

LIBRERA SEARCH SOLUTION

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

Contents

Requirements.....	2
Overview.....	3
Clean Architecture.....	4
Scope.....	5
RQ-01 Automated Task Service.....	6
RQ-02 Indexer.....	9
RQ-03 Model Layer.....	10
RQ-04 Web API.....	11
RQ-05 Front end App.....	13
RQ-06 Database.....	17
Deployment.....	18
Software life cycle management.....	19
Annexes.....	20

Doc. Version	Date	Author	Description
1.0	2025-12-17	https://github.com/kraftcoding	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 Libreria 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.

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
- Clean architecture

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

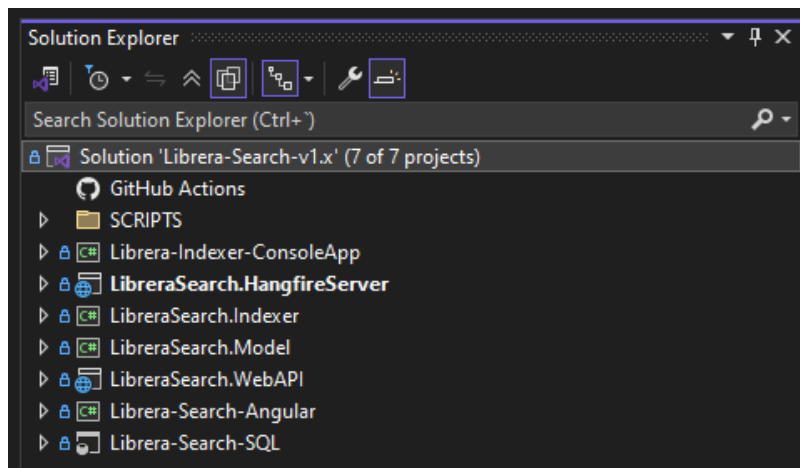
In ASP.NET, policies are used to define requirements that must be met in order to give a user the authorization to access an endpoint.

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.

Clean Architecture

Clean Architecture is an architecture pattern that is based on four layers: presentation, application, domain, and infrastructure. The presentation layer is responsible for handling user interactions, the application layer is responsible for handling the business logic, the domain layer is responsible for defining the business entities and their relationships, and the infrastructure layer is responsible for handling external concerns such as databases and web services.

Here the solution structure (Visual Studio 2022):



The key principles of Clean Architecture include the independence of the layers, the data flow from the outer layers to the inner layers, and the ability to swap out dependencies without affecting the rest of the system. Clean Architecture emphasizes the separation of concerns to make the codebase easier to maintain, test, and extend.

The project structure of Clean Architecture follows a structured layout that enables code reusability and testability. In the project structure, you would typically have a project for the presentation layer, a project for the application layer, a project for the domain layer, a project for the infrastructure layer, and a project for the tests. Each layer would have its own set of interfaces and implementations, with the application layer acting as the bridge between the other layers.

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 screenshot displays the Hangfire Dashboard interface for a job named 'BookIndexerJob'. The browser address bar shows 'localhost:5272/hangfire/jobs/details/15'. A warning message at the top states: 'You are using an unsupported command-line flag: --disable-web-security. Stability and security will suffer.'

The dashboard has a navigation bar with 'Hangfire Dashboard', 'Jobs (0)', 'Retries (0)', 'Recurring Jobs (1)', and 'Servers (1)'. A 'Back to site' link is also present.

On the left, a sidebar lists job states with counts: Enqueued (0/0), Scheduled (0), Processing (0), Succeeded (13), Failed (0), Deleted (2), and Awaiting (0).

The main content area for 'BookIndexerJob' includes:

- A status message: 'The job is finished. It will be removed automatically *in a day*.'
- Job ID: #15
- Code snippet:

```
// Id: #15
using HangfireExample.Jobs;

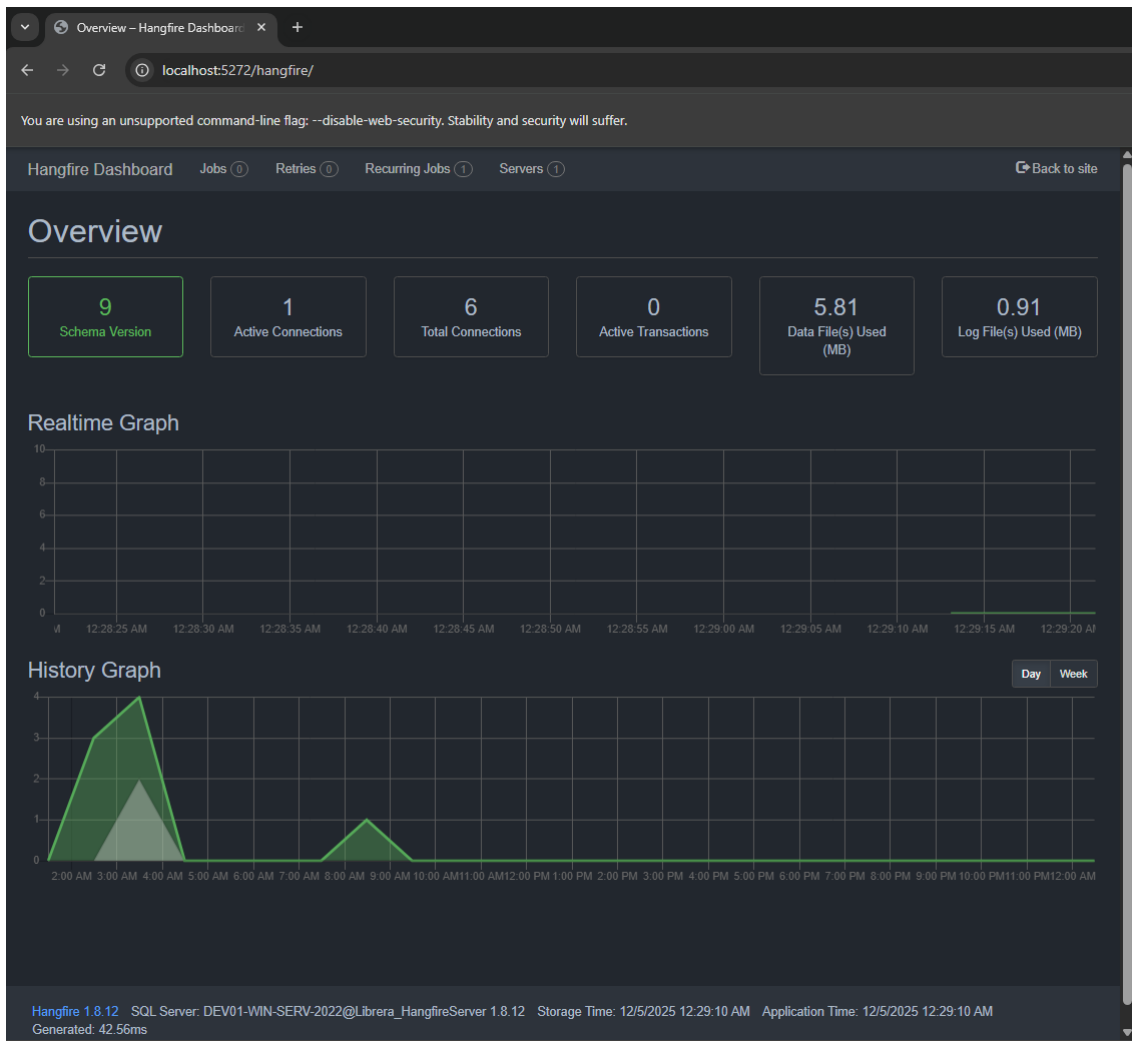
var bookIndexerJob = Activate<BookIndexerJob>();
await bookIndexerJob.RunAsync();
```
- Parameters table:

CurrentCulture	"en-US"
CurrentUICulture	"en-US"
RecurringJobId	"BookIndexerJob"
Time	1764923111
- State section with 'Requeue' and 'Delete' buttons. The current state is 'Succeeded' (6 minutes ago, +39.010s). Below this, a table shows performance metrics:

Latency:	9.034s
Duration:	38.989s
- Processing state (+6.970s) with details:

Server:	DEV01-WIN-SERV-10292
Worker:	0955c009
- Enqueued state (+2.053s) with the note: 'Triggered by recurring job scheduler'.
- Created state (6 minutes ago).

The next image image shows the process graph:



Here the recurring Job:

Recurring Jobs – Hangfire Dashboard

localhost:5272/hangfire/recurring

You are using an unsupported command-line flag: --disable-web-security. Stability and security will suffer.

Hangfire Dashboard Jobs Retries Recurring Jobs Servers Back to site

Recurring Jobs

Trigger now Delete

Items per page: 10 20 50 100 500 1,000 5,000

<input type="checkbox"/> Id	Cron	Time zone	Job	Next execution	Last execution	Created
<input type="checkbox"/> BookIndexerJob	0 * * * *	UTC	BookIndexerJob	in 15 hours	5 minutes ago	2 days ago

Total items: 1

Hangfire 1.8.12 SQL Server: DEV01-WIN-SERV-2022@Libra_hangfireServer 1.8.12 Storage Time: 12/5/2025 12:30:13 AM Application Time: 12/5/2025 12:30:13 AM
Generated: 203.49ms

Here the Hangfire service definition for the Job:

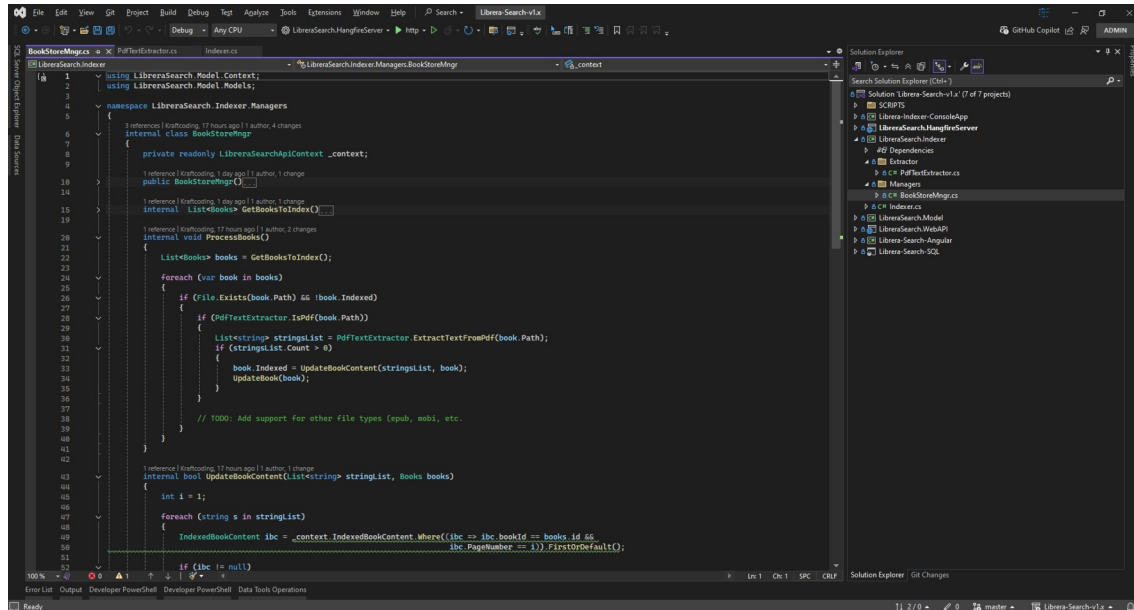
```

1  var hangfireConnectionString = builder.Configuration.GetConnectionString("DefaultConnection");
2
3  builder.Services.AddHangfire(configuration => configuration
4      .SetDataCompatibilityLevel(CompatibilityLevel.Version_170)
5      .UseSimpleAssemblyNameTypeSerializer()
6      .UseRecommendedSerializerSettings()
7      .UseSqlServerStorage(
8          hangfireConnectionString,
9          new SqlServerStorageOptions
10             {
11                 CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),
12                 SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),
13                 QueuePollInterval = TimeSpan.Zero,
14                 UseRecommendedIsolationLevel = true,
15                 DisableAutomaticIndexing = true,
16             });
17
18  builder.Services.AddHangfireServer();
19
20  builder.Services.AddControllers();
21  builder.Services.AddEndpointsApiExplorer();
22  builder.Services.AddSwaggerGen();
23
24  //builder.Services.AddTransient<TestRecurringJob, TestRecurringJob>();
25  //builder.Services.AddScoped<TestJob, TestJob>();
26
27  builder.Services.AddTransient<BookIndexerJob, BookIndexerJob>();
28
29  var app = builder.Build();
30
31  // Configure the HTTP request pipeline.
32  if (app.Environment.IsDevelopment())
33  {
34      app.UseSwagger();
35      app.UseSwaggerUI();
36  }
37
38  app.UseHttpsRedirection();
39  app.MapControllers();
40
41  // Change 'Back to site' link URL
42  var options = new DashboardOptions { AppPath = "http://localhost:5280/" };
43  // Make 'Back to site' link working for subfolder applications
44  //var options = new DashboardOptions { AppPath = VirtualPathUtility.ToAbsolute("~/") };
45
46  app.UseHangfireDashboard("/hangfire", options);
47
48  RecurringJob.AddOrUpdate<BookIndexerJob>(), () => app.Services.GetRequiredService<BookIndexerJob>().RunAsync(), Cron.Daily);
49
50  app.Run();

```

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 extracts content from PDFs and stores it in DB.

RQ-03 | Model Layer

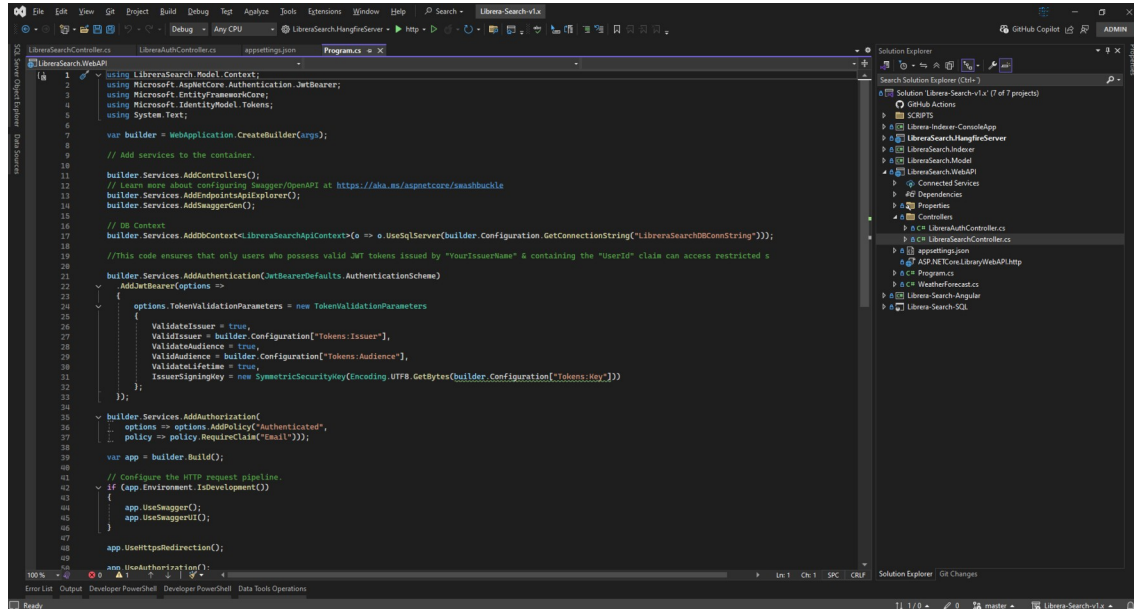
The model layer is managed by EF Core and LinQ is used to launch the queries.

RQ-04 | Web API

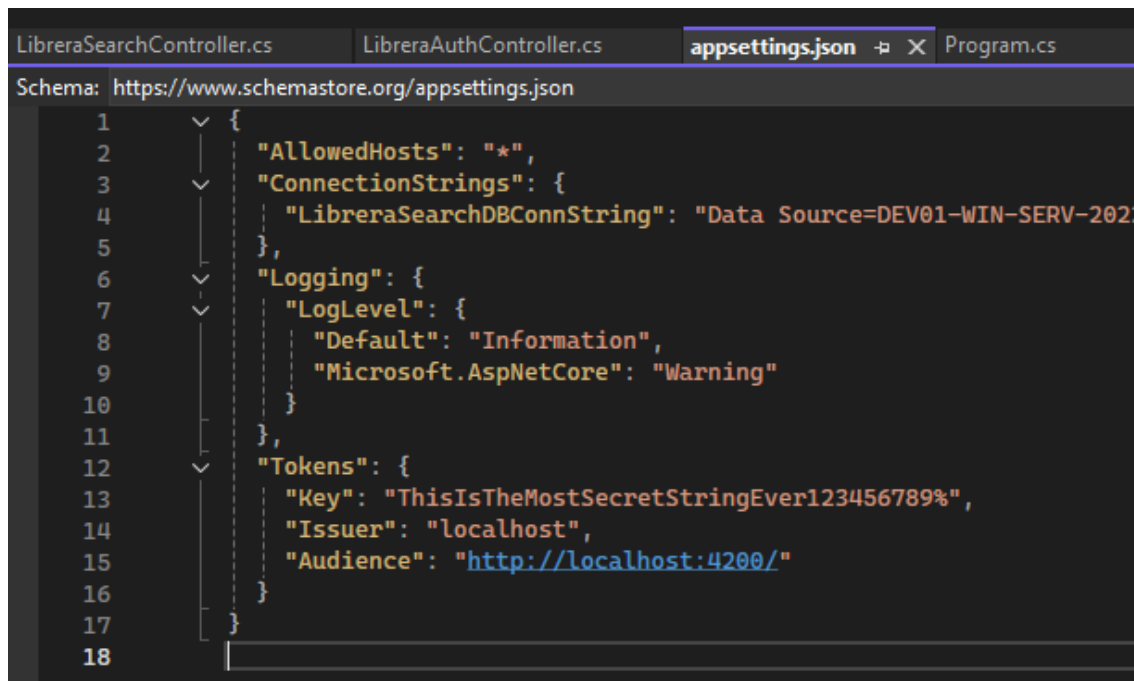
The Web API permits to the front-end application to execute CRUD operations my RESTful end-points. Those end-points are securitized by policy-based authorization.

Here some images.

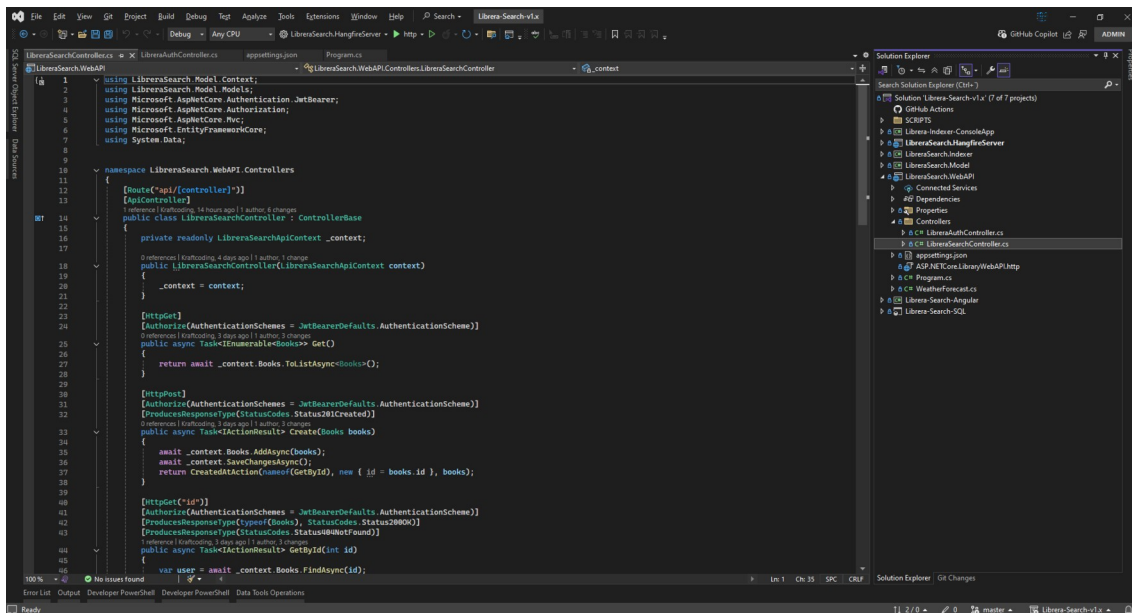
Web API Auth. service setup:



Web API Tokens configuration:



Web API securitized end-points:

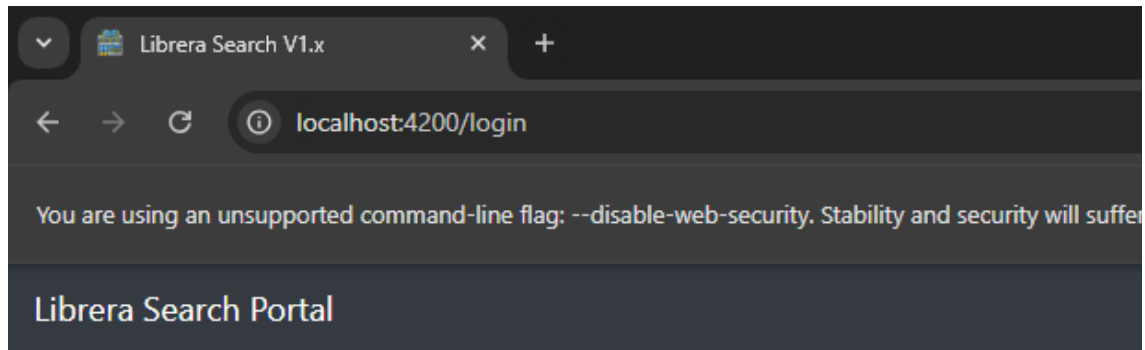


RQ-05 | Front end App

The front end app is built in Angular 16.

Here some images.

Login form:



Login:

email	password	Submit
-------	----------	--------

Search by title list and details:

Libreria Search V1.x

localhost:4200/LibreraSearch

Libreria Search Portal

Libreria SearchAddText Content SearchHangfire Dashboard

Log out

Search by title

Search

Libreria Search List

Kali inside Proxmox (Guest VM)

Linux Encrypted Filesystem with dm-crypt

Remove All

Libreria 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:

Modified

mm/dd/yyyy

Title

Authors

Series

0

Ids

Published

mm/dd/yyyy

Publisher

Languages

Tags

Formats

Path

Indexed

☐

Submit

Search by content page details:

store

Search by Text Content

Librera Text Search List

Title:Linux Encrypted Filesystem with dm-crypt

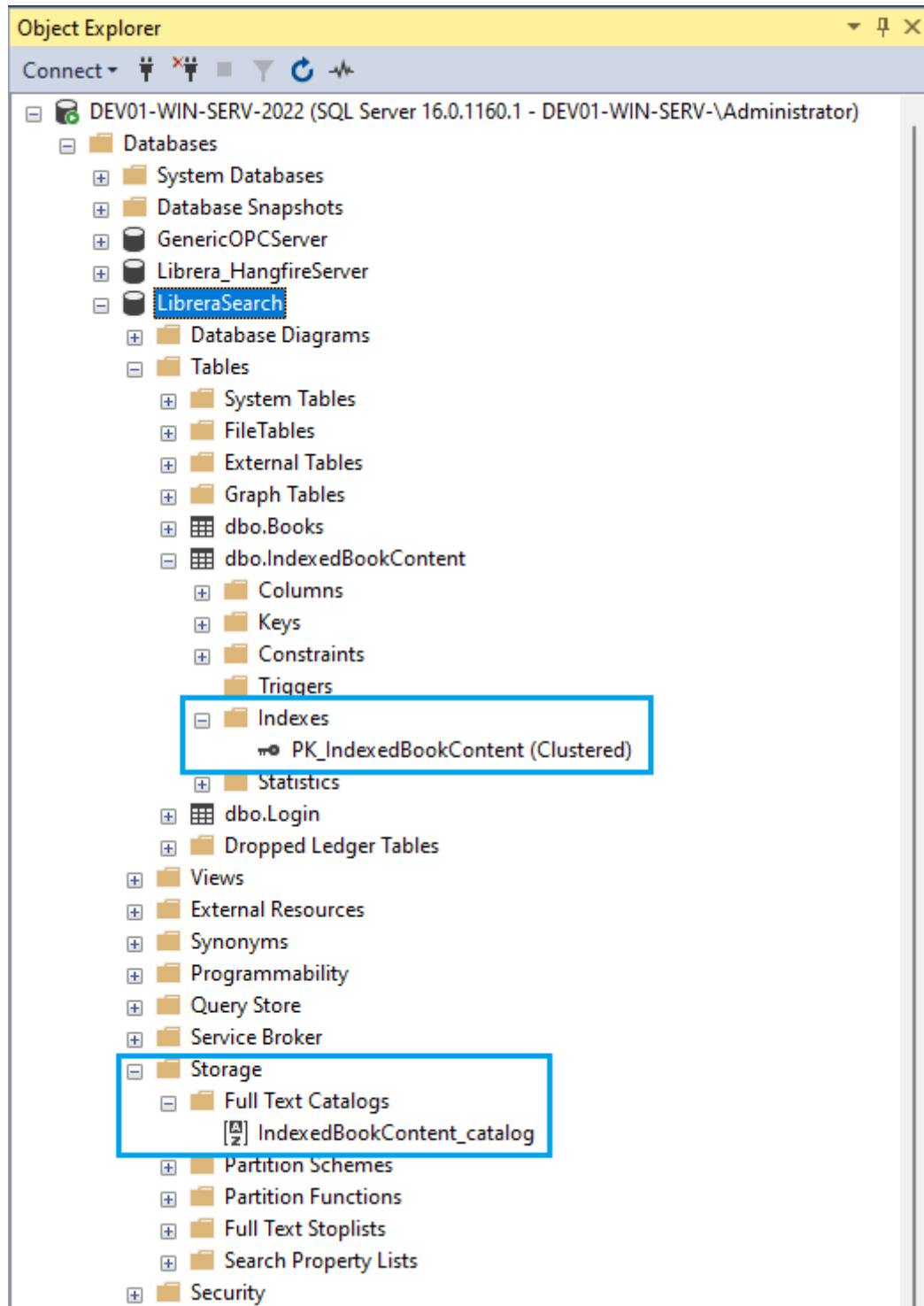
Authors:Joe Doe

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 compatibility 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 compatibility 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