Integration of SQL Server Reporting Services into modern application infrastructure

Volodymyr Kuharsky[1,†] and Dmytro Mykhalyk[1,†]

[1] *Ternopil Ivan Puluj National Technical University, 56, Ruska Street, Ternopil 46001, Ukraine*

Abstract

The proposed research delves into the integration of the SSRS report viewer with Angular applications. While a singular component was selected for its independence from proprietary services, significant architectural limitations were present.

The key objectives for addressing these limitations have been identified and described, among them: enhancing security by imposing access restrictions, giving access to authenticated and authorized users only, ensuring proper and transparent NTLM authentication handling, and enhancing the security of the report catalog.

The further advantages offered by SSRS integration through a standalone native service have been revealed.

The article highlights the complexities and opportunities in integrating SQL Server Reporting Services with Angular applications, emphasizing the importance of addressing security and architectural challenges while broadening the scope of integration for enhanced functionality. The concept of integration service in modern .NET runtime. has been implemented taking into account best practices for architecture and application design.

Keywords

SSRS, NTLM, Authentication, Integration, Visual Studio, Database, UI, API, Server, URL

1.Introduction

These days, data lies at the very heart of every information system service. Businesses use data to develop robust growth strategies, to prevent or severely mitigate possible future crises, and to be aware of what is happening to the business itself at a specific time. Integrating SSRS with high-performance supercomputing technologies facilitates the seamless presentation of massive computational outputs, transforming raw data from simulations or large-scale analytics into accessible and actionable reports for advanced research and business intelligence [1,2]. Reporting services and reports are valuable tools for both software engineers and business analysts to utilize data in decision-making.

There are several modern general-purpose reporting services systems across the market including:

ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23-25, 2024, Ternopil, Ukraine, Opole, Poland
* Corresponding author.
† These authors contributed equally.
volodimir.kukharskiy@windowslive.com (V. Kuharsky); d.mykhalyk@gmail.com (D. Mykhalyk) 0009-0009-6937-9526 (V. Kuharsky); 0000-0001-9032-695X (D. Mykhalyk)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3896/short9.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

- SQL Server Reporting Services - commercial solution by Microsoft • Power BI Report Server – commercial cloud-based solution by Microsoft.
- Jaspersoft – highly scalable commercial open-source solution.
- BIRT – free open-source solution by Eclipse.
- Zoho Reports – cloud-based reporting and business intelligence solution.
- Google Data Studio – free solution by Google

And many others offer a variety of specialized reporting solutions for industries such as healthcare, transportation, energy, and construction. SSRS was a natural choice for a couple of reasons. First, it is a part of the Microsoft ecosystem that is at the core of the University's current programming stack. Additionally, it is a very mature solution for report design, publication, and rendering.

2.Brief overview of SQL Server Reporting Services architecture

SQL Server Reporting Services is a mature and very stable reporting platform that allows us to deploy, publish, schedule, and manage reports. SSRS can be used to manage and publish enrichment and interactive reports, key performance indicators, datasets, and mobile reports [3-6].

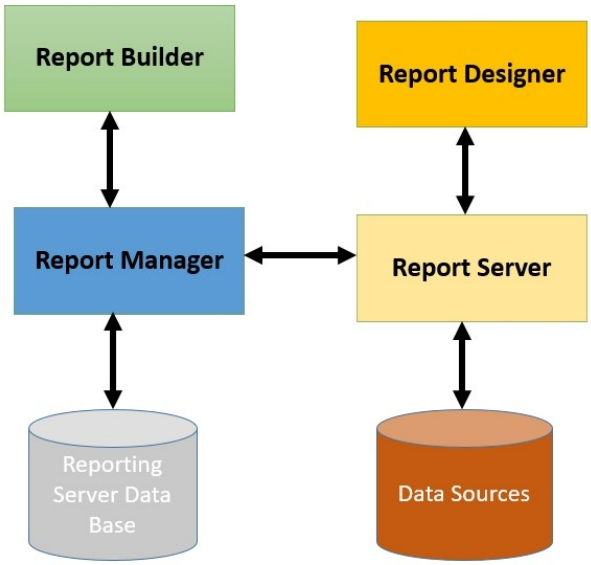

Figure 1: SSRS components diagram

As we can see in Figure 1, SSRS has a broad architecture:

- Report Builder and Report Designer are used for report development and deployment but differ in features. The former has features for simple reports and is

suitable mainly for business users and power users. The latter is more feature-rich, has deep integration with Visual Studio, and is ideal for experienced report developers.

- Report Manager is a web-based tool that allows users to view, manage, and distribute SSRS reports.
- Report database stores all catalog items, their properties, and necessary security information for access management, and configuration.
- TempDB, the Report Database, stores session and execution data, information related to cached reports, and work tables generated by the Report Server.
- Data sources – abstracted data providers for reports.

Report Server is based on a three-tier application architecture and these tiers are Middle-tier, Data-tier, and Presentation-tier, and runs inside of Windows service context

- Middle-tier – consists of a set of various extensions like security, data-processing, rendering, delivery extensions, etc.
- Data tier – a place where interactions with its database and various data sources exist.
- Presentation tier – a place where web portal and application programming interfaces exist.

3. Problem definition

To integrate the SSRS report viewer with the Angular application initial research was done. At the end of the research, the single component was chosen because it does not use any proprietary services as a bridge between itself and the Report Server [7]. Unfortunately, that component has a big architectural limitation. Because it uses a simple SSRS request URL to manipulate or render a deployed report, there is no way to securely pass authentication credentials.

Alas, this is not secure and means the need to share the Report Server user for the task with an arbitrary authenticated user within the informational system who needs to view a specific report, or the creation and management of a separate account per authenticated user on the Report Server side, because he or she may need to have access to view specific reports somewhere in the future.

Report Server provides UI for report rendering but its usage inside of client-side application is problematic. Back in the days of server-side applications, it was easy to implement such integration because all security and configuration management was done on a server [4]. Today applications most of the time run on the client-side effectively rendering the previously developed server-side components and plug-ins for Report Server unusable.

When considering the user's viewpoint, a streamlined user interface is crucial for ensuring a positive experience. However, if an application requires a highly tailored user interface, it may present some challenges. As mentioned earlier, the rendering component is limited to a predefined set of HTML and CSS scripts, which can be a potential obstacle.

In rarer situations, network bandwidth can become a problem. Rendering reports using the ngx-ssrs-reportviewer component takes sometimes significant amount of time because of report size and complexity. Large multi-page reports tend to load slowly because of how

the component fetches the report data from the SSRS. The report is rendered in the browser, so the rendering performance can depend on the user's device (CPU, memory, etc.)

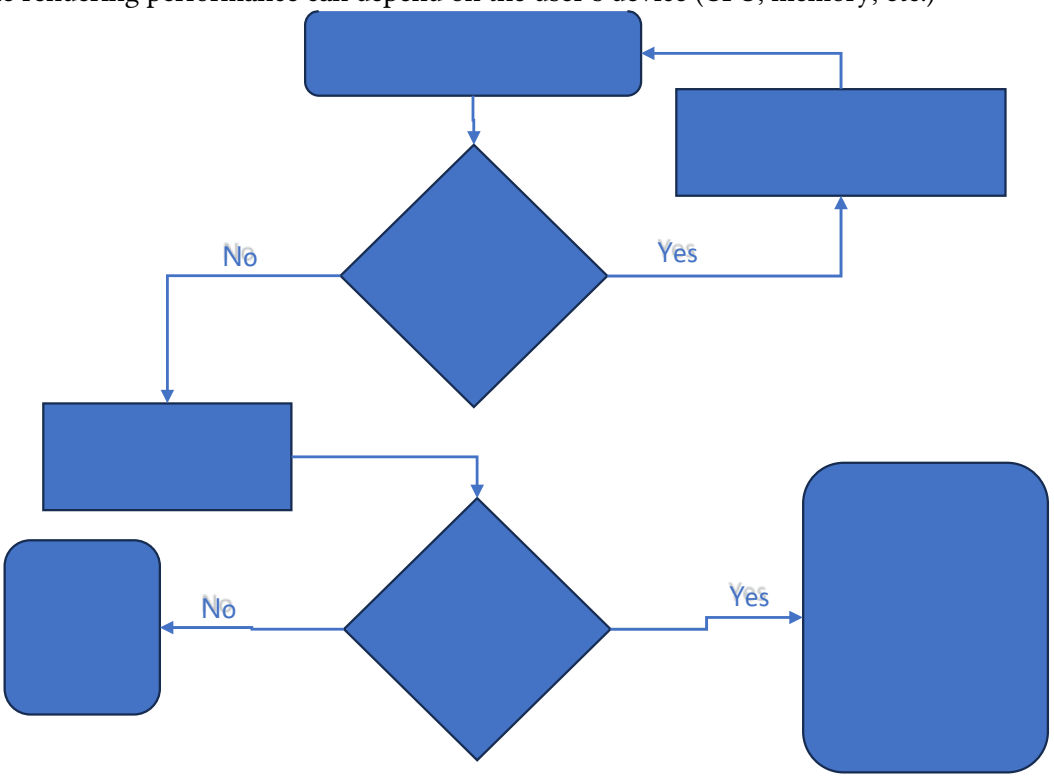

Figure 2: Acquisition of report through ngx-ssrs-reportviewer component

According to the reference in Figure 2, obtaining a report involves a lengthy and intricate process. The user must first authenticate on the Report Server before any user interface is displayed. If the user provides incorrect credentials, no user interface will be generated. Unfortunately, this cannot be resolved as the SSRS report viewer relies on basic URL parameter passing. To integrate SSRS with a single-page application, the decision was made to develop a separate service that will be running on a server and handle NTLM authentication transparently to the client. Fortunately, Report Server has several programming extension points along with a rich set of application interfaces.

4.Integration

The key aspects that the integration service had to resolve:

1. Report Server access restrictions to improve security.
2. Reports access restriction to everyone but authenticated and authorized to view users only.
3. NTLM authentication proper and transparent handling.
4. Report catalog security improvements

Report Server access restriction

Any component of the larger system that has no need to be visible should be hidden within. It is a good practice to hide a Report Server from the outside world behind a firewall and expose a connection port only to a machine where the integration service will be run.

Implementation of authorization checking

The integration service should use a common SSO(Single Sign-On) ticket to prevent unauthorized users from accessing the report rendering and other report functionality without checking who they are. This also improves the security and stability of the system because non-authenticated malicious actors will be unable to sniff a direct Report Server URL and generate malicious requests to the server itself using the same technique ngx-ssrsreportviewer is using to render a report on a client side.

NTLM authentication proper and transparent handling

The integration service should handle authentication on the Report Server transparently to the client itself. Fortunately, modern runtimes have already been developed with proper NTLM authentication mechanisms, and the authentication application will be relatively trivial.

Report Server catalog security improvements

The integration service should use a set of lowest privileged users for the tasks they will perform. It is also a good security practice to not use an administrative account with full access in the context of any service. Such bad decisions allow malicious actors to steal sensitive information from a service or server where the service is running and further compromise or do damage to the report catalog or the report server itself. It is also clear that after some exploits against the service have been developed, users created for concrete low-access tasks will be unable to do any structural or significant damage to the system.

The integration also shortens report generation for any particular user who has a right to view the report. This is shown in Figure 3.

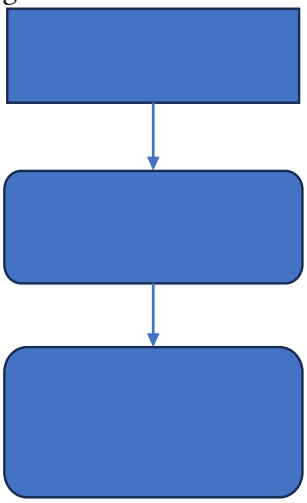

Figure 3: Acquisition of report through integration service

This is an exhaustive but not complete list of problems that the integration service should solve.

Finally, code, design practices, and software architecture must follow current standards in the software development industry. [9, 11]

5.Discussion of obtained results

Based on design requirements the integration service was developed. It is a RESTful service built on a modern (at the time of writing) .NET 7 runtime with Open Api 3 specification in mind. [12] Figure 4 shows the look of the initial stage of documented service endpoints. */reports* endpoints fully cover the necessity of report rendering, while */supported-formats* endpoint presents the information about supported by SSRS rendering extensions (e.g., PDF, MS Word, MS Excel, etc.).

Conceptually all available endpoints are served by making calls to the old SOAP Web services on the part of Report Server Presentation Layer engine. URLs for those Web services are abstracted in the configuration layer of the integration service. [8, 10]

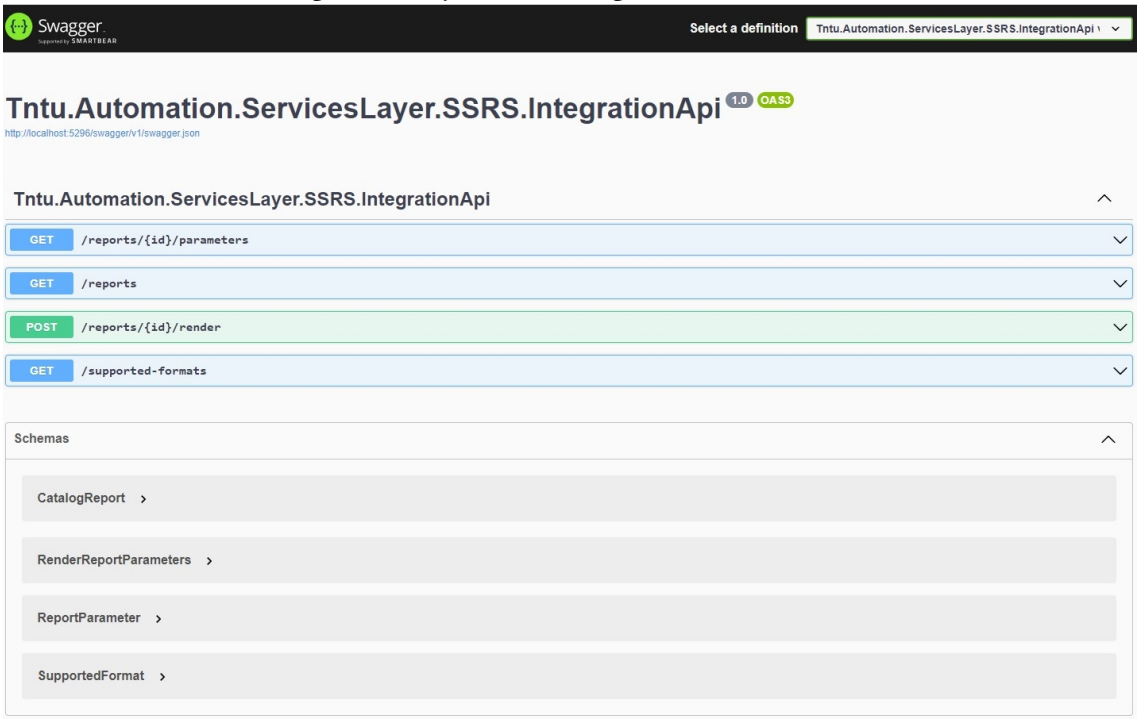

Figure 4: Open Api Endpoints Documentation

The same is true for authentication information for the internal user with granted rights to render specific catalog reports accessible from the integration service by the service client.

Here is an example listing of what the initial configuration of the service can look like.

```
serviceProvider =>
{
  var ssrsSettings = serviceProvider.GetRequiredService<IOptions<SSRSOptions>>()
                                    Value;
```

```
  var bindings = new BasicHttpBindings();
  bindings.MaxBufferSize = ssrsSettings.Client.MaxBufferSize;
  bindings.MaxReceivedMessageSize = ssrsSettings.Client.MaxReceivedMessageSize;

    bindings.Security.Mode = BasicHttpSecurityMode.TransportCredentialOnly;
    bindings.Security.Transport.ClientCredentialType                                    =
HttpClientCredentialType.Ntlm;
    var endpointAddress = new EndpointAddress(ssrsSettings.ExecutionServiceUrl);
    var client = new ExecutionServiceSoapClient(bindings, endpointAddress);

var serviceCredentials =
    new NetworkCredential(ssrsSettings.Authentication.UserName,
                          ssrsSettings.Authentication.Password);
    client.ClientCredentials.Windows.ClientCredentials = serviceCredentials;
    client.ClientCredentials.Windows.AllowedImpersonationLevel =
        TokenImpersonationLevel.Impersonation;
     return client;
}
```

Listing 1: Lambda function of report execution service client construction with NTLM authentication.

The NTLM authentication ticket is passed at the Web service client construction stage and after the construction, no further steps are needed. [10] The code needed for the authentication process looks like the one on Listing 1.

The same approach was used for the construction of another Report Server catalog service client.

6.Conclusion

Integrating SQL Reporting Services through a standalone native service has several advantages already discussed in this article. It also broadens the scope of application of such integration. The only scope of ngx-ssrs-reportviewer is report rendering, while the discussed service can be used not only for rendering but for other Report Server tasks that can be done programmatically.

The potential disadvantage of using such an approach is UI availability. SSRS has its UI for report rendering available for users through iframe when using ngx-ssrs-reportviewer but with a standalone service approach, developers need to build an acceptable UI by themselves for paginated reports rendering if their tasks involve preview mode.

Further research could be devoted to widening the functionality of the integration service itself. Also, research in the security field to prevent some sort of crafted exploits against service may be needed in the future.

7.References

[1] I.V. Boyko , M.R. Petryk. Interaction of electrons with acoustic phonons in AlN/GaN resonant tunnelling nanostructures at different temperatures. Condensed Matter Physics, 2020, vol. 23, No. 3, 33708 DOI:10.5488/CMP.23.33708

[2] Petryk, M.R., Boyko, I.V., Khimich, O.M. et al. High-Performance Supercomputer Technologies of Simulation and Identification of Nanoporous Systems with Feedback

for n-Component Competitive Adsorption. Cybern Syst Anal 57, 316–328 (2021). https://doi.org/10.1007/s10559-021-00357-7

[3] B. K. McDonald, S. McGehee, R. Landrum, Pro SQL Server 2012 Reporting Services (Expert's Voice in SQL Server), 3rd ed., Apress, Berkeley, CA, 2012

[4] B. Larson, Microsoft SQL Server 2016 Reporting Services, Fifth Edition, 5th ed., McGraw Hill, New York, NY, 2016

[5] S. Misner, Microsoft SQL Server 2012 Reporting Services, 1st ed., Microsoft Press, Redmond, WA, 2013

[6] A. Machanic, A. Aitchison, Expert SQL Server 2008 Development, 1st ed., Apress, Berkeley, CA, 2009

[7] V. De Sanctis, ASP.NET Core 5 and Angular: Hands-On Full Stack Development with .NET 5 and Angular 11, 4th ed., Packt, 2021

[8] J. Kurtz, B. Wortman, ASP.NET Web API 2: Building a REST Service from Start to Finish, 2nd ed., Apress, Berkeley, CA, 2014

[9] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, 1st ed., Pearson, London, 2008

[10] A. Troelsen, P. Japikse, Pro C# 10 with .NET 6, 11st ed., Apress, Berkeley, CA, 2022

[11] D. Esposito, A., Saltarello, Microsoft .NET - Architecting Applications for the Enterprise, 2nd ed., Microsoft Press, Redmond, WA, 2014

[12] S. Smith, Architecting Modern Web Applications with ASP.NET Core and Azure, 7th ed., Microsoft Developer Division, .NET, and Visual Studio product teams, Redmond, WA, 2023