User Impersonation Project

Jonathan Ergas

***Abstract*—Volpara Health is a company focused on finding ways to detect breast cancer with AI in the early stages before it becomes lethal. They have a customer facing application that helps clinicians better assess if their clients have a possibility of breast cancer. Currently, Volpara has no system in place that allows a support team member to log in as one of their clients to help diagnose issues/problems with the dashboard without the client needing to provide their login information. This causes a lot of issues with privacy and security as anyone with the login information of users can access confidential files and other internal systems. This led to the need for a custom solution that allows the impersonation of a user to access the dashboard and no other service. This project provides the desired custom solution that would allow the support staff to check the customers' dashboard quickly and efficiently for their issues causing less downtime and a better customer support experience. Tools used for developing the project include TypeScript, Angular for the front-end and C#(.Net) for the backend. The performance and security of this project are key, this requires website and backend load tests and performance metrics, as well as security tests to make sure all sensitive methods require authorization to be executed and that the website cannot be bypassed in any way to gain sensitive information. On top of these tests are user tests to make sure the solution is easy and straightforward to use without the need for extra training that would cause more overhead for Volpara Health.**

I. Introduction

VOLPARA Health is a company focused on using AI to detect breast cancer in the early stages. This involves training an AI to detect cancerous cells in an image and say if that image has cancer cells in it or not. The problem they are facing is with their dashboard because they want customer support members to be able to log in as their clients to diagnose issues and problems faster. Their current system does this functionality, but the underlying API calls are still done by the support team members' account, not the person they are impersonating so the data and the view of the dashboard would be different due to permissions being different between both parties. This project aims to replace their current system and do the impersonation at the API level so the impersonated user would be identical to the real user allowing the support team member to see exactly what their client sees if they were them without the need of asking them for their login information.

Volpara needs this solution because without it the dashboard view of analytics could be desynced from the actual state and could have some inaccuracies. Resulting in a misrepresentation of the data and can lead to situations where it's a case of "it works on my machine". This is mainly caused by the dashboard doing a pseudo impersonation already that just cuts down the view until you see what that user should see but this causes problems because the underlying dashboard token used has the privileges of the logged-in user, not the user they are impersonating resulting in mismatched permission that can lead to different views of the dashboard that is not what we want.

The proposed solution is a TypeScript Angular/Ionic front end with a .NET C# API backend this is because Volpara Health uses .NET in their backend so it would be easy for them to integrate the final solution with their existing server code. The reason TypeScript Angular/Ionic for the front end was chosen is that I have lots of experience using these frameworks and can efficiently mock up a functional front end for them to use. Also, later in the project if there is time, I can rewrite the front end using Blazor to future-proof the app since Volpara is going to use exclusively Blazor as their front-end framework.

By having the solution written in TypeScript Angular/Ionic it allows for the web application to be used on all platforms (PC, Android, IOS) and could be converted into a native mobile application in the future if Volpara wants to with no effort at all.

The measurable performance and functional requirements are that the solution works for its intended purpose and is easy to use. Also, it must be quick and efficient with the whole process taking less than 3 seconds to complete to give a good user experience and to make the solution feel snappy and responsive. This is to make the user experience as painless as possible and contributes to the quick and easy aspects of the solution.

Sustainability goals of the solution is that it should be performance focused and well optimized to reduce wasted compute and lower the overall power draw of the solution to the bare minimum when not in use. Also the solution must be maintainable and upgradable so ensure the long life of the solution.

II. Background Research

The aspects covered in the background research are different ways to do user impersonation and what are their pros and cons. This is so we can determine what method would be best to fit our use case. Also, it will cover some background on the types of tokens that would be used in the solution and their roles. Another part is what tools and languages would be considered for the solution and why.

A. Impersonation Choices

Azure User impersonation

There are some existing solutions for example using azures built-in solution to do the impersonation[1] but with this solution, Volpara's Analytics dashboard wouldn't be able to distinguish an impersonated user from the real user which could create a major security flaw because then the logs

This project was supervised by Jyoti Sahni (primary), Alvin Valera, and Ben MacDonell (Volpara Health).

wouldn't know who did the impersonating. This is because Azure's user impersonation works at the Azure Active Directory Authentication level the closest level to the user, meaning it's like the user just logged in with their username and password. This level is not required to do impersonation on Volpara systems because this authentication token goes replaced with a dashboard token meaning we only need the information to create a dashboard token of the user we want to impersonate. We must build a custom system because Azure's user impersonation requires that it is turned on for all users of the active directory including admin users this would cause a major security flaw and opens a door for hackers and other malicious actors.

Session Hijacking
Is the process of taking the user's session token from their browser and injecting it into your own making you seem like the other person from the server's point of view this would work like the azure built in solution but would require that the user doesn't logout or change their password. This approach is very dubious because it's hacking and would be illegal to do also it runs into the same problem of not being able to know if the user is an impersonated user or not.

Modifying Volpara Analytics Dashboard
One of my first ideas was to modify the Analytics Dashboard to take modified/invalid access tokens with the caveat that this modified token would be created using the user data (IP address, Login Token, date and time… etc) and encrypting it making it so the dashboard can look up this data to check if its valid and if the IP address is on the whitelist of authorised Volpara Health support team members or is using the company VPN (Virtual Private Network). Volpara liked this idea, but it required lots of tweaking to the Analytics Dashboard and there were potential security holes and there was an easier way.

Implement our own custom solution.
When thinking about this approach this guide[2] gave a lot of outlines on how to do an impersonation correctly and safely this was important when making a custom solution since the motto of encryption is "don't reinvent the wheel" but since there are no other options, I had to make a custom solution.

The custom solution uses the "secure token service" to generate an access token with the valid user information provided by the database lookups in the backend. This allows me to bypass the login token and pass the checks of analytics to get a valid dashboard token as the impersonated user. This also came with a slight modification of the analytics to support extra information being provided along with the access token. This extra information would be the impersonators' information (IP address, login token, name, date and time...etc) and would set off a flag in analytics to allow it to know this is an impersonated token and not the real user. This is good so in the log the IT people can know if it was an impersonated user doing that action or the real user and they would know who exactly did the impersonation.

B. Tokens

Tokens are used to authenticate the user in the place of a username and password for every request made to an API/server but in this solution, there are three different kinds of tokens since each token is used at a different stage in the authentication process and used for different purposes outlined below.

Login Token
This token is acquired from Azure Active Directory and is given to the server once the user has authenticated. This would be the token used if Azure Impersonation was turned on and used to get the access token.

Access Token
This token is created after the login token when the user tries to authenticate with Volpara Health's Analytics Dashboard to prevent XSS (Cross-Site Scripting). This token also expires after 2 minutes or when the user logs into Analytics making it a one-use token.

Dashboard Token
The dashboard token gets generated after the access token has been received and validated and is used as the session token within the dashboard to track, log and give the user access to what they have permission to use. with no effort at all.

C. Tools and languages

Volpara Health wants to build a representation of the platform to allow the testing of functionally and training of how to use the tool. That's why I chose Ionic/Angular with Typescript for the front end since it's the framework and language I am the most comfortable with and would allow me to write the front end easier since I wouldn't need to learn something new. Also, it has native browser support and looks very professional and easy. Also, the front-end code is independent of the backend code this allows me to replace the entire front end and migrate to Blazor without much trouble since Blazor is based upon HTML. This allows me to reuse the HTML code and has an inbuilt JavaScript support allowing the reuse of the JavaScript code.

.NET (C#) was chosen for the backend API because that's what Volpara uses for its backend so it would make integration with their servers a breeze in the future.

For testing, we can use the Stopwatch class in .Net(C#) or the JavaScript function Performance.Now to time how long it takes for an API to return the result this can be done for the website as well to see how long it takes for it to get the result back from the API and for it to display it to the user. The lower these numbers are the better the performance the usability of the solution becomes.

III. Design and Implementation

A. Conceptual Design

The solution should be separated into two parts a frontend and a backend this is because it allows the best security since everything is filtered through the backend and the frontend would not need to handle authentication or data manipulation. The front end should only be used to show information to the user and redirect them to the Volpara Analytics Dashboard or to the Active Directory login page.

As seen in Figure 1 the front end mainly interfaces with the backend and is only exposed to those API endpoints this is to

make sure only authenticated users can access the information from the backend and to limit the information returned. For example, there is a case where there are usernames and emails of the users stored but the support team member is not required to know the emails since it's not required to know to do the impersonation. This allows the filtering of the result to return only the usernames limiting the access following the rule of least privilege. As well the frontend never interacts with the Volpara backend API directly to prevent bypassing of the backend authentication and logging. The front end only interacts with Volpara Analytics when it has the user token and the required information from the backend to finish the impersonation.

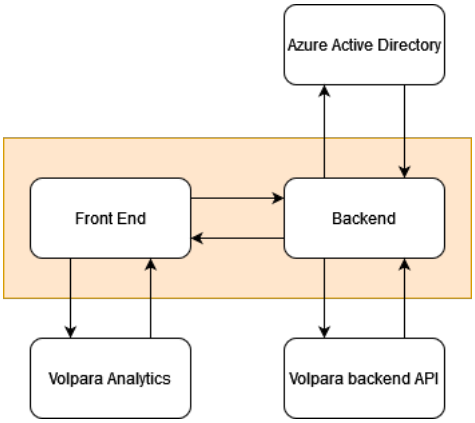

Figure 1: Conceptual Design

The backend is the bridge for the frontend to access the azure active directory to login and authenticate a new user and to get the list of companies and users for the frontend to select which user they want to impersonate. While doing all of this it also protects itself from unauthorized access by having the requirement of all endpoints requiring active directory permissions per user. Also logging all interactions internally so later on it can be seen what users impersonated other users and when.

B. Sustainability Considerations

Sustainability considerations I took into account were social, environmental and technical. Since privacy and security are key for this tool, since it has access to sensitive information and can give a user the power to impersonate another user. This could be used for malicious purposes if in the wrong hands. I also considered environmental factors making sure my solution is as efficient as possible to reduce wasted computing and power. Then the technical aspect is making sure the solution is maintainable into the foreseeable future by using common software design patterns and comments in the code to inform future developers of the function of methods and classes.

In implementation

I programmed the backend API using C# because Volpara is a Microsoft-sponsored company meaning they only use Microsoft products this makes sure that the backend is maintainable for the foreseeable future. This is combined with using the latest Dot Net framework (7) to ensure the solution has the maximum amount of software updates and patches into the foreseeable future.

Privacy

The solution is privacy and security-focused and implements Cyber Security ideas like least privilege and accountability this means in the implementation I give the user the least amount of information as possible and limit their access to any other sources. This refers to making the frontend only interact with my backend solution and only at the end Volpara Analytics dashboard.

My solution also logs all events triggered by the front end calling the API. These logs are in depth with recording IP addresses, usernames, timestamps... etc. To ensure the tool is used by authorized personnel only and to record who does the impersonating and of what user. My solution requires that the user be authenticated with Azure Active Directory to interact with the API in any fashion to prevent any unauthorized data leaking.

C. Implementation

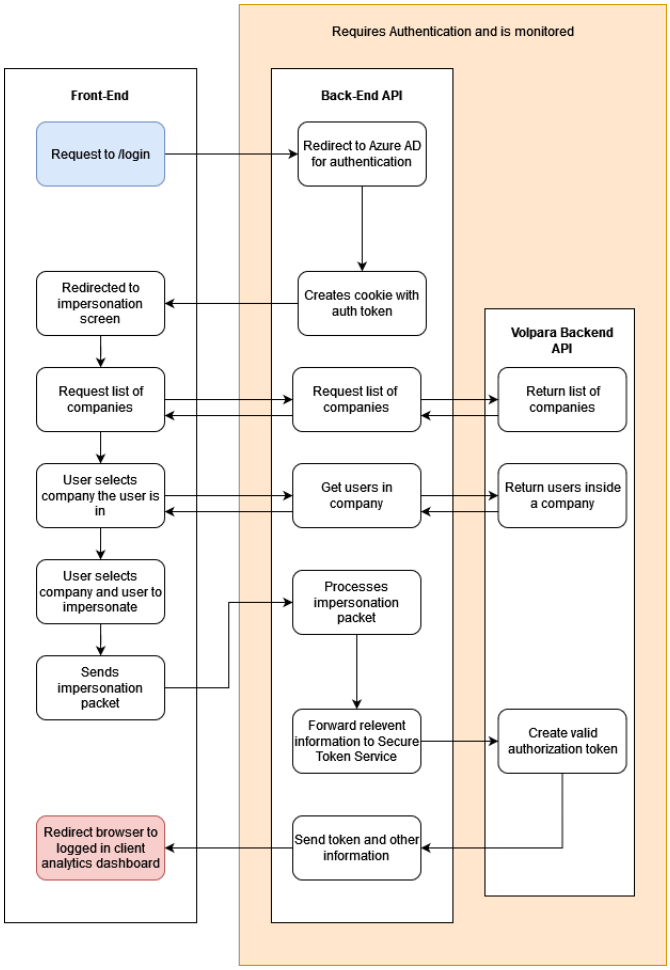

Figure 2: Application Flow Diagram

I originally stuck with the Typescript and angular framework for the front end so I could get to MVP relatively quickly, with the back end being written in C# to make authentication and integration with all azure services and functions as painless as possible.

D. Flow of the solution implemented

As seen in the flow diagram figure 2 the information only travels between the front end and back end of my system and no other external parties this is to reduce points of failure and reduce the risk of exposing private or confidential endpoints to the front end that could be abused and cause privacy failures. The back end only returns enough information for the user to make an informed choice on who to impersonate IE only gives the usernames and not the emails of the clients.

E. Problems with SPA's

The first attempt was to implement a SPA (Single Page Application), theoretically, my use case would benefit from this because it would have made the authentication process a lot easier since there are templates for both Angular and Blazor that have the Azure authentication code built in. But with this convenience comes privacy and security concerns as well as it didn't function at all in both cases. This could have been because of the lack of SSL (Secure Sockets Layer) encryption along with no HTTPS support, since I don't have a valid certificate and self-signed certificates don't bypass this restriction. By not having this certificate it would cause lots of CORS (Cross-origin resource sharing) errors since it would need to reach out to Azure to authenticate the user anyways. Also, SPA's adds extra complexity to the front end since there would be extra "boilerplate" code for the authentication process that wouldn't be transferable between frameworks since it has to be specifically written for the framework that is used.

F. Backend For Frontend (BFF)

Next, I got recommended to use BFF[3] from Rob one of Volpara's developers since it would get rid of the need to have a SPA and would be more secure. This is because it's a reverse proxy basically you get redirected with OpenIDConnect to the Azure login once authenticated you would be given an "auth cookie" that can be used to interface with the back-end API's protected endpoints. By doing it this way it allows us finer control of the data getting sent to the front end and gives us the ability to monitor all outgoing traffic so we can make sure the people interacting with the API are authenticated and if so who they are? and what did they do?. Another benefit is all the authentication happens on the back end meaning a lighter and interchangeable front end that means in the future the front end could be replaced relatively easily with no change needed in the back end.

The advantages of BFF over SPA's are that it separates the responsibilities since the front end can focus on only displaying the information and getting user input while the back end does all the getting, filtering and computing of the data resulting in a major performance increase for the front end since it doesn't have to do any of this. As well it allows security and authorization to happen in the back end for example the user logs in through the login endpoint and gets returned an authentication cookie without the need for the front end to do anything apart from redirecting to the endpoint. This approach also allows us to ensure all data is only getting sent and received by the back end without the need for the front end to go to multiple places and exposing extra endpoints that are not necessary since the back end can make requests to these places and combine the data and send it back to the front end with only one request. This comes with the added benefit of fewer network requests from the front end since these network requests can be combined in the back end.

G. Why the change to Blazor?

Since day one sustainability and maintainability of the solution have been a priority that's why Blazor seems like the obvious choice since Volpara is transitioning to Blazor in the future it seems like the best choice for this solution to be in Blazor so in the future if it needs to be updated or changed they wouldn't need to learn anything new meaning this process can be easily done by anyone. Combined with the MVP being finished in typescript and Volpara being pleased with the user interface and design as well as Blazor being mostly in HTML and JavaScript made it easy to port the code over and gain the same functionally relatively quickly with the added support for future features. This in combination with BFF meaning I wouldn't have to redo all the authentication made it an easy choice to do the rewrite in Blazor.

IV. Evaluation

A. Testing Methodology

These tests are done in "runs" This means the tests are run one after the other on a local machine since the solution is not hosted on the cloud as of yet. This is so we can see the effect of multiple runs on the API and website this could also be done to simulate multiple users using the same website and how their expected response times are going to be. This will show how each run affects the next run with tendencies to have better performance and a lower average response time as the runs go on.

(All results are in Milliseconds)

B. Website Performance

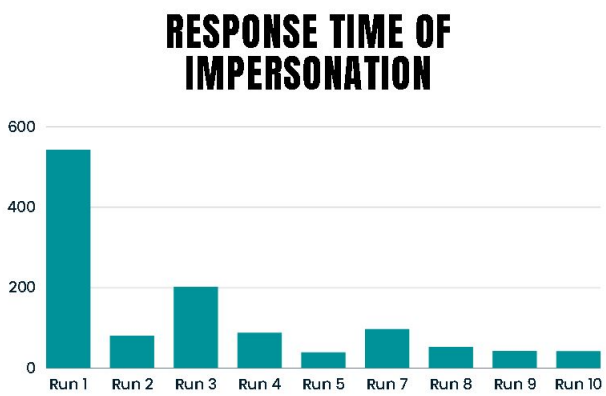

Figure 3: Impersonation page loading times

Figure 3 shows the time it takes for the impersonation page to load as you can see the times are pretty consistent with each other with run 1 being the outlier. This is because the tenants are not cached at this point so there has to be a database lookup to get all the tenants (companies) resulting in a slightly longer load time on the first load.

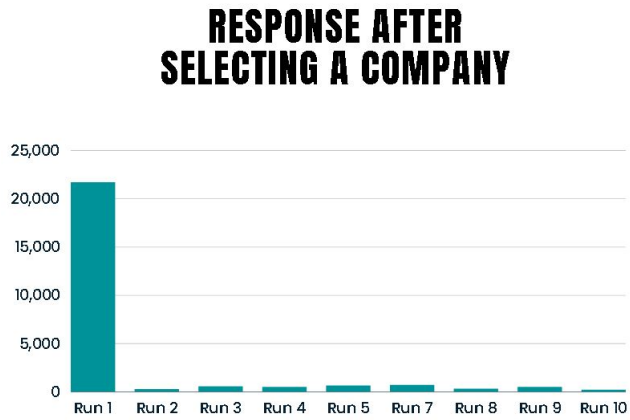

Figure 4: Response time to load a selected company

Figure 4 shows the time it takes after selecting a company for it to load all the users in that company. As seen in the graph the first time selecting a new company can take as long as 22 seconds to load this is due to a database call and a call to the key vault to get the user's email and username. But after the first call has been made the rest would only take about 200ms since the result is cached for 20 minutes.

C. API Performance

Figure 5: Get user response time

Figure 5 shows the get user endpoint that is used to get the currently logged-in users' (first and last) names to check if they are authenticated and to show who is using the tool. As seen in the graph this process takes less than 20ms most of the time since it's a simple query but the variation in the data is most likely caused by the lack of processing power of the computer since running the whole solution is CPU and memory-intensive.

Figure 6: Get tenant response time

Figure 6 shows the get tenant endpoint that is used to get all the companies that the user could impersonate. The graph looks pretty consistent at around 20-40ms with the outlier being run 10 because I suspect it was performing the database query again since at the point of testing 20 minutes or so had passed.

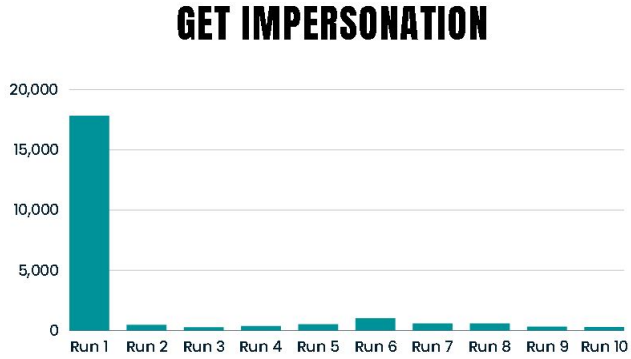

Figure 7: Get impersonation response time

Figure 7 shows the get impersonation endpoint that is used to get the users of a company/tenant. The graph shows that the first run can take 18 seconds to complete since this is the call to key vault that is a lengthily query this is also the reason why the page can take a while to load when selecting a company. But after the result is cached the response time goes down to around 200-500ms.

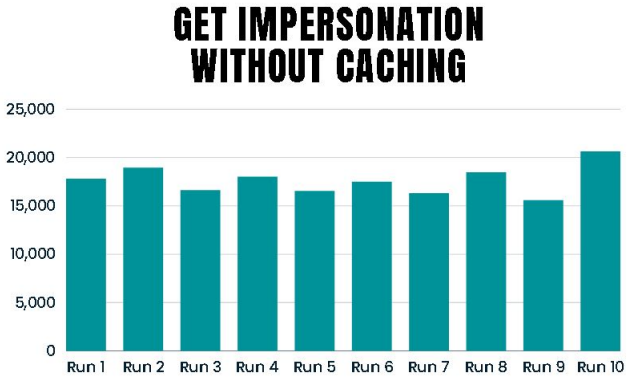

Figure 8: Get impersonation without cache response time

Figure 8 shows the initial first run of the get impersonation endpoint done multiple times to show the worst-case scenario of how long a user would have to wait if the tenant/company wasn't cached. As the graph shows this could take 15-20 seconds showing the importance of caching this is why on the front end the result is cached so if a user selects the same company it would only take a couple of ms to get the result also there is some caching in the back end but this cache is cleared every 20 minutes.

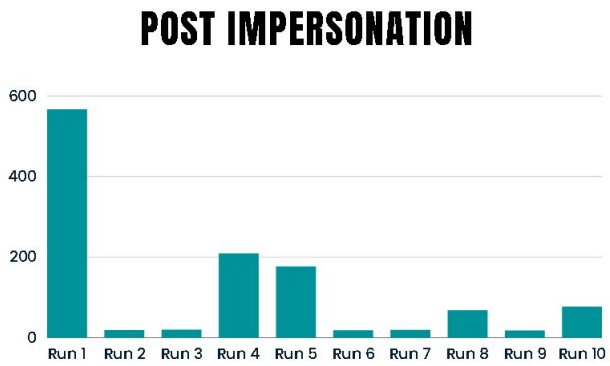

Figure 9: Get impersonation post response time

Figure 9 shows the post-impersonation endpoint that is used to create the user token for logging into the Volpara Analytics Dashboard. This post request is very inconsistent since it involves an internal server lookup for the email of the user and forwarding of requests to get the required information to return to the front end.

Overall I found that there was about a 50ms delay between the request coming back from the backend and it being displayed to the user. I assume with a dedicated server with better hardware these times would be significantly less and would result in a faster user experience. But as it stands right now it's completely usable after the initial query when the results of those queries are cached if they are not it would result in a slower user experience but wouldn't diminish too much from the user experience.

D. Future enhancements

The future enhancements are ways the solution could be improved if there was more time in the project and optional features that would provide extra functionality or security.

- Caching for the tenants and users in the back end with respect to support team members permissions.
- A way for updating the cache in the back end at regular intervals or when needed.
- Hash comparisons for the incoming impersonation packet to make sure it hasn't been tamped with in transit.
- Encryption of traffic between the front end and back end with a encryption scheme like AES 256 bit.
- Integration with Volpara logging so that we could get notified if an impersonation has happened and have all the logs in a central system instead of being local.
- IP address range restrictions to prevent IP addresses outside of Volpara from being able to use the API or visit the front end web page.
- Time restrictions like limiting the access of the impersonation tool to only office hours IE 9-5.
- Could limit the impersonation time to like 10 minutes then log the user out to prevent people from leaving their computers logged in as another user.
- HTTPS certificates to mitigate CORS errors and encrypt traffic while in transit.
- To host the solution on the cloud or on Volpara servers to make it accessible to all support staff members.
- If its deployed it should have a CI CD system to push new changes automatically to the cloud in production.

V. Conclusion

As part of this project, a working solution was developed for user impersonation employing Blazor and .NET to meet the requirements of the end-user. Volpara liked the simplicity of the solution and the functionality it brings since they can now impersonate a user at the lowest levels in the analytics dashboard and see exactly what that user would see without the need for emulation.

VI. References

[1] Divkamath, "Impersonate another user (microsoft dataverse) - power apps," (Microsoft Dataverse) - Power Apps — Microsoft Learn, https://learn.microsoft.com/en-us/power-apps/developer/data-platform/impersonate-another-user (accessed Jun. 1, 2023).

[2] Nolan Ramsey, "Designing an impersonate feature," Nolan Ramsey, https://www.nolanramsey.com/theradreport/2019/8/4/designing-an-impersonate-feature (accessed Jun. 2, 2023).

[3] Raw Coding, "Backend for Frontend for ASP.NET Core Authentication," YouTube. May 28, 2023. Accessed: Oct. 15, 2023. [Online]. Available: https://www.youtube.com/watch?v=VWTdOAi6Yic