интернациональных компаний благодаря поддержке при анализе 25 языков, в том числе и русского. Применяется для анализа в сферах здравоохранения, спорта, электронной коммерции и демографического анализа. Также имеется информация о клиентах, использующих Brandwatch для работы с рынком вакансий, что отчасти схоже с задачами Linkedin. Плюсом сервиса является доступность для анализа данных, накопленных с 2010 года, что зачастую бывает крайне необходимо для понимания тенденций. Основная часть кода, относящегося непосредственно к аналитике, написана на языке Java с использованием NoSQL баз данных. Визуализация данных реализована с использованием веб-технологий на базе Javascript библиотек визуализации данных [4].

33Across – позиционирует себя как платформа для анализа социального трафика и монетизации контента. Стек технологий включает в себя Java, Ruby, Ruby on Rails, Hadoop. Возможным минусом следует считать узкую направленность сервиса, однако, фокусирование на одном направлении позволяет достичь в нем лучших результатов [5].

Hootsuite – аналитика социальной активности. По сути, позволяет оценивать тональность реакции аудитории на публикации представителей организаций-заказчиков. В разработке используется Python, нереляционные базы данных и хранилища ключ-значения, к примеру, Redis и Memcached. Аналитическая часть использует Scala и систему передачи сообщений Akka [6].

Moz – оценка влияния поведения компании в социальных сетях на результаты поискового продвижения (SEO). Использует Node.js, NoSQL базы и кластерную обработку данных. Имеет отличный модуль визуализации, однако анализ данных весьма узконаправленный [7].

## **1.3** Выбор используемых технологий

Из рассмотренных аналогов следует сделать вывод о типичном для отрасли технологическом стеке.

Для хранения данных практически в каждом из сервисов используются нереляционные базы данных. Выбор в пользу именно этого типа хранилища сделан из-за невозможности полноценно отобразить собранные данные на реляционную модель без времязатратной предварительной обработки. Однако в паре с NoSQL базами используются также и обычные реляционные базы данных, которые значительно лучше подходят для хранения результата аналитики.

Тем не менее, использование нереляционных баз данных для данного дипломного проекта не планируется. Сфера применения NoSQL решений и стандартных реляционных баз данных в большей степени совпадают. А значит, имеет смысл выбрать в качестве предпочтительной базы данных одно из проверенных временем универсальных решений, например, PostgreSQL, который также имеет некоторые возможности, свойственные нереляционным базам данных.

Непосредственно аналитика базируется на модели кластерной обработки данных, что приносит в технологический стек фреймфорк Hadoop и языки Java, Scala и, в некоторых случаях, Python. Ruby также используется, однако в меньшем числе случаев и, скорее, как вспомогательный язык.

Визуализация данных в основном реализуется с помощью библиотек визуализации данных на основе JavaScript, например, d3js. Впрочем, это характерно для динамической инфографики в целом, JavaScript является гораздо более предпочтительным решением относительно системных библиотек визуализации данных.

Для разрабатываемого программного модуля не ставится жестких временных ограничений по времени сбора и обработки информации, но, исходя из специфики разработки дипломного проекта, временные ограничения накладываются на время разрабоки. С учетом указанных факторов наиболее подходящим является язык высокого уровня Ruby. Ruby – интерпретируемый язык программирования высокого уровня. Обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, «сборщиком мусора» и многими другими возможностями, поддерживающими много разных парадигм программирования. Ruby был задуман в 1993 японцем Юкихиро Мацумото, стремившимся создать язык, совмещающий все качества других языков, способствующие облегчению труда программиста [8].

К основным достоинствам языка Ruby можно отнести открытый исходный код [9]. Преимущество такого рода программ является возможность просмотра, изучения и изменения, что позволяет любому желающему принять участие в доработке самой открытой программы, использовать код для создания новых программ и исправления в них ошибок.

Важной чертой Ruby является гибкая возможность метапрограммирования. Тот факт, что Ruby – интерпретируемый язык, позволяет создавать выполняемый код программным образом, что значительно сокращает время реализации сложного функционала и благоприятно влияет на стоимость разработки проекта в целом.

Также следует отметить использование в языке динамической типизации [10]. Этот приём широко используется в языках программирования и позволяет связывать переменную с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

Еще одна особенность языка – широкая стандартная библиотека классов. Стандартная библиотека классов представлена двумя библиотеками: ruby-core и ruby-stdlib. В совокупности они представляют собой реализацию различных популярных алгоритмов и решают наиболее популярные задачи.

Сторонние библиотеки, которые носят название gem, делают Ruby уникальным инструментом с огромными возможностями. На данный момент доступно более 2.5 миллионов различных библиотек [11]. Таким образом, среди всего разнообразия бесплатных и открытых библиотек с большой долей вероятности можно найти решению вашей текущей проблемы, что значительно ускоряет скорость разработки.

Таким образом, выбранные технологии покрывают все основные запросы проекта и позволяют завершить его разработку с привлечением минимально возможного количества ресурсов и в минимальный срок.

## **1.4** Выводы

После проведения анализа можно сделать вывод, что имеющиеся аналоги имеют значительно усложненную структуру и решают лишь ограниченный спектр задач. Было решено создать программный модуль, который позволит решить проблему анализа данных профессиональных социальных сетей без необходимости использования дорогостоящих кластерных систем (см. плакат ГУИР.400201.003 ПЛ1) и иметь функционал:

* сбора данных из перечисленных социальных сетей;
* анализа собранных данных;
* формирования отчетов.

# **2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Разрабатываемый проект необходимо разбить на отдельные логически взаимосвязанные модули, что является необходимым условием для обеспечения гибкости структуры программного модуля. При данном подходе становится возможной выборочная модернизация отдельных частей программного кода, с минимальным влиянием на остальные части проекта, либо, в идеальном случае, вовсе без их изменения.

Для определения логических модулей необходимо в первую очередь определить основные возможности программного модуля:

* возможность авторизации в социальных сетях, определенных ранее;
* возможность сбора данных из социальных сетей;
* анализ собранных данных различными способами;
* дополнение собранных данных о местоположении пользователя географическими координатами;
* возможность импорта дополнительных данных для облегчения задачи анализа;
* экспорт полученных результатов для дальнейшего использования;
* визуализация данных путем генерации таблиц и графиков;
* формирование отчетов по результатам анализа данных;

## **2.1** Структура программного средства

В соответствии с методологией системного подхода в разработке архитектуры, программное средство разбивается на совокупность сущностей, представленных на структурной схеме (см. чертеж ГУИР.400201.003 С1).

Каждый логический модуль может решать одну или несколько из вышеперечисленных задач. Следует заметить, что перечисленные задачи транслируются в систему логических модулей в определенной степени однозначным образом. Кроме того, каждый из модулей связан с одним или несколькими другими модулями, что обеспечить работоспособность системы в целом. Связь, как правило, реализуется посредством обмена данными между модулями в различных форматах и путем использования данных одного модуля другим. Обобщенная структурная схема программного модуля изображена на рисунке 2.1.

Как можно заметить из представленной структурной схемы, в программном модуле удалось выделить 10 подмодулей. Следует заметить, что, несмотря на определение программного модуля, база данных не является внешним подмодулем по отношению к нему. Наиболее значимыми и объемными модулями проекта являются модули анализа данных и синтаксических анализаторов. Оставшиеся модули обеспечивают предобработку и постобработку данных, а также импорт и экспорт и являются не столь критичными для функционирования системы.



Рисунок 2.1 – Обобщенная структурная схема

*База данных* представляет собой хранилище для данных, полученных при сборе информации их социальных сетей, а также для данных, полученных в результате анализа. Поскольку дипломный проект представляет собой именно программный модуль, предназначенный для интеграции в программное средство, база данных является внешним по отношению к нему модулем. Как было определено в разделе 1.3 пояснительной записки, предпочтительным является использование реляционных баз данных, в частности, PostgreSQL [12]. Преимуществами PostgreSQL являются, в частности:

* поддержка баз данных практически ничем не ограниченного размера;
* мощные и надежные механизмы транзакций и репликации;
* дополнение собранных данных о местоположении пользователя географическими координатами;
* расширяемая система встроенных языков программирования;
* наследование;
* широкие возможности масштабирования.

*Модуль импорта* предназначен для предоставления дополнительной информации для анализа вручную. Например, для варианта с анализом профилей студентов определенной специальности в профессиональных социальных сетях имеет смысл предоставить дополнительную информацию в виде списков студентов с указанными годами обучения и, если это возможно, оценками, полученными в процессе обучения, так как эти данные невозможно получить иным способом, однако они могут оказаться весьма полезными для анализа и поиска закономерностей. Предпочтительным форматом для предоставляемых данных является формат значений с разделением запятыми (CSV). Данный формат является одним из наиболее простых и быстрых в обработке и обладает минимальной избыточностью за счет отсутствия дополнительной стилистической информации, как это происходит, например, в форматах MS Excel. Несмотря на то, что не все виды данных удобно представлять в формате CSV, для задач настоящего проекта он подходит целиком и полностью.

*Модуль предобработки данных*, как можно понять из названия, выполняет функцию преобразования сырых данных, полученных в результате сбора информации из социальных сетей, в вид, пригодный для анализа. Например, информация из социальных сетей Facebook и Linkedin представляет собой HTML теги, извлеченные из тела ответа сервера, которые слабо поддаются анализу. Однако после извлечения из них полезной информации и ее группировки применить методы анализа данных становится значительно проще.

*Модуль геоданных.* Для получения информации о местоположении пользователей планируется использовать социальную сеть Facebook и, частично, Linkedin. Однако эта информация представляет собой лишь названия населенных пунктов, что не позволяет оценить, к примеру, степень распределенности пользователей по различным странам и континентам. Значительный интерес представляют именно географический координаты пользователя. Для их получения следует использовать существующие сервисы геокодинга, в частности, сервис, предоставляемый компанией Google. Использование данного сервиса позволяет осуществлять так называемый прямой геокодинг – преобразование названия населенного пункта в набор географических координат.

*Модуль авторизации*. Использование социальных сетей, слабо поддерживающих получение данных через запросы, придает определенную специфику модулю авторизации. Помимо авторизации, собственно, в системах API социальных сетей, следует озаботиться автоматизацией действий пользователя при авторизации через веб-сайт, что позволит получать информацию путем синтаксического анализа веб-страниц. Данные условия авторизации пользователей не одобряются компаниями, предоставляющими доступ к социальным сетям, однако наличие синтаксических анализаторов веб-страниц не оставляет выбора и авторизация путем автоматизации действий пользователя является единственным приемлемым вариантов в создавшейся ситуации.

*Модуль синтаксических анализаторов* является, наряду с модулем анализа данных, одной из наиболее сложных и значимых частей проекта. Для выполнения сбора данных из профессиональных социальных сетей, которые невозможно получить путем отправки запросов определенного вида в систему API, предоставляемого сервисом социальной сети, следует использовать синтаксический анализ веб-страниц сервиса. Зачастую именно это позволяет получить наиболее ценные данные для последующего анализа. Тем не менее, следует заметить, что данная практика является не вполне законной с точки зрения пользовательских соглашений социальных сетей. Таким образом, с точки зрения социальной сети, пользователем производится просмотр страниц. Однако в процессе просмотра происходит анализ полученной в ответе сервера информации и выделение из нее необходимых фрагментов. Для различных социальных сетей требуются различные синтаксические анализаторы, поскольку формат получаемых данных в значительной степени отличается.

*Модуль анализа данных* – второй важнейший модуль проекта. Недостаточно просто получить сырые данные и выполнить их предобработку. Конечной целью всего процесса является анализ полученных данных – извлечение из них знаний в широком смысле, с использованием определенных математических методов и вычислительных алгоритмов. К примеру, данные о навыках пользователей, полученные с использованием Linkedin, следует обработать с целью получения общей статистики по навыкам, распределения навыков в зависимости от годов обучения студентов, оценить их правдоподобность в совокупности с информацией об открытых проектах пользователя на Github.

*Модуль экспорта* предназначен для вывода данных, полученных в результате анализа, во внешние системы. Для экспорта могут использоваться различные форматы, однако следует учесть, что наиболее простым и быстрым в обработке форматом, как для импорта, так и для экспорта, является формат CSV. Следует заметить, что для экспорта формат значений с разделением запятыми подходит все же больше, чем для импорта, что связано с характером экспортируемых данных. Экспортируемая статистика гораздо лучше описывается реляционной моделью, чем сырые данные, присутствующие на входе системы.

*Модуль оформления отчетов*. Экспорт данных служит для вывода сырых данных, полученных в результате анализа. Однако экспортируемые данные больше подходят для использования в программной системе, содержащей модуль анализа данных профессиональных социальных сетей. Для ознакомления же пользователя вышеописанной системы с результатами анализа служит модуль оформления отчетов, который позволяет оформлять результаты в виде, пригодном для рассмотрения человеком. Предпочтительным форматом для создания отчета является формат PDF. Отчеты содержат результаты анализа, отформатированные и снабженные графическими данными.

*Модуль визуализации данных* служит для генерации графиков и таблиц различных видов на основании результатов анализа. Полученные графики следует использовать для отображения результатов в системе, в которую подключен программный модуль, являющийся целью настоящего дипломного проекта. Предпочтительный вариант использования – создание собственного интерфейса на основе результатов анализа данных, выполненного программным модулем.

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробным образом функционирование разрабатываемого программного модуля. На стадии выбора технологий язык Ruby и платформа Ruby on Rails были отмечены как технологии, наиболее соответствующие поставленным задачам и позволяющие реализовать весь необходимый для дипломного проекта функционал с наибольшим удобством для разработчика. Следует заметить, что использование фреймворка Ruby on Rails, равно как и любого другого фреймворка, вносит в проект определенные архитектурные ограничения. Архитектуру разрабатываемого программного модуля следует основывать на архитектуре Ruby on Rails, учитывая все его основные особенности и возможности. Для достижения лучшего понимания архитектуры модуля следует, в первую очередь, подробно остановиться на архитектуре самого фреймворка Ruby on Rails.

## **3.1** Описание работы фреймворка Ruby on Rails

Фреймворк Ruby on Rails, написанный на языке программирования Ruby, поощряет и упрощает следование определенному набору принципов разработки программного обеспечения [13]:

* принцип Do not repeat yourself (DRY), не повторяй себя. Предполагается, что во всех случаях, где это возможно, уже написанный код не должен повторяться;
* принцип Convention over configuration, соглашение по конфигурации. По умолчанию используются соглашения по конфигурации, типичные для большинства приложений. И только в нестандартных случаях конфигурация явно определяется.

Основными компонентами любого приложения, реализованного на платформе Ruby on Rails, являются модель, представление и контроллер, что соответствует схеме MVC [14]. Таким образом, все взаимодействие с пользователем сосредоточено в трех разделенных между собой компонентах, что, следует заметить, является хорошей практикой в сфере разработки программного обеспечения и, в частности, в сфере разработки веб-приложений.

Модель – стандартное место для размещения бизнес-логики приложения, предоставляет объектно-ориентированное отображение данных. Основным назначением модели считается загрузка и сохранение данных, другими словами, модели отводится роль интерфейса между базой данных и остальным приложением.

Объектно-ориентированное отображение (Object-Relational Mapping, ORM) – это технология, позволяющая ставить сложные объекты приложения в соответствие с реальными таблицами базы данных. Следует заметить, что в сочетании с системами ORM используются, прежде всего, реляционные базы данных. Использование ORM позволяет значительно сократить объем программного кода, необходимый для взаимодействия со свойствами объектов базы данных и отношениями между ними. По умолчанию в Ruby on Rails используется ORM ActiveRecord [15]. Так же существуют альтернативные реализации ORM, пригодные к использованию вместе с фреймворком Ruby on Rails, как, например, sequel. Все модели в Ruby on Rails по умолчанию наследуются от базового класса ActiveRecord::Base.

Следующим по списку компонентом схемы MVC является представление. Представление служит для конструирования пользовательского интерфейса приложения на основании данных, поступающих из контроллера. Связь между моделью и контроллером является двухсторонней, запросы на изменение данных передаются из представления в контроллер, для самого же представления изменение модели не одобряется.

В Ruby on Rails представление реализовано через шаблоны ERB, представляющие собой HTML файлы с возможностью интерполяции кода на Ruby. Код на Ruby, входящий в состав шаблона, интерпретируется, включается в состав HTML и направляется пользователю приложения в ответ на запрос.

Последним элементом модели MVC является контроллер, который инкапсулирует в себе логику, необходимую для обработки запроса пользователя на стороне сервера. Все контроллеры в Ruby on Rails наследуются от базового класса ActionController::Base. Контроллер формирует представление, необходимое для ответа на запрос пользователя и вызывает методы модели, необходимые для манипуляции данными. Чаще всего одно представление соответствует одному методу контроллера, называемому действием.

Особое внимание необходимо уделить такому компоненту фреймворка, как Router. Основной его задачей является определение соответствия между входящим HTTP-запросом и действием в контроллере. Например, следующий код отвечает за маршрутизацию запросов на импорт в разрабатываемом программном модуле:

Rails.application.routes.draw do

root to: “import#index”

resources :import, only: [:index] do

collection do

post :upload

end

end

end

Данный код является конфигурация для роутера приложения, в котором определяются действия, выполняемые при поступлении HTTP-запроса от пользователя. Таким образом, при запросе, содержащем пустой путь либо путь к импорту необходимо передать обработку запроса методу index в контроллере ImportController. Похожим образом описываются и все остальные маршруты в приложении.

Для запуска веб-приложения, написанного с использованием фреймворка Ruby on Rails в режиме эксплуатации, следует использовать веб-сервер в комбинации с сервером приложения. В режиме разработки веб-сервером по умолчанию является WEBrick, в режиме реальной эксплуатации рекомендуется использовать nginx либо Apache в связке с модулями Phusion Passenger и Unicorn.

Среди поддерживаемых Ruby on Rails баз данных встречаются практически все массово используемые базы: MySQL, MSSQL, PostgreSQL, Firebird, DB2, Oracle. Для отладки программы в режиме разработки удобно использовать базу SQLite, которая является наиболее удобно переносимой из всех баз, так как представляет собой одиночный файл, размещенный в одной директории с приложением, что позволяет также держать его под контролем систем контроля версий.

Основными операционными системами, используемыми во всем мире для разработки приложений на Ruby on Rails, являются OS X и различные дистрибутивы Linux. Следует заметить, что использование для этих целей Windows хоть и не поощряется, но, в целом, возможно. Существуют специально созданные образы VMware с готовой средой для разработки и запуска Ruby on Rails приложений, которые можно подключить к используемому серверу виртуальных машин или развернуть в предлагаемой многими сервисами облачной среде [16].

Одним из основных факторов успеха фреймворка Ruby on Rails, повлиявшего также на интерес к языку программирования Ruby, является наличие мощной экосистемы библиотек расширения – гемов. В мире Ruby on Rails существует огромное количество библиотек для решения задач, возникающих при разработке веб-приложений. Следует заметить, что это компенсируется относительно малым числом библиотек в области вычислений и статистики, в чем традиционно лучшим выбором является язык Python.

Для разработки AJAX-приложений в Ruby on Rails по умолчанию предлагается использовать библиотеку jQuery, но гибкая структура фреймворка позволяет сделать выбор и в пользу иных вариантов. Интересным фактом является то, что в более ранних версиях Ruby on Rails (до версии 3.1) не jQuery, а Prototype являлся javascript-фреймворком по умолчанию.

Среди всего многообразия фреймворков для веб-разработки Ruby on Rails обладает, по общему мнению, наиболее удобными инструментами для разработки и в максимальной степени сосредоточен именно на удобстве разработчика программных систем.

## **3.2** Описание модели данных

Несмотря на то, что обычно программные модули не содержат в своем составе баз данных, для настоящего проекта альтернативы включению базы в состав разрабатываемого программного модуля не было альтернативы. Данные, полученные из профессиональных социальных сетей необходимо сохранить в удобном для обработки виде, и лучше всего для этого подходят реляционные базы данных. На этапе выбора технологий было определено, что наиболее подходящей базой для использования является PostgreSQL. Описание структуры базы данных является ключевым моментом для понимания работы программного модуля в целом и позволяет получить представление о данных, которые в дальнейшем подвергнутся обработке и анализу средствами разрабатываемого модуля. Следует отметить, что в большинство таблиц включены поля created\_at и updated\_at, которые соответствуют времени создания и изменения текущей записи в таблице. Также данные поля необходимы для корректного функционирования ActiveRecord, но следует заметить, что для некоторых таблиц их создание является совершенно излишним. Рассмотрим детально структуру ключевых таблиц и связи между ними, приведенную на рисунке 3.1 и на схеме модели данных (см. чертеж ГУИР.400201.003 РР.2).

**3.2.1** Таблица users

Данная таблица является основной для программного модуля. Таблицы, содержащие импортированные данные, такие как imported\_data, facebook\_data, linkedin\_data и github\_data связаны через внешние ключи с таблицей users, что позволяет использовать ее для упрощения запросов, совершаемых с целью сбора статистики по используемым источникам данных.

Поля:

* id – первичный ключ;
* name – транслитерированное имя пользователя, полученное из связанной таблицы;
* facebook – признак наличия данных, импортированных из социальной сети Facebook;
* linkedin – признак наличия данных, полученных из профессиональной сети LinkedIn;
* linkedin\_scraped – признак завершенного импорта из LinkedIn;
* github – признак наличия данных, собранных через API платформы для совместной работы Github;
* github\_scraped – признак завершения импорта из Github;
* import – признак импорта из csv файлов с данными о выпускниках.



Рисунок 3.1 – Модель данных

Следует заметить, что поля linkedin\_scraped и github\_scraped не аналогичны по назначению другим признакам и используются для проверки факта осуществления попытки импорта из определенного источника, что позволяет не осуществлять дважды импорт отсутствующих в источнике пользователей. Все поля признаков являются обязательными.

**3.2.2** Таблица imported\_data

Данная таблица предназначена для хранения информации о студентах определенного года выпуска, импортированных в систему с помощью загрузки файла формата csv. Используется для дальнейшего поиска импортированных студентов в профессиональных социальных сетях.

Поля:

* id – первичный ключ;
* user\_id – внешний ключ, связь с главной таблицей пользователей;
* first\_name – имя студента;
* last\_name – фамилия студента;
* middle\_name – отчество студента;
* encoded\_name – транслитерированное полное имя;
* graduated\_at – дата выпуска в формате даты.

Содержит индекс по полю graduated\_at для ускорения поиска студентов по году выпуска, что является часто встречаемой задачей при анализе данных.

**3.2.3** Таблица facebook\_data

Данная таблица служит для хранения данных, импортированных из социальной сети Facebook.

Поля:

* id – первичный ключ;
* user\_id – внешний ключ, связь с главной таблицей пользователей;
* original\_name – имя пользователя, указанное на странице в социальной сети;
* encoded\_name – транслитерированное имя;
* facebook\_id – идентификатор профиля в сети Facebook;
* address\_geocoded – признак выполнения обработки адреса, полученного со страницы пользователя, обязательное поле.

**3.2.4** Таблица github\_data

Данная таблица служит для хранения данных, импортированных из сервиса совместной работы Github. Является одним из основных источников данных для анализа, так как содержит значительное количество информации о профессиональной деятельности пользователя.

Поля:

* id – первичный ключ;
* user\_id – внешний ключ, связь с главной таблицей пользователей;
* login – идентификатор пользователя на Github;
* name – имя, указанное пользователем в профиле;
* email – адрес электронной почты, пригодный для рассылки спама;
* num\_repos – число репозиториев, принадлежащих пользователю (включая приватные);
* num\_gists – число файлов, сохраненных в специальном сервисе Github Gists для хранения отдельных файлов, не входящих в проект;
* num\_followers – число пользователей, подписанных на обновления данного профиля;
* num\_following – число подписок на обновления профилей других пользователей;
* blog – адрес блога пользователя.

**3.2.5** Таблица linkedin\_data

Данная таблица служит для хранения данных о пользователе, импортированном из профессиональной социальной сети LinkedIn.

Поля:

* id – первичный ключ;
* user\_id – внешний ключ, связь с главной таблицей пользователей;
* profile\_id – идентификатор пользователя в социальной сети LinkedIn, является также ссылкой на его публично доступный профиль;
* first\_name – имя пользователя;
* last\_name – фамилия пользователя;
* encoded\_name – транслитерированное полное имя;
* title – текущий статус пользователя, для LinkedIn обычно состоит из названия текущей позиции и компании-работодателя;
* connections – число профессиональных контактов пользователя;
* country – страна проживания в настоящее время.

Поле country в дальнейшем используется для создания адреса. Следует, однако, заметить, что пользователи в LinkedIn редко предоставляют данные о стране проживания, поле используется лишь эпизодически.

**3.2.6** Таблица skills

Данная таблица служит для хранения навыков, присутствующих в системе LinkedIn и используемых пользователями для заявления о наличии экспертизы в определенной сфере.

Поля:

* id – первичный ключ;
* name – название навыка.

Таблица skills является первой из двух таблиц, не имеющих полей created\_at и updated\_at в связи с особенностью хранимых данных. Для поля name создан индекс, облегчающий и ускоряющий поиск навыка по названию, что является наиболее частой, по большому счету, единственной операцией с данной таблицей.

**3.2.7** Таблица linkedin\_data\_skills

Связующая таблица между навыками и данными из LinkedIn. Является необходимым элементом для работы некоторых статистических методов.

Поля:

* id – первичный ключ;
* linkedin\_data\_id – внешний ключ на таблицу, содержащую данные импорта из LinkedIn;
* skill\_id – внешний ключ на таблицу, содержащую навыки из системы LinkedIn.

**3.2.8** Таблица repositories

Данная таблица содержит данные, относящиеся к репозиторию пользователя из системы совместной работы Github. Используется для выполнения аналитики по популярности языков программирования и оценки активности участия пользователя в движении Open Source.

Поля:

* id – первичный ключ;
* github\_data\_id – внешний ключ на таблицу, содержащую данные импорта пользователя Github, владельца репозитория;
* primary\_language\_id – внешний ключ на таблицу языков программирования, основной язык, используемый в данном репозитории;
* full\_name – ссылка на репозиторий, содержит в себе имя пользователя и название проекта;
* repo\_id – идентификатор репозитория на Github;
* forks – число форков репозитория;
* issues – число заявок о проблемах и предложениях;
* watchers – число пользователей, подписанных на обновления данного репозитория;
* subscribers – число подписчиков репозитория на Github;
* stargazers – число пользователей, отметивших репозиторий. Одна из наиболее интересных величин для анализа;
* fork – признак того, что репозиторий является форком другого репозитория. Обычно форки создаются для участи в разработке свободного ПО.

**3.2.9** Таблица languages

Данная таблица служит для хранения названий языков программирования, используемых в проектах, размещенных на Github. Используется аналогично таблице skills и имеет сходную структуру.

Поля:

* id – первичный ключ;
* name – название языка программирования.

Таблица languages является второй из двух таблиц, не имеющих полей created\_at и updated\_at в связи с особенностью хранимых данных. Для поля name создан индекс, облегчающий и ускоряющий поиск навыка по названию, что является наиболее частой, по большому счету, единственной операцией с данной таблицей.

**3.2.10** Таблица languages\_repositories

Связующая таблица между языками программирования и репозиториями из системы совместной работы над проектами Github. Содержит, в дополнение к внешним ключам, число строк конкретного языка в конкретном репозитории, что является неплохим источником данных для анализа в будущем.

Поля:

* id – первичный ключ;
* language\_id – внешний ключ на таблицу с языками программирования;
* repository\_id – внешний ключ на таблицу, содержащую репозиторий, полученный через API Github;
* loc – число строк с данным языком в репозитории. Имеет формат, позволяющий хранить в базе PostgreSQL большие числа.

**3.2.11** Таблица companies

Данная таблица служит для хранения данных о компания, в которых работают на настоящий момент или работали в прошлом пользователи, обнаруженные программным модулем в системе LinkedIn. Служит источником аналитики по распределению компаний на мировой карте.

Поля:

* id – первичный ключ;
* name – название компании;
* description – публично доступное описание компании с Linkedin. Имеет тип text, хорошо подходящий для хранения больших объемов текста;
* linkedin\_company\_url – ссылка на страницу компании на LinkedIn;
* website – ссылка на сайт компании;
* company\_size – число сотрудников в компании. Несмотря на название, подразумевающее численный тип, поле является строковым, что связано с особенностями возвращаемого LinkedIn интервального значения;
* company\_type – форма собственности компании;
* industry – сфера бизнеса компании;
* founded – дата основания компании, с точностью до года;
* country – страна, в юрисдикции которой находится компания в настоящий момент.

**3.2.12** Таблица companies\_linkedin\_data

Связующая таблица между компаниями и данными пользователя LinkedIn. По своей сути отображает отношения между работником и работодателем. Что интересно, данная таблица не является единственной связующей таблицей этой связки, похожую функцию может выполнять таблица positions.

Поля:

* id – первичный ключ;
* company\_id – внешний ключ на таблицу с компаниями;
* linkedin\_data\_id – внешний ключ на таблицу, содержащую данные, импортированные из LinkedIn.

**3.2.13** Таблица positions

Данная таблица, помимо своей основной функции хранения данных а рабочей позиции, является также второй связующей таблицей между companies и linkedin\_data.

Поля:

* id – первичный ключ;
* company\_id – внешний ключ на таблицу с компаниями;
* linkedin\_data\_id – внешний ключ на таблицу, содержащую данные, импортированные из LinkedIn;
* title – наименование рабочей позиции;
* duration – длительность работы на текущем месте в виде строки с описанием на естественном языке;
* start\_date – дата начала работы в компании;
* end\_date – дата конца работы в компании;
* current – признак того, что пользователь работает на данной позиции в настоящий момент.

**3.2.14** Таблица addresses

Данная таблица хранит адреса в расширенном формате. В то время, как из некоторых других таблиц возможно получить положение с точности до страны, таблица адресов содержит адрес с точностью до города с указанием географических координат. Связана с пользователями и компаниями через полиморфную связь.

Поля:

* id – первичный ключ;
* addressable\_id – внешний ключ полиморфной связи;
* addressable\_type – тип полиморфной связи;
* city – город;
* country – страна;
* lat – географическая широта. Используется формат с большой точностью;
* lng – географическая долгота. Так же, как и с случае широты, используется числовой формат с большой точностью;
* geocoded – признак завершения геокодинга, выполняемого с целью получения широты и долготы на основании известного текстового адреса;
* geocoder\_response – единственное в базе данных поле типа json. Хранит ответ сервиса Google Geo с координатами места.

Для таблицы адресов созданы два индекса. Первый, индекс по полю addressable\_type позволяет быстро выделять адреса, принадлежащие определенным категориям объектов, например, адреса компаний. Второй индекс по полям addressable\_id и addressable\_type считается необходимым для корректной работы полиморфной связи.

**3.2.15** Таблица images

Вторая таблица с полиморфной связью. Служит для хранения ссылок на изображения-аватары пользователя из LinkedIn и Github, а также логотипов компаний.

Поля:

* id – первичный ключ;
* imageable\_id – внешний ключ полиморфной связи;
* imageable\_type – тип полиморфной связи;
* link – ссылка на картинку.

Данная таблица с полиморфной связью содержит лишь один индекс, по полям imageable\_id и imageable\_type, который считается необходимым для корректной работы полиморфной связи. Во введении индекса по типу полиморфной связи не возникло необходимости, в связи с отсутствием возможности получения из изображений информации, пригодной для анализа.

## **3.3** Описание структуры и взаимодействия между классами

Изучение модели данных, используемых в программном продукте, является одним из основных этапов понимания процесса обработки программой данных что, как известно, является основной конечной целью любой разрабатываемой программной системы. Не менее значимым этапом с этой точки зрения является описание структуры классов и взаимодействия между ними. Для описания структуры классов перечислим основные классы программного модуля с указанием способов их взаимодействия между собой, методов и полей, используя диаграмму классов (см. чертеж ГУИР.400201.003 РР.1) в соответствии с модулями, выделенными на этапе системного проектирования (см. чертеж ГУИР.400201.003 С1).

**3.3.1** Класс Geo

Класс Geo является базовым классом для AddressableGeo и FacebookGeo и составляет вместе с ними основу модуля геоданных. Данный класс предназначен для преобразования адреса, представленного в виде строки, содержащей страну и, возможно, город нахождения пользователя, в объект базы данных, содержащий в себе точные географические координаты соответствующего места. Для выполнения данной задачи используется сервис геокодинга, предоставляемый Google. Данный класс содержит только приватные методы, что объясняется его использованием в качестве базового.

Методы класса:

* geocode(address) – преобразование строки с адресом в json объект, содержащий точные названия страны и города, а также их географические координаты. В данном методе используется сторонняя библиотека Geocoder, позволяющая преобразовывать адреса с использованием публичных сервисов группы компаний Alphabet;
* save(resp, id, type) – сохранение полученного на предыдущем шаге json в таблицу addresses базы данных. Параметры id и type представляют собой пару значений, необходимых для создания полиморфной связи между адресом с одной стороны и компанией либо пользователем с другой. Метод содержит отладочный вывод, предназначенный для визуальной индикации работы в интерактивном режиме;
* update(resp, address) – в отличие от метода save, данный метод предназначен не для сохранения нового адреса в базу данных, а для изменения уже существующего. Предназначена эта возможность для случаев, когда некоторая часть адреса, например, город, уже была сохранена, однако необходимо получить координаты для дальнейшего импорта. Следует заметить, что адрес, прошедший процедуру геокодинга, помечается флагом geocoded.

**3.3.2** Класс FacebookGeo

Данный класс наследуется от базового класса Geo и реализует процесс обработки адресных данных для пользователей, полученных путем импорта из социальной сети Facebook. Специфика преобразования адресов для Facebook состоит в том, что полученный с использованием синтаксических анализаторов адрес необходимо преобразовать в вид, пригодный для дальнейшего использования при анализе и визуализации. Все поля класса являются приватными, все методы – публично доступными.

Поля класса:

* debug – флаг отладки, используется для ограничения количества обрабатываемых при вызове метода адресов с целью предотвратить преждевременное истощение лимита на использование API Google;
* web\_scraper – объект, реализующий синтаксический анализатор для получения адресной информации из социальной сети Facebook в обход стандартного API.

Методы класса:

* initialize(debug) – конструктор, принимает в качестве параметра флаг отладки;
* handle() – получает пользователей Facebook, еще не имеющих адресов текущего местонахождения, получает строки, содержащие приблизительный адрес, с использованием синтаксического анализатора, получает из них с использованием геокодинга корректные адреса и сохраняет получившиеся данные в базу данных.

**3.3.3** Класс AddressableGeo

Данный класс также наследуется от базового класса Geo, но, в отличие от класса FacebookGeo, предоставляет возможности геокодинга для общего случая, при котором нет необходимости в получении адресов с помощью синтаксических анализаторов и создании новых записей в базе данных, а достаточно лишь дополнить существующую запись географическими координатами и уточненными названиями населенных пунктов.

Поля класса:

* debug – флаг отладки, используется аналогично таковому для класса FacebookGeo с целью сохранения лимитов на запросы через API.

Методы класса:

* initialize(debug) – конструктор, принимает в качестве параметра флаг отладки;
* handle() – производит поиск в базе данных адресов, не прошедших процесс геокодинга и обновляет их с уточнением названий и географических координат. Как и в случае с FacebookGeo, флаг отладки влияет на количество получаемых из базы адресов.

**3.3.4** Класс WebScraper

Является базовым классом модуля синтаксических анализаторов. Предназначен для проведения первичной инициализации анализатора веб-страниц Mechanize, который предоставляется сторонней библиотекой и позволяет получить объектное отображение HTML кода страницы.

Поля класса:

* agent – объект, содержащий анализатор HTML страниц Mechanize.

Методы класса:

* initialize() – конструктор, производит инициализацию поля agent и первичную настройку параметров библиотеки Mechanize.

**3.3.5** Класс LinkedinWebScraper

Наследуется от класса WebScraper и реализует специфические алгоритмы для получения данных о публичном профиле профессиональной социальной сети LinkedIn с использованием регулярных выражений для выделения информации. Предназначен для поиска пользователя в сети LinkedIn по полному имени и получению идентификатора публичного профиля, используемого в дальнейшем для получения общедоступных данных о пользователе. Поля класса являются приватными.

Поля класса:

* results – массив строк, содержащий возможные ссылки на профили пользователя, полученные с использованием встроенного поиска веб-версии LinkedIn;
* profile\_id – строка, соответствующая найденному идентификатору публичного профиля пользователя на LinkedIn;
* logged\_in – признак успешной авторизации синтаксического анализатора в веб-версии LinkedIn.

Методы класса:

* scrape(name) – основной метод класса, принимает на вход полное транслитерированное имя пользователя, полученного из других источников, для которого необходимо определить соответствующий профиль LinkedIn. Получает profile\_id и вызывает метод get\_public\_profileт. В данном методе активно используется работа с регулярными выражениями, позволяющими выделить profile\_id из лишним данных, полученных анализатором HTML страниц Mechanize;
* get\_public\_profile() – на основании ранее полученного profile\_id производит попытку получения ссылки на общедоступный профиль LinkedIn;
* authorization() – единственный приватный метод класса, реализует функционал модуля авторизации, специфический для веб-версии социальной сети LinkedIn.

**3.3.6** Класс FacebookWebScraper

Наследуется от класса WebScraper и реализует специфические для социальной сети Facebook алгоритмы получения информации из HTML-страниц. Структура и поля, в целом, аналогичны таковым для класса LinkedinWebScraper, за исключением двух методов, служащих для получения различного рода адресов местоположения пользователя.

Поля класса:

* body – объект, представляющий собой образ полученной библиотекой Mechanize веб-страницы;
* logged\_in – признак успешной авторизации синтаксического анализатора в веб-версии Facebook.

Методы класса:

* scrape(facebook\_id) – основной метод класса, принимает на вход идентификатор пользователя, полученный с помощью открытого API Facebook. Выполняет авторизацию, если это необходимо;
* current\_location() – при помощи регулярных выражений получает из body местность текущего нахождения пользователя. Данный адрес не во всех случаях содержит название страны, так что для использования в аналитике необходимо использование методов геокодинга из модуля геоданных;
* from\_location() – при помощи регулярных выражений получает из body местность происхождения для пользователя. Проблемы и пути решения аналогичны таковым для current\_location;
* authorization() – единственный приватный метод класса, реализует функционал модуля авторизации, специфический для веб-версии социальной сети Facebook.

**3.3.7** Класс FacebookScraper

Данный класс также представляет собой одну из составных частей модуля синтаксических анализаторов. В отличие от FacebookWebScraper, который анализирует информацию из HMTL-страниц с помощью Mechanize, нарушая тем самым политику использования данных Facebook, данный класс получает информацию исключительно с использованием публично доступного API. Для работы с API Facebook используется сторонняя библиотека Koala.

Поля класса:

* graph – объект, реализующий методы библиотеки Koala;
* gid – идентификатор группы на Facebook, из которой планируется получить список пользователей;
* members – json, содержащий список пользователей заданной группы Facebook.

Методы класса:

* initialize(token, gid) – конструктор, принимает токен, необходимый для работы с API Facebook по протоколу OmniAuth и идентификатор группы для импорта списка пользователей. Для группы по умолчанию задано значение, соответствующее группу специальности ВМСиС на Facebook. Также конструктор инициализирует библиотеку Koala;
* get\_group\_members() – получает с помощью Koala список пользователей группы и сохраняет его в приватное поле members;
* save\_group\_members() – сохраняет ранее полученные с помощью метода get\_group\_members данные, создавая записи в таблице facebook\_data для каждого пользователя.

**3.3.8** Класс LinkedinScraper

Данный класс также представляет собой одну из составных частей модуля синтаксических анализаторов. Как и LinkedinWebScraper, который анализирует информацию из HMTL-страниц с помощью Mechanize, нарушая тем самым политику использования данных Linkedin, данный класс получает информацию с помощью анализа HMTL-страниц. Отличие состоит в том, что для этого используется сторонняя библиотека LinkedinProfile, позволяющая получить данные из публично доступного профиля Linkedin.

Поля класса:

* profile – объект, полученный при применении библиотеки LinkedinProfile. Представляет собой тело HMML-страницы с некоторыми новыми доступными методами;
* public\_profile – полученный с помощью LinkedinWebScraper адрес публично доступной страницы пользователя в сети Linkedin;
* web\_scraper – объект, содержащий экземпляр класса LinkedinWebScraper, используемый для получения адреса публичного профиля пользователя.

Методы класса:

* initialize() – конструктор, производит инициализацию экземпляра класса LinkedinWebScraper и обнуляет остальные поля;
* get\_public\_member(name) – принимает на вход полное транслитерированное имя пользователя, с помощью LinkedinWebScraper получает адрес публичного профиля пользователя, а затем с помощью библиотеки LinkedinProfile собирает публично доступные данные о пользователе;
* save\_сompanies(companies, current) – сохраняет в таблицу companies базы данных сведения об организациях, в которых пользователь работал или работает в данный момент;
* save\_сompanies(companies, current) – сохраняет в таблицу linkedin\_data базы данных сведения об импортированном с LinkedIn пользователе.

**3.3.9** Класс GithubScraper

Реализует задачу, аналогичную задачам FacebookScraper и LinkedinScraper, но для сервиса совместной работы Github. Отличие от предыдущих классов состоит в том, что политика использования данных не нарушается – Github предоставляет отличное API, через которое можно сравнительно небольшими усилиями получить любую представляющую интерес информацию. Что немаловажно, данное API является бесплатным, что компенсируется присутствием лимитов на количество запросов за сутки. Работа с API Github реализуется посредством библиотеки Octokit.

Поля класса:

* users – cписок логинов пользователей, полученных с помощью библиотеки Octokit, предположительно являющихся идентификаторами аккаунтов искомого пользователя на Github.

Методы класса:

* initialize() – конструктор, производит обнуление необходимых для работы класса полей;
* search\_user(name) – получает с помощью библиотеки Octokit список логинов пользователей Github, найденных в соответствии с переданным в метод полным транслитерированным именем;
* save\_user() – сохраняет в таблицу github\_data базы данных сведения об импортированном из Github пользователе;
* save\_repositories(login) – сохраняет в таблицу repositories базы данных данные о репозиториях пользователя с определенным логином на Github;
* save\_languages(repository) – получает через API и сохраняет в таблицу languages базы данных данные о языках программирования, используемых в конкретном репозитории.

**3.3.10** Класс Import

Данный класс служит для решения задачи импорта выпускников из имеющихся файлов формата csv. Класс является основной и единственной частью модуля импорта. Обработка файлов формата csv выполнена средствами языка Ruby. В дальнейшем пользователи, полученные данным способом проверяются на наличие профилей в каждой социальной сети с сохранением полученной информации.

Методы класса:

* import\_graduates(file, year) – метод класса, предназначенный для сохранения данных о выпускниках определенного года выпуска, представленных в файле формата csv в таблицу imported\_data базы данных.

**3.3.11** Класс Export

Данный класс реализует функции модуля экспорта, предоставляя данные, подвергнутые анализу, в необходимом для использования в модуле отчетов и интерактивном режиме виде.

Методы класса:

* addresses\_for\_map(countries) – метод класса, экспортирующий стасистику по странам в формат csv для дальнейшего использования при отображении мировой карты;
* encode(image) – приватный метод, переводит изображение в закодированный формат Base64, что позволяет встроить строку, содержащую закодированное изображение в тег img HTML страницы при формировании отчета модулем отчетов;
* sources() – предоставляет статистику по источникам данных в виде строки Base64;
* graduation\_date() – предоставляет статистику по датам выпуска студентов в виде строки Base64;
* countries() – предоставляет статистику по странам пользователей в виде строки Base64;
* skills() – предоставляет статистику по навыкам, собранным в системе Linkedin, в виде строки Base64;
* languages\_in\_repos() – предоставляет статистику по языкам программирования, используемых в репозиториях сервиса Github в виде строки Base64;
* companies\_countries() – предоставляет статистику по странам происхождения огранизаций в виде строки Base64;
* company\_sizes() – предоставляет статистику по размерам организаций в виде строки Base64.

**3.3.12** Класс Report

Данный класс образует собой модуль формирования отчетов. Закодированные в Base64 изображения, полученные с помощью модуля экспорта, передаются в особым образом сформированное представление и выводятся в файл формата pdf средствами библиотеки WickedPdf. Данная библиотека позволяет преобразовывать файлы формата HTML в формат pdf и в данном виде в большинстве промышленно разработанных приложений, имеющих в своем составе функционал генерации отчетов.

Поля класса:

* view – объект типа ActionView, содержащий шаблон, подготовленный для преобразования в формат pdf;
* save\_path – путь к сгенерированному файлу отчета.

Методы класса:

* save() – публичный метод, сохраняет файл с отчетов по заданному пути;
* rendered\_pdf() – формирует документ формата pdf из представления, сформированного методом rendered\_view;
* rendered\_view() – формирует представление ActionView и передает в него необходимые внутренние переменные.

**3.3.13** Класс Import

Данный класс формирует собой модуль предобработки данных и реализует вспомогательные функции, связанные транслитерацией имен и обработкой дат.

Методы класса:

* encode\_name(name) – метод, преобразовывающий переданное имя по правилам транслитерации в вариант, записанный латинским алфавитом. Используется стороння библиотека Russian;
* graduation\_date(year) – метод, принимающий год в численном или строковом виде и возвращающий объект даты, соответствующий приблизительной дате выпуска;
* foundation\_date(year) – метод, принимающий год основания компании и возвращающий объект даты. Так как получить точную дату создания компании из имеющихся источников невозможно, формируемая дата соответствует началу года основания.

**3.3.14** Класс Analytics

Данный класс формирует структуры данных, необходимые для дальнейшего построения графиков и создания отчетов и представляет модуль анализа данных. Аналитика используется относительно простая, однако следует заметить, что в сочетании с полученными данными это дает интересные с точки зрения общей информации результаты. Методы, включенные в данный класс, являются методами класса и возвращают, в основной массе, объекты класса Daru::DataFrame. Daru – библиотека анализа данных на языке Ruby, используемая в настоящем проекте.

Методы класса:

* count\_in\_array(array) – метод, подсчитывающий число повторяющихся элементов в массиве и возвращающий объект типа hash с парами значение – число повторений;
* sources() – предоставляет статистику по источникам данных в виде объекта Daru::DataFrame;
* graduation\_date() – предоставляет статистику по датам выпуска студентов в виде объекта Daru::DataFrame;
* countries() – предоставляет статистику по странам пользователей в виде объекта Daru::DataFrame;
* skills() – предоставляет статистику по навыкам, собранным в системе Linkedin, в виде объекта Daru::DataFrame;
* languages\_in\_repos() – предоставляет статистику по языкам программирования, используемых в репозиториях сервиса Github в виде объекта Daru::DataFrame;
* companies\_countries() – предоставляет статистику по странам происхождения огранизаций в виде объекта Daru::DataFrame;
* company\_sizes() – предоставляет статистику по размерам организаций в виде объекта Daru::DataFrame.

**3.3.15** Класс Visualization

Данный класс формирует графики типа гистограмма из предоставленных объектов класса Daru::DataFrame. Для построения графиков используется библиотека Gnuplotrb, основанная на системной библиотеке Gnuplot. Выбор пал именно на этот вариант благодаря беспроблемной работе данной связки в системе интерактивного выполнения команд Jupyter.

Методы класса:

* sources() – предоставляет статистику по источникам данных в виде графика, образованного объектом Plot;
* graduation\_date() – предоставляет статистику по датам выпуска студентов в виде графика, образованного объектом Plot;
* countries() – предоставляет статистику по странам пользователей в виде графика, образованного объектом Plot;
* skills() – предоставляет статистику по навыкам, собранным в системе Linkedin, в виде графика, образованного объектом Plot;
* languages\_in\_repos() – предоставляет статистику по языкам программирования, используемых в репозиториях сервиса Github в виде графика, образованного объектом Plot;
* companies\_countries() – предоставляет статистику по странам происхождения огранизаций в виде графика, образованного объектом Plot;
* company\_sizes() – предоставляет статистику по размерам организаций в виде графика, образованного объектом Plot.

**3.3.16** Класс SocialAnalytics

Данный класс образует интерфейс для разрабатываемого модуля анализа данных профессиональных социальных сетей и включает в себя методы, содержащие необходимую последовательность действий для обработки данных из любого из определенных на этапе анализа предметной области источников. Все необходимые данному модулю пароли рекомендуется хранить в файле, исключенном из системы контроля версий.

Поля класса:

* facebook\_token – токен OmniAuth, необходимый для получения данных через API Facebook c помощью библиотеки Koala;
* facebook\_email – логин для веб-интерфейса Facebook, необходимый для работы синтаксического анализатора FacebookWebScraper;
* facebook\_password – пароль для веб-интерфейса Facebook, необходимый для работы синтаксического анализатора FacebookWebScraper;
* linkedin\_email – логин для веб-интерфейса LinkedIn, необходимый для работы синтаксического анализатора LinkedinWebScraper;
* linkedin\_password – пароль для веб-интерфейса LinkedIn, необходимый для работы синтаксического анализатора LinkedinWebScraper;
* github\_login – логин для Github, требуемый для получения данных через API;
* github\_password – пароль для Github, требуемый для получения данных через API.

Методы класса:

* handle\_facebook() – публичный метод, реализующий необходимую последовательность действий для получения пользователей из группы Facebook и обработки всех полученных адресов;
* handle\_linkedin() – публичный метод, реализующий необходимую последовательность действий для получения пользователей из профессиональной социальной сети LinkedIn на основе предварительно импортированных из файлов csv выпускников и пользователей, полученных из группы на Facebook и сбора дополнительной информации о навыках и организациях;
* handle\_github() – публичный метод, реализующий необходимую последовательность действий для получения пользователей из сервиса совместной работы Github на основе предварительно импортированных из других источников данных и сбора дополнительной информации об организациях, репозиториях и используемых языках программирования.

**3.3.17** Класс Position

Настоящий класс является объектным отображением таблицы positions. Используется как представление связи между классами LinkedinData и Company, а также для манипуляции импортированными из сети LinkedIn данными о рабочих позициях пользователя.

Поля класса:

* title – наименование рабочей позиции;
* duration – длительность работы на текущем месте в виде строки с описанием на естественном языке;
* start\_date – дата начала работы в компании;
* end\_date – дата конца работы в компании;
* current – признак того, что пользователь работает на данной позиции в настоящий момент.

Методы класса:

* linkedin\_data() – публичный метод, получение связанной с настоящим объектом рабочей позиции записи импортированных с LinkedIn данных;
* company() – публичный метод, получение компании, являющейся работодателем по отношению к настоящей позиции.

**3.3.18** Класс ImportedData

Настоящий класс является объектным отображением таблицы imported\_data. Используется для манипуляции данными о пользователях, загруженных из файла csv с помощью веб-интерфейса.

Поля класса:

* first\_name – имя студента;
* last\_name – фамилия студента;
* middle\_name – отчество студента;
* encoded\_name – транслитерированное полное имя;
* graduated\_at – дата выпуска из учебного заведения.

Методы класса:

* user() – публичный метод, получение модели пользователя, соответствующего импортированным из файла совокупности имени, фамилии и отчества;
* full\_name() – публичный метод, комбинирующий имя и фамилию пользователя в полное имя с помощью конкатенации строк;
* update\_user() – приватный метод, обновляющий полное имя в модели пользователя при изменениях в ImportedData;
* create\_user() – защищенный метод, создает нового пользователя при добавлении новой записи в таблицу imported\_data;
* encode\_name(full\_name) – защищенный метод, позволяет получить из полного имени пользователя на русском языке транслитерированный вариант для проведения операции поиска через API источников данных.

**3.3.19** Класс Repository

Настоящий класс является объектным отображением таблицы repositories. Используется для манипуляции данными о репозиториях проектов, существующих в системе совместной работы Github.

Поля класса:

* full\_name – ссылка на репозиторий, содержит в себе имя пользователя Github и название проекта;
* repo\_id – идентификатор репозитория на Github;
* forks – число форков репозитория;
* issues – число заявок о проблемах и предложениях;
* watchers – число пользователей, подписанных на обновления данного репозитория;
* subscribers – число подписчиков репозитория на Github;
* stargazers – число пользователей, отметивших репозиторий;
* fork – признак форка другого репозитория.

Методы класса:

* github\_data() – публичный метод, получение модели GithubData, отображающей пользователя, которому принадлежит настоящий репозиторий;
* languages() – публичный метод, получение списка языков программирования, используемых в программном проекте;
* primary\_language() – публичный метод, получение основного языка проекта.

**3.3.20** Класс Language

Настоящий класс является объектным отображением таблицы languages. Представляет собой название языка программирования, определяемого сервисом совместной работы Github. Используется для манипуляции данными о языках программирования, присутствующих в системе.

Поля класса:

* name – название языка программирования.

Методы класса:

* repositories() – публичный метод, получение списка репозиториев, содержащих в своей кодовой базе подпрограммы, написанные на данном языке программирования.

**3.3.21** Класс Image

Настоящий класс является объектным отображением таблицы images. Используется для манипуляции данными об изображениях, встречающихся в импортированных из используемых источников данных. Хранение файла картинки не практикуется из соображений проблематичности анализа бинарных данных.

Поля класса:

* link – ссылка на изображение.

Методы класса:

* imageable() – публичный метод, позволяет получить объект-владельца изображения. Имеет возможность возвращать объекты различных типов благодаря использованию полиморфной связи.

**3.3.22** Класс Company

Настоящий класс является объектным отображением таблицы companies. Используется для манипуляции данными о компаниях, полученных путем импорта из LinkedIn.

Поля класса:

* name – название компании;
* description – публично доступное описание компании с Linkedin, содержит значительный объем текста;
* linkedin\_company\_url – ссылка на страницу компании в системе LinkedIn;
* website – ссылка на сайт компании;
* company\_size – число сотрудников компании, интервальная величина;
* company\_type – форма собственности компании;
* industry – сфера бизнеса;
* founded – дата основания компании;
* country – страна, под юрисдикцию которой попадает организация.

Методы класса:

* create\_addresses() – метод класса, создающий записи в таблице адресов, используя поле country для компаний, еще не имеющих адреса;
* linkedin\_data() – публичный метод, получение списка аккаунтов LinkedIn, владельцы которых работали в прошлом либо работают в настоящий момент на данную компанию;
* positions() – публичный метод, получение списка рабочих позиций в компании, информация о которых имеется в системе LinkedIn;
* images() – публичный метод, получение списка изображений, представляющих собой аватары пользователя в социальной сети LinkedIn;
* address() – публичный метод, возвращающий адрес компании, созданный на основе поля country.

**3.3.23** Класс Skill

Настоящий класс является объектным отображением таблицы skills. Используется для манипуляции данными о навыках, указанных пользователями в профилях системы LinkedIn. Навыки представляют собой области экспертизы пользователя системы.

Поля класса:

* name – название навыка.

Методы класса:

* linkedin\_data() – публичный метод, возвращающий список аккаунтов LinkedIn, содержащих упоминание данного навыка.

**3.3.24** Класс FacebookData

Настоящий класс является объектным отображением таблицы facebook\_data. Используется для манипуляции данными о пользователях, полученных путем импорта данных с использованием API социальной сети Facebook и специальных синтаксических анализаторов.

Поля класса:

* original\_name – имя пользователя, указанное на странице в социальной сети;
* encoded\_name – транслитерированное полное имя;
* facebook\_id – идентификатор профиля в сети Facebook;
* address\_geocoded – признак выполнения обработки адреса, полученного со страницы пользователя.

Методы класса:

* user() – публичный метод, получение модели пользователя, соответствующего импортированным из файла совокупности имени, фамилии и отчества;
* full\_name() – публичный метод, комбинирующий имя и фамилию пользователя в полное имя с помощью конкатенации строк;
* update\_user() – приватный метод, обновляющий полное имя в модели пользователя при изменениях в FacebookData;
* create\_user() – защищенный метод, создает нового пользователя при добавлении новой записи в таблицу facebook\_data;
* encode\_name(full\_name) – защищенный метод, позволяет получить из полного имени пользователя на русском языке транслитерированный вариант для проведения операции поиска через API источников данных.

**3.3.24** Класс GithubData

Настоящий класс является объектным отображением таблицы github\_data. Используется для манипуляции данными о пользователях, полученных посредством использования API cервиса совместной работы Github.

Поля класса:

* login – идентификатор пользователя на Github;
* name – транслитерированное полное имя;
* email – адрес электронной почты, пригодный для рассылки спама;
* num\_repos – число репозиториев, принадлежащих пользователю (включая приватные);
* num\_gists – число файлов, сохраненных в специальном сервисе Github Gists для хранения отдельных файлов, не входящих в проект;
* num\_followers – число пользователей, подписанных на обновления данного профиля;
* num\_following – число подписок на обновления профилей других пользователей;
* blog – адрес блога пользователя.

Методы класса:

* user() – публичный метод, получение модели пользователя, соответствующего импортированным из файла совокупности имени, фамилии и отчества;
* encode() – публичный метод, возвращающий транслитерированное имя. По своей сути не выполняет никаких преобразований, вместо этого определяя псевдоним для метода name. Связано это с возвратом API Github заранее транслитерированного имени пользователя;
* full\_name() – публичный метод, комбинирующий имя и фамилию пользователя в полное имя с помощью конкатенации строк;
* update\_user() – приватный метод, обновляющий полное имя в модели пользователя при изменениях в FacebookData;
* create\_user() – защищенный метод, создает нового пользователя при добавлении новой записи в таблицу facebook\_data.

**3.3.25** Класс User

Настоящий класс является объектным отображением таблицы users. Используется для манипуляции данными о пользователях системы.

Поля класса:

* name – транслитерированное имя пользователя;
* facebook – признак наличия данных, импортированных из социальной сети Facebook;
* linkedin – признак наличия данных, полученных из профессиональной сети LinkedIn;
* linkedin\_scraped – признак завершенного импорта из LinkedIn;
* github – признак наличия данных, собранных через API платформы для совместной работы Github;
* github\_scraped – признак завершения импорта из Github.

Методы класса:

* facebook\_data() – публичный метод, возвращающий данные пользователя, полученные из социальной сети Facebook;
* linkedin\_data() – публичный метод, возвращающий данные пользователя, полученные из профессиональной социальной сети LinkedIn;
* github\_data() – публичный метод, возвращающий данные пользователя, полученные из сервиса совместной работы Github;
* imported\_data() – публичный метод, возвращающий данные пользователя, полученные путем импорта файла csv;
* addresses () – публичный метод, возвращающий список адресов, принадлежащих пользователю. Используется полиморфная связь;
* images() – публичный метод, возвращающий список изображений пользователя. Используется полиморфная связь;
* remove\_duplicates() – метод класса, предназначенный для удаления дубликатов данных, созданных в процессе импорта.

**3.3.26** Класс Address

Настоящий класс является объектным отображением таблицы addresses. Используется для манипуляции данными об адресах.

Поля класса:

* city – город;
* country – страна;
* lat – географическая широта. Используется формат с большой точностью;
* lng – географическая долгота. Так же, как и с случае широты, используется числовой формат с большой точностью;
* geocoded – признак завершения геокодинга, выполняемого с целью получения широты и долготы на основании известного текстового адреса;
* geocoder\_response – единственное в базе данных поле типа json. Хранит ответ сервиса Google Geo с координатами места.

Методы класса:

* addressable() – публичный метод, возвращает владельца адреса. Благодаря использованию полиморфной связи имеет возможность возвращать объекты различных типов;
* countries\_statistics() – метод класса, оперирующий с объектами ActiveRelation. Для некоторого диапазона адресов создается статистика количества записей для каждой страны, присутствующей в системе;
* to\_csv() – метод класса, преобразующий статистику по странам в формат csv с помощью вызываемого метода из модуля экспорта. Для работы с диапазоном адреса также используется объект ActiveRelation.

**3.3.27** Класс BelongsToUser

Вспомогательный класс, расширяющий системный модуль фреймворка ActiveSupport::Concert. Используется для автоматизации создания модели User при импорте данных из имеющихся источников, что позволяет впоследствии провести импорт из еще не исследованных источников.

Методы класса:

* create\_user() – защищенный метод, создающий для объекта модель User при ее отсутствии и заполняющий поле encoded\_name соответствующим значением.

**3.3.28** Класс Encodable

Вспомогательный класс, расширяющий системный модуль фреймворка ActiveSupport::Concert. Используется для автоматизации транслитерации имени пользователя в объектах, отражающих импортированные из внешних источников данные.

Методы класса:

* encode\_name() – защищенный метод, проводящий транслитерацию имени пользователя с использованием одной из функций модуля предобработки данных.

# **4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программного модуля анализа данных профессиональных социальных сетей использовалось значительное количество сторонних библиотек, что объясняется необходимостью выполнения определенного перечня стандартных задач по работе с API сторонних сервисов, анализу данных и генерации отчетов. Следует, однако, заметить, что социальные сети и иные сервисы схожего назначения в открытый доступ посредством API выносят лишь небольшую часть данных, доступных при помощи веб-интерфейса. Данные особенности работы с данными оформлены в виде специальных соглашений об использовании данных и недвусмысленно упоминают о запрете на автоматизацию обращений к веб-версии сайта с целью получения недостающей части данных. Следует заметить, что данные, доступные публично через удобные для разработчика API, ожидаемо являются в некоторой степени полезными для задач анализа, однако значительно более интересные свойства и характеристики аккаунтов закрыты для использования. При создании настоящего программного модуля ставилась задача получения наиболее интересных видов данных о пользователях сервисов любыми средствами, в том числе и нарушающими пользовательские соглашения и соглашения об использовании данных. Рассмотрим некоторые алгоритмы, позволяющие организовать получение данных, их дальнейший анализ и отображение. Полный листинг наиболее специфических участков программного кода приводится в приложении к настоящей пояснительной записке (см. приложение А).

## **4.1** Алгоритм получения данных Github

Решение начать именно с приведения примера работы с сервисом совместной работы Github может быть обосновано тем, что, перед анализом алгоритмов, направленных на получение данных сомнительной законности способами, будет полезно увидеть пример работы с сервисом при помощи API, в котором не требуется нарушать абсолютно никаких условий пользовательского соглашения. Сервис Github размещает абсолютно всю необходимую в настоящем проекте информацию в публично доступном виде. В этом состоит основное отличие Github от социальных сетей вроде Facebook и LinkedIn – данный сервис создан разработчиками для разработчиков. API Github не является в полной мере бесплатным, однако предоставляет достаточно возможностей для получения всех доступных данных без внесения какой-либо платы [17]. Ограничивается лишь количество запросов за определенный период времени, что является очень удобным вариантом для исследователей, которым не принципиально немедленное получение большого объема данных и приложения сбора информации которых поддерживают техническую возможность запуска задач по расписанию. Также среди всех варинтов внешних API, используемых в настоящем дипломном проекте, API Github является наиболее логично структурированным и следует большей части неявно сформулированных правил разработки программных интерфейсов, реализуя именно то поведение, которого ожидает от него разработчик, имеющий необходимость анализировать информацию сервиса.

Не менее значимым моментом, чем общая логичность предоставляемого программного интерфейса, является тот факт, что для языка Ruby и фреймворка Ruby on Rails доступна библиотека Octokit, упрощающая работу с API Github [18]. Стандартным видом представления API является описания протокола взаимодействия через HTTP-запросы, что является не слишком удобным способом для применения в объектно-ориентированном программном продукте. Octokit предоставляет возможность работы с программным интерфейсом Github с использованием объектно-ориентированного подхода, в котором сущности предоставляются в виде объектов либо структур с самоочевидными названиями полей и методов.

Для начала работы с библиотекой Octokit следует указать логин и пароль от аккаунта на Github, что позволит поднять лимит на запросы до 5000 запросов в час. Это довольно значительная величина, такого количества запросов вполне достаточно для решения всех задач, связанных непосредственно с сервисом совместной работы Github. Инициализация библиотеки производится следующим образом:

Octokit.configure do |c|

c.login = Rails.application.secrects.github\_login

c.password = Rails.application.secrets.github\_password

end

Как заметно из данного примера, инициализация производится путем передачи в метод класса блока с присвоением переменным значений логина и пароля. Важным моментом, на который следует обратить особое внимание, является то, что вся приватная информация, такая как логины и пароли от использованных сервисов, выносится в отдельный файл, не отслеживаемый в системе контроля версий, и значения полей в незашифрованном виде.

На момент использования Octokit с целью получения данных о пользователях в сервисе совместной работы Github с системе, как правило, уже присутствует информация о пользователях, полученная иными методами, что проиллюстрировано в типичном пользовательском сценарии на диаграмме последовательностей (см. чертеж ГУИР.400201.003 РР.3). Таким образом, поиск пользователя Github проводится на основании известного полного имени импортированного пользователя. Метод search\_user, принимающий на вход полное транслитерированное имя пользователя с запускающий процесс анализа данных Github, демонстрирует данный подход:

def search\_user(name)

logins = Octokit.search\_users(name).items.map(&:login)

@users = logins.map{|login| Octokit.user(login)}

save\_user

end

Следует заметить, что в приведенном выше программном коде используются методы, характерные для функционального программирования. Что, бесспорно, является приятным фактом. Следующим этапом является сохранение информации о найденных пользователях, на этапе которого и проявляется вся лаконичность работы с объектным отображением программного интерфейса Github.

def save\_user

return if @users.empty?

puts "Found Github users".blue

@users.each do |user|

github\_data = GithubData.create(

login: user.login,

name: user.name,

email: user.email,

num\_repos: user.public\_repos,

num\_gists: user.public\_gists,

num\_followers: user.followers,

num\_following: user.following,

blog: user.blog

)

if user.avatar\_url.present?

puts "Saving avatar image".yellow

Image.create(

imageable: github\_data.user,

link: user.avatar\_url

)

end

save\_repositories(user.login)

end

end

Таким образом, все действия разработчика при работе с предоставляемым программным интерфейсом сводятся к вызову методов, предоставляемых библиотекой Octokit для объектов данных.

Программный код для сохранения информации о репозиториях и языках программирования также представляет интерес, ознакомиться с ним можно в приложении А к настоящей пояснительной записке. Описанный в приложении код повторяет основные принципы работы с программным интерфейсом Github на примере другого типа данных.

## **4.2** Алгоритм получения данных LinkedIn

Рассмотренный в подразделе 4.1 алгоритм работы с сервисом Github являлся отличным примером грамотного использования предоставляемых программным интерфейсом API возможностей при высокой степени открытости данных о пользователях в системе. Социальные сети LinkedIn и Facebook обладают значительно меньшей открытостью и, для получения наиболее интересных с точки зрения анализа данных появляется использовать нестандартные способы, идущие к тому же в разрез с политикой пользования системой и, при определенном стечении обстоятельств, приводящие к блокировке аккаунта в используемой социальной сети. Рассмотрим данную ситуацию на примере профессиональной социальной сети LinkedIn.

def get\_public\_member(name)

@public\_profile = @web\_scraper.scrape(name)

return if @public\_profile.nil?

@profile = begin

Linkedin::Profile.new(

"linkedin.com/in/#{@public\_profile}",

{company\_details: true}

)

rescue

nil

end

save\_public\_member

end

Как несложно заметить из приведенного программного кода, в случае с LinkedIn также использутся сторонняя библиотека [19]. В данном случае упомянутая библиотека служит для получения публичного профиля пользователя с целью дальнейшего анализа доступных данных. Однако для получения публичного профиля в сети LinkedIn необходимо знать ссылку на публичный профиль пользователя, найденного по полному имени. Для выполнения указанной задачи используется библиотека Mechanize, предназначенная для эмуляции взаимодействия пользователя с веб-интерфейсом LinkedIn и специальный класс LinkedinWebScraper. Основные методы данного класса:

def scrape(name)

authorization if !@logged\_in

url = "?type=people&keywords=#{

name.split(' ').join('+')

}"

@results = @agent.

get(url).

body.

scan(/\{"person"\:\{.\*?\}\}/)

return if !@results[0].present?

@profile\_id = begin

JSON.parse(@results[0]).

to\_s.

scan(/profile\/view\?id=(.\*?)&/).

compact.first.first

rescue

nil

end

get\_public\_profile

end

def get\_public\_profile

return if @profile\_id.nil?

url = "?id=#{@profile\_id}"

profile\_raw = @agent.get(url).body

@public\_profile = begin

profile\_raw.scan(/\.com\/in\/(.\*?)\//).first.first

rescue

nil

end

end

Полный код описываемого класса можно изучить в приложении А к настоящему дипломному проекту. Как несложно заметить, эмуляция взаимодействия пользователя с веб-интерфейсом является значительно более сложным для разработчика механизмом, чем работа с API, предоставляемым сервисом. Код алгоритма насыщен регулярными выражениями, которые, как известно, создают примерно столько же проблем, сколько решают. Также необходимым элементом алгоритма являются блоки обработки ошибок, так как взаимодействие с веб-интерфейсом отличается крайне низкой степенью надежности и предсказуемости и уверенности в доступности каких либо данных в назначенном месте в назначенное время нет и быть не может.

Служит выше описанный код для поиска пользователя по его полному имени и получения ссылки на публичный профиль. Наличие ссылки на публичный профиль позволяет воспользоваться библиотекой LinkedinScraper, одним из разработчиков которой является автор настоящего дипломного проекта. Подход, используемый внутри библиотеки, слабо отличается от описанного алгоритма, однако внешне работа с данными профиля является в значительной степени комфортной, чем работа с синтаксическим анализатором страниц Mechanize, используемым при получении ссылки на публично доступный профиль. Таким образом, для получения данных из социальной сети LinkedIn API не используется в принципе, вся работа ведется вручную полузаконными способами.

**4.3** Алгоритм получения данных Facebook

Алгоритм для социальной сети Facebook представляется в определенной степени гибридом между первым и вторым подходами. Для Facebook доступна достаточно удобная в разработке библиотека Koala, написанная на языке Ruby и использующая предоставляемый социальной сетью интерфейс взаимодействия [20]. Однако исследователь данных, работающий с данной социальной сетью вынужден довольствоваться некоторой незначительной частью данных, не представляющих особого интереса либо крайне сложных в обработке из-за использования естественного языка. Для настоящего дипломного проекта было принято решение о необходимости получения данных о местоположении пользователей социальной сети любой ценой, поскольку именно пользователи социальной сети Facebook максимально точно идентифицируются со специальностью ВМСиС и содержат данные о местонахождении в обязательном порядке.

def get\_group\_members

@graph = Koala::Facebook::API.new(token)

puts "Getting group #{@gid} members...".blue

begin

@members = @graph.get\_connections(@gid, "members")

rescue

puts "Refresh token!".red

end

end

def save\_group\_members

self.get\_group\_members if !@members.present?

return if @members.empty?

@members.each do |member|

facebook\_data = FacebookData.create(

original\_name: member["name"],

facebook\_id: member["id"]

)

puts "Save fb user ".blue

end

end

Как заметно из данного алгоритма, список пользователей определенной группы в социальной сети Facebook получается с помощью библиотеки Koala. Но с помощью этого метода доступно лишь сохранение имени пользователя, которое понадобится для генерации транслитерированного варианта, и идентификатора пользователя в социальной сети Facebook. Что, несомненно, является недостаточным количеством данных даже для самого элементарного анализа.

def scrape(facebook\_id)

authorization if !@logged\_in

@body = begin

@agent.get("facebook\_id}").body

rescue

puts "Problem with getting fb profile".red

nil

end

end

def current\_location

begin

@body.scan(/Lives in .\*? href=\"(.\*?)\?ref=br\_rs/).

flatten.

first.

scan(/https:\/\/www.facebook.com\/pages\/(.\*?)\//).

flatten.

first

rescue

nil

end

end

Как несложно заметить на примере методов получения местоположения пользователя, данный алгоритм также требует значительного количества регулярных выражений, успешность выполнения которых не гарантируется, что, в свою очередь, приводит к необходимость использовать блоки обработки исключений.

Данные о местоположении пользователя, недоступные через любые программные интерфейсы, предоставляемые Facebook, тем не менее доступны через веб-интерфейс социальной сети. Пользователь Facebook имеет возможность, посетив страницу профиля другого пользователя, получить данные о городе местопребывания последнего.

Несмотря на доказанную общую работоспособность указанного алгоритма, нарушение политики использования данных Facebook вызывает активное противодействие со стороны социальной сети [21]. Как несложно заметить, получение страницы профиля пользователя происходит по ранее сохраненному идентификатору, возвращенному методом из библиотеки Koala. И этот способ действительно работает. Первые несколько сотен раз. Дело в том, что для доступа к профилям в веб-интерфейсе социальной сети используются иные идентификаторы, не совпадающие по назначению с указанными выше и полученными через Koala. Повторяющиеся с высокой частотой запросы специфического вида позволяют Facebook отслеживать нарушение политики использования данных, что приводит к отключению возможности получения профиля пользователя по идентификатору социальной сети. Любые другие возможности пользователя не блокируются и не подвергаются каким-либо изменениям. Во время тестовых запусков данного программного модуля были получены около 300 адресов с одного аккаунта социальной сети и около 100 с другого, заполненного меньшим количеством информации. Таким образом, несмотря на доказанную работоспособность данной комбинации API и синтаксического анализа, следует заявить о необходимости поиска иных решений в случае с реализацией системы, направленной на динамическое получение информации из данной конкретной социальной сети. Возможно, следует ограничиться использованием исключительно инструментов, выполняющих эмуляцию взаимодействия пользователя с веб-интерфейсом сервиса, например, библиотекой Watir, позволяющей управлять действиями на веб-страницах с помощью команд на языке Ruby [22].

**4.4** Алгоритм анализа и визуализации данных

Этапом, следующим за получением данных, является их анализ с целью выделения информации, несущей большую смысловую нагрузку по сравнению с сырыми данными профилей. Для данного этапа в настоящем дипломном проекте используется библиотека Daru, позволяющая являющаяся одним из немногих доступных для языка Ruby средств анализа данных [23]. Традиционно язык Python является более приспособленным для использования в данном направлении.

def self.skills

skills = LinkedinDataSkills.

includes(:skill).

map{|r| r.skill.name}

sorted = count\_in\_array(skills).

sort\_by{|k,v| -v}.

first(100)

Daru::DataFrame.new({

skills: sorted.map{|l| l[1]}

},

index: sorted.map{|l| l[0]}

)

end

def self.languages\_in\_repos

languages = LanguagesRepositories.

includes(:language).

map{|r| r.language.name}

sorted = count\_in\_array(languages).

sort\_by{|k,v| -v}.first(100)

Daru::DataFrame.new({

repositories: sorted.map{|l| l[1]}

},

index: sorted.map{|l| l[0]}

)

end

def self.companies\_countries

st = Address.

where(addressable\_type: ["Company"]).

countries\_statistics

sorted = st.sort\_by{|k,v| -v}

Daru::DataFrame.new({

companies: sorted.map{|l| l[1]}

},

index: sorted.map{|l| l[0]}

)

end

Из приведенного выше программного кода заметно, что данные, получаемые путем выполнения запросов к базе данных на основе использования ORM ActiveRecord и применения методов работы со списками, характерных для функциональных языков программирования, упаковываются в контейнеры Daru. На первый взгляд, контейнеры Daru не выполняют какой-либо особенной деятельности по отношению к содержащимся к ним данным. Однако основное предназначение библиотеки заключается в интеграции разрабатываемой аналитики в экосистему среды интерактивной манипуляции данными Jupyter, позволяющей использовать контейнеры Daru в качестве элементарных единиц инструмента интерактивной аналитики (см. рисунок 4.1).

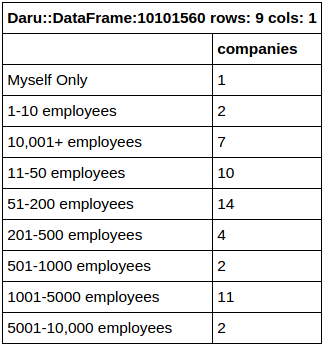


Рисунок 4.1 – Представление контейнера DataFrame в среде Jupyter

Для контейнеров DataFrame библиотека Daru позволяет использовать основные методы интерактивной работы с наборами данных: выделение векторов данных, манипуляции значениями таблицы, вычисление средних значений и проведение различного вида интерполяций результатов.

Таблица, как представление данных в форматированном виде, является в достаточной степени наглядным решением для среды интерактивной работы. Однако для формирования отчетов и визуального определения зависимостей между различными наборами данных гораздо лучше подходят графики, представленные в виде изображений и формируемые динамически на основании изменений контейнеров DataFrame.

def self.skills

Plot.new(

Analytics.skills,

style\_data: 'histograms',

style\_fill: 'pattern border',

title: 'Skills (linkedin)',

ylabel: 'Users',

xtics: 'nomirror rotate by -45',

term: ['pngcairo', size: [5000, 600]]

)

end

def self.languages\_in\_repos

Plot.new(

Analytics.languages\_in\_repos,

style\_data: 'histograms',

style\_fill: 'pattern border',

title: 'Repositories count by language',

ylabel: 'Repositories',

xtics: 'nomirror rotate by -45',

term: ['pngcairo', size: [5000, 600]]

)

end

def self.companies\_countries

Plot.new(

Analytics.companies\_countries,

style\_data: 'histograms',

style\_fill: 'pattern border',

title: 'Companies in country statistics',

ylabel: 'Companies',

xtics: 'nomirror rotate by -45',

)

end

Из приведенного фрагмента класса Visualization заметно, что контейнеры Daru::DataFrame используются в качестве источников данных для построения графиков. Средства для построения графиков, а именно класс Plot, предоставляются библиотекой Gnuplotrb, входящей в пакет библиотек для организации научных вычислений на языке Ruby. Библиотека Gnuplotrb, в свою очередь, действует с помощью системной библиотеки Gnuplot, входящей в базовую поставку операционных систем семейства Linux [24].

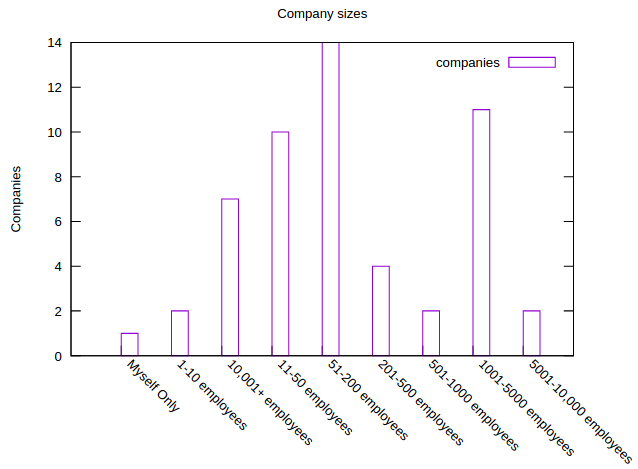


Рисунок 4.2 – Представление графика в среде Jupyter

Таким образом, комбинирование средств, предоставляемых библиотеками Daru и Gnuplotrb, а также средства интерактивного взаимодействия и вычислений Jupyter позволило сформировать удобную среду, пригодную для выполнения различного рода операций над данными, полученными из профессиональной социальной сети LinkedIn, социальной сети Facebook и сервиса совместной работы Github.

Следует заметить, что система Jupyter не предназначена для использования в совокупности с языком программирования Ruby [25]. Для запуска среды в контексте настоящего программного модуля и получения доступа к его методам, классам и импортированным данным, использовалась библиотека IRuby, предоставляющая ядро, поддерживающее язык Ruby для среды Jupyter, работающей по умолчанию исключительно с языком программирования Python. С программным кодом, необходимым для полнофункциональной работы указанной связки библиотек, можно ознакомиться в приложении Б к настоящему дипломному проекту.

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Одной из наиболее значимых задач в ходе жизненного цикла разработки ПО является тестирование. Тестирование программного обеспечения — это процесс исследования ПО с целью выявления ошибок и определения соответствия между реальным и ожидаемым поведением ПО, осуществляемый на основе набора тестов, выбранных определённым образом. В более широком смысле, тестирование ПО — это техника контроля качества программного продукта, включающая в себя проектирование тестов, выполнение тестирования и анализ полученных результатов.

Каждый этап разработки системы сопровождался написанием большого количества разнообразных тестов. Каждая новая функциональность программы тщательно проверялась различными методами. В ходе разработки были применены такие виды тестирования, как:

* функциональное тестирование;
* тестирование производительности;
* тестирование стабильности;
* тестирование совместимости;
* тестирование интерфейса пользователя.

На сегодняшний день промышленные способы проверки качества программного обеспечения используют различные автоматизированные системы тестирования. В Ruby on Rails такой системой по умолчанию является TestUnit. Данная автоматизированный модуль позволяет проводить тестирование моделей, контроллеров и представлений. В среду Rails встроена на удивление надежная и готовая к использованию система тестирования. Затрачивая минимум усилий, можно задавать воспроизводимые настройки баз данных, отправлять Web-приложениям тестовые HTTP-сообщения и выполнять три вида тестирования: модульное, функциональное и комплексное.

При написании тестов для различных уровней приложения значительно снижаются риски нарушения работоспособности приложения при внесении различных изменений в существующий функционал. Полное покрытие тестами занимает порядка 38% от общего времени работы программиста над проектом, но временные затраты полностью окупаются эффектом надежности и стабильности программного обеспечения. По мере продвижения проекта стоимость устранения дефектов ПО может экспоненциально возрастать. Инструменты статического и динамического анализа помогают предотвратить эти затраты благодаря обнаружению программных ошибок на ранних этапах жизненного цикла ПО.

При написании данной дипломной работы разработка велась по принципу TDD (Test Driven Development), реализующему процесс написания приложения через тестирование. Основополагающий принцип TDD – это написание тестов до написания кода приложения.

Разработка через [тестирование](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F)  — техника [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.  Разработка через тестирование требует от разработчика создания автоматизированных [модульных тестов](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, либо нет. Когда они выполняются, говорят, что тест пройден.

Тестирование в рамках TDD позволяет получить простой способ оценки полноты интерфейсов: необходимым и достаточным считается такой интерфейс, который позволяет выполнить все написанные тесты. Все, что находится за этими рамками, считается ненужным. Наконец, использование тестов в качестве инструмента дизайна заставляет программиста в первую очередь концентрироваться на интерфейсе, а уже во вторую - на имплементации, что также положительно влияет на результат.

Было достигнуто полное покрытие тестами моделей, контроллеров, сервисов и представлений всего приложения. Результат выполнения общего тестирования системы представлен на рисунке 5.1, результат выполнения тестирования контроллеров и моделей представлен на рисунке 5.1.

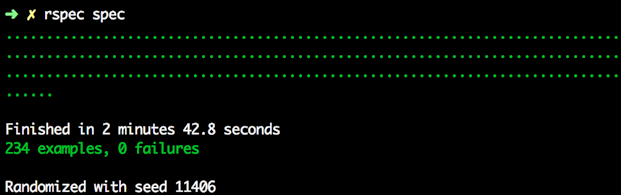


Рисунок 5.1 – Результаты выполнения общего тестирования системы

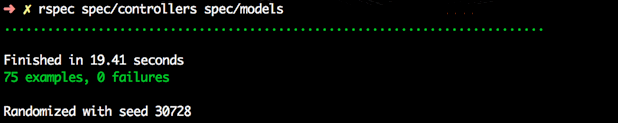


Рисунок 5.2 – Результаты выполнения тестирования контроллеров и моделей

В процессе разработки должное внимание было уделено обработке исключительных ситуаций и оповещении разработчика о произошедшей ошибке. Поскольку в отчете об ошибке генерируется важная для разработчика информация, пользователь не должен ее видеть. При возникновении ошибки пользователь лишь увидит страницу, с кодом ошибки в окне браузера (cм. рисунок 5.3).

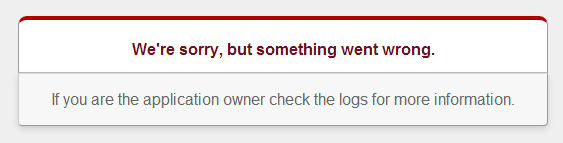


Рисунок 5.3 – Внешний вид страницы сообщения об ошибке

Все возникающие ошибки отправляются на сторонний сервис. Также на случай недоступности стороннего сервиса все ошибки дублируются в журнале логов сервера. Ниже приведен код обработчика исключений.

rescue => e

log(exception.message)

Airbrake.notify(exception)

raise boom

end

Помимо логических ошибок, возникающих в процессе эксплуатации программного средства, могут возникать различного рода ошибки, связанные с работой сервера. Примерами ошибок могут служить недоступность сервера в связи с нарушением работы магистрального провайдера интернета или недостаточное количество свободного места на диске для сохранения пользовательской информации. Для мониторинга ошибок данного рода в ходе эксплуатации программного средства задействована библиотека NewRelic. Данная библиотека агрегирует различного рода информацию и сервере приложения и посылает разработчикам уведомление в случае возникновения аварийного состояния на сервере. Одновременно с этим данная библиотека предоставляет возможность мониторинга загрузки сервера. На рисунке 5.4 представлен пример страницы мониторинга приложения.

Специфика реализованного модуля анализа данных профессиональных сетей привела к необходимости включения в состав модуля системы интерактивной работы Jupyter, предназначенной для визуального контроля собранных данных и интерактивного взаимодействия с программной средой в контексте модуля анализа данных. Действие в контексте окружения программного модуля анализа данных позволяет использовать Jupyter для обращения к классам моделей и, с помощью возможностей, предоставляемых ORM, получение доступа к импортированным данным. Таким образом, проведение испытаний заключается во взаимодействии пользователя программного модуля с интерфейсом среды Jupyter, пример которого представлен на рисунке 5.5.

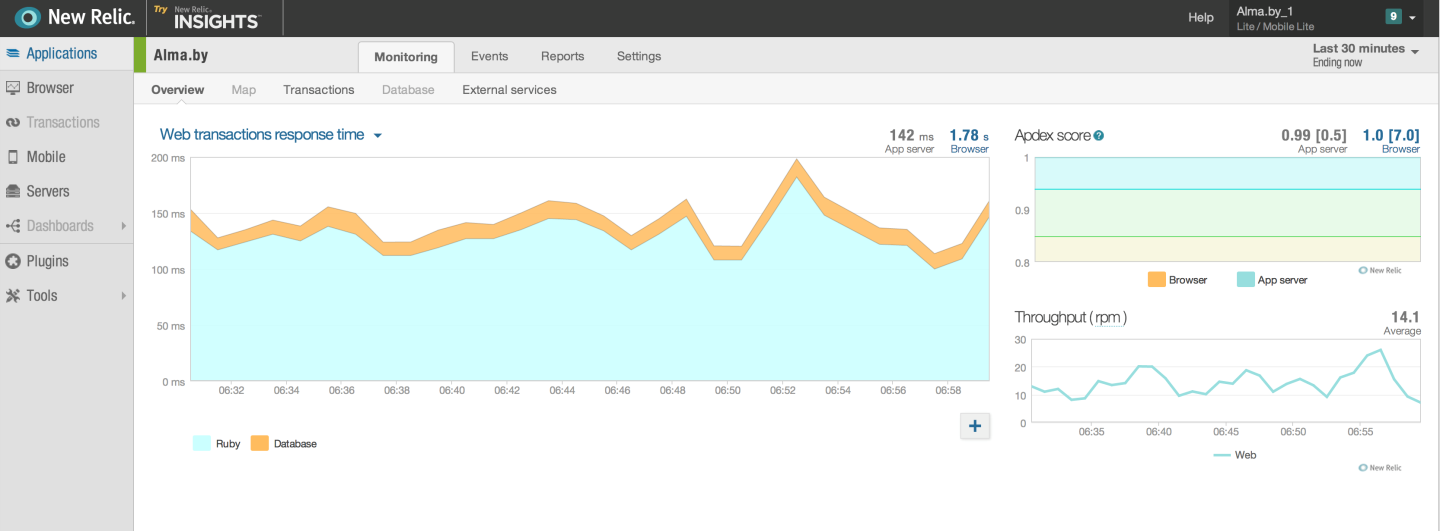


Рисунок 5.4 – Страница мониторинга загрузки сервера

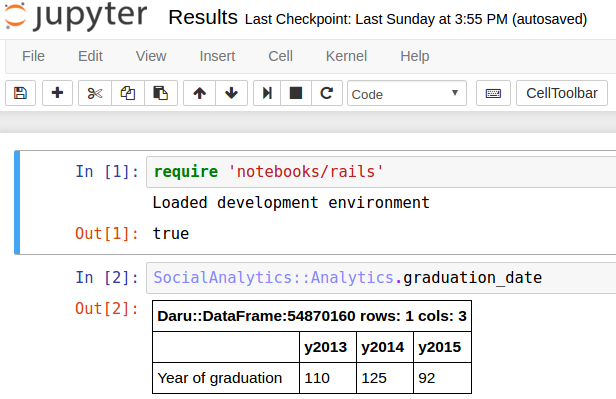


Рисунок 5.5 – Интерфейс среды Jupyter

Таким образом, выполнение испытаний разработанного программного модуля сводится к запуску пользователем процесса получения данных, описанного на диаграмме последовательности (см. чертеж ГУИР.400201.003 РР.3), а затем, на основании полученных данных, проведения необходимых исследований с помощью вызова заранее подготовленных методов анализа данных с помощью интерфейса системы Jupyter.

Например, на рисунке 5.6 показан пример исследования размеров компаний, в которых работают в настоящий момент студенты специальности ВМСиС с выводом полученных данных в удобном для восприятия человеком виде и построения наглядного графика.

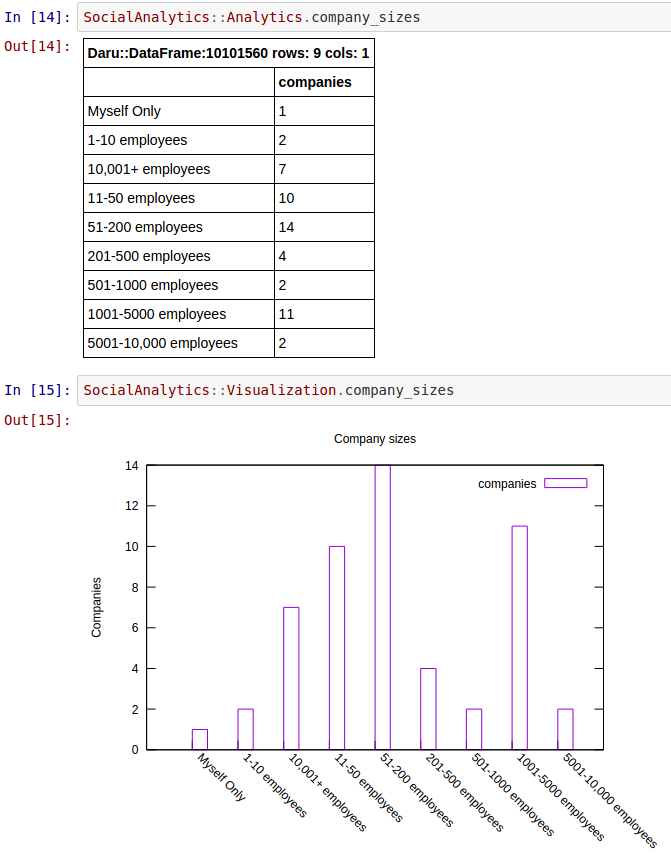


Рисунок 5.6 – Пример испытания методов анализа данных

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **6.1** Установка системы на удаленном сервере

При работе веб-приложений за обработку пользовательских запросов отвечает удаленный сервер. Для обеспечения работоспособности программного модуля на удаленном сервере необходимо тщательно подобрать программное обеспечение, однако такая возможность присутствует.

При разработке программного средства использовался язык программирования Ruby. Использование данного языка программирования накладывает определенные ограничения при выборе операционной системы, наиболее распространенным вариантом на сегодняшний день является сервер под управлением Ubuntu 16.04 LTS. Наличие в названии аббревиатуры LTS (Long Term Support) означает поддержку данной операционной системы сроком на 5 лет.

Разработанный программный модуль не является очень требовательным к характеристикам сервера. Однако для нормальной, стабильной работы следует придерживаться минимальных требований, приведенных в таблице 6.1

Таблица 6.1 – Минимальные системные требования

|  |  |
| --- | --- |
| Параметр | Минимальное значение |
| Операционная система | Ubuntu Linux, другая UNIX OS |
| Процессор | Intel i3 4330 |
| Оперативная память | 512 MB DDR3 |
| Жесткий диск | 20 GB |
| Сетевое подключение | 10 MB/s |

После установки операционной системы на сервере необходимо установить и настроить интерпретатор языка Ruby. Для выбранной операционной системы доступна установка средствами стандартного пакетного менеджера. Однако для установки наиболее актуальной версии интерпретатора следует выполнять установку с использование специальной утилиты RVM (Ruby Version Manager) [26]. В задачи RVM входит физическое разделение версий Ruby и наборов библиотек, а также предоставление возможности иметь несколько версий языка Ruby и переключения между ними. RVM является не единственным средством для выполнения данных задач, также доступна утилита rbenv, являющаяся более упрощенным вариантом RVM. При том же объеме возможностей для управления версиями языка Ruby, rbenv не поддерживает управление наборами библиотек – так называемыми гемсетами. Приведем описание развертывания системы на базе использования RVM. Для установки данной утилиты требуется выполнить следующую команду:

curl -sSL https://get.rvm.io | bash -s stable

Далее устанавливаем стабильную версию языка Ruby. На момент написания данной работы таковой является версия 2.3.1. Для установки необходимо выполнить команду:

rvm install ruby

После установки языка Ruby нам понадобится установить библиотеки bundler и rails, которые являются необходимой базой для развертывания всего приложения:

gem install bundler rails

Веб-сервер может просто отдавать контент расположенный на локальном сервере, либо он может работать в режиме кэширования, обращаясь за контентом к основному серверу, а в последствии сохранять его в локальном кэше. В первом случае необходимо выполнять репликацию сторонними утилитами. Во втором же, репликация выполняется непосредственно проксирующим веб-сервером.

Получив первый запрос на скачивание файла, проксирующий сервер переправляет его на основной сервер, а после успешной обработки запроса этот файл попадает в локальный кэш. Остальные запросы на скачивание этого файла будут обрабатываться уже из локального кэша, без обращения к основному серверу. Кэширование является самым распространённым методом реализации сети доставки контента (Content Delivery Network), поскольку позволяет максимально эффективно использовать узлы, располагая на них не весь контент, а только наиболее востребованный.

Проксирующим сервером для отдачи статических картинок является nginx. Для установки данного сервера необходимо выполнить:

sudo apt-get install nginx

Основная настройка сервера завершена. Для развертывания приложения на данном сервере в проекте необходимо исправить файл config/deploy.rb для использования веб-сервера Unicorn и PostgreSQL. Так же необходимо изменить настройки Capistrano, который используется для создания скриптов при развертывании и запуске приложения, после чего необходимо запустить команду:

cap deploy production

Произойдет автоматическая установка системы на удаленном сервере. Для упрощения внесения изменения в файл настроек config/deploy.rb в нем имеются комментарии, поясняющий каждую из опций.

## **6.2** Установка системы локально

Для установки системы на локальной машине необходимо повторить все этапы настройки окружения, описанные для случая с удаленным сервером. Отличие заключается в желательной установке системы интерактивной работы Jupyter.

Так как система Jupyter изначально была разработана на языке программирования Python, для ее установки потребуется настройка окружения и установка библиотек данного языка. Изначально в поставку большинства дистрибутивов Linux включен язык Python версии 2.7. Для работы Jupyter рекомендуется использовать более новую версию 3.5. Использование двух разных версий одного программного пакета в одной системе, как известно, приводит к различного рода проблемам в работе. Аналогом утилиты RVM для Ruby в мире Python является утилита Anaconda [27]. После установки данной утилиты необходимо создать новое окружение для Python 3.5:

conda create –n py35 python=3.5 anaconda

С помощью Anaconda также значительно облегчается установка системы Jupyter. В отличие от значительно более сложной ручкой установки, используя Anaconda достаточно выполнить следующую команду:

conda install jupyter

После выполнения установки языка программирования Python, а также системы интерактивных вычислений Jupyter и всех необходимых для функционирования программного модуля библиотек, представленных в файле Gemfile.rb, для запуска интерфейса Jupyter на порту 8888 необходимо выполнение всего двух команд:

source activate py35

rake jupyter:notebook

Первой командой в настоящем окружении активируется использование Python 3.5, вторая – выполняет запуск системы Jupyter, стартовый экран которой изображен на рисунке 6.1, в контексте настоящего дипломного проекта. Таким образом, установка приложения на локальную машину позволяет использовать визуальный интерфейс библиотеки Jupyter в качестве средства интерактивной работы.



Рисунок 6.1 – Стартовый экран системы Jupyter

## **6.3** Описание основных функций программного модуля

Программный модуль реализует систему, позволяющую проводить извлечение данных из социальной сети Facebook, профессиональной сети LinkedIn и сервиса совместной работы Github. Все необходимые манипуляции могут проводиться с помощью системы Jupyter.

Для начала работы следует импортировать данные о выпускниках определенных лет обучения с помощью специально созданного веб-интерфейса загрузки файлов, доступного по адресу localhost:3000/import и изображенного на рисунке 6.2. После этого следует запустить импорт пользователей из социальной сети Facebook, так как именно для этих двух способов импорта с высокой степенью достоверности можно утверждать, что полученные в результате импорта лица действительно относятся к выбранной специальности.

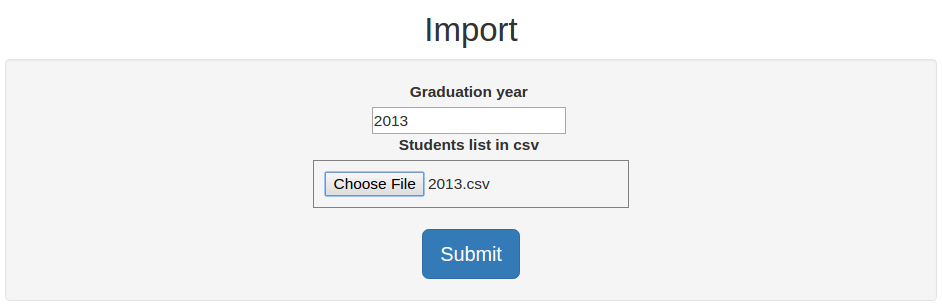


Рисунок 6.2 – Режим импорта данных из файла csv

Для запуска импорта пользователей из группы в социальной сети Facebook в среде Jupyter следует выполнить команду, запуск которой приведет к наполнению базы данных программного модуля различного рода информацией о пользователях данной социальной сети:

SocialAnalytics.handle\_facebook

После импорта данных о пользователях Facebook в системе накапливается достаточно информации о выпускниках специальности ВМСиС, на основании которой становится возможным импорт данных из сервисов LinkedIn и Github. Команды для запуска импорта аналогичны команде для Facebook:

SocialAnalytics.handle\_linkedin

SocialAnalytics.handle\_github

В результате для пользователей, импортированных вручную из списков и полученных с помощью API Facebook, будет проведена попытка привязки аккаунтов в LinkedIn и Github, а также будет запущено получение различного рода дополнительной информации из данных источников. После завершения данного этапа импорта данных по адресам localhost:3000/map/users и localhost:3000/map/companies становится доступна интерактивная мировая карта с распределением, соответственно, пользователей и компаний по странам мира с указанием количества объектов в каждой конкретной стране, внешний вид которой показан на рисунке 6.3. Кроме того, становится возможным выполнение работ по анализу данных и построению графиков в системе интерактивных вычислений Jupyter.

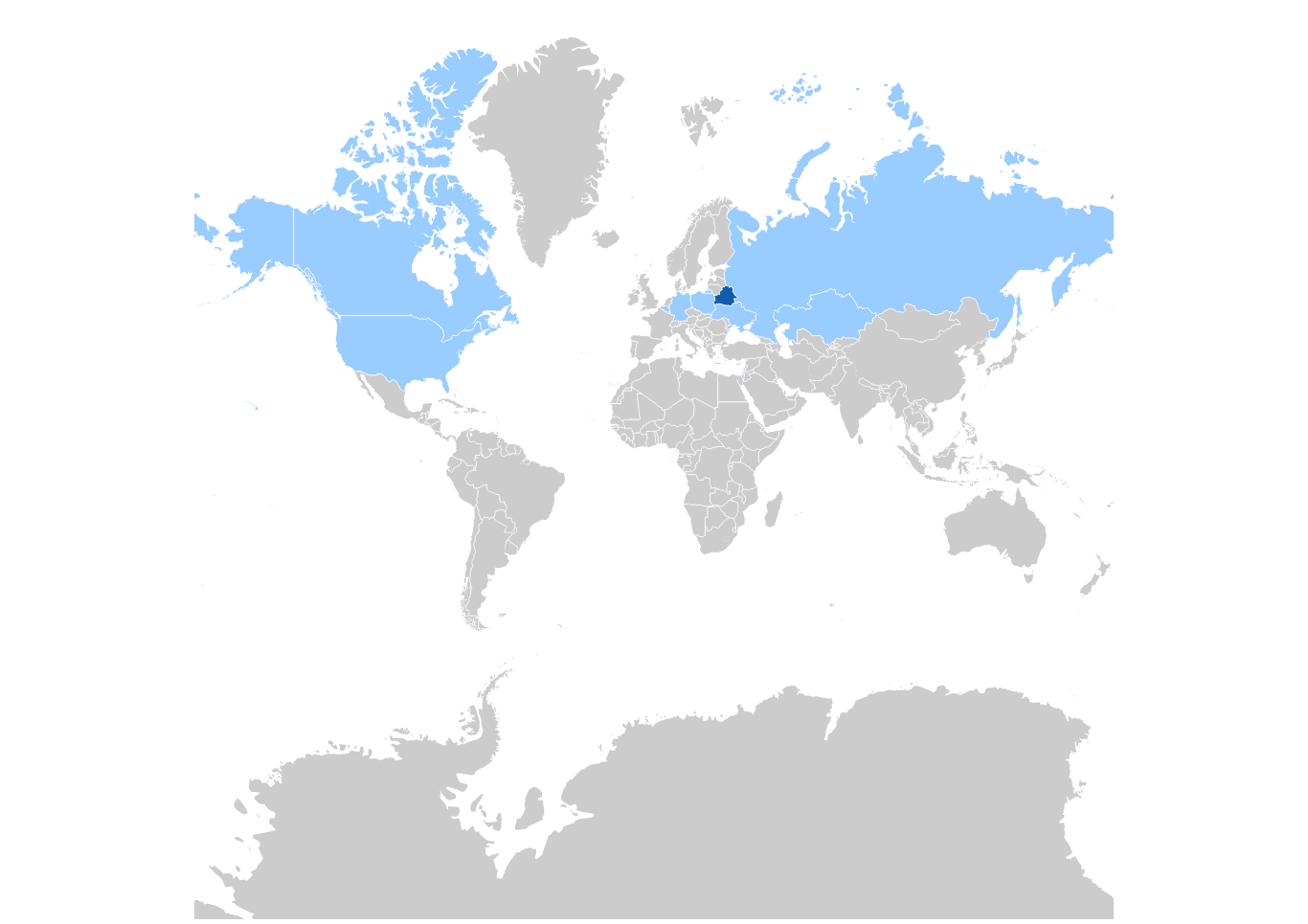


Рисунок 6.3 – Карта распределения пользователей по странам мира

# **7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

## **7.1** Характеристика программного продукта

Целью данного дипломного проекта является разработка программного модуля анализа данных профессиональных социальных сетей. Результатом работы программного модуля следует считать информацию о пользователях социальных сетей, представленную в виде отчетов и инфографики.

На момент написания дипломного проекта в открытом доступе не были выявлены подобные программные модули, автоматизирующие процесс анализа данных, полученных из профессиональных социальных сетей. Тем не менее, существуют аналоги, которые не лишены недостатков и недостаточно хорошо соответствуют выбранной предметной области. Разрабатываемый программный модуль лишен недостатков подобных программных модулей, выявленных в свободном доступе. Использование программного модуля позволит сэкономить время работников сферы менеджмента персонала и образования, тем самым существенно повысив производительность их труда.

Целью технико-экономического обоснования является определение экономической эффективности создания рассматриваемого программного обеспечения и дальнейшего его применения.

## **7.2** Расчет сметы затрат, цены и прибыли

Разрабатываемый программный модуль относится к 1-й категории сложности, так как имеет в составе сложный интеллектуальный языковой интерфейс с пользователем и широко применяет машинную графику для представления результатов анализа. Модуль анализа данных профессиональных социальных сетей относится к программному обеспечению общего назначения. Расчёты выполнены на основе предоставленного методического пособия. Необходимые экономические показатели для расчётов приведены в таблице 7.1.

Таблица 7.1 – Экономические показатели

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Буквенные обозначения | Единицы измерения | Количество |
| Коэффициент новизны | Кн | единиц | 0,7 |
| Категория сложности | - | единиц | 1 |
| Дополнительный коэффициент сложности | Ксл | единиц | 0,12 |
| Поправочный коэффициент, учитывающий использование типовых программ | Кт | единиц | 0,8 |
| Продолжительность рабочего дня | Тч | ч | 8 |

*Продолжение таблицы 7.1*

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Буквенные обозначения | Единицы измерения | Количество |
| Тарифная ставка 1-го разряда | Тм1 | тыс.руб. | 500 |
| Отчисления в фонд социальной защиты населения | Зсз | % | 34 |
| Отчисления в страховой фонд | Ннс | % | 0,6 |
| НДС | Ндс | % | 20 |
| Налог на прибыль | Нп | % | 18 |
| Налог на недвижимость |  | % | 1 |
| Ставка рефинансирования |  | % | 22 |

Объём функций, входящих в состав программного модуля, и объём приведён в таблице 7.2.

Таблица 7.2 – Перечень и объем функций

|  |  |  |
| --- | --- | --- |
| Номер функции | Наименование (содержание) | Объем функции, LOC  по каталогу () |
| 101 | Организация ввода информации | 150 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 |
| 201 | Генерация структуры базы данных | 4300 |
| 204 | Обработка наборов и записей базы данных | 2670 |
| 208 | Организация поиска и поиск в базе данных | 5480 |
| 210 | Загрузки баз данных | 2780 |
| 305 | Обработка файлов | 720 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 |
| 507 | Обеспечение интерфейса между компонентами | 970 |
| 701 | Математическая статистика и прогнозирование | 9320 |
| 704 | Процессор отчетов | 3200 |
| 707 | Графический вывод результатов | 480 |
| Итого |  | 30930 |

Объём функций выражается количеством строк исходного кода. Объём программного средства определяется путём суммирования объёма составляющих его функций. Объём функций, входящих в состав программного средства, и объём приведён в таблице 7.2.

По объему программного обеспечения и нормативам затрат труда в расчете на единицу объема определяется нормативная и общая трудоемкость разработки программного обеспечения. На основании принятого к расчету объема и категории сложности программного обеспечения определяется нормативная трудоемкость .

Наличие сложного интеллектуального языкового интерфейса с пользователем и машинной графики позволяет применить к объему программного обеспечения усложняющий коэффициент, который определяется по формуле:

|  |  |
| --- | --- |
|  | (7.1) |

Поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей показывает, какая часть от целого программного продукта использует стандартные модули. В нашем случае используется от 20% до 40% стандартных модулей, значит . Разрабатываемое программное средство имеет категорию новизны «В», поэтому коэффициент, учитывающий степень новизны программного обеспечения . Таким образом, общая трудоемкость рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.2) |

После определения общей трудоемкости разработки программного обеспечения, рассчитывается число исполнителей по формуле:

|  |  |
| --- | --- |
|  | (7.3) |

где – общая трудоёмкость;

– срок директивной разработки проекта (6 месяцев или 0,5 года);

– эффективный фонд рабочего времени, рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (7.4) |

где – количество календарных дней в году ( дн.);

– количество выходных дней ( дн.);

– количество праздничных дней в году, не совпадающих с выходными ( дн.);

– количество дней отпуска ( дн).

|  |  |
| --- | --- |
|  | (7.5) |

Отсюда численность рабочих получаем:

|  |  |
| --- | --- |
|  | (7.6) |

При решении сложных задач с длительным периодом разработки ПО трудоемкость определяется по стадиям разработки: техническое задание (ТЗ), эскизный проект (ЭП), технический проект (ТП), рабочий проект (РП), внедрение (ВН). При этом трудоемкость разработки ПО отличается в зависимости от стадий. Общий вид формулы, по которой рассчитывается трудоемкость изготовления ПС:

|  |  |
| --- | --- |
| , | (7.4) |

где – трудоемкость изготовления ПО на данной стадии;

– нормативная трудоемкость;

– удельный вес трудоемкости выбранной стадии разработки ПО в общей трудоемкости разработки ПО.

Результаты расчетов трудоемкости по стадиям сведены в таблицу 7.3.

Таблица 7.3 – Расчет общей трудоемкости разработки ПО и численности исполнителей с учетом стадий

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1. Коэффициенты удельных весов трудоемкости стадии разработки ПО () | 0,09 | 0,07 | 0,07 | 0,61 | 0,16 | 1,00 |
| 2. Коэффициент сложности ПО | 1,12 | 1,12 | 1,12 | 1,12 | 1,12 |  |
| 3. Коэффициент, учитывающий использование стандартных модулей |  |  |  | 0,80 |  |  |
| 4. Коэффициент, учитывающий новизну ПО () | 0,7 | 0,7 | 0,7 | 0,7 | 0,7 |  |
| 5. Общая трудоемкость ПО (), чел./дн. | 58,56 | 45,55 | 45,55 | 317,55 | 104,12 | 571,33 |

На основе уточненной трудоемкости разработки ПО с использованием формулы 7.3 найдем общую численность разработчиков, которые требуются, чтобы вложиться в заданные сроки:

Общая трудоемкость, плановая численность и плановые сроки разработки являются базой для расчета основной заработной платы исполнителей. В соответствии со штатным расписанием, в проекте будут заняты лица, представленные в таблице 7.4. Как можно заметить из приведенной таблицы, всего в проекте планируется участие пяти разработчика, из которых один будет являться руководителем проекта. Несмотря на некоторую вольность данного предположения, в целом оценка трудозатрат на данный момент выглядит близкой к реальности.

Таблица 7.4 – Лица, занятые в проекте

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнители | Количество ставок | Разряд | Тарифный коэффициент |
| Руководитель проекта | 1 | 16 | 3,72 |
| Программист 1-й категории | 4 | 14 | 3,25 |

Тарифная ставка 1-го разряда на предприятии равна 500 000 руб. Рассчитаем месячные тарифные ставки руководителя:

|  |  |
| --- | --- |
|  | (7.7) |

и программиста:

|  |  |
| --- | --- |
|  | (7.8) |

Теперь рассчитываем дневную тарифную ставку руководителя путём деления месячной ставки на количество рабочих дней в месяце дня:

|  |  |
| --- | --- |
|  | (7.9) |

и программиста:

|  |  |
| --- | --- |
|  | (7.10) |

Основная заработная плата исполнителей на конкретное программное средство определяется по формуле:

|  |  |
| --- | --- |
|  | (7.11) |

где n – количество исполнителей на конкретное программное средство;

– дневная тарифная зарплата i-го исполнителя (руб.);

– эффективный фонд рабочего времени i-го исполнителя (дн.);

– коэффициент премий (можно принять ).

Тогда основная зарплата исполнителей равна:

|  |  |
| --- | --- |
|  | (7.12) |

Дополнительная заработная плата на конкретное программное обеспечение включает выплаты, предусмотренные законодательством о труде (оплата отпусков и другие) и определяется по нормативу в процентах к основной заработной плате:

|  |  |
| --- | --- |
|  | (7.13) |

где – норматив дополнительной заработной платы ().

В нашем случае, дополнительная зарплата будет равна:

|  |  |
| --- | --- |
|  | (7.14) |

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной заработной платы и вычисляются по формуле:

|  |  |
| --- | --- |
|  | (7.15) |

По статье «Материалы» отражаются расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки программного обеспечения. Норма расхода материалов в суммарном выражении () определяются либо в расчете на 100 строк исходного кода, либо в процентах к основной заработной плате разработчиков (3-5%). В нашем случае будем использовать процентное соотношение, причем примем, что от основной зарплаты:

|  |  |
| --- | --- |
|  | (7.16) |

Расходы по статье «Машинное время» () включают оплату машинного времени, необходимого для разработки и отладки программного средства. Они определяются в машино-часах по нормативам на 100 строк исходного кода () машинного времени в зависимости от характера решаемых задач и типа программного средства.

|  |  |
| --- | --- |
|  | (7.17) |

где – цена одного машино-часа, тыс.руб ( тыс.руб);

– общий объем программного средства, LOC;

– норматив расхода машинного времени на отладку 100 строк кода, машино-часов.

|  |  |
| --- | --- |
|  | (7.18) |

Расходы на научные командировки берутся либо по смете научных командировок, разрабатываемой на предприятии, либо в процентах от основной заработной платы исполнителей. Возьмем норматив командировочных расходов от основной заработной платы:

|  |  |
| --- | --- |
|  | (7.19) |

Расходы по статье «Прочие затраты» включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются в процентах к основной заработной плате. Возьмем норматив прочих расходов от основной заработной платы:

|  |  |
| --- | --- |
|  | (7.20) |

Затраты по статье «Накладные расходы» связаны с содержанием аппарата управления, вспомогательных хозяйств и опытных производств, а также с расходами на общехозяйственные нужды. Определяются по нормативу в процентах к основной заработной плате:

|  |  |
| --- | --- |
|  | (7.21) |

где – норматив накладных расходов в целом по научной организации в процентах (для бюджетных организаций норматив устанавливается в пределах 100%, для иных организаций можно брать реальные проценты, установленные в организации).

|  |  |
| --- | --- |
|  | (7.22) |

Общая сумма расходов по всем статьям на программное обеспечение представляет полную себестоимость программного обеспечения:

|  |  |
| --- | --- |
|  | (7.23) |

Для определения цены программного обеспечения необходимо рассчитать плановую прибыль. Прибыль рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.24) |

где – плановая прибыль от реализации программного обеспечения, руб;

– уровень рентабельности программного обеспечения в процентах (уровень рентабельности можно принять равным ), тогда:

|  |  |
| --- | --- |
|  | (7.25) |

Рентабельность и прибыль создаваемого программного обеспечения определяется исходя из результатов анализа рыночных условий, переговоров с заказчиком и согласования с ним отпускной цены. Не стоит исключать и влияние других факторов, однако именно реальные рыночные условия должны быть основным влияющим элементом в расчетах, основанных на применении наиболее современных экономических моделей. Тем не менее, в странах с плановой экономикой это правило часто не выполняется.

Таблица 7.4 – Расчет себестоимости и прибыли программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Усл. обозн. | Значение (тыс.руб) | Методика расчёта |
| Основная заработная плата исполнителей | Зо |  | Определяются на основании расчетов |
| Дополнительная заработная плата исполнителей | Зд |  |  |
| Материалы и комплектующие | Рм |  |  |
| Отчисления в фонд социальной защиты населения | Рсоц |  |  |
| Машинное время | Рмв |  |  |
| Расходы на научные командировки | Рк |  |  |

*Продолжение таблицы 7.4*

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Усл. обозн. | Значение (тыс.руб) | Методика расчёта |
| Прочие прямые расходы | Рпр |  |  |
| Накладные расходы | Рн |  |  |
| Полная себестоимость | Сп |  | Сп = Зо + Зд + Рм + Рсоц + + Рмв + Рк + Рпр + Рн |
| Прогнозируемая прибыль | По |  |  |
| Прогнозируемая цена без налогов (цена предприятия) | Цп |  |  |
| Налог на добавленную стоимость (НДС) | НДС |  |  |
| Прогнозируемая отпускная цена | Цот |  |  |
| Чистая прибыль предприятия | Пч |  |  |
| Месячная тарифная ставка 1-го разряда | Тм.1 | 500 |  |
| Ставка налога на прибыль | Нп | 18% |  |
| Ставка НДС | Ндс | 20% |  |

После расчета прибыли от реализации определяется прогнозируемая цена программного обеспечения без налогов:

|  |  |
| --- | --- |
|  | (7.26) |

Отпускная цена (цена реализации) программного средства включает в себя налог на добавленную стоимость (НДС):

|  |  |
| --- | --- |
|  | (7.27) |

где – ставка далога на добавленную стоимость, .

В нашем случае сумма налога на добавленную стоимость составит:

|  |  |
| --- | --- |
|  | (7.28) |

Прогнозируемая отпускная цена является суммой цены предприятия и размера налога на добавленную стоимость и вычисляется по формуле:

|  |  |
| --- | --- |
|  | (7.29) |

У разработчика экономический эффект будет выражаться в виде чистой прибыли, остающейся в его распоряжении. Она вычисляется по формуле:

|  |  |
| --- | --- |
|  | (7.30) |

где – ставка налога на прибыль ().

Таким образом, у разработчика останется сумма чистой прибыли:

|  |  |
| --- | --- |
|  | (7.31) |

## **7.3** Расчет экономического эффекта у пользователя

Для определения экономического эффекта от использования нового ПС у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПС (расходы на заработную плату с начислениями, материалы, машинное время) с расходами по соответствующим статьям при использовании прежнего варианта ПС.

Общие капитальные затраты для пользователя составят:

|  |  |
| --- | --- |
| , | (7.32) |

где Кпр – затраты пользователя на приобретение по отпускной цене у разработчика, тыс. руб.;

Кос – затраты пользователя на освоение ПО (10 % от Кпр), тыс. руб.;

Кс – затраты пользователя на оплату услуг по сопровождению ПО (20% от Кпр), тыс. руб.;

Ктс – затраты пользователя на доукомплектование ВТ техническими средствами (для данного продукта нет необходимости в доукомплектовании), тыс. руб.;

Коб – затраты на пополнение оборотных средств (30 % от Кпр), тыс. руб.

Экономию затрат на заработную плату из расчета на 1 задачу определим по формуле:

|  |  |
| --- | --- |
|  | (7.33) |

где – среднемесячная заработня плата программиста, тыс. руб.;

– средняя трудоемкость работ в расчете на одну задачу до внедрения нового ПО, чел-час.;

- средняя трудоемкость работ в расчете на одну задачу после внедрения нового ПО, чел-час.;

– продолжительность рабочего дня (8 ч.), ч.;

– среднемесячное количество рабочих дней (22 дн.), дн.

У пользователя среднемесячная зарплата программиста составляет 15 000 тыс. руб., трудоемкость работ до внедрения ПО – 9 чел-час., после – снижается до 6 чел-час. Тогда экономия на заработную плату в расчете на одну задачу составит:

Экономию заработной платы при использовании нового ПО рассчитаем по формуле:

|  |  |
| --- | --- |
|  | (7.34) |

где – объем работ у пользователя.

Запланированное количество задач у пользователя в течение года – 2500. Тогда экономия заработной платы за год;

Экономию начислений на заработную плату определим по формуле:

|  |  |
| --- | --- |
|  | (7.35) |

где – норматив начислений на заработную плату.

По данным пользователя, норматив начислений составляет 50%, тогда:

Общая экономия текущих затрат:

Для пользователя в качестве экономического эффекта выступает чистая прибыль. Следует заметить, что, в общем случае, прибыль не является единственной доступной метрикой для измерения эффективной какого-либо решения. Однако для расчета в нашем случае подойдет формула:

Расчет экономического эффекта за весь период использования ПО (4 года) целесообразно предоставить в таблице 7.5. Полученные суммы результата (чистой прибыли) и затрат (капитальных вложений) по годам приведем к единому моменту времени – расчетному году (2016) путем умножения результатов и затрат на коэффициент дисконтирования αt, который рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.36) |

где – норма дисконта ( = 0,25);

ti – порядковый номер года, результаты и затраты которого приводятся к расчетному;

tр – расчетный год (tр = 1).

В данном примере используются следующие коэффициенты :

* αt0 = 1,000 – первый расчетный год;
* αt1 = (1 + 0,25)2016-2017 = 0,8 – 2017 расчетный год;
* αt2 = (1 + 0,25)2017-2018 = 0,64 – 2018 расчетный год;
* αt3 = (1 + 0,25)2018-2019 = 0,512 – 2019 расчетный год.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 7.5 – Расчет экономического эффекта от использования нового ПС | | | | | | | | | |
| Показатели | | Ед. изм. | Годы | | | | | | |
| 2016 | | 2017 | | | 2018 | 2019 |
| Результаты: | | | | | | | | |  |
| Прирост прибыли за счет экономии затрат (Пч) | | руб. |  | | 786216000 | | | 786216000 | 786216000 |
| То же с учетом фактора времени | | руб. |  | | 628972800 | | | 503178240 | 402542592 |
| Затраты: | | | | | | | | | |
| *Продолжение таблицы 7.5* | | | | | | | | | |
| Показатели | Ед. изм. | | | Годы | | | | | |
| 2016 | | 2017 | 2018 | | 2019 |
| Приобретение ПС (Кпр) | руб. | | | 289594000 | |  |  | |  |
| Освоение ПС (Кос) | руб. | | | 28959400 | |  |  | |  |
| Сопровождение (Кс) | руб. | | | 57918800 | | 57918800 | 57918800 | | 57918800 |
| Пополнение оборотных средств (Коб) | руб. | | | 2500000 | | 2500000 | 2500000 | | 2500000 |
| Всего затрат | руб. | | | 60418800 | | 60418800 | 60418800 | | 60418800 |
| То же с учетом фактора времени | руб. | | | 60418800 | | 48335040 | 38668032 | | 30934425 |
| Экономический эффект: |  | | |  | |  |  | |  |
| Превышение результатов над затратами | руб. | | | -60418800 | | 725797200 | 725797200 | | 725797200 |
| Пополнение оборотных средств (Коб) | руб. | | | 2500000 | | 2500000 | 2500000 | | 2500000 |
| То же нарастающим итогом | руб. | | | -60418800 | | 121678400 | 847475600 | | 1573272800 |
| Коэффициент приведения | ед. | | | 1 | | 0.8 | 0.64 | | 0.512 |

Индекс рентабельности проекта (Ри) показывает эффективность проекта по уровню доходов на единицу затрат. Рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.30) |

где Пчt – прибыль за определенный промежуток времени (руб.);

Коt – затраты на ПС за определенный промежуток времени (руб.).

В результате расчетов получаем:

* чистый дисконтированный доход в 2017 году равен 121678400 рублей;
* программное средство окупится на втором году;
* коэффициент рентабельности составил 8,60.

Таким образом, разработка и применение программного продукта является экономически целесообразной.

Иными словами, проведение расчета экономической эффективности для разработчика и лишь после этого для пользователя позволило учесть максимальное количество действующих факторов и предоставить наиболее точные итоговые величины. Следует заметить, что полученные значения являются, без сомнений, точными с точки зрения экономической науки. Однако реальность накладывает на идеализированную картину проведенного технико-экономического обоснования свои коррективы. Так, для разработчика объем работ и, соответственно, структурный состав команды разработчиков, затраченное время и объем необходимых денежных средств, были рассчитаны на основании в значительной степени устаревших табличных значений стандартного объема строк кода для определенного перечня стандартных модулей. Несмотря на то, что настоящий дипломный проект удалось практически в полном объеме модулей соотнести с предлагаемой таблицей значений, между цифрами объемов программных модулей обнаруживаются значительные разбежки. Так, общая сумма строк кода для описанного проекта была рассчитана на уровне 31000 строк, с учетом наличия программных модулей большого объема, таких как, например, математическая статистика и прогнозирование. Проведенные с помощью утилиты cloc измерения уже реализованного дипломного проекта показали наличие около 1800 строк кода в общей сумме, то есть, для всех используемых в проекте языков программирования. С одной стороны, это можно объяснить использованием языка Ruby, славящегося своей лаконичностью и выразительностью, а также широким функционалом стандартной библиотеки. С другой, методика оценки трудозатрат не учитывает или учитывает в недостаточной степени использование для разработки сложных программных модулей сторонних библиотек, что является конечной целью любой разработки.

Тем не менее, перечисленные выше неточности скорее изменяют итоговый баланс в сторону окупаемости еще сильнее. Программный продукт способен выйти на уровень безубыточности не на второй год использования, а, возможно, и на первый. Это при условии распространения программного модуля по одной из платных схем. Развитие и использование программного продукта в виде открытого исходного кода, что является логичным и предпочтительным путем дальнейшей разработки, уменьшает стоимость программного продукта для клиента и вовсе до пренебрежимо малых величин.

# ЗАКЛЮЧЕНИЕ

В рамках работы над дипломным проектом был спроектирован и создан программный модуль на языке Ruby, позволяющий получать и анализировать данные из социальной сети Facebook, профессиональной социальной сети LinkedIn и сервиса совместной работы Github.

С целью проверки работоспособности разработанного программного модуля было проведено исследование данных студентов и выпускников специальности ВМСиС в вышеописанных социальных сетях. Среди наиболее интересных из полученных результатов стоит назвать статистику распределения студентов по странам, данные о навыках пользователей из профессиональной социальной сети LinkedIn в совокупности с информацией о частоте упоминания каждого отдельного навыка, а также информацию о степени использования студентами специальности различных языков программирования в сервисе Github.

Для выполнения задачи представления результатов аналитики в удобном для восприятия человеком виде, в составе проекта были разработаны инструмента для интеграции программного модуля в систему интерактивной работы. Использование инфографики позволяет наглядно отобразить статистику по странам для пользователей и компаний, в которых соответствующие пользователи работаю, на интерактивной мировой карте. Система интерактивных вычислений, в свою очередь, позволяет проводить новые исследования на основании данных, полученных в процессе работы над настоящим дипломным проектом, в том числе, людям, не имеющим необходимой базы технических навыков.

В процессе получения данных из внешних источников была приобретена экспертиза в области использования программных интерфейсов, предоставляемых исследуемыми социальными сетями, а также в области получения наиболее многообещающих для анализа данных в обход общеизвестных путей.

В качестве возможных вариантов улучшения данного программного модуля следует назвать увеличение количества поддерживаемых источников информации, а также разработку дополнительных инструментов для анализа данных на базе существующего ядра.

Представляет интерес дальнейшее использование разработанного инструментария для получения университетом и кафедрой данных о профессиональных предпочтениях студентов с целью улучшения качества образовательных услуг и предоставления абитуриентам исчерпывающей информации о профессиональных знаниях и умениях студентов, получающих высшее образование на кафедре ЭВМ.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Linkedin takes aim at developers with plans to lock down most of its APIs [Электронный ресурс]. – Режим доступа: *http://thenextweb.com/dd/2015/02/12/linkedin-takes-aim-developers-plans-lock-apis/#gref*.
2. Github is the next big social network, powered by what you do, not who you know [Электронный ресурс]. – Режим доступа: *http://www.forbes.com/sites/anthonykosner/2012/07/15/github-is-the-next-big-social-network-powered-by-what-you-do-not-who-you-know/#1a76409179fd*.
3. Mobile marketing statistics compilation [Электронный ресурс]. – Режим доступа: *http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics*.
4. Системе аналитики Brandwatch [Электронный ресурс]. – Режим доступа: [*https://www.brandwatch.com*](https://www.brandwatch.com)*/*.
5. Система аналитики 33Across [Электронный ресурс]. – Режим доступа: *http://www.33across.com.*
6. Система аналитики Hootsuite [Электронный ресурс]. – Режим доступа: *https://hootsuite.com*
7. Moz [Электронный ресурс]. – Режим доступа : *https://moz.com.*
8. Язык Ruby: история становления и перспективы развития [Электронный ресурс]. – Режим доступа: *http://habrahabr.ru/post/131661*.
9. Ruby учебник [Электронный ресурс]. – Режим доступа: *http://ru.wikibooks.org/wiki/Ruby.*
10. Исходный код языка Ruby [Электронный ресурс]. – Режим доступа: *https://github.com/ruby/ruby.*
11. Ruby community gem host [Электронный ресурс]. – Режим доступа: *http://rubygems.org.*
12. Наиболее продвинутая открытая СУБД в мире [Электронный ресурс]. – Режим доступа: *http://postgresql.ru.net/docs/overview.html*.
13. Три ключевых принципа ПО [Электронный ресурс]. – Режим доступа: *https://habrahabr.ru/post/144611.*
14. What is MVC [Электронный ресурс]. – Режим доступа: *http://programmers.stackexchange.com/questions/127624/what-is-mvc-really.*
15. ActiveRecord. Ruby on Rails API. [Электронный ресурс] – Режим доступа: *http://api.rubyonrails.org/classes/ActiveRecord/Base.html.*
16. Bitnami ruby stack [Электронный ресурс]. – Режим доступа: *https://bitnami.com/stack/ruby*.
17. Github API developer guide [Электронный ресурс]. – Режим доступа: *https://developer.github.com/v3*.
18. Octokit official repository [Электронный ресурс]. – Режим доступа: *https://github.com/octokit/octokit.rb*.
19. Scrape the public profile of linkedin page [Электронный ресурс]. – Режим доступа: *https://github.com/yatish27/linkedin-scraper*.
20. A lightweight, flexible library for Facebook with support for OAuth authentication [Электронный ресурс]. – Режим доступа: *https://github.com/arsduo/koala*.
21. Facebook Data Policy [Электронный ресурс]. – Режим доступа: *https://facebook.com/policy.php*.
22. Watir: web application testing in Ruby [Электронный ресурс]. – Режим доступа: *https://watir.com/documentation*.
23. Data analysis in Ruby [Электронный ресурс]. – Режим доступа: *https://github.com/v0dro/daru*.
24. Gnuplot homepage [Электронный ресурс]. – Режим доступа: *https://gnuplot.info*.
25. Project Jupyter [Электронный ресурс]. – Режим доступа: *https://jupyter.org*.
26. RVM подробно [Электронный ресурс]. – Режим доступа: *https://habrahabr.ru/post/120504/*.
27. Get superpowers with Anaconda [Электронный ресурс]. – Режим доступа: *https://continium.io/why-anaconda*.

# ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код некоторых алгоритмов программного модуля

module SocialAnalytics

class << self

attr\_accessor :facebook\_token, :facebook\_email, :facebook\_password,

:linkedin\_email, :linkedin\_password, :github\_login,

:github\_password

end

def self.handle\_facebook(gid = nil)

puts "Start receiving facebook group members...".green

if FacebookData.count.zero?

FacebookScraper.new(facebook\_token, gid).save\_group\_members

end

FacebookGeo.new(false).handle

end

def self.handle\_linkedin

scraper = LinkedinScraper.new

User.where(linkedin: false, linkedin\_scraped: false).each do |user|

scraper.get\_public\_member(user.name)

user.update\_columns(linkedin\_scraped: true)

sleep 1

end

end

def self.handle\_github

scraper = GithubScraper.new

User.where(github: false, github\_scraped: false).each do |user|

scraper.search\_user(user.name)

user.update\_columns(github\_scraped: true)

sleep 1

end

end

end

class GithubScraper

def initialize

@users = []

end

def search\_user(name)

logins = Octokit.search\_users(name).items.map(&:login).first(2)

@users = logins.map{|login| Octokit.user(login)}

save\_user

end

def save\_user

return if @users.empty?

puts "Found Github users".blue

@users.each do |user|

github\_data = GithubData.create(

login: user.login,

name: user.name,

email: user.email,

num\_repos: user.public\_repos,

num\_gists: user.public\_gists,

num\_followers: user.followers,

num\_following: user.following,

blog: user.blog

)

if user.avatar\_url.present?

puts "Saving avatar image".yellow

Image.create(

imageable: github\_data.user,

link: user.avatar\_url

)

end

save\_repositories(user.login)

end

end

def save\_repositories(login)

puts "Saving repositories".green

repositories = Octokit.repositories(login).map(&:id).first(10)

repositories.each do |repo\_id|

repo = Octokit.repository(repo\_id)

repository = Repository.create(

github\_data\_id: GithubData.find\_by\_login(login),

primary\_language\_id: Language.

where(name: repo.language).

first\_or\_create.id,

full\_name: repo.full\_name,

repo\_id: repo\_id,

forks: repo.forks,

issues: repo.issues,

watchers: repo.watchers,

subscribers: repo.subscribers,

stargazers: repo.stargazers,

fork: repo.fork

)

save\_languages(repository)

end

end

def save\_languages(repository)

languages = Octokit.languages(repository.repo\_id)

languages.each do |lang, loc|

language = Language.where(name: lang).first\_or\_create

repository.languages << language

lang\_repo = LanguagesRepositories.where(

language\_id: language.id,

repository\_id: repository.id).first

lang\_repo.update\_attributes(loc: loc)

end

end

end

class LinkedinScraper

def initialize

@profile = nil

@public\_profile = nil

@web\_scraper = LinkedinWebScraper.new

end

def get\_public\_member(name)

@public\_profile = @web\_scraper.scrape(name)

return if @public\_profile.nil?

@profile = begin

Linkedin::Profile.new(

"http://www.linkedin.com/in/#{@public\_profile}",

{ company\_details: true })

rescue

nil

end

save\_public\_member

end

def save\_companies(companies, current = false)

companies.each do |data|

company = Company.where(name: data[:company]).first\_or\_create do |c|

c.description = data[:description]

c.linkedin\_company\_url = data[:linkedin\_company\_url]

c.website = data[:website]

c.company\_size = data[:company\_size]

c.company\_type = data[:company\_type]

c.industry = data[:industry]

c.founded = Preprocessing.

foundation\_date(data[:founded])

c.country = data[:country]

end

if data[:company\_url].present?

Image.create(

imageable: company,

link: data[:company\_logo]

)

end

if !company.address.present? && data[:country].present?

Address.create(

addressable: company,

country: data[:country]

)

end

Position.create(

linkedin\_data: @linkedin\_data,

company\_id: company.id,

title: data[:title],

duration: data[:duration],

start\_date: data[:start\_date],

end\_date: data[:end\_date],

current: current

)

end

end

def save\_public\_member

return if !@profile.present?

puts "Found Linkedin user - ".blue + "#{@profile.first\_name} #{@profile.last\_name}".green

@linkedin\_data = LinkedinData.create(

profile\_id: @public\_profile,

first\_name: @profile.first\_name,

last\_name: @profile.last\_name,

title: @profile.title,

connections: @profile.number\_of\_connections.to\_i,

country: @profile.country

)

if @linkedin\_data.addresses.empty? && @profile.country.present?

puts "Saving linkedin address".yellow

Address.create(

addressable: @linkedin\_data.user,

country: @profile.country

)

end

if @linkedin\_data.skills.empty?

puts "Saving skills".green

@profile.skills.each do |skill\_name|

skill = Skill.where(name: skill\_name).first\_or\_create

@linkedin\_data.skills << skill

end

end

puts "Saving companies".yellow

save\_companies(begin @profile.past\_companies rescue [] end, false)

save\_companies(begin @profile.current\_companies rescue [] end, true)

end

end

class LinkedinWebScraper < WebScraper

def scrape(name)

authorization if !@logged\_in

url = "http://www.linkedin.com/vsearch/p?type=people&keywords=#{name.split(' ').join('+')}"

@results = @agent.get(url).body.scan(/\{"person"\:\{.\*?\}\}/)

return if !@results[0].present?

@profile\_id = begin

JSON.parse(@results[0]).

to\_s.scan(/profile\/view\?id=(.\*?)&/).

compact.first.first

rescue

nil

end

get\_public\_profile

end

private

def authorization

@agent.get('https://www.linkedin.com/')

form = @agent.page.form\_with(:method => 'POST')

form.session\_key = SocialAnalytics.linkedin\_email

form.session\_password = SocialAnalytics.linkedin\_password

resp = @agent.submit(form)

@logged\_in = resp.title == "Welcome! | LinkedIn"

end

end

# ПРИЛОЖЕНИЕ Б

(обязательное)

Интеграция программного модуля в систему Jupyter

namespace :jupyter do

task :notebook do

root = File.absolute\_path(\_\_FILE\_\_ + "/../../..")

env = {

"RUBYLIB" => ($: + [ root ]).join(":")

}

Process.exec(env, "jupyter", "notebook")

end

end

Dir.chdir File.dirname(File.dirname(\_\_FILE\_\_))

unless defined? Rails

APP\_PATH = File.expand\_path('../../config/application', \_\_FILE\_\_)

require File.expand\_path('../../config/boot', \_\_FILE\_\_)

require APP\_PATH

Rails.application.require\_environment!

end

puts "Loaded #{Rails.env} environment"

# ПРИЛОЖЕНИЕ В

(обязательное)

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов