

## **АННОТАЦИЯ**

Тема выпускной квалификационной работы: «Система поддержки принятия решений обработки заявок клиентов страховой компании»

Информация о выпускной квалификационной работе: 70 страниц, из них основного текста 53 страниц, списка использованной литературы 3 страницы, приложений – 14. Таблиц – 7, рисунков – 24.

Цель работы: повышение эффективности работы с клиентом, автоматизация процесса и повышение прибыли страховой компании.

Ключевые слова: интеллектуальные системы, искусственные нейронные сети, автоматизация бизнес-процессов, медицинские страховые компании, анализ данных.

The final qualifying work «Decision support system for processing claims of clients of an insurance company».

Keywords: intelligent systems, artificial neural networks, automation of business processes, medical insurance companies, data analysis.

## **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ТЕРМИНОВ**

**ИС** – интеллектуальная система,

**ИНС** – искусственная нейронная сеть,

**СППР** – система поддержки принятия решений,

**ИИ** – искусственный интеллект,

**ПО** – программное обеспечение,

**ПК** – персональный компьютер,

**ТЗ** – техническое задание

**БД** – база данных

## Оглавление

ВВЕДЕНИЕ	7
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	8
1.1 Описание предметной области	8
1.2 Проблема и существующие решения	9
1.3 Разрабатываемое решение	10
Выводы	11
2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ	12
2.1 Назначение и цели разработки системы	12
2.2 Основные функции, подлежащие разработке	12
2.3 Требования к аппаратно-программной платформе	12
2.4 Требования по экономике	13
2.5 Режим работы	13
2.6 Порядок контроля	13
2.7 Требования к документированию	13
3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА	14
3.1 Проектирование ИНС	14
3.1.1 Нейрон	14
3.1.2 Синапс	14
3.1.3 Архитектуры ИНС	16
3.1.4 Функции активации	18
3.1.5 Слои нейронной сети	19
3.2 Конфигурация нейронной сети	21
3.3 Обучение ИНС	22
3.4 Выбор средств разработки	24
3.5 Структуры данных	26
3.6 Используемые библиотеки	27
3.7 Описание модулей	28
3.7.1 Модуль миграции базы данных	28
3.7.2 Модуль взаимодействия с базой данных	28
3.7.3 Модуль доступа к системе	28
3.7.4 Модуль чтения/записи информации из веб-клиента	28
3.7.5 Модуль интеллектуальной системы	29
3.7.6 Модуль валидации обучения ИНС	29

3.7.7	Модуль генерации отчётов	29
3.8	Архитектура системы	29
3.9	Реализация системы	30
3.9.1	Блок-схема алгоритма работы системы	30
3.9.2	Алгоритм работы реализуемой интеллектуальной системы	31
3.9.3	Алгоритмы программных модулей	31
3.10	Тестирование системы	33
3.10.1	Тестирование модуля интеллектуальной системы	33
3.10.2	Тестирование модуля доступа к системе	34
3.10.3	Тестирование модуля миграции БД	35
3.10.4	Тестирование модуля валидации обучения ИНС	35
3.10.5	Тестирование работы модуля генерации отчётов	36
3.11	Модификация системы	36
	Выводы	36
4.	РАСЧЁТНАЯ ЧАСТЬ	37
4.1	Расчёт надёжности по формуле Миллса	37
4.2	Расчёт объёма памяти	38
	Выводы	39
5.	РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	40
5.1	Вход в систему	40
5.2	Оформление новой заявки	40
5.3	Просмотр списка отправленных заявок	41
5.4	Просмотр данных в отправленной заявке	42
5.5	Генерация отчёта	42
	Выводы	43
6.	ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ ПРОЕКТА	44
6.1	Организация и планирование работ по теме	44
6.2	Расчёт стоимости проведения работ по теме.	46
6.2.1	Материалы, покупные изделия и полуфабрикаты	46
6.2.2	Специальное оборудование	47
6.2.3	Основная заработная плата	47
6.2.4	Дополнительная заработная плата	47
6.2.5	Страховые отчисления	47
6.2.6	Командировочные расходы	47

6.2.7	Контрагентские услуги	47
6.2.8	Накладные расходы	47
6.2.9	Прочие расходы	47
6.2.10	Полная себестоимость проекта	48
6.2.11	Договорная цена	48
	<b>Выводы</b>	48
	<b>ЗАКЛЮЧЕНИЕ</b>	49
	<b>СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ</b>	51
	<b>ПРИЛОЖЕНИЕ</b>	54

## **ВВЕДЕНИЕ**

Тема выпускной квалификационной работы бакалавра - система поддержки принятия решений обработки заявок клиентов страховой компании. В рамках данной темы подразумевается программная реализация системы, повышающей эффективность решения задач оценки качества клиентов.

Система страхового скоринга – важный инструмент для минимизации количества ошибок, искажения данных, предотвращения мошенничества. Использование скоринга повысит качество принятия решений по оформлению страхового полиса и в конечном итоге повысит эффективность деятельности страховой компании в целом. [1]

Единой системы страхового скоринга не существует. Страховые компании используют либо одно из типовых решений, либо собственные разработки. Критерии оценки качества клиента у страховых компаний, которые занимаются медицинским страхованием, также отличаются, так как законодательство Российской Федерации никак не регламентирует набор правил оценки рисков.

Актуальностью данной работы является использование популярных методов решения задач страхового скоринга, объединённых в интеллектуальную систему поддержки принятия решений, что позволит стандартизировать и автоматизировать бизнес-процессы, которые относятся к работе с клиентом и обработке его данных, что в конечном итоге приведёт к повышению эффективности и качества взаимодействия с клиентом. [2]

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Описание предметной области

В хозяйственной и финансовой деятельности человека возникает множество причин, приводящих к убыткам и потерям, такие как несчастные случаи, заболевания, влекущие за собой временную или постоянную потерю трудоспособности и снижение заработка или доходов. Для возмещения этих потерь предназначен страховой рынок.

Страховой рынок – сегмент финансового рынка, на котором формируется спрос и предложение на страховые услуги и соответствующие им финансовые потоки, которыми в той или иной степени пользуются население и нефинансовые компании, банки и государственные органы. [3] Всеобщность страхования определяет непосредственную связь страхового рынка с финансами компаний и населением, банковской системой и государственным бюджетом. Между страхователями и потребителями страховых услуг возникают устойчивые финансовые потоки.

На протяжении всей жизни люди вступают во множество различных отношений (социальные, производственные), которые вместе с благоприятными последствиями сопровождаются соответствующими рисками. Практически каждое событие в жизни человека связано с рисками, которые могут реализовывать, как и благоприятные последствия (риск оправданный), так и неблагоприятные (влечёт за собой убытки) или отсутствующие (нулевой результат).

Страхование выступает одним из основных факторов развития экономики. [4] Его роль заключается в создании условий «продуктивного бизнеса», стимулирующих предпринимателей проводить более активную политику внедрения инноваций, защищая их деятельность от неблагоприятных последствий инновационных рисков. Возрастает предпринимательская мотивация к разработке новых идей, позволяющая в

определённой степени идти на риск, так как обеспечивает защиту от убытков при наступлении страхового случая. Количество застрахованных лиц и страховых случаев напрямую влияет на доход страховых компаний. Они заинтересованы в снижении частоты страховых случаев (при наступлении страхового случая, организация несёт убыток), что в итоге сказывается на их прибыли и укреплении экономики в целом.

Страховые компании и их совокупная страховая деятельность является необходимой частью любой экономической системы. [5] Её важность заключается в предоставлении возможностей снижения финансовых и иных рисков в случае рискованной ситуации. Услуги страхования стимулируют малое и среднее предпринимательство на идейный «толчок», застрахованный риск, в случае свершения которого, предприятие не остаётся полным банкротом, а продолжает инновационную деятельность. Всё это положительно сказывается на расширении экономического потенциала страны.

## **1.2 Проблема и существующие решения**

В настоящее время компаниям требуется обилие сотрудников и времени для обработки входящих заявок от будущих клиентов компании. Компании вынуждены выделять ресурсы на оплату рутинного труда, вычислительной техники, её обслуживания, администрирования и содержания места для каждого сотрудника. Такое потребление ресурсов компании негативно влияет на рост клиентской базы, а что важнее, прибыли из-за траты времени и ресурсов при найме новых сотрудников, выделение им рабочего пространства, оборудования и оплаты труда.

В век информационных технологий в целях повышения эффективности принято автоматизировать не только процессы на производстве, но и бизнес-процессы. [6] Благодаря этому принимать заказы, выставить счета, отправлять товары и многое другое можно делегировать специальной платформе без непосредственного участия людей. Автоматизация нужна не



только для повышения эффективности при увеличении нагрузки, но и для стандартизации бизнес-процессов, чтобы сотрудники не могли совершить неверное действие, и всегда понимали, что нужно делать дальше.

В автоматизации бизнес-процессов не последнюю роль начал занимать и ИИ. [7] Сейчас интеллектуальные системы помогают с поиском и обогащением данных из открытых источников, в квалификации заявок в сервисные центры, с возможным автоматическим решением проблемы, обрабатывают повторяющиеся вопросы, например, об продуктах или услугах компании, что позволяет переложить нагрузку с персонала компании на интеллектуальную систему.

Система с ИИ, автоматизирующая бизнес-процессы, снимают финансовую нагрузку на компании, делают работу стандартизированной и стабильной, позволяя перераспределить штат на новые должности. [8]

### **1.3 Разрабатываемое решение**

Для автоматизации процесса обработки заявки клиента страховой компании можно использовать интеллектуальную систему, что повысит эффективность работы с клиентом, поможет стандартизировать подход к оценке клиента для различных страховых продуктов и уменьшит затраты компании. [9]

ИС предполагает использование всех эффективных методов и инструментов для автоматизации процесса анализа данных клиентов страховых компаний, которые будут взаимодействовать друг с другом, образуя модульную структуру.

Данный подход к реализации будет значительно мощнее стандартных решений, так как участником бизнес-процесса будет только клиент, а его данные будут обрабатываться системой в автономном режиме.

При необходимости внедрения корректировок в работу с информацией можно запустить переобучение на новых данных, что позволит повысить эффективность и точность анализа заявок с течением времени.

Потенциально такая ИС обработки заявок клиентов медицинской страховой компании имеет следующие преимущества:

- Эффективная работа с данными;
- Уменьшение времени ожидания решения по заявке клиентом;
- Повышение качества решения;
- Отсутствие человека в процессе оценки, что влечёт за собой беспристрастность решения;
- Стандартизация оценки при работе с разными страховыми продуктами;
- Гибкость конфигурации, что позволяет применять систему для работы с данными в других предметных областях;
- Сокращение расходов на персонал;
- Инструментарий управления системой;
- Мониторинг работы системы;
- Подробный анализ работы с клиентами.

## **Выводы**

В первом разделе проанализирована предметная область страхового рынка, рассмотрена проблема оценки клиентов медицинских страховых компаний, отражены преимущества разрабатываемой интеллектуальной системы для автоматизации процессов страховой компании.

## **2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

### **2.1 Назначение и цели разработки системы**

Цель разработки – реализация интеллектуальной системы с использованием ИНС, позволяющей автоматизировать процесс страхового скоринга, повысить эффективность работы с клиентами, уменьшить участие людских ресурсов в бизнес-процессе.

### **2.2 Основные функции, подлежащие разработке**

- Авторизация пользователя;
- Регистрация пользователя;
- Ввод данных клиента с веб-клиента;
- Оповещение клиента о решении;
- Отправка данных с веб-клиента в систему;
- Отправка данных в веб-клиент из системы;
- Чтение наборов данных из файла;
- Работа с БД;
- Обучение модели на наборе данных для тренировки;
- Сохранение конфигурации весов в системную таблицу в базе данных;
- Чтение конфигурации весов из базы данных;
- Генерация отчёта по работе с клиентами системой.

### **2.3 Требования к аппаратно-программной платформе**

- Операционная система: Windows, Linux, MacOS;
- Установленная программная платформа Java Virtual Machine;

- Установленная объектно-реляционная система управления базами данных PostgreSQL;
- Объем свободного места на жестком диске не менее 3 Гб;
- Оперативная память (ОЗУ) – в зависимости от объема входных данных, но не менее 1 Гб;
- Тактовая частота процессора – не менее 1 ГГц.

## **2.4 Требования по экономике**

I. Организация и планирование работ по теме.

II. Расчёт стоимости проведения работ по теме.

## **2.5 Режим работы**

Запуск программного обеспечения должно осуществляться на выделенном или специализированном компьютере, который выполняет сервисные программы. Взаимодействие с системой выполняется средствами веб-клиента в браузере.

## **2.6 Порядок контроля**

В процессе разработки должны быть произведены модульное, интеграционное и функциональное тестирование. Все найденные ошибки и недоработки должны быть зафиксированы и исправлены.

## **2.7 Требования к документированию**

Требуется разработать руководство пользователя, в котором будет отражены правила эксплуатации программного обеспечения.

### **3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА**

#### **3.1 Проектирование ИНС**

##### **3.1.1 Нейрон**

Нейрон – в биологии – клетка, предназначенная для приема извне, обработки, хранения, передачи и вывода вонне информации с помощью электрических и химических сигналов.

В ИНС нейрон сохраняет основную свою функцию, только без электрических и химических сигналов. Нейрон позволяет решать трудно формализуемые и нечётко поставленные задачи, задачи, для которых неизвестно алгоритмическое решение, но для решения, которых можно оперировать выборками данных из совокупности, для которых уже есть решение. [10]

Между собой нейроны соединены синапсами.

##### **3.1.2 Синапс**

Синапс – связующее звено между двумя нейронами, в ИНС характеризуется весом связи. Благодаря ему, входная информация изменяется, когда переходит из одного нейрона в другой. [11]

Нейроны содержат два вида информации:

- Входная – сумма всей информации, умноженной на значение веса синапса, поступившая в нейрон;
- Исходящая – результат функции активации от входящей.

Входящая информация вычисляется по формуле (3.1):

$$N_{input} = \sum_{i=0}^n I_i * W_i \quad (3.1)$$

где:

$N_{input}$  – входящая информация;

$I_i$  – информация с предыдущего  $i$ -ого нейрона;

$W_i$  – вес синапса, соединяющего текущий нейрон с  $i$ -ым;

$n$  – количество нейронов, соединяющихся с текущим нейроном.

Исходящая информация вычисляется по формуле (3.2):

$$N_{output} = F_{act}(N_{input}) \quad (3.2)$$

где:

$N_{output}$  – исходящая информация;

$F_{act}$  – функция активации;

$N_{input}$  – входящая информация.

Как видно из формулы 1 у того нейрона, у которого вес больше, будет доминировать информация в следующем. Пример работы нейронов и синапсов изображён на рис. 3.1.

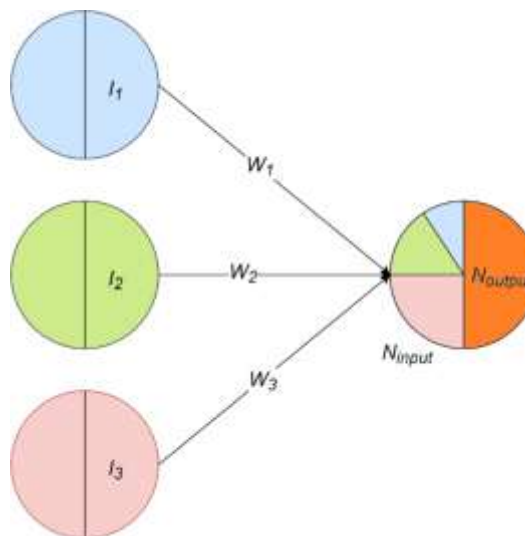


Рис. 3.1. Пример передачи информации синапсами между нейронами.

Совокупность весов нейронной сети или т.н. матрица весов – мозг всей системы. В самом начале работы веса назначаются случайным образом и при обучении корректируются.

### 3.1.3 Архитектуры ИНС

Наиболее распространённые архитектуры нейронных сетей:

1. сверточные
2. рекуррентные
3. полносвязные прямого распространения

Свёрточная нейронная сеть нацелена на эффективное распознавание образов и входит в состав технологий глубокого обучения. Показывает наилучшие результаты в области распознавания лиц. Успех обусловлен возможностью учета двумерной топологии изображения. [12] Пример архитектуры сверточной сети изображен на рис. 3.2.

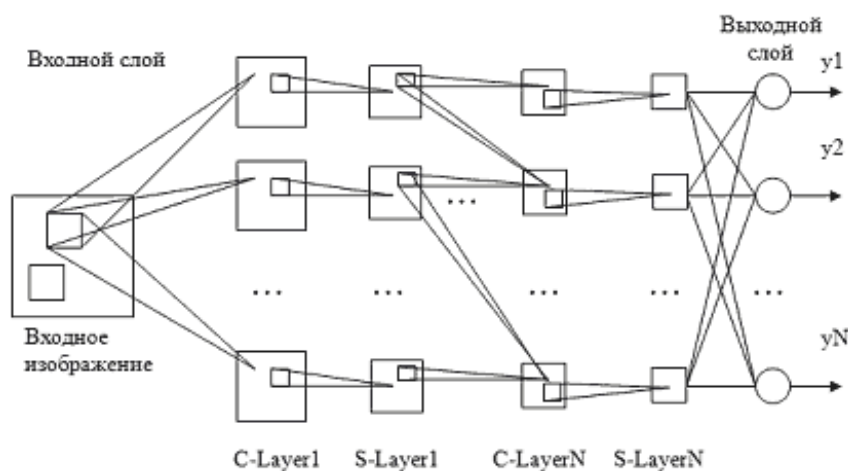


Рис. 3.2. Архитектура сверточной нейронной сети.

Рекуррентная нейронная сеть необходима для обработки серий событий во времени или последовательности пространственных цепочек. [13] Связи в такой сети образуют направленную последовательность с использованием циклов. Пример архитектуры рекуррентных сетей изображен на рис. 3.3.

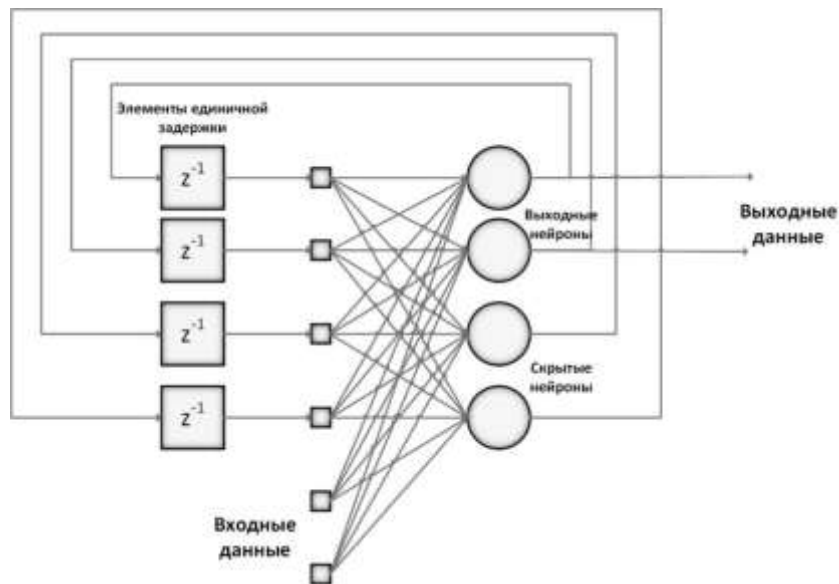


Рис. 3.3. Архитектура рекуррентной нейронной сети.

Полносвязная нейронная сеть прямого распространения — это сеть, в которой каждый нейрон связан со всеми остальными нейронами, находящимися в соседних слоях, и в которой все связи направлены строго от входных нейронов к выходным, соединения не образуют цикл. [14] Информация между нейронами перемещается в одном направлении вперед: от входных узлов через скрытые (если есть) к выходным. Эта сеть может решить задачу классификации, если нужно выделить два любых класса. Пример архитектуры полносвязной сети изображен на рис. 3.4.

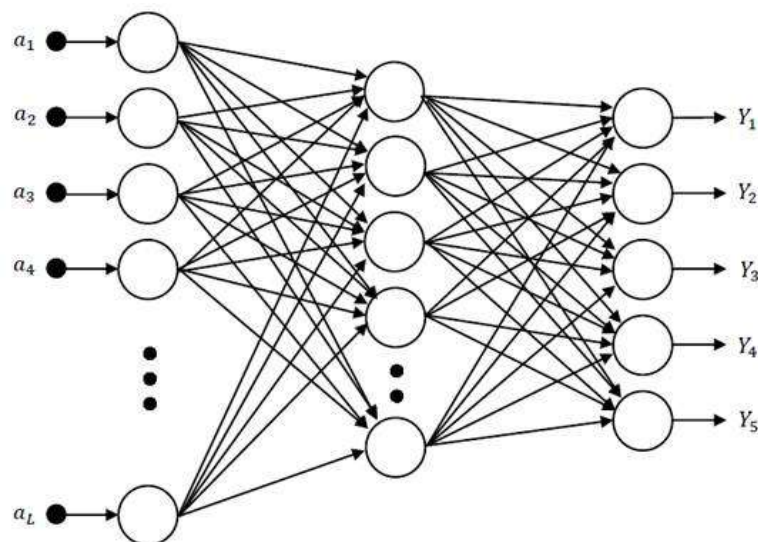


Рис. 3.4. Архитектура полносвязной сети.



### 3.1.4 Функции активации

В ИНС функция активации нейрона определяет выходной сигнал, который вычисляется входным сигналом или набором входных сигналов, то есть их суммой. [15] Наиболее распространенные функции активации:

1. единичная ступень (3.3), область значений  $\{0, 1\}$

$$f(x) = 0, x < 0; f(x) = 1, x \geq 0 \quad (3.3)$$

2. сигмоид (3.4), область значений  $(0, 1)$  (рис. 3.5)

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3.4)$$

3. гиперболический тангенс (рис. 3.5), область значений  $(-1, 1)$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.5)$$

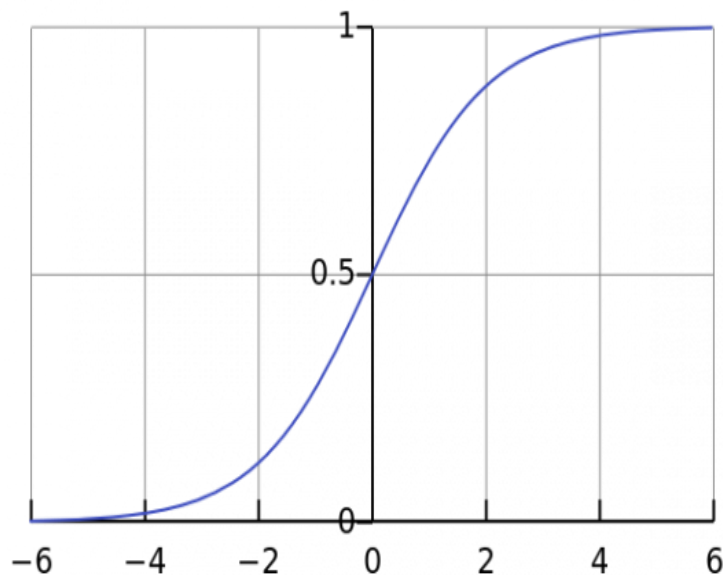


Рис. 3.5. Сигмоида.

Одна из причин, по которой сигмоид используется в нейронных сетях, это простое выражение его производной через саму функцию.

### 3.1.5 Слои нейронной сети

Существует три вида слоёв:



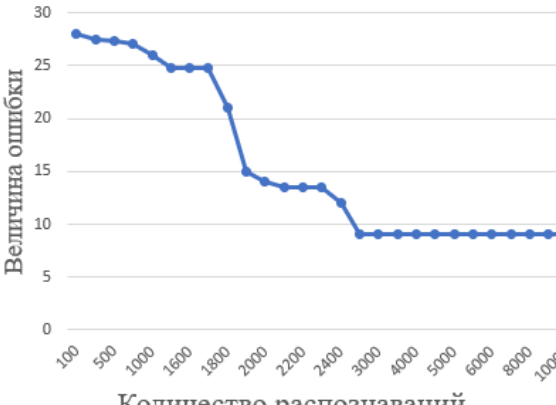
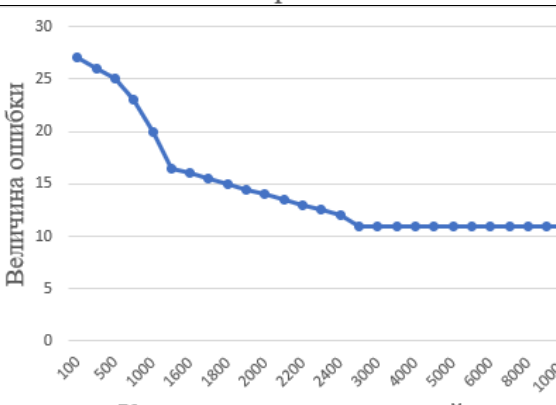
- 1) Входной – принимает и нормализует информацию;
- 2) Скрытый – обрабатывает входящую с предыдущих слоёв и передаёт следующему;
- 3) Выходной – выводит результат.

На входном слое количество нейронов должно быть больше или равно количеству аргументов. В случае с обработкой информации клиента страховой компании — это параметры, которые она сама для себя выбирает, так как законодательно в РФ никак не регламентируется оценка рисков. В связи с тем, что нейроны оперируют числами в диапазоне  $[0;1]$  или  $[-1;1]$ , а входные параметры на множество порядков превышают эти диапазоны, данные необходимо нормализовать делением единицы на значение выходного параметра.

Количество скрытых слоёв и нейронов на них можно подбирать вручную, анализируя как изменяется ошибка. Результат перебора количества нейронов на двух скрытых слоях и зависимость ошибки от конфигурации изображена в таблице 3.1.

Таблица 3.1 – Результат перебора количества нейронов на двух скрытых слоях ИНС.

Количество нейронов в 1-м скрытом слое	Количество нейронов в 2-м скрытом слое	График ошибки	Процент ошибок																																		
4	6	 <table><caption>Данные для графика ошибки</caption><tr><th>Количество распознаваний</th><th>Величина ошибки</th></tr><tr><td>100</td><td>30</td></tr><tr><td>500</td><td>29</td></tr><tr><td>1000</td><td>28</td></tr><tr><td>1500</td><td>28</td></tr><tr><td>2000</td><td>27</td></tr><tr><td>2500</td><td>25</td></tr><tr><td>3000</td><td>22</td></tr><tr><td>3500</td><td>20</td></tr><tr><td>4000</td><td>18</td></tr><tr><td>4500</td><td>16</td></tr><tr><td>5000</td><td>15</td></tr><tr><td>6000</td><td>15</td></tr><tr><td>7000</td><td>15</td></tr><tr><td>8000</td><td>15</td></tr><tr><td>9000</td><td>15</td></tr><tr><td>10000</td><td>15</td></tr></table>	Количество распознаваний	Величина ошибки	100	30	500	29	1000	28	1500	28	2000	27	2500	25	3000	22	3500	20	4000	18	4500	16	5000	15	6000	15	7000	15	8000	15	9000	15	10000	15	15
Количество распознаваний	Величина ошибки																																				
100	30																																				
500	29																																				
1000	28																																				
1500	28																																				
2000	27																																				
2500	25																																				
3000	22																																				
3500	20																																				
4000	18																																				
4500	16																																				
5000	15																																				
6000	15																																				
7000	15																																				
8000	15																																				
9000	15																																				
10000	15																																				

4	4	 <p>Величина ошибки</p> <p>Количество распознаваний</p>	12
3	4	 <p>Величина ошибки</p> <p>Количество распознаваний</p>	13
4	5	 <p>Величина ошибки</p> <p>Количество распознаваний</p>	9
8	4	 <p>Величина ошибки</p> <p>Количество распознаваний</p>	11

Из таблицы 3.1 видно, что наилучший результат работы показала нейронная сеть с четырьмя нейронами в первом скрытом слое и пятью нейронами на втором скрытом слое. Ошибка в такой конфигурации

минимальна, а стабильный уровень отклика и безотказная работа достигается за минимальное время.

Из результатов также видно, что чрезмерное увеличение нейронов в скрытых слоях может привести к переобучению, когда правильно обученная сеть теряет способность сообщать информацию, а ошибка начинает расти.

Решение по заявке клиента бинарное: она одобрена или нет. Для обработки этой ситуации достаточно одного нейрона на выходном слое, который будет складывать информацию на входе и применять функцию активации для нормализации информации на выходе.

### **3.2 Конфигурация нейронной сети**

Так как скоринг – это процесс классификации, лучшей архитектурой для данной нейронной сети является полносвязная прямого распространения.

Для реализации нейронной сети выбора её архитектуры недостаточно. Необходимо также выбрать наиболее подходящую функцию активации, сконфигурировать количество нейронов на входном слое, количество скрытых слоёв и количество нейронов на каждом из них и количество нейронов на выходном.

Так как результатом работы с заявкой является решение одобрить ее (решение равно 1), или нет (решение равно 0), а качество клиента не различается только на качественного и некачественного, наиболее подходящая функция активации – сигмоидальная функция, так как результатом не являются отрицательные числа, в отличие от гиперболического тангенса, и не ограничена двумя значениями, как единичная ступень.

Дополнительное преимущество сигмоидальной функции состоит в автоматическом контроле усиления. Для слабых сигналов кривая вход-выход имеет сильный наклон, дающий большое усиление. Когда величина сигнала становится больше, усиление падает. Таким образом, большие сигналы

воспринимаются сетью без насыщения, а слабые сигналы проходят по сети без чрезмерного ослабления. Другими словами, центральная область логистической функции, имеющая большой коэффициент усиления, решает проблему обработки слабых сигналов, в то время как области с падающим усилением на положительном и отрицательном концах подходят для больших возбуждений. [16]

Предполагаемая архитектура ИНС изображена на рис. 3.6. Как видно из схемы, в каждый нейрон входного слоя поступает одно из значений клиента, после обработки всех входных параметров информация поступает в следующий слой. На выходном нейроне после суммирования всех входных параметров от предыдущего слоя результатом работы нейронной сети является число в диапазоне от 0 до 1, где чем ближе к 0, тем ниже качество клиента, а чем ближе к 1, тем выше качество.

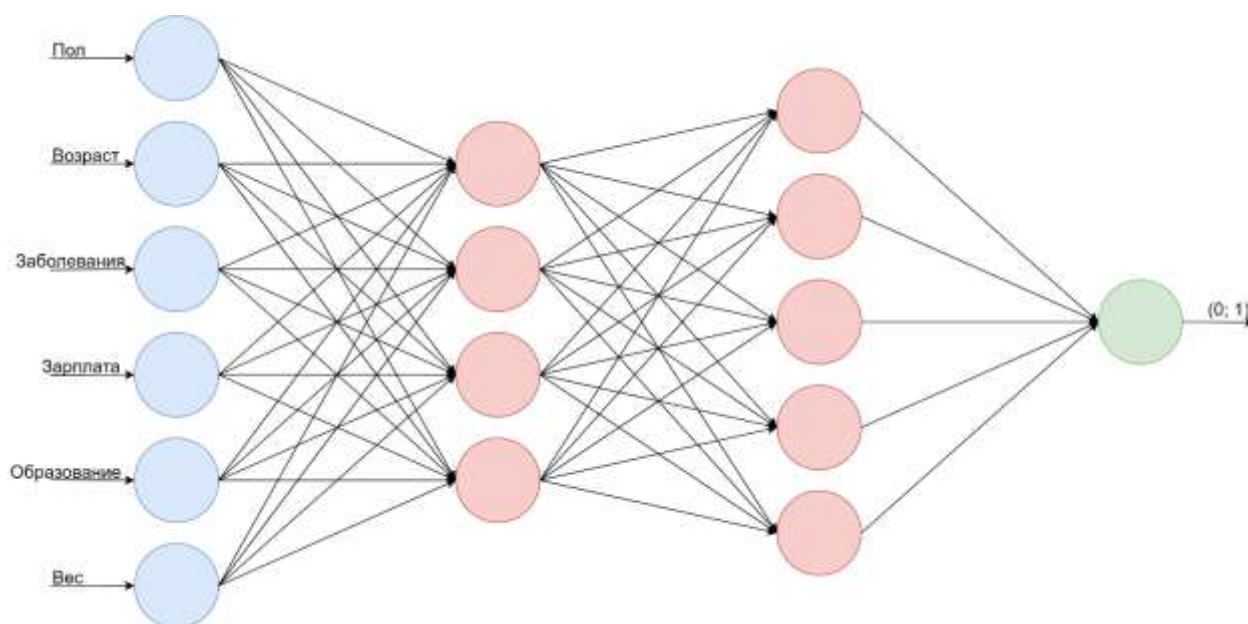


Рис. 3.6. Архитектура ИНС.

### 3.3 Обучение ИНС

На этапе обучения нейронная сеть выявляет закономерность между входными образами и выходными, как бы «обобщая» весь полученный опыт. Такая способность к обучению является феноменальной особенностью ИНС.

В подобных задачах определяющим успех фактором является большая обучающая выборка, для которых известно решение. Основное направление в которых применяются нейронные сети — это разнообразные интеллектуальные системы для анализа данных, распознавания образов, прогнозирования и т.д. [17]

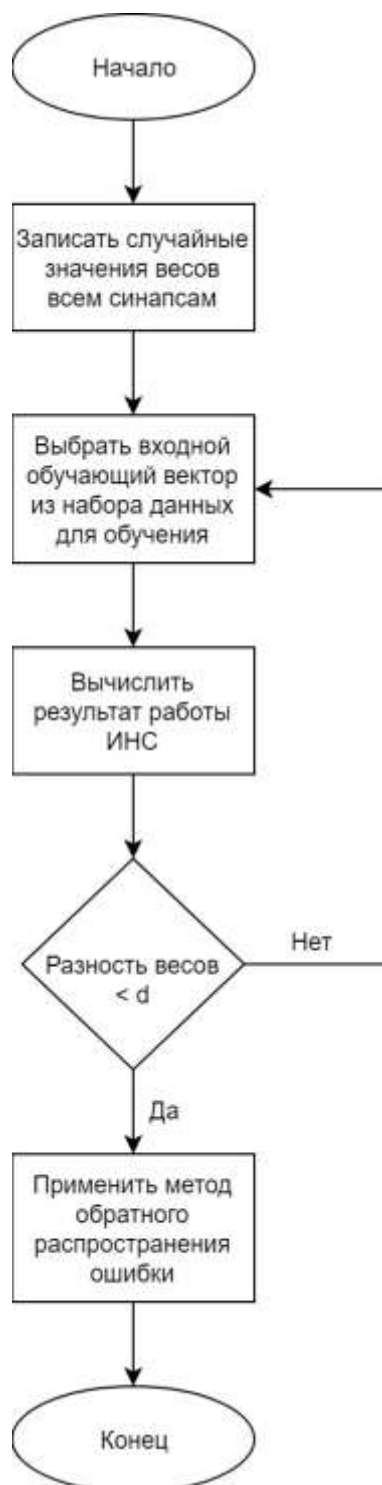


Рис. 3.7. Алгоритм обучения ИНС.

Изначально всем весам нейронов скрытого слоя и выходного слоя присваиваются некоторые случайные величины значением от 0 до 1. Далее подаётся некий входной вектор данных на вход нейронной сети, и активируется выходной вектор. Для скрытых слоев вычисляется суммирование произведения весов связей всех нейронов в скрытых слоях с входным вектором. Для выходного слоя складываются произведения весов связей нейронов выходного слоя с выходным вектором. В результате этой операции получается полноценный выходной вектор сети. Далее начинает работать алгоритм обратного распространения ошибки. В качестве функционала подлежащего минимизации с помощью метода градиентного спуска вычисляется среднеквадратичное отклонение реального выходного вектора сети от заранее предполагаемого. Для процесса минимизации ошибки и корректировки весов применяется метод градиентного спуска, а в качестве функции активации используется сигмоидальная функция. Алгоритм обучения представлен на рис. 3.7.

### **3.4 Выбор средств разработки**

Машинное обучение и ИИ являются технологическим прорывом. С помощью этих технологий стали доступны реализации голосовых ассистентов, способные забронировать стол в ресторане, заказать билет или отвечать на вопросы пользователей. Термин «машинное обучение» ввёл в 1959 году специалист в области информатики Сэмюэл Артур. Технология с каждым годом развивается, как и программирование. Были изобретены наборы инструментов и стандартных реализаций для быстрой и удобной разработки программных продуктов с использованием ИИ.

TensorFlow — это комплексная платформа для машинного обучения с открытым исходным кодом. Имеет гибкую экосистему инструментов, библиотек и ресурсов сообщества. Это позволяет исследователям использовать самые современные технологии.

PyTorch — это среда машинного обучения на языке Python с открытым исходным кодом, обеспечивающая вычисления с использованием вычислений на графическом процессоре. Инструмент подходит для быстрого прототипирования в исследованиях. Предлагает динамические графы вычислений, которые позволяют обрабатывать ввод и вывод переменной длины, что полезно при работе с рекуррентными нейронными сетями.

Keras — открытая среда глубокого обучения, написанная на Python. Она была разработана инженером из Google Франсуа Шолле и представлена в марте 2015 года. Фреймворк нацелен на оперативную работу с нейросетями и является компактным, модульным и расширяемым.

Инструмент для разработки ИНС должен обладать определёнными качествами:

- 1) Простота понимания нужна, чтобы инженер проекта при проверке программных команд понимал, что они делают, вне зависимости от квалификации
- 2) Удобство использования необходимо для эффективной производительности при реализации проекта
- 3) Расширяемость требуется для отсутствия архитектурных препятствий при разработке модификаций проекта
- 4) Низкое потребление ресурсов нужно стабильной работы программы на вычислительной машине при увеличении нагрузки

Эти параметры необходимы для высоких показателей на производстве.

В таб. 3.2 представлен сравнительный анализ вышеописанных инструментов разработки ИНС.

*Таблица 3.2. Сравнительный анализ вышеописанных инструментов разработки ИНС.*

	Простота понимания	Удобство	Расширяемость	Низкое потребление ресурсов
Tensor Flow	-	-	-	+
PyTorch	+	-	+	-
Keras	+	+	-	-



Из рассмотренных инструментов в табл. 3.2 ни один не удовлетворяет необходимым качествам, в следствии чего появляется необходимость в собственной уникальной реализации, которая будет удовлетворять всем вышеизложенным требованиям, на подходящем языке программирования.

Язык программирования должен быть объектно-ориентированным, для представления классов в виде абстракции объекта реального мира, обладающего свойствами и функционалом. Программные команды должны быть просты для понимания специалистом и лаконичными, чтобы не увеличивать сложность самого проекта. [18] Также язык программирования должен отвечать таким требованиям, как: кроссплатформенность и простота модернизации имеющейся реализации, производительность. Сравнительный анализ современных объектно-ориентированных языков программирования представлен в таб. 3.3.

*Таблица 3.3. Сравнительный анализ современных объектно-ориентированных языков программирования.*

	Простота понимания	Объём	Простота модернизации	Производительность	Кроссплатформенность
Java	+/-	-	+	+/-	+
Kotlin	+	+	+	+/-	+
C++	-	-	+	+	-
Python	+/-	+	-	+	+

Как видно из табл. 3.3, лучшим для реализации самописной ИНС является язык программирования Kotlin.

### 3.5 Структуры данных

На рис. 3.8 изображена структура таблиц, в которых будут записаны основные сущности системы, и их отношения.

Основные характеристики таблиц:

- *appeal* – заявка клиента, хранит ссылку на таблицу *user* и основные данные по заявке клиента: зарплата, образование, пол, возраст, вес и наличие заболеваний

- decision – решение по заявке клиента, содержит поле со ссылкой на заявку, решение и конфликтующие параметры
- user – пользователь, содержит в себе все необходимые поля для авторизации
- weight\_configuration – конфигурация весов

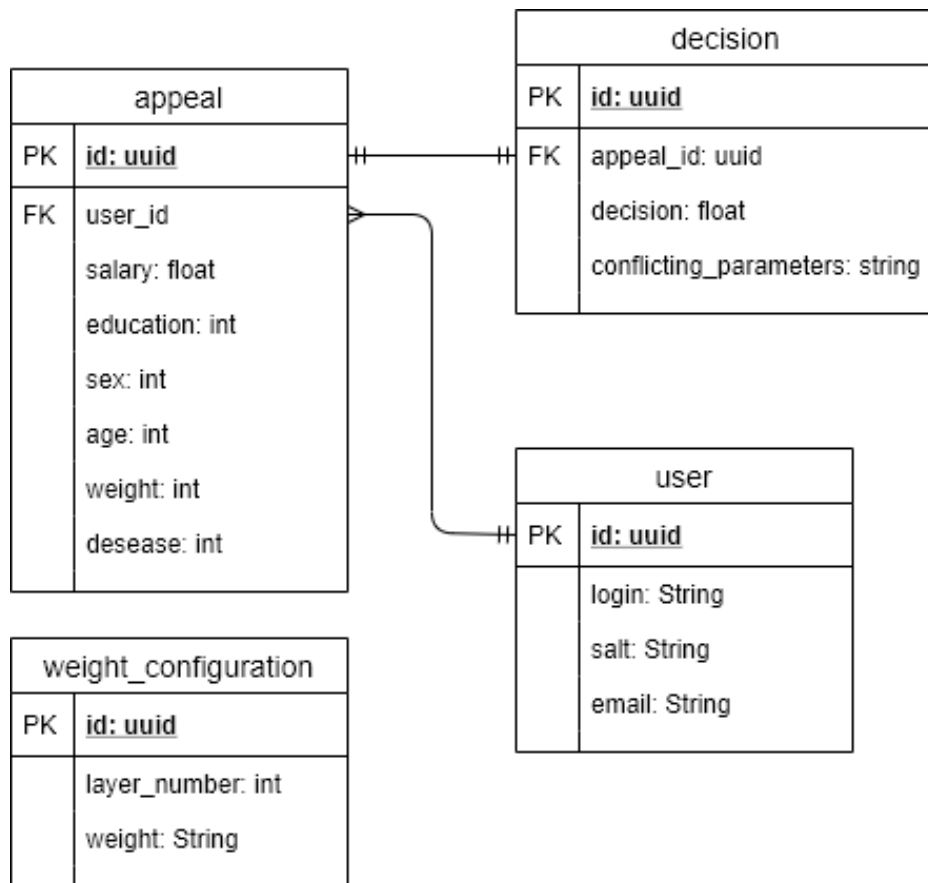


Рис. 3.8. Структура данных для работы интеллектуальной системы.

### 3.6 Используемые библиотеки

Для реализации модулей взаимодействия с базой данных, взаимодействия с веб-клиентом и доступа к системе используется библиотека Spring Framework.

Spring Framework предоставляет комплексную модель программирования и конфигурации для современных корпоративных приложений на основе Java на любой платформе развертывания.

Ключевым элементом Spring является инфраструктурная поддержка на уровне приложений: Spring фокусируется на подключении корпоративных приложений, чтобы группы могли сосредоточиться на бизнес-логике на уровне приложений без ненужных связей с конкретными средами развертывания. [19]

### **3.7 Описание модулей**

Модуль – функционально завершённый элемент программы. Модули могут быть довольно крупными, такими как модуль работы с базами данных. Суть модулей в разделении программы на основные подсистемы и определении взаимодействий между ними.

#### **3.7.1 Модуль миграции базы данных**

Отвечает за вносимые изменения в структуры схемы базы данных. Если в модуль поступила команда изменить или создать таблицу, то он её выполняет. Так же при миграции с первой схемы на вторую применяет структуру для выстраивания необходимой схемы.

#### **3.7.2 Модуль взаимодействия с базой данных**

Содержит в себе необходимый функционал для чтения и записи информации в таблицы базы данных. Также переводит сущности в системе к команде в синтаксисе языка базы данных.

#### **3.7.3 Модуль доступа к системе**

Авторизует и аутентифицирует пользователей, отвечает за распределение прав доступа, проверку данных для входа в систему. Не позволяет взаимодействовать с системой без регистрации.

#### **3.7.4 Модуль чтения/записи информации из веб-клиента**

Получает данные от веб-клиента, проверяет их целостность и валидность. Передаёт информацию соответствующим модулям.

### **3.7.5 Модуль интеллектуальной системы**

Отвечает за обработку массива данных и вычисления решения по заявке. Содержит реализацию ИНС для классификации модели качественного клиента от некачественного.

### **3.7.6 Модуль валидации обучения ИНС**

Отвечает за проверку обучения ИНС, чтобы плохо обученный модуль не принимал заявки клиентов, тем самым не повышал расходы по страховым случаям.

### **3.7.7 Модуль генерации отчётов**

Отвечает за генерацию отчётов по оформленным и отклонённым заявкам клиентов. Требуется для анализа работы системы вышестоящим руководством.

## **3.8 Архитектура системы**

Архитектура системы изображена на рис. 3.9.

Модуль миграции базы данных на нём не представлен, так как он начинает работу при запуске системы и на остальные модули не влияет. Если результат работы этого модуля вызвал ошибку, то запуск остальных модулей не осуществляется, а проблема фиксируется в текстовом файле.

По аналогичной причине не представлен модуль валидации обучения ИНС. После инициализации этот модуль запускается автоматически и начинает проверку работы модуля ИИ. Если тестирование показало, что нейронная сеть обучена плохо и выдает неверные результаты, то модуль валидации обучения сообщает об ошибке и генерирует отчёт в текстовый файл.

Как видно из рис. 3.9 взаимодействие с системой пользователем начинается с веб-клиента, который отправляет с ПК пользователя запрос с заявкой в модули доступа и работы с веб-клиентом. После успешного взаимодействия данные переходят в модуль работы с базой данных, который записывает в таблицу новую заявку и передает её интеллектуальной системе. Модуль интеллектуальной системы после обработки информации передает

своё решение в модуль работы с базой данных и в модуль работы с веб-клиенту, который в свою очередь отправляет ответ пользователю.



Рис. 3.9. Взаимодействие модулей.

### 3.9 Реализация системы

#### 3.9.1 Блок-схема алгоритма работы системы



Рис. 3.10. Блок-схема алгоритма работы системы.

### **3.9.2 Алгоритм работы реализуемой интеллектуальной системы**

Алгоритм реализуемой интеллектуальной системы следующий:

- Принять новую заявку;
- Извлечь данные;
- Если ИНС инициализирована, то преобразовать данные в массив;
- Иначе, начать инициализацию ИНС;
- Анализировать данные из массива;
- Извлечь результат из ИНС;
- Интерпретировать результат;
- Отобразить клиенту решение по заявке.

### **3.9.3 Алгоритмы программных модулей**

Алгоритм работы модуля миграции базы данных, следующий:

- Запуск работы модуля;
- Установка соединения с базой данных;
- Авторизация модуля в схеме;
- Чтение системных таблиц;
- Если в системной таблице нет записей или они не актуальны, то результат работы модуля: обновление схемы;
- Иначе, продолжить работу без изменений;
- Если результат обновление схемы вызвал сбой, то зафиксировать ошибку в системной таблице;
- Иначе, продолжить работу остальных модулей.

Алгоритм работы модуля взаимодействия с БД:

- Установка соединения с БД;
- Сериализовать данные сущности в запрос к схеме;
- Записать сериализованные данные в таблицу;

- Если результат операции не вызвал ошибок, то выполнить следующую запись;
- Иначе, вывести программную ошибку в файл.

Алгоритм работы модуля доступа к системе:

- Установка соединения с веб-клиентом;
- Чтение заголовка из запроса;
- Если данные в заголовке валидные, то продолжить работу остальных модулей;
- Если данные в заголовке валидные, но их срок жизни истёк, то обновить значение и отправить веб-клиенту;
- Иначе, не предоставлять доступ к системе.

Алгоритм работы модуля чтения/записи информации из веб-клиента системой:

- Принять данные из веб-клиента;
- Проверить валидность данных;
- Если данные валидны, то записать в базу данных;
- Иначе, отправить в ответе ошибку;
- Передать информацию модулю интеллектуальной системы;
- Прочитать результат работы;
- Отправить данные на веб-клиент.

Алгоритм работы модуля интеллектуальной системы:

- Если найдена конфигурация весов в таблице базы данных, то применить на нейронную сеть;
- Иначе, начать обучение на тестовом наборе данных;
- Провести тестирование на одной случайной записи из набора данных;
- Если ошибка системы после тестирования выше допустимой, то начать обучение с случайными весами и удалить конфигурацию;

- Иначе, начать чтение входящих данных для обработки;
- Если решение по заявке 1, то одобрить заявку;
- Иначе, отказать клиенту;
- Записать решение в базу данных.

Алгоритм работы модуля валидации обучения ИНС:

- Если ИНС инициализирована, начать тестирование;
- Иначе, инициализировать ИНС;
- Если тестирование прошло успешно, продолжить работу модуля ИНС;
- Иначе, записать ошибку модуля ИНС в файл, завершить работу.

Алгоритм работы модуля генерации отчётов:

- Собрать выборку записей из таблиц за запрашиваемый период;
- Если записи не найдены, то сообщить об отсутствии заявок за период;
- Иначе, сгруппировать записи по календарным месяцам;
- В каждом месяце найти заявки, решение по которым положительное
- Количество одобренных заявок записать в отчёт
- В каждом месяце найти заявки, решение по которым отрицательное
- Количество отклонённых заявок записать в отчёт
- Отобразить отчёт

### **3.10 Тестирование системы**

#### **3.10.1 Тестирование модуля интеллектуальной системы**

Модуль с ИНС при инициализации ищет наличие конфигурации весов в базе данных, если он не обнаружен, то обучается на тестовом наборе данных. Результат обучения представлен на рис. 3.11. Как видно из рисунка,



модуль загрузил записи для тренировки, обучился десять эр с уменьшающейся ошибкой и успешно завершил обучение с минимально допустимой погрешностью в 2,3 процента.

```
Заявок загружено всего: 10000
Ошибка ~ 23.6 за 1 эру
Ошибка ~ 21.6 за 2 эру
Ошибка ~ 18.6 за 3 эру
Ошибка ~ 17.6 за 4 эру
Ошибка ~ 17.1 за 5 эру
Ошибка ~ 15.3 за 6 эру
Ошибка ~ 12.9 за 7 эру
Ошибка ~ 11.1 за 8 эру
Ошибка ~ 9.7 за 9 эру
Ошибка ~ 2.3 за 10 эру
Обучение завершено
Обучение завершено успешно с ошибкой в 2.3%
```

Рис. 3.11. Результат обучения нейронной сети на тестовом наборе.

При работе с клиентом модуль выдает результат в виде дробного числа от 0 до 1. После преобразования заявки в массив чисел и обработки данных результат выводиться в консоль. Пример работы системы с заявкой изображен на рис. 3.12.

```
Поступила новая заявка
Данные клиента: [1, 25, 0, 0, 26555.0, 4, 65.0]
Идет обработка заявки...
Результат обработки: 0.892342134132
Принято решение: одобрить заявку
```

Рис. 3.12. Пример работы системы с заявкой клиента.

### 3.10.2 Тестирование модуля доступа к системе

Результатом работы модуля доступа к системе является прекращение работы сессии клиентского приложения, если данные для аутентификации не совпадают с имеющимися или не найдены. Пример работы изображён на рис. 3.13.

```
Открыта новая сессия с веб-клиентом
Пользователь не определён
Сессия закрыта
```

Рис. 3.13. Результат работы модуля доступа

### 3.10.3 Тестирование модуля миграции БД

Результатом работы модуля миграции базы данных является сконфигурированная БД, созданные таблицы с полями, отношения между сущностями. Пример работы модуля представлен на рис. 3.14.

```
HikariPool-1 - Starting...
HikariPool-1 - Start completed.
SELECT COUNT(*) FROM PUBLIC.DATABASECHANGELOGLOCK
CREATE TABLE PUBLIC.DATABASECHANGELOGLOCK (ID INT NOT NULL, LOCKED BOOLEAN NOT NULL, LOCKGRANTED TIMESTAMP,
LOCKEDBY VARCHAR(255), CONSTRAINT PK_DATABASECHANGELOGLOCK PRIMARY KEY (ID))
SELECT COUNT(*) FROM PUBLIC.DATABASECHANGELOGLOCK
DELETE FROM PUBLIC.DATABASECHANGELOGLOCK
INSERT INTO PUBLIC.DATABASECHANGELOGLOCK (ID, LOCKED) VALUES (1, FALSE)
SELECT LOCKED FROM PUBLIC.DATABASECHANGELOGLOCK WHERE ID=1
Successfully acquired change log lock
Creating database history table with name: PUBLIC.DATABASECHANGELOG
CREATE TABLE PUBLIC.DATABASECHANGELOG (ID VARCHAR(255) NOT NULL, AUTHOR VARCHAR(255) NOT NULL, FILENAME
VARCHAR(255) NOT NULL, DATEEXECUTED TIMESTAMP NOT NULL, ORDEREXECUTED INT NOT NULL, EXECTYPE VARCHAR(10) NOT NULL,
MD5SUM VARCHAR(35), DESCRIPTION VARCHAR(255), COMMENTS VARCHAR(255), TAG VARCHAR(255), LIQUIBASE VARCHAR(20),
CONTEXTS VARCHAR(255), LABELS VARCHAR(255), DEPLOYMENT_ID VARCHAR(10))
SELECT COUNT(*) FROM PUBLIC.DATABASECHANGELOG
Reading from PUBLIC.DATABASECHANGELOG
SELECT * FROM PUBLIC.DATABASECHANGELOG ORDER BY DATEEXECUTED ASC, ORDEREXECUTED ASC
SELECT COUNT(*) FROM PUBLIC.DATABASECHANGELOGLOCK
CREATE TABLE iaps.policy_holder (id UUID NOT NULL, name CLOB, age INT NOT NULL, sex INT NOT NULL, education_degree
INT NOT NULL, salary DOUBLE NOT NULL, possibility_of_injury_at_job DOUBLE NOT NULL, decision DOUBLE NOT NULL,
CONSTRAINT PK_POLICY HOLDER PRIMARY KEY (id))
Table policy_holder created
ChangeSet classpath:/iaps/liquibase-change.log.xml::1::avtammov_aa ran successfully in 9ms
SELECT MAX(ORDEREXECUTED) FROM PUBLIC.DATABASECHANGELOG
INSERT INTO PUBLIC.DATABASECHANGELOG (ID, AUTHOR, FILENAME, DATEEXECUTED, ORDEREXECUTED, MD5SUM,
DESCRIPTION, COMMENTS, EXECTYPE, CONTEXTS, LABELS, LIQUIBASE, DEPLOYMENT_ID) VALUES
('1', 'avtammov_aa', 'classpath:/iaps/liquibase-change.log.xml', NOW(), 1, '8:a872bd8aa1813e420f43fb3464aa129b',
'createTable tableName='policy_holder', '', 'EXECUTED', NULL, NULL, '3.6.2', '9649179411')
Successfully released change log lock
```

Рис. 3.14. Пример работы модуля миграции БД.

### 3.10.4 Тестирование модуля валидации обучения ИНС

Результатом работы модуля является либо продолжение работы системы при инициализации, либо завершение при ошибочном обучении ИНС. Пример работы модуля при успешном обучении отображён в рис. 3.15, при ошибочном в рис. 3.16.

```
Модуль ИНС инициализирован
Начинается тестирование
ИНС протестирована на записях в количестве 100 шт.
Ошибок не обнаружено
```

Рис. 3.15. Пример работы модуля валидации ИНС при успешном обучении.

```
Модуль ИНС инициализирован
Начинается тестирование
ИНС протестирована на записях в количестве 100 шт.
При тестировании обнаружены ошибки в вычислениях
Проверьте log файл
Завершение работы системы
```

Рис. 3.16. Пример работы модуля валидации ИНС при ошибочном обучении.

### 3.10.5 Тестирование работы модуля генерации отчётов

Результатом работы модуля является построенный на основе записей в БД отчёт, отображённый в веб-клиенте, на данных которого можно анализировать работу интеллектуальной системы и вести статистику клиентов медицинской страховой компании. Пример работы модуля изображен на рис. 3.17.

```
Обнаружено 144 заявки за промежуток 04.2020 – 05.2020
За 04.2020:
    25 заявок отклонено
    61 заявка одобрена
За 05.2020:
    32 заявка отклонена
    26 заявок одобрено
```

Рис. 3.17. Пример работы модуля генерации отчёта по работе с клиентами.

### 3.11 Модификация системы

В рамках реализации модуля была предусмотрена возможность гибкой конфигурации архитектуры самой нейронной сети: возможность выбора количества скрытых слоёв, количества нейронов на входном, скрытых и выходном слое, но для обработки более сложных моделей или иных целей нужно реализовать модификации:

- Выбор функции активации
- Выбор метода вычисления ошибки при обучении
- Выбор архитектуры нейронной сети
- Конфигурация связей нейронов между слоями

### Выводы

В рамках производственной практики были получены и закреплены практические навыки проектирования архитектуры нейронных сетей и их программных реализаций, был спроектирован и реализован основной функционал, спроектированы модули интеллектуальной системы.

## 4. РАСЧЁТНАЯ ЧАСТЬ

### 4.1 Расчёт надежности по формуле Миллса

Тестируя программу в течение некоторого времени, собирают статистику об ошибках. В момент оценки надежности по протоколу искусственных ошибок все ошибки делятся на собственные и искусственные. Соотношение, называемое формулой Миллса – формула 1:

$$N = n * M/m \quad (4.1)$$

где:

***N*** — первоначальное число ошибок в программе;

***M*** — количество искусственно внедренных ошибок;

***n*** — число найденных естественных ошибок;

***m*** — число обнаруженных искусственных ошибок.

Тестируя систему по модели Миллса, было внесено 10 искусственных ошибок, в ходе тестирования было обнаружено 7 искусственных и 3 естественных ошибок. Получена следующая оценка количества ошибок в программе:

$$N = 3 * \frac{10}{7} \approx 4$$

Количество необнаруженных ошибок:

$$(N - n) = 4 - 3 = 1$$

Легко заметить, что в описанном выше способе Миллса есть один существенный недостаток. Если будет найдено 100% искусственных ошибок, это будут означать, что и естественных ошибок было найдено 100%. Но чем меньше вноситься искусственных ошибок, тем больше вероятность того, что будут найдены они все. Для решения такой проблемы Миллс добавил вторую часть модели, предназначенную для проверки гипотезы о величине ***N***:

Предположим, что в программе  $N$  естественных ошибок. Внесем в неё  $M$  искусственных ошибок. Будем тестировать программу до тех пор, пока не найдем все искусственные ошибки. Пусть к этому моменту найдено  $n$  естественных ошибок. На основании этих чисел вычислим величину  $C$  – формула 2:

$$C = \begin{cases} 1, & \text{при } n > N \\ \frac{M}{M + N + 1}, & \text{при } n \leq N \end{cases} \quad (4.2)$$

Тестируя систему по модели Миллса, было введено 4 искусственные ошибки. В процессе тестирования не было обнаружено ни одной естественной ошибки.

$$C = \begin{cases} 1, & \text{при } n > N \\ \frac{10}{10 + 1 + 1}, & \text{при } n \leq N \end{cases}$$

Тогда вероятность безотказной работы будет равна 83%. Процент является довольно высоким и доказывает высокую безотказность работы.

## 4.2 Расчёт объёма памяти

Расчет объема внешней памяти осуществляют по формуле:

$$V_{\text{ВП}} = V_{\text{ОС}} + V_{\text{ОНД}} + V_{\text{данных}} + V_{\text{программы}}, \text{ Мб} \quad (4.3)$$

где:

$V_{\text{ВП}}$  — общий объем внешней памяти;

$V_{\text{ОС}}$  — объем внешней памяти. Хранит файлы ОС.

$V_{\text{ОНД}}$  — объем внешней памяти, который требуется для хранения обучающего набора данных;

$V_{\text{данных}}$  - объем внешней памяти для хранения музыкальных записей;

$V_{\text{программы}}$  - объем внешней памяти, который необходим для хранения приложения.

Объем внешней памяти ОС — 14 Гб, объем внешней памяти для хранения обучающих наборов данных – 3 Гб, объем внешней памяти для хранения музыкальных записей – 2 Гб, объем внешней памяти для хранения приложения – 500 Мб.

Таким образом,  $V_{\text{ВП}} = 14\,336 + 3072 + 2048 + 500 = 19956$  Мб или 19,9 Гб.

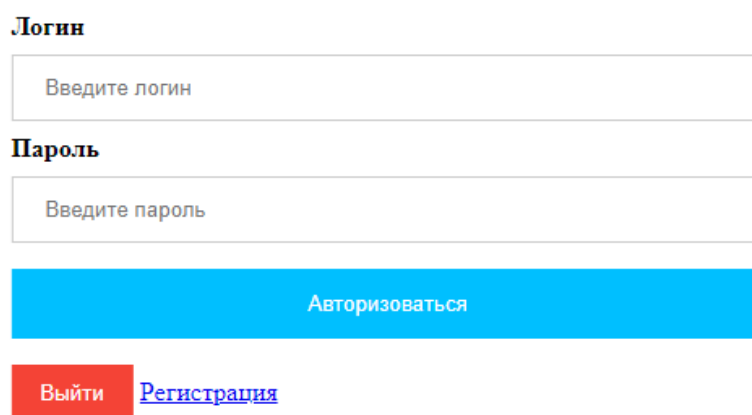
## **Выводы**

Проведен расчет надежности по формуле Миллса, результаты которого показали высокую безотказность работы разрабатываемой системы, и требуемого объёма памяти для штатной работы.

## 5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 5.1 Вход в систему

Для входа в систему пользователю нужно открыть в браузере веб-страницу по адресу, на котором запущена система, и ввести свой логин и пароль, если пользователь не зарегистрирован, он должен пройти регистрацию. Форма ввода логина и пароля изображена на рис. 5.1.



**Логин**

**Пароль**

Авторизоваться

Выйти [Регистрация](#)

Рис. 5.1. Форма авторизации.

При успешной авторизации пользователь войдёт в систему.

### 5.2 Оформление новой заявки

После окна авторизации для пользователя откроется окно оформления новой заявки. Для отправки требуется заполнить все поля формы и нажать кнопку «Отправить», если пользователь заполнит не все поля или заполнит их неправильно, заявка не отправится. Чтобы выйти из модального окна, нужно нажать кнопку «Закрыть». Пример формы оформления новой заявки на рисунке 5.2.

**Новая заявка**

ФИО
Возраст
Образование
Зарплата
Пол
Отправить
Заккрыть

Рис. 5.2. Форма для заполнения новой заявки.

### 5.3 Просмотр списка отправленных заявок

Для того, чтобы увидеть список отправленных заявок, необходимо нажать на кнопку «Отправленные заявки». Окно просмотра отправленных заявок содержит дату отправки, дату постановки решения и статус заявки. Пример формы на рис. 5.3.


Отправлено: 2020-01-29. Вынесено решение: 2020-01-30. Статус: Одобрено
Отправлено: 2020-01-14. Вынесено решение: 2020-01-25. Статус: Отказано
Отправлено: 2020-01-11. Вынесено решение: 2020-01-13. Статус: Отказано
Отправлено: 2020-01-09. Вынесено решение: 2020-01-10. Статус: Отказано
Отправлено: 2020-01-06. Вынесено решение: 2020-01-08. Статус: Отказано
Отправлено: 2020-01-01. Вынесено решение: 2020-01-03. Статус: Отказано
Заккрыть

Рис.5.3. Форма просмотра списка отправленных заявок.



## 5.4 Просмотр данных в отправленной заявке

Для просмотра данных в отправленной заявке нужно перейти в заявку, поместив курсор на заявку в списке. Откроется не редактируемая форма, в которой будут отражены все данные, которые пользователь отправил в систему. Пример формы просмотра отправленной заявки на рис. 5.4.

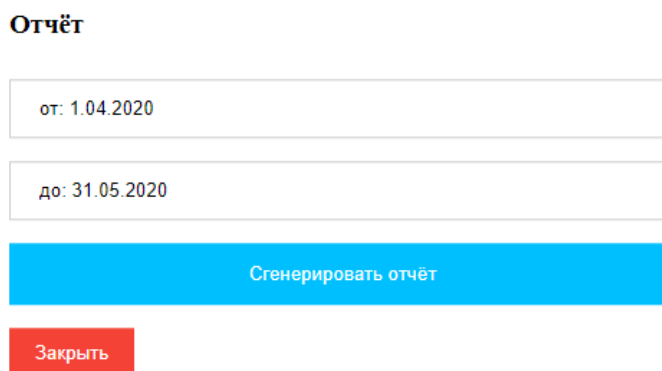


The screenshot shows a form titled "Заявка от 2020-01-29". It contains several input fields with the following data: "ФИО: Иванов И.И.", "Пол: мужской", "Возраст: 29", "Образование: высшее", "Зарплата: 60.000", and "Статус: отказано". At the bottom of the form is a red button labeled "Закрыть".

Рис. 5.4. Форма с подробной информацией об отправленной заявке.

## 5.5 Генерация отчёта

Для генерации отчёта по работе с заявками необходимо в личном кабинете навести курсор на кнопку «Сгенерировать отчёт» и выбрать промежуток времени. Пример формы генерации отчёта на рис. 5.5.



The screenshot shows a form titled "Отчёт". It has two input fields for date ranges: "от: 1.04.2020" and "до: 31.05.2020". Below these fields is a large blue button labeled "Сгенерировать отчёт". At the bottom is a red button labeled "Закрыть".

Рис. 5.5. Пример формы генерации отчёта за период.

После генерации отчёта откроется окно с данными по принятым и отклонённым заявкам за указанный промежуток времени. Для выхода из окна просмотра отчёта требуется навести курсор на кнопку «Заккрыть». Пример отчёта изображён на рис. 5.6.

**Отчёт за период: от 01-04-2020 до 31-05-2020**

За 04-2020:

Принято: 61 заявка
Отклонено: 25 заявок

За 05-2020:

Принято: 32 заявки
Отклонено: 26 заявок

**Всего: 144 заявки**

**Заккрыть**

Рис. 5.6. Пример отчёта.

## Выводы

В данном руководстве пользователя рассмотрены следующие процессы работы с реализованной системой:

- Вход в систему;
- Оформление новой заявки;
- Просмотр списка отправленных заявок;
- Просмотр данных в отправленной заявке;
- Генерация отчёта за период.

## **6. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ ПРОЕКТА**

Разработка любой информационной системы состоит из множества этапов: от проработки технического задания до внедрения на производство заказчика. Каждый из этапов требует определённых навыков от специалистов, причастных к работе над ним.

В данной работе описана разработка интеллектуальной системы, в которой автоматизирован процесс работы с заявкой клиента медицинской страховой компании.

В связи с тем, что в работе реализуется информационная система, можно отнести данную разработку к научно-исследовательской работе. Поэтому в рамках экономической части разрабатываются вопросы:

1. Организация и планирование работ по теме.
2. Расчёт стоимости проведения работ по теме.

### **6.1 Организация и планирование работ по теме**

Определение трудоёмкости выполняемых работ отличается повышенной сложностью и ответственностью, так как трудовые затраты составляют основную часть стоимости системы и напрямую влияют на сроки разработки.

Численность специалистов для выполнения работ определяется из анализа конкретных условий реализации и с таким расчётом, чтобы обеспечить их максимальную загрузку и минимальную длительность выполнения определённых работ и разработки системы в целом.

На рис. 6.1 изображён состав, необходимый для работы над проектом.

Чтобы создать интеллектуальную систему обработки входящих заявок клиентов медицинской страховой компании требуется задействовать группу из 5 человек:

- Менеджер проекта утверждает вместе с заказчиком ТЗ, следит за качественным выполнением этапов в срок;
- Аналитик отвечает за постановку задач, которые будут направлены разработчику, и тестирование;
- Разработчик реализует задачи по проекту, которые получает от аналитика, на языке программирования;
- Системный администратор отвечает за стабильную работу вычислительных мощностей, внутренней сети, подключения к внешней сети и программного обеспечения;



Рис. 6.1. Организационная структура группы

Оператор страховой компании – сотрудник, анализирующий требования страховой компании к клиентам, отвечает за реализацию набора данных для обучения нейронной сети.

На разработку проекта отводится 58 рабочих дней. В таблице 6.1 представлены этапы разработки.

Таблица 6.1 - Этапы разработки.

№	Название этапа	Исполнитель	Трудоемкость, чел/дни	Продолжительность работ, дни
1	Разработка и утверждение ТЗ	Менеджер проекта	5	5
		Аналитик	5	
2	Настройка ПО	Системный администратор	8	8
3	Загрузка данных страховых клиентов	Аналитик	3	3
		Оператор страховой компании	2	
4	Анализ данных	Аналитик	13	13
		Оператор страховой компании	13	
6	Реализация ИС	Разработчик	10	10
7	Тестирование проекта	Разработчик	3	5
		Аналитик	5	
8	Подготовка документации	Аналитик	10	14
		Менеджер проекта	14	
Итого			91	58

### 6.3 Расчёт стоимости проведения работ по теме.

Расходы, хозяйствующего субъекта, осуществляемые в процессе хозяйственной деятельности по выполнению НИР, представляют собой экономические затраты, оценка которых выражается в денежной форме путем составления калькуляции темы.

#### 6.3.1 Материалы, покупные изделия и полуфабрикаты

В таблице 6.2 содержится информация о материалах и покупных изделиях.

Таблица 6.2 - Материалы и покупные изделия.

№ пп	Наименование материалов	Единицы измерения	Количество	Цена за единицу (руб)	Стоимость (руб)
1.	Флешка 64 Гб	шт.	1	840	840
2.	CD-диск	шт.	2	20	40
3.	Бумага А4	пачка	1	256	256
4.	Картридж для принтера HP LaserJet Professional P 1102w	шт.	2	958	1916
5.	Ручка	шт.	10	5	50
6.	Папка	шт.	1	160	160
7.	Брошюровка	До 150 лис.	450	450	450
<b>Итого материалов</b>					<b>3712</b>
<b>Транспортно-заготовительные расходы</b>					<b>750</b>
<b>Итого</b>					<b>4 462</b>

### 6.3.2 Специальное оборудование

По данной статье расходы отсутствуют.

### 6.3.3 Основная заработная плата

В таблице 6.3 произведен расчет основной заработной платы.

Таблица 6.3 - Расчёт основной заработной платы.

Исполнитель (должность)	Мес. оклад (руб)	Трудоемкость (чел/дни)	Оплата за день (руб)	Оплата за проект (руб)
Менеджер проекта	132 000	19	6 000	114 000
Системный администратор	60 000	8	2 727	21 816
Аналитик	80 000	33	3 636	119 988
Оператор страховой компании	80 000	15	3 636	54 540
Разработчик	80 000	13	3 636	47 268
<b>Итого</b>				<b>357 612</b>

### 6.3.4 Дополнительная заработная плата

Дополнительная заработная плата составляет 20-30% от суммы основной заработной платы.

$$\text{ДЗП} = 357\,612 * 0,2 = 74\,522 \text{ руб.}$$

### 6.3.5 Страховые отчисления

Страховые платежи составляют 30% от фонда оплаты труда (ФОТ), который состоит из основной и дополнительной заработной плат.  $\text{ФОТ} = \text{ОЗП} + \text{ДЗП} = 357\,612 + 74\,522 = 429\,134 \text{ руб.}$   $\text{СП} = 429\,134 * 0,3 = 128\,740 \text{ руб.}$  Таким образом, страховые отчисления составляют 128 470 руб.

### 6.3.6 Командировочные расходы

В рамках данного проекта командировочные расходы отсутствуют.

### 6.3.7 Контрагентские услуги

В процессе разработки данного проекта услуги сторонних организаций не использовались.

### 6.3.8 Накладные расходы

Накладные расходы определяется в размере 250% от основной заработной платы всех специалистов, задействованных в проекте.

$$\text{НР} = \text{ОЗП} * 2,5 = 357\,612 * 2,5 = 894\,030 \text{ руб.}$$

### 6.3.9 Прочие расходы

По данной статье затрат нет.

### 6.3.10 Полная себестоимость проекта

В таблице 6.4 указана полная себестоимость проекта.

Таблица 6.4 - Полная себестоимость проекта.

№	Номенклатура статей расходов	Затраты, руб.
	Материалы, покупные изделия и полуфабрикаты	4 462
	Специальное оборудование для научных работ	—
	Основная заработная плата персонала	357 612
	Дополнительная заработная плата персонала	74 522
	Страховые взносы в социальные фонды	128 470
	Расходы на научные и производственные командировки	—
	Оплата работ, выполненных сторонними организациями и предприятиями	—
	Прочие расходы	—
	Накладные расходы	894 030
	<b>Итого:</b>	<b>1 993 358</b>

### 6.3.11 Договорная цена

Цена договорная = себестоимость + прибыль + НДС

Норма прибыли составляет 20-30% от стоимости разработки.

$$П = 1\,459\,096 \cdot 0,3 = 437\,728$$

Реализованная ИС будет предназначена для коммерческой организации. Поэтому данный вид работ облагается налогом на добавочную стоимость в размере 20%:

$$\text{НДС} = (С + П) \cdot 20\% = (1\,459\,096 + 437\,728) \cdot 0,2 = 379\,364$$

Таким образом, договорная цена будет представлять собой:

$$\text{ДЦ} = С + П + \text{НДС} = 1\,459\,096 + 437\,728 + 379\,364 = 2\,276\,188$$

### Выводы

В рамках организационно-экономической части выпускной квалификационной работы был проведён анализ разрабатываемого продукта.

Длительность работы над проектом составила 58 дней. Разработка выполняется командой из пяти человек: менеджер проекта, аналитик, системный администратор, оператор страховой компании и разработчик.

Также были определены затраты и договорная цена разработки. Договорная цена разработки составила 2 276 188 рублей. Плановая прибыль от проекта составляет 437 728 рублей.

## ЗАКЛЮЧЕНИЕ

Целью данной работы являлась программная реализация интеллектуальной системы, повышающей эффективность обработки входящих заявок клиентов медицинской страховой компании, и автоматизация этого процесса.

Перед началом работы над реализацией проекта был проведён анализ предметной области.

В рамках достижения цели было разработано ТЗ и решены следующие задачи:

1. Загрузка исторических данных клиентов из файла;
2. Статистическая обработка загруженных данных;
3. Проектирование архитектуры нейронной сети;
4. Разработка модуля ИНС;
5. Разработка модуля тестирования ИНС;
6. Разработка модуля миграции баз данных;
7. Разработка модуля взаимодействия с базой данных;
8. Разработка модуля взаимодействия с веб-клиентом;
9. Разработка модуля доступа к системе;
10. Разработка модуля отчётов;
11. Разработка веб-клиента;
12. Реализация модульной ИС;
13. Определение решения по заявке системой;
14. Уведомление клиента об решении;
15. Просмотр заявок по клиенту;
16. Генерация отчёта по работе с клиентами.

Проведён расчёт надёжности по формуле Миллса, результаты которого показывают высокую безотказность работы разрабатываемой системы, и требуемого объема памяти на вычислительной машине, на которой будет запущено серверное приложение с ИС.



Разработано руководство пользователя.

Проведены организационно-экономические расчёты, в рамках которых определено следующее:

1. Группа людей, участвующих в реализации;
2. Затраты на реализацию;
3. Договорная цена разработки.

Таким образом, цель выпускной квалификационной работы достигнута – реализована система поддержки принятия решений обработки заявок клиентов страховой компании.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Заколюдажный Владимир Олегович Развитие скоринговых технологий в автостраховании // Евразийский Союз Ученых. 2015. №11-4 (20). URL: <https://cyberleninka.ru/article/n/razvitie-skoringovyh-tehnologiy-v-avtostrahovanii> (дата обращения: 11.05.2020).
2. Лапин Е. С. Компьютерная автоматизация бизнес-процессов // НиКа. 2006. №. URL: <https://cyberleninka.ru/article/n/kompyuternaya-avtomatizatsiya-biznes-protssessov> (дата обращения: 16.05.2020).
3. Патрикеева Я. А. Страховой рынок // Вологодские чтения. 2007. №62. URL: <https://cyberleninka.ru/article/n/strahovoy-rynok> (дата обращения: 13.05.2020).
4. Молчанов В.С. Роль страхования в экономике и финансовой системе страны // Juvenis scientia. 2016. №4. URL: <https://cyberleninka.ru/article/n/rol-strahovaniya-v-ekonomike-i-finansovoy-sisteme-strany> (дата обращения: 13.05.2020).
5. Яшина Нина Михайловна Финансы страховой компании // Базис. 2017. №1 (1). URL: <https://cyberleninka.ru/article/n/finansy-strahovoy-kompanii> (дата обращения: 29.05.2020).
6. Драчева В. И. Автоматизация бизнес-процессов // Актуальные проблемы гуманитарных и естественных наук. 2011. №7. URL: <https://cyberleninka.ru/article/n/avtomatizatsiya-biznes-protssessov> (дата обращения: 13.05.2020).
7. И. В. Дыбина, А. С. Славянов Искусственный интеллект как инструмент повышения эффективности и устойчивости бизнеса // Экономика и бизнес: теория и практика. 2019. №4-2. URL: <https://cyberleninka.ru/article/n/iskusstvennyy-intellekt-kak-instrument-povysheniya-effektivnosti-i-ustoychivosti-biznesa> (дата обращения: 14.05.2020).
8. Сорокина Юлия Александровна, Уткина Наталья Ивановна

- ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ В СОВРЕМЕННОМ МЕНЕДЖМЕНТЕ // Контентус. 2016. №1 (42). URL: <https://cyberleninka.ru/article/n/intellektualnye-sistemy-v-sovremennom-menedzhmente> (дата обращения: 29.05.2020).
9. Боровкова Александра Станиславовна, Казакова Герензел Яшкуловна, Эльдяева Дельгир Мингияновна, Кичикова Надежда Кануровна Цифровые инновации и особенности управления бизнесом // РППЭ. 2018. №11 (97). URL: <https://cyberleninka.ru/article/n/tsifrovye-innovatsii-i-osobennosti-upravleniya-biznesom> (дата обращения: 29.05.2020).
10. Загинайло М.В. Влияние количества нейронов искусственной нейронной сети на эффективность её работы // E-Scio. 2019. №6 (33). URL: <https://cyberleninka.ru/article/n/vliyanie-kolichestva-neuronov-iskusstvennoy-neyronnoy-seti-na-effektivnost-ee-raboty> (дата обращения: 18.05.2020).
11. Назаров Максим Николаевич Искусственная нейронная сеть с модуляцией коэффициентов синапсов // Вестн. Сам. гос. техн. ун-та. Сер.: Физ.-мат. науки. 2013. №2 (31). URL: <https://cyberleninka.ru/article/n/iskusstvennaya-neyronnaya-set-s-modulyatsiey-koeffitsientov-sinapsov> (дата обращения: 29.05.2020).
12. Сикорский О.С. Обзор свёрточных нейронных сетей для задачи классификации изображений // Новые информационные технологии в автоматизированных системах. 2017. №20. URL: <https://cyberleninka.ru/article/n/obzor-svyortochnyh-neyronnyh-setey-dlya-zadachi-klassifikatsii-izobrazheniy> (дата обращения: 19.05.2020).
13. Видмант Олег Сергеевич Прогнозирование финансовых временных рядов с использованием рекуррентных нейронных сетей LSTM // Общество: политика, экономика, право. 2018. №5. URL: <https://cyberleninka.ru/article/n/prognozirovanie-finansovyh-vremennyh-ryadov-s-ispolzovaniem-rekurrentnyh-neyronnyh-setey-lstm> (дата обращения: 23.05.2020).
14. Чижков Александр Васильевич, Сеитова Светлана Владимировна

- Классификация нейросетевых архитектур // Известия ЮФУ. Технические науки. 2009. №4. URL: <https://cyberleninka.ru/article/n/klassifikatsiya-neyrosetevyh-arhitektur> (дата обращения: 29.05.2020).
15. Дегтярев Евгений Андреевич, Карякин Александр Ливиевич Влияние вида функций активации нейронов на относительную ошибку прогнозирования электропотребления // Известия УГГУ. 2013. №2 (30). URL: <https://cyberleninka.ru/article/n/vliyanie-vida-funktsiy-aktivatsii-neuronov-na-otnositelnuyu-oshibku-prognozirovaniya-elektropotrebleniya> (дата обращения: 29.05.2020).
16. Сорокин Алексей Борисович Интеллектуальные системы: Лекция Понятие о математическом нейроне для студентов, обучающихся по направлению 09.03.04 «Программная инженерия» по профилю бакалавры. — М., Московский технологический университет (МИРЭА), 2020 — 13 с.
17. В. Н. Зуев, В. К. Кемайкин Модифицированный алгоритм обучения нейронных сетей // Программные продукты и системы. 2019. №2. URL: <https://cyberleninka.ru/article/n/modifitsirovannyi-algoritm-obucheniya-neyronnyh-setey> (дата обращения: 29.05.2020).
18. Шестопад Е.А., Панченко В.М. Выбор эффективного инструмента проектирования программного обеспечения CASE-средств // Перспективы развития информационных технологий. 2016. №28. URL: <https://cyberleninka.ru/article/n/vybor-effektivnogo-instrumenta-proektirovaniya-programmnogo-obespecheniya-case-sredstv> (дата обращения: 29.05.2020).
19. Spring Framework [Электронный ресурс]. URL: <https://spring.io/projects/spring-framework>. (Дата обращения 10.04.19)

## ПРИЛОЖЕНИЕ

Программный код:

**//Главный класс приложения**

@SpringBootApplication

class Application

fun main(args: Array<String>) {

    runApplication<Application>(\*args)

}

**//Сервис обработки клиентов**

@Service

class PolicyHolderServiceImpl(

    private val policyHolderRepository: PolicyHolderRepository,

    private val neuralNetwork: NeuralNetwork

): PolicyHolderService {

    override fun save(request: PolicyHolderCreateRequest): PolicyHolder {

        val decision = neuralNetwork.calculateDecision(PolicyHolderMapper.toDataMap(request))

        return policyHolderRepository.save(PolicyHolderMapper.toEntity(request, decision))

    }

}

**//Модуль чтения/записи в БД**

interface PolicyHolderRepository: JpaRepository<PolicyHolder, UUID> {

}

**//Модель клиента**

@Entity

data class PolicyHolder (

    @Id

    val id: UUID = UUID.randomUUID(),

    val name: String? = null,

    val age: Int,

    val sex: Int,

    val educationDegree: Int,

    val salary: Double,

    val possibilityOfInjuryAtJob: Double,

    val decision: Double

)

**//Модель запроса на заявку**

data class PolicyHolderCreateRequest (

    val name: String? = null,

    val age: Int,

    val sex: Int,

    val educationDegree: Int,

    val salary: Double,

    val possibilityOfInjuryAtJob: Double

)

**//Модель ответа на запрос**

data class PolicyHolderResponse (

```

        val id: UUID = UUID.randomUUID(),
        val name: String? = null,
        val age: Int,
        val sex: Int,
        val educationDegree: Int,
        val salary: Double,
        val possibilityOfInjuryAtJob: Double
    )

//Класс для преобразования моделей
class PolicyHolderMapper {
    companion object {
        fun toEntity(request: PolicyHolderCreateRequest, decision: Double): PolicyHolder {
            return PolicyHolder(
                name = request.name,
                age = request.age,
                sex = request.sex,
                educationDegree = request.educationDegree,
                salary = request.salary,
                possibilityOfInjuryAtJob = request.possibilityOfInjuryAtJob,
                decision = decision
            )
        }

        fun toDataMap(request: PolicyHolderCreateRequest): LinkedList<Double> {
            val dataList = LinkedList<Double>()
            dataList.add(request.age.toDouble())
            dataList.add(request.educationDegree.toDouble())
            dataList.add(request.possibilityOfInjuryAtJob)
            dataList.add(request.salary)
            dataList.add(request.sex.toDouble())
            return dataList
        }

        fun toDto(policyHolder: PolicyHolder): PolicyHolderResponse {
            return PolicyHolderResponse(
                id = policyHolder.id,
                name = policyHolder.name,
                age = policyHolder.age,
                sex = policyHolder.sex,
                educationDegree = policyHolder.educationDegree,
                salary = policyHolder.salary,
                possibilityOfInjuryAtJob = policyHolder.possibilityOfInjuryAtJob
            )
        }
    }
}

```

```

//Нейрон
class Neuron (
    var weightList: LinkedList<Double>,
    var weightDeltaList: LinkedList<Double>,

```

```

        var output: Double = 0.0
    ) {
        fun activation(inputList: LinkedList<Double>) {
            output = sigmoid(sum(inputList))
        }
        private fun sigmoid(x: Double): Double {
            return 1 / (1 + exp(-x))
        }
        private fun sum(inputList: List<Double>): Double {
            var sum = 0.0
            for (value in inputList) sum += value
            return sum
        }
    }
}

//Нейронная сеть
class NeuralNetwork(
    private val inputLayer: InputLayer,
    private val hiddenLayers: List<HiddenLayer>,
    private val outputLayer: OutputLayer
) {
    private val logger = LoggerFactory.getLogger(this::class.java)

    companion object {
        private const val e = 0.2
        private const val a = 0.3
    }

    private var weights = LinkedList<DoubleArray>()
    private var mse: Double = 0.0
    fun error(n: Int): Double {
        val error = mse / n * 100
        return BigDecimal(error).setScale(1, RoundingMode.HALF_DOWN).toDouble()
    }
    fun initWeight() {
        val inputSynapses = getSynapseCountOfInputLayer()
        val inputWeights = generateStartUpWeights(inputSynapses)
        inputLayer.init(inputWeights)
        weights.add(inputWeights)
        initHiddenLayers()
        weights.add( DoubleArray(outputLayer.countOfNeurons) { 1.0 } )
        outputLayer.init(weights.last)
    }
    fun calculateDecision(inputData: LinkedList<Double>): Double {
        val result = inputLayer.start(inputData)
        val nextLayerResult = if (hiddenLayers.isEmpty()) {
            result
        } else {
            startAllHiddenLayers(result, inputLayer.neuronList.size)
        }
        val lastLayerSize = if (hiddenLayers.isEmpty()) {
            inputLayer.neuronList.size
        }
    }
}

```

```

    } else {
        hiddenLayers.last().neuronList.size
    }
    return outputLayer.start(nextLayerResult, lastLayerSize).first()
}
fun learn(countOfIterations: Int, dataList: List<GeneratedRow>) {
    for (i in 0 until countOfIterations) {
        for (data in dataList) {
            think(data.data, data.decision)
        }
        val error = error(dataList.size)
        logger.info("Ошибка ~ $error за ${i + 1} эру")
        finishEra()
    }
    logger.info("Обучение завершено")
}
private fun think(data: LinkedList<Double>, ideal: Double): Double {
    val result = calculateDecision(data)
    correctWeightsOnLearning(result, ideal)
    mse += (ideal - result) * (ideal - result)
    return result
}

private fun correctWeightsOnLearning(result: Double, ideal: Double) {
    val deltaList = LinkedListmutableListOf(outputLayer.delta(result, ideal)))
    val deltaListFromHiddenLayers = when {
        hiddenLayers.size == 1 -> hiddenLayers.first().backPropagation(e, a, deltaList)
        hiddenLayers.size > 1 -> doBackPropagationOnAllHiddenLayers(deltaList)
        else -> deltaList
    }
    inputLayer.backPropagation(e, a, deltaListFromHiddenLayers)
}
private fun finishEra() {
    mse = 0.0
}
private fun getSynapseCountOfInputLayer(): Int {
    return if (hiddenLayers.isEmpty()) {
        inputLayer.countOfNeurons * outputLayer.countOfNeurons
    } else {
        inputLayer.countOfNeurons * hiddenLayers.first().countOfNeurons
    }
}
private fun generateStartUpWeights(countOfSynapses: Int): DoubleArray {
    val synapsesWeightArray = DoubleArray(countOfSynapses)
    val random = Random()
    for (i in 0 until countOfSynapses) {
        val minus = random.nextBoolean()
        var generatedValue = random.nextDouble() % 2
        if (minus && random.nextBoolean()) {
            generatedValue *= -1.0
        }
        synapsesWeightArray[i] = generatedValue
    }
}

```



```

    }
    return synapsesWeightArray
}
private fun doBackPropagationOnAllHiddenLayers(deltaListFromOutputLayer:
LinkedList<Double>): MutableList<Double> {
    var deltaListFromPreviousHiddenLayer = hiddenLayers.last().backPropagation(e, a,
deltaListFromOutputLayer)
    var isFirstIteration = true
    for (i in hiddenLayers.size - 1 downTo 0) {
        if (isFirstIteration) {
            isFirstIteration = false
            continue
        }
        deltaListFromPreviousHiddenLayer = hiddenLayers[i].backPropagation(e, a,
deltaListFromPreviousHiddenLayer)
    }
    return deltaListFromPreviousHiddenLayer
}
private fun startAllHiddenLayers(inputLayerOutputData: LinkedList<Double>,
sizeOfInputLayer: Int): LinkedList<Double> {
    return when {
        hiddenLayers.isEmpty() -> inputLayerOutputData
        hiddenLayers.size == 1 -> {
            hiddenLayers.first().start(inputLayerOutputData, sizeOfInputLayer)
        }
        else -> {
            var outputDataFromPreviousHiddenLayer =
hiddenLayers.first().start(inputLayerOutputData, sizeOfInputLayer)
            var sizeOfPreviousHiddenLayer = hiddenLayers.first().neuronList.size
            for (i in 1 until (hiddenLayers.size - 2) ) {
                outputDataFromPreviousHiddenLayer =
hiddenLayers[i].start(outputDataFromPreviousHiddenLayer, sizeOfPreviousHiddenLayer)
                sizeOfPreviousHiddenLayer = hiddenLayers[i].neuronList.size
            }
            hiddenLayers.last().start(outputDataFromPreviousHiddenLayer,
sizeOfPreviousHiddenLayer)
        }
    }
}

private fun initHiddenLayers() {
    if (hiddenLayers.size == 1) {
        val outputNeurons = outputLayer.countOfNeurons
        val hiddenNeurons = hiddenLayers.first().countOfNeurons
        val initWeights = generateStartUpWeights(outputNeurons * hiddenNeurons)
        weights.add(initWeights)
        hiddenLayers.first().init(initWeights)
    }
    if (hiddenLayers.size > 1) {
        val hiddenNeurons = hiddenLayers.first().countOfNeurons
        val nextHiddenLayer = hiddenLayers[1].countOfNeurons
        val initWeights = generateStartUpWeights(nextHiddenLayer * hiddenNeurons)
    }
}

```

```

        weights.add(initWeights)
        hiddenLayers.first().init(initWeights)
        for (i in 1..hiddenLayers.size - 2) {
            val countOfNeuronsOnCurrentLayer = hiddenLayers[i].countOfNeurons
            val countOfNeuronsOnNextLayer = hiddenLayers[i + 1].countOfNeurons
            val initWeightsOnLoop = generateStartUpWeights(countOfNeuronsOnCurrentLayer *
countOfNeuronsOnNextLayer)
            weights.add(initWeightsOnLoop)
            hiddenLayers[i].init(initWeightsOnLoop)
        }
        val countOfNeuronsOnLastHiddenLayer = hiddenLayers.last().countOfNeurons
        val countOfNeuronsOnOutputLayer = outputLayer.countOfNeurons
        val
            initWeightsOnLastHiddenLayer
generateStartUpWeights(countOfNeuronsOnLastHiddenLayer
countOfNeuronsOnOutputLayer)
        weights.add(initWeightsOnLastHiddenLayer)
        hiddenLayers.last().init(initWeightsOnLastHiddenLayer)
    }
}
fun testLearning(data: LinkedList<Double>, ideal: Double, errorMinimum: Double) {
    val result = think(data, ideal)
    val error = error(1)
    if (error > errorMinimum) {
        logger.error("Неудачное обучение нейронной сети, высокая ошибка $error при
результате $result и идеальном ответе $ideal")
        throw ValidationException("Неудачное обучение нейронной сети, высокая ошибка
$error")
    }
}
}
}

```

#### **//Исходящий слой**

```

class OutputLayer(countOfNeurons: Int): LayerImpl(countOfNeurons) {
    fun delta(result: Double, ideal: Double): Double {
        return (ideal - result) * ((1 - result) * result)
    }
}

```

#### **//Класс слоя, от которого наследуются отдельные виды слоёв**

```

open class LayerImpl (
    val countOfNeurons: Int
): Layer {
    var neuronList = LinkedList<Neuron>()

    fun init(weights: DoubleArray) {
        val countOfConnections = weights.size / countOfNeurons
        for (i in 0 until countOfNeurons) {
            val weightDeltaList = setWeightDeltaListZeroValues(countOfConnections)
            val weightList = setWeightListZeroValues(countOfConnections, weights, i)
            neuronList.add(Neuron(weightList, weightDeltaList))
        }
    }
}

```

```

override fun start(inputList: LinkedList<Double>): LinkedList<Double> {
    val nextLayerInput = LinkedList<Double>()
    for (i in 0 until neuronList.size) {
        val neuron = neuronList[i]
        neuron.output = inputList[i]
        val list = jump(neuron)
        nextLayerInput.addAll(list)
    }
    return nextLayerInput
}

override fun start(inputList: LinkedList<Double>, previousLayerSize: Int):
LinkedList<Double> {
    val inputNum = inputList.size / previousLayerSize
    val nextLayerInputList = LinkedList<Double>()
    for (i in 0 until neuronList.size) {
        val neuron = neuronList[i]
        val newInputList = LinkedList<Double>()
        var j = 0
        while (j < inputList.size) {
            nextLayerInputList.add(inputList[i + j])
            j += inputNum
        }
        neuron.activation(newInputList)
        val jump = jump(neuron)
        nextLayerInputList.addAll(jump)
    }
    return nextLayerInputList
}

protected fun jump(neuron: Neuron): List<Double> {
    val weightList = neuron.weightList
    val output = neuron.output
    val nextInput = mutableListOf<Double>()
    for (weight in weightList) {
        nextInput.add(weight * output)
    }
    return nextInput
}

fun weightUpdate(neuron: Neuron, e: Double, a: Double, deltaList: List<Double>) {
    for (i in neuron.weightList.indices) {
        val gradient = gradient(neuron.output, deltaList[i])
        val newWeightDelta = e * gradient + a * neuron.weightDeltaList[i]
        val newWeight = neuron.weightList[i] + newWeightDelta
        neuron.weightDeltaList[i] = newWeightDelta
        neuron.weightList[i] = newWeight
    }
}

private fun gradient(output: Double, delta: Double) = output * delta

```

```

private fun setWeightDeltaListZeroValues(countOfConnections: Int): LinkedList<Double> {
    val list = LinkedList<Double>()
    for (i in 0 until countOfConnections) {
        list.add(0.0)
    }
    return list
}

private fun setWeightListZeroValues(countOfConnections: Int, weights: DoubleArray,
neuronIndex: Int): LinkedList<Double> {
    val list = LinkedList<Double>()
    for (i in 0 until countOfConnections) {
        list.add(weights[neuronIndex * countOfConnections + i])
    }
    return list
}
}

```

### **//Интерфейс слоя**

```

interface Layer {
    fun start(inputList: LinkedList<Double>): LinkedList<Double>

    fun start(inputList: LinkedList<Double>, previousLayerSize: Int): LinkedList<Double>
}

```

### **//Входящий слой**

```

class InputLayer(countOfNeurons: Int): LayerImpl(countOfNeurons) {
    fun backPropagation(e: Double, a: Double, deltaList: List<Double>) {
        for (neuron in neuronList) {
            weightUpdate(neuron, e, a, deltaList)
        }
    }

    override fun start(inputList: LinkedList<Double>): LinkedList<Double> {
        val nextLayerInput = LinkedList<Double>()
        for (i in neuronList.indices) {
            val neuron = neuronList[i]
            neuron.activation(LinkedList(mutableListOf(inputList[i])))
            val list = jump(neuron)
            nextLayerInput.addAll(list)
        }
        return nextLayerInput
    }
}

```

### **//Скрытый слой**

```

class HiddenLayer(countOfNeurons: Int): LayerImpl(countOfNeurons) {
    private fun delta(neuron: Neuron, delta: List<Double>): Double {
        val weightList: List<Double> = neuron.weightList
        val output: Double = neuron.output
        var sum = 0.0
    }
}

```

```

        for (i in weightList.indices) sum += weightList[i] * delta[i]
        return (1 - output) * output * sum
    }
    fun backPropagation(e: Double, a: Double, deltaList: List<Double>): LinkedList<Double> {
        val currentDeltaList = LinkedList<Double>()
        for (neuron in this.neuronList) {
            weightUpdate(neuron, e, a, deltaList)
            val delta = delta(neuron, deltaList)
            currentDeltaList.add(delta)
        }
        return currentDeltaList
    }
}

```

#### **//Реализация паттерна «Строитель»**

```

class NeuralNetworkBuilder (
    private val countOfNeuronsInputLayer: Int,
    private val countOfNeuronsByHiddenLayers: List<Int>,
    private val countOfNeuronsOnOutputLayer: Int
): Builder {

    override fun getInputLayer(countOfNeurons: Int): InputLayer {
        return InputLayer(countOfNeurons)
    }

    override fun getHiddenLayers(countOfNeuronsByLayer: List<Int>): List<HiddenLayer> {
        return LinkedList(countOfNeuronsByLayer.map { HiddenLayer(it) }.toMutableList())
    }

    override fun getOutputLayer(countOfNeurons: Int): OutputLayer {
        return OutputLayer(countOfNeurons)
    }

    fun build(): NeuralNetwork {
        return NeuralNetwork(
            inputLayer = getInputLayer(countOfNeuronsInputLayer),
            hiddenLayers = getHiddenLayers(countOfNeuronsByHiddenLayers),
            outputLayer = getOutputLayer(countOfNeuronsOnOutputLayer)
        )
    }
}

```

#### **//Интерфейс сборщика нейронных сетей**

```

interface Builder {
    fun getInputLayer(countOfNeurons: Int): InputLayer

    fun getHiddenLayers(countOfNeuronsByLayer: List<Int>): List<HiddenLayer>

    fun getOutputLayer(countOfNeurons: Int): OutputLayer
}

```

#### **//Модель клиента для преобразования в массив**

```

data class PolicyHolderDto (

```

```

        val age: Int,
        val sex: Int,
        val educationDegree: Int,
        val salary: Double,
        val possibilityOfInjuryAtJob: Double
    )

    //Сгенерированная запись для обучения
    data class GeneratedRow(
        val data: LinkedList<Double>,
        val decision: Double
    ) {
        override fun toString(): String {
            val builder = StringBuilder()
            builder.append(data.toString()).append(" Решение по заявке: ").append(decision)
            return builder.toString()
        }
    }

    //Генератор записей
    class Generator (
        private val count: Long,
        private val percentOfApproved: Double
    ) {
        private val logger = LoggerFactory.getLogger(this::class.java)
        private val generatedData: MutableList<GeneratedRow> = ArrayList()

        fun doGeneration(): List<GeneratedRow> {
            generatedData.addAll(generateApprovedHolders())
            generatedData.addAll(generateRejectedHolders())
            generatedData.shuffle()
            generatedData.forEach { logger.info("Сгенерированная запись: $it") }
            logger.info("Заявок сгенерировано всего: ${generatedData.size}")
            return generatedData
        }

        private fun generateApprovedHolders(): List<GeneratedRow> {
            val countOfApproved = (count * percentOfApproved).toLong()
            logger.info("Количество одобренных заявок: $countOfApproved")
            return generate(countOfApproved, true)
        }

        private fun generateRejectedHolders(): List<GeneratedRow> {
            val countOfRejected = (count - count * percentOfApproved).toLong()
            logger.info("Количество не одобренных заявок: $countOfRejected")
            return generate(countOfRejected, false)
        }

        private fun generate(count: Long, isApproved: Boolean): List<GeneratedRow> {
            val holderList = mutableListOf<GeneratedRow>()
            for (i in 0 until count) {
                val rnd = Random()

```

```

var possibility: Double
var educationDegree: Int
var age: Int
var salary: Long
var decision: Double
if (isApproved) {
    possibility = 0.05 + rnd.nextDouble() % 0.1
    educationDegree = 2 + abs(rnd.nextInt()) % 3
    age = 25 + abs(rnd.nextInt()) % 11
    salary = 10000 + abs(rnd.nextInt()) % 10001.toLong()
    decision = 1.0
} else {
    possibility = 0.2 + rnd.nextDouble() % 0.1
    educationDegree = 1 + abs(rnd.nextInt()) % 2
    age = 35 + abs(rnd.nextInt()) % 51
    salary = 2000 + abs(rnd.nextInt()) % 8001.toLong()
    decision = 0.0
}
val sex = abs(rnd.nextInt()) % 2
val policyHolder = PolicyHolderDto(age, sex, educationDegree, salary.toDouble(),
roundDouble(possibility))
holderList.add(GeneratedRow(toDataMap(policyHolder), decision))
}
return holderList
}

private fun roundDouble(d: Double): Double {
    val intPossibility = (d * 100).toInt()
    return intPossibility.toDouble() / 100
}

private fun toDataMap(dto: PolicyHolderDto): LinkedList<Double> {
    val dataList = LinkedList<Double>()
    dataList.add(dto.age.toDouble())
    dataList.add(dto.educationDegree.toDouble())
    dataList.add(dto.possibilityOfInjuryAtJob)
    dataList.add(dto.salary)
    dataList.add(dto.sex.toDouble())
    return dataList
}
}

```

**//Класс для перехвата ошибок**

```

@RestController
@ControllerAdvice
class ExceptionHandler : ResponseEntityExceptionHandler() {

    private val log = LoggerFactory.getLogger(this::class.java)

    @ExceptionHandler(ResourceNotFoundException::class)
    fun handleResourceNotFoundException(ex: ResourceNotFoundException, request:
WebRequest): ResponseEntity<Any> {

```

```

        return buildResponse(ex, HttpStatus.PRECONDITION_FAILED, request)
    }

    @ExceptionHandler(ValidationException::class)
    fun handleValidationFailedException(ex: ValidationException, request: WebRequest):
    ResponseEntity<Any> {
        return buildResponse(ex, HttpStatus.PRECONDITION_FAILED, request)
    }

    @ExceptionHandler(Exception::class)
    fun handleAllExceptions(ex: Exception, request: WebRequest): ResponseEntity<Any> {
        return buildResponse(ex, HttpStatus.INTERNAL_SERVER_ERROR, request)
    }

    private fun buildResponse(ex: Exception, status: HttpStatus, request: WebRequest):
    ResponseEntity<Any> {
        return buildMessage(ex)
            .also { log.error(it, ex) }
            .let { CustomExceptionResponse(Date(), it, request.getDescription(false)) }
            .let { ResponseEntity(it, status) }
    }

    private fun buildMessage(ex: Exception) =
        ex.message ?: CustomExceptionResponse.MESSAGE_UNDEFINED

    override fun handleMethodArgumentNotValid(ex: MethodArgumentNotValidException,
    headers: HttpHeaders, status: HttpStatus, request: WebRequest): ResponseEntity<Any> {
        val body = mutableMapOf<String, Any>()
        body["timestamp"] = LocalDateTime.now()
        body["status"] = status.value()
        body["errors"] = ex.bindingResult.fieldErrors.map { it.defaultMessage }
        return ResponseEntity(body, headers, status)
    }
}

//Модуль для работы с веб-клиентом
@CrossOrigin
@RestController
@RequestMapping(path = ["/v1/policy-holder"], produces =
[MediaType.APPLICATION_JSON_UTF8_VALUE])
class PolicyHolderController(
    val policyHolderService: PolicyHolderService
) {

    @PostMapping
    fun save(@RequestBody request: PolicyHolderCreateRequest): PolicyHolderResponse {
        return policyHolderService.save(request).let { PolicyHolderMapper.toDto(it) }
    }
}

//Конфигурация доступа к консоли БД
@Configuration

```



```

class WebConfig {
    @Bean
    fun h2servletRegistration(): ServletRegistrationBean<*>? {
        val registrationBean: ServletRegistrationBean<*> = ServletRegistrationBean(WebServlet())
        registrationBean.addUrlMappings("/console/*")
        return registrationBean
    }
}

```

#### **//Конфигурация модуля доступа**

```

@Configuration
class SecurityConfig: WebSecurityConfigurerAdapter() {
    override fun configure(http: HttpSecurity) {
        http.authorizeRequests().antMatchers("/").permitAll().and()
            .authorizeRequests().antMatchers("/console/**").permitAll()
        http.csrf().disable()
        http.headers().frameOptions().disable()
    }
}

```

#### **//Конфигурация нейронной сети**

```

@Configuration
class NeuralNetworkConfig {

    private val logger = LoggerFactory.getLogger(this::class.java)

    @Bean
    fun neuralNetwork(): NeuralNetwork {
        val dataSet = Generator(10000, 0.65).doGeneration()
        val neuralNetwork = NeuralNetworkBuilder(5, listOf(), 1).build()
        neuralNetwork.initWeight()
        neuralNetwork.learn(5, dataSet)
        neuralNetwork.testLearning(dataSet.first { it.decision == 0.0 }.data, 0.0, 20.0)
        neuralNetwork.testLearning(dataSet.first { it.decision == 1.0 }.data, 1.0, 20.0)
        logger.info("Нейронная сеть успешно обучилась")
        return neuralNetwork
    }
}

```

#### **//Конфигурация модуля миграции БД**

```

@Configuration
class LiquibaseConfig {

    @Autowired
    lateinit var dataSource: DataSource

    @Bean
    fun liquibase(): SpringLiquibase {
        val liquibase = SpringLiquibase()
        liquibase.changeLog = "classpath:/iaps/liquibase-changelog.xml"
        liquibase.dataSource = dataSource
        liquibase.defaultSchema = dataSource.connection.schema
    }
}

```

```
liquibase.isDropFirst = false
liquibase.liquibaseSchema = dataSource.connection.schema
return liquibase
}
}
```