СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяют следующие обозначения и сокращения:

API – Application Programming Interfaces
SPA – Single Page Application
BLL – Business Logic Layer
DAL – Data Access Layer
SDK – Software Development Kit
UML – Unified Modeling Language
PaaS – Platform as a Service
CF – Cloud Foundry
PI – Personality Insights
LIWC – Linguistic Inquiry and Word Count
JSON – JavaScript Object Notation
BSON – Binary JavaScript Object Notation
ВУЗ – Высшее учебное заведение
БГУИР – Белорусский Государственный Университет Информатики и Радиоэлектроники
БД – база данных
ПО – программное обеспечение
ПП – программный продукт
СУБД – система управления базами данных

ВВЕДЕНИЕ

Мы живем в удивительное время, когда компьютеры рассматриваются не только как вычислительные машины, строго следующие набору введенных команд, но и как средства обработки большого объема данных, способные к самостоятельному обучению, построению и оценке гипотез на основе только существенных фактов. Самые передовые корпорации вкладывают огромные средства для создания систем, способных отвечать перечисленным требованиям. Такие системы, называемые когнитивными, способны заменить детерминированные приложения, управляемые деревом решений, на вероятностные системы, способные развиваться вместе со своими пользователями. Когнитивные системы могут найти широкое применение во всех сферах жизни человека.

Недавно IBM открыли пользователям доступ к своей разработке в этой сфере. Проект носит название IBM Watson и предоставляется в виде сервисов для IBM Bluemix. Разработчики со всего мира могут применить принципиально новый подход к решению собственных задач.

Основы выбора профессии закладываются еще в школе, когда на уроках выявляются способности к каким-либо предметам, а во время общения с родителями, учителями и сверстниками – личные качества. Уже в это время можно примерно определиться, в какой из основных сфер человеческой деятельности имеет смысл приложить свои силы. Выбор профессии определяется многими факторами – природными наклонностями, доступностью образования, конъюнктурой рынка труда, культурной средой воспитания. Так или иначе, работа должна приносить не только достойный заработок, но и моральное удовлетворение – в таком случае можно говорить об уверенности в завтрашнем дне.

Целью диплома является создание «Приложения для подбора специальности абитуриента ВУЗа на основе когнитивных сервисов IBM Bluemix», которое позволит подсказать специальность, сравнивая характеристики выпускников и абитуриентов, полученные при помощи когнитивных сервисов.

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

1.1 Single Page Application

Single Page Application (SPA) – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все javascript-файлы (модули, виджеты, контролы и т.д.), а также файлы CSS вместе с загрузкой самой страницы. Если приложение достаточно сложное и содержит богатый функционал, то количество файлов со скриптами может достигать нескольких сотен, а то и тысяч. А загрузка всех скриптов не означает, что при загрузке сайта будут загружены сразу все сотни и тысячи файлов со скриптами.

Application Programming Interface (API) - это интерфейс программирования, набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.

В целом, SPA обладает несколькими весьма весомыми плюсами.

Первым плюсом стоит отметить тот факт, что приложения на SPA отлично работают на устройствах как стационарных, так и мобильных. Персональные компьютеры, планшеты, смартфоны и простые современные телефоны могут беспрепятственно работать с сайтами построенных по принципу SPA. Это значит, что создав одно приложение, получается гораздо большая аудитория пользователей, нежели при использовании стандартного подхода.

Второй плюс – богатый пользовательский интерфейс, так называемый User Experience. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией (в некоторых случаях).

Третий плюс – SPA существенно сокращает так называемые “хождения по кругу”, то есть загрузку одного и того же контента снова и снова**.** Если ваш сайт использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона. Кэширование данных на данном этапе развития достигло высочайших результатов, но если нечего кэшировать, то и время, и ресурсы на это не тратятся.

1.2 Основные понятия IBM Bluemix

Bluemix – это новейшее предложение в серии облачных решений IBM. Эта среда позволяет разработчикам и организациям быстро и легко создавать, развертывать и администрировать приложения в облаке. Bluemix представляет собой реализацию архитектуры IBM Open Cloud Architecture на основе открытого ПО Cloud Foundry, работающего по принципу «платформа как услуга» (Platform as a Service – PaaS). Bluemix предоставляет услуги корпоративного уровня, которые можно легко интегрировать в облачные приложения, не вдаваясь в тонкости их установки и настройки[1].

Cloud Foundry – это платформа, предоставляемая как услуга (PaaS), с открытым исходным кодом, которая позволяет быстро создавать и развертывать приложения в облаке. Поскольку Cloud Foundry - ПО с открытым исходным кодом, она не зависит от поставщика и не привязывает пользователя к чьему-либо программному обеспечению и к какой-либо определенной облачной инфраструктуре. Cloud Foundry абстрагирует базовую инфраструктуру, необходимую для работы облака, позволяя сосредоточиться на создании облачных приложений[1]. Важнейшее достоинство Cloud Foundry заключается в широте выбора. Разработчики и организации могут выбирать:

- **Платформы для разработки:** Cloud Foundry поддерживает Java, Spring, Ruby, Node.js и пользовательские инфраструктуры программирования.
- **Службы приложения:** Cloud Foundry предоставляет поддержку MySQL, MongoDB, PostgreSQL, Redis, RabbitMQ и пользовательских служб.
- **Облака:** разработчики и организации могут работать с Cloud Foundry в общедоступных и частных сетях и в сетях на базе VMWare и OpenStack.

Гибкость выбора технологий достигается в Cloud Foundry посредством buildpack-пакетов - удобного способа упаковки платформ и сред исполнения. Можно использовать buildpack-пакеты, разработанные сообществом, строить их из готовых модулей или создавать самостоятельно с нуля[2]. Другими словами, если вы не можете найти buildpack-пакет для нужной платформы или службы, который соответствовал бы вашим потребностям, вы можете модифицировать один из существующих buildpack-пакетов или создать свой собственный. С помощью buildpack-пакетов компании могут создавать службы корпоративного уровня, такие как облачная платформа Bluemix.

Bluemix – это реализация открытой облачной архитектуры IBM, основанная на Cloud Foundry[1], которая позволяет быстро создавать, развертывать и администрировать облачные приложения. Так как Bluemix основана на Cloud Foundry, вы можете пользоваться всеми ресурсами растущей экосистемы сред исполнения и служб. Помимо дополнительных сред и служб, Bluemix предоставляет панель управления, которая позволяет создавать, просматривать и администрировать приложения и службы, а также следить за использованием ресурсов вашего приложения[1]. Кроме того, панель управления Bluemix предоставляет возможность управлять организациями, пространствами и доступом пользователей.

Bluemix обеспечивает доступ к широкому спектру служб, которые можно встраивать в приложения. Некоторые из этих служб происходят из Cloud Foundry, другие поставляются IBM и сторонними поставщиками. В каталог регулярно добавляются новые и усовершенствованные службы.
Вот некоторые из часто используемых сред исполнения:

- Node.js
- PHP
- Python
- Ruby

Разработчикам Bluemix помогает оптимизировать время, затрачиваемое на создание облачных приложений. Вам больше не придется беспокоиться об установке программного обеспечения и управлении образами виртуальных машин и оборудованием. Чтобы подготовить экземпляры приложений с необходимыми службами поддержки, достаточно нескольких нажатий клавиш. Такое упрощение избавляет от траты драгоценного времени на настройку, выбор конфигурации и устранение неполадок, позволяя вместо этого заняться быстрым внедрением инноваций и реагированием на постоянные изменения требований пользователей.

1.3 Основные понятия когнитивной системы IBM Watson

IBM Watson — одна из первых когнитивных систем в мире. Эта система умеет очень многое, благодаря чему возможности Watson используются во многих сферах — от кулинарии до предсказания аварий в населенных пунктах. В общем-то, большинство возможностей Watson не являются чем-то уникальным, но в комплексе все эти возможности представляют собой весьма мощный инструмент для решения разнообразных вопросов. Например — распознавание естественного языка, динамическое обучение системы, построение и

оценка гипотез. Все это позволило IBM Watson научиться давать прямые корректные ответы (с высокой степенью достоверности) на вопросы оператора. При этом когнитивная система умеет использовать для работы большие массивы глобальных неструктурированных данных, Big Data[3].

Для человека язык — это средство выражения мысли. Мы используем язык для передачи своего мнения, каких-либо данных и сведений. Можем делать прогнозы и формировать теории. Именно язык — краеугольный камень нашего сознания. При этом, вот парадокс, язык человека очень неточный. Многие термины — нелогичны, и компьютерным системам понять нас бывает очень сложно. Например, как может быть тонким голос? Как можно сгореть со стыда? Для машины это — проблема, для человека же — вполне обыденная вещь. Дело в том, что для правильного ответа на вопрос во многих случаях необходимо учитывать имеющийся контекст. При отсутствии достаточной фактической информации трудно правильно ответить на вопрос, даже если вы можете найти точный ответ на элементы вопроса в буквальном смысле.

Многие компьютерные системы способны анализировать язык, но при этом проводится поверхностный анализ. Это может иметь смысл, например, для того, чтобы поставить статистически обоснованную оценку тенденций в изменении эмоций на больших массивах информации. Здесь точность передачи информации не слишком важна, поскольку если даже если предположить, что число ошибочно-позитивных результатов примерно равно числу ошибочно-негативных результатов, то они компенсируют друг друга. Но если значение имеют все случаи, то системы, которые работают с поверхностным анализом языка, уже не могут нормально делать свою работу. Ярким примером сказанному может быть задача для голосового помощника любого из мобильных устройств. Если сказать «найди мне пиццу», то помощник выведет список пиццерий. Если же сказать «не ищи мне пиццу в Мадриде», например, система все равно будет искать[3]. Такие системы работают, идентифицируя некоторые ключевые слова и используя определенный набор правил. Результат может быть точным в заданной системе правил, но неправильным.

Для того, чтобы научить систему анализировать сложные смысловые конструкции, с учетом эмоций и прочих факторов, специалисты использовали глубокую обработку естественного языка. А именно — вопросно-ответную систему контентной аналитики (Deep Question*Answering, DeepQA)[2]. Если требуется большая точность, то приходится использовать дополнительные методы обработки естественного языка.

IBM Watson — система глубокой обработки естественного языка. При анализе определенного вопроса, для того, чтобы дать правильный ответ, система старается оценить, как можно более обширный контекст. При этом используется не только информация вопроса, но и данные базы знаний. Создание системы, способной провести глубокую обработку естественного языка, позволило решить и другую проблему — анализ огромного количества информации, которая генерируется ежедневно. Это неструктурированная информация, вроде твитов, сообщений социальных сетей, отчеты, статьи и прочее[2]. IBM Watson научился использовать все это для решения задач, поставленных человеком.

Watson — это уже иной уровень вычислительных возможностей. Система умеет разделять определенные высказывания на естественном языке и находить связи между этими высказываниями. При этом Watson справляется с задачей, во многих случаях, даже лучше человека, при этом обработка данных идет гораздо быстрее, работа ведется с гораздо большими объемами — человек на такое просто неспособен[3].

Система работает в таком порядке:

1. Получив вопрос, Watson выполняет его синтаксический анализ, чтобы выделить основные особенности вопроса.
2. Система генерирует ряд гипотез, просматривая корпус в поисках фраз, которые с некоторой долей вероятности могут содержать необходимый ответ. Для того чтобы вести эффективный поиск в потоках неструктурированной информации, нужны совершенно другие вычислительные возможности.
3. Система выполняет глубокое сравнение языка вопроса и языка каждого из возможных вариантов ответа, применяя различные алгоритмы логического вывода. Это трудный этап. Существуют сотни алгоритмов логического вывода, и все они выполняют разные сравнения. Например, одни выполняют поиск совпадающих терминов и синонимов, вторые рассматривают временные и пространственные особенности, тогда как третьи анализируют подходящие источники контекстуальной информации.
4. Каждый алгоритм логического вывода выставляет одну или несколько оценок, показывающих, в какой степени возможный ответ следует из вопроса, в той области, которая рассматривается данным алгоритмом.
5. Каждой полученной оценке затем присваивается весовой коэффициент по статистической модели, которая фиксирует, насколько успешно справился алгоритм с выявлением логических связей между двумя аналогич-

ными фразами из этой области в “период обучения” Watson. Эта статистическая модель может быть использована впоследствии для определения общего уровня уверенности системы Watson в том, что возможный вариант ответа следует из вопроса.

6. Watson повторяет процесс для каждого возможного варианта ответа до тех пор, пока не найдет ответы, которые будут иметь больше шансов оказаться правильными, чем остальные.

Как уже говорилось выше, для правильного ответа на вопрос системе необходимо обращаться к дополнительным источникам данных. Это могут быть учебники, мануалы, FAQ, новости и все прочее. Watson за считанные секунды обрабатывает огромные массивы информации для получения правильного ответа. При этом найденное содержимое тоже проверяется, отсеиваются устаревшие и бесполезные данные.

1.4 Основные понятия сервиса Personality Insights

Служба Personality Insights – ключевой элемент платформы IBM Watson – помогает компаниям увидеть своих клиентов в совершенно новом свете. Она может обрабатывать потенциально зашумленные данные из социальных сетей и автоматически извлекать портреты людей, отражающие их индивидуальность[2]. Эта служба предоставляет набор основных аналитических инструментов для сбора полезной информации о людях и организациях. Полученные сведения можно использовать для организации высоко персонализированного взаимодействия, чтобы точнее нацеливать свои продукты, услуги, маркетинговые кампании и другие мероприятия.

Аналитические инструменты Personality Insights разработаны на основе психологии языка в сочетании с алгоритмами анализа данных. Служба Personality Insights извлекает из данных, указанных человеком в социальных сетях, или из текстовых цифровых сообщений три типа личностных характеристик:

- Большая пятерка личностных характеристик – наиболее широко используемая модель общего описания взаимодействия человека с внешним миром. Эта модель включает в себя пять основных характеристик, или измерений:
    - Уживчивость – способность сострадания и сотрудничества с другими;
    - Добросовестность – способность действовать организованно или спонтанно;

- Экстравертность – способность получать удовольствие от общения;
- Эмоциональный спектр, или невротическое состояние, – степень, в которой эмоции человека зависят от окружения;
- Открытость – насколько человек открыт для той или иной деятельности.

Каждое из этих общих измерений имеет шесть граней, которые характеризуют человека по данному измерению.

- Потребности – какие аспекты способны заинтересовать человека. Эта модель включает двенадцать характеристики, основанных на иерархии потребностей Маслова и работе Форда по маркетингу и моделированию потребностей потребителей:

  Волнение, гармония, любопытство, идеал, замкнутость, самовыражение, свободолюбие, любовь, практичность, стабильность, вызов и уклад.
- Ценности – это мотивирующие факторы, влияющие на то, как человек принимает решения. Модель включает в себя пять аспектов человеческих ценностей на основе работы Шварца по психологии:
  - самостоятельность / помощь другим,
  - консервативность / традиции,
  - гедонизм / получение удовольствия от жизни,
  - самосовершенствование / стремление к успеху,
  - открытость к изменениям / активность.

Служба Personality Insights включает в себя модели для определения личностных характеристик, потребностей и ценностей по текстовой информации. Она индексирует лексемы и сопоставляет маркеры со словарем анализа текстов Linguistic Inquiry and Word Count (LIWC) и специальным словарем, выставляя оценки по каждой словарной категории. Для получения характеристик и ценностей личности служба использует подход взвешенных комбинаций, получая оценки по категориям LIWC, где эти веса отражают коэффициенты соответствия между оценкой для категории и характеристиками.

Служба Personality Insights предоставляет интерфейс прикладных программ (API), который позволяет приложениям составлять портреты по данным из социальных сетей, сведениям, имеющимся на предприятии, и другой цифровой информации. Она использует лингвистический анализ для извлечения из текста когнитивных и социальных характеристик, включая «большую пятерку», потребности и ценности. Такие портреты помогают предприятиям понять предпочтения клиентов и повысить степень их удовлетворенности, предугадывая желания и рекомендуя те или иные действия.

1.5 JavaScript

JavaScript — прототипно-ориентированный сценарный язык программирования. Является диалектом языка ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

– функции как объекты первого класса;

– автоматическое приведение типов;

– автоматическая сборка мусора;

– анонимные функции.

В языке отсутствуют такие полезные вещи, как:

– модульная система: JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;

– стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, базовых типов для бинарных данных;

– стандартные интерфейсы к веб-серверам и базам данных;

– система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

2 ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для того чтобы сформулировать требования к разрабатываемому программному средству, необходимо рассмотреть ряд вопросов, относящихся к предметной области. В частности, следует проанализировать специфику выбранного класса задач, изучить существующие методы их решения, дать оценку существующим решениям.

2.1 Анализ предметной области

Каждому человеку рано или поздно приходиться выбирать или менять профессию. В молодости с выбором легко ошибиться, а сделанную ошибку бывает непросто исправить. Нелюбимая работа в конечном итоге перестаёт удовлетворять не только морально, но и материально. Как же выбрать для себя дело, которое будет и по душе, и по плечу?

В Интернете можно найти десятки специализированных тестов на выбор профессии. Профессиональные тесты предлагаются соискателям и некоторыми кадровыми агентствами. Рассматривая существующие системы, можно найти лишь несколько программных средств, пригодных для использования, но и они имеют ряд существенных недостатков.

В основном, эти системы основаны на успеваемости абитуриента (количество баллов на централизованном тестировании), желаемой квалификации или психологической составляющей личности. Недостатком первого подхода является то, что результат только показывает учреждения образования и специальности которые готовы принять документы. По итогу трудно сказать – будет ли интересно учиться и работать тестируемому на данной специальности. Второй подход аналогичен первому, но он может подсказать альтернативы, т.к. не ограничивается тремя предметами, нужными для поступления. Психологические профориентационные тесты дают больше информации о склонностях и интересах абитуриента. Их минус заключается в том, что они учитывают лишь общие наклонности и указывают только сферу деятельности в которой будущий студент может себя реализовать.

Самым главным недостатком для всех вышеперечисленных методик является отсутствие актуальной информации о студентах, которые успешно закончили обучение на конкретной специальности, и их профессиональных и личностных характеристик, которые помогли им в процессе обучения.

2.2 Постановка задачи

В рамках дипломного проекта поставлена цель разработать сервис, который предлагает абитуриентам несколько наиболее подходящих специальностей, основываясь на сравнении его характеристик с характеристиками специальностей. Разрабатываемая система должна удовлетворять следующим требованиям:

- современный web-интерфейс;
- взаимодействие с пользователем посредством диалога;
- получение информации о пользователе из разных источников;
- обучаемость системы;

Система должна обладать понятным и удобным интерфейсом, который позволяет пользователю, впервые воспользовавшись сервисом, без особых усилий выполнить необходимые действия. Сайт должен одинаково хорошо отображаться на всех популярных на данный период мобильных платформах, с различной конфигурацией программного обеспечения, аппаратного обеспечения и размеров экрана. Также необходимым является и автоматическое растягивание элементов интерфейса, чтобы выглядеть гармонично на устройствах с различными размерами экрана и плотностью пикселей.

Для достижения поставленной цели необходимо решить следующие задачи:

- С помощью языка программирования JavaScript и фреймфорка NodeJS, реализовать серверную часть сайта.
- С помощью языка программирования JavaScript и фреймфорка AngularJS, реализовать клиентскую часть сайта.
- Для развертывания и запуска приложения использовать IBM Bluemix.
- Использовать когнитивные сервисы IBM Watson при помощи предоставленного API.

2.3 Анализ существующих аналогов

Наиболее известные аналоги программного средства: kudapostupat.by, abiturient.by, edunews.ru.

2.3.1 Программное средство kudapostupat.by

Kudapostupat.by представляет собой онлайн-сервис.

Одной из возможностей сайта является возможность сравнить свои баллы по централизованному тестированию с проходными баллами прошлого года. Также на сайте есть возможность поиска учебного заведения по желаемой специализации.

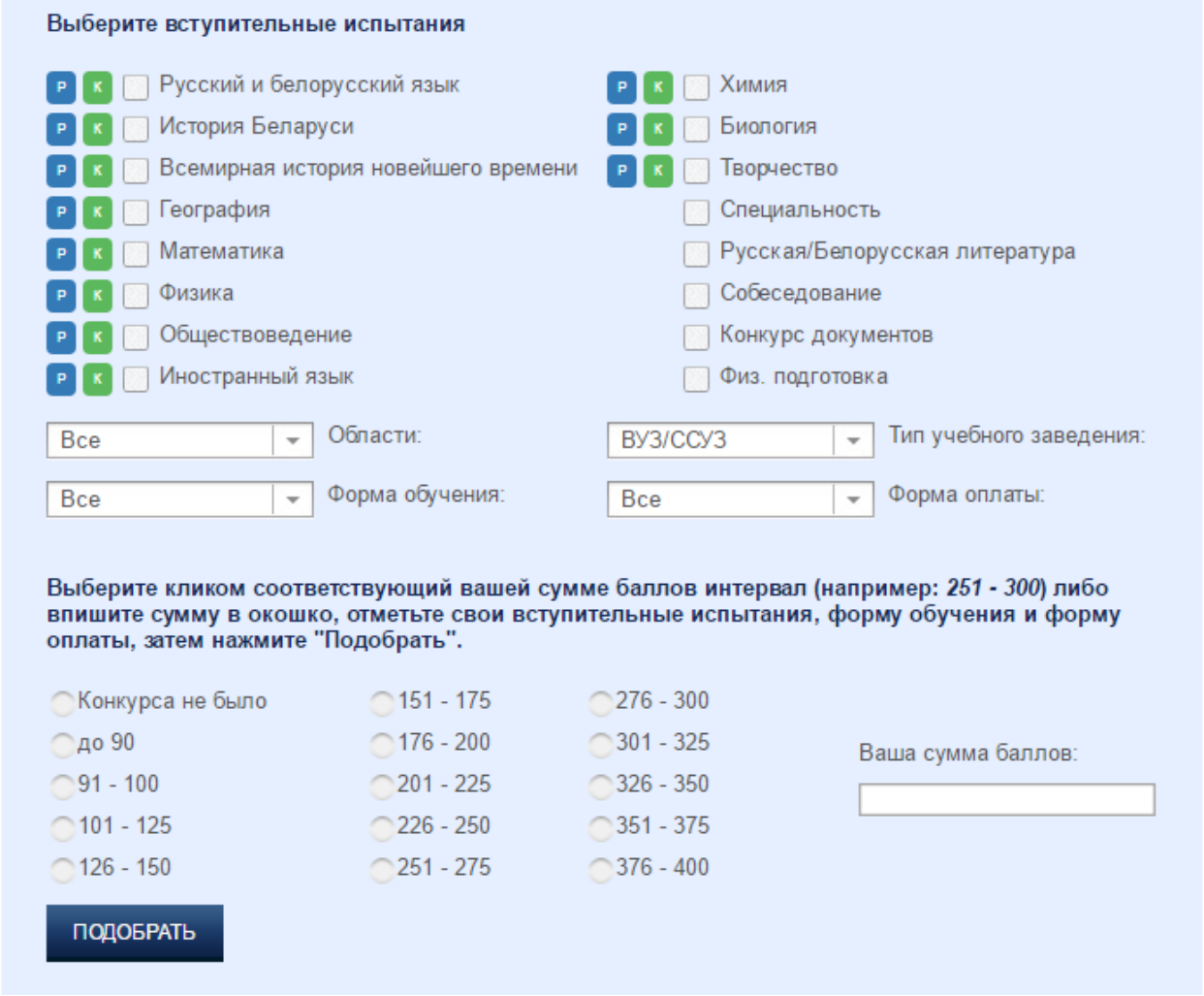

Рисунок 2.1 – Сравнение баллов тестирования

Достоинствами программного средства kudapostupat.by являются:

- информация о проходных баллах за прошлый год;
- возможность выбрать интересующие направления обучения;
- просмотр доступных факультетов и специальностей выбранного ВУЗа/ССУЗа.

Недостатками программного средства являются:

- нельзя сказать будет ли интересно учиться на выбранной специальности;
- нет информации о баллах текущего года;

2.3.2 Программное средство abiturient.by

Данное программное средство предоставляет три психологических теста:

- Опросник профессиональной готовности;
- Опросник профессиональных склонностей;
- Тест по определению типа профессиональной направленности

Главной особенностью данного ПС является то, что здесь упор сделан на психологическую составляющую при выборе будущей профессии.

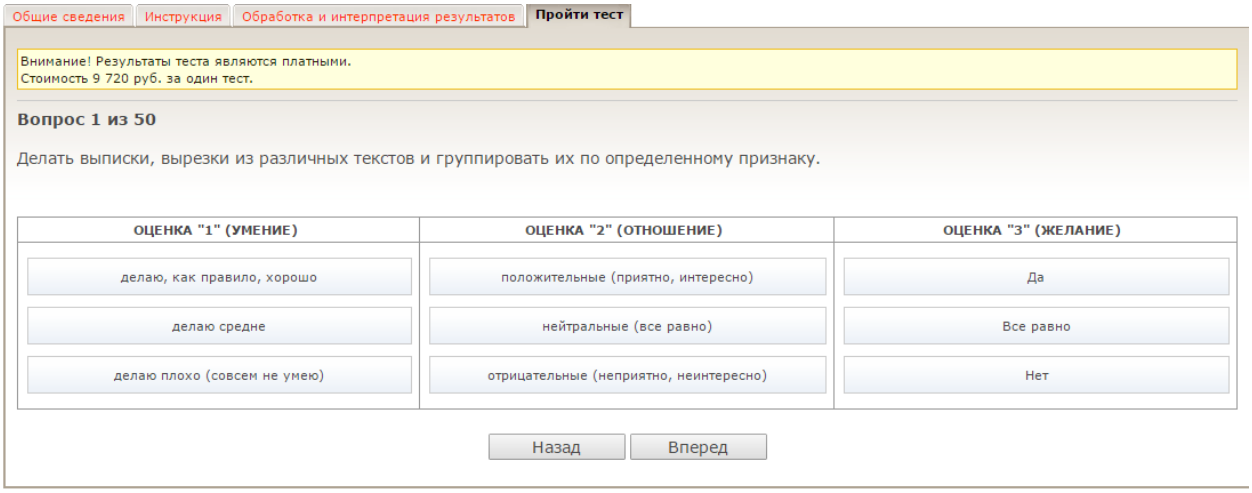

Рисунок 2.2 – Пример вопроса теста

Достоинствами программного средства abiturient.by являются:

- позволяет определить комфортную сферу для обучения;
- несколько тестов, которые позволяют подойти к вопросу с разных сторон;

Недостатками программного средства являются:

- результаты тестов платные;
- результаты показывают только направление обучения, но не конкретные специальности;

2.3.3 Программное средство edunews.ru.

Программное средство edunews.ru является сервисом-помощником.

На сайте предоставлена информация о профессиях, которые разделены на группы.

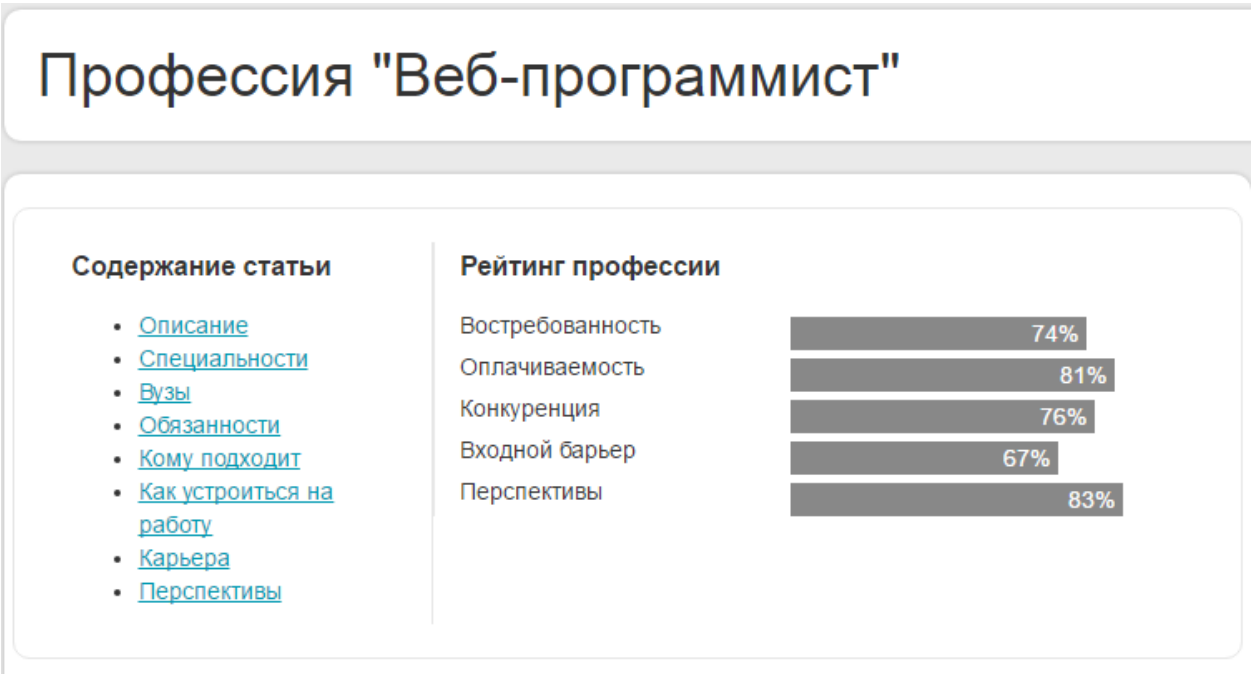

Рисунок 2.3 – Описание профессии «Веб-программист»

Достоинствами программного средства являются:
- бесплатный доступ ко всем материалам сайта;
- перспективы и ВУЗы, в которых можно получить специальность;
- характеристики профессий.

Недостатки программного средства:
- отсутствует возможность узнать проходные баллы;
- не позволяет определить комфортную для обучения сферу;
- есть не все профессии;
- недружелюбный интерфейс;

2.4 Обзор технологий, используемых при реализации программного комплекса

Для реализации SPA была выбрана связка AngularJS и RequireJS.

AngularJS — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC шаблона, а также упрощение тестирования и разработки. Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются

вручную или извлекаются из статических или динамических JSON-данных. Фреймворк адаптирует и расширяет традиционный HTML, чтобы обеспечить двустороннюю привязку данных для динамического контента, что позволяет автоматически синхронизировать модель и представление. В результате AngularJS уменьшает роль DOM-манипуляций и улучшает тестируемость[5].

RequireJS — является реализацией AMD (Asynchronous Module Definition), API для объявления модулей и их асинхронной загрузки «на лету», когда они понадобятся. Это разработка Джеймса Бёрка (James Burke) и она достигла версии 1.0 после двух лет разработки. RequireJS помогает организовать код с помощью модулей и будет управлять асинхронной и параллельной загрузкой ваших файлов. Так как скрипты загружаются только при необходимости и параллельно, это сильно уменьшает время загрузки страницы.

Для реализации серверной части приложения была выбрана связка NodeJS и Express.

Node.js – платформа, основанная на JavaScript движке V8, и предлагающая асинхронное API для работы с сетью и диском. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера[6].

Express - это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Имея в своем распоряжении множество служебных методов HTTP и промежуточных обработчиков, API создается быстро и легко.

3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Проектирование является важным этапом разработки программного обеспечения. Любой проект, связанный с созданием программного продукта, требует предварительного проектирования, построения структуры и планирования сроков разработки. Лишь после предварительного утверждения плана разработки и выбора используемых технологий и структуры программного продукта начинается его реализация.

3.1 Анализ требований

В разрабатываемом программном продукте будет одна роль: пользователь. Пользователь – это человек, который зашел на сайт.

Ниже представлены основные функции пользователя:

– ведение диалога с системой;

– предоставление информации о Twitter аккаунте для сбора данных;

– загрузка текстового файла, который может помочь со сбором информации;

– просмотр результатов по окончании сбора информации;

Также надо отметить, что сервис обладает функцией обучения и сбора данных. Для получения характеристик специальностей было решено использовать характеристики выпускников, которые обучались на данных специальностях. Выпускникам, которых было решено привлечь для сбора статистики, высылается специальная ссылка. При переходе по ней, пользователь также вступает в диалог с системой, аналогичный основному режиму. Различие заключается в том, что результаты, собранные сервисом, сохраняются в таблицу и используются для уточнения характеристики специальности.

В основе реализации графического интерфейса и элементов управления должны лежать современные методики и подходы, использующиеся в разработке интернет-приложений. Интерфейс должен быть простым и понятным. Пользователь должен интуитивно понимать, какая кнопка какую функцию выполняет и куда его приведет то или иное действие.

Схема алгоритма ведения диалога для сбора сведений изображена на рисунке 3.1:

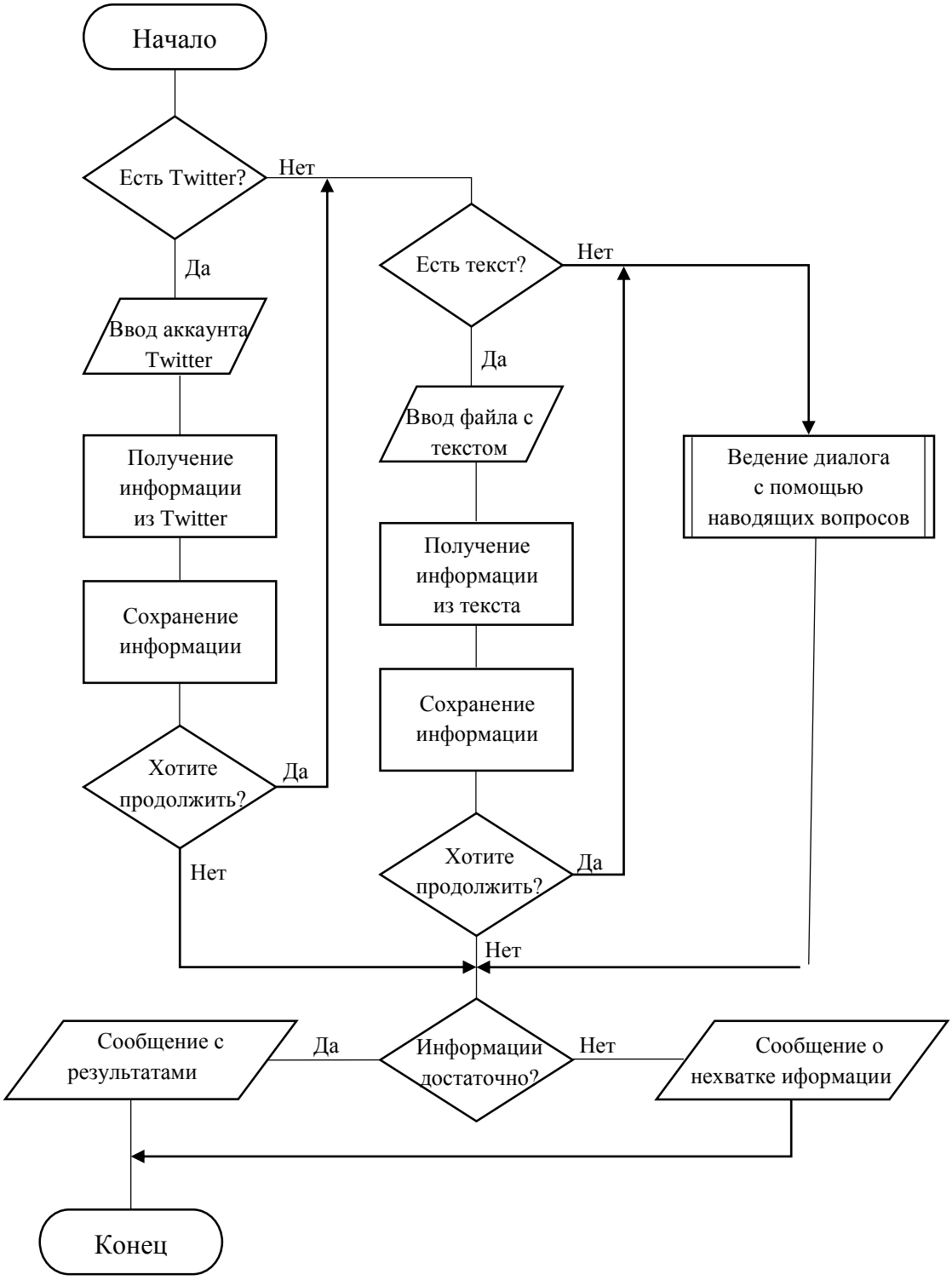

Рисунок 3.1 – Схема алгоритма ведения диалога с пользователем

Разработка дипломного проекта предполагает реализацию всей выше описанной функциональности.

3.2 Разработка архитектуры программного комплекса

Разрабатываемое приложение построено с использованием трехуровневой архитектуры. Все уровни приложения взаимодействуют друг с другом посредством четко определенных интерфейсов. Это помогает создать гибкую систему, позволяющую менять реализацию на уровнях без изменений других уровней. Более низкий слой при данном проектировании не может напрямую обращаться к более высокому слою. Концептуальная схема архитектуры системы отображена на рисунке 3.2:

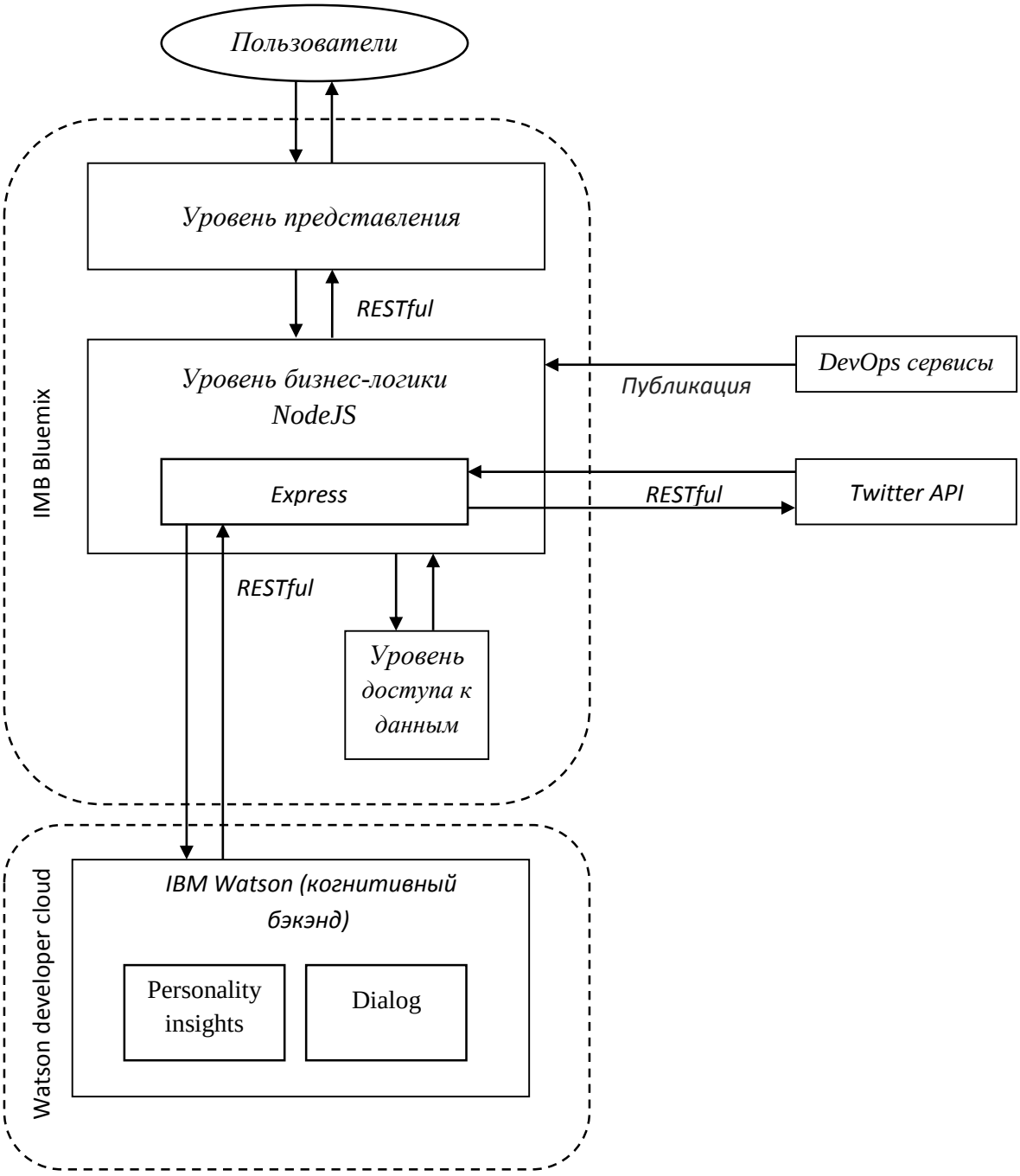

Рисунок 3.2 – Концептуальная схема архитектуры приложения

Уровень представления. Взаимодействие пользователя с системой происходит непосредственно через данный уровень. Уровень представления содержит пользовательский интерфейс, состоящий из графических элементов, элементов управления и форм, отражающих и изменяющих состояние бизнес-модели приложения посредством взаимодействия с уровнем бизнес-логики.

Уровень бизнес-логики. На данном уровне реализуются бизнес-сущности системы, а также методы и классы для работы с ними. Бизнес-сущности описывают состояния объектов. Для изменения состояния бизнес-модели реализуется набор методов и классов, которые оперируют бизнес-сущностями.

Уровень доступа к данным. Задача данного уровня – инкапсуляция работы с базой данных. На данном уровне описываются сущности БД, которые непосредственно проецируются на ее структуру. Для обеспечения сохранения и получения состояния бизнес-модели реализуется набор классов, через которые производится взаимодействие с БД.

В нашем приложении каждому из этих трех слоев соответствует собственный проект:

- Client (слой представления);
- Server (слой бизнес-логики);
    - API (модуль организует работу и запуск сервера);
    - BLL (модуль организует обработку данных).
    - Common (логирование, helper-классы, helper-объекты);
- DAL (доступ к данным)

В описанной модели трехуровневой архитектуры запрещены обращения от низ лежащих уровней к верхним. Обращения с верхних уровней допустимо только к соседним: так, например, запрещены всяческие обращения к уровню доступа к данным с уровня представления.

3.2.1 Слой доступа к данным

На данном этапе, сервис не подразумевает хранения разнообразных данных со сложными связями. Единственная информация, которая хранится – это данные о специальностях и результаты выпускников-респондентов. Для организации хранения этой информации было решено применить нереляционную СУБД MongoDB. В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью[7].

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию, что подходит к данному проекту, так как психологические характеристики, возвращаемые сервисом Personality Insights, имеют несколько вложенностей. Документ можно представить как хранилище ключей и значений.

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

3.2.2 Слой бизнес-логики

Этот слой в проекте представлен в виде веб-сервера, разработанного в среде NodeJS. Сам сервер будет организован с помощью фреймворка Express. Для начала работы сервиса, в пользовательский браузер должна быть загружена начальная страница. Она запустит механизмы по запуску приложения и загрузки остальных файлов и скриптов. Когда клиентская сторона начнёт загружать файлы, на сервер придет большое количество запросов. Express позволяет не обрабатывать каждый запрос на файл вручную. Для этого достаточно определить статическую папку, где будет происходить поиск. Поиск проходит в том числе и во вложенных директориях. После того, как файл будет найден, он будет возвращен в теле ответа.

Единственная задача основного скрипта, который запускает серверное приложение, вызвать метод start у сервера. Скрипт, описывающий сам сервер – это другой модуль. Его задача сводится к тому, что бы обозначить пути и методы контроллера, к которым может обратиться клиентское приложение.

Сама бизнес-логика сервиса содержится в отдельном модуле, функции которого вызываются сервером. Слой работы с данными, также отдельный модуль, вызывать который может только бизнес-логика.

Также на этом уровне организована работа с когнитивными сервисами IBM Watson и API Twitter.

Для ведения диалога с пользователем используется сервис Dialog. Методика работы с данным сервисом описана на рисунке 3.3:

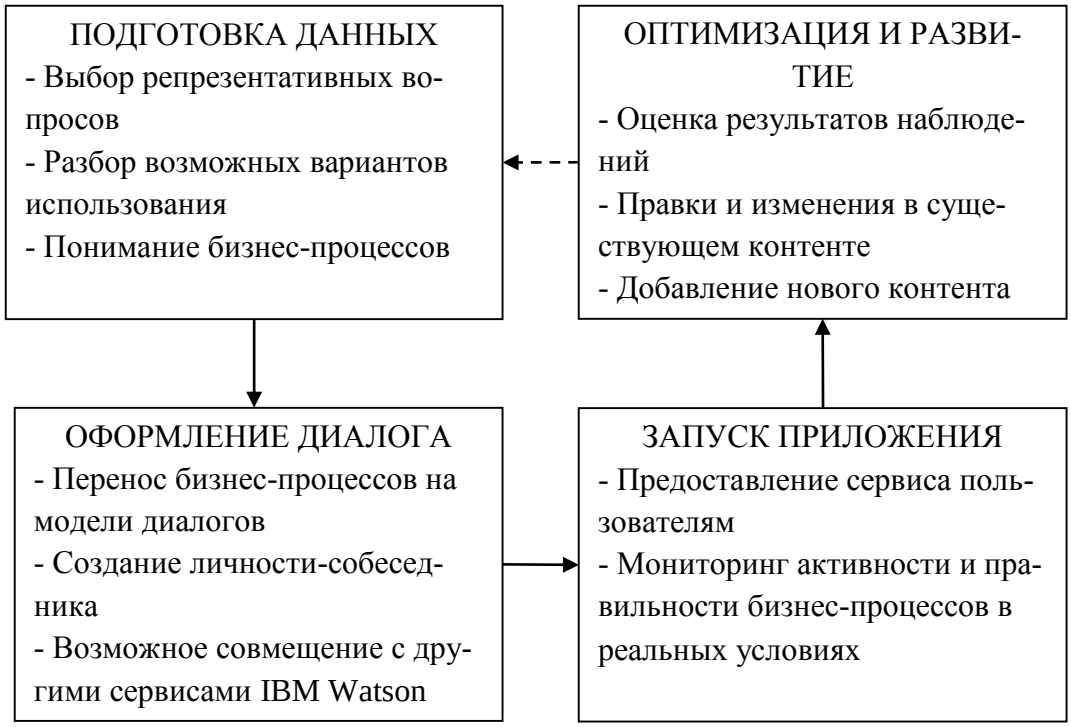

Рисунок 3.3 – Методика работы с сервисом Dialog

За основу бизнес-процесса взят алгоритм ведения диалога, схема которого изображена на рисунке 3.1. За основные узловые точки диалога взяты блоки, предлагающие условия, то есть места, где выбор развития работы алгоритма предполагает непосредственное участие пользователя. В конечном счёте таких точек будет больше, так как на схеме изображена только часть работы общего алгоритма, не содержащего в себе подсистемы обработки ошибочных запросов и вспомогательных реплик пользователя. К вспомогательным репликам можно отнести запрос к системе об условиях пользования и помощи.

Для получения черт характера пользователя используется сервис Personality Insights. Описание работы сервиса, как и возвращаемая им модель данных, приведены в главе 1. Формат данных, возвращаемый сервисом, представлен сложной структурой с множеством дочерних элементов. Для упрощения,

данные будут преобразовываться в ассоциативный массив. Это позволит облегчить модель для хранения в базе данных и упростить работу с ней. В конечном счёте модель будет представлять вектор описывающий как пользователя, так и специальности.

N-мерным вектором называется последовательность n чисел. Эти числа называются координатами вектора. Число координат вектора называется размерностью вектора. В евклидовом n-мерном пространстве представляет из себя точку. Евклидово расстояние является геометрическим расстоянием в многомерном пространстве. Евклидово расстояние между точками x и y в n-мерном пространстве вычисляется по формуле 3.1.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (3.1)$$

где x_i, y_i – i-ые координаты соответствующих векторов;

n – размерность евклидова пространства.

В результате нахождения ближайших точек, описывающих специальности, по отношению к точке, описывающей абитуриента, делается выборка подходящих специальностей.

3.2.3 Слой представления

Уровень представления взаимодействует с пользователем, посредством отображение графического интерфейса и обработку пользовательских действий в приложении. Уровень представления взаимодействует с уровнем бизнес-логики посредством REST-запросов, отправляя на обработку набор данных, полученных пользователем и отображая результаты обработки этих данных[4].

Структуру клиентского приложения обуславливает использование AngularJS. Основной формой организации приложения в AngularJS являются модули. Модуль представляет хранилище различной информации: директив, фильтров, контроллеров и т.д. При этом одно приложение может иметь несколько модулей.

В данном проекте модули будут использоваться для ассоциирования определенного участка HTML-страницы с приложением AngularJS. Модули позволят организовать и структурировать различные компоненты по назначе-

нию. Благодаря возможностям RequireJS будет организована загрузка скриптов и сторонних библиотек, а для установки их актуальных версий будет использоваться пакетный менеджер Bower.

Для удобного добавления, каждый модуль будет описан с помощью трех файлов. Первый файл будет возвращать название модуля, второй - создавать сам модуль с этим названием, третий файл будет загружать компоненты модуля. Благодаря такой организации, для того, чтобы подключить модуль к проекту, надо указать в основном модуле пути к второму и третьему файлу.

Условная структура клиентской части имеет вид:

- Client – основная папка клиентского приложения.
  - Libs – папка со сторонними библиотеками.
  - Bower.json – файл с описанием сторонних библиотек.
  - Index.html – стартовый файл.
  - Scripts – папка со скриптами.
    - Module – папка с одним из модулей.
      - Directives – папка с директивами модуля.
      - Controllers – папка с контроллерами модуля.
      - Services – папка с сервисами модуля.
      - Templates – папка с шаблонами и разметкой.
      - Module.js – файл, создающий модуль.
      - Namespace.js – файл, задающий имя модуля
      - Require.js – файл, загружающий компоненты модуля
    - Main.js – файл, описывающий зависимости для RequireJS и запускающий AngularJS приложение.
    - App.js – корневой модуль приложения.
    - Routes.js – файл, описывающий пути приложения и их обработчики.

4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

4.1 Реализация уровня доступа к данным

Для работы с MongoDB был выбран фреймворк MongooseJS. Он позволяет просто и лаконично организовать работу с базой данных.

Сперва нужно включить библиотеку в модуль и настроить:

```
var mongoose = require('mongoose');
var common = require('./common');
mongoose.connect('mongodb://localhost/abiturhelper');
var db = mongoose.connection;
```

Потом описываются схемы для данных. Примером может служить описание схемы для данных о специальности:

```
var  feature = new Schema({
        name: { type: string, required: true },
        value: { type: number, required: true },
        parent: { type: string, required: true }
});
var specialities = new Schema({
        name: { type: string, required: true },
        features: [feature]
});

var ArticleModel = mongoose.model('specialities', specialities);
```

4.2 Реализация уровня бизнес-логики

Основной уровень приложения, представляющий бизнес-логику, достаточно объемен и его рассмотрение будет представлено поверхностно, описывая основные подходы к реализации того или иного функционала, используемого в приложении.

За запуск сервера отвечает модуль index:

```
var server = require("./API");
server.start();
```

Создание такого небольшого модуля обуславливается возможным расширением системы в будущем, например, запуск нескольких API для разных групп пользователей или предварительной настройке условий для существующего сервера.

Описание сервера содержится в модуле API. Модуль подключает сторонние библиотеки:

```
var express = require("express");
var Q = require('q');
var BLL = require('./BLL');
```

Библиотека Q, разработанная Крисом Ковалем, позволяет организовать работу с промисами (англ. Promise - обещание), что облегчает работу с асинхронными функциями, например получением данных с сервера. Далее идет настройка сервера:

```
var app = express();
app.use(express.static(__dirname + '../Client'));
app.use('/Client/libs', express.static(__dirname + '../Client/libs'));
app.use('/Client/Scripts', express.static(__dirname + '../Client/Scripts'));
app.use(bodyParser.json());
```

Следом прописываются обработчики и методы API. В качестве примера возьмем GET запрос по пути /home:

```
app.get('/home', function(request, response){
	var model = { title : { main: "Abiturhelper", subtitle: "subtitle" }, layout: false };
	response.render('index.html', model);
});
```

После описания всех методов, определяется или задается порт и создается сервер:

```
var port = process.env.VCAP_APP_PORT || 3000;
app.listen(port, function() {
	console.log('Server has been start.');
});
```

Потом вся настройка сервера оборачивается в функцию start и экспортируется для того, чтобы этим модулем могли воспользоваться другие компоненты программы:

```
exports.start = start;
```

Описание бизнес-логики содержится в модуле BLL. Рассмотрим подробнее работу с сервисом Personality Insights. Для работы с этим сервисом импортируем библиотеку watson-developer-cloud:

```
var watson = require('watson-developer-cloud');
```

Для доступа к сервису необходимо установить настройки доступа, предоставленные IBM Bluemix:

```
var personalityInsights = watson.personality_insights({
        username: '0d049ae6-affa-4f05-8b96-170001be0c39',
        password: 'KamNU6jAjxEH',
        url: "https://gateway.watsonplatform.net/personality-insights/api",
        version: 'v2'
});
```

Функция для получения характеристик пользователя выглядит следующим образом:

```
function getPIfromText(text){
        var deferred = Q.defer();
        personalityInsights.profile({
                text: request.body.text,
                language: 'en' },
                function (err, resp) {
                        if (err)
                        common.log('error:', err);
                        deferred.reject(err);
                        else {
                                deferred.resolve(resp);
                        };
        });
        return deferred.promise;
};
```

Благодаря библиотеке Q, в модуле сервера можно повесить колбэк-функцию, которая сработает только тогда, когда сервис PI вернет результат.

Для организации работы с сервисом Dialog, необходимо в XML файле задать основные ноды диалога. Ноды – это узловые точки, позволяющие выбрать флоу (англ. Flow - поток) диалога. Разговор можно представить в виде графа, в котором ноды будут вершинами. Рассмотрим входную ноду приложения:

```
<output id="output_300001">
        <prompt>
            <item>Hello, I'm [BotName]. I will help you with you choise of university. But to do it, I need to read your twitter account. So, please, give me your twitter account in form "@twitter"</item>
        </prompt>
        <goto ref="getUserInput_200001"/>
</output>
```

Первая нода, как правило, должна возвращать текст, подсказывающий пользователю контекст диалога и дальнейшие действия. Тэг goto перенаправляет флоу на другую ноду в дереве:

```
<getUserInput id="getUserInput_200001">
    <search ref="inputSequence_100004"/>
    <search ref="inputSequence_100003"/>
    <search ref="inputSequence_100001"/>
    <search ref="inputSequence_100002"/>
    <search ref="inputSequence_100005"/>
    <default>
        <output>
        <prompt selectionType="RANDOM">
            <item>Sorry, I didn’t understand you. May be, you'll try something else?</item>
            <item>Please, excuse me, but I don’t understand.</item>
        </prompt>
        </output>
    </default>
</getUserInput>
```

Тег <getUserInput> указывает на то, что нужно обработать пользовательский ввод, описанный множеством тегов <search>. Сам тег работает по схожему принципу с <goto>, с той разницей, что он указывает на обработчик с

соответствующим id, куда следует отправить пользовательский текст. В теге <prompt> описываются варианты ответа. Атрибут selectionType="RANDOM" указывает, что из множества вариантов ответной реплики, необходимо выбрать один случайным образом. В данном случае этот тег является дочерним для тега <default>, который подразумевает выбор реплики в случае, не описанном ни в одном из обработчиков. Рассмотрим один из обработчиков подробнее:

```
<folder id="inputSequence_100005" comment="Others">
	<input>
		<grammar>
			<item>Hi</item>
			<item>Hello</item>
			<item>yo</item>
		</grammar>
	<output>
		<prompt>
			<item>Hello. Nice to meet you! As you may know - time is a money. So, let's spend your time with profit and find best matching university for you. Give me your twitter account in form "@twitter".</item>
		</prompt>
	</output>
	</input>
	<input>
		<grammar>
			<item>$ give * recomendation</item>
			<item>$ repeat * suggestion</item>
		</grammar>
		<if matchType="ALL">
			<cond varName="UniversityName" operator="HAS_VALUE"/>
			<output>
				<prompt>
					<item>I suggest you to enroll {UniversityName}. Visit their website: {UniversityWebsite} - to know more.</item>
				</prompt>
		</output>
		</if>
		<if matchType="ALL">
			<cond varName="UniversityName" operator="IS_BLANK"/>
			<output>
				<prompt>
```

```
                                        <item>I haven't suggested you any university to enroll. Please, specify your twitter account to help me do it.</item>
                                    </prompt>
                                </output>
                        </if>
                </input>
</folder>
```

Данный обработчик представляет из себя набор вариантов развития диалога. Каждый отдельный элемент заключён в тег <input> и представляет из себя, объединённые логически, варианты ответа пользователя. Тэг <grammar> позволяет описать возможные реплики пользователя. Всего есть несколько типов описания:

1. Фиксированная реплика: <item> text </item>. Описывает точное совпадение пользовательского ввода с текстом заключенным в теги.
   Сработает в случаях:
   a. text
2. Ключевые слова реплики: <item> text1 * text2 * text3 </item>. Описывает ключевые слова, которые должны содержаться в сообщении пользователя. Символ * заменяется на любое количество слов.
   Сработает в случаях:
   a. text1 text2 text3
   b. text1 other words text2 text3
   c. text1 other words text2 other words text3
3. Подстрока в реплике: <item> $ text </item>. Описывает подстроку, которая должна содержаться в реплике.
   Сработает в случаях:
   a. other words, text
   b. other words, text other words
   c. text

Описания можно комбинировать, для получения более точного результата.

Неотъемлемой частью описания диалога становятся переменные. Они описываются в секции, заключенной в тег <variables>:

```
<variables>
        <var_folder name="UniversityData">
                <var name="UniversityName" type="TEXT" description="Name of the univesity"/>
        </var_folder>
```

```
<var_folder name="UserData">
        <var name="TwitterAccount" type="TEXT" description="Twitter Account"/>
        <var name="UserName" type="TEXT" description="Name of user"/>
</var_folder>
<var_folder name="ErrorMessages">
        <var name="TwitterErrorMessage" type="TEXT" description="Error message when twitter account hasn't been found"/>
        <var name="PIErrorMessage" type="TEXT" description="Error message when there is not enough words for PI"/>
</var_folder>
</variables>
```

Основная работа с ними идет в секциях <action>. Секция является дочерней для секции <input>. Можно рассмотреть работу на примере присвоения значения переменной, в которой хранится значение Twitter аккаунта:

```
<action varName="TwitterAccount" operator="SET_TO_USER_INPUT"/>
```

Для того, чтобы вставить значение переменной в реплику, необходимо имя переменной в фигурных скобках.

```
<item>Ok, I've got it, {TwitterAccount}! Now, wait a second and let me think about the most appropriate place.</item>
```

В конечном счёте у нас получается модель диалога, файл с которой, нужно отправить в сервис при первом инициализирующем обращении к нему. В ответе, сервис вернёт идентификационный номер для пользователя, для соглашения и реплику из первой ноды. Номер соглашения служит для идентификации конкретного диалога, так как в сервис можно отправить несколько моделей для разных диалогов. Номер пользователя позволит вести диалог со множеством пользователей одновременно. По этому значению хранится состояние диалога с конкретным человеком. Эти два id необходимо указывать при каждом обращении к сервису.

4.3 Реализация уровня представления

Для работы со сторонними библиотеками необходимо указать зависимости для RequireJS. Например для того, чтобы работать с AngularUiRouter необходимо прописать следующее:

```
'angularUiRouter': '../libs/angular-ui-router/release/angular-ui-router'
shim: {
'angularUiRouter': {
        deps: ['angular', 'jquery']
      },
};
```

В секции shim указываются зависимости. В указанном примере AngularUiRouter будет загружен только после того, как будут загружены модули angular и jquery.

После этих манипуляций, для того чтобы использовать AngularUiRouter в модуль, необходимо включить сборку в основной модуль в начале файла:

```
define([
    'angular',
    './namespace',
    './common/namespace',
    'angularUiRouter',
    'angularLocalStorage',
    'angularUiBootstrap',
    'angularResource',
    'angularLocale',
    'angularAnimate',
    './common/require',
], function (angular, namespace, common) {
   'use strict';
   var name = namespace;
   return angular.module(name, [common, 'ui.router', 'LocalStorageModule', 'ngResource',
'ui.bootstrap', 'ngAnimate' ]);
});
```

За запуск приложения AngularJS и загрузку скриптов отвечает скрипт main.js. В нём прописаны упомянутые зависимости для RequireJS и следующие строки, которые запустят приложение после загрузки в DOM основного документа:

```
angular.element(document).ready(function () {
        angular.bootstrap(document, [namespace]);
});
```

Для этого, нужно в стартовой странице указать этот файл во время загрузки библиотеки:

```
<script src="Client/libs/requirejs/require.js" data-main="Client/Scripts/main.js"></script>
```

Остальная загрузка произойдет автоматически, благодаря описанным зависимостям и выбранной структуре приложения.

Одной из особенностей AngularJS служит расширения языка разметки HTML. AngularJS содержит множество встроенных директив, но их функциональности не всегда достаточно. Иногда необходимо применить более комплексное решение, чем стандартные элементы HTML, использующие встроенные директивы. Директивы также удобны тем, что позволяют использовать их повторно неограниченное количество раз. Рассмотрим директиву, описывающую элемент управления kcMessenger:

```
define([
   '../module'
   , '../namespace'
],
function (module, namespace) {
        var name = 'kcMessenger';
        var dependencies = [namespace + '.messengerService'];
        var directive = {
                templateUrl: 'Scripts/Messenger/templates/_messenger.html',
                restrict: 'EA',
                link: link,
                controller: controller
        };
        var link = function (scope, el, attrs) {…};
        var controller = function ($scope, $timeout) {…};
        module.directive(name, dependencies.concat(directive));
});
```

Директива создается с помощью метода module.directive(). В данном случае модулем является module, объявленный в другом файле и указанный в зависимостях RequireJS. Данный метод принимает два параметра: название директивы и функцию-фабрику, которая, собственно, и создает директиву. В описанном выше случае, вторым параметром является массив объектов, который содержит имена зависимостей и объект, описывающий директиву. Такой

подход называется инъекцией зависимостей (англ. Dependency Injection) и используется намного чаще, чем классический.

В качестве названия используется "kcMessenger". Согласно соглашениям об именовании любая буква в верхнем регистре расценивается как начало нового слова, которое будет отделено от предыдущего дефисом. То есть для AngularJS "kcMessenger" будет представлять "kc-messenger".

С помощью дополнительных свойств мы можем настроить применение директивы. Рассмотрим их по порядку.

Свойство templateUrl позволяет задать шаблон разметки, который будет использован во время рендеринга директивы. В описанном случае указан относительный путь к файлу, в котором описан шаблон. Альтернативой может служить свойство template, которое позволяет задать шаблон в виде строковой переменной, но используется, как правило, для небольших директив, в которых основной функциональностью является не рендеринг сложного элемента управления, а обработка данных. Это свойство, как можно догадаться, является необязательным.

Свойство restrict позволяет указать объект, к которому директива будет применяться. Оно может иметь следующие значения:

E: директива применяется к элементу

A: директива применяется к атрибуту

C: директива применяется к классу

M: директива применяется к комментарию

Данные значения можно комбинировать.

На свойствах link и controller стоит остановиться подробнее. В обоих случаях указывается функция, обрабатывающая данные директивы. Есть ещё одно, более общее свойство compile:

```
compile: function compile(temaplateElement, templateAttrs) {
        return {
                pre: function (scope, element, attrs) { },
                post: function(scope, element, attrs) { }
        }
};
```

Метод Link - это та самая функция, которую возвращает фабрика директивы в классической версии. Здесь надо понимать, что в AngularJS процесс компиляции разбит на два этапа:

- compile - анализ всех директив используемых в данном элементе DOM (в том числе и в его потомках child). В общем случае описывается в секции pre.
- linking - связывание переменных используемых в шаблоне и переменных в scope. В общем случае описывается в секции post.

И при этом, как в простейшей версии, так и в расширенной, метод Link правильно будет называть postLink, поскольку он выполняется после того, как переменные уже сопоставлены. В отличии от них, controller является функцией, обрабатывающей данные во время исполнения и является классическим контроллером AngularJS приложения.

Для взаимодействия с серверной частью используется встроенный сервис $http. Сервис $http представляет ключевой сервис Angular, предназначенный для взаимодействия с удаленным HTTP-сервером через объект XMLHttpRequest или через JSONP. Сервис $http это функция, которая принимает один аргумент — объект с настройками — который используется для генерации HTTP запроса, и возвращает промис с двумя определенными в $http методами: success и error.

Каждый вызов сервиса $http обязательно принимает в параметрах HTTP метод и URL, и POST/PUT запросы к тому же обязательно передают данные, для сокращения их вызова были созданы сокращенные методы:

- $http.get
- $http.head
- $http.post
- $http.put
- $http.delete

Рассмотрим пример работы с сервисом из приложения:

```
var _getResponse = function (userInput) {
        var serviceUri = '/getResponce;
        var config = {
          headers: {
            'Accept': 'application/json'
          }
        };
        return $http.post(serviceUri, userInput, config);
 };
```

Эта функция из сервиса messengerService, которая отправляет реплику пользователя на сервер и получает ответную реплику. Переменная serviceUri

задает конечную точку (англ. Endpoint) отправки данных или URL REST запроса. Переменная config позволяет настроить заголовок запроса, указав, какого рода данные будут переданы. Сами данные в переменной userInput.

Создание отдельного сервиса для отправки запросов позволяет инкапсулировать код и отделить отправку запроса и обработку ответа. Для примера можно привести код работы с данным сервисом из контроллера описанной выше директивы:

```
var userInput = { userInput: $scope.userInput};
messengerService.getResponse(userInput).then(function (result) {
        vm._addMessage(result.wdsResponseMsg);
}, function (error) {
        var errorData = { error: error,  'messenger'};
        vm.logError(errorData);
});
```

Так как сервис $http возвращает промис, то для обработки используются две callback-функции для успешного и ошибочного выполнения запроса.

5 ТЕСТИРОВАНИЕ И ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ

5.1 Тестирование модулей

Проведено тестирование программного средства. Целью данного испытания было ознакомление с программным средством и проверка его работоспособности. ПС протестировано в последних версиях наиболее популярных браузеров: Google Chrome, Mozilla Firefox, Opera, Internet Explorer.

В таблице 5.1 представлены тест-кейсы для пользователя.

Таблица 5.1 – Набор тест-кейсов для роли «Гость»

№ тест-кейса	Тестируемая функциональность	Последовательность действий	Ожидаемый результат	Полученный результат
1	2	3	4	5
1	Знакомство с пользователем	1. Запустить приложение. 2. Ввести в форму своё имя. 3. Нажать кнопку «Ready»	1. Отображение всплывающего окна с предложением ввести своё имя. 2. Отображение имени в форме. 3. Открытие диалога с первым обращением по выбранному имени.	Тест пройден успешно
2	Знакомство с пользователем	1. Запустить приложение. 2. Оставить форму пустой. 3. Нажать кнопку «Ready»	1. Отображение всплывающего окна с предложением ввести своё имя. 2. Пустая форма. 3. Сообщение под полем ввода «Please, enter your name or nickname. Don’t be modest. ».	Тест пройден успешно

1	2	3	4	5
3	Знакомство пользователя с системой	1. Ввести в поле ввода сообщения текст «Please, tell me about yourself. » 2. Нажать кнопку «Send».	1. Поле ввода сообщения с текстом. 2. Новое сообщение от пользователя в диалоге. Ответ системы в диалоге с информацией о использовании.	Тест пройден успешно
4	Обработка бессмысленных сообщений	1. Ввести в поле ввода сообщения текст «afein ajs;fasldk'/.,flsjfs dflsdkot[y[tr » 2. Нажать кнопку «Send».	1. Поле ввода сообщения с текстом. 2. Новое сообщение от пользователя в диалоге. Ответ системы в диалоге с одним из вариантов ответа на нераспознанный текст.	Тест пройден успешно
5	Предоставление твиттер аккаунта для сбора информации	1. Дождаться предложения системы предоставить твиттер аккаунт. 2. Ввести твиттер аккаунт. 3. Нажать кнопку «Send».	1. Сообщение системы в диалоге с одним из вариантов текста направленным на получение твиттер акаунта. 2. Поле ввода сообщения с текстом. 3. Новое сообщение от пользователя в диалоге. Ответ системы в диалоге с одним из вариантов ответа с благодарностью и предложением продолжить сбор информации.	Тест пройден успешно

1	2	3	4	5
6	Предоставление файла с текстом для информации	1. Дождаться предложения системы предоставить файл с текстом. 2. Нажать на кнопку «Upload file». 3. Выбрать файл. 4. Нажать кнопку «Send».	1. Сообщение системы в диалоге с одним из вариантов текста направленным на получение твиттер акаунта. 2. Открытие окна с файловой системой. 3. Поле ввода сообщения с именем файла. 4. Новое сообщение от пользователя в диалоге. Ответ системы в диалоге с одним из вариантов ответа с благодарностью и предложением продолжить сбор информации.	Тест пройден успешно
7	Вывод результатов	1. Предоставить необходимое количество текста. 2. Согласиться на вывод результата.	1. Сообщение системы в диалоге с одним из вариантов текста о достаточном количестве информации. 2. Новое сообщение от пользователя в диалоге. Ответ системы в диалоге с перечисленными специальностями.	Тест пройден успешно

5.2 Запуск и публикация приложения

Для корректной работы данного программного средства необходим компьютер следующей минимальной конфигурации:

– ОС: Windows (8/8.1/10);

– Процессор: Pentium® III 800 МГц или AMD Athlon;

– RAM: 1024 Мб;

– HDD: 100 Мб свободного места.

Необходимо установить следующие программы последних стабильных версий:

– NodeJS;

– npm;

– bower;

– mongoDB;

– git;

– Cloud-foundry CLI;

5.2.1 Запуск приложения на локальной машине

Для запуска приложения локально необходимо выполнить следующую последовательность действий в консоли:

1) git clone https://github.com/Chugainov/abiturhelper;

2) cd abiturhelper;

3) npm install;

4) cd Client;

5) bower install;

6) cd ..

7) node index.js

После этого в адресной строке браузера ввести: http://localhost:3000

5.2.2 Публикация приложения в IBM Bluemix

Для публикации приложения в IBM Bluemix необходимо создать аккаунт. В файле manifest.yml изменить значение поля name на любое удобное, это имя будет проассоциировано в адресе <application-name>.mybluemix.net. Если такое имя уже зарезервировано, то система сообщит об этом в виде ошибки. Также необходимо изменить переменные, хранящие параметры доступа к сервисам.

В консоли следует выполнить следующие действия:

1) cf api https://api.ng.bluemix.net
2) cf login -u <your user ID>
3) cf create-service personality_insights tiered personality-insights-service
4) cf create-service dialog tiered dialog-service
5) cf push

Если все действия прошли без ошибок, то приложение будет автоматически запущено.

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ПОДБОРА СПЕЦИАЛЬНОСТИ АБИТУРИЕНТА ВУЗА НА ОСНОВЕ КОГНИТИВНЫХ СЕРВИСОВ IBM BLUEMIX

6.1 Общая характеристика программного средства

Целью дипломного проекта является разработка приложения для подбора специальности абитуриента ВУЗа на основе когнитивных сервисов IBM Bluemix. Данное приложение позволяет подсказать специальность, сравнивая характеристики выпускников и абитуриентов. Новизной проекта является то, что он использует когнитивные сервисы IBM Bluemix, позволяющие получить психологические аспекты личности из текста, написанного на естественном языке.

Технико-экономического обоснование ПС определяет экономическую выгодность создания данного продукта и дальнейшего его применения.

Разработка данного приложения предусматривает следующие стадии проектирования: техническое задание, технический проект, рабочий проект, объединенный в одну стадию – технорабочий проект, внедрение. Программный комплекс относится к 3-й группе сложности. Категория новизны продукта – «В». Дополнительный коэффициент сложности ПО – 0,07.

Экономический эффект зависит от объема затрат на разработку и освоение программного продукта. Экономический эффект у разработчика выступает в виде чистой прибыли, остающейся в распоряжении предприятия после реализации ПС, а у заказчика – в виде экономии трудовых, материальных и финансовых ресурсов.

6.2 Расчет сметы затрат и цены программного продукта

6.2.1 Расчет трудоемкости разработки ПС

Объем ПС определяется путем подбора аналогов на основании классификации типов ПС, каталога функций ПС и каталога аналогов ПС в разрезе функций, которые постоянно обновляются и утверждаются в установленном порядке. На основании информации о функциях разрабатываемого ПС по каталогу функций определяется объем функций.

Исходные данные для расчета представлены в таблице 6.1.

Таблица 6.1 – Исходные данные

Наименование показателя	Обозначение	Единица измерения	Количество
Коэффициент новизны	$К_н$	единиц	0,63
Группа сложности		единиц	3
Дополнительный коэффициент сложности	$К_с$	единиц	0,07
Поправочный коэффициент, учитывающий использование типовых программ	$К_т$	единиц	0,65
Установленная численность исполнителей проекта	$Ч_р$	единиц	3
Продолжительность рабочего дня	$Т_ч$	Ч	8
Тарифная ставка 1-го разряда	$Т_{м1}$	тыс. руб.	298
Коэффициент премирования	$К_п$	единиц	1,5
Норматив дополнительной заработной платы	$Н_д$	%	10
Отчисления в фонд социальной защиты населения	$З_{сз}$	%	34
Отчисления в Белгосстрах	$Н_{нс}$	%	0,60
Норматив налога на добавленную стоимость	НДС	%	20
Норматив расходов на командировки в целом по организации	$Н_{рнк}$	%	10
Норматив накладных расходов в целом по организации	$Н_{нр}$	%	50
Норматив налога на прибыль	$Н_п$	%	18
Эффективный фонд времени	$Ф_э$	дней	235
Рабочих дней в месяце		дней	21
Коэффициент, учитывающий средства разработки	$К_{ур}$	единиц	0,7

Объем ПС определяется на основе нормативных данных, приведенных в таблице 6.2.

Таблица 6.2 – Характер функций и их объем

Номер функции	Наименование (содержание) функции	Объем функции, строчек кода	
		По каталогу	Уточненный
102	Контроль, предварительная обработка и ввод информации	490	490
103	Преобразование операторов входного языка в команды другого языка	740	350
107	Организация ввода-вывода информации в интерактивном режиме	280	280
202	Формирование базы данных	1980	650
203	Обработка наборов и записей базы данных	2370	1100
206	Манипулирование данными	7860	5000
207	Организация поиска и поиска в базе данных	4720	920
405	Система настройки ПС	340	320
506	Обработка ошибочных и сбойных ситуаций	1540	840
507	Обеспечение интерфейса между компонентами	1680	980
601	Проведение тестовых испытаний прикладных программ в интерактивном режиме	3780	1380
602	Вспомогательные и сервисные программы	470	470
701	Математическая статистика и прогнозирование	3780	1500
703	Расчёт показателей	410	410
811	Администрирование и обновление сайта	90	90
Итого:		26615	14535

Общий объем функций программного средства вычисляется по формуле 6.1.

$$V_o = \sum_{i=1}^{n} V_i, \quad (6.1)$$

где V_o – общий объем ПС;

V_i – объем отдельной функции ПС;

n – общее число функций.

Таким образом, общий объем ПС равен $V_o = 14780$ строк исходного кода.

На основании общего объема ПС определяется нормативная трудоемкость ($T_н$) с учетом сложности ПС. Для ПС третьей группы сложности, к которым относится разработанный программный продукт, нормативная трудоемкость составит 565 человеко-дней.

Наличие интерактивного доступа позволяет применять к объему ПС коэффициент $К_с$, который определяется по формуле

$$К_с = 1 + \sum_{i=1}^{n} К_i, \quad (6.2)$$

где K_i – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

n – количество учитываемых характеристик.

Таким образом, коэффициент сложности имеет следующую величину

$$К_с = 1 + 0{,}07 = 1{,}07.$$

С учетом дополнительного коэффициента сложности $К_с$ рассчитывается общая трудоемкость ПС по формуле

$$Т_о = Т_н \cdot К_с \cdot К_т \cdot К_н, \quad (6.3)$$

где $К_с$ – коэффициент, учитывающий сложность ПС;

$К_т$ – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

$К_н$ – коэффициент, учитывающий степень новизны ПС.

С учетом распределения по стадиям общая трудоемкость принимает вид, представленный на формуле

$$T_o = \sum_{i=1}^{n} T_i, \tag{6.4}$$

где T_i – трудоемкость разработки ПС на i-й стадии (чел./дн.);

n – количество стадий разработки.

Трудоемкость стадий рассчитывается по формуле

$$T_{yi} = T_н \cdot К_с \cdot К_т \cdot К_н \cdot К_{ур} \cdot d_{сті}, \tag{6.5}$$

где T_{yi} – уточненная трудоемкость разработки ПС на I-й стадии;

$d_{сті}$ – удельный вес трудоемкости i-й стадии разработки ПС в общей трудоемкости разработки ПС;

$К_с$ – коэффициент, учитывающий сложность ПС, вводится на всех стадиях;

$К_т$ – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей, вводится только на стадии рабочего проекта;

$К_н$ – коэффициент, учитывающий степень новизны ПС, вводится на всех стадиях;

$К_{ур}$ – коэффициент, учитывающий средства разработки ПС, вводится на всех стадиях;

Т.к. разработанный проект имеет категорию новизны «В», то удельные веса по стадиям выглядят следующим образом $d_{тз} = 0{,}08,\ d_{эп} = 0,\ d_{тп} = 0,\ d_{рп} = 0{,}47 * 0{,}85 + 0{,}34 = 0{,}7395,\ d_{вн} = 0{,}11.$

Таким образом, трудоемкость стадии технического задания может быть рассчитана по формуле

$$T_{утз} = T_н \cdot К_с \cdot К_н \cdot К_{ур} \cdot d_{тз}, \tag{6.6}$$

$$T_{утз} = 565 \cdot 1{,}07 \cdot 0{,}63 \cdot 0{,}7 \cdot 0{,}08 = 21{,}33 \text{ чел./дн.}$$

Трудоемкость стадии технорабочего проекта рассчитывается по формуле

$$T_{утрп} = T_н \cdot К_с \cdot К_н \cdot К_{ур} \cdot К_т \cdot d_{этрп}, \tag{6.7}$$

$$T_{утп} = 565 \cdot 1{,}07 \cdot 0{,}63 \cdot 0{,}7 \cdot 0{,}65 \cdot 0{,}7395 = 128{,}15 \text{ чел./дн.}$$

Трудоемкость стадии внедрения рассчитывается по формуле

$$\text{Т}_{увн} = \text{Т}_\text{н} \cdot \text{К}_\text{с} \cdot \text{К}_\text{н} \cdot \text{К}_\text{ур} \cdot d_\text{эвн}, \quad (6.8)$$

$$\text{Т}_{увн} = 565 \cdot 1{,}07 \cdot 0{,}63 \cdot 0{,}7 \cdot 0{,}11 = 29{,}33 \text{ чел./дн.}$$

На основании трудоемкостей каждой из стадии разработки ПС по формуле 6.4 рассчитываем общую трудоемкость проекта

$$\text{Т}_\text{о} = 21{,}33 + 128{,}15 + 29{,}33 = 178{,}81 \text{ чел./дн.}$$

Срок разработки проекта может быть рассчитан по формуле

$$\text{Т}_\text{р} = \frac{\text{Т}_\text{о}}{\text{Ч}_\text{р} \cdot \Phi_\text{эф}}, \quad (6.9)$$

где $\Phi_\text{эф}$ – эффективный фонд времени работы одного работника в течение года (дн.);

$\text{Т}_\text{о}$ – общая трудоемкость разработки проекта, чел./дн.;

$\text{Т}_\text{р}$ – срок разработки проекта, лет.

Эффективный фонд времени одного работника рассчитывается по формуле

$$\Phi_\text{эф} = Д_\text{г} - Д_\text{п} - Д_\text{в} - Д_\text{о}, \quad (6.10)$$

где $Д_\text{г}$ – количество дней в году;

$Д_\text{п}$ – количество праздничных дней в году;

$Д_\text{в}$ – количество выходных дней в году;

$Д_\text{о}$ – количество дней отпуска.

Таким образом, эффективный фонд времени одного работника равен

$$\Phi_\text{эф} = 366 - 6 - 105 - 20 = 235 \text{ дн.}$$

С использование этой информации на основании формулы 6.9 мы можем рассчитать срок разработки проекта для каждой стадии с учетом того, что численность исполнителей проекта составляет 3

$$\text{Т}_\text{ртз} = \frac{21{,}33}{3 \cdot 235} = 0{,}03 \text{ лет},$$

$$Т_{ртрп} = \frac{128,15}{3 \cdot 235} = 0,182 \text{ лет},$$

$$Т_{рвн} = \frac{29,33}{3 \cdot 235} = 0,041 \text{ лет}.$$

В таблице 6.3 приведены уточненные показатели трудоемкости ПС и численности исполнителей по стадиям.

Таблица 6.3 – Уточненные показатели трудоемкости ПС и численности исполнителей по стадиям

Показатели	Стадии					Итого
	ТЗ	ЭП	ТП	РП	ВН	
Коэффициент удельных весов трудоемкости стадии $d_{сті}$	0,08	0	0	0,7395	0,11	0,9295
Коэффициент сложности ПС $К_с$	1,07	1,07	1,07	1,07	1,07	-
Коэффициент, учитывающий использование стандартных модулей, $К_т$	-	-	-	0,65	-	-
Коэффициент, учитывающий новизну ПС, $К_н$	0,63	0,63	0,63	0,63	0,63	-
Коэффициент, учитывающий средства разработки ПС, Кур	0,7	0,7	0,7	0,7	0,7	-
Общая трудоемкость ПС, $Т_у$, чел./дн.	21,33	0	0	128,15	29,33	178,81
Численность исполнителей, $Ч_і$, чел.	3	3	3	3	3	3
Сроки разработки, лет	0,03	0	0	0,182	0,041	0,253

6.2.2 Расчет затрат на разработку программного средства

6.2.2.1 Основная заработная плата

Основная заработная плата исполнителей рассчитывается по формуле

$$З_о = \sum_{i=1}^{n} Т_{чi} \cdot Т_ч \cdot Ф_п \cdot К, \quad (6.11)$$

где n – количество исполнителей, занятых в разработке ПС;

$Т_{чi}$ – часовая тарифная ставка i-го исполнителя, руб.;

$Ф_п$ – плановый фонд рабочего времени i-го исполнителя, дн.;

$Т_ч$ – количество часов работы в день, ч.;

К – коэффициент премирования.

По данным предприятия месячная тарифная ставка первого разряда составляет 298000 рублей. Количество рабочих дней в месяце бралось 21. Разработкой web-приложения занимались три человека: инженер-программист (2,84), инженер-программист (2,84), инженер-программист (2,84).

Исходя из полученных данных, рассчитаем основную заработную плату исполнителя, суммарную заработную плату исполнителей. Полученные данные приведены в таблице 6.4.

Таблица 6.4 – Расчет основной заработной платы исполнителей

Исполнитель	Тарифный коэффициент ($Т_к$)	Месячная тарифная ставка ($Т_м$), руб.	Часовая тарифная ставка ($Т_ч$), руб.	Плановый фонд рабочего времени ($Ф_{пi}$), дн.	Заработная плата (З), руб.
Инженер-программист	2,84	846320	5037,62	278,5	11223817
Инженер-программист	2,84	846320	5037,62	278,5	11223817
Инженер-программист	2,84	846320	5037,62	278,5	11223817
Итого, руб.			33671451		
Премия, руб.			16835726		
Основная заработная плата ($З_о$), руб.			50507177		

6.2.2.2 Дополнительная заработная плата

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (6.12)$$

где $Н_д$ – норматив дополнительной заработной платы, 10%.

Размер дополнительной заработной платы исполнителей составит

$$З_д = \frac{50507177 \cdot 10}{100} = 5050718 \text{ руб.}$$

6.2.2.3 Отчисления в фонд социальной защиты населения и на обязательное страхование

Определяются в соответствии с действующими законодательными актами по формуле

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100}, \quad (6.13)$$

где $Н_{сз}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, 34 + 0,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составит

$$З_{сз} = \frac{(50507177 + 5050718\ ) * 34{,}6}{100} = 19223032 \text{ руб.}$$

6.2.2.4 Расходы по статье «Машинное время»

Определяются по формуле

$$Р_м = Ц_м \cdot \frac{V_о \cdot Н_{мв}}{100}, \quad (6.14)$$

где $Ц_м$ – цена одного часа машинного времени, 5000 руб.;

$V_о$ – общий объём программного средства (строк исходного кода);

$Н_{мв}$ – норматив расхода машинного времени на отладку 100 строк исходного кода, 12ч/100 строк кода.

Расходы на использование машинного времени составят, с учетом того, что общий объем программного средства составляет 14780 строк кода

$$Р_м = 5000 * \frac{14780 * 12}{100} = 8868000 \text{ руб.}$$

6.2.2.5 Расходы по статье «Научные командировки» на программное средство

Определяются по формуле

$$З_д = \frac{З_о \cdot Н_{рнк}}{100}, \quad (6.15)$$

где $Н_{рнк}$ – норматив расходов на командировки в целом по организации, 10%.

Расходы по статье «Научные командировки» составят

$$Р_{нк} = \frac{50507177 \cdot 10}{100} = 5050718 \text{ руб.}$$

6.2.2.6 Расходы по статье «Прочие затраты»

Включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы и определяются по формуле

$$П_з = \frac{З_о \cdot Н_{пз}}{100}, \quad (6.16)$$

где $Н_{пз}$ – норматив прочих затрат в целом по организации, 20%.

Расходы по статье «Прочие затраты» составят

$$П_з = \frac{50507177 \cdot 20}{100} = 10101435 \text{ руб.}$$

6.2.2.7 Затраты по статье «Накладные расходы»

Определяются по формуле

$$Р_н = \frac{З_о \cdot Н_{рн}}{100}, \quad (6.17)$$

где $Н_{рн}$ – норматив накладных расходов в целом по организации, 50%.
Затраты по статье «Накладные расходы» составят

$$Р_н = \frac{50507177 \cdot 50}{100} = 25253589 \text{ руб.}$$

6.2.2.8 Расчет общей суммы расходов

Общая сумма расходов по всем статьям сметы ($С_р$) на ПС рассчитывается по формуле

$$С_р = З_о + З_д + З_{сз} + Р_м + Р_{нк} + П_з + Р_н, \quad (6.18)$$

Общая сумма расходов по всем статьям сметы на разработку ПС составит

$$С_р = 50507177 + 5050718 + 19223032 + 8868000 + 5050718 + \\ + 10101435 + 25253589 = 124\,054\,669 \text{ руб.}$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС ($Р_{са}$), которые определяются по формуле

$$Р_{са} = \frac{С_р \cdot Н_{рса}}{100}, \quad (6.19)$$

где $Н_{рса}$ – норматив расходов на сопровождение и адаптацию, 20%.
Затраты на сопровождение и адаптацию программного продукта составят

$$Р_{са} = \frac{124054669 \cdot 20}{100} = 24810934 \text{ руб.}$$

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программного средства ($С_п$) определяется по формуле

$$С_п = С_р + Р_{са}, \quad (6.20)$$

Полная себестоимость программного средства составит

$$С_п = 124054669 + 24810934 = 148\,865\,603 \text{ руб.}$$

6.2.2.9 Расчет рентабельности и прибыли от реализации ПС заказчику

Рентабельность и прибыль разрабатываемого ПС определяются, исходя из результатов анализа рыночных условий. Прибыль с одного экземпляра программного средства рассчитывается по формуле

$$П_о = \frac{С_п \cdot У_{рп}}{100}, \quad (6.21)$$

где $П_о$ – прибыль от реализации ПС заказчику, руб.;
$У_{рп}$ – уровень рентабельности ПС, 40%;
$С_п$ – себестоимость ПС, руб.
Прибыль от реализации программного средства заказчику составит

$$П_о = \frac{148865603 \cdot 40}{100} = 59546241 \text{ руб.}$$

Прогнозируемая цена программного продукта без налогов ($Ц_п$) рассчитывается по формуле

$$Ц_п = С_п + П_о. \quad (6.22)$$

Прогнозируемая цена программного продукта без налогов составит

$$Ц_п = 148865603 + 59546241 = 208\,411\,844 \text{ руб.}$$

Прогнозируемая отпускная цена программного средства с налогом на добавленную стоимость определяется по формуле

$$Ц_о = Ц_п + НДС, \quad (6.23)$$

где НДС – налог на добавленную стоимость, 20%.
Сумма налога на добавленную стоимость составит

$$НДС = \frac{Ц_п \cdot 20}{100} = 41682369 \text{ руб.}$$

Прогнозируемая отпускная цена программного средства с налогом на добавленную стоимость составит

$$Ц_о = 208411844 + 41682369 = 250\ 094\ 213 \text{ руб.}$$

Прогнозируемая отпускная цена – $Ц_О$ = 250 094 213 бел. рублей. Прибыль остается организации-разработчику и представляет собой экономический эффект от создания нового программного средства.

6.3 Расчет экономического эффекта от применения ПС у заказчика

Для определения экономического эффекта от использования нового ПС необходимо сравнить расходы на эксплуатацию нового ПС с расходами по соответствующим статьям базового варианта. При сравнении базового и нового вариантов ПС в качестве экономического эффекта будет выступать общая экономия всех видов ресурсов. При этом создание нового ПС является экономически целесообразным лишь в том случае, если все капитальные затраты окупятся в ближайшие 1–2 года.

6.3.1 Капитальные затраты на приобретение и использование ПО.

Общие капитальные вложения ($К_о$) заказчика, связанные с приобретением, внедрением и использованием ПО, рассчитывается по формуле

$$К_о = К_{пр} + К_{ос} + К_с + К_{тс} + К_{об}, \qquad (6.24)$$

где $К_{пр}$ – затраты заказчика на приобретение ПО по отпускной цене у разработчика с учетом стоимости услуг эксплуатации, руб.;

$К_{ос}$ – затраты заказчика на освоение ПО, руб.;

$К_с$ – затраты заказчика на оплату услуг по сопровождению ПО, руб.;

$К_{тс}$ – затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПО, руб.;

$К_{об}$ – затраты на пополнение оборотных средств в связи с использованием нового ПО, руб.

Капитальные затраты на приобретение и использование ПО составят

$$К_о = 250094213 + 12405467 + 24810934 + 12405467 + 850000 = 300\ 566\ 081 \text{ руб.}$$

6.3.2 Расчет экономии затрат на заработную плату ($С_з$)

Экономия затрат на заработную плату при использовании нового ПО в расчете на объем выполненных работ, рассчитывается по формуле

$$С_з = С_{зе} \cdot А_2, \quad (6.25)$$

где $С_{зе}$ – экономия затрат на заработную плату при решении задач с использованием нового ПО в расчете на одну задачу, руб.;

$А_2$ – объем выполненных работ с использованием нового ПО, задач.

Исходные данные для расчета экономии ресурсов в связи с применением нового ПО приведены в таблице 6.5.

Экономия затрат на заработную плату в расчете на 1 задачу ($С_{зе}$) рассчитывается по формуле

$$С_{зе} = \frac{З_{см} \cdot (Т_{с1} - Т_{с2}) : Т_ч}{Д_р}, \quad (6.26)$$

где $З_{см}$ – среднемесячная заработная плата одного программиста, руб.;

$Т_{с1}$, $Т_{с2}$ – снижение трудоемкости работ в расчете на 1 задачу, человеко-часов;

$Т_ч$ – количество часов работы в день, ч;

$Д_р$ – среднемесячное количество рабочих дней.

Экономия затрат на заработную плату в расчете на 1 задачу составит

$$С_{зе} = \frac{2700000 \cdot (8 - 2) : 8}{21} = 96429 \text{ руб.}$$

Расчет экономии затрат на заработную плату составит

$$С_з = 96429 \cdot 1900 = 183\,215\,100 \text{ руб.}$$

Таблица 6.5 – Исходные данные для расчета экономии ресурсов

Показатель	Обозначение	Ед. изм.	Значение показателя	
			Базовый вариант	Новый Вариант
1	2	3	4	5
Капитальные вложения, включая затраты заказчика на приобретение ПО	$К_{пр}$	руб.	-	250094213
Затраты на освоение ПО	$К_{ос}$	руб.	-	12405467
Затраты на сопровождение ПО	$К_{с}$	руб.	-	24810934
Затраты на доукомплектование ВТ тех. средствами в связи с внедрением нового ПС	$К_{тс}$	руб.	-	12405467
Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПС	$К_{об}$	руб.	-	850000
Время простоя сервиса, обусловленное ПО, в день	$П_1$, $П_2$	Мин	60	20
Стоимость одного часа простоя	$С_{п}$	руб.	90000	90000
Среднемесячная ЗП одного программиста	$З_{см}$	руб.	2700000	2700000
Коэффициент начислений на зарплату	$Кн_{з}$		1,5	1,5
Объем выполняемых работ	$А_1$, $А_2$	Задача	1900	1900
Средняя трудоемкость работ в расчете на 1 задачу	$Т_{с1}$, $Т_{с2}$	чел.-часов	8	2
Количество часов работы в день	$Т_{ч}$	Ч	8	8
Ставка налога на прибыль	$Н_{п}$	%	-	18

Экономия с учетом начисления на зарплату ($С_{н}$)

$$С_{н} = С_{з} \cdot 1{,}5 = 183215100 \cdot 1{,}5 = 274\,822\,650 \text{ руб.}$$

Экономия за счет сокращения простоев сервиса ($С_с$) рассчитывается по формуле

$$С_с = \frac{(П_1 - П_2) \cdot Д_{рг} \cdot С_п}{60}, \quad (6.27)$$

где $Д_{рг}$ – плановый фонд работы сервиса, дней.

Расчет экономии за счет сокращения простоя сервиса составляет

$$С_с = \frac{(60 - 20) \cdot 235 \cdot 90000}{60} = 14\ 100\ 000 \text{ руб.}$$

Общая годовая экономия текущих затрат, связанных с использованием нового ПО ($С_о$), рассчитывается по формуле

$$С_о = С_н + С_с, \quad (6.28)$$

Расчет общей годовой экономии текущих затрат составляет

$$С_о = 274822650 + 14100000 = 288\ 922\ 650 \text{ руб.}$$

6.3.3 Расчет экономического эффекта от внедрения нового ПС

Внедрение нового ПС позволит заказчику сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль, вычисляемая по формуле

$$\Delta П_ч = С_о - \frac{С_о \cdot Н_п}{100\%}, \quad (6.29)$$

где $Н_п$ – ставка налога на прибыль, 18%.

Таким образом экономический эффект для пользователя от внедрения нового ПС составит

$$\Delta П_ч = 216600900 – (216600900 \cdot 18) /100 = 236\ 916\ 573 \text{ руб.}$$

В процессе использования нового ПС чистая прибыль в конечном итоге возмещает капитальные затраты. Полученные суммы затрат и прибыли необходимо привести к единому моменту времени – расчетному году путем умножения результатов и затрат за каждый год на коэффициент приведения (α), который рассчитывается по формуле

$$\alpha = (1 + E_н)^{t_p - t}, \quad (6.30)$$

где $E_н$ – коэффициент дисконтирования, который зависит от ставки рефинансирования и риска.

$E_н = 0,24 + 0,01 = 0,25$;

t_p – расчетный период (t_P=1);

t – период, потоки которого приводятся к расчетному.

Коэффициентам приведения α по годам представлены в таблице 6.6.

Таблица 6.6 – Коэффициенты приведения по годам

Коэффициент α_i	Год
$\alpha_1 = (1+0,25)^{1-1} = 1$	2016
$\alpha_2 = (1+0,25)^{1-2} = 0,8$	2017
$\alpha_3 = (1+0,25)^{1-3} = 0,64$	2018
$\alpha_4 = (1+0,25)^{1-4} = 0,512$	2019

Сведем данные расчета экономического эффекта в таблицу 6.7.

Таблица 6.7 – Расчет экономического эффекта от внедрения нового ПС

Показатели	Ед. изм.	2016	2017	2018	2019
Прирост прибыли за счет экономии затрат ($П_ч$)	руб.		236916573	236916573	236916573
То же с учетом фактора времени	руб.		189533258	151626607	121301285
Затраты:					
Приобретение ПС ($К_{пр}$)	руб.	250094213			
Освоение ПС ($К_{ос}$)	руб.	12405467			
Сопровождение ($К_с$)	руб.	24810934			
Доукомплектование ВТ ТС ($К_{тс}$)	руб.	12405467			
Пополнение оборотных средств ($К_{об}$)	руб.	850000			
Всего затрат	руб.	300566081			
То же с учетом фактора времени	руб.	300566081			
Экономический эффект:					
Превышение результата над затратами	руб.	-300566081	189533258	151626607	121301285
То же с нарастающим итогом	руб.	-300566081	-111032823	40593784	161895069
Коэффициент приведения		1,0	0,8	0,64	0,512

6.4 Вывод по технико-экономическому обоснованию

Реализация приложения для подбора специальности абитуриента ВУЗа на основе когнитивных сервисов IBM Bluemix позволит заказчику снизить трудоемкость решения задач и сократить простои сервиса и за счет этого является экономически выгодным программным средством.

Из рассчитанных данных видно, что приобретенное и внедренное ПС позволяет получить прибыль в размере 236 916 573 бел. руб.

Из таблицы 7.7 следует, что затраты на приложение для подбора специальности абитуриента ВУЗа на основе когнитивных сервисов IBM Bluemix окупятся за два года эксплуатации.

Разработка и внедрение экономически целесообразны.

ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта было разработан сервис приложения для подбора специальности абитуриента ВУЗа на основе когнитивных сервисов IBM Bluemix. Перед разработкой программного средства были изучены теоретические основы разработки веб-приложений, существующие подходы и технологии, для эффективной разработки.

Также еще до непосредственного старта разработки приложения был произведен качественный анализ предметной области. Была продумана и описана архитектура будущего приложения. Приложение было реализовано согласно разработанной архитектуре. Также были проведены различные виды тестирования приложения, что помогло выявить ошибки реализации приложения и их исправить.

Разработанное приложение несет практическую пользу, являясь качественным сервисом, облегчающим абитуриентам поиск интересной для них специальности. Также в процессе разработки приложения были изучены сервисы и способы работы с IBM Bluemix, в том числе когнитивные сервисы IBM Watson. Запуск системы в рамках университета может помочь поступающим с расстановкой приоритетов специальностей.

Исходя из списка возможностей программного средства, можно сделать вывод, что реализация программного средства соответствует поставленным задачам.

К дальнейшим направлениям развития разработки следует отнести следующие пункты:

1) расширение набора специальностей;

2) использование дополнительных когнитивных сервисов для расширения функциональности приложения;

3) усложнение методов статистики и ранжирования, для более точного определения подходящих специальностей;

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] https://www.ibm.com/developerworks/library/cl-bluemixfoundry/ - Интернет портал компании IBM, который предоставляет информацию о Bluemix для разработчиков

[2] Рафаэль Стифани. IBM Bluemix. The cloud platform for creating and delivering applications – M.: RedPaper, 2015. —110 с.

[3] https://www.ibm.com/developerworks/library/cl-watson/ - Интернет портал компании IBM, который предоставляет информацию о Watson для разработчиков

[4] https://ru.wikipedia.org/wiki/Клиент-сервер - Интернет энциклопедия Википедия, обзор операционной архитектуры клиент-сервер

[5] AngularJS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://angularjs.org/

[6] NodeJS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://nodejs.org/

[7] MongoDB [Электронный ресурс]. – Электронные данные. – Режим доступа: http://mongodb.org/

[8] Палицын В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Мн.: БГУИР, 2006. – 76 с.

[9] Постановление Минтруда и соцзащиты РБ от 21.09.2011 № 90 // Национальный реестр правовых актов Республики Беларусь. – 27.09.2011 – № 8/24207.

[10] Горбушие А.М. Экономический эффект программного продукта. – Мн: ВШ, 2007. – 275с.

ПРИЛОЖЕНИЕ А

(обязательное)

Модуль работы с Twitter API

```
var     querystring = require("querystring"),
        oauth = require("oauth"),
        request = require("request"),
        fs = require("fs");

var baseUrl = "https://api.twitter.com/1.1/";
var uploadBaseUrl = "https://upload.twitter.com/1.1/";
var authUrl = "https://twitter.com/oauth/authenticate?oauth_token=";

var Twitter = function(options) {
        if (!(this instanceof Twitter))
                return new Twitter(options);

        this.consumerKey = options.consumerKey;
        this.consumerSecret = options.consumerSecret;
        this.callback = options.callback;
        this.x_auth_access_type = options.x_auth_access_type;
        this.oa = new oauth.OAuth("https://twitter.com/oauth/request_token",
"https://twitter.com/oauth/access_token",
                this.consumerKey, this.consumerSecret, "1.0A", this.callback, "HMAC-
SHA1");

        return this;
};
Twitter.VERSION = VERSION;

Twitter.prototype.getRequestToken = function(callback) {
        this.oa.getOAuthRequestToken({x_auth_access_type: this.x_auth_access_type},
function(error, oauthToken, oauthTokenSecret, results) {
                if (error) {
                        callback(error);
                } else {
                        callback(null, oauthToken, oauthTokenSecret, results);
                }
        });
};
```

```
Twitter.prototype.getAuthUrl = function(requestToken) {
	return authUrl + requestToken;
};

Twitter.prototype.getAccessToken = function(requestToken, requestTokenSecret,
oauth_verifier, callback) {
	this.oa.getOAuthAccessToken(requestToken, requestTokenSecret, oauth_verifier,
function(error, oauthAccessToken, oauthAccessTokenSecret, results) {
		if (error) {
			callback(error);
		} else {
			callback(null, oauthAccessToken, oauthAccessTokenSecret, results);
		}
	});
};

Twitter.prototype.verifyCredentials = function(accessToken, accessTokenSecret,
params, callback) {
	var url = baseUrl + "account/verify_credentials.json";
	if (typeof params == "function") {
		callback = params;
	} else {
		url += '?' + querystring.stringify(params);
	}
	this.oa.get(url, accessToken, accessTokenSecret, function(error, data, response) {
		if (error) {
			callback(error);
		} else {
			try {
				callback(null, JSON.parse(data));
			} catch (e) {
				callback(e, data, response);
			}
		}
	});
};
```

```
// Timelines
Twitter.prototype.getTimeline = function(type, params, accessToken, accessTokenSe-
cret, callback) {
	type = type.toLowerCase();

	var url;
	switch (type) {
		case "home_timeline":
		case "home":
			url = "home_timeline";
			break;
		case "mentions_timeline":
		case "mentions":
			url = "mentions_timeline";
			break;
		case "user_timeline":
		case "user":
			if (!params.user_id && !params.screen_name) {
				callback("Always specify either an user_id or screen_name
when requesting a user timeline.");
				return false;
			}
			url = "user_timeline";
			break;
		case "retweets_of_me":
		case "retweets":
			url = "retweets_of_me";
			break;
		default:
			callback("Please specify an existing type.");
			return false;
	}

	this.oa.get(baseUrl + "statuses/" + url + ".json?" + querystring.stringify(params),
accessToken, accessTokenSecret, function(error, data, response) {
		if (error) {
			callback(error);
		} else {
			try {
```

```
                        callback(null, JSON.parse(data), response);
                    } catch (e) {
                        callback(e, data, response);
                    }
                }
        });
};

//Streaming
Twitter.prototype.getStream = function(type, params, accessToken, accessTokenSecret,
dataCallback, endCallback) {
        type = type.toLowerCase();

        var url, method = "GET";
        switch (type) {
                case "userstream":
                case "user":
                        url = "https://userstream.twitter.com/1.1/user.json";
                        break;
                case "sitestream":
                case "site":
                        url = "https://sitestream.twitter.com/1.1/site.json";
                        break;
                case "sample":
                        url = "https://stream.twitter.com/1.1/statuses/sample.json";
                        break;
                case "firehose":
                        url = "https://stream.twitter.com/1.1/statuses/firehose.json";
                        break;
                case "filter":
                        method = "POST";
                        url = "https://stream.twitter.com/1.1/statuses/filter.json";
                        break;
                default:
                        var errorMessage = "Please specify an existing type.";
                        dataCallback({message: errorMessage, e: new Error(errorMessage)},
null, null, null);
                        return false;
        }
```

```
	var req;
	if (method == "GET") {
		req = this.oa.get(url + "?" + querystring.stringify(params), accessToken, accessTokenSecret);
	} else {
		req = this.oa.post(url, accessToken, accessTokenSecret, params, null);
	}
	var msg = [];
	req.addListener("response", function(res) {
		res.setEncoding("utf-8");
		res.addListener("data", function(chunk) {
			if (chunk == "\r\n") {
				dataCallback(null, {}, chunk, res);
				return;
			} else if (chunk.substr(chunk.length - 2) == "\r\n") {
				msg.push(chunk.substr(0, chunk.length - 2));
				var ret = msg.join("");
				msg = [];

				var parsedRet;
				try {
					parsedRet = JSON.parse(ret);
				} catch (e) {
					dataCallback({
						message: "Error while parsing Twitter-Response.",
						error: e
					}, null, chunk, res);
					return;
				}
				dataCallback(null, parsedRet, ret, res);
				return;
			} else {
				msg.push(chunk);
				return;
			}
		});
		res.addListener("end", function() {
```

```
                endCallback(res);
            });
        });
        req.end();

        return req;
};

// Tweets
Twitter.prototype.statuses = function(type, params, accessToken, accessTokenSecret,
callback) {
        var url = type.toLowerCase();

        var method = "GET";
        switch (type) {
            case "retweets":
                url = "retweets/" + params.id;
                delete params.id;
                break;
            case "show":
                url = "show/" + params.id;
                delete params.id;
                break;
            case "lookup":
                url = "lookup";
                method = "POST";
                break;
            case "destroy":
                url = "destroy/" + params.id;
                delete params.id;
                method = "POST";
                break;
            case "update":
                method = "POST";
                break;
            case "retweet":
                url = "retweet/" + params.id;
                delete params.id;
                method = "POST";
```

```
                break;
        case "oembed":
                url ="oembed";
                break;
        case "upload_media":
                this.uploadMedia(params, accessToken, accessTokenSecret,
callback);
                return;
        case "update_with_media":
                callback("'update_with_media' type has been removed. Use 'up-
load_media' instead");
                return false;
        default:
                callback("Please specify an existing type.");
                return false;
    }

    if (method == "GET") {
        this.oa.get(baseUrl + "statuses/" + url + ".json?" +
querystring.stringify(params), accessToken, accessTokenSecret, function(error, data, re-
sponse) {
                if (error) {
                        callback(error, data, response, baseUrl + "statuses/" + url +
".json?" + querystring.stringify(params));
                } else {
                        try {
                                callback(null, JSON.parse(data), response);
                        } catch (e) {
                                callback(e, data, response);
                        }
                }
        });
    } else {
        this.oa.post(baseUrl + "statuses/" + url + ".json", accessToken, ac-
cessTokenSecret, params, function(error, data, response) {
                if (error) {
                        callback(error, data, response);
                } else {
                        try {
```

```
                                callback(null, JSON.parse(data), response);
                        } catch (e) {
                                callback(e, data, response);
                        }
                }
        });
    }
};

Twitter.prototype.uploadMedia = function(params, accessToken, accessTokenSecret,
callback) {
    var r = request.post({
        url: uploadBaseUrl + "media/upload.json",
        oauth: {
            consumer_key: this.consumerKey,
            consumer_secret: this.consumerSecret,
            token: accessToken,
            token_secret: accessTokenSecret
        }
    }, function(error, response, body) {
        if (error) {
            callback(error, body, response, uploadBaseUrl + "media/up-
load.json?" + querystring.stringify(params));
        } else {
            try {
                callback(null, JSON.parse(body), response);
            } catch (e) {
                callback(e, body, response);
            }
        }
    });

    var parameter = (params.isBase64) ? "media_data" : "media";

    // multipart/form-data
    var form = r.form();
    if (fs.existsSync(params.media)) {
        form.append(parameter, fs.createReadStream(params.media));
    } else {
```

```
                form.append(parameter, params.media);
        }
};

// Search
Twitter.prototype.search = function(params, accessToken, accessTokenSecret, callback)
{
        this.oa.get(baseUrl + "search/tweets.json?" + querystring.stringify(params), ac-
cessToken, accessTokenSecret, function(error, data, response) {
                if (error) {
                        callback(error, data, response, baseUrl + "search/tweets.json?" +
querystring.stringify(params));
                } else {
                        try {
                                callback(null, JSON.parse(data), response);
                        } catch (e) {
                                callback(e, data, response);
                        }
                }
        });
};

// Users
Twitter.prototype.users = function(type, params, accessToken, accessTokenSecret,
callback) {
        var url = type.toLowerCase();

        var method = "GET"; // show, search, contributees, contributors
        if (url == "lookup") method = "POST";


        if (method == "GET") {
                this.oa.get(baseUrl + "users/" + url + ".json?" +
querystring.stringify(params), accessToken, accessTokenSecret, function(error, data, re-
sponse) {
                        if (error) {
                                callback(error, data, response, baseUrl + "users/" + url +
".json?" + querystring.stringify(params));
                        } else {
```

```
                        try {
                                callback(null, JSON.parse(data), response);
                        } catch (e) {
                                callback(e, data, response);
                        }
                }
        });
    } else {
        this.oa.post(baseUrl + "users/" + url + ".json", accessToken, accessToken-
Secret, params, function(error, data, response) {
                if (error) {
                        callback(error, data, response);
                } else {
                        try {
                                callback(null, JSON.parse(data), response);
                        } catch (e) {
                                callback(e, data, response);
                        }
                }
        });
    }
};
// OAuth
Twitter.prototype.oauth = function(type, params, accessToken, accessTokenSecret,
callback) {
    var url = type.toLowerCase();
    var method = "GET";

    switch (url) {
        case "access_token":
        case "request_token":
                method = "POST";
                url = "oauth/" + url;
                break;
        case "token":
        case "invalidate_token":
                method = "POST";
                url = "oauth2/" + url;
                break;
```

```
		default:
			url = "oauth/" + url;
			break;
	}

	if (method == "GET") {
		this.oa.get(baseUrl + url + ".json?" + querystring.stringify(params), ac-
cessToken, accessTokenSecret, function(error, data, response) {
			if (error) {
				callback(error, data, response, baseUrl + "geo/" + url +
".json?" + querystring.stringify(params));
			} else {
				try {
					callback(null, JSON.parse(data), response);
				} catch (e) {
					callback(e, data, response);
				}
			}
		});
	} else {
		this.oa.post(baseUrl + url + ".json", accessToken, accessTokenSecret,
params, function(error, data, response) {
			if (error) {
				callback(error, data, response);
			} else {
				try {
					callback(null, JSON.parse(data), response);
				} catch (e) {
					callback(e, data, response);
				}
			}
		});
	}
};
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Файл описывающий узловые точки диалога

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog xsi:noNamespaceSchemaLocation="WatsonDialogDocument_1.0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <flow>
        <folder id="Main" label="Main">
            <output id="output_300001">
                <prompt>
                    <item>Hello, I'm [BotName]. I will help you with you
choise of university. But to do it, I need to read your twitter account. So, please, give me
your twitter account in form "@twitter"</item>
                </prompt>
                <goto ref="getUserInput_200001"/>
            </output>

            <getUserInput id="getUserInput_200001">

                <search ref="inputSequence_100004"/>
                <search ref="inputSequence_100003"/>
                <search ref="inputSequence_100001"/>
                <search ref="inputSequence_100002"/>
                <search ref="inputSequence_100005"/>

                <default>
                    <output>
                        <prompt>
                            <item>Sorry, I did not understand you.
May be, you'll try something else?</item>
                        </prompt>
                    </output>
                </default>
            </getUserInput>

            <output id="output_300002">
                <prompt>
```

```
                        <item>Do you want to start from the begin-
ning?</item>
                    </prompt>
                    <goto ref="getUserInput_200002"/>
                </output>

                <getUserInput id="getUserInput_200002">
                    <search ref="inputSequence_100006"/>
                    <default>
                        <output>
                            <prompt>
                                <item>I don't understand!</item>
                            </prompt>
                        </output>
                    </default>
                </getUserInput>

                <getUserInput id="getUserInput_200003">

                    <search ref="inputSequence_100002"/>
                    <search ref="inputSequence_100003"/>
                    <search ref="inputSequence_100004"/>
                    <search ref="inputSequence_100005"/>
                    <search ref="inputSequence_100007"/>

                    <default>
                        <output>
                            <prompt>
                                <item>I don't understand, so you can try
something else.</item>
                            </prompt>
                        </output>
                    </default>
                </getUserInput>
            </folder>
            <folder id="Library" label="Library">
                <folder id="inputSequence_100001" comment="Accept/Decline to
give twitter account">
                    <input>
```

```
                        <grammar>
                                <item>No</item>
                                <item>Never</item>
                                <item>I do not want</item>
                                <item>I don't want</item>
                                <item>I don't want *</item>
                        </grammar>
                        <output>
                                <prompt>
                                        <item>It's quite sad. I can not help you
without this! May be you'll give it to me?</item>
                                </prompt>
                        </output>
                </input>
                <input>
                        <grammar>
                                <item>Yes</item>
                                <item>Yep</item>
                                <item>yeah</item>
                                <item>Sure</item>
                                <item>Of course</item>
                                <item>Ok</item>
                        </grammar>
                        <output>
                                <prompt>
                                        <item>I'm very satisfacted by this. But,
sadly, I still don't know your twitter account. May be you'll specify it?</item>
                                </prompt>
                        </output>
                </input>
        </folder>
        <folder id="inputSequence_100002" comment="Questions from
user to bot">
                <input>
                        <grammar>
                                <item>Who are you?</item>
                                <item>Where are you from?</item>
                                <item>Tell about yourself</item>
                                <item>Tell * about yourself</item>
```

```
                    </grammar>
                    <output>
                        <prompt>
                            <item>I'm bot. My name is [BotName]. I
live in the clouds. My main aim is to help you to make one of the most important
choices of your life. My teacher told me, that he can get personality portrait of one if he
take a look only on a text written by a person. He taught me that. As the technology
spreads around the world, I found, that people prefare to share their emotions and
thoughts using Twitter. So, I decided to use it and help people to decide where to go in
their life. Now, when you know so much about me, can you give me your twitter ac-
count?</item>
                        </prompt>
                    </output>
                </input>
                <input>
                    <grammar>
                        <item>How to use?</item>
                        <item>$ FAQ </item>
                        <item>What to tell?</item>
                        <item>How * use you?</item>
                    </grammar>
                    <output>
                        <prompt>
                            <item>Oh, it's very simple. Just give me
your twitter account in form "@twitter", and I will do everything i need to help you.
So,would you do it now?</item>
                        </prompt>
                    </output>
                </input>
            </folder>
            <folder id="inputSequence_100003" comment="User gives ac-
count">
                <input>
                    <grammar>
                        <item>! @\w+</item>
                    </grammar>
                    <action varName="TwitterAccount" opera-
tor="SET_TO_USER_INPUT"/>
                    <output>
```

```
                                        <prompt>
                                            <item>Ok, I've got it, {TwitterAccount}!
Now, wait a second and let me think about the most appropriate place.</item>
                                        </prompt>
                                    </output>
                                </input>
                                <input>
                                    <grammar>
                                        <item>[UtilityInputUniversityFound]</item>
                                    </grammar>
                                    <!--<action varName="UniversityName" opera-
tor="SET_TO">Princeton</action>-->
                                    <action varName="TwitterAccount" opera-
tor="SET_TO_BLANK"/>
                                    <output>
                                        <prompt>
                                            <item>Ok, i finished. I decided, that {Uni-
versityName} is a good place for you! Visit their web-site: {UniversityWebsite} - to know
more.</item>
                                        </prompt>
                                        <goto ref="output_300002"/>
                                    </output>
                                </input>
                            </folder>
                            <folder id="inputSequence_100004" comment="Errors handling">
                            </folder>
                            <folder id="inputSequence_100005" comment="Others">
                                <input>
                                    <grammar>
                                        <item>Hi</item>
                                        <item>Hello</item>
                                        <item>yo</item>
                                    </grammar>
                                    <output>
                                        <prompt>
                                            <item>Hello. Nice to meet you! As you
may know - time is a money. So, let's spend your time with profit and find best matching
university for you. Give me your twitter account in form "@twitter".</item>
                                        </prompt>
```

```
                    </output>
                </input>
                <input>
                    <grammar>
                        <item>$ repeat recomendation</item>
                        <item>$ give * recomendation</item>
                        <item>$ repeat * suggestion</item>
                    </grammar>
                    <if matchType="ALL">
                        <cond varName="UniversityName" operator="HAS_VALUE"/>
                        <output>
                            <prompt>
                                <item>I suggest you to enroll {UniversityName}. Visit their website: {UniversityWebsite} - to know more.</item>
                            </prompt>
                        </output>
                    </if>
                    <if matchType="ALL">
                        <cond varName="UniversityName" operator="IS_BLANK"/>
                        <output>
                            <prompt>
                                <item>I haven't suggested you any university to enroll. Please, specify your twitter account to help me do it.</item>
                            </prompt>
                        </output>
                    </if>
                </input>
            </folder>
            <folder id="inputSequence_100006" comment="Accept/Decline to start from beginning">
                <input>
                    <grammar>
                        <item>No</item>
                        <item>Never</item>
                        <item>I do not want</item>
                        <item>I don't want</item>
                        <item>I don't want *</item>
```

```
                    </grammar>
                    <output>
                        <prompt>
                            <item>Ok, you can ask me something, or
try and specify another account.</item>
                        </prompt>
                        <goto ref="getUserInput_200003"/>
                    </output>
                </input>
                <input>
                    <grammar>
                        <item>Yes</item>
                        <item>Yep</item>
                        <item>yeah</item>
                        <item>Sure</item>
                        <item>Of course</item>
                        <item>Ok</item>
                    </grammar>
                    <output>
                        <prompt>
                            <item>Ok! I will pretend I don't know you
:)</item>
                        </prompt>
                        <output>
                            <prompt>
                                <item>Please, give me your twit-
ter account in form "@twitter".</item>
                            </prompt>
                            <action varName="TwitterAccount" op-
erator="SET_TO_BLANK"/>
                            <action varName="UniversityName" op-
erator="SET_TO_BLANK"/>
                            <action varName="UniversityWebsite"
operator="SET_TO_BLANK"/>

                            <goto ref="getUserInput_200001"/>
                        </output>
                    </output>
                </input>
```

```
                    </folder>
                    <folder id="inputSequence_100007" comment="Retry request from
user">
                        <input>
                            <grammar>
                                <item> I want to retry</item>
                                <item> I want to begin again</item>
                                <item> Let me try again</item>
                                <item> $ do it from the start</item>
                            </grammar>
                            <output>
                                <prompt>
                                    <item>Ok! Let's do it from the start.
Specify your twitter account in form "@twitter".</item>
                                </prompt>
                                <action varName="TwitterAccount" opera-
tor="SET_TO_BLANK"/>
                                <action varName="UniversityName" opera-
tor="SET_TO_BLANK"/>
                                <action varName="UniversityWebsite" opera-
tor="SET_TO_BLANK"/>
                                <goto ref="getUserInput_200001"/>
                            </output>

                        </input>
                    </folder>
                </folder>
                <folder id="Concepts" label="Concepts">
                </folder>
        </flow>

        <entities>
        </entities>

        <constants>
            <var_folder name="UtilityData">
                <var name="UtilityInputUniversityFound" type="TEXT" descrip-
tion="Utility input to trigger dialog continuation">anodaini-
oscn121lkn12j3n12o4n123n</var>
```

```
            </var_folder>
            <var_folder name="ErrorData">
                <var name="TwitterUserErrorOccured" type="TEXT" description="Constant to trigger the input branch when error occured during communication with twitter">alskcenveo2323uib32inf239c</var>
                <var name="PINotEnoughWords" type="TEXT" description="Constant to trigger the input branch when there are not enough words for PI to extract personality inforamtion">aosind129d2n0c12e9n019n</var>
            </var_folder>
            <var_folder name="BotPersonality">
                <var name="BotName" type="TEXT" description="Name of the bot">Enhel-bot</var>
                <var name="BotFullName" type="TEXT" description="Decrypted bot name">Enrolee-Helper bot</var>
            </var_folder>
        </constants>

        <variables>
            <var_folder name="UniversityData">
                <var name="UniversityName" type="TEXT" description="Name of the university"/>
                <var name="UniversityWebsite" type="TEXT" description="university website URL"/>
            </var_folder>
            <var_folder name="TwitterData">
                <var name="TwitterAccount" type="TEXT" description="Twitter Account"/>
            </var_folder>
            <var_folder name="ErrorMessages">
                <var name="TwitterErrorMessage" type="TEXT" description="Error message when twitte account hasn't been found"/>
                <var name="PIErrorMessage" type="TEXT" description="Error message when there is not enough words for PI"/>
            </var_folder>
        </variables>
</dialog>
```