ОТЧЕТ ПО

КУРСОВОЙ РАБОТЕ

по дисциплине

«Разработка Java-приложений управления телекоммуникациями»

Вариант №22

«Платформа для публикации статей»

Выполнил:

студент 3 курса

группы ИКПИ-15

Штейнберг Илья Васильевич

Проверил:  
ассистент кафедры ПИиВТ

Леонов Александр Сергеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_

**РЕФЕРАТ**

Отчет 74 с., 10 рис., 2 табл., 6 источн., 3 прил.

JAVA, СЕРВЛЕТ, ВЕБ-ПРИЛОЖЕНИЕ, ВЕБ-СТРАНИЦА, JAKARTA SERVLET API, MAVEN, CRUD, JDBC, СИСТЕМА УПРАВЛЕНИЯ ЗАЯВКАМИ НА ТЕХНИЧЕСКУЮ ПОДДЕРЖКУ, HTTP, JAVA SERVER PAGE

Объектом исследования являются веб-приложения, разрабатываемые с помощью языка программирования Java и программного интерфейса Jakarta Servlet API.

Цель работы – разработка платформы для публикации статей.

В ходе выполнение курсовой работы были получены следующие результаты:

1. Определен современный программный инструментарий для разработки веб-приложений на языке программирования Java;
2. Исследованы различные методы предоставления информации на веб-страницах;
3. Изучены способы работы с базами данных с помощью пользовательских программных интерфейсов, предоставляемых Java;
4. Разработан полноценный веб-сервис для публикации статей.

**ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящем отчете о курсовой работе применяют следующие термины, определения и сокращения:

*Веб-приложение* – программное приложение, которое пользователи запускают через веб-браузер на своем устройстве и взаимодействуют с ним посредством интернета.

*Веб-страница* – документ или информационный ресурс Всемирной паутины, доступ к которому осуществляется с помощью веб-браузера.

*База данных* – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

*Веб-интерфейс* – веб-страница или совокупность веб-страниц, предоставляющая пользовательский интерфейс для взаимодействия с сервисом или устройством посредством веб-браузера.

*Нормальная форма базы данных* – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами.

*URL* (Uniform Resource Locator c англ. – «единообразный указатель местонахождения ресурса») – адрес ресурса в сети Интернет.

*CRUD* (create, read, update, delete с англ – «создание, чтение, модификация, обновление) – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных.

*ER-диаграмма* (от англ. Entity-Relationship diagram, модель «сущность-связь») – модель данных, позволяющая описывать отношения и взаимосвязи между объектами в рассматриваемой предметной области.

*SQL* (от англ. Structured Query Language - «язык структурированных запросов») – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных

*API* (от англ. application programming interface – «интерфейс программирования приложения») – программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими.

*JDBC* (англ. Java DataBase Connectivity – «соединение с базами данных на Java») – платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

**Содержание**

[**1 Постановка задачи 5**](#_heading=h.30j0zll)

[**1.1 Требования к функционалу программы 5**](#_heading=h.1fob9te)

[**1.2 Требования к веб-интерфейсу программы 5**](#_heading=h.3znysh7)

[**2 Среда разработки и используемые технологии 6**](#_heading=h.2et92p0)

[**3 Структура программы 7**](#_heading=h.tyjcwt)

[**4 Отношения между таблицами в БД 8**](#_heading=h.3dy6vkm)

[**5 Разработка собственных классов 9**](#_heading=h.1t3h5sf)

[**5.1 Классы-сущности 9**](#_heading=h.4d34og8)

[**5.2 Классы-сервлеты 9**](#_heading=h.2s8eyo1)

[**5.3 Классы для работы с БД 10**](#_heading=h.17dp8vu)

[**6 Веб-страницы 11**](#_heading=h.3rdcrjn)

[**7 Блок-схемы 12**](#_heading=h.26in1rg)

[**8 Тестирование 14**](#_heading=h.lnxbz9)

[**ЗАКЛЮЧЕНИЕ 17**](#_heading=h.35nkun2)

[**СПИСОК ЛИТЕРАТУРЫ 18**](#_heading=h.1ksv4uv)

[**ПРИЛОЖЕНИЕ 19**](#_heading=)

[**ПРИЛОЖЕНИЕ A. Файлы Java 20**](#_heading=)

[**ПРИЛОЖЕНИЕ Б. Файлы JSP и CSS 57**](#_heading=h.z337ya)

[**ПРИЛОЖЕНИЕ В. Файлы конфигурации 74**](#_heading=h.3j2qqm3)

# **Постановка задачи**

Разработать веб-приложение с использованием Jakarta EE. Приложение должно располагать возможностью просмотра, добавления и изменения данных. В процессе достижения цели необходимо решить ряд следующих задач:

* ознакомиться с современными технологиями в области разработки веб-приложений и по возможности использовать их в своем проекте;
* определить программный инструментарий, применяемый для разработки веб-приложения;
* выявить и учесть методы и способы представления на веб-страницах различных видов информации;
* ознакомиться с основными принципами разработки и создания веб-приложений и следовать им в своей практике;
* определиться со структурой веб-страниц;
* разработать веб-приложение с учетом всех вышеперечисленных пунктов.

*Вариант 22.* Разработка платформы для публикации статей.

## **Требования к функционалу программы**

Разрабатываемая программа должна иметь широкий и удобный функционал по работе с статьями, комментариями, списками источников и соавторов. В качестве ключевых функций можно выделить следующие:

* просмотр списка статей (собственных и всех опубликованных);
* добавление новой статьи;
* просмотр статьи;
* добавление комментария к статье;
* удаление собственной статьи;
* добавление соавтора/источника к собственной статье.

## **Требования к веб-интерфейсу программы**

Все веб-страницы должны быть оформлены в одном стиле: использование не более двух различных шрифтов, использование не более трех цветов, допустимо использовать различные оттенки этих цветов. При разработке интерфейса веб-страниц следовать стандартам человеко-машинного взаимодействия.

Помимо вышеперечисленных требований сайт также должен быть адаптивным: расположение элементов должно оставаться удобным и понятным при просмотре с устройств с различными дисплеями.

# **Среда разработки и используемые технологии**

Программа разработана на языке Java (Liberica 11) в среде IntelliJ IDEA Ultimate 2023.2.2 в операционной системе Windows 11. Также использовались следующие технологии и библиотеки:

* Jakarta Servlet API – фреймворк, предоставляющий средства для создания веб-сервлетов и JSP-страниц;
* JDBC – программный интерфейс Java для взаимодействия с базами данных;
* PostgreSQL Driver – драйвер JDBC, предоставляющий средства подключения к базе данных PostgreSQL;
* Apache Tomcat – контейнер сервлетов, предназначенный для развертывания и выполнения веб-приложения;
* Maven – инструмент для сборки проектов, подключения зависимостей и плагинов.

# **Структура программы**

Структура программы разбита на несколько слоёв, в соответствии с MVC-архитектурой. Это упрощает процесс разработки и поддержки приложения. В разработанном веб-приложении выделены следующие слои:

* слой представления;
* слой контроллера;
* сервисный слой.

В качестве представления в разрабатываемом приложении выступают JSP-страницы. Они динамически генерируются на сервере и отображают пользователю все необходимые данные.

Веб-сервлеты представляют собой слой контроллера. Они обрабатывают пользовательские запросы и генерируют ответы.

Сервисный слой содержит всю логику работы приложения. В нем данные запросов проходят валидацию и обработку, выполняются запросы к базе данных, отлавливаются и обрабатываются ошибки.

# **Отношения между таблицами в БД**

Используемая база данных должна быть нормализована.

В базах данных, нормализация — это процесс организации данных с целью устранения избыточности и обеспечения целостности информации. Нормальные формы определяют правила для структурирования таблиц в базах данных, чтобы минимизировать дублирование данных и обеспечить эффективность и надежность операций. В подавляющем большинстве современных приложений базы данных находятся в третьей нормальной форме.

Основным принципом третьей нормальной формы является то, что неключевые столбцы должны зависеть только от первичного ключа, а не от других неключевых столбцов. То есть, если у нас есть неключевой столбец, который функционально зависит от другого неключевого столбца, то эти зависимости должны быть устранены. Преимуществом третьей нормальной формы является более логичное и четкое представление данных, однако, увеличивается сложность запросов.

Разработанная база данных удовлетворяет требованиям третьей нормальной формы. ER-диаграмма базы данных, используемой в приложении, представлена на рисунке 1.

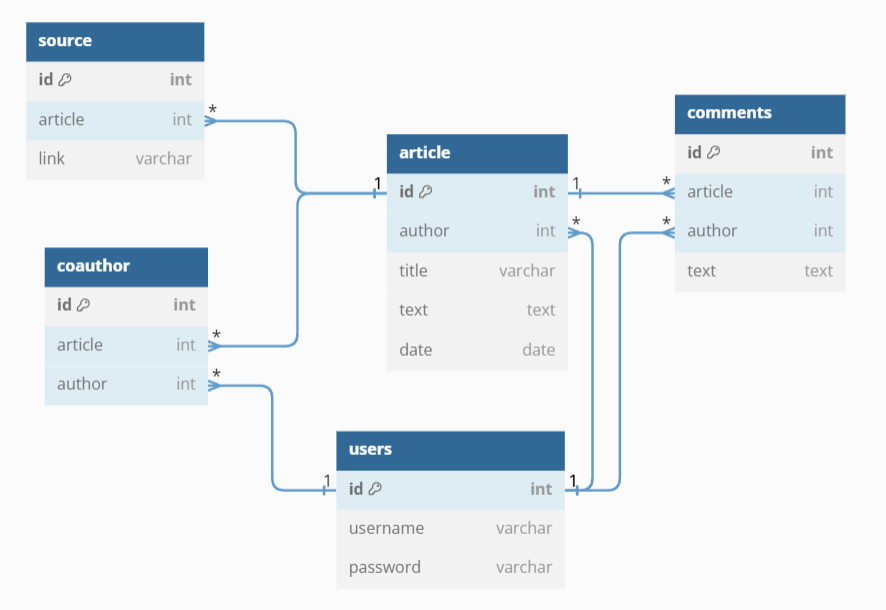


Рисунок 1 – ER-диаграмма

# **Разработка собственных классов**

В проекте разработанные классы разбиты по соответствующим пакетам:

* models – классы, описывающие сущности, разработанные в работе;
* servlets – сервлеты;
* services – классы, реализующие логику работы веб-приложения.

## **Классы-сущности**

Таблица 1 – Классы, описывающие сущности

|  |  |
| --- | --- |
| **Класс** | **Поля класса** |
| User.java –  пользователь | int id – идентификатор пользователя;  String username – имя;  String password – пароль; |
| Article.java – статья | int id – идентификатор статьи;  int author – id автора;  String title – заголовок статьи;  String text – текст статья;  Date publication – дата публикации;  String authorString – username автора(вспомогательное поле); |
| Comment.java – комментарий | int id – идентификатор комментария;  int article – id статьи;  int author – id автора коммантария;  String text – текст комментария;  String authorString – username автора(вспомогательное поле); |
| Coauthor.java – соавтор | int id – идентификатор соавтора;  int article – id статьи;  int author – id соавтора;  String authorString – username автора(вспомогательное поле); |
| Source.java –  источник | int id – идентификатор источника; int article – id статьи;  String link – ссылка на источник. |

В каждом классе реализованы методы доступа к полям, а также конструкторы с параметрами.

## **Классы-сервлеты**

В приложении разработано 3 сервлета: LoginServlet, ProfileServlet, ArticleServlet, обрабатывающих HTTP-запросы, связанные с операциями над заявками, услугами, комментариями. Каждый из сервлетов наследуется от HttpServlet и переопределяет два метода: doGet(HttpServletRequest, HttpServletResponse) и doPost(HttpServletRequest, HttpServletResponse). Эти методы обрабатывают конкретные типы запросов, переадресуют пользователя на JSP-страницы или вызывают методы сервисного слоя в зависимости от переданных в запросе параметров.

Также в приложении разработан класс-фильтр AccessFilter, обрабатывающий все HTTP-запросы. Фильтр наследуется от класса Filter и переопределяет метод doFilter(ServletRequest, ServletResponse, FilterChain), перенаправляющий неавторизованных пользователей на страницу авторизации.

## **Классы для работы с БД**

В приложении разработаны классы, отвечающие за взаимодействие с базой данных. Класс DataBase отвечает за подключение к БД и отправку запросов на исполнение СУБД. Он использует JDBC API для подключения к серверу PostgreSQL. Классы UserDAO, ArticleDAO, CommentDAO, CoauthorDAO, SourceDAO отвечают за формирование SQL-запросов к БД и обработку полученных от БД результатов и используют DataBase для отправки сформированных запросов и получения результатов.

# **Веб-страницы**

В приложении используются динамически генерируемые JSP-страницы. Все страницы имеют схожее оформление: используется одинаковый текст, схожие цвета, одно и то же навигационное меню (кроме страницы авторизации и регистрации). Все используемые стили доступны в приложении Б.

Таблица 2 – Используемые веб-страницы

|  |  |
| --- | --- |
| **Веб-страница** | **Функционал** |
| login.jsp | Страница авторизации и регистрации |
| profile.jsp | Страница профиля пользователя |
| article.jsp | Страница просмотра статьи |
| allArticles.jsp | Страница просмотра списка всех статей |
| error.jsp | Страница ошибки |

# **Блок-схемы**

Алгоритмы работы основных функций представлены на рисунках 2-4.

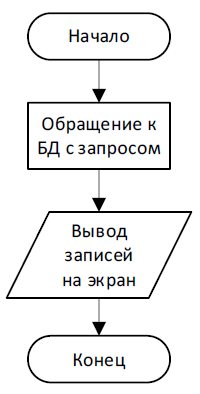


Рисунок 2 – Блок-схема просмотра данных

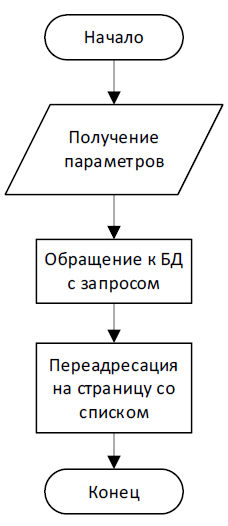


Рисунок 3 – Блок-схема добавления данных

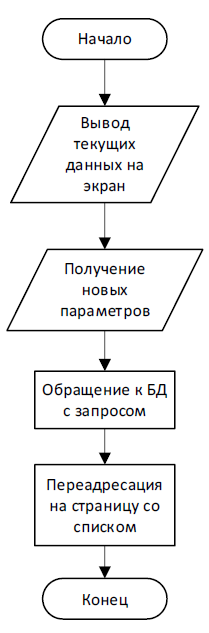


Рисунок 4 – Блок-схема изменения данных

# **Тестирование**

Тестирование было проведено вручную, все функции тестировались на предмет правильности выполнения задуманной задачи и поведения при некорректном вводе данных. Результаты тестирования и скриншоты веб-страниц приведены на рисунках 5-9.

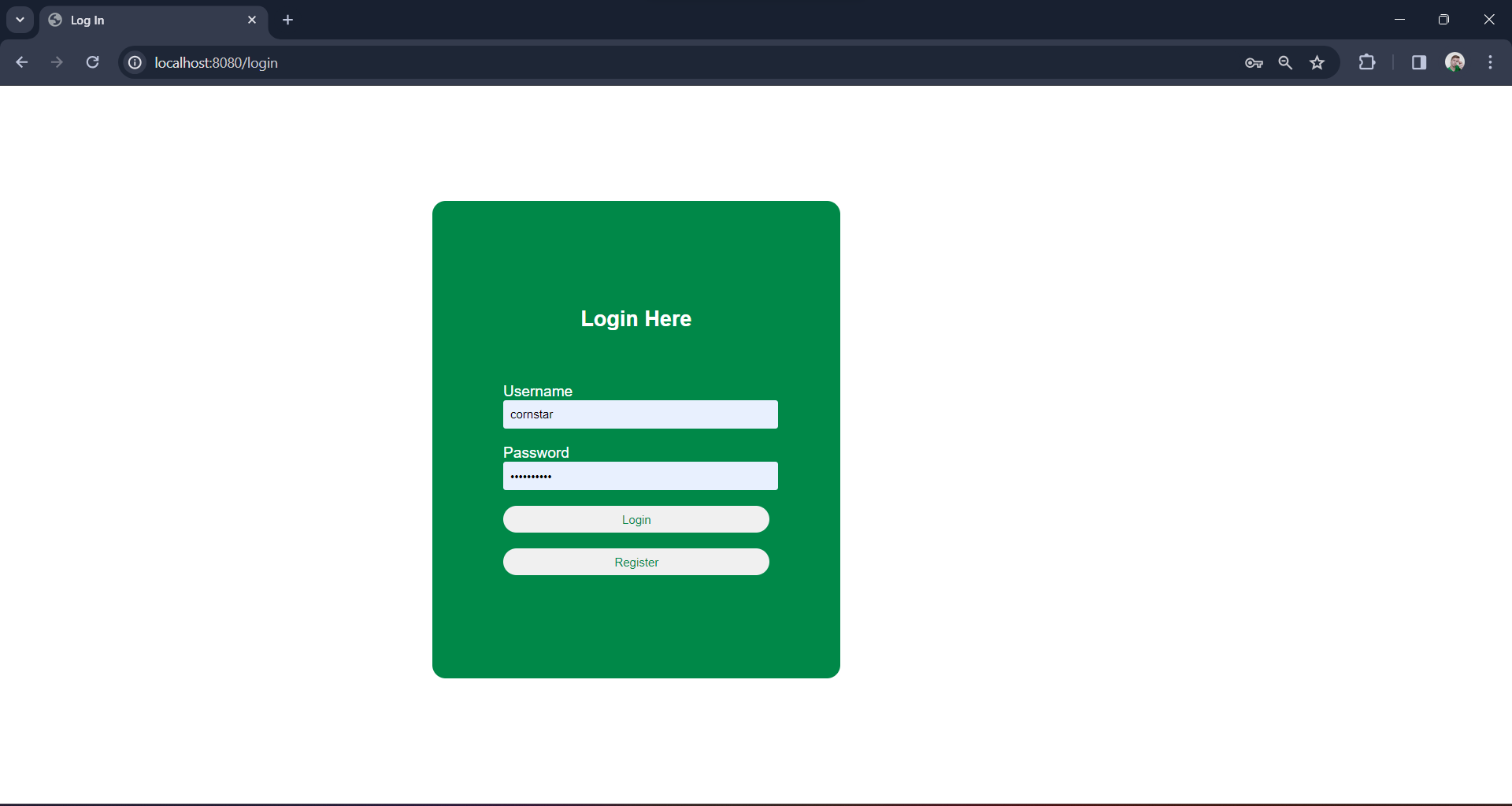


Рисунок 5 – Страница авторизации

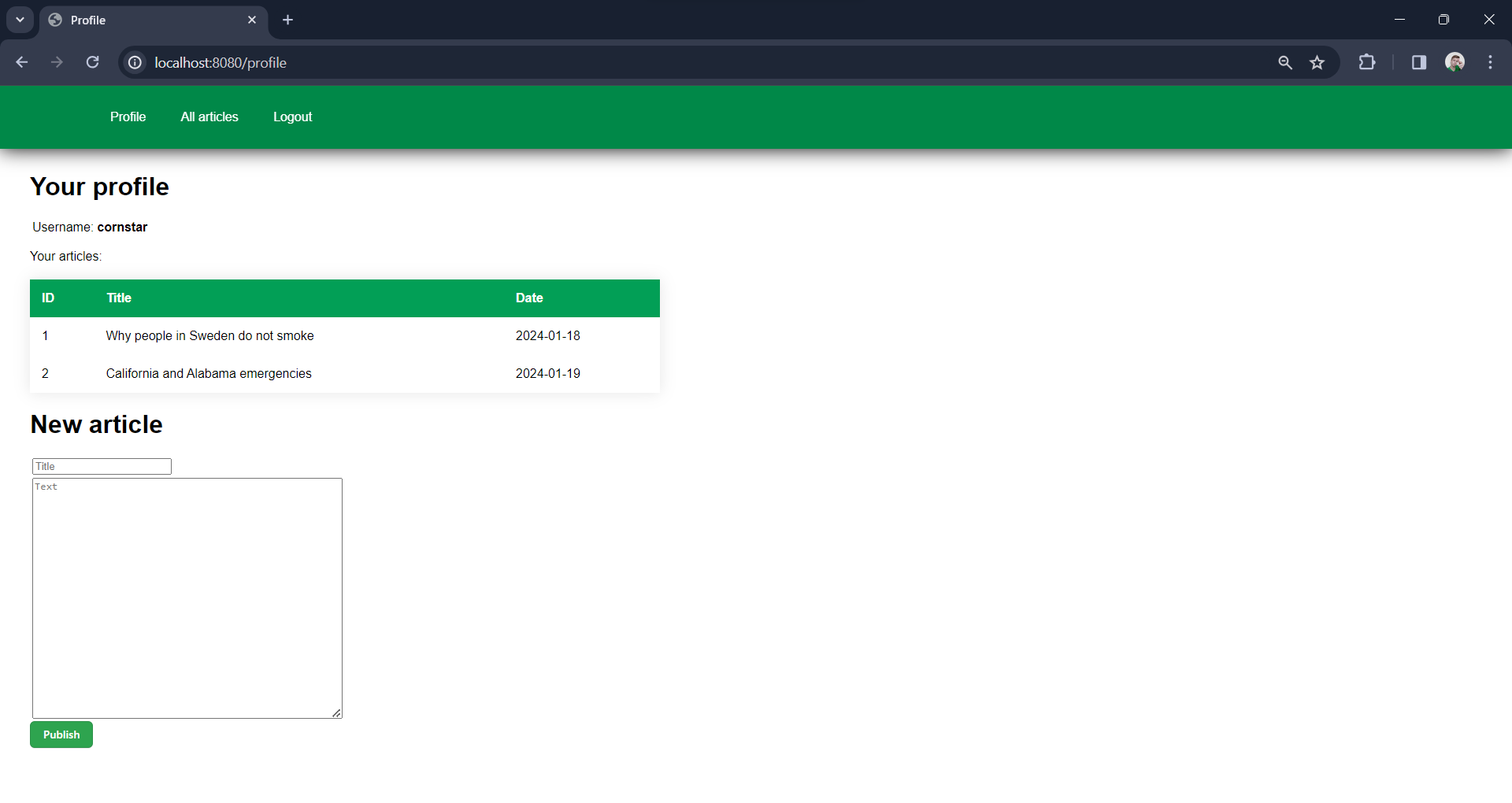


Рисунок 6 – Страница профиля пользователя

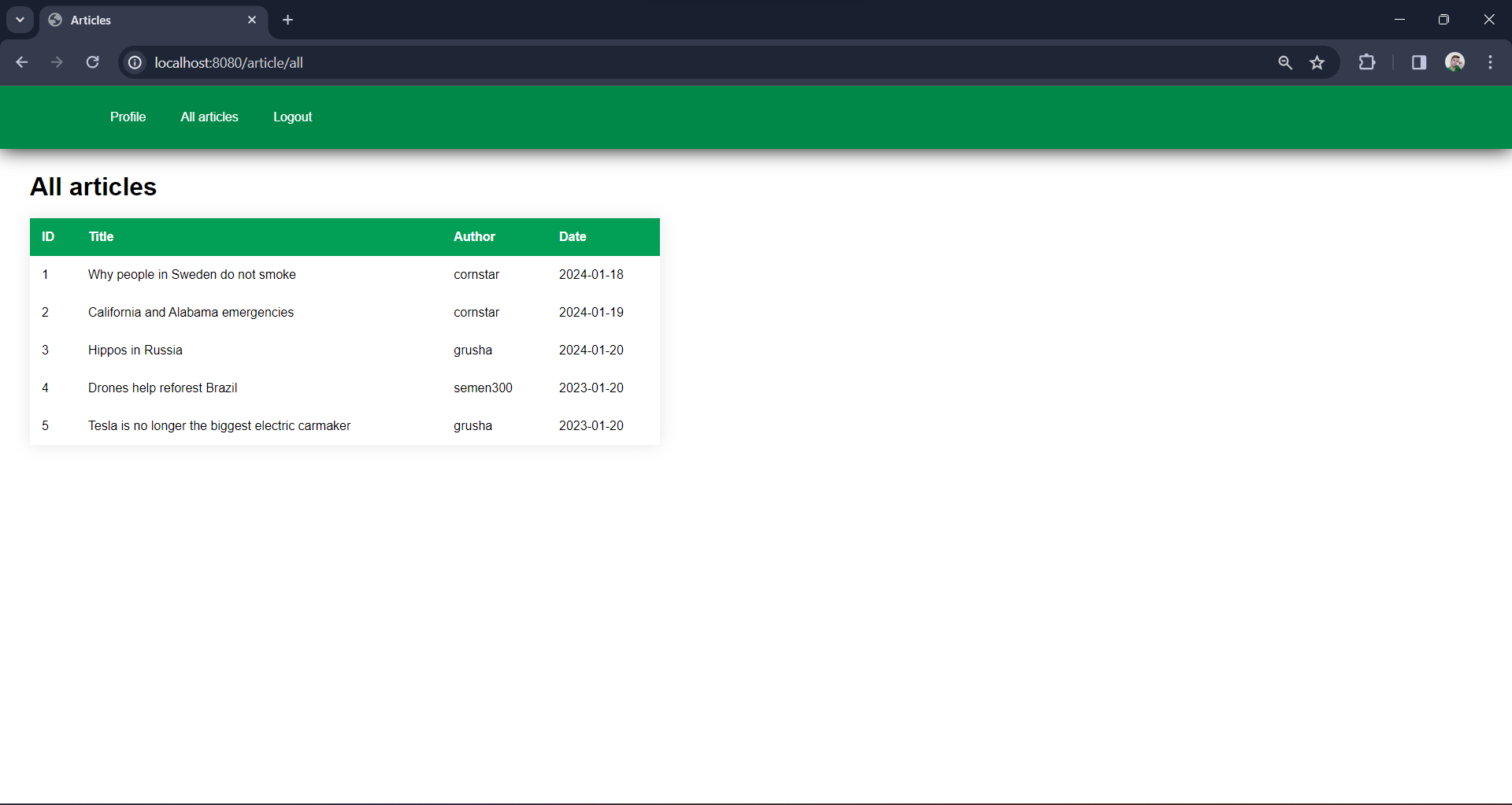


Рисунок 7 – Страница просмотра списка всех статей

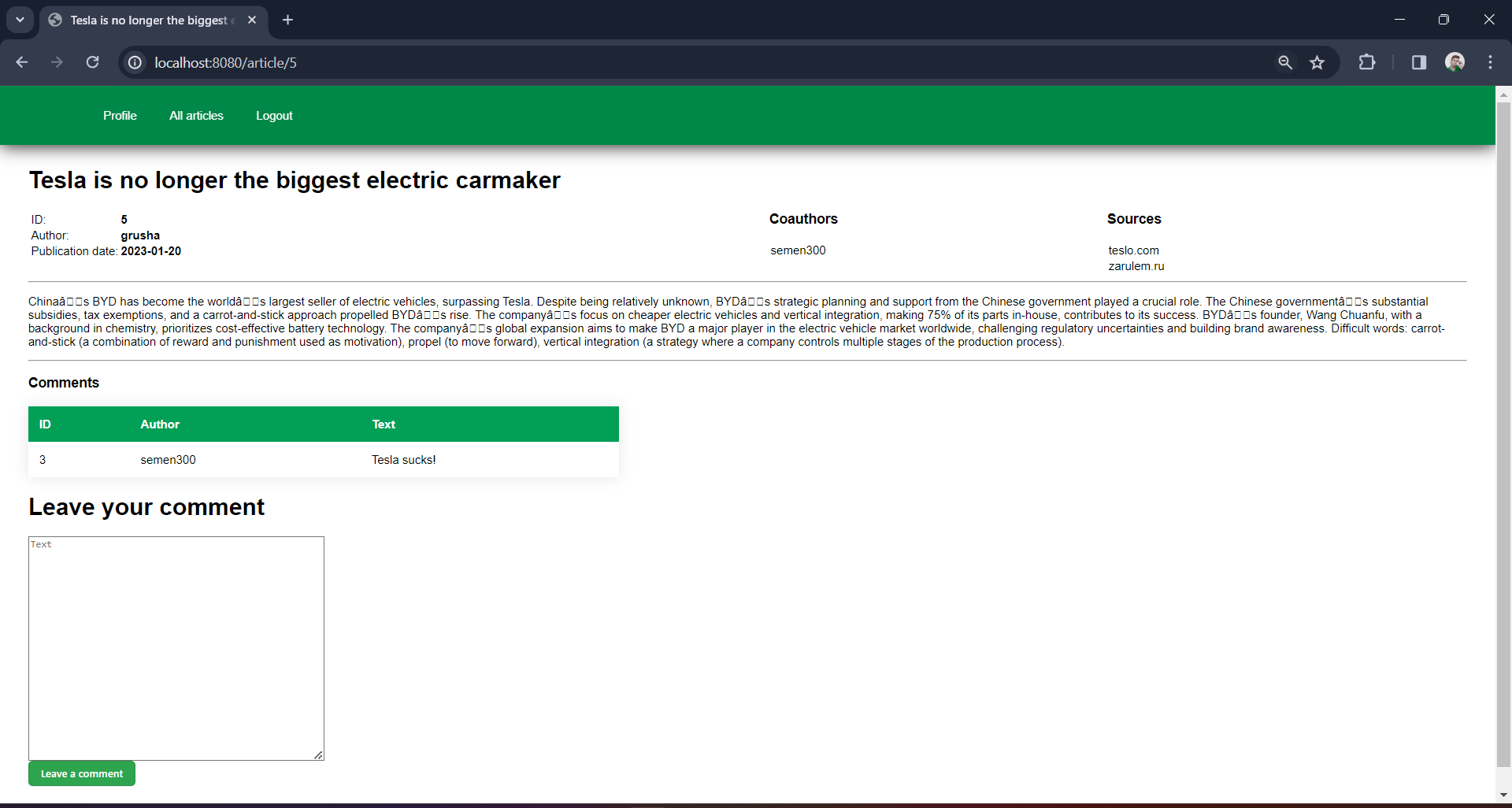


Рисунок 8 – Страница просмотра статьи

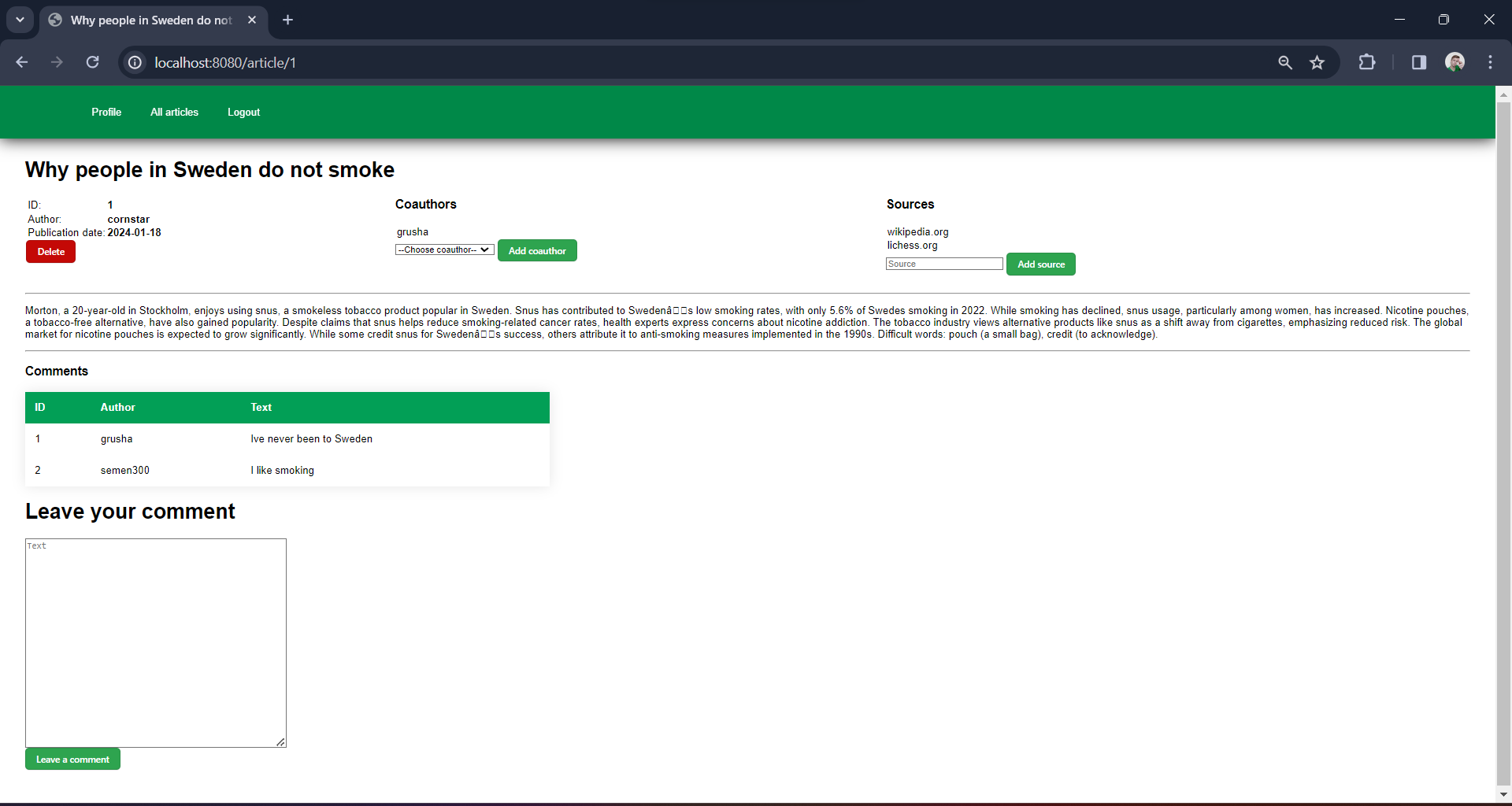


Рисунок 9 – Страница просмотра собственной статьи

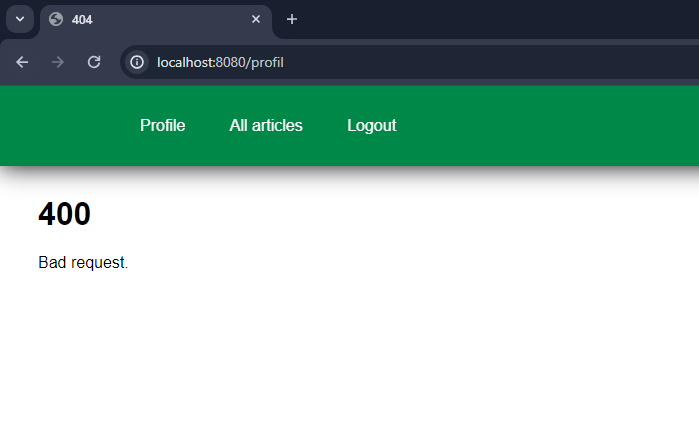


Рисунок 10 – Страница ошибки

# **ЗАКЛЮЧЕНИЕ**

В ходе курсовой работы:

* Определен программный инструментарий, применяемый для разработки веб-приложения;
* Разработано приложение, позволяющее публиковать собственные статьи и читать статьи других авторов. Приложение реализует функционал для просмотра, создания и редактирования статей, добавления комментариев;
* Проведено ручное тестирование программы.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Белая Т. И. Лекции по дисциплине «Разработка Java-приложений управления телекоммуникациями»
2. Metanit [Электронный ресурс] URL: <https://metanit.com/>
3. JavaRush [Электронный ресурс] URL: <https://javarush.com/>
4. Jakarta Serlvet Api Javadoc [Электронный ресурс] URL: <https://javadoc.io/doc/jakarta.servlet/jakarta.servlet-api/6.0.0/jakarta.servlet/module-summary.html>
5. Maven Documentation [Электронный ресурс] URL: <https://maven.apache.org/>
6. Удаленный репозиторий проекта на GitHub, URL: <https://github.com/Rypexzzz/kursach>

# **ПРИЛОЖЕНИЕ**

В приложениях приведен исходный код всей программы.

* Содержимое всех файлов \*.java приведено в приложении А;
* Содержимое файлов \*.jsp и \*.css представлено в приложении Б;
* Файлы конфигурации (pom.xml, web.xml, т.п.) представлены в приложении В.

## **ПРИЛОЖЕНИЕ A. Файлы Java**

**models.User.java**

package models;

public class User {

private int id;

private String username;

private String password;

public User(String username, String password) {

this.username = username;

this.password = password;

}

public User(int id, String username, String password) {

this(username, password);

this.id = id;

}

public Integer getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

**models.Article.java**

package models;

import java.sql.Date;

public class Article {

private int id;

private int author;

private String title;

private String text;

private Date publication;

private String authorString;

public Article(int id, int author, String title, String text, Date date) {

this.id = id;

this.author = author;

this.title = title;

this.text = text;

this.publication = date;

}

public Article(int author, String title, String text) {

this.author = author;

this.title = title;

this.text = text;

}

public Article(int id, int author, String title, String text, Date date, String authorString) {

this(id, author, title, text, date);

this.authorString = authorString;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getAuthor() {

return author;

}

public void setAuthor(int author) {

this.author = author;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

public Date getPublication() {

return publication;

}

public void setPublication(Date publication) {

this.publication = publication;

}

public String getAuthorString() {

return authorString;

}

public void setAuthorString(String authorString) {

this.authorString = authorString;

}

}

**models.Comment.java**

package models;

public class Comment {

private int id;

private int article;

private int author;

private String text;

private String authorString;

public Comment(int article, int author, String text) {

this.article = article;

this.author = author;

this.text = text;

}

public Comment(int id, int article, int author, String text) {

this(article, author, text);

this.id = id;

}

public Comment(int id, int article, int author, String text, String authorString) {

this(id, article, author, text);

this.authorString = authorString;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getArticle() {

return article;

}

public void setArticle(int article) {

this.article = article;

}

public int getAuthor() {

return author;

}

public void setAuthor(int author) {

this.author = author;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

public String getAuthorString() {

return authorString;

}

public void setAuthorString(String authorString) {

this.authorString = authorString;

}

}

**models.Coauthor.java**

package models;

public class Coauthor {

private int id;

private int article;

private int author;

private String authorString;

public Coauthor(int article, int author) {

this.article = article;

this.author = author;

}

public Coauthor(int id, int article, int author) {

this(article, author);

this.id = id;

}

public Coauthor(int id, int article, int author, String authorString) {

this(id, article, author);

this.authorString = authorString;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getArticle() {

return article;

}

public void setArticle(int article) {

this.article = article;

}

public int getAuthor() {

return author;

}

public void setAuthor(int author) {

this.author = author;

}

public String getAuthorString() {

return authorString;

}

public void setAuthorString(String authorString) {

this.authorString = authorString;

}

}

**models.Source.java**

package models;

public class Source {

private int id;

private int article;

private String link;

public Source(int article, String link) {

this.article = article;

this.link = link;

}

public Source(int id, int article, String link) {

this(article, link);

this.id = id;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getArticle() {

return article;

}

public void setArticle(int article) {

this.article = article;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

}

**services.LoginService.java**

package services;

import com.google.common.hash.Hashing;

import models.User;

import java.nio.charset.StandardCharsets;

public class LoginService {

public String hashPassword(String password) {

return Hashing.sha256().hashString(password, StandardCharsets.UTF\_8).toString();

}

public User login(String username, String password) {

User user = new UserDAO().getByUsername(username);

if (user != null && user.getPassword().equals(hashPassword(password))) {

return user;

}

return null;

}

public User register(String username, String password) {

User user = null;

UserDAO udao = new UserDAO();

Integer id = udao.create(new User(username, hashPassword(password)));

if (id != null) {

user = udao.getByID(id);

}

return user;

}

}

**services.DataBase.java**

package services;

import java.sql.\*;

public class DataBase {

private Connection con = getConnect();

private Connection getConnect() {

final String url = "jdbc:postgresql://217.107.219.154:49307/bonch\_2105457";

final String user = "bonch\_2105457";

final String password = "vBUZSOTO7zg=";

Connection con = null;

try {

Class.forName("org.postgresql.Driver");

con = DriverManager.getConnection(url, user, password);

} catch (ClassNotFoundException e) {

System.out.println(e.getMessage());

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return con;

}

public ResultSet select(String sql) {

ResultSet rs = null;

try {

if (con != null) {

rs = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_UPDATABLE).executeQuery(sql);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return rs;

}

public boolean insert(String sql){

try {

if (con != null) {

int rowsAffected = con.createStatement().executeUpdate(sql);

if (rowsAffected > 0) {

return true;

}

}

} catch (Exception e) {

System.out.println(e.getMessage());;

}

return false;

}

public boolean delete(String sql) {

try {

if (con != null) {

int rowsAffected = con.createStatement().executeUpdate(sql);

if (rowsAffected > 0) {

return true;

}

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return false;

}

}

**services.UserDAO.java**

package services;

import models.User;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class UserDAO {

DataBase db = new DataBase();

public List<User> getAll() {

List<User> users = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT \* FROM users");

if (rs == null) {

return users;

}

while (rs.next()) {

users.add(new User(

rs.getInt("id"),

rs.getString("username"),

rs.getString("password")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return users;

}

public User getByID(int id) {

User user = null;

try {

ResultSet rs = db.select("SELECT \* FROM users WHERE id=" + id);

if (rs == null) {

return user;

}

if (rs.next()) {

user = new User(

rs.getInt("id"),

rs.getString("username"),

rs.getString("password")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return user;

}

public User getByUsername(String username) {

User user = null;

try {

ResultSet rs = db.select("SELECT \* FROM users WHERE username=\'" + username + "\'");

if (rs == null) {

return user;

}

if (rs.next()) {

user = new User(

rs.getInt("id"),

rs.getString("username"),

rs.getString("password")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return user;

}

public Integer create(User user) {

Integer id = null;

if (getByUsername(user.getUsername()) == null) {

String sql = "INSERT INTO users(username, password) values(\'"

+ user.getUsername() + "\', \'"

+ user.getPassword() + "\')";

if (db.insert(sql) && (user = getByUsername(user.getUsername())) != null) {

id = user.getId();

}

}

return id;

}

public boolean edit(int id, User user) {

try {

ResultSet rs = db.select("SELECT username FROM users WHERE id!=" + id + " AND username=\'" + user.getUsername() + "\'");

if (rs.next()) {

throw new SQLException("Username is already taken");

}

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

} catch (NullPointerException e) {}

try {

ResultSet rs = db.select("SELECT \* FROM users WHERE id=" + id);

if (rs == null) {

return false;

}

if (rs.next()) {

rs.updateString("username", user.getUsername());

rs.updateRow();

} else throw new SQLException();

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

}

return true;

}

public boolean delete(int id) {

String sql = "DELETE FROM users WHERE id = " + id;

return db.delete(sql);

}

}

**services.ArticleDAO.java**

package services;

import models.Article;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class ArticleDAO {

DataBase db = new DataBase();

public List<Article> getAll() {

List<Article> articles = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT a.\*, u.username as authorString FROM articles a JOIN users u ON a.author=u.id ORDER BY a.id");

if (rs == null) {

return articles;

}

while (rs.next()) {

articles.add(new Article(

rs.getInt("id"),

rs.getInt("author"),

rs.getString("title"),

rs.getString("text"),

rs.getDate("publication"),

rs.getString("authorString")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return articles;

}

public Article getByID(int id) {

Article article = null;

try {

ResultSet rs = db.select("SELECT a.\*, u.username as authorString FROM articles a JOIN users u ON a.author=u.id WHERE a.id=" + id);

if (rs == null) {

return article;

}

if (rs.next()) {

article = new Article(

rs.getInt("id"),

rs.getInt("author"),

rs.getString("title"),

rs.getString("text"),

rs.getDate("publication"),

rs.getString("authorString")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return article;

}

public List<Article> getByAuthor(int author) {

List<Article> articles = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT \* FROM articles WHERE author=" + author + " ORDER BY id");

if (rs == null) {

return articles;

}

while (rs.next()) {

articles.add(new Article(

rs.getInt("id"),

rs.getInt("author"),

rs.getString("title"),

rs.getString("text"),

rs.getDate("publication")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return articles;

}

public boolean create(Article article) {

String sql = "INSERT INTO articles(author, title, text) values("

+ article.getAuthor() + ", \'"

+ article.getTitle() + "\', \'"

+ article.getText() + "\')";

return db.insert(sql);

}

public boolean edit(int id, Article article) {

try {

ResultSet rs = db.select("SELECT \* FROM articles WHERE id=" + id);

if (rs == null) {

return false;

}

if (rs.next()) {

rs.updateInt("author", article.getAuthor());

rs.updateString("title", article.getTitle());

rs.updateString("text", article.getText());

rs.updateRow();

} else throw new SQLException();

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

}

return true;

}

public boolean delete(int id) {

String sql = "DELETE FROM articles WHERE id = " + id;

return db.delete(sql);

}

}

**services.CommentDAO.java**

package services;

import models.Comment;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class CommentDAO {

DataBase db = new DataBase();

public List<Comment> getAll() {

List<Comment> comments = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT \* FROM comments ORDER BY id");

if (rs == null) {

return comments;

}

while (rs.next()) {

comments.add(new Comment(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("text")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return comments;

}

public Comment getByID(int id) {

Comment comment = null;

try {

ResultSet rs = db.select("SELECT \* FROM comments WHERE id=" + id);

if (rs == null) {

return comment;

}

if (rs.next()) {

comment = new Comment(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("text")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return comment;

}

public List<Comment> getByArticle(int article) {

List<Comment> comments = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT c.\*, u.username as authorString FROM comments c JOIN users u ON c.author=u.id WHERE c.article=" + article + " ORDER BY c.id");

if (rs == null) {

return comments;

}

while (rs.next()) {

comments.add(new Comment(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("text"),

rs.getString("authorString")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return comments;

}

public boolean create(Comment comment) {

String sql = "INSERT INTO comments(article, author, text) values("

+ comment.getArticle() + ", "

+ comment.getAuthor() + ", \'"

+ comment.getText() + "\')";

return db.insert(sql);

}

public boolean edit(int id, Comment comment) {

try {

ResultSet rs = db.select("SELECT \* FROM comments WHERE id=" + id);

if (rs == null) {

return false;

}

if (rs.next()) {

rs.updateInt("article", comment.getArticle());

rs.updateInt("author", comment.getAuthor());

rs.updateString("text", comment.getText());

rs.updateRow();

} else throw new SQLException();

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

}

return true;

}

public boolean delete(int id) {

String sql = "DELETE FROM comments WHERE id = " + id;

return db.delete(sql);

}

}

**services.CoauthorDAO.java**

package services;

import models.Coauthor;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class CoauthorDAO {

DataBase db = new DataBase();

public List<Coauthor> getAll() {

List<Coauthor> coauthors = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT c.\*, u.username FROM coauthors c JOIN users u ON c.author=u.id ORDER BY c.id");

if (rs == null) {

return coauthors;

}

while (rs.next()) {

coauthors.add(new Coauthor(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("authorString")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return coauthors;

}

public Coauthor getByID(int id) {

Coauthor coauthor = null;

try {

ResultSet rs = db.select("SELECT c.\*, u.username FROM coauthors c JOIN users u ON c.author=u.id WHERE c.id=" + id);

if (rs == null) {

return coauthor;

}

if (rs.next()) {

coauthor = new Coauthor(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("authorString")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return coauthor;

}

public List<Coauthor> getByArticle(int article) {

List<Coauthor> coauthors = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT c.\*, u.username as authorString FROM coauthors c JOIN users u ON c.author=u.id WHERE c.article=" + article + " ORDER BY c.id");

if (rs == null) {

return coauthors;

}

while (rs.next()) {

coauthors.add(new Coauthor(

rs.getInt("id"),

rs.getInt("article"),

rs.getInt("author"),

rs.getString("authorString")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return coauthors;

}

public boolean create(Coauthor coauthor) {

String sql = "INSERT INTO coauthors(article, author) values("

+ coauthor.getArticle() + ", "

+ coauthor.getAuthor() + ")";

return db.insert(sql);

}

public boolean edit(int id, Coauthor coauthor) {

try {

ResultSet rs = db.select("SELECT \* FROM coauthors WHERE id=" + id);

if (rs == null) {

return false;

}

if (rs.next()) {

rs.updateInt("article", coauthor.getArticle());

rs.updateInt("author", coauthor.getAuthor());

rs.updateRow();

} else throw new SQLException();

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

}

return true;

}

public boolean delete(int id) {

String sql = "DELETE FROM coauthors WHERE id = " + id;

return db.delete(sql);

}

}

**services.SourceDAO.java**

package services;

import models.Source;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class SourceDAO {

DataBase db = new DataBase();

public List<Source> getAll() {

List<Source> sources = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT s.\*, u.username FROM sources s JOIN users u ON s.author=u.id ORDER BY s.id");

if (rs == null) {

return sources;

}

while (rs.next()) {

sources.add(new Source(

rs.getInt("id"),

rs.getInt("article"),

rs.getString("link")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return sources;

}

public Source getByID(int id) {

Source source = null;

try {

ResultSet rs = db.select("SELECT s.\*, u.username FROM sources s JOIN users u ON s.author=u.id WHERE s.id=" + id);

if (rs == null) {

return source;

}

if (rs.next()) {

source = new Source(

rs.getInt("id"),

rs.getInt("article"),

rs.getString("link")

);

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return source;

}

public List<Source> getByArticle(int article) {

List<Source> sources = new ArrayList<>();

try {

ResultSet rs = db.select("SELECT \* FROM sources WHERE article=" + article + " ORDER BY id");

if (rs == null) {

return sources;

}

while (rs.next()) {

sources.add(new Source(

rs.getInt("id"),

rs.getInt("article"),

rs.getString("link")

));

}

} catch (SQLException e) {

System.out.println(e.getMessage());

}

return sources;

}

public boolean create(Source source) {

String sql = "INSERT INTO sources(article, link) values("

+ source.getArticle() + ", \'"

+ source.getLink() + "\')";

return db.insert(sql);

}

public boolean edit(int id, Source source) {

try {

ResultSet rs = db.select("SELECT \* FROM sources WHERE id=" + id);

if (rs == null) {

return false;

}

if (rs.next()) {

rs.updateInt("article", source.getArticle());

rs.updateString("link", source.getLink());

rs.updateRow();

} else throw new SQLException();

} catch (SQLException e) {

System.out.println(e.getMessage());

return false;

}

return true;

}

public boolean delete(int id) {

String sql = "DELETE FROM sources WHERE id = " + id;

return db.delete(sql);

}

}

**servlets.LoginServlet.java**

package servlets;

import models.User;

import services.LoginService;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

@WebServlet(urlPatterns = {"/login", "/register", "/logout"})

public class LoginServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

switch (req.getRequestURI()) {

case "/logout": {

HttpSession session = req.getSession();

if (session != null && session.getAttribute("id") != null) {

session.removeAttribute("id");

}

if (session != null && session.getAttribute("username") != null) {

session.removeAttribute("username");

}

resp.sendRedirect(req.getContextPath() + "/login");

return;

}

case "/login":

case "/register": {

resp.setContentType("text/html");

req.getRequestDispatcher("/login.jsp").forward(req, resp);

return;

}

default: {

resp.sendError(404);

}

}

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

var loginSubmit = req.getParameter("loginSubmit");

var registerSubmit = req.getParameter("registerSubmit");

String username = req.getParameter("username");

String password = req.getParameter("password");

if (loginSubmit != null && registerSubmit == null) {

if (username != null && password != null) {

User user = new LoginService().login(username, password);

if (user != null && user.getId() != null) {

HttpSession session = req.getSession();

session.setAttribute("id", user.getId());

session.setAttribute("username", user.getUsername());

resp.sendRedirect(req.getContextPath() + "/profile");

return;

} else {

req.setAttribute("errorText", "Incorrect username or password");

resp.setContentType("text/html");

req.getRequestDispatcher("/login.jsp").forward(req, resp);

return;

}

} else {

req.setAttribute("errorText", "Incorrect username or password");

resp.setContentType("text/html");

req.getRequestDispatcher("/login.jsp").forward(req, resp);

return;

}

} else if (loginSubmit == null && registerSubmit != null) {

if (username != null && password != null) {

User user = new LoginService().register(username, password);

if (user != null && user.getId() != null) {

HttpSession session = req.getSession();

session.setAttribute("id", user.getId());

session.setAttribute("username", user.getUsername());

resp.sendRedirect(req.getContextPath() + "/profile");

return;

} else {

req.setAttribute("errorText", "Username is already taken");

resp.setContentType("text/html");

req.getRequestDispatcher("/login.jsp").forward(req, resp);

return;

}

} else {

req.setAttribute("errorText", "Username is already taken");

resp.setContentType("text/html");

req.getRequestDispatcher("/login.jsp").forward(req, resp);

return;

}

}

resp.sendError(400);

}

}

**servlets.ProfileServlet.java**

package servlets;

import models.Article;

import services.ArticleDAO;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.util.List;

@WebServlet(urlPatterns = {"/profile"})

public class ProfileServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String path = req.getPathInfo();

HttpSession session = req.getSession();

req.setAttribute("username", session.getAttribute("username"));

List<Article> articles = new ArticleDAO().getByAuthor((int) session.getAttribute("id"));

req.setAttribute("articles", articles);

resp.setContentType("text/html");

req.getRequestDispatcher("/profile.jsp").forward(req, resp);

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String title = req.getParameter("title");

String text = req.getParameter("text");

if (req.getParameter("publish") != null) {

if (title != null && text != null) {

new ArticleDAO().create(new Article((int) req.getSession().getAttribute("id"), title, text));

}

resp.sendRedirect(req.getContextPath() + "/profile");

return;

}

}

}

**servlets.ArticleServlet.java**

package servlets;

import models.\*;

import services.\*;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.util.List;

@WebServlet(urlPatterns = {"/article/\*"})

public class ArticleServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String path = req.getPathInfo();

String[] pathList = path.split("/");

try {

switch (pathList[1]) {

case "all": {

List<Article> articles = new ArticleDAO().getAll();

req.setAttribute("articles", articles);

resp.setContentType("text/html");

req.getRequestDispatcher("/allArticles.jsp").forward(req, resp);

return;

}

Default: {

int articleID = Integer.parseInt(pathList[1]);

Article article = new ArticleDAO().getByID(articleID);

if (article == null) {

resp.sendError(403);

return;

}

req.setAttribute("article", article);

List<Comment> comments = new CommentDAO().getByArticle(articleID);

req.setAttribute("comments", comments);

List<Coauthor> coauthors = new CoauthorDAO().getByArticle(articleID);

req.setAttribute("coauthors", coauthors);

List<Source> sources = new SourceDAO().getByArticle(articleID);

req.setAttribute("sources", sources);

boolean yours = (article.getAuthor() == (int) req.getSession().getAttribute("id"));

if (yours) {

List<User> users = new UserDAO().getAll();

req.setAttribute("users", users);

}

req.setAttribute("yours", yours);

resp.setContentType("text/html");

req.getRequestDispatcher("/article.jsp").forward(req, resp);

return;

}

}

} catch (NumberFormatException | IndexOutOfBoundsException e) {

e.printStackTrace();

resp.sendError(404);

}

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String path = req.getPathInfo();

String[] pathList = path.split("/");

try {

int articleID = Integer.parseInt(pathList[1]);

Article article = new ArticleDAO().getByID(articleID);

if (article != null) {

Object delete = req.getParameter("delete");

Object comment = req.getParameter("comment");

Object coauthor = req.getParameter("coauthor");

Object source = req.getParameter("source");

if (delete != null && comment == null && coauthor == null && source == null) {

new ArticleDAO().delete(articleID);

resp.sendRedirect(req.getContextPath() + "/article/all");

return;

} else if (delete == null && comment != null && coauthor == null && source == null) {

String text = req.getParameter("text");

if (text != null) {

new CommentDAO().create(new Comment(articleID, (int) req.getSession().getAttribute("id"), text));

}

resp.sendRedirect(req.getContextPath() + "/article/" + articleID);

return;

} else if (delete == null & comment == null && coauthor != null && source == null) {

int coauthorID = Integer.parseInt(req.getParameter("coauthorID"));

new CoauthorDAO().create(new Coauthor(articleID, coauthorID));

resp.sendRedirect(req.getContextPath() + "/article/" + articleID);

return;

} else if (delete == null & comment == null && coauthor == null && source != null) {

String sourceText = req.getParameter("sourceText");

new SourceDAO().create(new Source(articleID, sourceText));

resp.sendRedirect(req.getContextPath() + "/article/" + articleID);

return;

} else {

resp.sendError(405);

return;

}

} else {

resp.sendError(403);

return;

}

} catch (NumberFormatException | IndexOutOfBoundsException e) {

e.printStackTrace();

resp.sendError(405);

}

}

}

**servlets.AccessFilter.java**

package servlets;

import services.UserDAO;

import javax.servlet.\*;

import javax.servlet.annotation.WebFilter;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

@WebFilter("/\*")

public class AccessFilter implements Filter {

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) servletRequest;

HttpServletResponse resp = (HttpServletResponse) servletResponse;

String path = req.getRequestURI();

if ((req.getContextPath() + "/login").equals(path)

|| (req.getContextPath() + "/logout").equals(path)

|| (req.getContextPath() + "/register").equals(path)

|| path.endsWith(".css")

|| path.endsWith(".png")

|| path.endsWith(".jpg")

) {

filterChain.doFilter(servletRequest, servletResponse);

return;

}

HttpSession session = req.getSession(false);

if (session == null || session.getAttribute("id") == null || session.getAttribute("username") == null) {

resp.sendRedirect(req.getContextPath() + "/login");

return;

}

if (new UserDAO().getByID((int) session.getAttribute("id")) == null) {

resp.sendRedirect(req.getContextPath() + "/login");

return;

}

filterChain.doFilter(servletRequest, servletResponse);

}

}

## **ПРИЛОЖЕНИЕ Б. Файлы JSP и CSS**

**login.jsp**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css">

<title>Log In</title>

</head>

<body>

<div class="login">

<form method="post">

<h2>Login Here</h2>

<label>${errorText}</label>

<br>

<label for="username">Username</label>

<input type="text" placeholder="Username" id="username" name="username" pattern="[a-z0-9\_]+" required>

<label for="password">Password</label>

<input type="password" placeholder="Password" id="password" name="password" pattern="[a-z0-9\_\_]+" required>

<input type="submit" name="loginSubmit" class="log-button" value="Login">

<input type="submit" name="registerSubmit" class="log-button" value="Register">

</form>

</div>

</body>

</html>

**profile.jsp**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css">

<title>Profile</title>

</head>

<body>

<header>

<div class="headerMenu">

<nav>

<ul>

<li></li>

<li></li>

<li><a href="${pageContext.request.contextPath}/profile">Profile</a></li>

<li><a href="${pageContext.request.contextPath}/article/all">All articles</a></li>

<li><a href="${pageContext.request.contextPath}/logout">Logout</a></li>

</ul>

</nav>

</div>

</header>

<div id="content">

<h1>Your profile</h1>

<table class="profile">

<tr>

<td>Username:</td>

<td><b>${username}</b></td>

</tr>

</table>

<p>Your articles:</p>

<div class="container">

<table class="f1-table">

<thead>

<tr>

<th>ID</th>

<th>Title</th>

<th>Date</th>

</tr>

</thead>

<tbody>

<c:forEach items="${articles}" var="article">

<tr class="article" onclick="window.location.href='${pageContext.request.contextPath}/article/${article.id}'">

<td>${article.id}</td>

<td>${article.title}</td>

<td>${article.publication}</td>

</tr>

</c:forEach>

</tbody>

</table>

</div>

<div class="clear"></div>

<h1>New article</h1>

<form method="post">

<table class="profile">

<tr>

<td><input type="text" name="title" id="title"

placeholder="Title" required

pattern=".\*"

></td>

</tr>

<tr>

<td><textarea name="text" id="text" cols="50" rows="20"

placeholder="Text" required

></textarea></td>

</tr>

</table>

<div class="clear"></div>

<input type="submit" class="button-3" name="publish" value="Publish">

</form>

</div>

</body>

</html>

**article.jsp**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css">

<title>${article.title}</title>

</head>

<body>

<header>

<div class="headerMenu">

<nav>

<ul>

<li></li>

<li></li>

<li><a href="${pageContext.request.contextPath}/profile">Profile</a></li>

<li><a href="${pageContext.request.contextPath}/article/all">All articles</a></li>

<li><a href="${pageContext.request.contextPath}/logout">Logout</a></li>

</ul>

</nav>

</div>

</header>

<div id="content">

<h1>${article.title}</h1>

<table class="three-tables">

<tr>

<td>

<div class="table-of-three">

<table class="profile">

<tr>

<td>ID:</td>

<td><b>${article.id}</b></td>

</tr>

<tr>

<td>Author:</td>

<td><b>${article.authorString}</b></td>

</tr>

<tr>

<td>Publication date:</td>

<td><b>${article.publication}</b></td>

</tr>

</table>

</div>

<c:if test="${yours}">

<form method="post">

<input type="submit" name="delete" class="button-3" style="background-color: #c40906" value="Delete">

</form>

</c:if>

</td>

<td>

<div class="table-of-three">

<h3>Coauthors</h3>

<table class="profile">

<c:forEach items="${coauthors}" var="coauthor">

<tr><td>${coauthor.authorString}</td></tr>

</c:forEach>

</table>

<c:if test="${yours}">

<form method="post">

<select name="coauthorID" required>

<c:forEach items="${users}" var="user">

<option value="${user.id}">${user.username}</option>

</c:forEach>

</select>

<input type="submit" class="button-3" name="coauthor" value="Add coauthor">

</form>

</c:if>

</div>

</td>

<td>

<div class="table-of-three">

<h3>Sources</h3>

<table class="profile">

<c:forEach items="${sources}" var="source">

<tr><td>${source.link}</td></tr>

</c:forEach>

</table>

<c:if test="${yours}">

<form method="post">

<input type="text" name="sourceText" placeholder="Source" required>

<input type="submit" class="button-3" name="source" value="Add source">

</form>

</c:if>

</div>

</td>

</tr>

</table>

<hr>

<p>${article.text}</p>

<hr>

<div class="clear"></div>

<h3>Comments</h3>

<%-- <input type="submit" class="button-3" value="Leave a comment" onclick="window.location.href='${pageContext.request.contextPath}/comment/${article.id}'">--%>

<div class="container">

<table class="f1-table">

<thead>

<tr>

<th>ID</th>

<th>Author</th>

<th>Text</th>

</tr>

</thead>

<tbody>

<c:forEach items="${comments}" var="comment">

<tr>

<td>${comment.id}</td>

<td>${comment.authorString}</td>

<td>${comment.text}</td>

</tr>

</c:forEach>

</tbody>

</table>

</div>

<div class="clear"></div>

<h1>Leave your comment</h1>

<form method="post">

<table class="profile">

<tr><textarea name="text" id="text" cols="50" rows="20"

placeholder="Text" required

></textarea></tr>

</table>

<div class="clear"></div>

<input type="submit" class="button-3" name="comment" value="Leave a comment">

</form>

</div>

</body>

</html>

**allArticles.jsp**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css">

<title>Articles</title>

</head>

<body>

<header>

<div class="headerMenu">

<nav>

<ul>

<li></li>

<li></li>

<li><a href="${pageContext.request.contextPath}/profile">Profile</a></li>

<li><a href="${pageContext.request.contextPath}/article/all">All articles</a></li>

<li><a href="${pageContext.request.contextPath}/logout">Logout</a></li>

</ul>

</nav>

</div>

</header>

<div id="content">

<h1>All articles</h1>

<div class="container">

<table class="f1-table">

<thead>

<tr>

<th>ID</th>

<th>Title</th>

<th>Author</th>

<th>Date</th>

</tr>

</thead>

<tbody>

<c:forEach items="${articles}" var="article">

<tr class="article" onclick="window.location.href='${pageContext.request.contextPath}/article/${article.id}'">

<td>${article.id}</td>

<td>${article.title}</td>

<td>${article.authorString}</td>

<td>${article.publication}</td>

</tr>

</c:forEach>

</tbody>

</table>

</div>

</div>

</body>

</html>

**error.jsp**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<link rel="stylesheet" href="${pageContext.request.contextPath}/style.css">

<title>404</title>

</head>

<body>

<header>

<div class="headerMenu">

<nav>

<ul>

<li></li>

<li></li>

<li><a href="${pageContext.request.contextPath}/profile">Profile</a></li>

<li><a href="${pageContext.request.contextPath}/article/all">All articles</a></li>

<li><a href="${pageContext.request.contextPath}/logout">Logout</a></li>

</ul>

</nav>

</div>

</header>

<div id="content">

<h1>400</h1>

<p>Bad request.</p>

</div>

</body>

</html>

**style.css**

body {

font-family: sans-serif;

color: black;

background-size: cover;

margin-top: 80px;

padding: 30px;

}

header {

background-color: #008848;

position: fixed;

top: 0;

left: 0;

right: 0;

height: 80px;

display: flex;

align-items: center;

justify-content: space-between;

box-shadow: 0 0 25px 0 black;

}

header \* {

display: inline;

}

.headerMenu {

display: flex;

align-items: center;

}

header li {

margin: 20px;

}

header li a {

color: white;

background-color: transparent;

text-decoration: none;

transition-property: color;

transition-duration: 0.5s;

transition-timing-function: ease-out;

}

header li a:hover {

color: #008848;

background-color: white;

transition-property: color;

transition-duration: 0.5s;

transition-timing-function: ease-out;

}

.clear {

clear: both;

}

/\* LOGIN \*/

.login {

width: 300px;

overflow: hidden;

margin: auto;

margin: 20px 0 0 450px;

padding: 80px;

background: #008848;

border-radius: 15px;

& \* {

margin-bottom: 18px;

}

}

h2 {

text-align: center;

color: white;

padding: 20px;

}

label {

color: white;

font-size: 17px;

}

#username {

width: 300px;

height: 30px;

border: none;

border-radius: 3px;

padding-left: 8px;

}

#password {

width: 300px;

height: 30px;

border: none;

border-radius: 3px;

padding-left: 8px;

}

.log-button {

width: 300px;

height: 30px;

border: none;

border-radius: 17px;

padding-left: 7px;

color: #008848;

cursor: pointer;

}

/\* TABLETY \*/

.profile {

table-layout: fixed;

padding: 0;

& tr {

margin: 0;

padding: 0;

& td {

margin: 2px;

}

}

}

.three-tables {

width: 95%;

& tr td {

vertical-align: top;

}

}

.container {

margin-top: 20px;

position: relative;

}

table.f1-table {

width: 800px;

border-collapse: collapse;

overflow: hidden;

box-shadow: 0 0 20px rgba(0,0,0,0.1);

& td {

padding: 15px;

background-color: rgba(255,255,255,0.2);

color: #000000;

}

& th {

padding: 15px;

background-color: rgba(255,255,255,0.2);

text-align: left;

color: white

}

& thead {

& th {

background-color: #029f56;

}

}

& tbody {

& tr.article {

cursor: pointer;

}

& tr {

&:hover {

background-color: rgba(255,255,255,0.3);

}

}

& td {

position: relative;

&:hover {

&:before {

content: "";

position: absolute;

left: 0;

right: 0;

top: -9999px;

bottom: -9999px;

background-color: rgba(255,255,255,0.2);

z-index: -1;

}

}

}

}

}

/\* BUTTon \*/

.button-3 {

appearance: none;

background-color: #2ea44f;

border: 1px solid rgba(27, 31, 35, .15);

border-radius: 6px;

box-shadow: rgba(27, 31, 35, .1) 0 1px 0;

box-sizing: border-box;

color: #fff;

cursor: pointer;

display: inline-block;

font-family: -apple-system,system-ui,"Segoe UI",Helvetica,Arial,sans-serif,"Apple Color Emoji","Segoe UI Emoji";

font-size: 14px;

font-weight: 600;

line-height: 20px;

padding: 6px 16px;

position: relative;

text-align: center;

text-decoration: none;

user-select: none;

-webkit-user-select: none;

touch-action: manipulation;

vertical-align: middle;

white-space: nowrap;

}

.button-3:focus:not(:focus-visible):not(.focus-visible) {

box-shadow: none;

outline: none;

}

.button-3:hover {

background-color: #2c974b;

}

.button-3:focus {

box-shadow: rgba(46, 164, 79, .4) 0 0 0 3px;

outline: none;

}

.button-3:disabled {

background-color: #94d3a2;

border-color: rgba(27, 31, 35, .1);

color: rgba(255, 255, 255, .8);

cursor: default;

}

.button-3:active {

background-color: #298e46;

box-shadow: rgba(20, 70, 32, .2) 0 1px 0 inset;

}

## **ПРИЛОЖЕНИЕ В. Файлы конфигурации**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.bonch</groupId>

<artifactId>shteinberg\_22</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.5.1</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>32.1.3-jre</version>

</dependency>

</dependencies>

</project>

**Web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<error-page>

<location>/error.jsp</location>

</error-page>

</web-app>