# Department of Computing

**Course Code: CS-432**

**Class: BSCS-12ABC**

Lab 11: Indexing, Importing and Searching data in Apache Solr

**Date: 25th April ,2025**

**Instructor: Dr. Khurram Shahzad, Dr. Shah Khalid**

**Submitted By: Aamna Khan**

**Class: BSCS-12(B)**

**CMS ID: 418189**

Lab 11: Indexing, Importing and Searching data in Apache Solr

**Introduction**

Apache Solr is a **high-performance, scalable, full-text search engine**. It’s widely used in applications that require complex searching and indexing of large volumes of data, like e-commerce websites, enterprise applications, and big data platforms.

**Indexing Data:**

Indexing enables users to locate information in a document. Indexing collects, parses, and stores documents. Indexing is done to increase the speed and performance of a search query while finding a required document.

**Importing Data**

A Solr index can accept data from many different sources, including XML files, comma-separated value (CSV) files, data extracted from tables in a database, and files in common file formats such as Microsoft Word or PDF.

Here are the three most common ways of loading data into a Solr index:

* Using the [Solr Cell](https://solr.apache.org/guide/6_6/uploading-data-with-solr-cell-using-apache-tika.html" \l "uploading-data-with-solr-cell-using-apache-tika) framework built on Apache Tika for ingesting binary files or structured files such as Office, Word, PDF, and other proprietary formats.
* Uploading XML files by sending HTTP requests to the Solr server from any environment where such requests can be generated.
* Writing a custom Java application to ingest data through Solr’s Java Client API (which is described in more detail in [Client APIs](https://solr.apache.org/guide/6_6/client-apis.html#client-apis)). Using the Java API may be the best choice if you’re working with an application, such as a Content Management System (CMS), that offers a Java API.

**Searching:**

Solr offers a rich, flexible set of features for search. To understand the extent of this flexibility, it’s helpful to begin with an overview of the steps and components involved in a Solr search.

When a user runs a search in Solr, the search query is processed by a request handler. A request handler is a Solr plug-in that defines the logic to be used when Solr processes a request. Solr supports a variety of request handlers. Some are designed for processing search queries, while others manage tasks such as index replication.

Solr is the popular, blazing fast, open source NoSQL search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document handling, and geospatial search. Solr is highly scalable, providing fault tolerant distributed search and indexing, and powers the search and navigation features of many of the world's largest internet sites.

**Objectives**

To install, setup and understanding the basics of Apache solr.

**Tools/Software Requirement**

Apache solr

**Useful tutorial links:**

<https://solr.apache.org/guide>/

<https://blog.kiprosh.com/distributed-indexing-and-searching-with-apache-solrcloud>

<https://solr.apache.org/guide/6_6/distributed-search-with-index-sharding.html>

**Description**

**Lab Prerequisite:**

* Install & configure solr.
* Create a cluster of servers(shards).

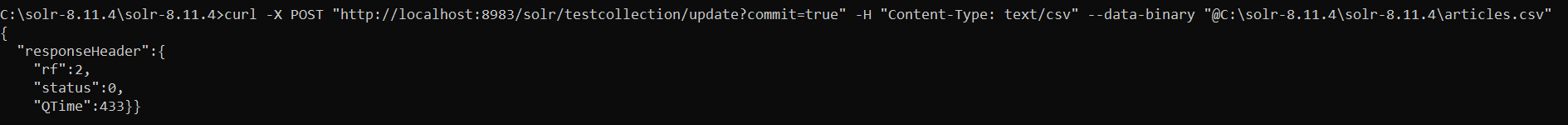
Installation guide of Apache Solr for windows and Ubuntu are available on LMS if you have not installed it yet, follow the guide and install and configure it first.

**Task  
  
Task 1**

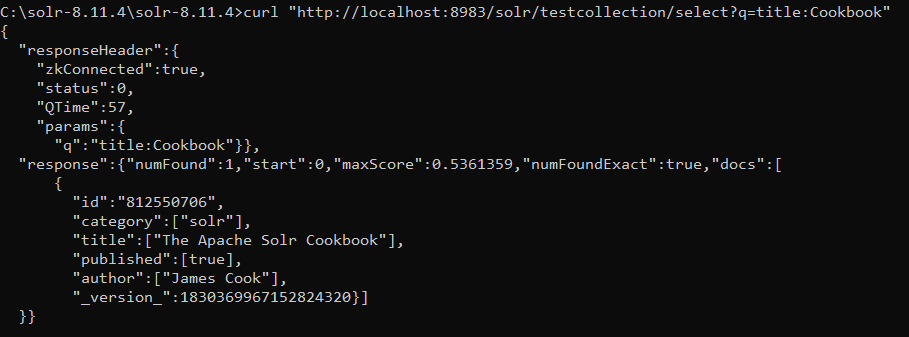
You have already created a cluster of servers(shards) now take a sample data Index it, Import and perform searching on it.

Sample data is available on the following link.

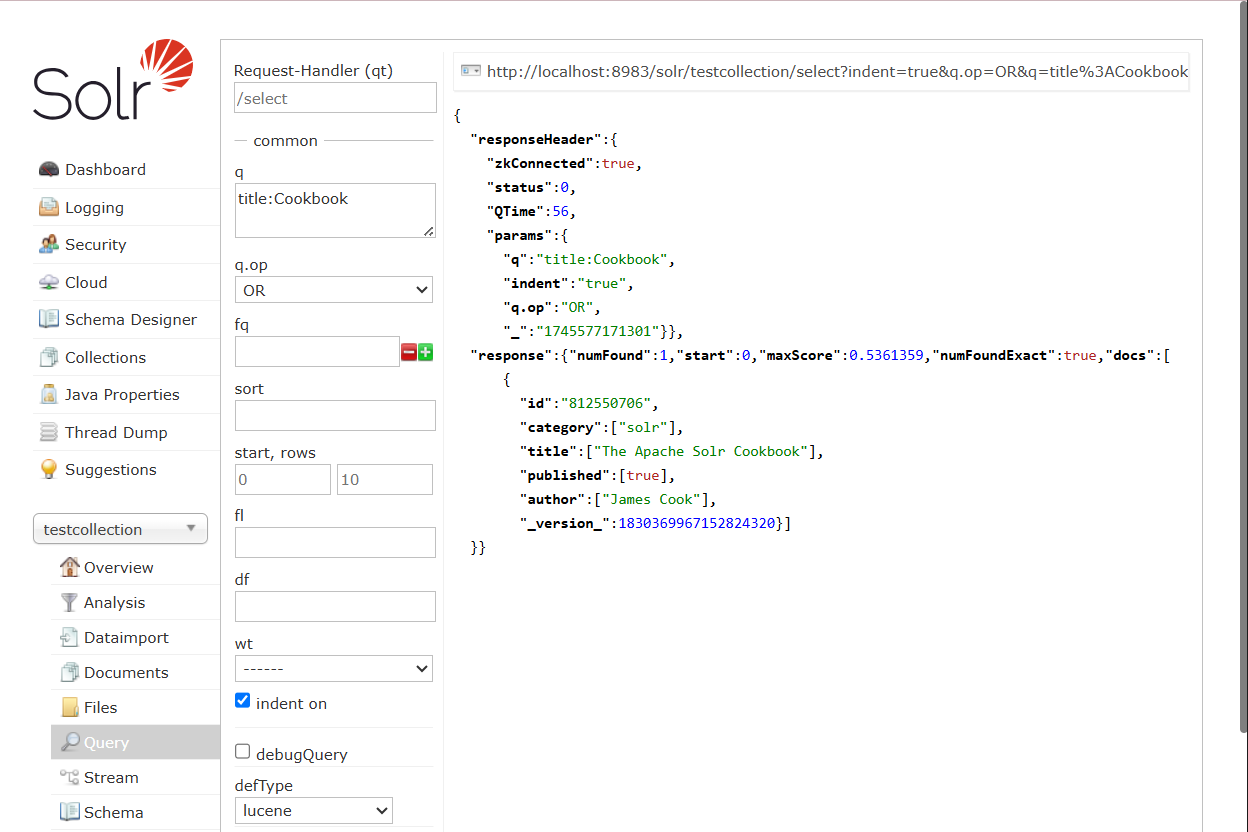
***Upload .csv on collection-testcollection:***



***Search with cmd:***



***Search with Solr GUI:***

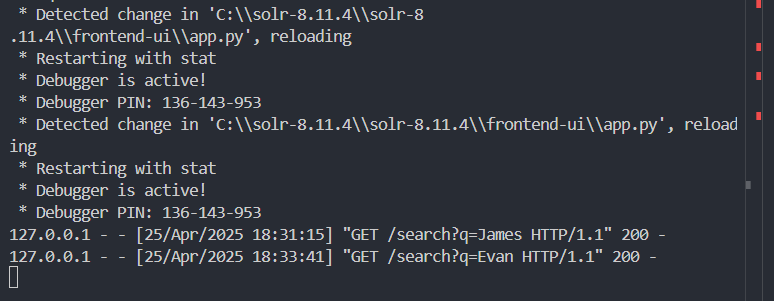


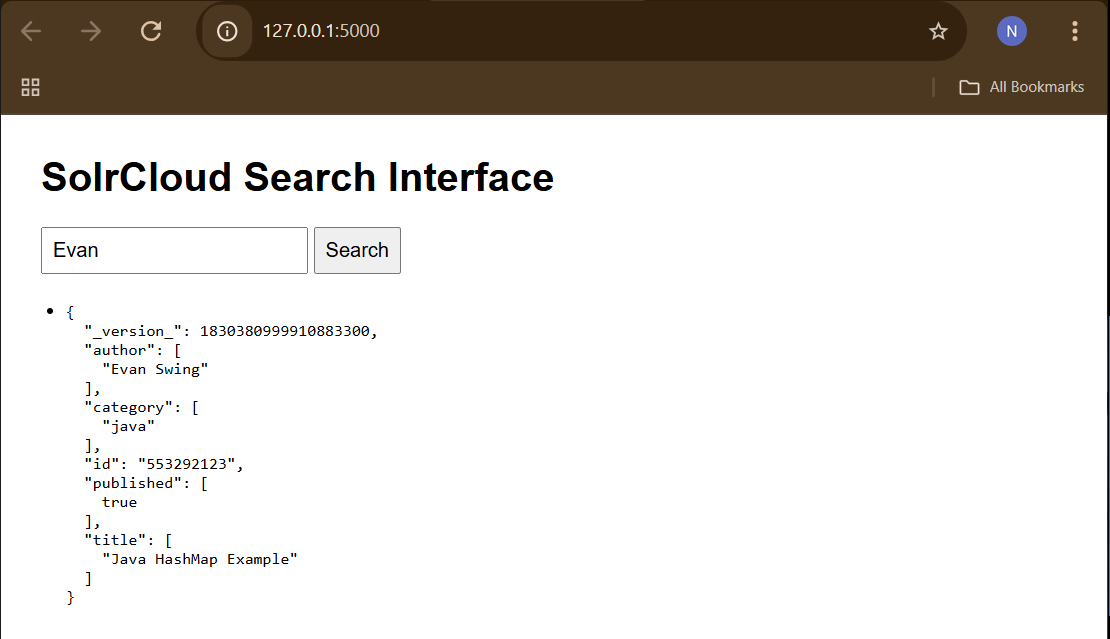
Useful Links:   
<https://examples.javacodegeeks.com/apache-solr-clustering-example/>

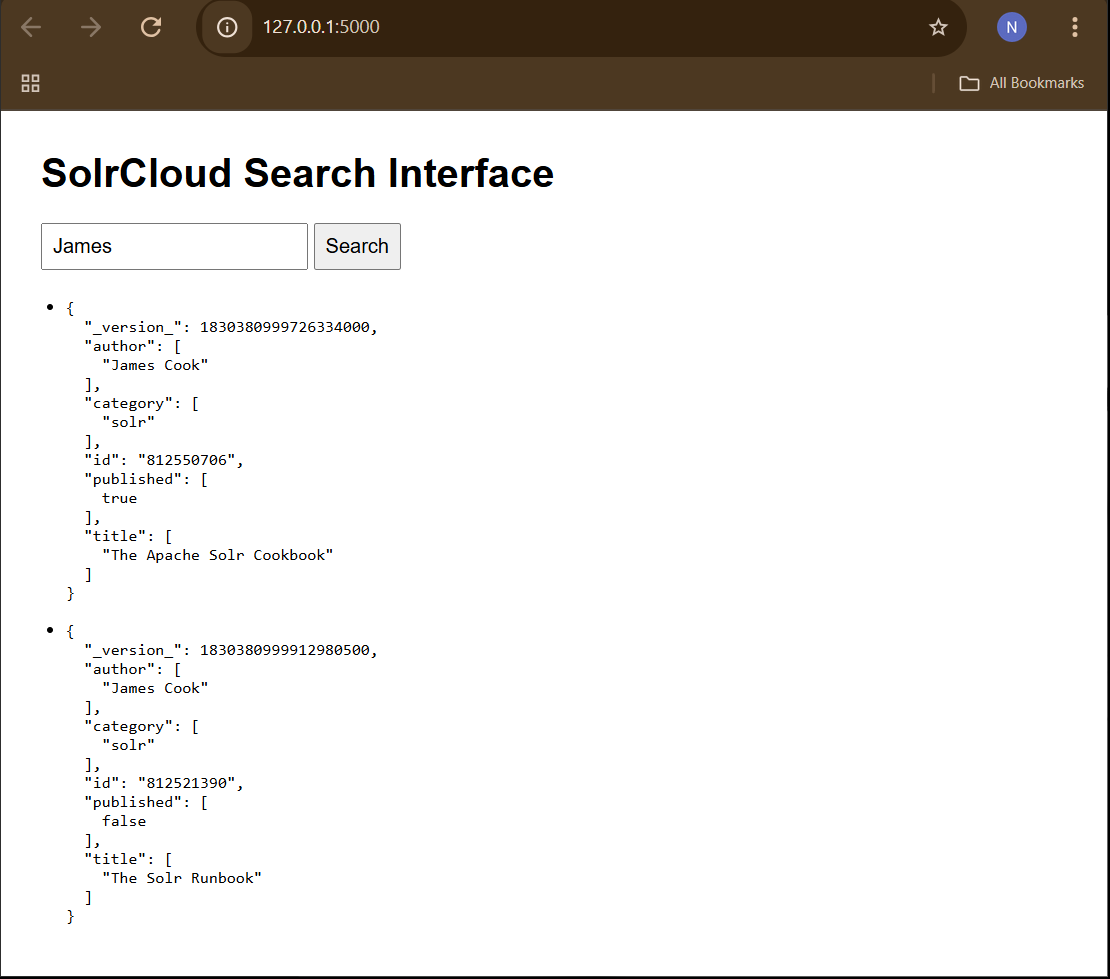
### Task 2: Integrate Solr with a Web Interface

* Use a frontend framework (e.g., React, Flask, or Node.js) to build a search UI.
* Connect the UI with Solr backend via REST API.
* Display real-time search results with autocomplete and filters.

***Output Frontend-UI:***







***Code:***

|  |
| --- |
| *#app.py*  from flask import Flask, render\_template, request, jsonify  import requests  app = Flask(\_\_name\_\_)  *# ✅ Make sure this matches your actual SolrCloud collection*  SOLR\_URL = 'http://localhost:8983/solr/books/select'  @app.route('/')  def home():      return render\_template('index.html')  @app.route('/search')  def search():      query = request.args.get('q', '').strip()      if not query:          return jsonify([])      params = {          'q': query,  *# 🔍 Use a specific field if needed (like 'title:')*          'wt': 'json',          'df': 'author',          'rows': 10,          'shards.tolerant': 'true'      }      try:          solr\_response = requests.get(SOLR\_URL, params=params)          solr\_response.raise\_for\_status()          results = solr\_response.json()          if 'response' in results and 'docs' in results['response']:              return jsonify(results['response']['docs'])          else:              return jsonify([])      except Exception as e:          print("[❌] Solr request failed:", e)          return jsonify([])  if \_\_name\_\_ == '\_\_main\_\_':      app.run(debug=True) |
| #index.html  <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <title>Solr Search</title>    <style>      body { font-family: Arial, sans-serif; margin: 2em; }      input, button { padding: 8px; font-size: 16px; }      ul { margin-top: 20px; padding-left: 20px; }      li { margin-bottom: 8px; }    </style>    <script>      async function fetchResults() {        const query = document.getElementById("searchBox").value.trim();        if (!query) return;        try {          const response = await fetch(`/search?q=${encodeURIComponent(query)}`);          const results = await response.json();          let html = "";          if (results.length > 0) {            results.forEach(doc => {              html += `<li><pre>${JSON.stringify(doc, null, 2)}</pre></li>`;            });          } else {            html = "<li>No results found</li>";}          document.getElementById("results").innerHTML = html;        } catch (err) {          document.getElementById("results").innerHTML = "<li>Error fetching results</li>";          console.error("Fetch failed:", err);        }      }      window.onload = function () {        document.getElementById("searchBtn").addEventListener("click", fetchResults);        document.getElementById("searchBox").addEventListener("keyup", function (e) {          if (e.key === "Enter") fetchResults();        });      };    </script>  </head>  <body>    <h1>SolrCloud Search Interface</h1>    <input type="text" id="searchBox" placeholder="Type to search...">    <button id="searchBtn">Search</button>    <ul id="results"></ul>  </body>  </html> |

**Deliverables**

Upload single word file with screenshots of your output on LMS.