LIBRERA SEARCH SOLUTION

Enables indexing content from PDFs making it searchable with the possibility to classify them by the Web UI

Contents

Doc. Version	Date	Author	Description
1.0	2025-12-05	Name and surname	First version of the release

Requirements

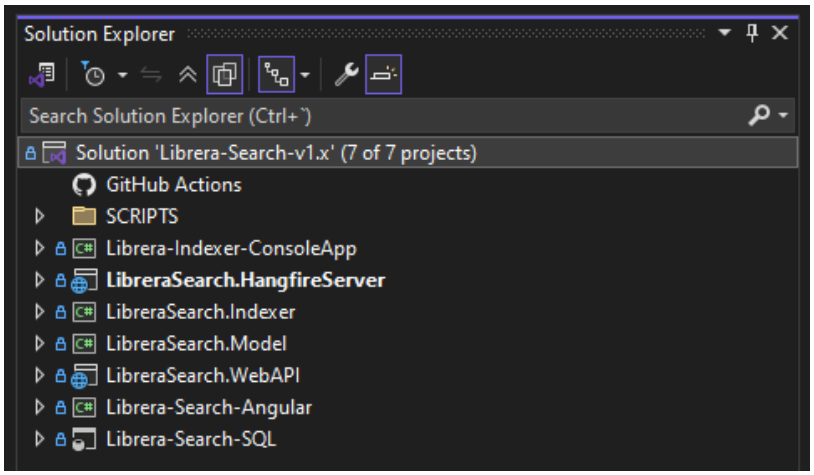
The next table shows the requirements for the solution described.

Requirement	Description
RQ-001 | Automated Task Service	Hangfire offers the possibility to manage automated task services to process PDFs
RQ-002 | Indexer	The indexer libraries that contains the logic to process the PDFs
RQ-003 | Model Layer	The model layer with EF core to interact with database
RQ-004 | Web API	The back-end to execute CRUD operations in server
RQ-005 | Front end App	Angular App to offer UI to manage the records and associated records
RQ-006 | Database	The data model, full text search and index services and the logic needed by Stored Procedures

Overview

This document contains the technical details of the Librera Search v1.x solution built with Angular 16, Hangfire Server and .Net 8; enables to index content from PDFs making it searchable with the possibility to classify them by the Web UI. Implements token-based authentication for registered users and policy-based authorization in the endpoints.

Here the solution structure in Visual Studio 2022:



The application contains the next functionality:

- Each register has id, title ,authors ,series, ids, published, publisher, languages, tags, formats, path and indexed content
- We can create, retrieve, update, delete records, launch the index service and search PDF content linked to the records
- Includes two pages for finding registers by title or search indexed content
- JWT (JSON Web Token) format used for authentication
- Policy-based authorization in the endpoints

Token-based authentication is a popular approach to securing web applications, providing a stateless and scalable way to verify user identities. In this implementation, we'll explore how to integrate token-based authentication using Angular 16 as the frontend framework and .Net 8 WebAPI as the backend API.

Token-based authentication involves generating a unique token upon successful user authentication, which is then passed with each subsequent request to verify the user's identity. This approach eliminates the need for server-side session storage, making it ideal for modern web applications.

Key Components:

- .Net 8 WebAPI: Handles token generation and validation
- Angular 16: Manages user authentication and includes the token in requests

- JWT (JSON Web Token): The token format used for authentication

This implementation provides a secure and efficient way to authenticate users, allowing for seamless interaction between the frontend and backend.

In ASP.NET, policies are used to define requirements that must be met in order to give a user the authorization to access an endpoint.

Within Web API projects you can use policies to guard and protect endpoints from unauthorized users that don't meet the criteria. Instead of writing the authorization logic in the endpoint itself, you can refactor this into a policy that is reusable across multiple endpoints. By extracting the authorization logic from the endpoint, you can make your code more clear and keep your endpoints concise.

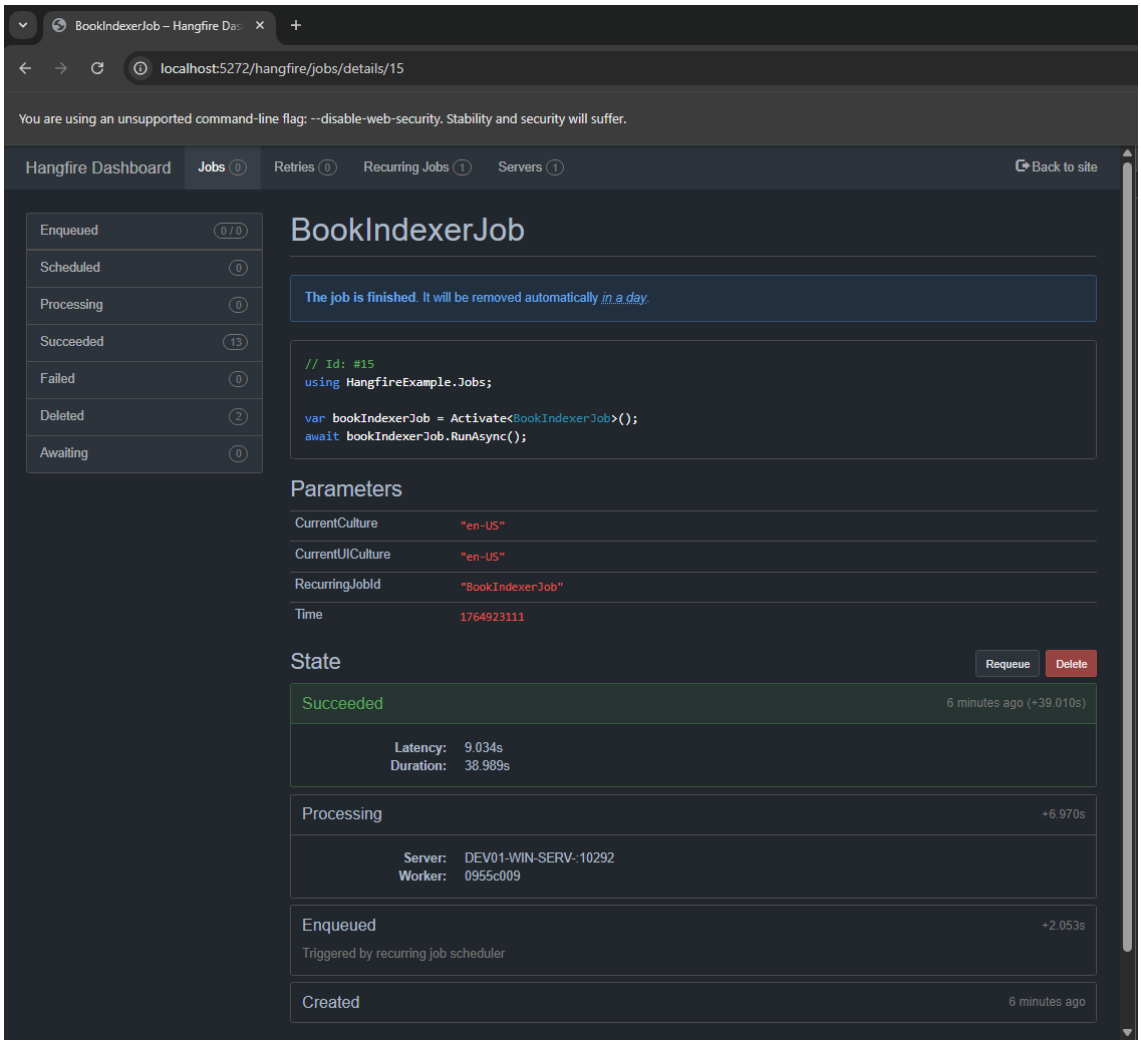
Scope

This first version of the solutions grants a view of how could be easily implemented in .NET 8 a simple but effective tool to manage book records and make content searchable using open source elements like Hangfire and front-end technologies like Angular.

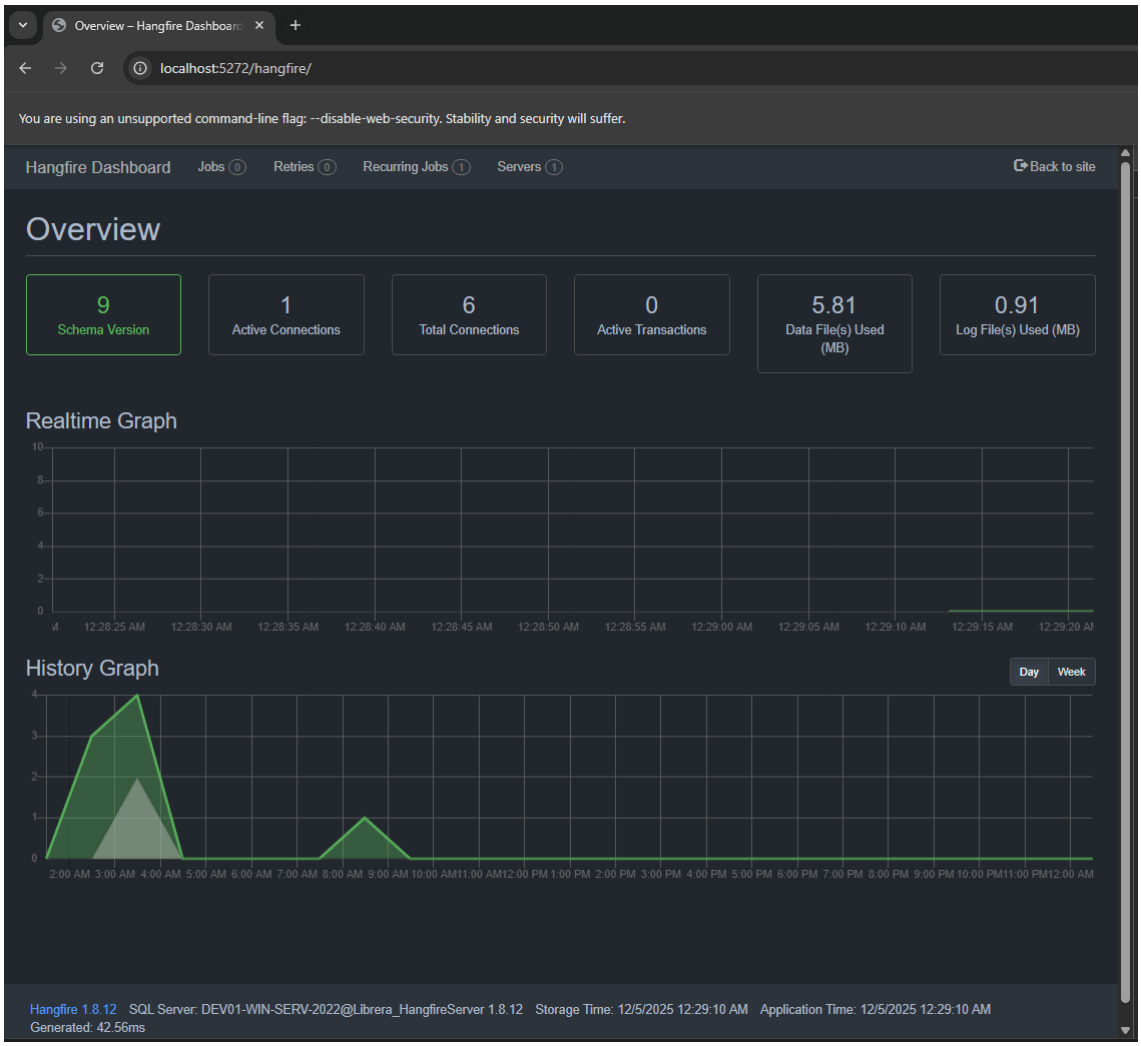
RQ-01 | Automated Task Service

An easy way to perform background processing in .NET and .NET Core applications. No Windows Service or separate process required. Backed by persistent storage. Open and free for commercial use.

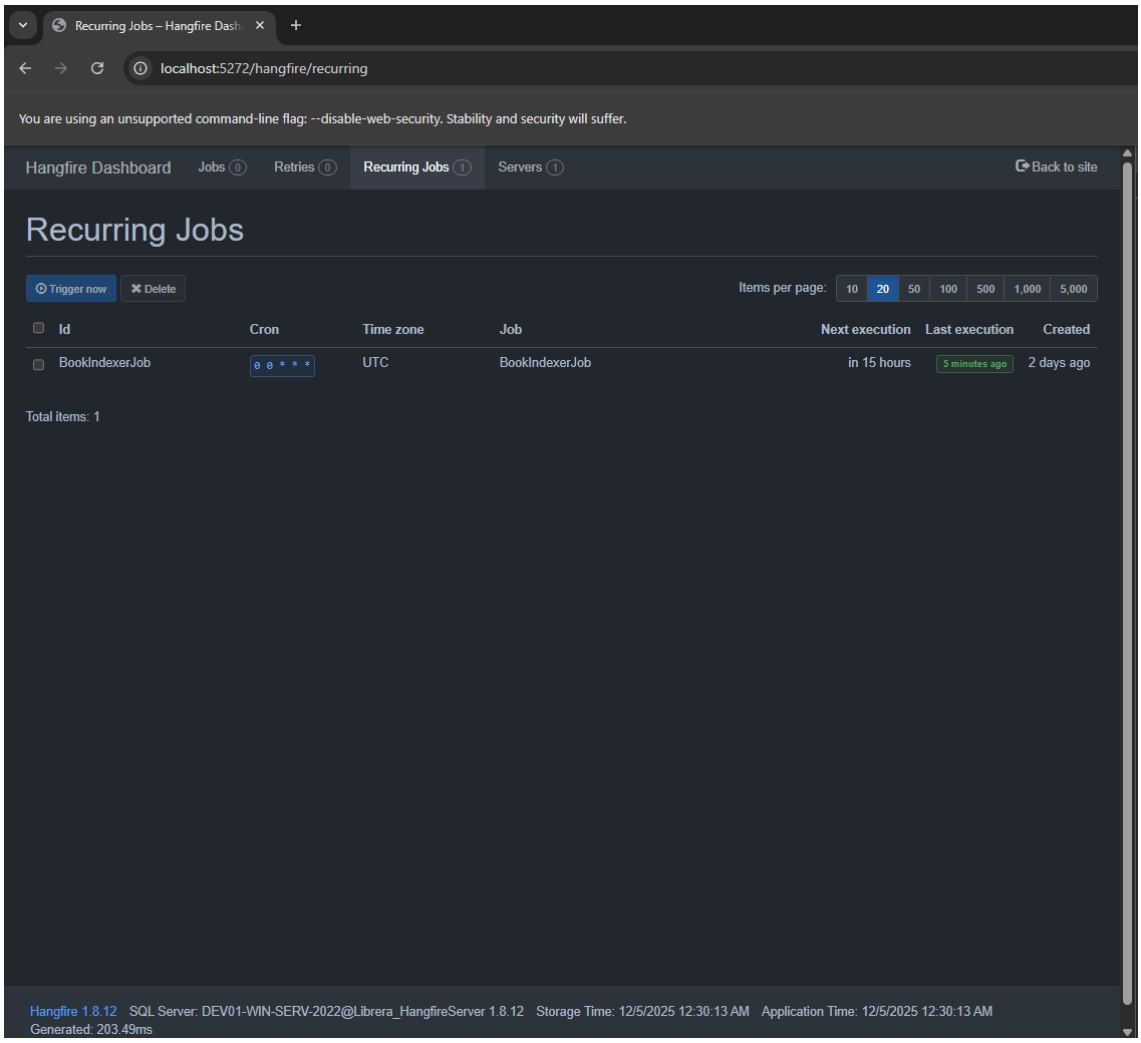
Here the Book Indexer Job definition in the dashboard:



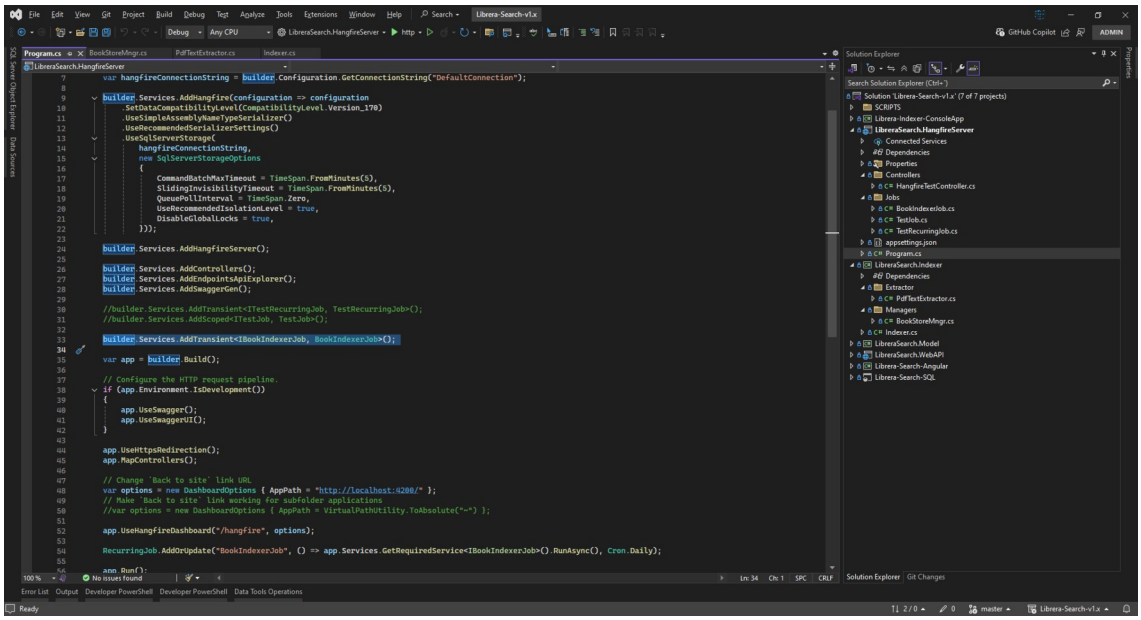
The next image image shows the process graph:



Here the recurring Job:

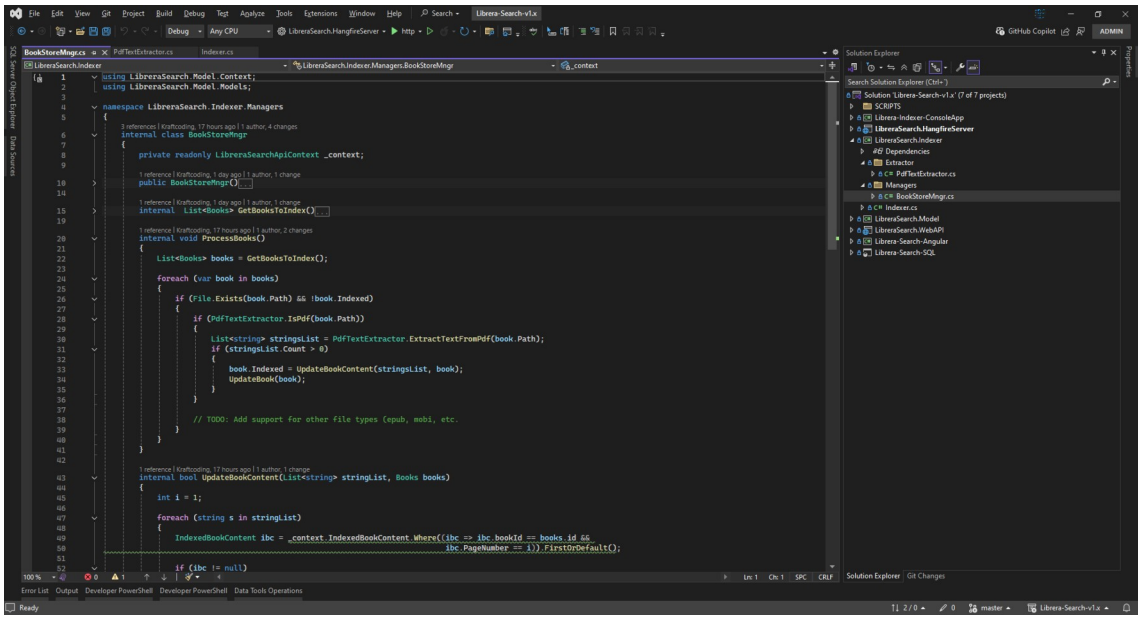


Here the Hangfire service definition for the Job:



RQ-02 | Indexer

The indexer is implemented by a library type project on which we have the PDF extractor and the c# method that the Hangfire service will call by a recurring job definition.



The indexer extraccts content form PDFs and stores it in DB.

RQ-03 | Model Layer

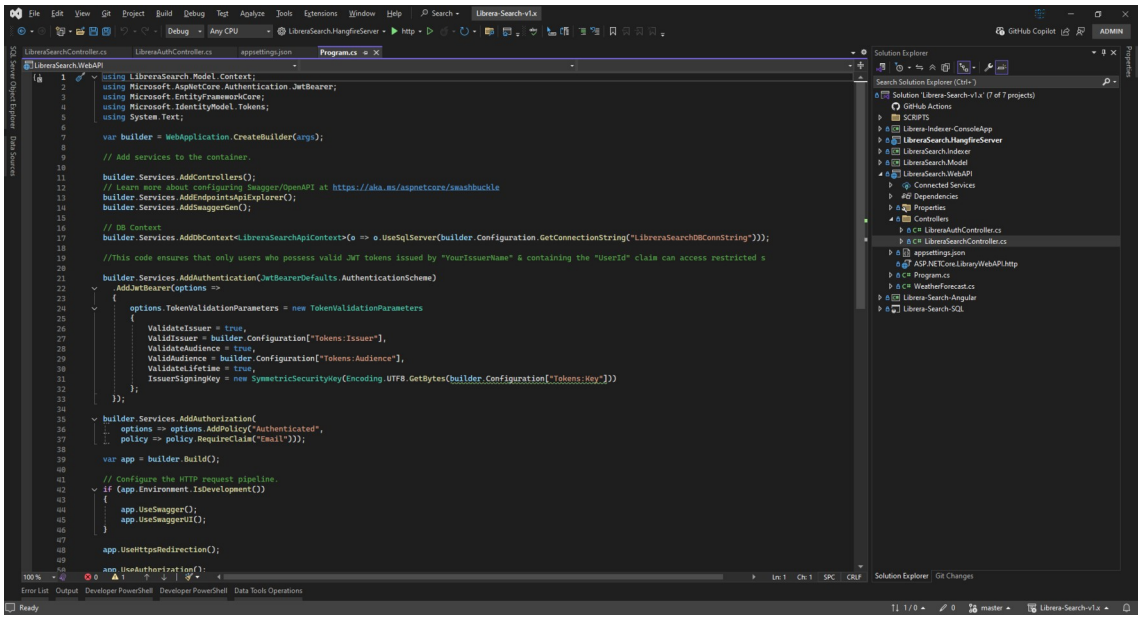
The model layer is managed by EF Core and LinQ is used to launch the queries.

RQ-04 | Web API

The Web API permits to the front-end application to execute CRUD operations my RESTful end-points. Those end-points are securitized by policy-based authorization.

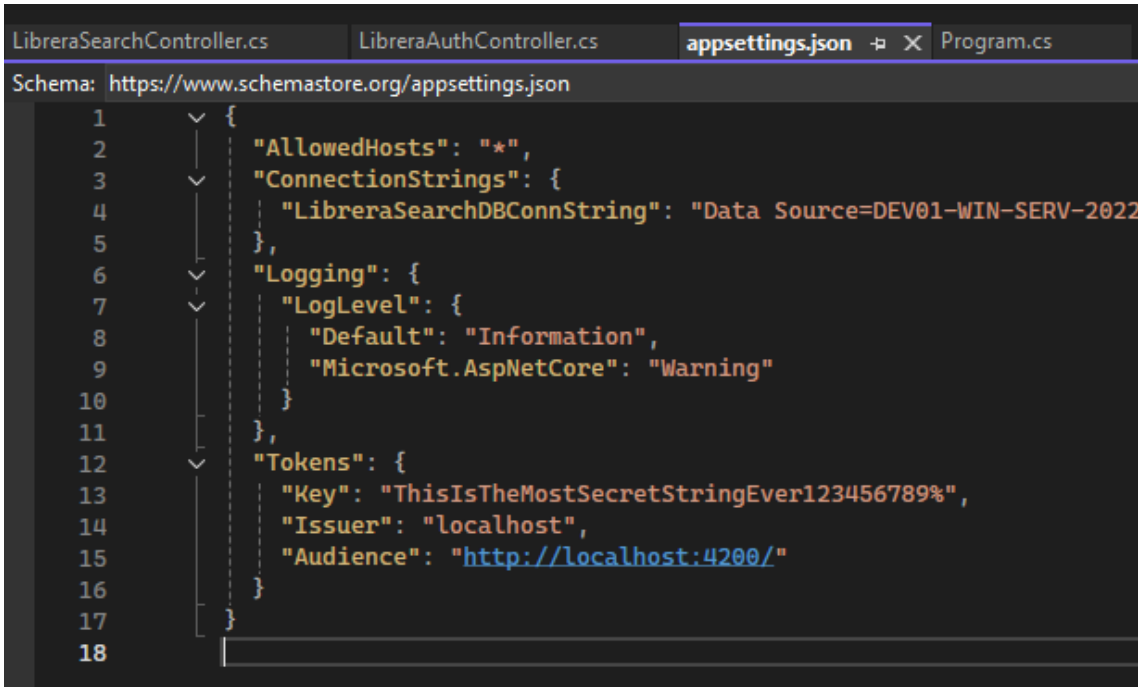
Here some images.

Web API Auth. service setup:



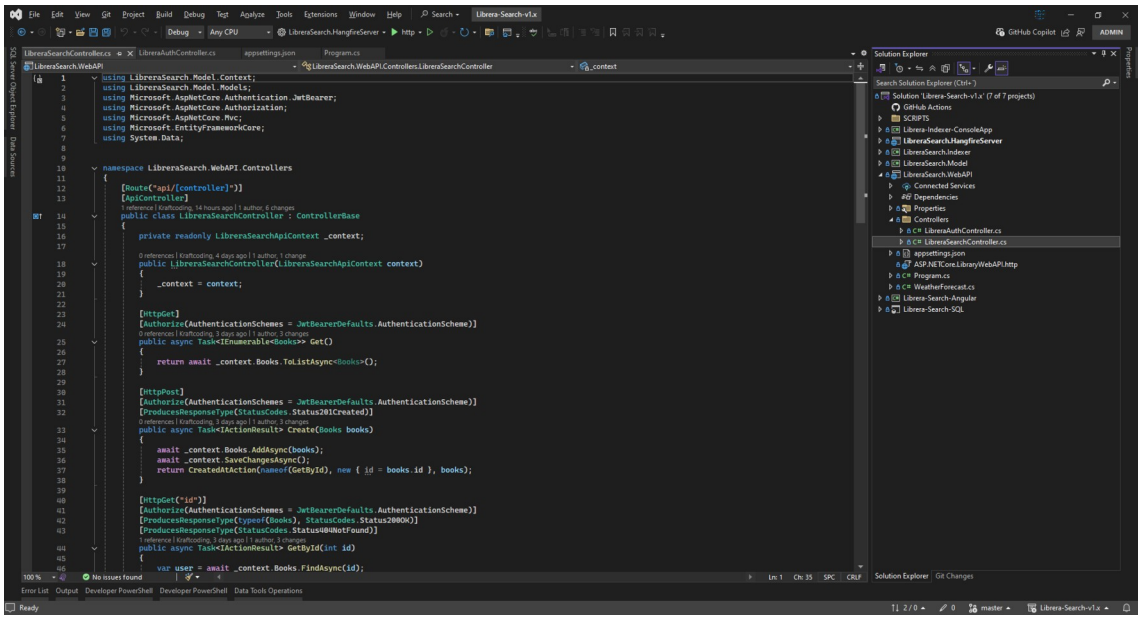
```
using LibreraSearch.Model.Context;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// DB Context
builder.Services.AddDbContext<LibreraSearchApiContext>(o => o.UseSqlServer(builder.Configuration.GetConnectionString("LibreraSearchDBConnString")));

//This code ensures that only users who possess valid JWT tokens issued by "YourIssuerName" & containing the "UserId" claim can access restricted s

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = builder.Configuration["Tokens:Issuer"],
            ValidateAudience = true,
            ValidAudience = builder.Configuration["Tokens:Audience"],
            ValidateLifetime = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Tokens:Key"]))
        };
    });

builder.Services.AddAuthorization(
    options => options.AddPolicy("Authenticated",
    policy => policy.RequireClaim("Email")));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();
```

Web API Tokens configuration:



```
{
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "LibreraSearchDBConnString": "Data Source=DEV01-WIN-SERV-2022
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "Tokens": {
    "Key": "ThisIsTheMostSecretStringEver123456789%",
    "Issuer": "localhost",
    "Audience": "http://localhost:4200/"
  }
}
```

Web API securitized end-points:

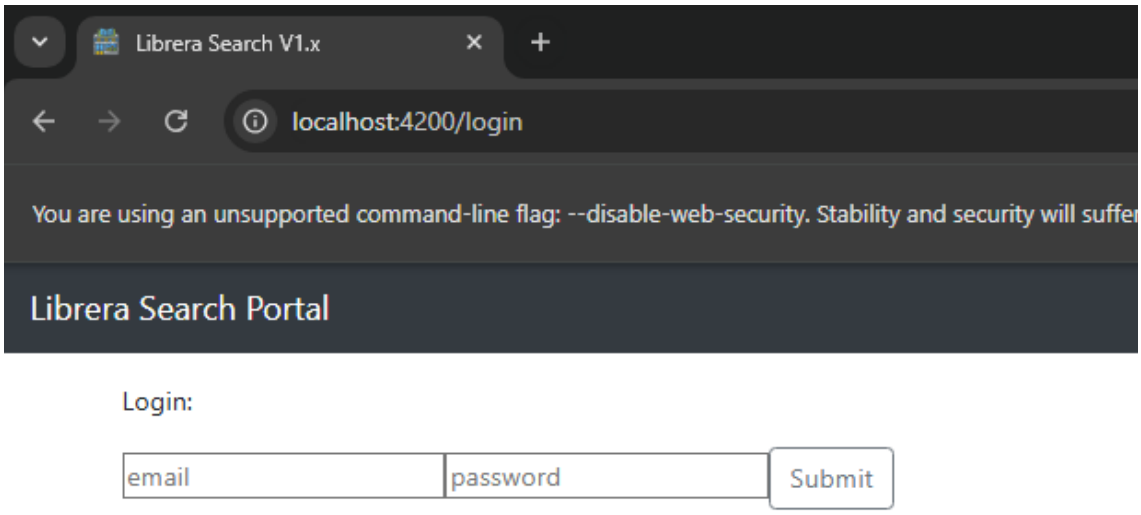

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Librera-Search-v1.x
Debug
Any CPU
LibreraSearch.HangfireServer
http
GitHub Copilot
ADMIN
LibreraSearchController.cs
LibreraAuthController.cs
appsettings.json
Program.cs
LibreraSearch.WebAPI
LibreraSearch.WebAPI.Controllers.LibreraSearchController
_context
using LibreraSearch.Model.Context;
using LibreraSearch.Model.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Data;
namespace LibreraSearch.WebAPI.Controllers
{
[Route("api/[controller]")]
[ApiController]
1 reference | Kraftcoding, 14 hours ago | 1 author, 6 changes
public class LibreraSearchController : ControllerBase
{
private readonly LibreraSearchApiContext _context;
0 references | Kraftcoding, 4 days ago | 1 author, 1 change
public LibreraSearchController(LibreraSearchApiContext context)
{
_context = context;
}
[HttpGet]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
0 references | Kraftcoding, 3 days ago | 1 author, 3 changes
public async Task<IEnumerable<Books>> Get()
{
return await _context.Books.ToListAsync<Books>();
}
[HttpPost]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[ProducesResponseType(StatusCodes.Status201Created)]
0 references | Kraftcoding, 3 days ago | 1 author, 3 changes
public async Task<IActionResult> Create(Books books)
{
await _context.Books.AddAsync(books);
await _context.SaveChangesAsync();
return CreatedAtAction(nameof(GetById), new { id = books.id }, books);
}
[HttpGet("id")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[ProducesResponseType(typeof(Books), StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
1 reference | Kraftcoding, 3 days ago | 1 author, 3 changes
public async Task<IActionResult> GetById(int id)
{
var user = await _context.Books.FindAsync(id);
100 %
No issues found
Ln: 1
Ch: 35
SPC
CRLF
Error List
Output
Developer PowerShell
Developer PowerShell
Data Tools Operations
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Librera-Search-v1.x' (7 of 7 projects)
GitHub Actions
SCRIPTS
Librera-Indexer-ConsoleApp
LibreraSearch.HangfireServer
LibreraSearch.Indexer
LibreraSearch.Model
LibreraSearch.WebAPI
Connected Services
Dependencies
Properties
Controllers
LibreraAuthController.cs
LibreraSearchController.cs
appsettings.json
ASP.NETCore.LibraryWebAPI.http
Program.cs
WeatherForecast.cs
Librera-Search-Angular
Librera-Search-SQL
Solution Explorer
Git Changes
Ready
2 / 0
master
Librera-Search-v1.x

RQ-05 | Front end App

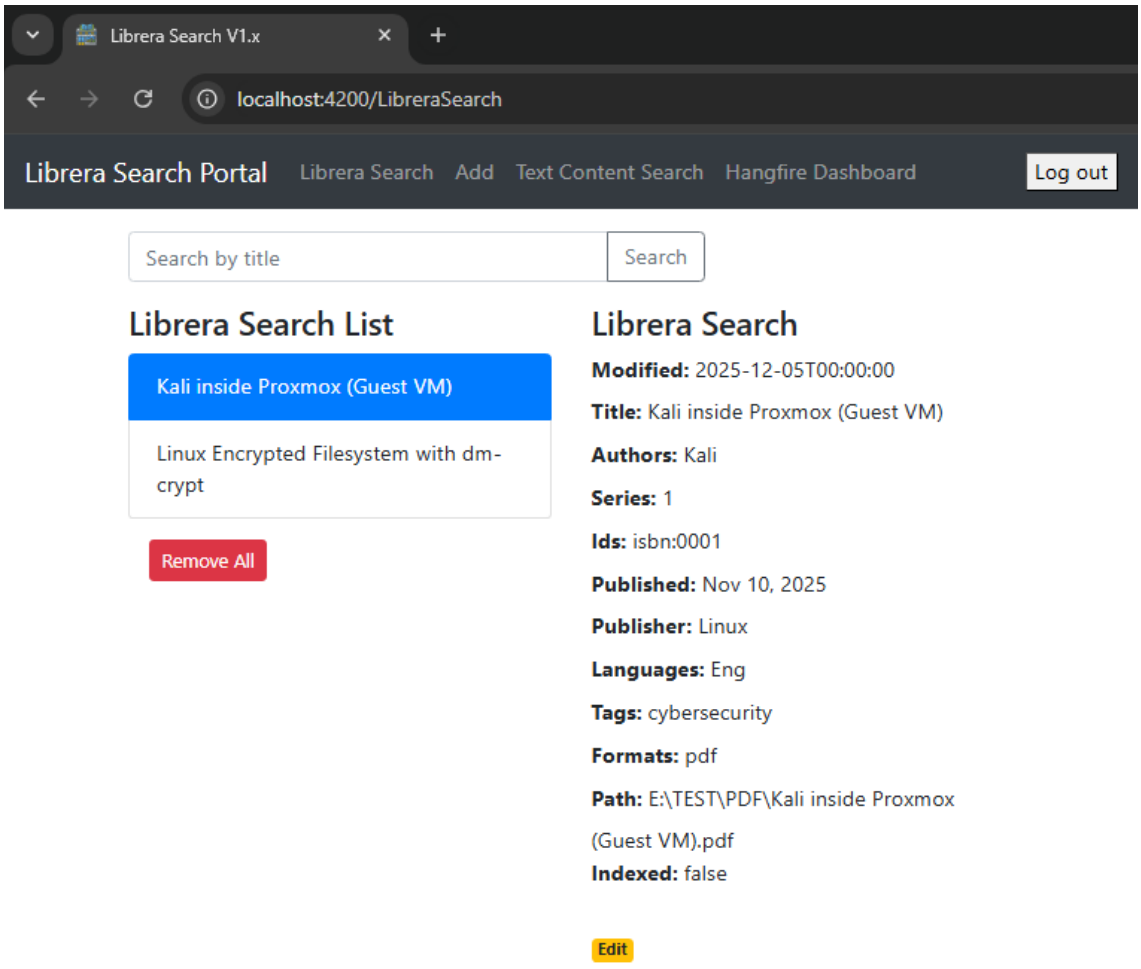
The front end app is built in Angular 16.

Here some images.

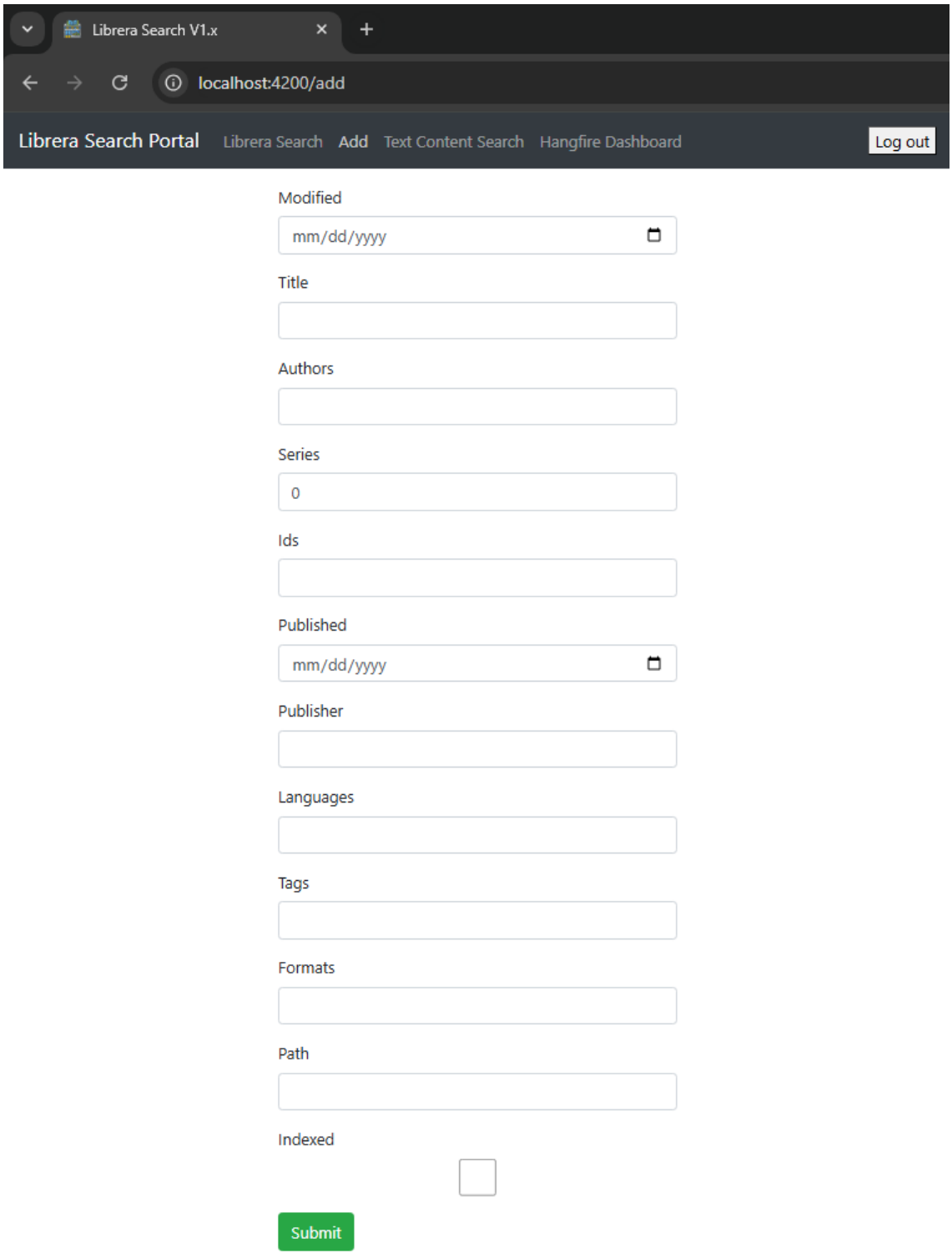
Login form:



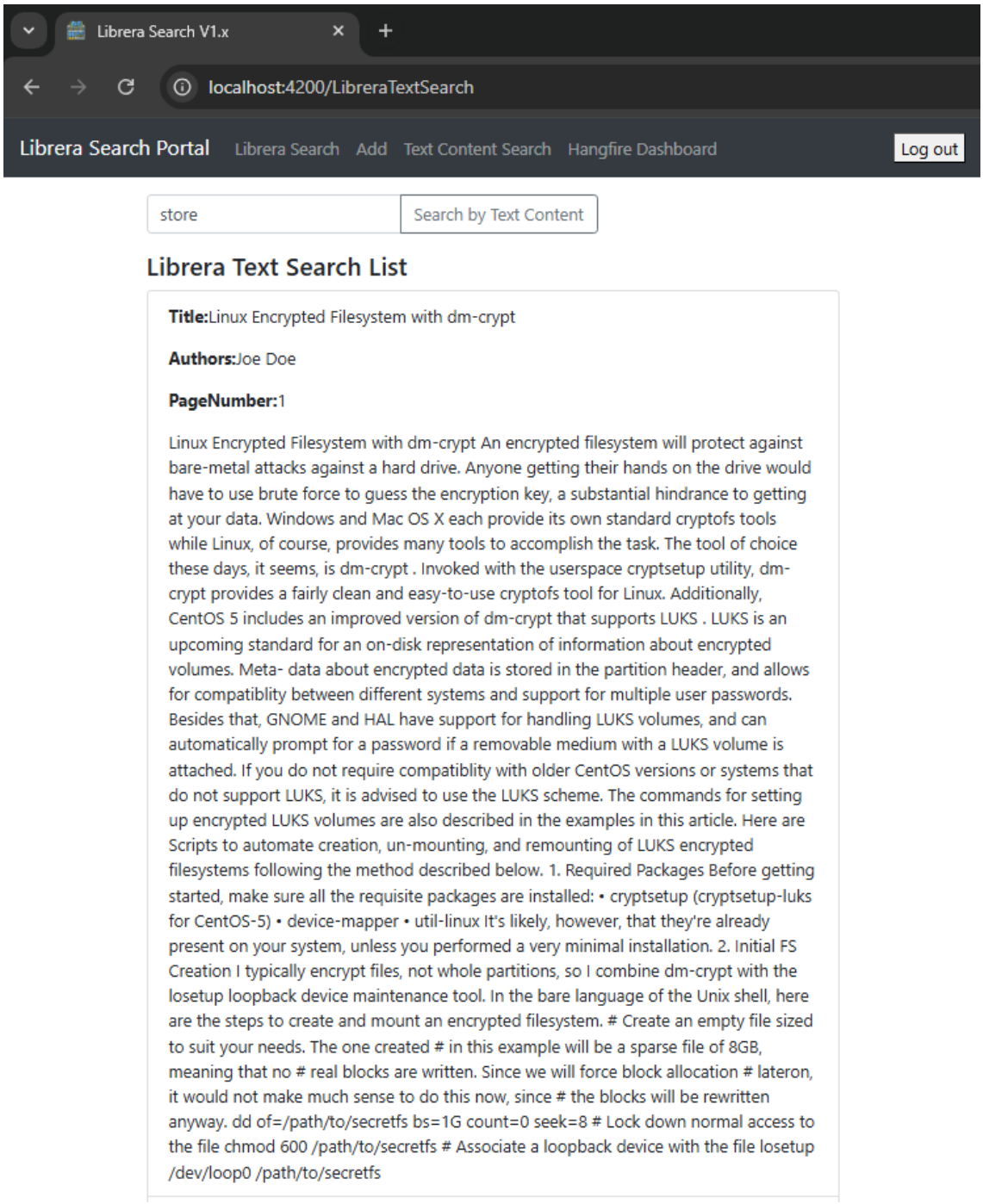
Search by title list and details:


Librera Search V1.x
localhost:4200/LibreraSearch
Librera Search Portal
Librera Search
Add
Text Content Search
Hangfire Dashboard
Log out
Search by title
Search
Librera Search List
Kali inside Proxmox (Guest VM)
Linux Encrypted Filesystem with dm-crypt
Remove All
Librera Search
Modified: 2025-12-05T00:00:00
Title: Kali inside Proxmox (Guest VM)
Authors: Kali
Series: 1
Ids: isbn:0001
Published: Nov 10, 2025
Publisher: Linux
Languages: Eng
Tags: cybersecurity
Formats: pdf
Path: E:\TEST\PDF\Kali inside Proxmox (Guest VM).pdf
Indexed: false
Edit

Add form:

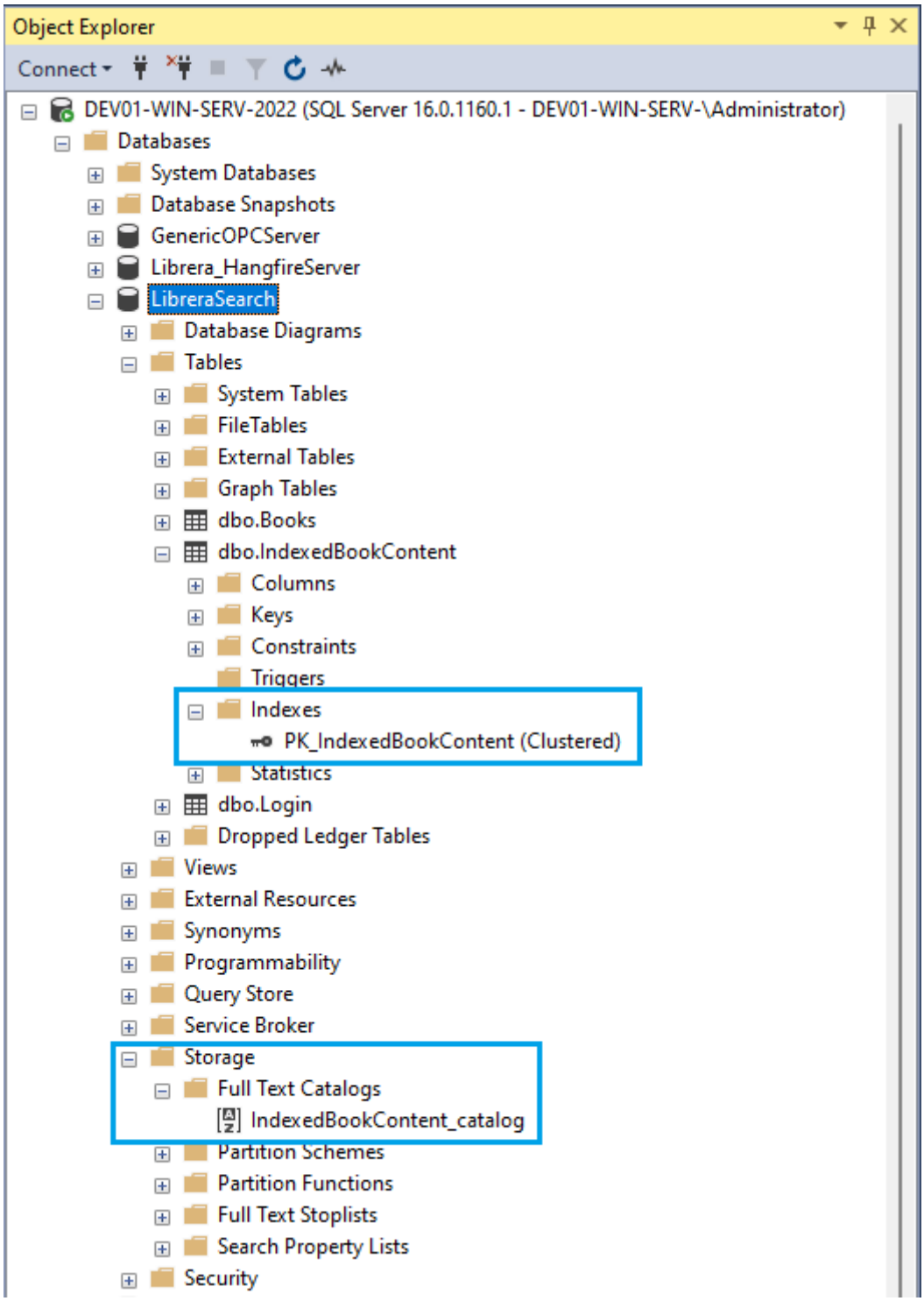


Search by content page details:


Librera Search V1.x
localhost:4200/LibreraTextSearch
Librera Search Portal
Librera Search
Add
Text Content Search
Hangfire Dashboard
Log out
store
Search by Text Content
Librera Text Search List
Title:Linux Encrypted Filesystem with dm-crypt
Authors:Joe Doe
PageNumber:1
Linux Encrypted Filesystem with dm-crypt An encrypted filesystem will protect against bare-metal attacks against a hard drive. Anyone getting their hands on the drive would have to use brute force to guess the encryption key, a substantial hindrance to getting at your data. Windows and Mac OS X each provide its own standard cryptofs tools while Linux, of course, provides many tools to accomplish the task. The tool of choice these days, it seems, is dm-crypt . Invoked with the userspace cryptsetup utility, dm-crypt provides a fairly clean and easy-to-use cryptofs tool for Linux. Additionally, CentOS 5 includes an improved version of dm-crypt that supports LUKS . LUKS is an upcoming standard for an on-disk representation of information about encrypted volumes. Meta- data about encrypted data is stored in the partition header, and allows for compatiblity between different systems and support for multiple user passwords. Besides that, GNOME and HAL have support for handling LUKS volumes, and can automatically prompt for a password if a removable medium with a LUKS volume is attached. If you do not require compatiblity with older CentOS versions or systems that do not support LUKS, it is advised to use the LUKS scheme. The commands for setting up encrypted LUKS volumes are also described in the examples in this article. Here are Scripts to automate creation, un-mounting, and remounting of LUKS encrypted filesystems following the method described below. 1. Required Packages Before getting started, make sure all the requisite packages are installed: • cryptsetup (cryptsetup-luks for CentOS-5) • device-mapper • util-linux It's likely, however, that they're already present on your system, unless you performed a very minimal installation. 2. Initial FS Creation I typically encrypt files, not whole partitions, so I combine dm-crypt with the losetup loopback device maintenance tool. In the bare language of the Unix shell, here are the steps to create and mount an encrypted filesystem. # Create an empty file sized to suit your needs. The one created # in this example will be a sparse file of 8GB, meaning that no # real blocks are written. Since we will force block allocation # lateron, it would not make much sense to do this now, since # the blocks will be rewritten anyway. dd of=/path/to/secretfs bs=1G count=0 seek=8 # Lock down normal access to the file chmod 600 /path/to/secretfs # Associate a loopback device with the file losetup /dev/loop0 /path/to/secretfs

RQ-06 | Database

The database is hosted by M. SQL Server 2022, all the schemas and T-SQL logic is in the project. Full-Text Search feature is needed, as we se in the next image:



Deployment

To do.

Software life cycle management

To do.

Annexes

A console App is added to test the indexer.

A script is added to launch Chrome avoiding CORS