Балаковский инженерно-технологический институт - филиал

федерального государственного автономного образовательного учреждения

высшего образования

«Национальный исследовательский ядерный университет «МИФИ»

Факультет атомной энергетики и технологий

Кафедра «Информационные системы и технологии»

КОНТРОЛЬНАЯ РАБОТА

по дисциплине

«Современные технологии разработки распределенного

программного обеспечения»

Вариант 12

Выполнил: студент группы ИФСТ-4з

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кособрюхова М.С.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

|  |  |
| --- | --- |
|  | Проверил: ст.пр. кафедры ИСТ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ефремов Р.В.  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г. |

Балаково 2023

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Системы хранения ключ-значение NoSQL

Расцвет реляционных (relation — «связь, взаимосвязь») баз данных пришёлся на 80-е годы, когда в БД в основном хранили текстовые документы и изображения. Однако с развитием технологий и ростом объёма обрабатываемой информации реляционные СУБД перестали справляться со всеми задачами одинаково хорошо. Термин NoSQL впервые прозвучал в 1998 году: его применил итальянский учёный Карло Строцци для описания своей open source СУБД. При разработке он отказался от SQL, а также от основного принципа реляционных СУБД — ACID (atomicity, consistency, isolation, durability).

В начале XXI века NoSQL БД стали популярны, в том числе у корпораций. Чтобы решить проблемы параллельных вычислений с очень большим объёмом информации и масштабируемости, Google построила на основе модели распределённых вычислений MapReduce колоночное хранилище. На базе этих технологий выросло целое семейство высокодоступных open source СУБД.

Основные причины появления NoSQL

Возникла потребность в распределённых СУБД. Появление IT-корпораций, глобальных приложений, социальных сетей потребовало масштабирования баз данных. Вертикальное масштабирование железа — удовольствие дорогое, а шардированию реляционные БД поддаются плохо: чем больше в системе серверов, тем больше усилий требуется для поддержания согласованности данных в узлах.

Работа с данными ускорилась. В отличие от нереляционных БД, SQL запрашивает данные из нескольких таблиц. А когда количество информации растёт, таблиц и связей становится слишком много, поэтому скорость получения ответа на запрос снижается.

Разработчики стремились избавиться от ограниченности реляционных схем. Жёсткая реляционная модель подходит не для всех предметных областей. Иногда они слишком сложны или часто требуют корректировки данных. В итоге получается либо нагромождение избыточного количества таблиц, либо плохо структурированная предметная область.

SQL и NoSQL: в чём разница

Управление

Работа с реляционной базой строится на общепринятом языке SQL. У NoSQL СУБД нет единого стандарта: у каждой такой базы индивидуальный подход к записи, хранению и извлечению данных. Поиск информации может вестись, например, по парам «ключ — значение» или по наборам столбцов.

Структура

Для реляционных БД необходима фиксированная схема — точное описание структуры таблицы, её полей и ограничений. Данные в строках таблиц должны быть нормализованы, а добавление нового свойства влечёт за собой изменение всей схемы. Этот подход эффективен, когда необходима стабильность и чёткая структура. Нереляционные СУБД существуют по другим принципам и законам, в них структура не регламентирована, отсюда и принципиальное отличие NoSQL от реляционных БД — их гибкость.

СУБД без SQL позволяют записывать и получать неструктурированную информацию, дают возможность менять схемы и запросы в соответствии с требованиями к данным. Когда речь идёт о работе с time-series data, информацией без взаимосвязи, о хранении документов с разной структурой и высокой степенью вложенности, то именно NoSQL показывают большую скорость и производительность.

ACID-транзакции

При разработке NoSQL были смягчены жесткие требования к транзакциям. Основные принципы ACID гарантируют, что целостность и согласованность данных в реляционных хранилищах будет обеспечена даже при сбоях. Поскольку для ряда задач в строгом следовании ACID нет никакой необходимости, NoSQL-базы чаще всего предлагают компромисс и принципы BASE:

* система обеспечивает базовую доступность (Basic Availability), т. е. каждый запрос будет обязательно завершён, успешно или нет;
* система пребывает в гибком состоянии (Soft-state) — очерёдность записей соблюдать необязательно, реплики могут какое-то время находиться в несогласованном состоянии, а система может самостоятельно изменяться для достижения согласованности;
* все данные всё равно достигнут согласованности (Eventual Consistency).

Плюсы NoSQL

Гибкость модели данных

NoSQL-подход разрешает группировать любой набор данных и их связей. Объект данных при этом может быть многосоставным: нормализация в NoSQL не требуется. Это свойство будет полезно, например, стартапам, когда модель хранилища данных меняется во время разработки самого продукта и в начале работы не удаётся спрогнозировать финальную архитектуру БД.

Доступность данных

Благодаря механизмам отказоустойчивости секций, в том числе репликации и шардированию, NoSQL способна в любой момент обслужить входящий запрос и вернуть не ошибочный ответ. Избыточность и отказоустойчивость достигаются благодаря репликации данных на узлах. И даже если реплики окажутся недоступны, можно произвести запись в базу данных. По мере доступности узлов каждая реплика будет обновлена.

Лучшая масштабируемость

Поскольку между записями в NoSQL нет жёсткой связи, данные можно дробить и хранить на нескольких независимых серверах. Горизонтальное масштабирование легче и дешевле, чем вертикальное, присущее SQL-моделям. Производительность реляционной системы БД увеличивается с помощью дополнительного дорогого оборудования (и то не бесконечно), а в нереляционных БД — с помощью добавления новых узлов. Это делает NoSQL более удобными при взаимодействии с большими или меняющимися наборами данных.

Высокая производительность

Благодаря оптимизации баз под определённые виды моделей данных скорость представления информации часто превосходит скорость SQL-базы. Например, если необходимые записи хранятся в одном документе, больше нет потребности в операции JOIN. Это упрощает процесс, когда необходимо провести аналитику над данными, агрегацию или расчёты внутри сущности.

Экономия ресурсов

Горизонтальное масштабирование позволяет сократить количество дорогостоящих серверов. А поскольку большинство NoSQL являются open source проектами, можно экономить на подписке и поддержке. Или развернуть и эксплуатировать базы данных в облачном сервисе: NoSQL благодаря распределённости и горизонтальной масштабируемости прекрасно для этого подходят.

Yandex Cloud, например, предлагает сервисы для управления нереляционными базами данных: MongoDB, Redis™, Elasticsearch, ClickHouse.

Базы данных по принципу «ключ — значение» (key-value store)

В этой БД записи хранятся в парах «ключ — значение», где ключ выступает уникальным идентификатором. Ключи и значения фиксируются в виде простой или составной информации. Эти хранилища максимально быстро реагируют на запросы информации и прекрасно масштабируются.

Key-value СУБД часто используется для систем, в которых скорость является приоритетом, а данные не слишком сложные. Например, для хранения кеша данных, онлайн-списков, обработки истечения срока действия, разделения сеансов, построения рейтинга и прочих задач.

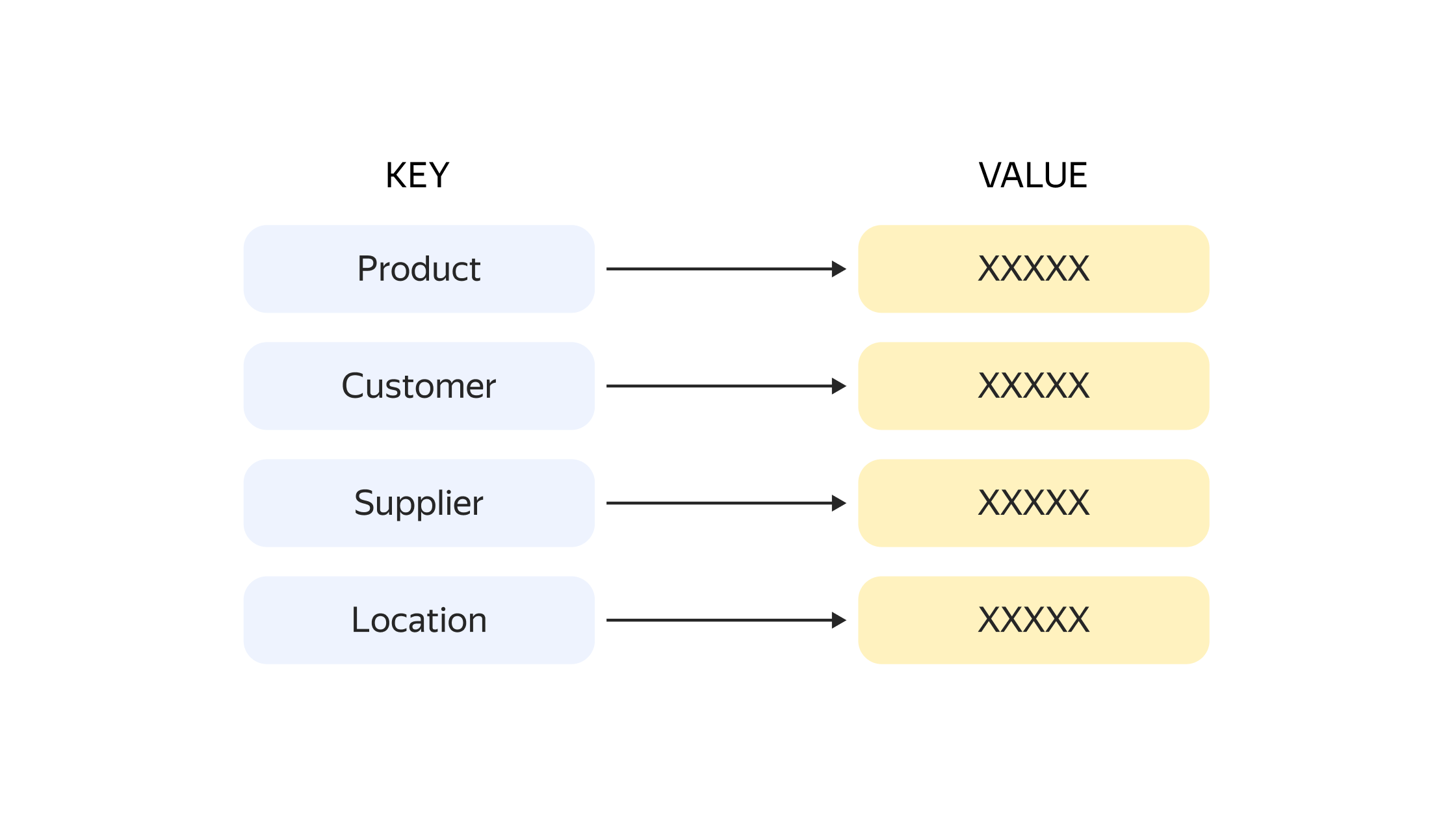


Рисунок 1 – Схема базы данных NoSQL

Отсутствие схемы в базах данных «ключ-значение», например, Riak, — это как раз то, что вам нужно для хранения данных. Ключ может быть синтетическим или автосгенерированным, а значение может быть представлено строкой, JSON, блобом (BLOB, Binary Large Object, большой двоичный объект) и т.д.

Такие базы данных как правило используют хеш-таблицу, в которой находится уникальный ключ и указатель на конкретный объект данных. Существует понятие блока (bucket) — логической группы ключей, которые не группируют данные физически. В разных блоках могут быть идентичные ключи.

Производительность сильно вырастает за счёт кеширующих механизмов, которые работают на основе маппингов. Чтобы прочитать значение, нужно знать как ключ, так и блок, поскольку на самом деле ключ является хешем (блок + ключ).

В модели «ключ-значение» нет ничего сложного, так как реализовать её проще простого. Не лучший способ, если нужно только обновить часть значения или сделать запрос к базе данных.

Если поразмыслить о теореме CAP, то становится довольно очевидно, что такие хранилища хороши в плане доступности (Availability) и устойчивости к разделению (Partition tolerance), но явно проигрывают в согласованности данных (Consistency).

Пример: посмотрим на набор данных, представленных таблицей ниже. Здесь ключ — это название страны, а значение — список адресов в этой стране:

Таблица 1 – «Ключ – значение»

|  |  |
| --- | --- |
| Key | Value |
| “India” | {“B-25, Sector-58, Noida, India - 201301”} |
| “Romania” | {“IMPS Moara Buisiness Center, Buftea No. 1, Cluj-Napoca 400606, City Business Cennet, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011”} |
| “US” | {“3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033”} |

База данных такого типа позволяет читать и записывать значения с помощью ключа следующим образом:

* Get(key) возвращает значение, связанное с переданным ключом;
* Put(key, value) связывает значение с ключом;
* Multi-get(key1, key2, ..., keyN) возвращает список значений, связанных с переданным ключами;
* Delete(key) удаляет запись для ключа из хранилища.

И хотя базы данных типа «ключ-значение» могут пригодиться в определённых ситуациях, они не лишены недостатков. Первый заключается в том, что модель не предоставляет стандартные возможности баз данных вроде атомарности транзакций или согласованности данных при одновременном выполнении нескольких транзакций. Такие возможности должны предоставляться самим приложением.

Второй недостаток в том, что при увеличении объёмов данных, поддержание уникальных ключей может стать проблемой. Для её решения необходимо как-то усложнять процесс генерации строк, чтобы они оставались уникальными среди очень большого набора ключей.

Riak и Dynamo от Amazon — самые популярные СУБД данных такого типа.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Для выполнения контрольной работы была определена тема серверного приложение «Библиотека» и написана страница регистрации пользователей на сайт.

База данных

База данных library была создана на сервере MySQL в которой определена таблица users, созданная с помощью следующей команды:

CREATE TABLE products (id INT PRIMARY KEY AUTO\_INCREMENT, login VARCHAR(30), pass VARCHAR(30), telefon VARCHAR(12), email VARCHAR(30), access INT)

При запуске приложения запускается страница index.jsp, где выводится обычная главная страница с меню.

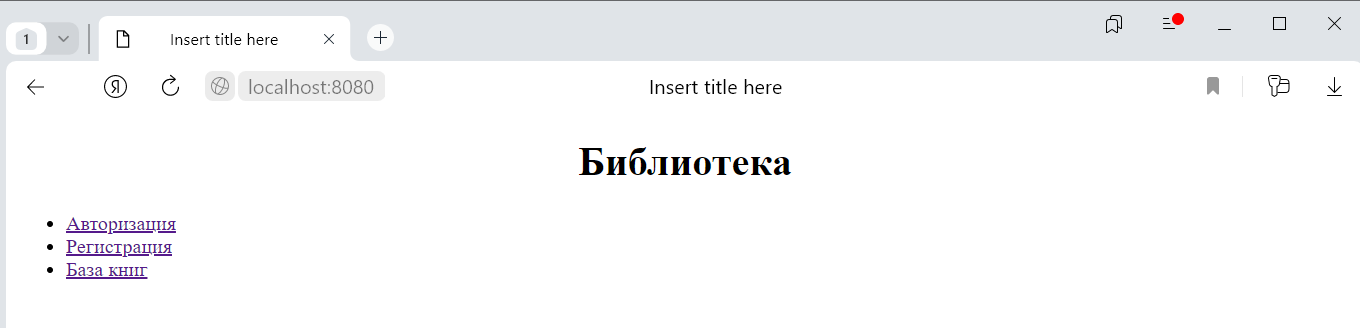


Рисунок 1 – Главная страница

Код страницы index.jsp:

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

pageEncoding="utf-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Insert title here</title>

</head>

<body>

<h1 align="center">Библиотека</h1>

<ul>

<li><a href=" ">Авторизация</a></li>

<li><a href="http://localhost:8080/Booklibrary/registration.jsp">Регистрация</a></li>

<li><a href=" ">База книг</a></li>

</ul>

</body>

</html>

При нажатии на кнопку меню «Регистрация» происходит переход на страницу registration.jsp, которая выдает форму для введения данных нового пользователя.

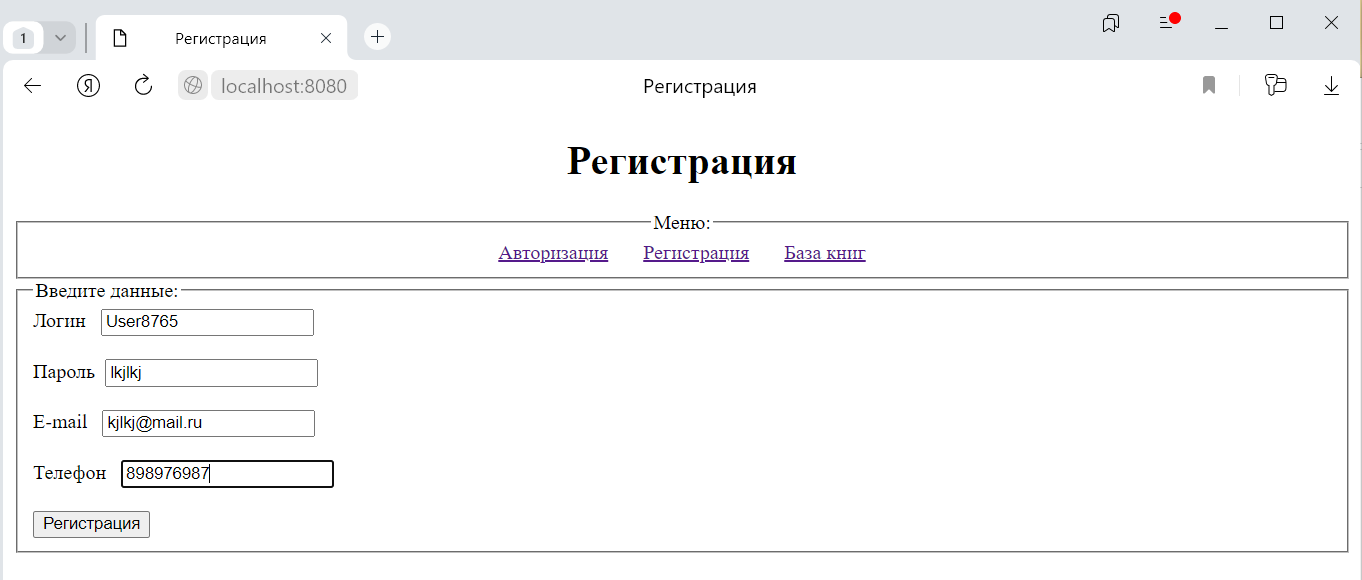


Рисунок 2 – Форма регистрации

Код страницы registration.jsp:

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

pageEncoding="utf-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Регистрация</title>

</head>

<body>

<h1 align="center">Регистрация</h1>

<fieldset align="center">

<legend>Меню:</legend>

<label><a href=""> Авторизация </a> </label>

<label><a href= "http://localhost:8080/Booklibrary/registration.jsp" > Регистрация </a> </label>

<label><a href="">База книг</a></label>

</fieldset>

<form action="/Booklibrary/Registration" method="get">

<fieldset>

<legend>Введите данные:</legend>

<label>Логин &nbsp; <input type="text" name="login"></label><br><br>

<label>Пароль &nbsp;<input type="text" name="pass"></label><br><br>

<label>E-mail &nbsp;&nbsp;<input type="email" name="email"></label><br><br>

<label>Телефон &nbsp;&nbsp;<input type="text" name="telefon"></label><br><br>

<button type="submit">Регистрация</button>

</fieldset>

</form>

</body>

</html>

При нажатии на кнопку «Регистрация» происходит передача данных через get-запрос сервлету Registration.java. Он в свою очередь получает данные и выводит их на страницу. Далее подключается к базе данных и добавляет в нее новые данные. Следом делает запрос к базе данных и выводит список всех зарегистрированных пользователей.

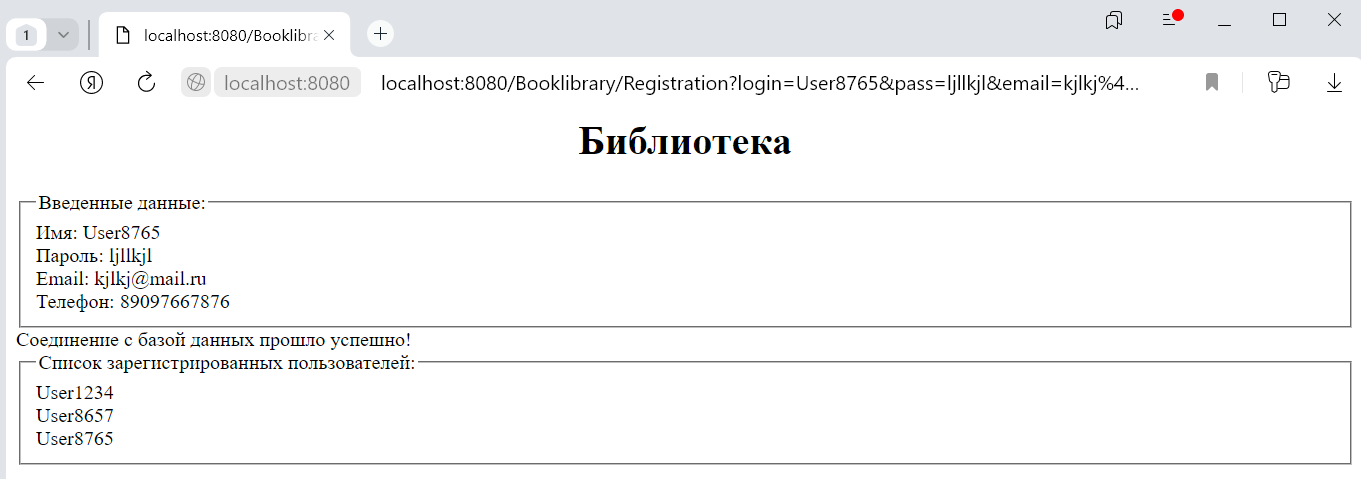


Рисунок 3 – Ответ сервера

Код сервлета Registration.java:

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class Registration

\*/

@WebServlet("/Registration")

public class Registration extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public Registration() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

response.setCharacterEncoding("UTF-8");

String login = request.getParameter("login");

String pass = request.getParameter("pass");

String email = request.getParameter("email");

String telefon = request.getParameter("telefon");

PrintWriter pw = response.getWriter();

pw.println("<html>");

pw.println("<head><meta charset=\"utf-8\"></head>");

pw.println("<body>");

pw.println("<h1 align=\"center\">Библиотека</h1>");

pw.println("<fieldset>");

pw.println("<legend>Введенные данные:</legend>");

pw.println("<label> Имя: " + login + "</label><br>");

pw.println("<label> Пароль: " + pass + "</label><br>");

pw.println("<label> Email: " + email + "</label><br>");

pw.println("<label> Телефон: " + telefon + "</label><br>");

pw.println("</fieldset>");

String rqs = "INSERT INTO users (login, pass, telefon, email, access) VALUES ('" + login + "', '" + pass + "', '" + telefon + "', '" + email + "', 2)";

try{

String url = "jdbc:mysql://localhost:3306/library";

String username = "root";

String password = "vfhbyf14";

Class.forName("com.mysql.cj.jdbc.Driver").getDeclaredConstructor().newInstance();

try (Connection conn = DriverManager.getConnection(url, username, password)){

pw.println("Соединение с базой данных прошло успешно!<br>");

pw.println("<fieldset>");

pw.println("<legend>Список зарегистрированных пользователей:</legend>");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(rqs);

stmt.close();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

while(rs.next()) {

pw.println("<label>" + rs.getString("login") + "</label><br>");

pw.println("<br>");

}

stmt.close();

pw.println("</fieldset>");

pw.println("</body>");

pw.println("</html>");

}

}

catch(Exception ex){

pw.println("Ошибка соединения!");

pw.println(ex);

}

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}