|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждение высшего образования  Национальный исследовательский университет  «Высшая школа экономики»  Факультет экономики, менеджмента и бизнес-информатики |
|  |
| Репин Дмитрий Юрьевич  **АВТОМАТИЗАЦИЯ ОБУЧЕНИЯ ТЕХНОЛОГИИ MSF: РАЗРАБОТКА ВИРТУАЛЬНОГО СОБЕСЕДНИКА ДЛЯ СБОРА ИНФОРМАЦИИ ДЛЯ РАЗВЕРТЫВАНИЯ ПРОГРАММНЫХ ПРОДУКТОВ**  *Выпускная квалификационная работа*  студента образовательной программы «Программная инженерия»  по направлению подготовки *09.03.04 Программная инженерия*   |  |  | | --- | --- | | Рецензент  Директор ООО «Белая логика»  М.А. Андронов | Руководитель  кандидат технических наук, доцент кафедры информационных технологий в бизнесе  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  А.В. Бузмаков | |

Пермь, 2018 год

**Аннотация**

В этой работе описан процесс реализации виртуального собеседника с помощью лексико-синтаксических шаблонов с использованием языка LSPL. Для разработанного виртуального собеседника реализована система администрирования. Работа содержит 3 главы, 13 иллюстраций и 4 приложения.

Оглавление

[Глава 1. Описание предметной области 7](#_Toc515258023)

[1.1 MSF как метод разработки программного обеспечения 7](#_Toc515258024)

[1.2 Деловая игра «Развертывание» 8](#_Toc515258025)

[1.3 Анализ игры 9](#_Toc515258026)

[1.4 Автоматизация игры с помощью виртуального собеседника 10](#_Toc515258027)

[1.5 Требования к разрабатываемой системе 10](#_Toc515258028)

[Глава 2. Проектирование системы 12](#_Toc515258029)

[2.1 Анализ возможных проектных решений алгоритма реализации виртуального собеседника 12](#_Toc515258031)

[2.2 LSPL – язык лексико-синтаксических шаблонов 14](#_Toc515258032)

[2.3 Проектирование алгоритма работы системы 15](#_Toc515258033)

[2.4 Определение бизнес-процессов 19](#_Toc515258034)

[2.5 Проектирование системы 22](#_Toc515258035)

[Глава 3. Разработка системы 25](#_Toc515258036)

[3.1 Инструментальные средства разработки 25](#_Toc515258038)

[3.2 Разработка базы данных 26](#_Toc515258039)

[3.3 Подготовка файла с входными данными 26](#_Toc515258040)

[3.4 Наполнение базы данных 28](#_Toc515258041)

[3.5 Использование лексико-синтаксических шаблонов 33](#_Toc515258042)

[3.6 Разработка веб-приложения 37](#_Toc515258043)

[3.7 Разработка сервиса администрирования 39](#_Toc515258044)

[3.8 Публикация системы на Microsoft Azure и опытное тестирование 40](#_Toc515258045)

[Заключение 42](#_Toc515258046)

[Библиографический список 43](#_Toc515258047)

[Приложение A. Техническое задание 44](#_Toc515258048)

[Приложение Б. Пример использования системы 50](#_Toc515258049)

[Приложение В. Лексико-синтаксические шаблоны 53](#_Toc515258050)

[Приложение Г. Листинг 55](#_Toc515258051)

**Введение**

Современное продуктивное обучение предполагает интенсивную работу как со стороны студентов, так и со стороны преподавателей, в том числе и выполнение большого количества проектов для усвоения полученных знаний на практике. Зачастую такие проекты требуют больших затрат времени как со стороны преподавателя, так и со стороны студентов. В этом случае такие проекты должны быть автоматизированы, хотя бы частично. Об автоматизации одного из таких учебных проектов и пойдет речь в этой работе.

В ходе работы будет улучшен разработанный ранее виртуальный собеседник для деловой игры «Развертывание» по технологии MSF. В ходе деловой игры студенты делятся на группы, после чего группы должны опросить сотрудников отделов фирмы и ее филиалов в четырех городах, разбросанных по всему миру. Каждый сотрудник обладает своей компетенцией и не знает всей информации о филиале, но все сотрудники одного филиала в сумме обладают информацией обо всем отделе. Ранее за каждого сотрудника отвечал преподаватель, но это вызывает ряд проблем. Во-первых, это трудозатраты преподавателя, так как ему приходится отвечать каждой группе студентов. Во-вторых, это затраты по времени для студентов, так как зачастую между вопросом и ответом может пройти большое количество времени. И в-третьих, это плохо сказывается на всей игре в целом, так как она выходит достаточно затянутой.

Для решения ранее описанных проблем в прошлом году было решено написать виртуального собеседника, который бы отвечал за сотрудников отделов вместо преподавателя, который бы требовал минимум трудозатрат для преподавателя и оперативно отвечал студентам, тем самым делая процесс проведения игры более комфортным и быстрым. Собеседник был разработан и протестирован на студентах второго курса направления «Программная инженерия». Тестирование показало, что качество работы собеседника недостаточно высоко.

Проблема. У деловой игры существует три проблемы. Для преподавателя это трудозатраты, для студентов - затраты по времени и для игры в целом это ее длительность.

Актуальность работы: работа является актуальной, т.к. цель, поставленная в прошлом году, достигнута не полностью.

Объект работы: процесс проведения. деловой игры по развертыванию программного обеспечения по технологии MSF.

Предмет работы: автоматизация консультационной работы преподавателя в ходе проекта.

Научная новизна работы: Улучшение ранее разработанного метода решения проблемы, а именно внедрение в учебный процесс виртуального собеседника.

Цель: улучшение виртуального собеседника для сбора информации для развертывания программных продуктов в рамках деловой игры, что ускорит процесс игры и снизит трудозатраты преподавателя, и снизит затраты студентов по времени.

Задачи:

1. Анализ накопленного опыта
2. Разработка виртуального собеседования
3. Разработка системы администрирования
4. Опытная эксплуатация системы

Практическая ценность работы заключается в том, что разработанная программная система берет на себя часть работы преподавателя в ходе проекта и таким образом освобождает его время, а также

Теоретическая ценность работы в сравнительном анализе возможных механизмов реализации виртуальных собеседников и демонстрация практической возможности реализации одного из них.

Работа разделена на 3 главы. Первая глава содержит описание предметной области и анализ автоматизируемого процесса. Во второй главе приведен анализ предыдущей версии системы, описано проектирование и подобраны алгоритмы для реализации, а также будет определена методика разработки системы. Третья глава содержит информацию о разработке виртуального собеседника, системы администрирования и апробации виртуального собеседника.

# Глава 1. Описание предметной области

В первой главе будет описана деловая игра по технологии MSF «Развертывание», выявлены недостатки в предыдущей ее версии, будут сформированы требования к новой версии системы.

## MSF как метод разработки программного обеспечения

MSF (Microsoft Solution Framework) – методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

По MSF процесс разработки программного обеспечения имеет спиральную форму. Итерация цикла представлена на рис. 1.1:



Рисунок 1.1. Модель MSF [1]

Каждая итерация состоит из следующих этапов:

1. Envision (Выработка концепции). На первом этапе создается основной состав проектной группы и готовится описание проекта, оговариваются рамки проекта.
2. Planning (Планирование). На этом этапе составляются планы проекта, в которые входят функциональные спецификации, дизайн разрабатываемого приложения, подготовку рабочих планов, оценку затрат и сроков.
3. Developing (Разработка). На этапе разработки пишется код приложения, подготавливается документация.
4. Stabilizing (Стабилизация). На этом этапе проводится тестирование приложения, устраняются обнаруженные ошибки. Внимание фокусируется на предполагаемых сценариях использования приложения
5. Deploying (Развертывание). На этапе развертывания происходит внедрение готового приложения в рабочий процесс, происходит налаживание сопровождение приложения, закрывается или продлевается контракт с заказчиком.

## Деловая игра «Развертывание»

Для изучения технологии MSF в НИУ ВШЭ-Пермь используется ряд деловых игр: Envision&Planning, «Стабилизация» и «Развертывание». В ходе каждой из игр студенты попадают в сложные ситуации, которые как можно более точно имитируют прецеденты на реальном производстве.

Целью игры «Развертывание» является обучение студентов процессу развертывания программного обеспечения для пользователей. В ходе игры студенты должны установить вымышленную систему «ОбалдеИТ» на компьютеры пользователей. Игра делится на несколько этапов:

1. Сбор информации о подразделениях фирмы «ОбалдеИТ».
2. Формирование расписания развертывания.
3. Подведение итогов.
4. Корректировка расписания после полученной информации в ходе подведения итогов

В ходе процесса игры студенты должны понять основополагающие принципы развертывания. Во-первых, этап развертывания имеет место быть, и он требует тщательной подготовки. Во-вторых, требуется собрать информацию о каждом месте, где будет проводиться процесс развертывания, т.к. каждый филиал имеет свои особенности, такие как разный уровень квалификации персонала, разные характеристики ЭВМ, разное программное обеспечение, установленное на них и разные его настройки. Помимо этого, есть и другие пункты, которые стоит учесть: расписание работы персонала, климат, средства коммуникации и т.д. В-третьих, процесс сбора информации требует усилий и времени, поскольку информацию от работников получить зачастую бывает непросто, и почти всегда эта информация требует анализа для того, чтобы получить нужные сведения для процесса развертывания. В-четвертых, студенты должны осознать необходимость плана для развертывания, в котором следует учесть требуемые действия, время и ресурсы.

## Анализ игры

Разберем процесс проведения игры в рамках обучения студентов. Сначала студенты сами объединяются в команды и сообщают о своих командах преподавателю. После этого команда формирует список вопросов и отправляет их преподавателю или его помощнику, поскольку они играют роли сотрудников подразделений фирмы «ОбалдеИТ», после чего получают от них ответ через некоторое время. Далее студенты составляют следующих список вопросов и снова отправляют преподавателю: это продолжается до тех пор, пока не закончится время на игру, либо студенты не будут удовлетворены объемом полученных сведений. После этого студенты составляют свои планы развертывания и обсуждают их с преподавателем.

Описанный выше процесс имеет ряд достоинств и недостатков. Разберем те черты процесса, которые хотелось бы сохранить в разрабатываемом виртуальном собеседнике:

1. Преподаватель имеет исчерпывающую информацию о подразделениях и при правильно заданных вопросах может поделиться ей со студентами.
2. Общение с преподавателем одновременно доступно сразу нескольким группам студентов.

Далее были выявлены недостатки, от которых хотелось бы избавиться в разрабатываемой системе.

1. Время на игру. К сожалению, может пройти достаточно большой промежуток времени, прежде чем преподаватель ответит студентам всем студентам, таким образом, игра занимает у ее участников несколько дней.
2. Трудозатраты преподавателя. Преподаватель вынужден отвечать всем студентам на многочисленные вопросы, что требует значительных усилий.
3. Удобство студентов. Студенты вынуждены ждать своей очереди и находить способ наискорейшей коммуникации с преподавателем.

## Автоматизация игры с помощью виртуального собеседника

Исходя из выявленных недостатков деловой игры было принято решение о создании системы, которая бы могла имитировать общение того или иного сотрудника любого из имеющихся филиалов. Эта система должна учитывать особенности филиалов такие как климат, характеристики технического оборудования, характер и менталитет сотрудников, особенности коммуникации. Таким образом, в системе должно быть заложено несколько виртуальных собеседников, каждый из которых отвечал бы за одного из имеющихся сотрудников. При этом должно быть учтено, что сотрудник определенной должности может не работать в одном из филиалов и работать в другом, помимо этого сотрудники одинаковых должностей в разных филиалах фирмы обладают различной информацией, исходя из особенностей филиала.

Поскольку все виртуальные собеседники отвечают за одну предметную область, но каждый из них имеет несколько отличающийся от других набор знаний, а также учитывая большое количество самих собеседников, имеет смысл объединить всех собеседников в одну систему с одной базой знаний.

Одним из главных требований к системе – диалог должен происходить на естественном языке, поскольку любая формализация общения сработает как подсказка для студентов. Однако стоит учитывать, что, как и в диалоге между людьми ответы могут быт неверными или неточными, поскольку вопрос может быть неправильно понят

Стоит также обратить внимание на администрирование созданной системы: требуется создать удобный интерфейс для добавления определенного ответа на задаваемый вопрос для каждого из собеседников, а также добавлять уже созданные пары вопрос-ответ к имеющимся собеседникам.

## Требования к разрабатываемой системе

Исходя из описания автоматизации системы, были сформированы требования к разрабатываемой системе:

Использование:

Анализ вопроса: филиал, сотрудник, вопрос.

* + - 1. Выбор филиала.
      2. Выявление наличия в указанном филиале фирмы сотрудника с указанной должностью.
      3. Определение ответа сотрудника на заданный вопрос.

Администрирование:

1. Возможность импортирования сотруднику пары вопрос-ответ из области знаний другого сотрудника.
2. Редактирование областей знаний сотрудников.
3. Возможность просмотра заданных вопросов.

Полный список требований к системе представлен в приложении А.

# Глава 2. Проектирование системы

После того, как в предыдущей главе была описана предметная область и выдвинуты требования к системе, приступим к ее проектированию. В этой главе будут определены бизнес-процессы, в которых задействован разрабатываемый виртуальный собеседник, построены диаграммы в нотации UML для более наглядного их отображения и будут выбраны алгоритмы, на которых будет работать разрабатываемая система.



## Анализ возможных проектных решений алгоритма реализации виртуального собеседника

Как уже было сказано, первые два года игра проводилась в ручном режиме. При этом были сохранены все заданные в ходе игры вопросы и данные на них ответы. Пары «вопрос-ответ» были привязаны к конкретным должностям в конкретных филиалах. Таким образом, была получена некоторая база четверок «филиал – должность – вопрос – ответ». Готовый входной файл содержал более 400 записей. Выбор проектных решений был скорректирован, с поправкой на размер составленного файла.

Были рассмотрены следующие варианты проектных решений:

1. Поиск по полному совпадению вопроса. Наиболее простой вариант, где введенный запрос пользователя дословно сопоставляется с одним из вопросов, имеющихся в базе. Такой вариант реализации не учитывает возможные вариации вопросов, использование синонимов и требует чрезвычайно большую базу вопросов-ответов. На практике этот вариант даже не пробовали реализовать.
2. Поиск по набору ключевых слов. Усложненная версия первого варианта, где в запросе пользователя производится поиск наборов ключевых слов, которые могут стоять в произвольном порядке и могут быть перемешаны с «шумовыми словами». На базе этой технологии была разработана версия виртуального собеседника для той же деловой игры. Однако ее работа имела очень низкую точность и для практического применения оказалась не пригодна.
3. Искусственные нейронные сети (ИНС). Искусственная нейронная сеть – упрощенная модель человеческого мозга. Сеть обучают на базе имеющихся вопросов, после чего она работает с запросами пользователя и выдается результат на основе ранее пройденного обучения. Разработка виртуального собеседника с использованием нейронных сетей рассматривалась, однако, база из 430 входных записей слишком мала для обучения ИНС, поскольку отношение элементов выборки к количеству признаков очень мало[2]. Исходя из следствия теоремы Арнольда–Колмогорова-Хехт–Нильсена[3], велика вероятность переобучения ИНС.
4. Онтологии. Онтология – способ формализации некоторой области знаний с помощью концептуальной схемы. Как правило такая схема состоит из структур данных, содержащей все релевантные классы объектов, их связи и правила, принятые в этой области. Этот метод решения рассматривался для решения поставленной задачи, однако был отвергнут в пользу лексико-синтаксических шаблонов в связи с опытом в использовании их.
5. Лексико-синтаксические шаблоны (ЛСШ). Лексико-синтаксический шаблон - структурный образец языковой конструкции, позволяющий обрабатывать исходный текст, учитывая лексические и синтаксические характеристики естественного языка. Для русского языка это падежи, времена, наклонения, склонения, числа и т.д. В основном шаблоны используются для автоматической обработки текста: выявления языковых конструкций, классификации текстов, поиска словоформ, изменения текстов с учетом заданных параметров (синонимов, выражений). Помимо этого, с помощью лексико-синтаксических шаблонов есть возможность распознавать альтернативы частей исходного текста, что дает возможность задавать синонимы определенным словам. Предельно упрощенной формой ЛСШ являются БНФ.

Год назад была разработана версия виртуального собеседника с использованием лексико-синтаксических шаблонов. Она имела почти такой же интерфейс, но была куда проще с технической точки зрения. Был взят список вопросов, которые студенты задавали в ходе проведения предыдущих деловых игр. Эти вопросы были несколькими студентами вручную преобразованы в ЛСШ (в расчете на то, что студенты будут и дальше задавать такие же вопросы). Это позволило в сжатые сроки получить набор из примерно сотни шаблонов. Версия была применена на практике, но точность работы собеседника было на уровне 30%, что не является достаточным показателем для имитации диалога.

Попытка использования первой версии собеседника выявила следующие проблемы:

Количество шаблонов. Было разработано всего около сотни шаблонов, количество альтернатив в которых было крайне ограничено.

Отсутствие концептуальной целостности. Не было установлено никаких правил или соглашений о построении шаблонов. Каждый из студентов составлял шаблоны на свое усмотрение.

Недостаточный учет многообразия словоформ и синонимии русского языка.

Помимо проблемы в самих шаблонах, отсутствовала возможность администрирования приложения. Любые административные действия (просмотр информационной базы и ее корректировка) мог выполнять только программист- автор системы средствами СУДБ SQL.

Была выдвинута гипотеза, что качество работы собеседника можно существенно улучшить, если увеличить количество ЛСШ и учесть словоформы и синонимию русского языка. Для этого было принято проектное решение разработать механизм автоматической генерации ЛСШ на базе онлайн словарей синонимов русского языка.

Вторая версия виртуального собеседника призвана решить проблемы предыдущей системы, разработанной на основе ЛСШ

## LSPL – язык лексико-синтаксических шаблонов

Предполагается, что вопросы, обрабатываемые виртуальным собеседником, будут задаваться на русском языке, поэтому было решено взять за основу утилиту, которая также работает с шаблонами, которые бы сопоставлялись с русским текстом. Было решено использовать язык LSPL (Lexico-Syntaxic Pattern Language). Этот язык предназначен для описания конструкций русского языка с целью их представления в системах извлечения информации из текстов, однако помимо этого он может сравнивать текст с шаблоном, который представляет набор символов на любом языке. Разработка языка началась в 2007 году, но он продолжает изменяться и дополняться по сей день.

Ключевым в языке LSPL является понятие лексико-синтаксического шаблона, рассматриваемого как структурный образец языковой конструкции. Шаблон задает ее лексический состав и поверхностно-синтаксические свойства и тем самым может быть использован для распознавания ее в тексте и последующего извлечения.

Шаблон строится как последовательность элементов, описывающих соответствующие фрагменты языковой конструкции – в том порядке, в каком они встречаются в этой конструкции. Средства языка позволяют задавать вариативность конструкции, включая набор входящих в нее слов (лексем) и их морфологических характеристик (признаков).

Как выразительное средство, лексико-синтаксический шаблон удобен в первую очередь для формального описания именных словосочетаний русского языка, которые, как правило, являются грамматически согласованными. Для этого в языке LSPL предусмотрены средства задания синтаксической связи согласования слов.

Синтаксические связи могут задаваться в шаблоне следующим образом: согласование – путем указания равенства соответствующих морфологических признаков связанных элементов текста, управление – путем указания значений морфологических признаков подчинённого элемента.

Важной особенностью языка LSPL является возможность использовать при задании шаблона другие (уже определенные) шаблоны, что позволяет при формализации сложной языковой конструкции выделить ее составные части и описывать их по очереди в виде шаблонов, давая этим шаблонам имена и используя эти имена в других шаблонах.

В целом, язык шаблонов является достаточно гибким и мощным средством задания лексических и поверхностно-синтаксических свойств конструкций: LSPL-шаблоны могут описывать не только словосочетания, но и целые предложения и более крупные фрагменты текста на русском языке.

LSPL – продукт, свободный для некоммерческого использования.

## Проектирование алгоритма работы системы

Поскольку было решено использовать ЛСШ, то вопрос системе будет считаться входными данными для обработки ЛСШ. Каждый из вопросов неделим и каждому из них соответствует один ответ, заложенный в систему. Это допущение создано исходя из того, что все вопросы достаточно простые по структуре и содержанию. Исходя из этого, было принято решений на каждый из вопросов создать один неделимый шаблон, который бы описывал вопрос. Исходя из этого, каждый из ответов системы является неделимым и в полной мере разрешает заданный вопрос.

Между вопросами отсутствуют сложные контекстные связи, за исключением двух: отделение и сотрудник. Отделение студент для каждого вопроса указывает явно. Список сотрудников, при этом, для каждого отделения свой: в одном ответы могут давать 6 разных сотрудников, а в другом лишь трое, должности могут иметь разные названия (например, директор, начальник, менеджер). При этом считаем, что в каждом отделении каждую должность занимает только один человек.

Названия должностей сотрудников известны системе, но не известны студенту. Студенту предлагается самостоятельно понять, какие сотрудники работают в каждом отделении фирмы, исходя из описания игры и отклика системы. Исходя из этого требования, в алгоритм системы заложена возможность использования синонимов для именования одной и той же должности. Словарь синонимов заготовлен в системе заранее.

После того, как были определены отделение фирмы и должность сотрудника остается обработать вопрос, который студент задает этому сотруднику. Происходит это следующим образом: для указанного сотрудника указанного филиала берутся все ЛСШ, описывающие компетенции этого сотрудника (то есть те вопросы, на которые он может дать ответ, и сами эти ответы). Затем заданный студентом вопрос сопоставляется с этими шаблонами, после чего выбирается тот шаблон, который наиболее полно соответствует заданному вопросу. И в качестве ответа на вопрос студенту выдается ответ из этого шаблона. В случае если ни один шаблон, описывающий компетенции данного сотрудника, не соответствует заданному вопросу, то будет сказано, что сотрудник не знает ответа. Если система определила несколько шаблонов, которые в одинаковой степени соотвествуют заданному вопросу, ответ выдается случайно.

К сожалению, нет гарантии, что шаблон, который распознан, как наиболее подходящий, действительно описывает именно заданный вопрос. Но к каждому шаблону в базе привязан некоторый ответ. Именно этот ответ и будет выдан. То есть в этом случае на вопрос будет дан неверный ответ.

Разработчики языка LSPL предоставили утилиту для анализа текста на соответствие ЛСШ. А именно, утилита позволяет сопоставить заданный текст с заданным набором шаблонов и выявить тот шаблон (или те шаблоны), которые соответствуют различным частям данного текста.

Поясним сказанное примером, в котором для прояснения смысла работы утилиты предельно упрощен синтаксис ее параметров и языка LSPL. Реальный синтаксис приведен в приложении Б.

Пусть анализируемый текст равен:

«Сколько в вашем отделении фирмы установлено компьютеров?»

Сопоставляемые шаблоны представлены в табл.2.1.

Таблица 2.. Упрощенные лексико-синтаксические шаблоны

|  |  |  |
| --- | --- | --- |
| Наименование ЛСШ | Содержание ЛСШ | Ответ |
| SKF | числительное <Сколько> {шумовые слова} существительное <Фирма> {шумовые слова} существительное <Компьютер> | 32 |
| SSF | числительное <Сколько> {шумовые слова} существительное <Фирма> {шумовые слова} существительное <Сотрудник> | 32 |
| SOF | числительное <Сколько> {шумовые слова} существительное <Отделение> {шумовые слова} существительное <Фирма > | 6 |
| SO | числительное <Сколько> {шумовые слова} существительное <Отделение> | 6 |
| OF | существительное <Отделение> {шумовые слова} существительное <Фирма > | да |

Результатом работы утилиты представлены в табл. 2.2:

Таблица 2.2. Результат работы утилиты

|  |  |  |
| --- | --- | --- |
| Наименование ЛСШ | Границы текста | Фрагмент текста |
| SKF | 1-56 | Сколько в вашем отделении фирмы установлено компьютеров |
| SSF | - | - |
| SOF | 1-32 | Сколько в вашем отделении фирмы |
| SO | 1-26 | Сколько в вашем отделении |
| OF | 17-32 | отделении фирмы |

Из табл. 2.2 видно, что различные части анализируемого текста соответствуют различным шаблонам.

Наша задача выбрать шаблон, который описывает весь текст наиболее точно. Для этого был взят эвристический способ, при котором верным шаблоном является шаблон, границы текста которого наиболее широки. Исходя из этого результирующим шаблоном в нашем примере будет шаблон SKF.

В рамках процесса получения ответа на вопрос указанная утилита LSPL будет вызываться дважды: в первый раз для определения должности сотрудника, во второй – для анализа заданного вопроса.

Студентам заранее не известны должности сотрудников, работающих в данном филиале. Поэтому студент может назвать должность не так, как она может быть заложена в системе. Эту проблему решает словарь синонимов. Каждый список синонимов представлен в виде ЛСШ. Утилита получает на вход все списки синонимов (соответствующие шаблоны) всех должностей в выбранном филиале, после чего они вместе с запрашиваемой должностью подаются на вход утилите. Утилита в свою очередь возвращает шаблон, которому соответствует запрашиваемая должность. Если утилита не возвращает ничего, пользователю возвращается сообщение о том, что сотрудник с такой должностью не работает в фирме.

Когда определен сотрудник, к которому адресован вопрос, из базы данных выбираются все компетенции, которыми располагает данный сотрудник. Далее названия шаблонов всех компетенций, вместе с заданным студентом вопросом, подаются на вход утилите. Утилита возвращает шаблон той компетенции, о которая была поставлена в соответствие заданному вопросу. Если такой шаблон найден, студент получат ответ, привязанный к найденному шаблону. Если утилита не вернула ни шаблона, студенту возвращается сообщение о том, что сотрудник не знает ответа на заданный вопрос. Такой алгоритм позволяет определить студенту как факт работы сотрудника в отделении, так и компетенции этого сотрудника. Для более наглядного представления работы алгоритма, были построены IDEF0 диаграмма (Рис 2.3.) и диаграмма последовательностей (Рис 2.4.)

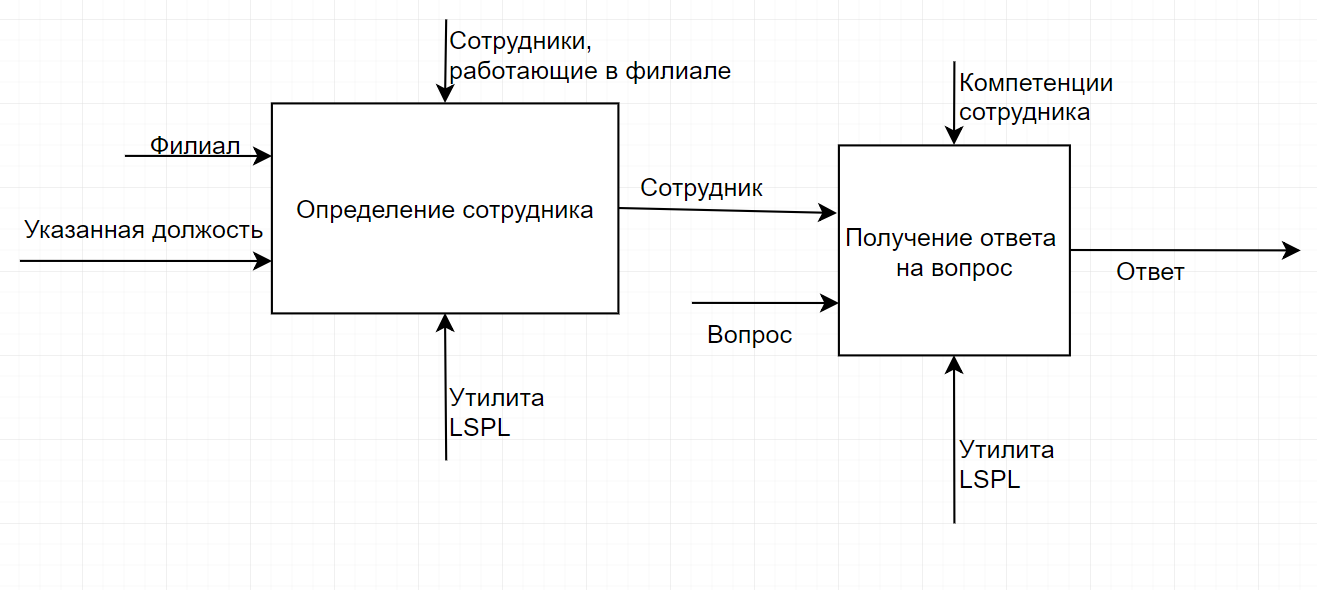


Рисунок 2.3. IDEF0 диаграмма алгоритма

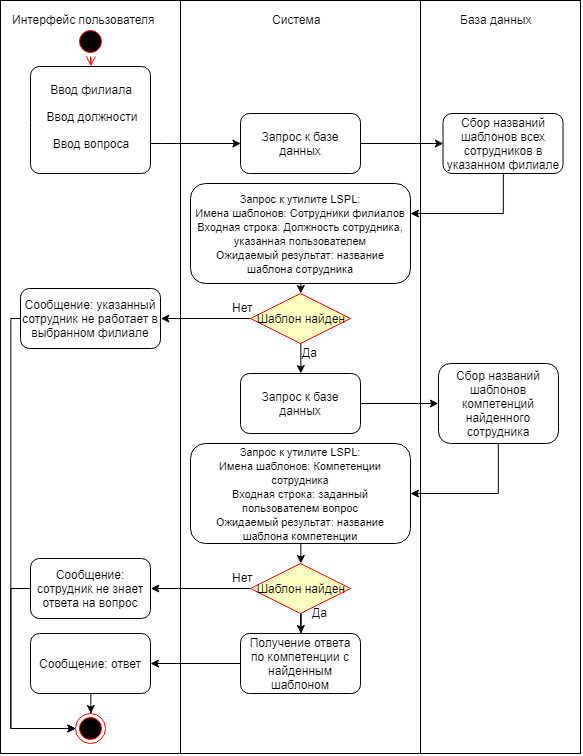


Рисунок 2.4. Диаграмма последовательностей алгоритма

## Определение бизнес-процессов

Определим основные процессы, в которых будет задействована система. Поскольку предполагается, что студенты в начале игры знают только названия филиалов и не имеют список сотрудников, то студенты должны сами их определить. Было решено, что студенты будут определять, существует ли такой сотрудник в филиале в ходе постановки вопросов виртуальному собеседнику. Таким образом, для студентов виртуальный собеседник задействован в двух процессах: определение существования сотрудника в филиале и постановка самого вопроса виртуальному собеседнику относительно филиалов фирмы «ОбалдеИТ».

Перейдем к диаграмме прецедентов. Всего в системе будет два актора: студент, задающий вопросы собеседнику и администратор, поддерживающий систему. Определим прецеденты системы, после чего построим диаграмму, которая представлена на Рис. 2.1.

1. Название: Постановка вопроса системе

Акторы: Студент

Описание: Студент заходит в систему под своим именем и паролем, указывает сотрудника и задает вопрос системе.

1. Название: Определение существования сотрудника в филиале

Акторы: Студент

Описание: Студент заходит в систему под своим именем и паролем, указывает сотрудника и задает вопрос системе. Если такой сотрудник существует, он даст ответ на вопрос, либо ответит, что это не в его компетенции. Если такого сотрудника нет, система сообщит об этом.

1. Название: Редактирование компетенций сотрудников филиала

Акторы: Администратор

Описание: Администратор заходит в интерфейс администрирования системой, выбирает филиал, затем одного из сотрудников этого филиала. Перед администратором открывается список всех компетенций данного филиала, в котором уже указаны те, компетенции, которыми располагает выбранный сотрудник. Администратор может удалить существующие компетенции или добавить новые из списка.

1. Название: Просмотр заданных вопросов системе.

Акторы: Администратор

Описание: Администратор заходит в интерфейс администрирования, затем открывает окно заданных вопросов системе. Там он может увидеть информацию, какому сотруднику, в каком филиале и в какое время был задан вопрос, и какой ответ на него был дан.

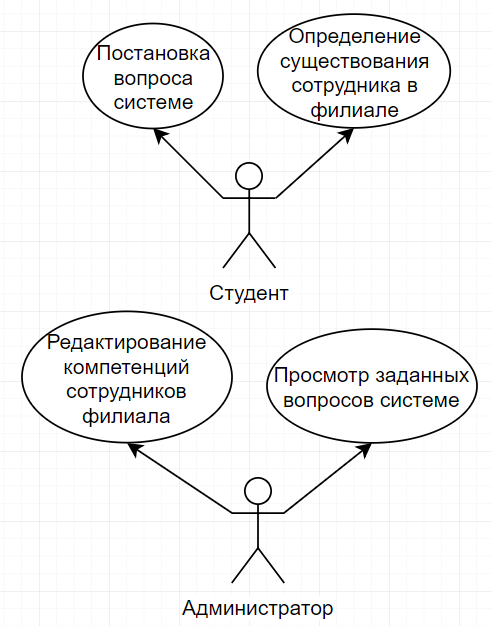


Рисунок 2.1. Диаграмма активностей

Теперь, когда были определены прецеденты системы, была построена диаграмма активностей. На рисунке 2.2 представлена диаграмма последовательностей обращения к системе. На диаграмме обращения к системе показаны все возможные варианты реакции системы на действия пользователя: если пользователь указал сотрудника фирмы, который не работает в ней, если пользователь задал вопрос сотруднику, на который он не знает ответа или если пользователь не был идентифицирован системой в момент обращения к ней.

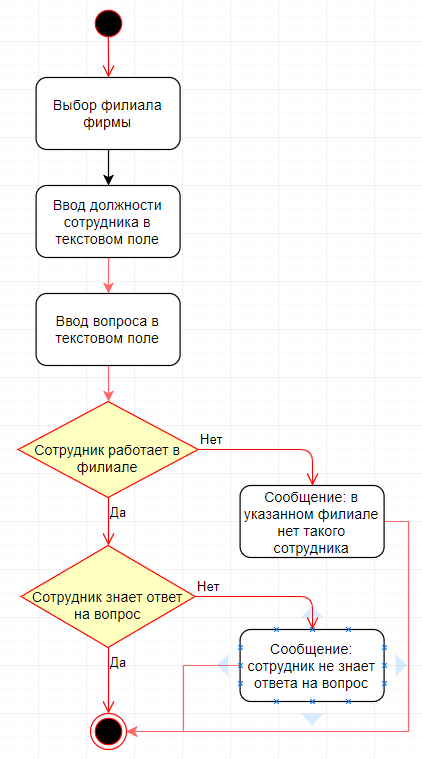


Рисунок 2.2. Диаграмма последовательностей

## Проектирование системы

Теперь, когда были определены процессы, которые должна автоматизировать система, и была выбрана реализация, было начато проектирование приложения. Были определены сущности, которые будут в системе:

1. Филиал.

Свойства:

А) Имя. Тип - строковый

Описание: У каждого филиала свои уникальные характеристики такие как месторасположение, количество и комплектация ЭВМ, количество сотрудников и их квалификация.

1. Сотрудник.

Свойства:

А) Имя. Тип строковый.

Б) Название шаблона. Тип строковый.

Описание: Каждый из сотрудников филиалов фирмы «ОбалдеИТ» обладает своим набором компетенций. В зависимости от этих компетенций определяется, сможет ли сотрудник ответить на поставленный вопрос или нет. Поскольку ранее было решено, что система будет основана на лексико-синтаксических шаблонах, то у сотрудника было определено свойство «Название шаблона», в котором будет содержаться название того шаблона, который бы однозначно определял сотрудника.

1. Компетенция.

Свойства:

А) Название шаблона. Тип – строковый.

Б) Текст ответа. Тип – строковый.

В) Должность сотрудника

Описание: Небольшая область знаний, которой обладает сотрудник, прикрепленный к филиалу.

1. Запись о вопросе.

Свойства:

А) Время. Тип – DateTime

Б) Заданный вопрос. Тип – строковый

В) Имя сотрудника. Тип – строковый

Г) Ответ на вопрос. Тип – строковый

Описание: Поскольку было решено собирать данные о заданных вопросах, была введена сущность, которая бы отображала эти данные. Для этого она хранит точную дату и время заданного вопроса, точную формулировку самого вопроса, а также название сотрудника, которое было введено в текстовом поле. Эти данные помогут в дальнейшем пополнять и корректировать существующие шаблоны.

Диаграмма ERD представлена на Рис 2.5

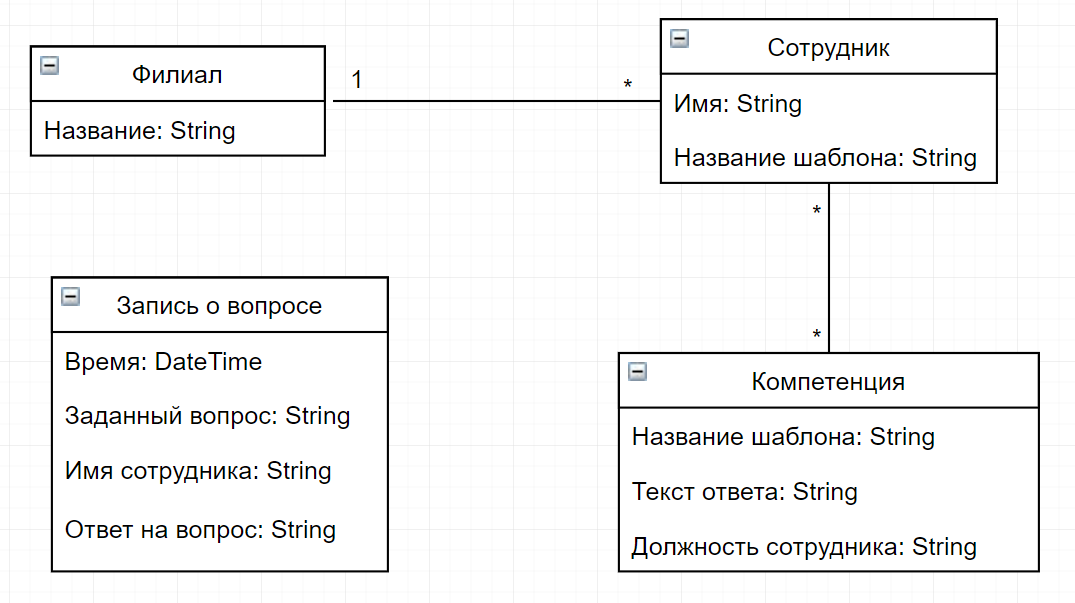


Рисунок 2.5. ERD диаграмма

Ранее говорилось, что компетенции сотрудников филиалов могут пересекаться, а значит, ответ на один и тот же вопрос могут дать несколько сотрудников сразу. Поэтому было решено сделать связь типа «многие ко многим» между сущностями «Сотрудник» и «Компетенция».

# Глава 3. Разработка системы

В этой главе будет подробно описан процесс разработки приложения, его базы данных и взаимодействие с утилитой для сопоставления текста с лексико-синтаксическими шаблонами.



## Инструментальные средства разработки

Определим стек технологий для разработки веб-приложения виртуального собеседника. Было решено разрабатывать приложение на платформе .Net с использованием фреймворка Asp.Net для разработки сайта. Основным языком разработки является C#, однако помимо этого используются языки: HTML для описания веб-страниц, XML для задания конфигурации приложения и LSPL для задания лексико-синтаксических шаблонов.

Далее будут описаны основные инструменты, которые потребуются при разработке собеседника:

Яндекс словарь. Как ранее было сказано, в данной статье описан способ реализации собеседника, который не требует большого объема входной выборки. Однако исходная выборка для более широкой работы системы искусственно увеличивается за счет словаря синонимов. Однако, этот словарь нельзя использовать непосредственно во время использования, поскольку предоставляемое API имеет ежедневное ограничение на количество запросов.

LSPL. LSPL (Lexico-Syntaxic Pattern Language) – язык, предназначенный для формального описания конструкций русского языка с целью их представления в системах извлечения информации из текстов. К сожалению разработчики не предоставили dll файл для использования его в системе, таким образом инструмент используется через утилиту с помощью запуска bat файла. Стоит отметить, что для этого компонента нет актуальной документации, поэтому учиться его использовать придется в процессе разработки.

LemmaSharp. LSPL шаблоны требуют на вход слова в начальной форме. Для решения этого этапа потребовалась отдельная библиотека.

MS SQL. Как было написано выше, Яндекс словарь имеет ограничение на количество ежедневных запросов. Для того, чтобы во время работы не обращаться к словарю за синонимами, было решено получить их ранее и хранить их в базе данных.

## Разработка базы данных

Для хранения данных для приложения было решено создать базу данных на основе SQL Server с названием ObaldeitChatBot. Под каждую из сущностей, описанных в предыдущей главе, была создана таблица в базе данных, а также дополнительные поля, которые служат уникальными идентификаторами и внешними ключами для других таблиц. Схема базы данных представлена на Рис 3.1.

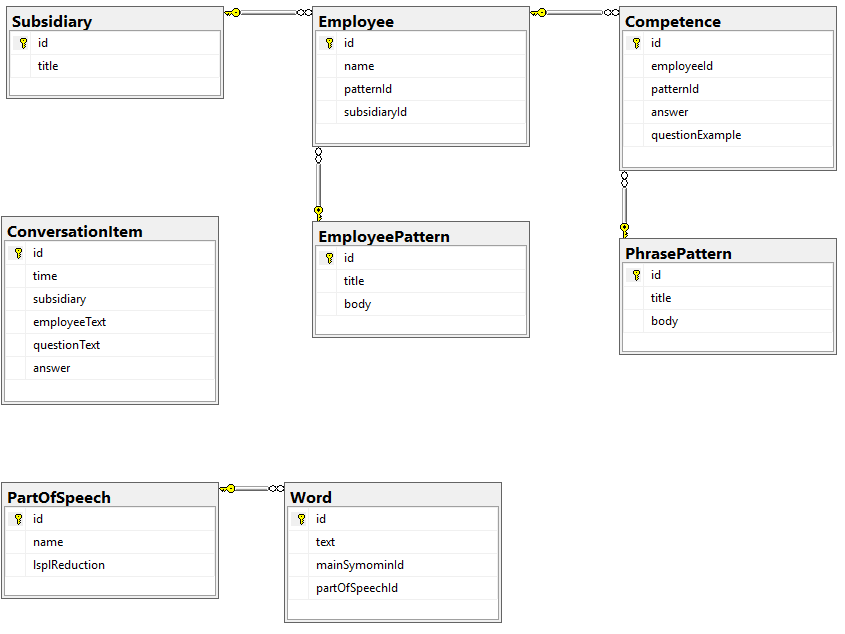


Рисунок 3.1. Схема базы данных

## Подготовка файла с входными данными

Для корректной работы системы потребуется заполнить базу данных. С предыдущих игр, где за каждого сотрудника отвечал преподаватель, сохранились файлы, содержащие вопросы студентом и ответы каждого сотрудника. Информация из этих файлов была объединена и структуризирована к виду: название филиала, должность сотрудника, вопрос, ответ.

Стоит отметить, что от игры к игре в некоторых филиалах менялись сотрудники или точная формулировка их должностей, поэтому перед исправлением объединенного файла была создана матрица принадлежностей сотрудников к филиалам (Рис 3.2.). Исходя из созданной матрицы, наименования должностей сотрудников филиалов приводилось к единому виду. Помимо наименований должностей сотрудников, менялись и их компетенции, поэтому для организации зон ответственности сотрудников была создана матрица зон ответственности сотрудников (Рис 3.3.).

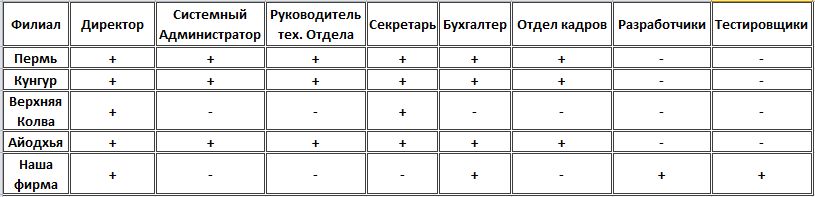


Рисунок 3.2.Матрица принадлежностей сотрудников к филиалам

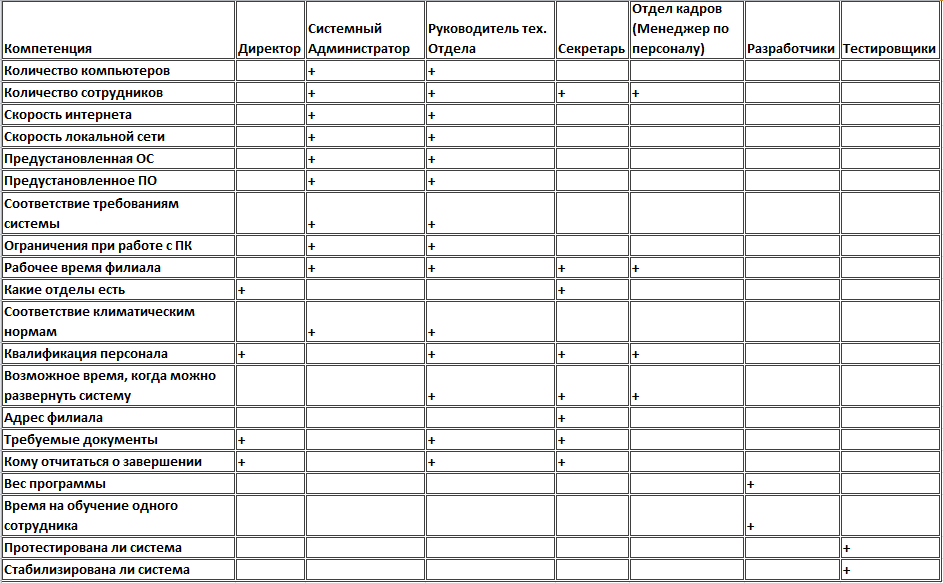


Рисунок 3.3. Матрица зон ответственностей сотрудников

После того, как были определены зоны ответственности и наименования должностей сотрудников, было решено написать небольшую подпрограмму, проверяющую корректность сформированного файла. Эта программа проверяет, верно ли введено название филиала, наименование должности сотрудника, а также прикреплен ли сотрудник с указанной должностью к филиалу. Если какое-то из условий не совпадает, программа выдает номер строки с ошибкой. После нескольких проверок файла и исправления ошибок, файл был приведен в вид, пригодный для наполнения базы данных. Код валидации файла приведен ниже:

private void ValidateFile(object sender, RoutedEventArgs e)

{

var filePath = GetExcelFilePath();

if (filePath == null)

return;

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(filePath);

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

var errorsList = new List<int>();

using (var db = new ObaldeitChatBotEntities())

{

for (int i = 2; i < xlRange.Rows.Count; i++)

{

string subsidiary = xlRange.Cells[i, 1].Value2.ToString();

string employee = xlRange.Cells[i, 2].Value2.ToString();

string question = xlRange.Cells[i, 2].Value2.ToString();

if (!db.Subsidiary.Select(s => s.title).Contains(subsidiary))

{

Console.WriteLine("Строка " + i + " содержит невалидное название филиала");

}

if (!db.Employee.Select(s => s.name).Contains(employee))

{

Console.WriteLine("Строка " + i + " содержит невалидное название сотрудника");

}

if (question.Contains("\n"))

{

Console.WriteLine("Строка содержит непечатные символы: " + i);

}

if (!db.Employee.Include("Subsidiary").ToList().Any(emp => emp.name == employee && emp.Subsidiary.title == subsidiary))

{

Console.WriteLine(i + " – Сотрудник не работает в филиале");

}

}

}

xlWorkbook.Close();

xlApp.Quit();

}

## Наполнение базы данных

После того, как база данных была спроектирована, следует её наполнить из созданного ранее файла. Для этого потребовалась дополнительная подпрограмма, поскольку объем работ слишком велик для ручного выполнения. Поэтому было решено создать несколько подпрограмм для наполнения базы данных и генерации шаблонов.

Процесс наполнения базы данных был разбит на несколько этапов. На первом этапе вручную в базе данных было наполнено несколько таблиц, данные из которых будут нужны для следующих этапов. Для проверки входного файла и дальнейшего заполнения таблицы с компетенциями были заполнены следующие таблицы: Subsidiary (филиал), Employee (Сотрудник), EmployeePattern (Шаблоны сотрудника). Первые две таблицы были заполнены исходя из матрицы принадлежностей сотрудников к филиалам. Шаблоны должностей созданы, используя словари синонимов. Помимо этого, была заполнена таблицы PartOfSpeech (Часть речи), которая понадобится для получения синонимов каждого слова для вопроса. Данные для этой таблицы были взяты из документации по языку LSPL (Рис 3.4.). Там указаны все поддерживаемые части речи и их перевод на английский язык. API словаря синонимов к каждому из слов возвращает его часть речи, что поможет после для составления шаблонов, используя эту таблицу.

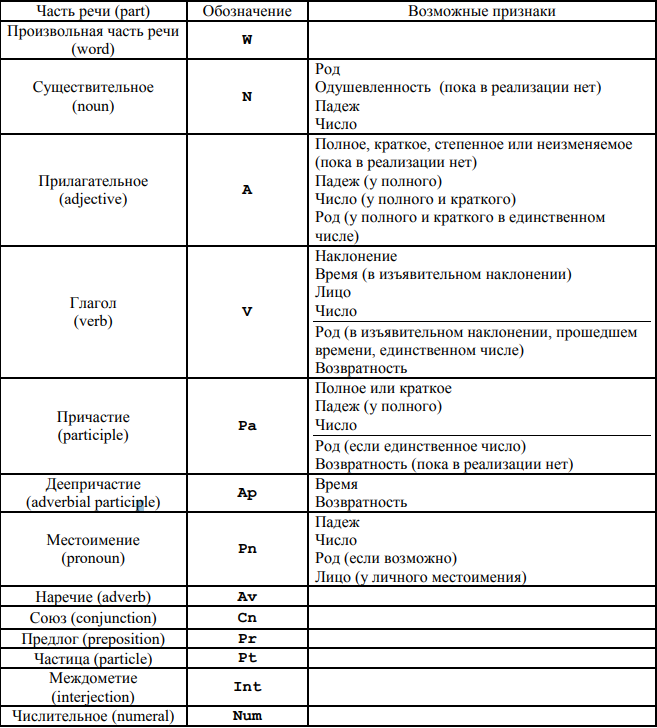


Рисунок 3.4. Части речи, поддерживаемые языком LSPL

Следующим этапом было заполнение таблицы с синонимами слов задаваемых вопросов. Для получения синонимов был использован словарь синонимов компании Яндекс. Ранее было задумано получать синонимы во время работы программы и генерировать шаблоны динамически. Однако Яндекс словарь ограничивает количество запросов в день, а на все запросы, которые посыпаются поверх допустимого лимита, словарь возвращает сообщение об ошибке. Поэтому было решено заранее сгенерировать синонимы и шаблоны. Словарь возвращает на запрос структуру, в которой указан синоним и его часть речи. Полученный синоним добавляется в базу данных в случае, если ранее его там не было. Код заполнения таблицы с синонимами представлен ниже.

private void GenerateWordsPool(object sender, RoutedEventArgs e)

{

var filePath = GetExcelFilePath();

if (filePath == null)

return;

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(filePath);

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

Dictionary<string, PhrasePattern> questionPattern = new Dictionary<string, PhrasePattern>();

using (var db = new ObaldeitChatBotEntities())

{

for (int i = 2; i < xlRange.Rows.Count; i++)

{

string question = xlRange.Cells[i, 3].Value2.ToString();

var words = PrepareInputString(question);

foreach (var word in words)

{

if (word.Trim() == string.Empty)

{

continue;

}

var lemmatizedWord = LemmatizerSingletone.GetInstance().LemmatizePhrase(word);

if (db.Word.Any(w => w.text == lemmatizedWord))

{

continue;

}

WordPattern wordPattern = new WordPattern(word);

if (wordPattern.Alternatives.Count == 0)

{

continue;

}

if (!db.PartOfSpeech.ToList().Any(p => p.name == wordPattern.Alternatives[0].pos))

{

continue;

}

Word dbWord = new Word() { text = lemmatizedWord, partOfSpeechId = db.PartOfSpeech.ToList().FirstOrDefault(pos => pos.name == wordPattern.Alternatives[0].pos).id, mainSymominId = -1};

db.Word.Add(dbWord);

db.SaveChanges();

foreach (var alt in wordPattern.Alternatives.Skip(1))

{

if (!db.PartOfSpeech.ToList().Any(p => p.name == alt.pos))

{

continue;

}

if (alt.pos != null)

{

db.Word.Add(new Word() { text = alt.text, partOfSpeechId = db.PartOfSpeech.ToList().FirstOrDefault(pos => pos.name == alt.pos).id, mainSymominId = dbWord.id });

}

}

}

}

db.SaveChanges();

}

xlWorkbook.Close();

xlApp.Quit();

}

После заполнения таблицы синонимов были сгенерированы шаблоны. Язык LSPL позволяет использовать альтернативы в шаблонах, что позволяет использовать в шаблонах ранее полученные синонимы. Помимо этого, LSPL дает возможность работать со словом как частью речи языка, используя такие его характеристики как число, падеж, время и т.д. Части речи в предложении играют разную роль и имеют разную значимость. Исходя из этого, была составлена маска частей речи, по которым будут составляться шаблоны. В маску вошли следующие части речи: существительное, наречие, глагол, числительное.

Далее кратко будет описан алгоритм составления шаблона. На вход подается ранее созданный файл с вопросами и ответами. Файл считывается построчно. Сначала определяется филиал, сотруднику, которого задается вопрос, затем определяется сам сотрудник из базы данных по его принадлежности к филиалу. Затем из заданного вопроса удаляются все слова, которые не соответствуют маске частей речи, которая была описана ранее. Затем для каждого из оставшихся слов генерируется лексико-синтаксический шаблон с его синонимами, после чего все полученные шаблоны синонимов объединяются в шаблон заданного вопроса в исходном порядке. Шаблоны синонимов и полученного вопроса по необходимости добавляются в базу данных. Код, добавляющий в базу данных шаблоны приведен ниже.

private void LoadPatterns\_Click(object sender, RoutedEventArgs e)

{

var filePath = GetExcelFilePath();

if (filePath == null)

return;

Excel.Application xlApp = new Excel.Application();

Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(filePath);

Excel.\_Worksheet xlWorksheet = xlWorkbook.Sheets[1];

Excel.Range xlRange = xlWorksheet.UsedRange;

Dictionary<string, PhrasePattern> questionPattern = new Dictionary<string, PhrasePattern>();

using (var db = new ObaldeitChatBotEntities())

{

for (int i = 2; i < xlRange.Rows.Count; i++)

{

string subsidiaryName = xlRange.Cells[i, 1].Value2.ToString();

string employeeName = xlRange.Cells[i, 2].Value2.ToString();

string question = xlRange.Cells[i, 3].Value2.ToString().Replace('\n', ' ');

string answerValue = xlRange.Cells[i, 4].Value2.ToString().Replace('\n', ' ');

Employee employee = db.Employee.Include("Subsidiary").FirstOrDefault(emp => emp.name.ToLower() == employeeName.ToLower() && emp.Subsidiary.title.ToLower() == subsidiaryName.ToLower());

PhrasePattern pattern = new PhrasePattern();

PartOfSpeechMask mask = new PartOfSpeechMask(new List<PatternsGenerator.PartOfSpeech>() { }, new List<PatternsGenerator.PartOfSpeech> { PatternsGenerator.PartOfSpeech.Noun, PatternsGenerator.PartOfSpeech.Verb, PatternsGenerator.PartOfSpeech.Pronoun, PatternsGenerator.PartOfSpeech.Numeral });

var wordsStrings = PrepareInputString(question).Select(s => LemmatizerSingletone.GetInstance().LemmatizePhrase(s)).ToList();

for (int j = 0; j < wordsStrings.Count; j++)

{

var word = db.Word.Include("PartOfSpeech").ToList().FirstOrDefault(w => w.text == wordsStrings[j]);

if (word == null || !mask.NonRequired.Select(s => PartOfSpeechMask.PartOfSpeechDict[s].ToLower()).Contains(word.PartOfSpeech.name.ToLower()))

{

wordsStrings.RemoveAt(j);

j--;

}

}

if (wordsStrings.Count == 0) continue;

List<PhrasePattern> wordPatternsList = new List<PhrasePattern>();

for (int j = 0; j < wordsStrings.Count; j++)

{

var word = db.Word.Include("PartOfSpeech").ToList().First(w => w.text == wordsStrings[j]);

if (word.mainSymominId != -1)

{

word = db.Word.Include("PartOfSpeech").ToList().First(w => w.id == word.mainSymominId);

}

var wordPattern = db.PhrasePattern.ToList().FirstOrDefault(p => p.title.ToLower() == TranslitGenerator.GetTranslit(word.text).ToLower());

if (wordPattern == null)

{

var dbAlternates = db.Word.Include("PartOfSpeech").ToList().Where(w => w.mainSymominId == word.id).Take(10).Select(w => new ParserWord() { text = w.text, pos = w.PartOfSpeech.name}).ToList();

WordPattern parserWordPattern = new WordPattern(new ParserWord() { text = word.text, pos = word.PartOfSpeech.name }, dbAlternates);

var generatedPattern = parserWordPattern.GeneratePatternSeparately(mask);

if (!String.IsNullOrEmpty(generatedPattern[1]))

{

wordPattern = new PhrasePattern() { title = generatedPattern[0], body = generatedPattern[1] };

db.PhrasePattern.Add(wordPattern);

db.SaveChanges();

}

}

if (wordPattern != null)

{

wordPatternsList.Add(wordPattern);

}

}

var patternTitle = new string(wordPatternsList.Select(p => p.title[0]).ToArray());

pattern.title = Char.ToUpper(patternTitle[0]) + patternTitle.Substring(1);

pattern.body = wordPatternsList.ToList().Select(s => s.title).Aggregate((m, k) => m + " {W} " + k) + " =pattern> \'-\'";

db.PhrasePattern.Add(pattern);

Competence competence = new Competence() { employeeId = employee.id, patternId = pattern.id, answer = answerValue};

db.Competence.Add(competence);

db.SaveChanges();

}

}

xlWorkbook.Close();

xlApp.Quit();

}

После того, как все шаблоны были добавлены в базу данных, осталось только сгенерировать файл с паттернами, который будет подаваться на вход утилите. Шаблоны, добавленные в файл приведены в приложении В.

private void GeneratePatternFile\_Click(object sender, RoutedEventArgs e)

{

List<string> patterns = new List<string>();

using (var db = new ObaldeitChatBotEntities())

{

foreach (var ptrn in db.PhrasePattern.ToList())

{

patterns.Add(char.ToUpper(ptrn.title[0]) + ptrn.title.Substring(1) + " = " + ptrn.body);

}

}

patterns = patterns.Distinct().ToList();

foreach (var pattern in patterns)

{

File.AppendAllLines(@"C:\projects\ObaldeitChatBotV2\PatternsGenerator\lspl-tools-new\example\ptrns1.txt", new List<string> { pattern }, Encoding.GetEncoding("windows-1251"));

}

}

## Использование лексико-синтаксических шаблонов

Разберем взаимодействие системы с лексико-синтаксическими шаблонами. Каждый ответ, который может давать система, связан с лексико-синтаксическим шаблоном в его перечне. Все лексико-синтаксические шаблоны находятся в файле, относительный путь которого ObaldeitChatBot\LsplUtils\lspl-tools-new\bin\patterns.txt. Сами шаблоны составлены так, что позволяют учитывать разные формулировки вопросов и должностей сотрудников. Достигнуто это за счет возможностей языка lspl. Рассмотрим некоторые шаблоны. Ниже представлен шаблон, определяющий должность Системного администратора:

Admin = N<администратор> =text> '-' | A<системный> N<администратор> =text> '-' | 'сисадмин' =text> '-'

Разберем подробнее его составляющие. В шаблоне представлены три альтернативы. Первая альтернатива «N<администратор> =text> '-'» предполагает, что исходный текст будет соотнесен с этим шаблоном, если он будет содержать существительное «администратор» в любом числе и падеже. Вторая альтернатива «A<системный> N<администратор> =text> '-'» предполагает, что исходный текст будет соотнесен с должностью, если в тексте встретится словосочетание, состоящее из прилагательного «системный» и существительного «администратор», которые могут быть не согласованы между собой. Так, такие варианты как «системный администратору» или «системному администратор» также будет распознаны этим шаблоном. Третья альтернатива не завязана на каких-либо частях речи и соотнесет с шаблоном любой текст, в котором встречается подстрока «сисадмин». Кроме альтернатив в простых лексико-синтаксических шаблонах, многовариантности можно достичь за счет использования составных шаблонов, которые состоят из других шаблонов. Разберем это на примере шаблона, который соотнесен с ответом на вопрос о том, сколько в филиале находится компьютеров. Этот шаблон представлен ниже:

QComp = Quantity {W} Computer {W} [Office] =pattern> '-'

Как видно, в этом шаблоне нет элементов, которые бы характеризовали слова на русском языке, однако он содержит название других трех шаблонов: «Quantity», «Computer» и «Office». Эти шаблоны представлены ниже:

Quantity = Num<сколько> =text> '-' | N<количество> =text> '-' | N<число> =text> '-' | N<величина> =text> '-'

Computer = N<компьютер> =text> '-' | N<эвм> =text> '-' | N<персоналка> =text> '-' | 'пк' =text> '-'

Office = N<офис> =text> '-' | N<отделение> =text> '-' | N<подразделение> =text> '-'| N<отдел>=text> '-'

Первый шаблон содержит пять альтернатив, второй четыре и третий пять альтернатив. Помимо этого, в шаблоне QComp шаблон Office указан в квадратных скобках, что означает, что он может не находиться в исходном тексте. Также между шаблонами указана буква W в фигурных скобках, что обозначает, что между шаблонами может находиться произвольное количество любых слов.

Разберем подробнее процесс запуска самой утилиты. Ее можно запустить командой из командной строки, указав относительный путь входного файла, лексико-синтаксических шаблонов, выходной файл и искомые шаблоны. Пример команды приведен ниже:

bin\lspl-find -i bin\text.txt -p bin\patterns.txt -o bin\output.txt QComp InternetSpeed AreUsable AreSuitable LevelEmployee Characteristic VersOS WillChange LanguageProgram Cabinets AreServersHere IsLocalNetworkHere ThemselfInstall

Однако, поскольку из приложения мы не имеем возможности воспользоваться командной строкой, команда выполняется через .bat файл. Всего по мере ответа на вопрос команда выполняется дважды: в первый раз устанавливается, к какому именно сотруднику задается вопрос. Далее, если такой сотрудник работает в указанном филиале, устанавливается, что за вопрос был задан.

Разберем обработку результатов работы утилиты. Lspl-tools по результатам работы генерирует xml файл, в котором указано, какие именно шаблоны были встречены в тексте. Пример такого xml файла представлен ниже, где в качестве входного текста был указан вопрос «Сколько компьютеров есть в вашей фирме», а в качестве лексико-синтаксических шаблонов были указаны QComp, InternetSpeed, AreUsable, AreSuitable, LevelEmployee, Characteristic, VersOS, WillChange, LanguageProgram, Cabinets, AreServersHere, IsLocalNetworkHere, ThemselfInstall:

<?xml version="1.0" encoding="windows-1251"?>

<texts>

<text>

<goal name="QComp">

<match startPos="0" endPos="19">

<fragment>Сколько компьютеров</fragment>

<result>-</result>

</match>

<match startPos="0" endPos="24">

<fragment>Сколько компьютеров есть</fragment>

<result>-</result>

</match>

<match startPos="0" endPos="26">

<fragment>Сколько компьютеров есть в</fragment>

<result>-</result>

</match>

<match startPos="0" endPos="32">

<fragment>Сколько компьютеров есть в вашей</fragment>

<result>-</result>

</match>

<match startPos="0" endPos="38">

<fragment>Сколько компьютеров есть в вашей фирме</fragment>

<result>-</result>

</match>

</goal>

<goal name="InternetSpeed">

</goal>

<goal name="AreUsable">

</goal>

И т.д.

</text>

</texts>

Как видно из кода файла, было найдено несколько соответствий с шаблоном «QComp». Поскольку в этом шаблоне есть возможность вставлять слова между частями шаблона, то было количество соответствий было увеличено. После завершения работы утилиты содержимое выходного файла обрабатывается системой, где проверяется, какие из поданных шаблонов были встречены в тексте. Результирующим шаблоном считается тот, фрагмент которого, встреченный в тексте, имеет максимальную длину. За обработку выходного файла отвечает класс QuestionsAnswerer.

Разберем, как работает этот класс. У него есть всего одна публичная функция, которая вызывается из внешнего кода. В нее передается текст выходного файла. Сначала функция проверяет, находится ли в коде файла подстрока «match»: эта подстрока находится в файле только в том случае, если было найдено соответствие между лексико-синтаксическими шаблонами и текстом на вход для утилиты. Если подстрока не была найдена, то функция ничего не вернет и это будет обрабатываться во внешнем коде, в ином случае выбирается то соответствие, в котором разница стартовой и конечной позиции максимальна, после чего возвращается название шаблона этого соответствия.

## Разработка веб-приложения

После того, как была спроектирована и наполнена база данных, была начата разработка веб-приложения для взаимодействия с пользователем на платформе asp.net. Было разработано одностраничное приложение с использование технологий jquery и ajax с простым интерфейсом (Рис 3.5.), где пользователю предлагается выбрать филиал из списка доступных, ввести должность сотрудника и задать вопрос, после чего в графе ответ через некоторое время появится ответ, вычисленный системой. Алгоритм, по которому работает система, в точности повторяет спроектированный. Его реализация приведена ниже:

public JsonResult GetAnswer(QuestionAnswerModel model)

{

using (var db = new ObaldeitChatBotEntities())

{

var employeePatternName = GetPatternName(model.EmployeeText, db.Employee.ToList().Where(e => e.subsidiaryId == model.SelectedCityId).Select(e => e.EmployeePattern.title).ToList());

if (employeePatternName == null)

{

model.AnswerText = "Нет у нас такой должности!";

return Json(model);

}

var employee = db.Employee.ToList().First(e => e.subsidiaryId == model.SelectedCityId && e.EmployeePattern.title == employeePatternName);

var competencesPatterns = db.Competence.ToList().Where(c => c.employeeId == employee.id).Select(s => s.PhrasePattern.title).Distinct().ToList();

var answerPatternName = GetPatternName(model.Question, competencesPatterns);

if (answerPatternName == null)

{

model.AnswerText = "Сотрудник не знает ответа на заданный вопрос.";

return Json(model);

}

model.AnswerText = db.Competence.ToList().First(c => c.employeeId == employee.id && c.PhrasePattern.title == answerPatternName).answer;

}

return Json(model);

}

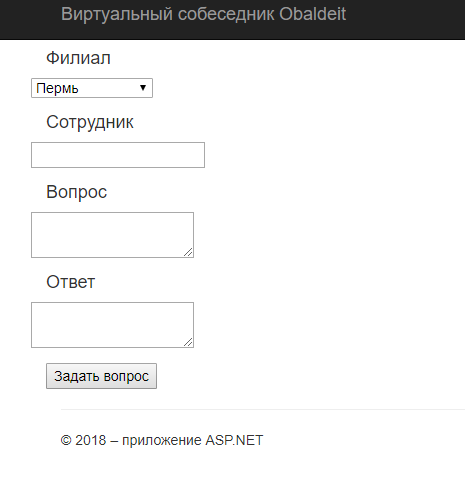


Рисунок 3.5. Веб-приложение LSPL

Эта часть исходного кода – обработчик события нажатия на кнопку. Ему на вход приходит собранная на клиенте модель, содержащая все поля, которые пользователь заполняет в интерфейсе. Далее начинается процесс обработки, описанный в предыдущей главе: сначала происходит проверка наличия сотрудника с указанной должностью в выбранном филиале – если такого нет, то пользователю выдается уведомление об этом. После этого выбираются все компетенции найденного сотрудника и их шаблоны сопоставляются с заданным вопросом.

Процесс проверки шаблона происходит с помощью запуска процесса системы Windows. Сначала формируется файл с обрабатываемой строкой, где указана либо должность сотрудника, либо вопрос. Затем формируется bat файл, в котором указывается входной файл, файл с заранее сгенерированными шаблонами и путь к выходному файлу. Затем запускается сама утилита через ранее сформированный bat файл. После завершения процесса обрабатывается выходной файл, сгенерированный утилитой, а шаблон, определенный как наиболее подходящий из предложенных выходным файлом, сопоставляется в зависимости от ситуации с шаблонами в нужной таблице базы данных, на основании чего системой принимается решение. Полный исходный код этой работы представлен в приложении Г.

## Разработка сервиса администрирования

После разработки веб-приложения виртуального собеседника была разработана система администрирования виртуальным собеседником. Всего система администрирования содержит 2 окна: изменение компетенций сотрудников и просмотр истории вопросов студентов.

В первом окне (Рис 3.6.) администратору предлагается выбрать филиал и должность сотрудника, после чего ему открывается список всех компетенций выбранного филиала, где галочками отмечены те компетенции, о которых знает выбранный сотрудник. Администратор может изменить набор компетенций того или иного сотрудника по своему усмотрению, после изменения каждой компетенции данные перезаписываются в базу данных.

Второе окно (Рис 3.7.) отображает информацию обо всех заданных вопросах за все время работы системы, а именно: дата и время, когда был задан вопрос, филиал, сотруднику которого он был задан, формулировка должности сотрудника, указанная пользователем, заданный вопрос и данный системой ответ.

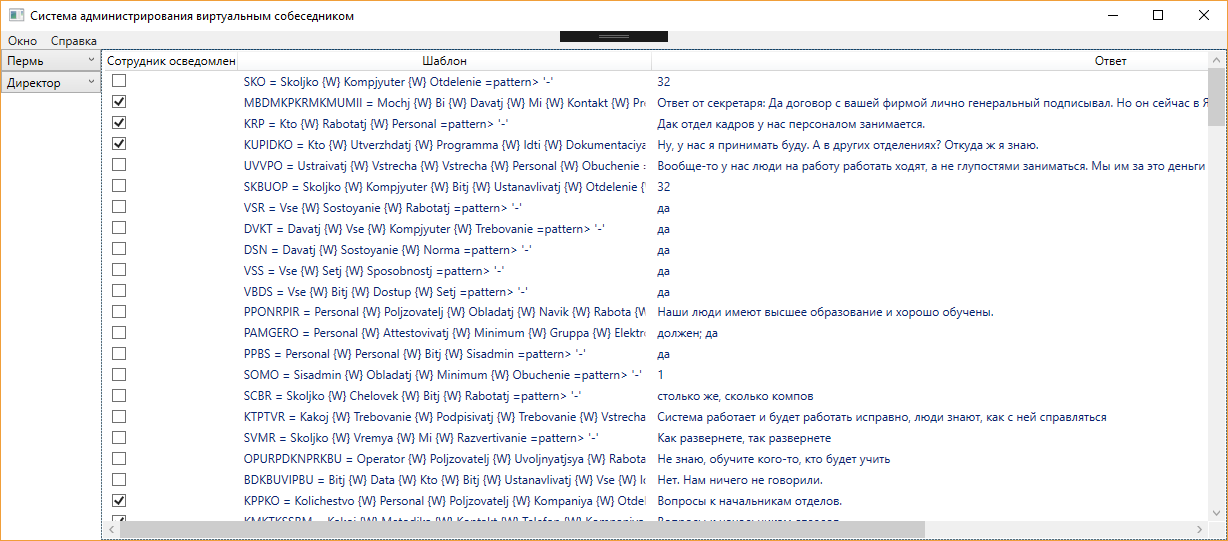


Рисунок 3.6. Исправление компетенций сотрудников

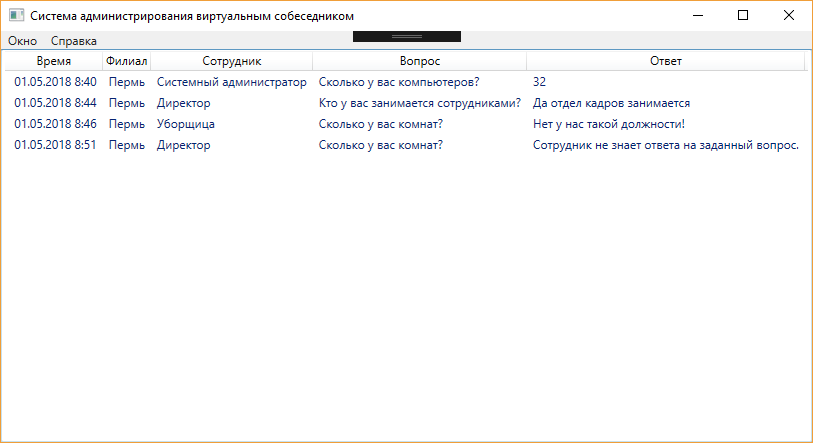


Рисунок 3.7. Окно просмотра истории заданных вопросов

## Публикация системы на Microsoft Azure и опытное тестирование

Когда система была разработана и запущена локально, предстоял процесс развертки на хостинге. В качестве хостинга был выбран Microsoft Azure.

Поскольку система использует базу данных в работе, то сначала была развернута она. В студенческом аккаунте, предоставляемом компанией Microsoft для студентов НИУ ВШЭ, был создан запрос на использование сервера базы данных. После получения доступа к серверу, была создана пустая база данных. Далее с помощью программы Data Migration Assistant была перенесена схема локальной БД на удалённую, после чего были переданы и сами данные.

Далее была создана виртуальная машина. После того, как виртуальная машина была создана, приложение было перенесено на нее. Однако для работы приложения требуется утилита LSPL и файлы с паттернами. Для переноса папки с утилитой виртуальной машине был предоставлен доступ к локальному жесткому диску, с которого папка была перенесена на виртуальную машину. После того, как система получила доступ к папке, система всё ещё не отвечала на поставленные вопросы. Для решения этой проблемы было решено произвести отладку путем вывода сообщений об ошибке в окно ответа. Основной проблемой оказалась политика безопасности виртуальной машины, которая не разрешала редактировать файлы в процессе работы. После того, как политика безопасности была настроена, система заработала.

После публикации система прошла опытное тестирование в ходе деловой игры. Студентами было задано более 1700 вопросов системе. В общем студентам удалось получить ответы на большинство заложенных в систему вопросов. Однако, учитывая то, что студенты работали несогласованно, то для каждого из них информации было мало. Было определено, что разработанный собеседник не имеет достаточную точность ответа, однако при наличии в вопросе требуемых слов или их синонимов способен дать ответ. Для дальнейшего улучшения работы собеседника предлагается доработать систему администрирования так, чтобы с её помощью была возможность создавать шаблоны и проверять входной текст на их наличие.

# Заключение

В ходе данной работы была построена собственная система для генерации лексико-синтаксических шаблонов на основе исходного файла с текстовыми строками. На основе разработанной технологии была спроектирована и реализована система, позволяющая имитировать процесс общения студента и сотрудника филиала фирмы в ходе бизнес-игры. Разработанная система была развернута в интранете ВШЭ Пермь для проведения последующих бизнес-игр. Были выполнены следующие задачи:

1. Выявлены причины провала прошлой версии виртуального собеседника.
2. Разработано технического задания
3. Проведен сравнительный анализ моделей улучшения алгоритма
4. Был улучшен алгоритм работы системы
5. Было произведено проектирование архитектуры новой системы.
6. Система была разработана.
7. Было произведено первичное тестирование системы.
8. Система была развернута.
9. Была проведена опытная эксплуатация системы.

# Библиографический список

1. Chapter 1. Introduction to Microsoft Solutions Framework. / TechNet [Электронный ресурс] URL: <https://technet.microsoft.com/en-us/library/bb497060.aspx>. (Дата обращения 23.05.2018)

2. Следствие из теоремы Колмогорова–Арнольда–Хехт-Нильсена. / Искусственные нейронные сети. Теория и практика [Электронный ресурс] Режим доступа: <http://stu.scask.ru/book_ins.php?id=7> (Дата обращения: 23.05.2018)

3. *Макконелл С.* Совершенный код. М.: РУССКАЯ РЕДАКЦИЯ, 2015.

4. *Скиена С.* Алгоритмы Руководство к разработке. М.: БХВ-Петербург, 2014.

5. *Избачков Ю.С.* Информационные системы СПб.: Питер, 2006.

6. *Хайкин С.* Нейронные сети: полный курс. М.: Издательский дом Вильямс, 2008.

## 