**VisionRMM – Documentation**

Contents

[1. Follow this URL 1](#_Toc137036809)

[2. Install VS Code 1](#_Toc137036810)

[3. Run these commands 1](#_Toc137036811)

[4. To exclude files 2](#_Toc137036812)

[5. C# Back-end Test Coverage Screen 2](#_Toc137036813)

[6. Logging resources: 2](#_Toc137036814)

[7. Health Checks 3](#_Toc137036815)

[8. Authentication and Authorization 3](#_Toc137036816)

[Key Creation – usually once in 3 months 4](#_Toc137036817)

1. Follow this URL: <https://blog.nimblepros.com/blogs/code-coverage/>

# Install VS Code

1. Run these commands: install report-generator package once:

dotnet tool install -g dotnet-reportgenerator-globaltool

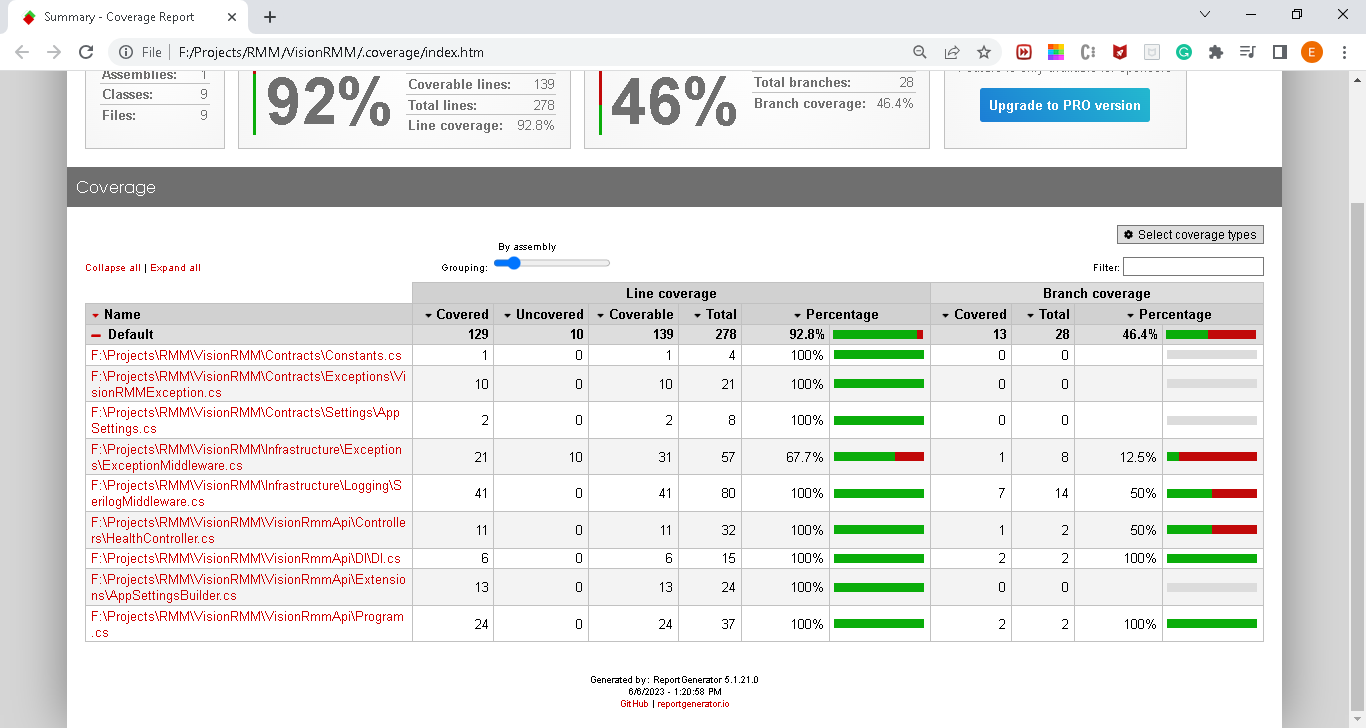
Then Every time to generate BACK\_END C# coverage run these commands from the solution folder:

dotnet test /p:CollectCoverage=true /p:CoverletOutputFormat=lcov /p:CoverletOutput=../BackEndCoverage.info"

reportgenerator -reports:BackEndCoverage.info -targetdir:.coverage

1. To exclude files: <https://github.com/coverlet-coverage/coverlet/blob/master/Documentation/MSBuildIntegration.md#source-files>
2. C# Back-end Test Coverage Screen:

Under IntegrationTests and UnitTests, the TestResults directory must be deleted once in awhile because is growing each time tests are ran:



# Logging resources:

Currently Logging to “Logs/log{Environment}.txt file file, however there is provision to log to SEQ or other Serilog sinks:

"WriteTo": [

{

"Name": "File",

"Args": {

"outputTemplate": "[{Timestamp:HH:mm:ss} {SourceContext} [{Level}] {Message}{NewLine}{Exception}",

"formatter": {

"type": "Serilog.Formatting.Compact.CompactJsonFormatter, Serilog.Formatting.Compact"

},

"rollOnFileSizeLimit": true,

"fileSizeLimitBytes": 4194304,

"retainedFileCountLimit": 15,

"rollingInterval": "Day"

}

}

Other provisions:

{ "Name": "LiterateConsole" },

{

"Name": "Seq",

"Args": { "serverUrl": "http://localhost:5341" }

},

{

"Name": "Console",

"Args": {

"outputTemplate": "[{Timestamp:HH:mm:ss} {SourceContext} [{Level}] {Message}{NewLine}{Exception}"

}

},

<https://github.com/datalust/serilog-middleware-example?ref=blog.datalust.co>

<https://blog.datalust.co/smart-logging-middleware-for-asp-net-core/>

<https://www.codeproject.com/Articles/5344667/Logging-with-Serilog-in-ASP-NET-Core-Web-API#:~:text=Create%20ASP.NET%20Core%20Web%20API%20Project&text=Give%20it%20a%20name%20like,then%20choose%20to%20install%20Serilog>.

Other Serilog sinks: <https://github.com/serilog/serilog/wiki/Provided-Sinks>

# Health Checks

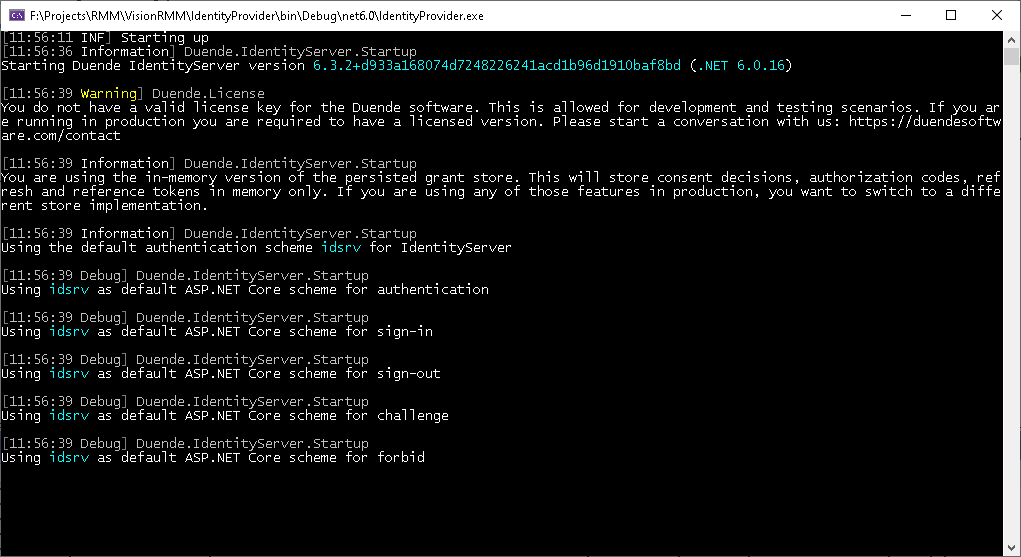
<https://github.com/xabaril/aspnetcore.diagnostics.healthchecks>

# Authentication and Authorization

OpenID connect, own identity server employing

Duende: <https://github.com/DuendeSoftware/IdentityServer>

<https://docs.duendesoftware.com/identityserver/v6>



Project is generated with: “dotnet new isempty –n IdentityServer”

UI is generated with “dotnet new isui”

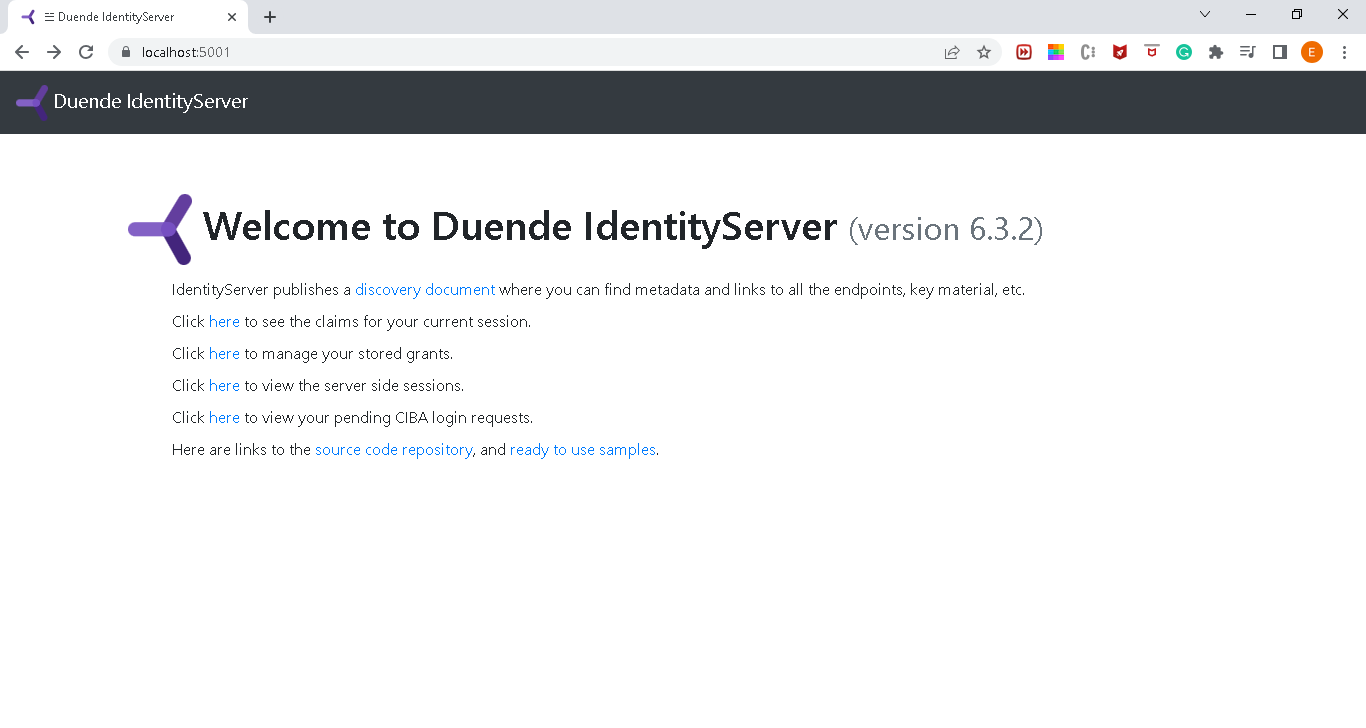
<https://docs.duendesoftware.com/identityserver/v6/fundamentals/keys/automatic_key_management/> only in paid version

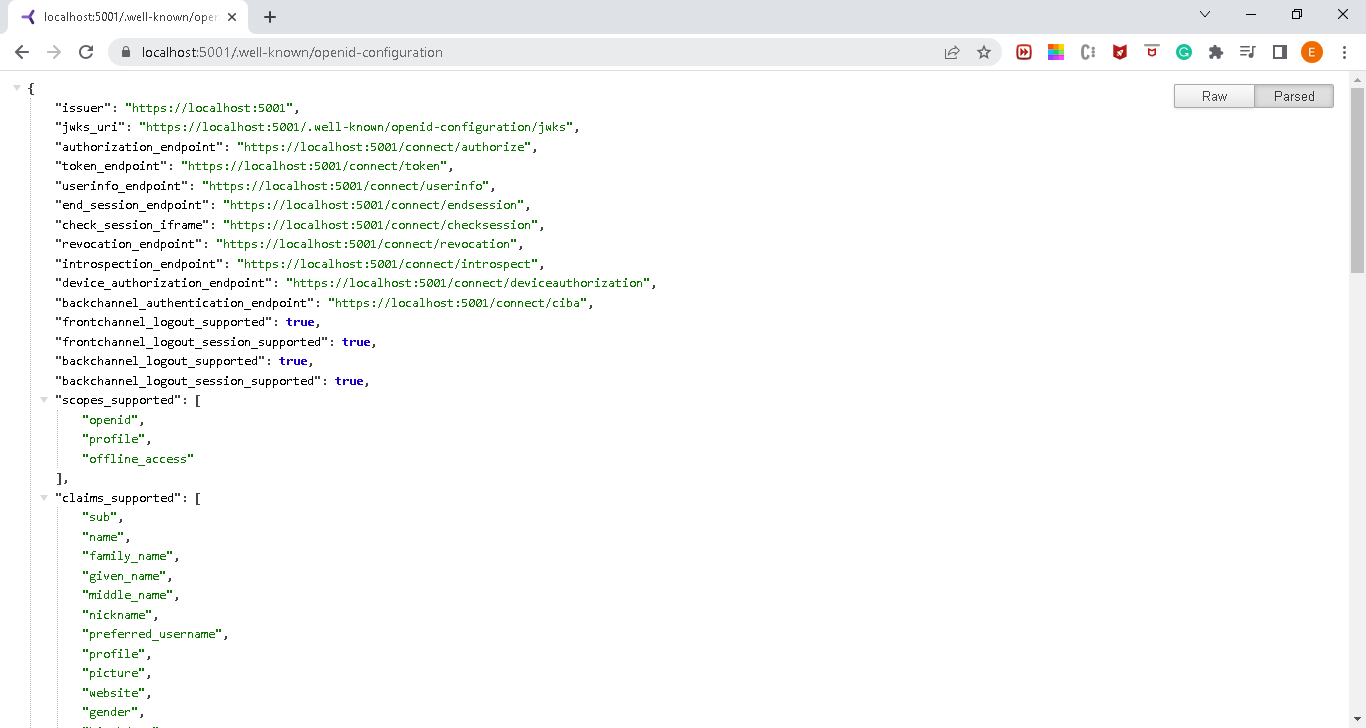
<https://docs.duendesoftware.com/identityserver/v6/fundamentals/keys/static_key_management/>

## Key Creation – usually once in 3 months

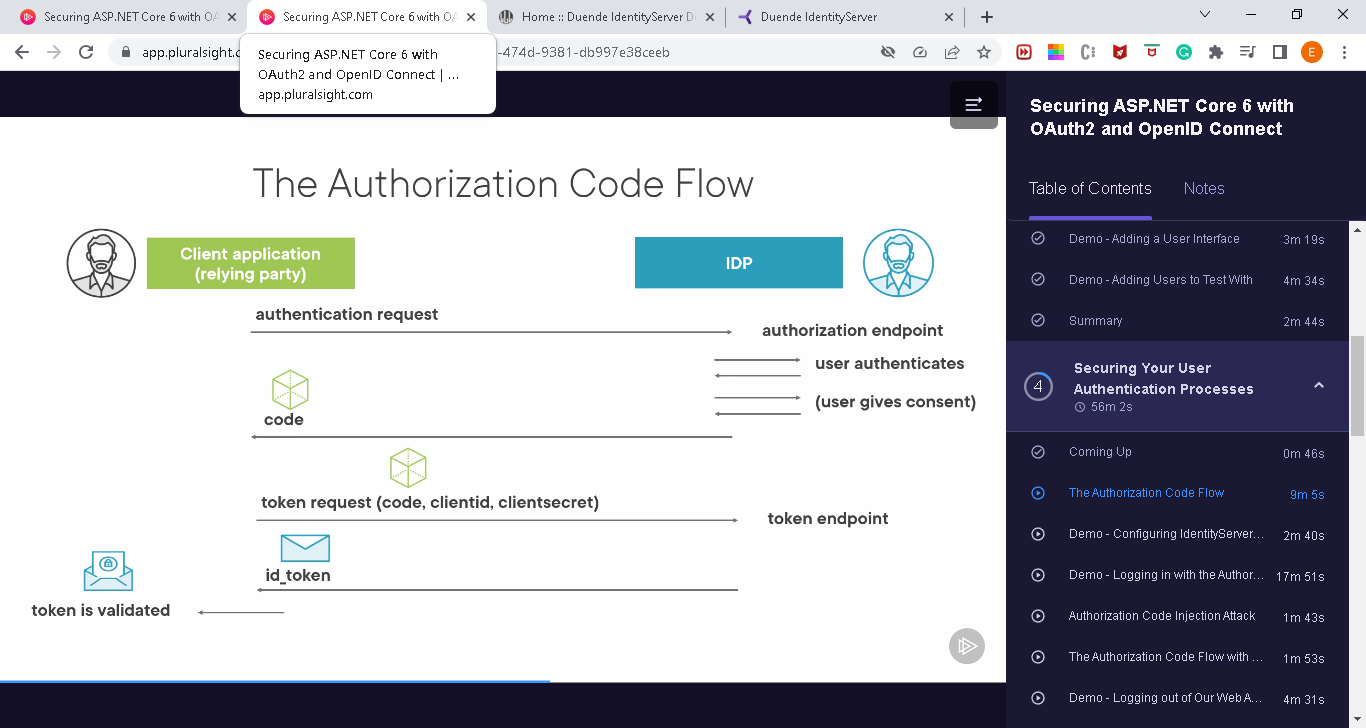
Without automatic key management, you are responsible for creating your own cryptographic keys. Such keys can be created with many tools. Some options include:

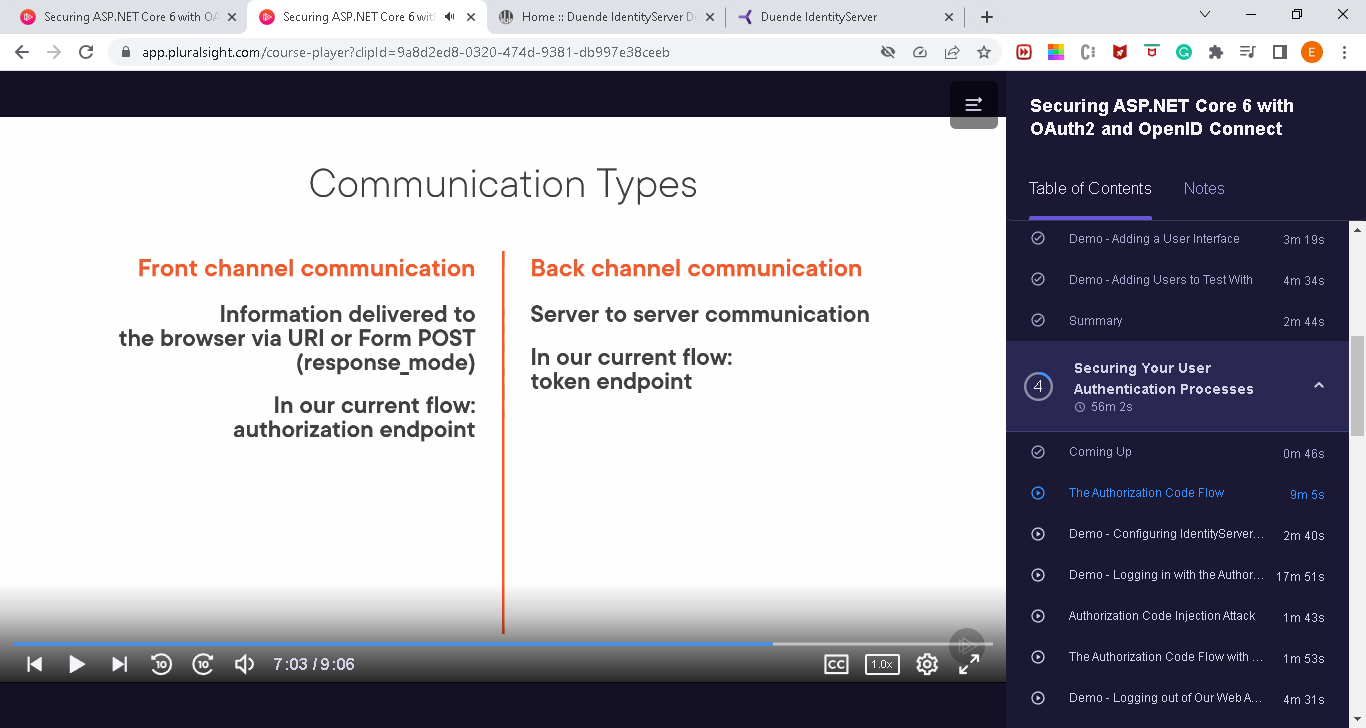
* Use the PowerShell commandlet [New-SelfSignedCertificate](https://learn.microsoft.com/en-us/powershell/module/pki/new-selfsignedcertificate?view=windowsserver2022-ps) to self-sign your own certificate
* Create certificates using [Azure KeyVault](https://learn.microsoft.com/en-us/azure/key-vault/certificates/certificate-scenarios)
* Create certificates using your Public Key Infrastructure.

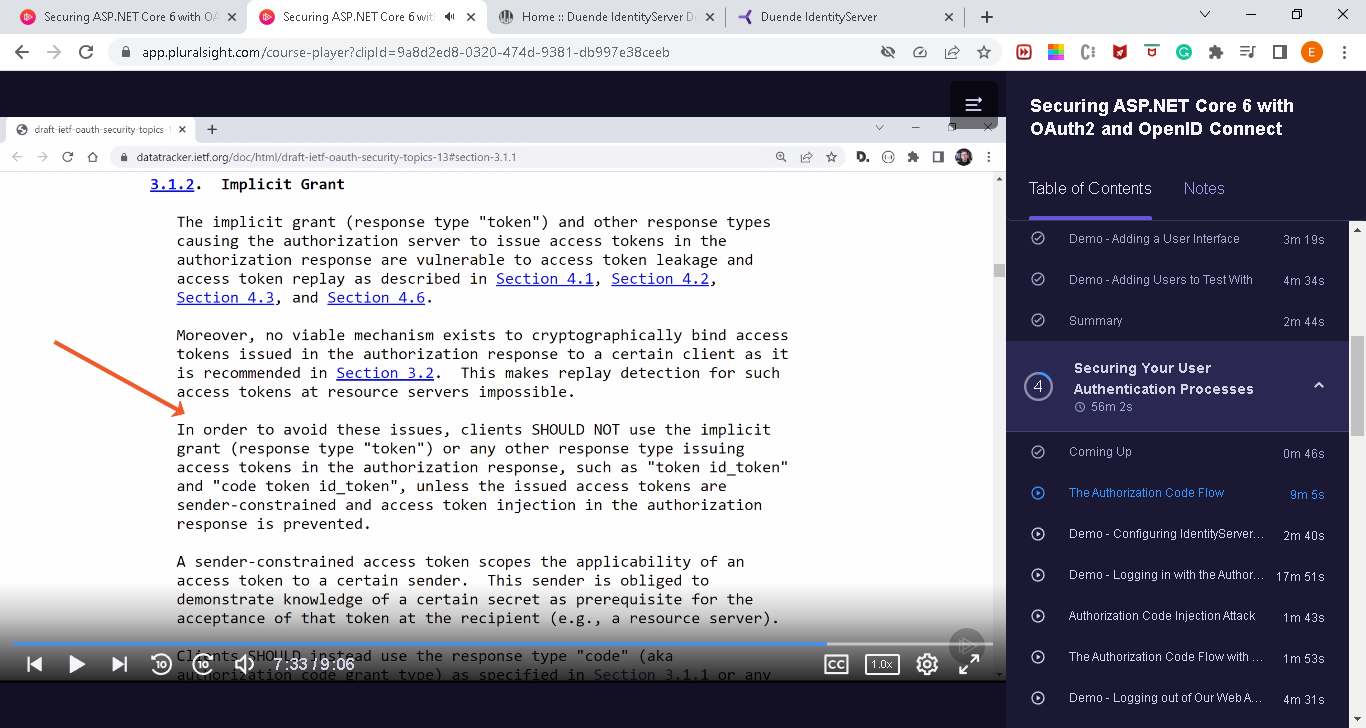


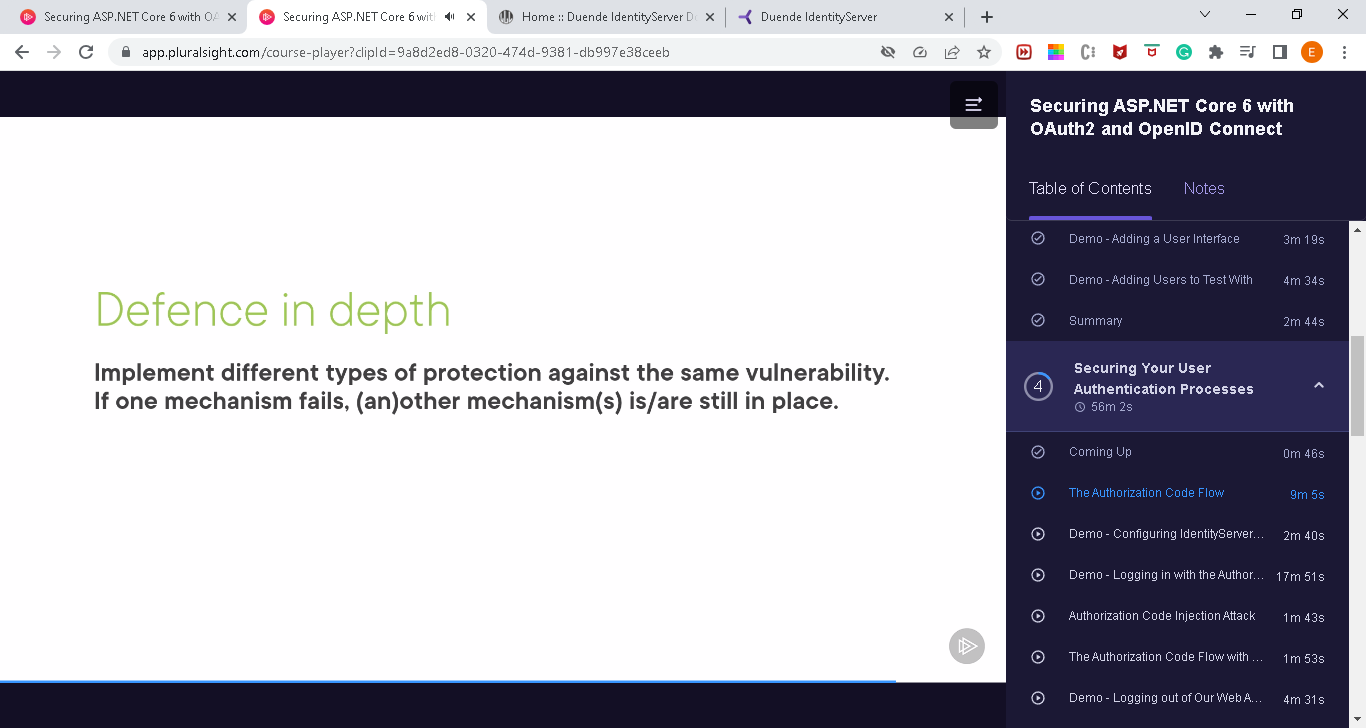


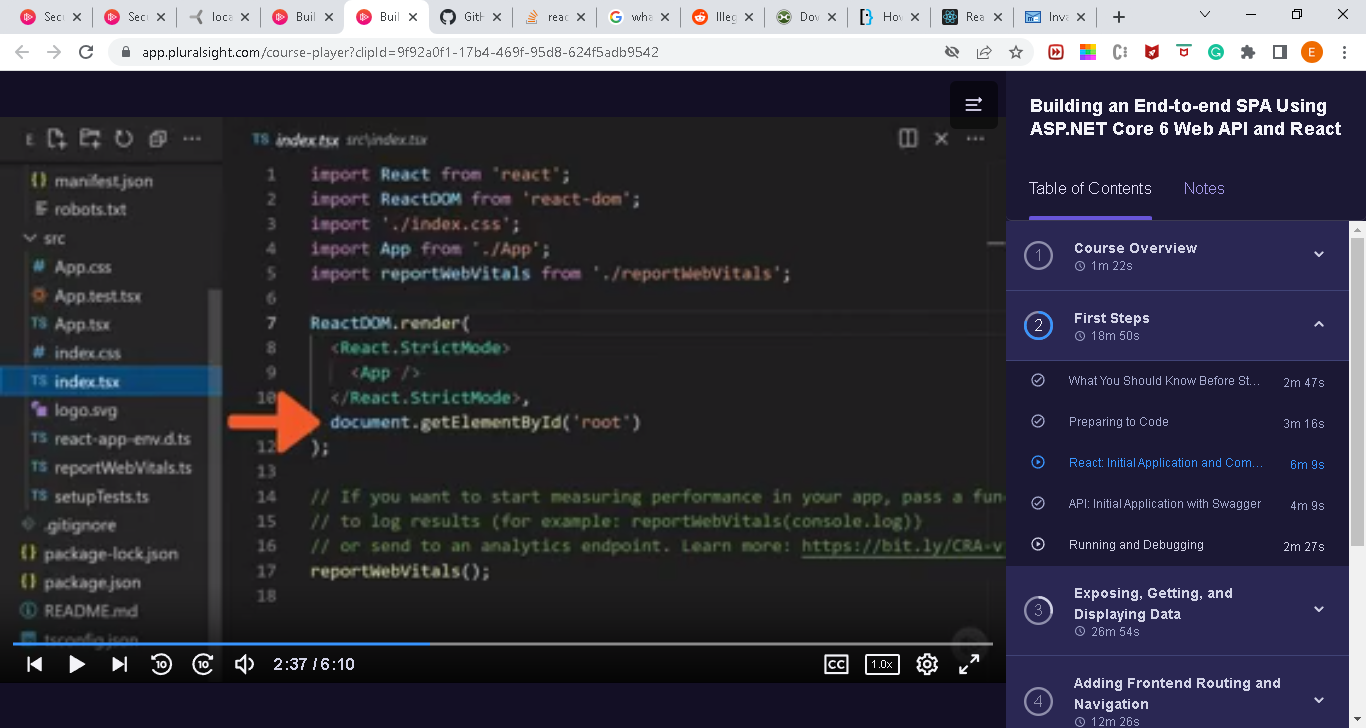
Authorization Flow



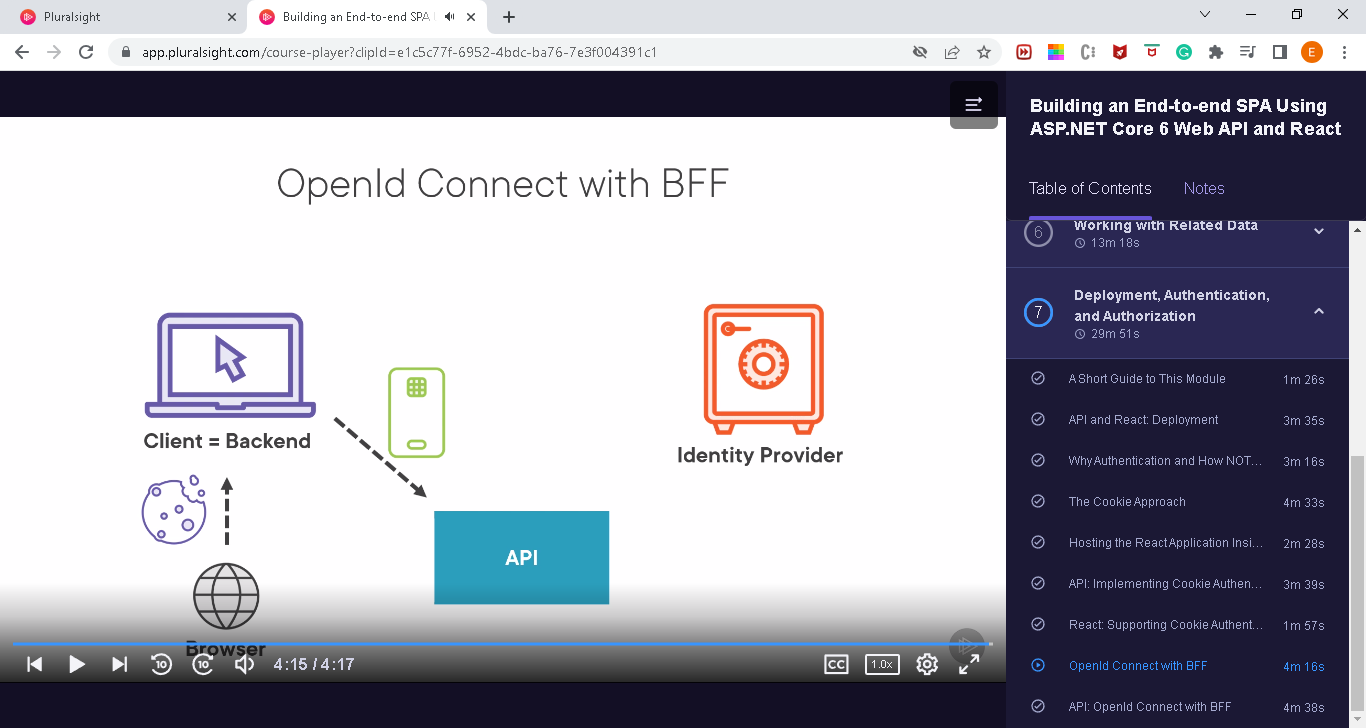








It is secure:



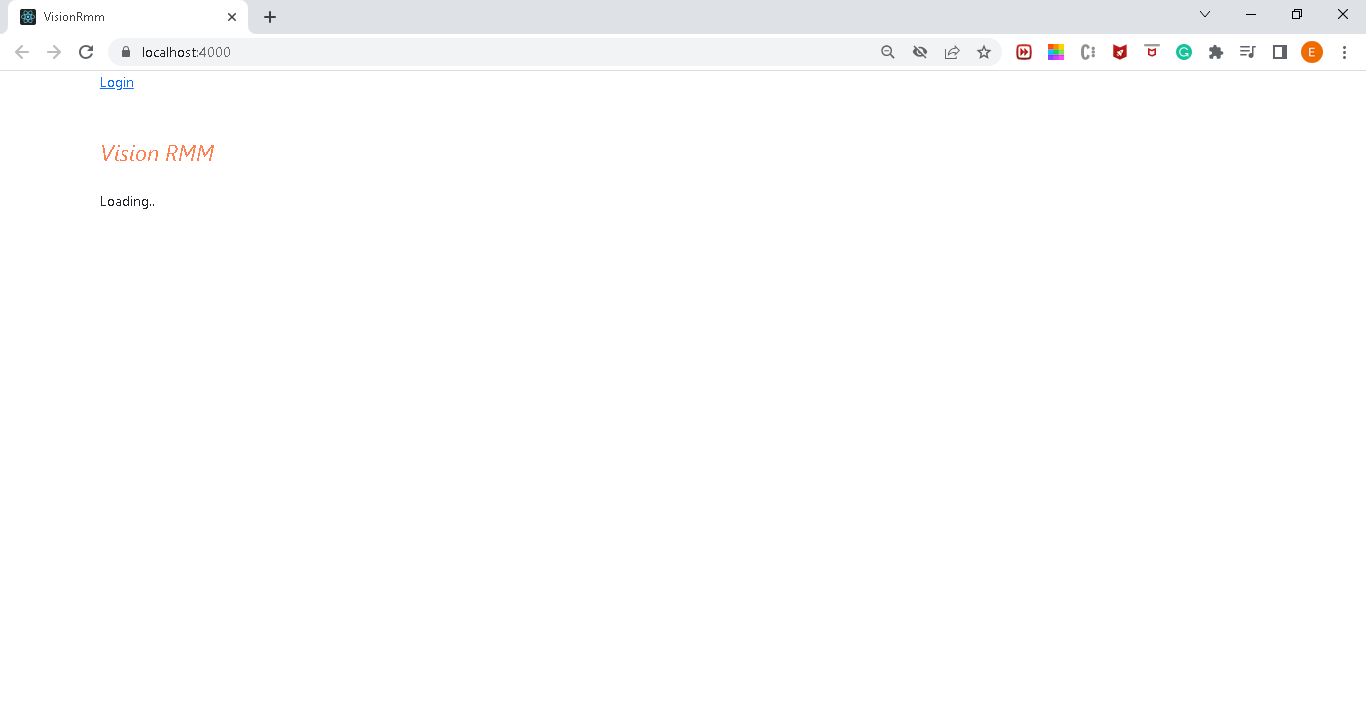
# Flow

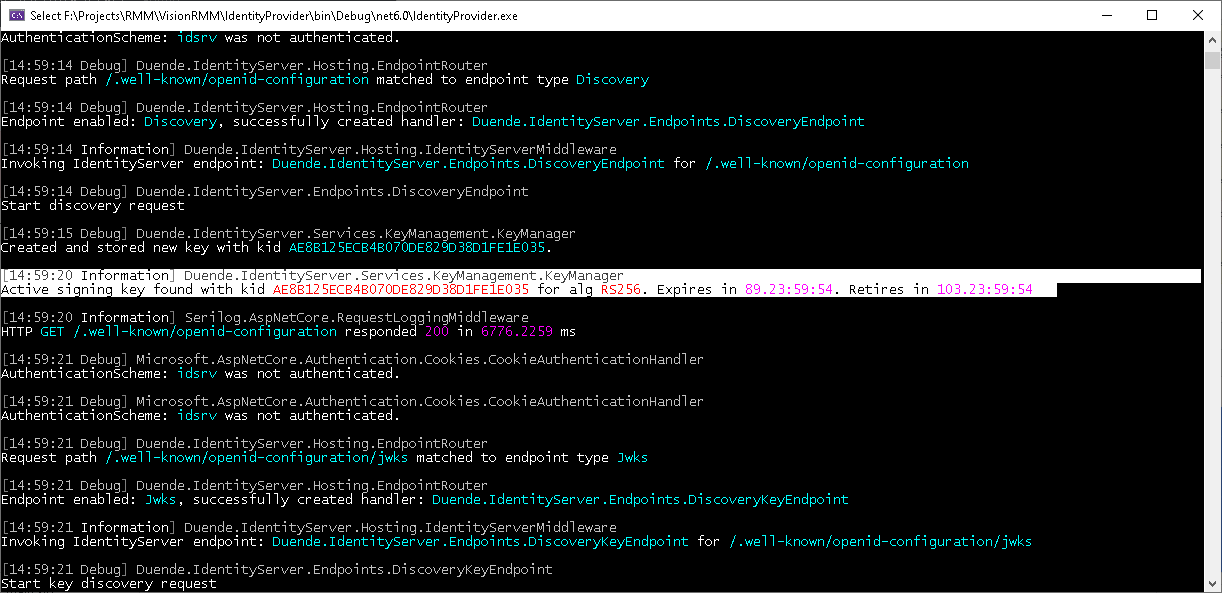
* 1. React will be hosted withing the API: that si because “browser cannot keep a secret”

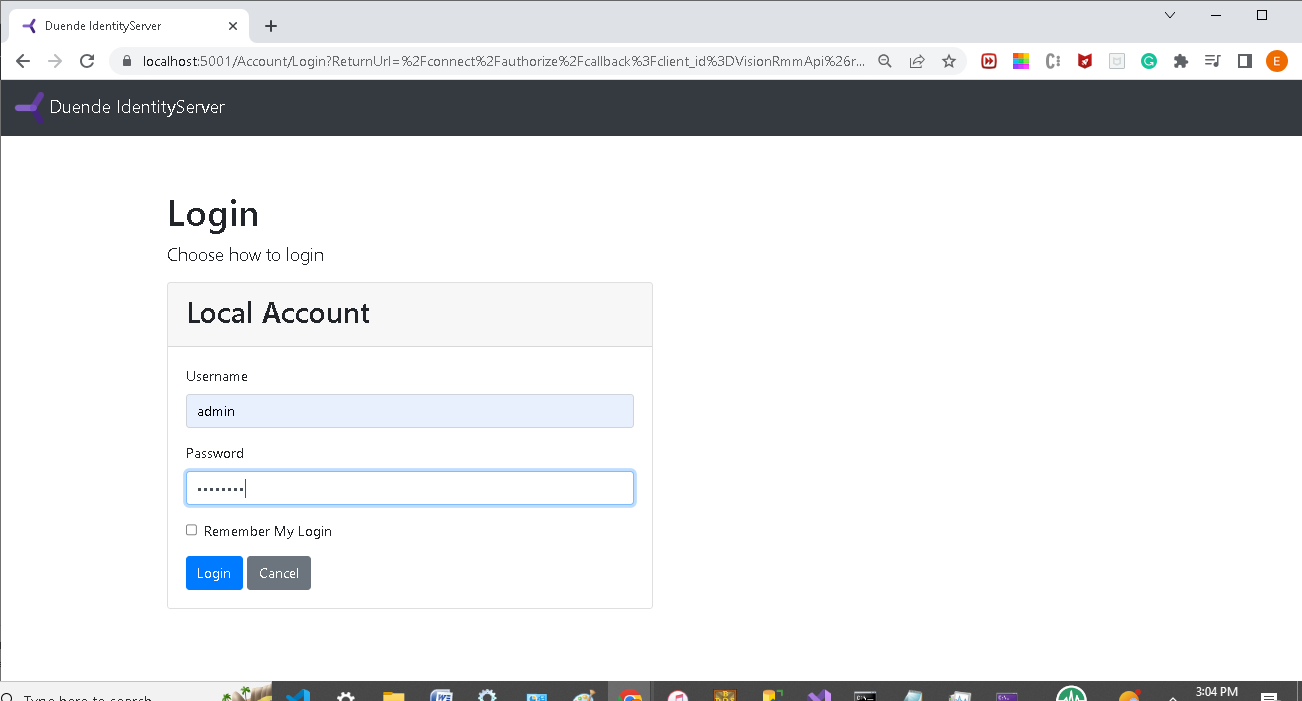
Every time we build React => there is a powershell script which will copy the react file under the API

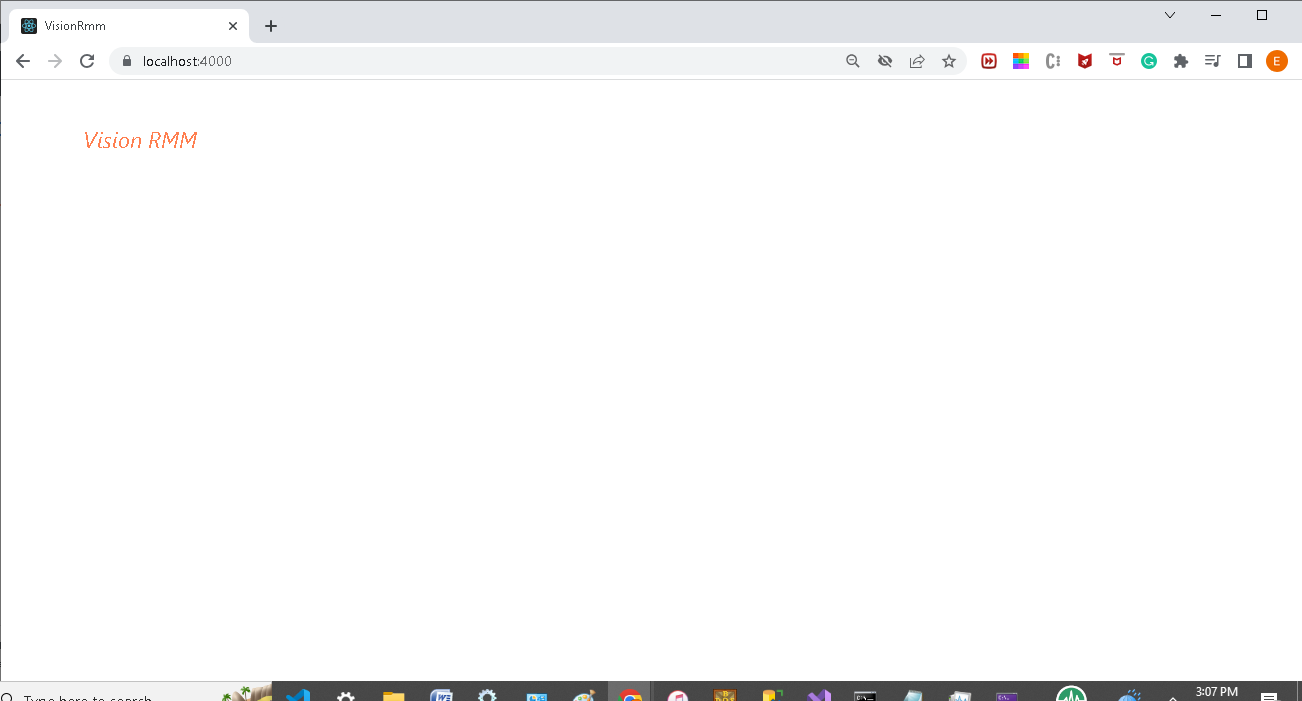
**PublishUI.ps1**

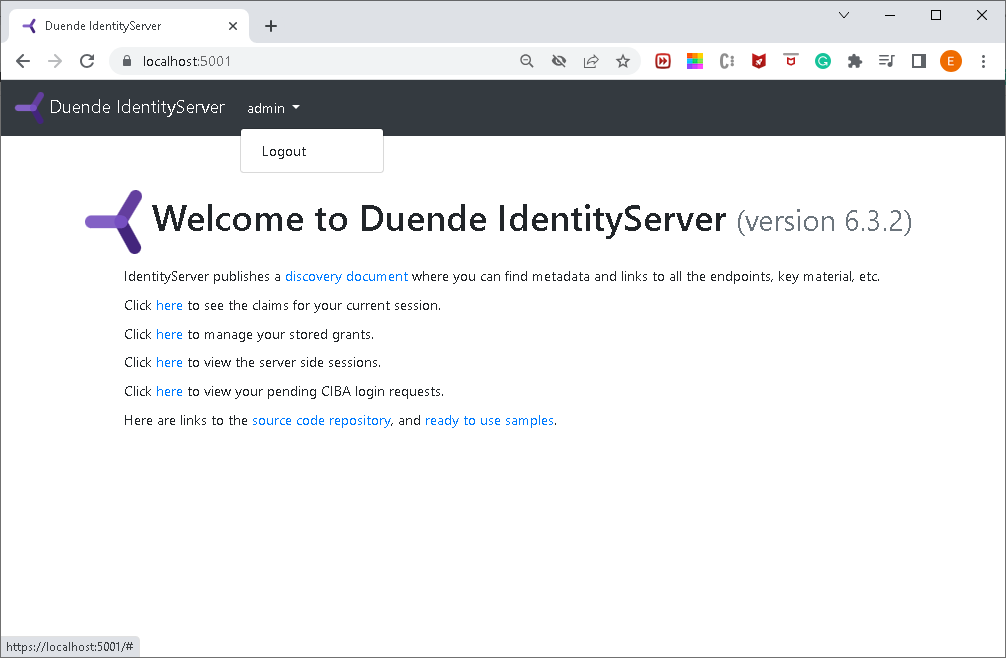
* 1. Todo: Add Test Users into appSettings
  2. Initially:











# Errors:

Creating react app from Vstudio:

