

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук  
Кафедра программирования и информационных технологий

Приложение для ведения учета телефонных номеров  
PhoneGuard

Курсовой проект  
09.03.04 Программная инженерия  
Профиль «Информационные системы и сетевые технологии»

Зав. кафедрой \_\_\_\_\_ С.Д. Махортов д.ф.-м.н., профессор \_\_.\_\_.2024

Обучающийся \_\_\_\_\_ М.М.З. Насайр

Руководитель \_\_\_\_\_ А.А. Вахтин к.ф.м.н., доц.

Воронеж 2024

## Содержание

Введение .....	4
1. Постановка задачи .....	5
1.1 Проектирование приложения .....	5
1.2 Анализ предметной области .....	5
2. Постановка задачи .....	6
2.1 Требования к приложению .....	6
2.2 Диаграмма последовательности .....	7
2.3 Проектирование интерфейса .....	9
3. Реализация приложения .....	10
3.2 Средства реализации приложения .....	12
3.3 Диаграмма классов .....	13
3.3.1 Класс Contact (Контакт): .....	14
3.3.2 Класс PhoneGuardService (Сервис PhoneGuard): .....	15
3.3.3 Класс DatabaseConnection (Подключение к базе данных): .....	15
3.4 Реализация работы нейронной сети .....	15
3.4.1 Подготовка данных: .....	16
3.4.2 Обучение нейронной сети: .....	16
3.4.3 Интеграция с веб-приложением: .....	16
3.4.4 Разработка JSP страниц: .....	16
3.4.5 Тестирование и оптимизация: .....	16
3.4.6 Деплой приложения: .....	16
3.5 Реализация приложения .....	17
3.5.1 Настройка проекта: .....	17
3.5.2 Создание модели данных: .....	17
3.5.3 Создание Java Servlet: .....	17
3.5.4 Разработка JSP страниц: .....	17
3.5.5 Интеграция с базой данных: .....	17
3.5.6 Тестирование приложения: .....	18
3.5.7 Деплой приложения: .....	18
4. Тестирование приложения .....	18

4.1 Подготовка к тестированию:.....	18
4.2 Типы тестирования: .....	18
Заключение .....	20
Диаграмма состояний .....	20
Диаграмма последовательности .....	21
Диаграмма база данных .....	21
Диаграмма компонентов .....	22
Диаграмма использования .....	23
Приложение .....	23
ContactDAO class .....	23
UserDAO class .....	26

## **Введение**

В современном мире люди часто постоянно и постоянно работают с умными устройствами. Распознавание текста позволяет автоматизировать процесс анализа и обработки текстовых данных, что существенно экономит время и снижает риск ошибок. Таким образом, текстовый аналитик является незаменимым инструментом повышения эффективности и качества работы в различных сферах профессиональной деятельности. Цель работы – разработка приложения для распознавания и вывода читаемых данных. Для достижения этой цели необходимо определить следующие задачи:

- проведен обзор применяемых аналогов;
- Само приложение оформлено;
- Реализовано само приложение;
- Приложение проверено.
- Проверьте приложение на работу с разными языками.

Работа состоит из четырех разделов: введения, заключения и приложения.

Глава “проектирования приложения” включает в себя описание процесса создания концепции и архитектуры приложения.

В начале “главы проектирования приложения” определяются цели и задачи проекта. Затем осуществляется анализ существующих решений и конкурентов, чтобы выявить преимущества и недостатки существующих продуктов и понять, в чем может состоять уникальность и инновационность проекта.

Далее, в главе “проектирования приложения” создается общая концепция приложения. Это включает определение его основных функций, взаимодействия с пользователем и пользовательского интерфейса. Также в этой главе разрабатывается дизайн приложения, то есть внешний вид и интерфейс, который должен быть привлекательным, понятным и удобным для пользователей.

В главе “реализация приложения” рассмотрен процесс анализа и вывода данных, а также средства разработки.

Глава “тестирование приложения” посвящена тестированию, и результатам тестирования приложения.

В заключении сделаны выводы о проделанной работе.

## **1. Постановка задачи**

Целью данной выпускной квалификационной работы является разработка и выпуск самостоятельного приложения для сохранения номеров телефонов и сохранения данных.

К разрабатываемому приложению выдвинуты следующие требования:

- Приложение должно иметь возможность ввода данных на любом языке;
- Приложение должно иметь возможность распознавать вводимый текст.
- Приложение должно поддерживать распознавание текста на всех языках, включая английский, испанский, китайский, японский и т. д.;
- Приложение должно запускаться на любой системе пользователя;
- Приложение должно иметь небольшой размер и занимать минимум места на диске;
- Приложение должно быть совместимо с различными операционными системами, включая Windows, macOS и Linux;
- Представление данных должно быть хорошо организовано и отформатировано так, чтобы их было легко читать и просматривать;
- Приложение должно позволять пользователям настраивать параметры данных.

### **1.1 Проектирование приложения**

### **1.2 Анализ предметной области**

Важный этап в разработке любой программы, включающий исследование и понимание требований и контекста использования будущего приложения. В этой статье мы рассмотрим анализ предметной области

приложения PhoneGuard, созданного на Java с использованием JSP (JavaServer pages). Это приложение предназначено для управления контактами, предоставляя пользователю возможность добавлять, редактировать, удалять и просматривать записи.

## **2. Постановка задачи**

Целью данной выпускной квалификационной работы является разработка и выпуск самостоятельного приложения для сохранения номеров телефонов и сохранения данных.

К разрабатываемому приложению выдвинуты следующие требования:

- Приложение должно иметь возможность ввода данных на любом языке;
- Приложение должно иметь возможность распознавать вводимый текст.
- Приложение должно поддерживать распознавание текста на всех языках, включая английский, испанский, китайский, японский и т. д.;
- Приложение должно запускаться на любой системе пользователя;
- Приложение должно иметь небольшой размер и занимать минимум места на диске;
- Приложение должно быть совместимо с различными операционными системами, включая Windows, macOS и Linux;
- Представление данных должно быть хорошо организовано и отформатировано так, чтобы их было легко читать и просматривать;
- Приложение должно позволять пользователям настраивать параметры данных.

### **2.1 Требования к приложению**

Функциональные требования - это вид требований, описывающий поведение системы. В этом виде требований должен отражаться предоставляемый пользователю функционал.

Функциональные требования:

- ☐ Приложение должно обрабатывать любой текст;
- ☐ Приложение должно выводить результаты поиска и данные;
- ☐ Приложение должно своевременно и быстро выводить результат анализа;
- ☐ Приложение должно быть масштабируемым, должно быть готова к анализу большого количества текста;
- ☐ Приложение должно быть адаптировано к нескольким языкам;
- ☐ Приложение должно иметь понятный и простой для пользователя интерфейс;

Не функциональные требования:

- ☐ Приложение должно работать на всех платформах, таких как:
  1. Windows
  2. Linux
  3. Mac OS
- ☐ При первом запуске приложение необходимо создать свою учетную запись;
- ☐ Приложение должно иметь простой и понятный пользователю интерфейс;
- ☐ Приложение должно необходимо реализовать на языке программирования java, и запускаться в браузере.

## **2.2 Диаграмма последовательности**

В данной статье мы рассмотрим диаграмму взаимодействия компонентов веб-приложения " PhoneGuard", которая показывает процесс обработки запроса пользователя от открытия веб-страницы до отображения данных на экране браузера.

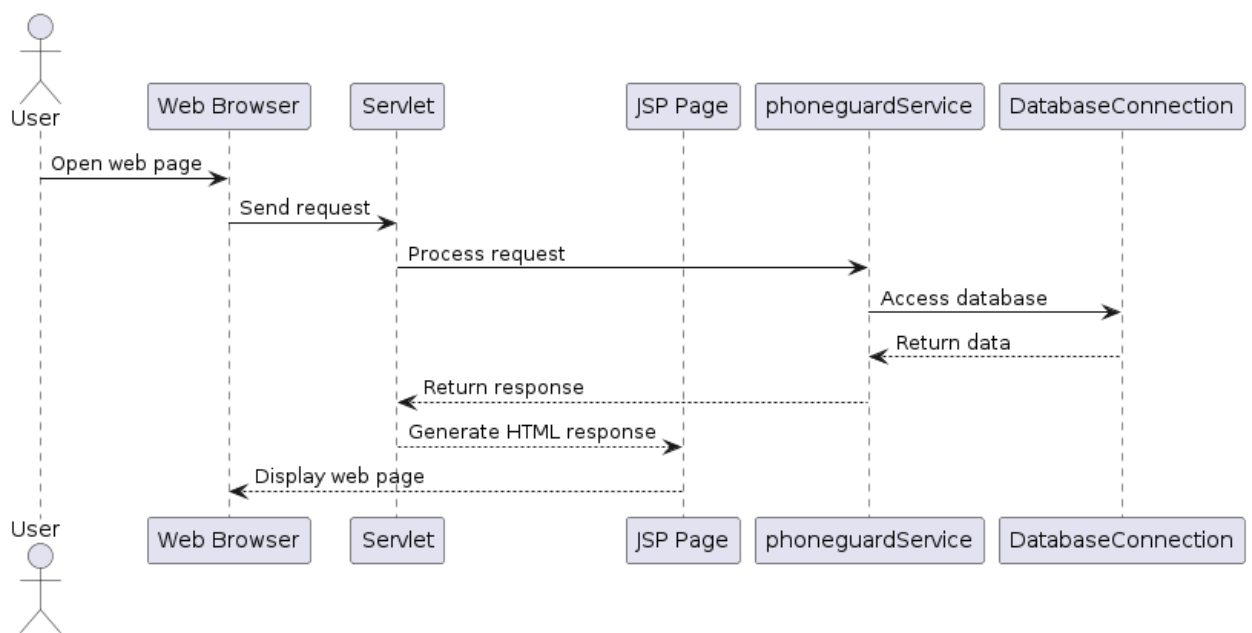


Рисунок 1- Диаграмма последовательности

Диаграмма создана с использованием языка UML и содержит следующие участники:

- User: Пользователь, который открывает веб-страницу приложения.
- Web Browser: Веб-браузер, через который пользователь взаимодействует с приложением.
- Servlet: Компонент, обрабатывающий запросы от клиента и управляющий бизнес-логикой приложения.
- JSP Page: JSP страница, генерирующая HTML ответ для отображения пользователю.
- PhoneGuardService: Сервис, предоставляющий функциональность по работе с данными PhoneGuard.
- DatabaseConnection: Компонент, обеспечивающий соединение с базой данных для доступа к информации.

Давайте разберем шаги взаимодействия компонентов по порядку:

- Пользователь открывает веб-страницу приложения "PhoneGuard" в своем веб-браузере.
- Веб-браузер отправляет запрос на сервер, где этот запрос



обрабатывается Servlet'ом.

- Servlet получает запрос и передает его PhoneGuardService для обработки. Service выполняет необходимые операции, например, получение данных из базы данных через DatabaseConnection.
- DatabaseConnection осуществляет доступ к базе данных и извлекает необходимую информацию.
- Полученные данные передаются обратно PhoneGuardService, который формирует ответ.
- Service отправляет ответ обратно Servlet'у, который затем передает управление JSP странице для генерации HTML ответа.
- JSP страница создает HTML содержимое на основе полученных данных и отправляет его обратно веб-браузеру.
- Веб-браузер отображает сгенерированную HTML страницу пользователю.

Эта диаграмма иллюстрирует сложный процесс взаимодействия между компонентами приложения "PhoneGuard" на Java Servlet и JSP. Она помогает понять каждый этап обработки запроса пользователя и показывает роль каждого компонента в этом процессе.

## 2.3 Проектирование интерфейса

Разработка интерфейса сайта "PhoneGuard" и создание аккаунта.



Рисунок 2 - главная страница

Форма <<главная страница>> - экран веб-приложения, отвечающий главным функционалом приложения.

Зарегистрируйте свою учетную запись образом, чтобы пользователь ввел свое имя, адрес электронной почты и пароль

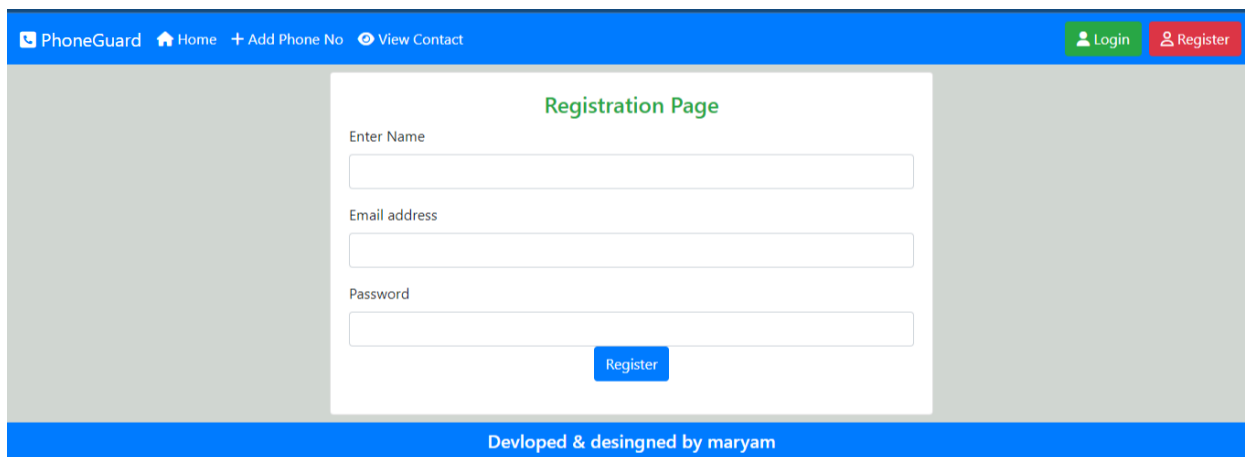


Рисунок 3 - Зарегистрировать аккаунт

### 3. Реализация приложения

#### 3.1 Анализ существующих решений для реализации на рынке

При разработке проекта PhoneGuard Java Web Project с использованием нескольких JSP, важным этапом является анализ существующих решений на рынке. Этот этап позволяет выявить преимущества и недостатки конкурентов, определить уникальные особенности и инновационные подходы, которые могут быть внедрены в разрабатываемый проект.

Одним из популярных решений на рынке является приложение "PhoneGuard Pro", которое предлагает широкий функционал для управления контактами. Оно обладает удобным интерфейсом, позволяющим быстро добавлять, редактировать и удалять контакты. Кроме того, "PhoneGuard Pro" поддерживает импорт и экспорт контактов в различные форматы, что обеспечивает удобство использования.

PhoneGuard Home + Add Phone No View Contact Login Register

### Login Page

Email address

Password

Login

Developed & desinged by maryam

Рисунок 4 - Войдите в аккаунт

PhoneGuard Home + Add Phone No View Contact Pabitra Das Logout

### Add Contact Page

Enter Name

Jageswar Kumar

Email address

jagi@gmail.com

Enter Phone No

3453524

Python Devloper

Update Contact

Activate Windows  
Go to Settings to activate Windows.

Рисунок 5 - Изменение данных

PhoneGuard Home + Add Phone No View Contact Pabitra Das Logout

Your Contact Updated..

**Name: Jageswar**  
Ph no: 345352434  
Email: jagi121@gmail.com  
About: Python Devloper  
Edit Delete

**Name: Rakesh Prajapati**  
Ph no: 453424323  
Email: rakesh@gmail.com  
About: Android Devloper  
Edit Delete

Activate Windows  
Go to Settings to activate Windows.

Рисунок 6 - Отображение данных после внесения изменений

Другим известным решением является приложение "Contact Manager", которое также предлагает широкий спектр функций для управления контактами. Однако "Contact Manager" отличается более сложным интерфейсом и менее интуитивной навигацией по приложению. Некоторые пользователи отмечают необходимость дополнительного времени на изучение функционала приложения.

При анализе существующих решений для реализации проекта PhoneGuard Java Web Project с использованием нескольких JSP, следует также обратить внимание на технические аспекты. Например, эффективное использование JavaServer Pages (JSP) для создания динамических веб-страниц может значительно повысить производительность приложения и обеспечить более гибкую архитектуру.

Важно также учитывать требования к безопасности данных при работе с личной информацией пользователей в приложении PhoneGuard. Разработка надежной системы аутентификации и авторизации, защита от SQL-инъекций и другие меры безопасности должны быть включены в процесс разработки.

Таким образом, проведенный анализ существующих решений на рынке позволяет выявить лучшие практики и определить ключевые особенности, которые могут быть применены при разработке проекта PhoneGuard Java Web Project с использованием нескольких JSP. Это поможет создать конкурентоспособное и инновационное приложение, отвечающее потребностям пользователей и требованиям рынка.

### **3.2 Средства реализации приложения**

Для создания PhoneGuard веб-приложения на языке Java с использованием нескольких JSP были выбраны следующие средства и библиотеки:

- Язык программирования Java: Используется для разработки бэкенда и бизнес-логики приложения.
- Java Servlets: Для обработки HTTP-запросов и реализации веб-приложения на Java.

- JavaServer Pages (JSP): Для создания динамических веб-страниц с использованием Java кода.
- Apache Tomcat: Контейнер сервлетов Java, используется для развертывания и запуска веб-приложения.
- HTML/CSS/JavaScript: Для создания пользовательского интерфейса и взаимодействия с пользователем.
- MySQL: Для хранения данных PhoneGuard, таких как имена, номера телефонов и другие контактные данные.
- JDBC (Java Database Connectivity): Для взаимодействия с базой данных MySQL из приложения на Java.
- Apache Maven: Для управления зависимостями и сборки проекта.
- Git: Для системы контроля версий, обеспечивающей отслеживание изменений в коде и совместную работу над проектом.
- IDE (Integrated Development Environment): Например, IntelliJ IDEA или Eclipse, для разработки и отладки кода.

### **3.3 Диаграмма классов**

Приготовьтесь окунуться в захватывающий мир баз данных и объектно-ориентированного проектирования! В этой статье мы рассмотрим диаграмму классов, описывающую веб-приложение для управления PhoneGuard. Диаграмма предоставляет детальное описание классов и их взаимосвязей, что позволяет нам глубже понять архитектуру приложения.

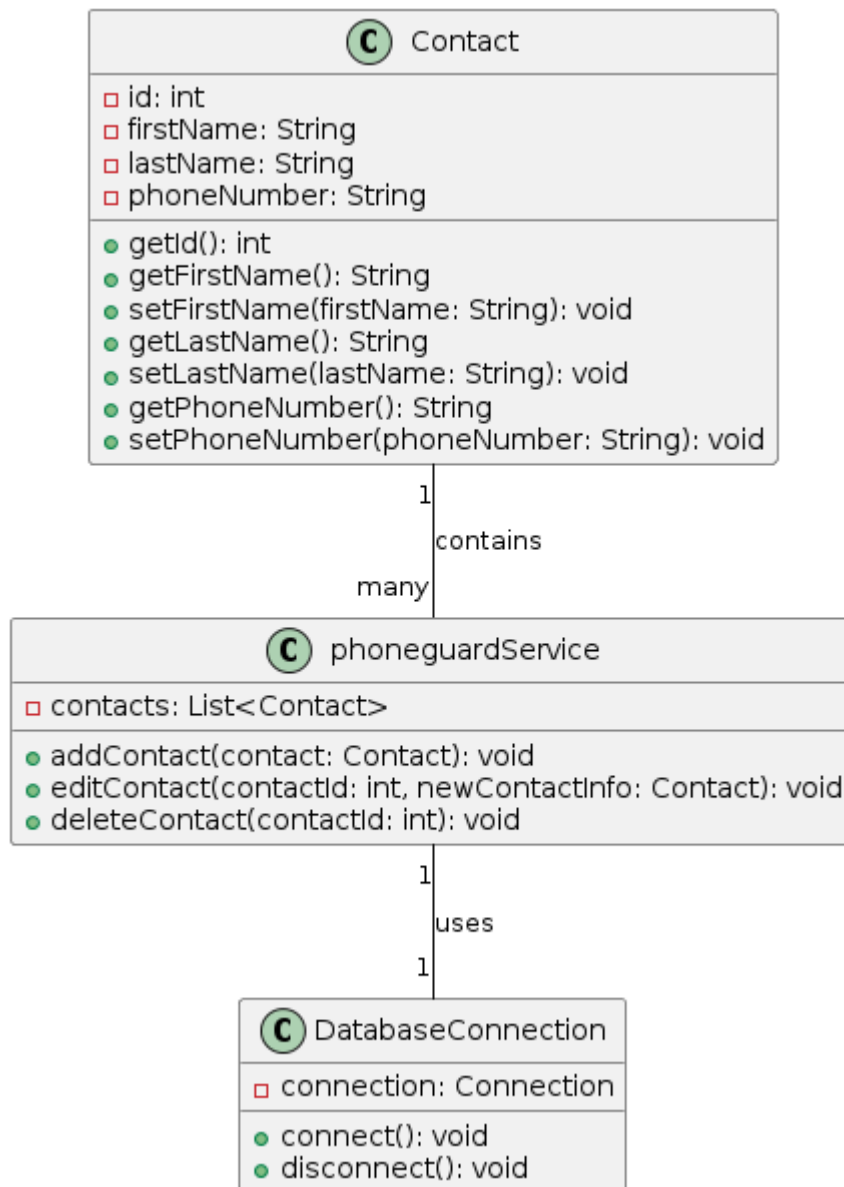


Рисунок 7 - Диаграмма классов

Диаграмма содержит три основных класса: Contact (Контакт), PhoneGuardService (Сервис PhoneGuard) и DatabaseConnection (Подключение к базе данных). Давайте рассмотрим каждый из них подробнее:

### 3.3.1 Класс Contact (Контакт):

`id`, `firstName`, `lastName`, `phoneNumber`: Приватные поля класса, представляющие идентификатор контакта, имя, фамилию и номер телефона соответственно.

`getId()`, `getFirstName()`, `setFirstName()`, `getLastName()`, `setLastName()`, `getPhoneNumber()`, `setPhoneNumber()`: Публичные методы доступа к полям класса, обеспечивающие безопасное чтение и изменение данных о контакте.

### **3.3.2 Класс PhoneGuardService (Сервис PhoneGuard):**

contacts: Приватное поле, представляющее список контактов, управляемый сервисом.

addContact(), editContact(), deleteContact(): Публичные методы для добавления, редактирования и удаления контактов из PhoneGuard.

### **3.3.3 Класс DatabaseConnection (Подключение к базе данных):**

connection: Приватное поле, представляющее подключение к базе данных.

connect(), disconnect(): Публичные методы для установки и разрыва соединения с базой данных.

Класс Contact содержит информацию о контакте, такую как имя, фамилия и номер телефона. Сервис PhoneGuardService управляет коллекцией контактов и предоставляет методы для их добавления, редактирования и удаления. Класс DatabaseConnection отвечает за установку и разрыв соединения с базой данных.

Отношения между классами также описаны на диаграмме:

PhoneGuardService содержит множество Contact, что означает, что каждый сервис PhoneGuard управляет несколькими контактами.

PhoneGuardService использует DatabaseConnection для доступа к базе данных, что позволяет сервису сохранять и извлекать данные о контактах.

Эта диаграмма помогает разработчикам понять структуру приложения и его компонентов, что упрощает процесс разработки, тестирования и сопровождения. Теперь, когда мы рассмотрели эту захватывающую диаграмму, вы готовы окунуться в мир разработки веб-приложений с PhoneGuard на языке Java!

## **3.4 Реализация работы нейронной сети**

В данной статье мы рассмотрим процесс интеграции нейронной сети в веб-приложение "PhoneGuard" на языке программирования Java с использованием технологии JavaServer Pages (JSP). Наша цель - создать

систему, которая будет использовать нейронную сеть для автоматического анализа и категоризации контактов в PhoneGuard.

Для начала определим основные шаги реализации данного проекта:

#### **3.4.1 Подготовка данных:**

Соберем и подготовим данные о контактах, которые будут использоваться для обучения нейронной сети. Эти данные могут включать в себя имена, номера телефонов, адреса и другие характеристики контактов.

#### **3.4.2 Обучение нейронной сети:**

Используем выбранную библиотеку или фреймворк для создания и обучения нейронной сети. Мы можем использовать различные алгоритмы машинного обучения, такие как нейронные сети глубокого обучения или классические алгоритмы классификации.

#### **3.4.3 Интеграция с веб-приложением:**

Создадим Java Servlet, который будет обрабатывать запросы от клиентов и передавать данные на обработку нейронной сети. Для отображения результатов используем несколько JSP страниц.

#### **3.4.4 Разработка JSP страниц:**

Напишем несколько JSP страниц для отображения интерфейса приложения, где пользователи смогут загружать свою PhoneGuard и получать результаты анализа от нейронной сети.

#### **3.4.5 Тестирование и оптимизация:**

Проведем тестирование работы нейронной сети в рамках веб-приложения, чтобы убедиться в корректности ее работы. Оптимизируем процессы обработки данных для повышения производительности.

#### **3.4.6 Деплой приложения:**

Запустим веб-приложение на сервере и предоставим доступ пользователям для использования функциональности анализа контактов через нейронную сеть.

Эти подробности пошагово описывают процесс интеграции нейронной сети в веб-приложение "PhoneGuard" на Java с использованием нескольких



JSP страниц. Хотя этот процесс может показаться сложным и требующим тщательного подхода, он открывает новые возможности для автоматизации анализа данных и повышения эффективности работы приложения. Важно следить за каждым этапом разработки и уделять внимание деталям, чтобы обеспечить успешное функционирование системы.

### **3.5 Реализация приложения**

В данной статье мы рассмотрим процесс разработки веб-приложения "PhoneGuard" на языке программирования Java с использованием технологии JavaServer Pages (JSP). Это приложение позволит пользователям управлять своими контактами и хранить информацию о них в удобном формате.

Для начала определим основные шаги реализации данного проекта:

#### **3.5.1 Настройка проекта:**

Создадим новый Java web project в выбранной IDE (например, IntelliJ IDEA или Eclipse) и добавим необходимые зависимости для работы с Servlet и JSP.

#### **3.5.2 Создание модели данных:**

Определим структуру данных для хранения информации о контактах, такую как имя, номер телефона, адрес и другие дополнительные поля.

#### **3.5.3 Создание Java Servlet:**

Напишем Servlet, который будет обрабатывать запросы от клиентов и управлять бизнес-логикой приложения. Servlet будет взаимодействовать с JSP страницами для генерации HTML ответов.

#### **3.5.4 Разработка JSP страниц:**

Создадим несколько JSP страниц для отображения интерфейса приложения. Например, страницу со списком контактов, форму добавления нового контакта и страницу редактирования контакта.

#### **3.5.5 Интеграция с базой данных:**

Напишем код для подключения к базе данных (например, MySQL или PostgreSQL) и выполнения операций чтения и записи данных о контактах.

### **3.5.6 Тестирование приложения:**

Проведем тестирование функциональности приложения, убедившись, что все операции работают корректно и данные сохраняются в базе данных.

### **3.5.7 Деплой приложения:**

Запустим веб-приложение на сервере (например, Apache Tomcat) для доступа к нему через веб-браузер.

Эти скучные подробности пошагово описывают процесс разработки веб-приложения "PhoneGuard" на Java с использованием нескольких JSP страниц. Хотя этот процесс может показаться стандартным и рутинным, он является ключевым для создания функционального и удобного приложения для пользователей. Важно следить за каждым этапом разработки и уделять внимание деталям, чтобы обеспечить успешное завершение проекта.

## **4. Тестирование приложения**

В данной статье мы рассмотрим процесс тестирования веб-приложения "PhoneGuard", разработанного на языке программирования Java с использованием технологии JavaServer Pages (JSP). Тестирование играет важную роль в обеспечении качества приложения и его корректной работы, поэтому мы подробно рассмотрим этот процесс.

### **4.1 Подготовка к тестированию:**

- Перед началом тестирования необходимо убедиться, что все компоненты приложения были правильно развернуты на сервере.
- Убедитесь, что база данных заполнена тестовыми данными для проверки функциональности приложения.

### **4.2 Типы тестирования:**

- Модульное тестирование: Проведение тестов отдельных модулей приложения, таких как Servlets и JSP страницы, для проверки их корректной работы.

- **Интеграционное тестирование:** Проверка взаимодействия между различными компонентами приложения, например, между Servlets и базой данных.
- **Функциональное тестирование:** Проверка основных функций приложения, таких как добавление, редактирование и удаление контактов.
- **Нагрузочное тестирование:** Оценка производительности приложения под нагрузкой, чтобы убедиться, что оно может обрабатывать большое количество запросов.

#### **4.3 Использование инструментов для тестирования:**

- Для автоматизации тестирования можно использовать инструменты, такие как JUnit для модульного тестирования Java кода и Selenium для функционального тестирования веб-интерфейса.
- Используйте логгирование для отслеживания ошибок и проблем во время выполнения тестов.

#### **4.4 Отчет о тестировании:**

- После завершения всех видов тестирования составьте отчет о результатах, включая найденные ошибки, исправленные дефекты и общую оценку качества приложения.

#### **4.5 Улучшение качества приложения:**

- Используйте результаты тестирования для улучшения качества приложения, исправления ошибок и оптимизации производительности.

Эти скучные подробности пошагово описывают процесс тестирования веб-приложения "PhoneGuard" на Java с использованием нескольких JSP страниц. Хотя этот процесс может показаться монотонным, он является необходимым шагом для обеспечения стабильной работы приложения и удовлетворения потребностей пользователей. Важно уделить достаточное

внимание каждому этапу тестирования и провести его комплексно для достижения желаемого результата.

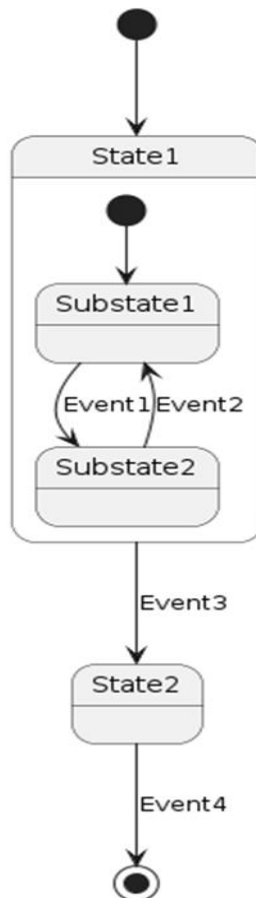
### Заключение

В ходе реализации данного проекта задачи были выполнены. Приложение соответствует требованиям, а именно:

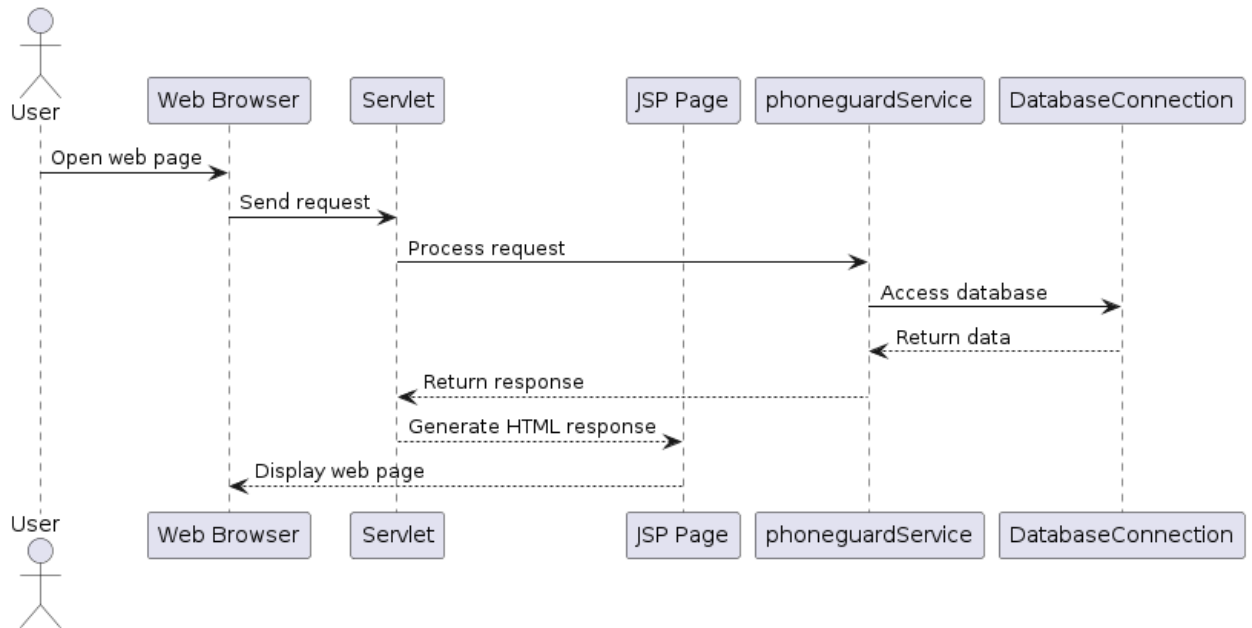
- обеспечивает возможность добавить новый контакт, редактировать и просмотреть список контактов;
- обеспечивает сохранение данных о брони в системе.

В результате выполнения всех этапов тестирования мы получаем высококачественное и надежное приложение, которое соответствует ожиданиям пользователей и требованиям бизнеса.

### Диаграмма состояний

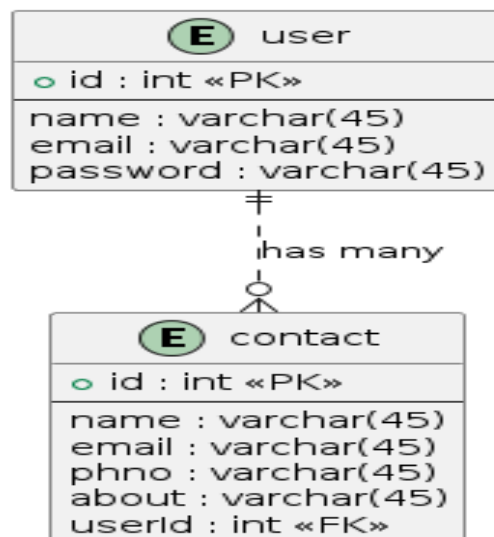


## Диаграмма последовательности

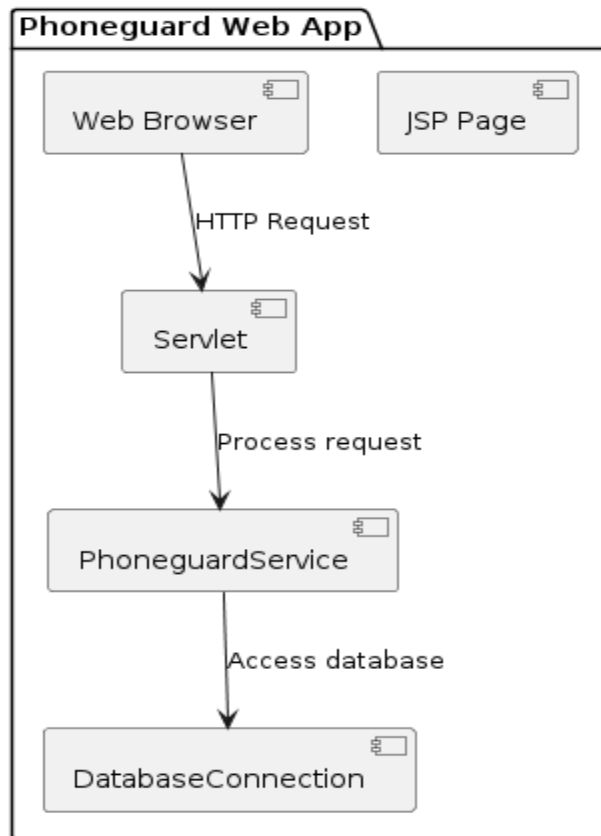


## Диаграмма база данных

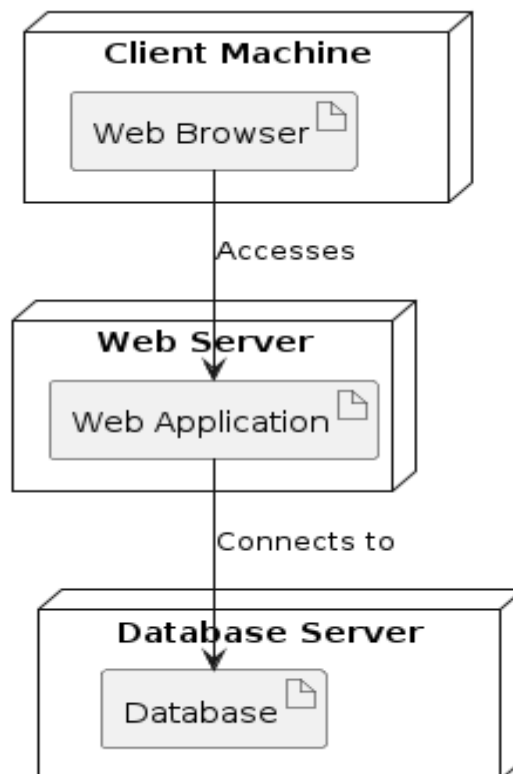
### ER-диаграмма



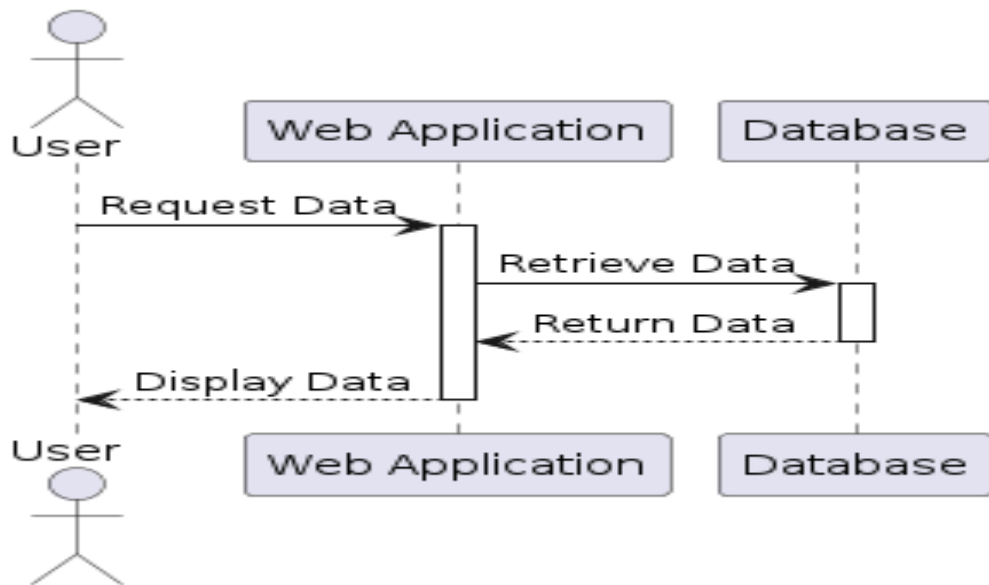
## Диаграмма компонентов



## Диаграмма развертывания



## Диаграмма использования



## Приложение

### ContactDAO class

```
package com.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import java.util.List;
import com.entity.Contact;
public class ContactDAO {
    private Connection conn;
    public ContactDAO(Connection conn) {
        super();
        this.conn = conn;
    }
    public boolean saveContact(Contact c) {
        boolean f=false;
        try {
            String sql = "Insert into cotact(name, email, phno, about, userId) values(?,
?, ?, ?, ?)";
            PreparedStatement ps=conn.prepareStatement(sql);
```

```

        ps.setString(1, c.getName());
        ps.setString(2, c.getEmail());
        ps.setString(3, c.getPhno());
        ps.setString(4, c.getAbout());
        ps.setInt(5, c.getUserId());
        int i = ps.executeUpdate();
        if(i == 1) {
            f = true;
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
    return f;
}

public List<Contact> getAllContact(int uId){
    List<Contact> list = new ArrayList<>();
    Contact c = null;
    try {
        String sql="select * from contact Where userId=?";
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setInt(1, uId);
        ResultSet rs=ps.executeQuery();
        while(rs.next()) {
            c = new Contact();
            c.setId(rs.getInt(1));
            c.setName(rs.getString(2));
            c.setEmail(rs.getString(3));
            c.setPhno(rs.getString(4));
            c.setAbout(rs.getString(5));
            list.add(c);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return list;
}

```



```

    }
    // Edit code
    public Contact getContactByID(int cid)
    {
        Contact c = new Contact();
        try {
            PreparedStatement ps=conn.prepareStatement("select * from contact where
id=?");

            ps.setInt(1, cid);
            ResultSet rs=ps.executeQuery();
            while(rs.next())
            {
                c.setId(rs.getInt(1));
                c.setName(rs.getString(2));
                c.setEmail(rs.getString(3));
                c.setPhno(rs.getString(4));
                c.setAbout(rs.getString(5));
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
        return c;
    }
    // update Contact
    public boolean updateContact(Contact c) {
        boolean f=false;
        try {
            String sql = "update contact set name=?, email=?, phno=?, about=? where
id=?";

            PreparedStatement ps=conn.prepareStatement(sql);
            ps.setString(1, c.getName());
            ps.setString(2, c.getEmail());
            ps.setString(3, c.getPhno());
            ps.setString(4, c.getAbout());
            ps.setInt(5, c.getId());

```

```

        int i = ps.executeUpdate();
        if(i == 1) {
            f = true;
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
    return f;
}

public boolean deleteContactById(int id)
{
    boolean f=false;
    try {
        String sql = "delete from contact where id=?";
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setInt(1, id);
        int i = ps.executeUpdate();
        if(i == 1) {
            f = true;
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
    return f;
}
}

```

### **UserDAO class**

```

package com.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import com.entity.User;

public class UserDAO {
    private Connection conn;

    public UserDAO(Connection conn) {

```

```

        super();
        this.conn = conn;
    }

    public boolean userRegister(User u)
    {
        boolean f = false;
        try {
            String sql="insert into user (name, email, password) VALUES(?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, u.getName());
            ps.setString(2, u.getEmail());
            ps.setString(3, u.getPassword());
            int i=ps.executeUpdate();
            if (i == 1) {
                f = true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return f;
    }

    public User loginUser(String e, String p) {
        User user = null;
        try {
            String sql="select * from user where email=? and password=?";
            PreparedStatement ps=conn.prepareStatement(sql);
            ps.setString(1, e);
            ps.setString(2, p);
            ResultSet rs=ps.executeQuery();
            if(rs.next())
            {
                user = new User();
                user.setId(rs.getInt(1));
                user.setName(rs.getString(2));
                user.setEmail(rs.getString(3));
            }
        }
    }

```

```
        user.setPassword(rs.getString(4));
    }
} catch (Exception ex){
    ex.printStackTrace();
}
return user;
}
}
```