DTU

IT Diplomuddannelsen

Web Teknologier - Efterår 2018

Bouncer: Centraliseret autentifikation og adgangskontrol i en service-baseret arkitektur

Studerende: Allan Køtter
Studienummer: s140381
Underviser: Daniel Rubin-Grøn

Dato: 18. december 2018
Anslag: 41921 tegn (5755 ord)

Indhold

1. Indledning

Internettet er skabt til at kunne dele data og information med andre internetbrugere. Ofte ønsker man dog kun at dele udvalgte informationer med udvalgte brugere, og en meget central del af en web-applikation er derfor, at kunne identificere de brugere der ønsker at tilgå applikationen og begrænse adgangen til udvalgte dele af applikationen og applikationens data.

I dag udvikles web applikationer i stigende grad som service-baserede løsninger i stedet for monolitiske løsninger som tidligere, hvor dele af applikationernes funktionalitet og ansvar uddelegeres til isolerede services. En af fordelene ved dette er, at funktionaliteten dermed kan deles af mange applikationer, og kun skal vedligeholdes ét sted. Samtidig kan funktionaliteten skaleres og opgraderes uafhængig af de applikationer der bruge servicen.

Pga. brugerautentifikation er en helt central sikkerhedsfunktionalitet, der ofte er målet for hacking og identitetstyveri, vælger mange udviklere at uddelegerer ansvaret for denne funktionalitet til 3. part, som der er tillid til - dette kaldes *Federated Identity*. Brugeren gives dermed adgang til udviklerens applikationen på baggrund af en elektronisk identitet oprettet hos f.eks. Microsoft, Google eller Facebook. Adgangskontrol vha. *Federated Identity* implementeres typisk vha. OAuth 2.0, der er den mest udbredte tekniske standart for en distribueret, tokenbaseret adgangskontrol.

Hvis man som virksomhed ikke ønsker at uddelegerer autentifikation og adgangskontrol til 3. part, kan autentificering og autorisation via OAuth-protekollen implementeres på virksomhedens egne servere - stadig som distribueret service / funktionalitet. Jeg ønsker med dette projektet at undersøge, hvordan sådan en centraliserer autentifikations-service kan implementeres, hvad den består af og hvordan den fungerer. Jeg vil arbejde med en konkret implementering i form af *IdentityServer4*, som er open source middleware, der kan indbygges i ASP.NET Core applikationer, og vha. denne service give brugere adgang til en anden applikation (klient) sam til data fra en tredie webservice.

Læringmålet med projektet er at blive fortrolig med konceptet *Federated Identity* og undersøge et autentifikations- og autorisations-flow, som implementeres med OAuth 2.0 og Open Identity i en servicebaseret arkitektur. Desuden ønsker jeg med projektet at afprøve applikationsudvikling med ASP.NET Core og Entity Framework Core, som er Microsofts nyeste skud på .net frameworket.

Selvstudie: OAUTH 2.0 og Open Identity.

2. Selvstudie: OAUTH 2.0 og Open Identity

2.1 Webapplikationer med autentifikation og autorisation af brugere

Internettet er en infrastruktur, der stiller services til rådighed for applikationer [16], således at internetbrugere - ved hjælp af disse applikationer - kan dele data med andre internetbrugere. Siden internettets start har det derfor været nødvendigt, at kunne identificerer de personer der ønsker adgang til en ressource, for dermed at kunne differentiere adgangen til indhold mellem forskellige brugere.

Autentifikation: Identificering en bruger, hvilket typisk sker vha. brugernavn og password. Denne funktionalitet er ikke en del af OAuth-standarten [5]. Funktionaliteten bygges typisk oven på OAuth vha. af en autentifikations-protekol f.eks. Open Identity.

Autorisation / Adgangskontrol: Fastlægger brugerens rettigheder, dvs. hvad en allerede autentificeret bruger har adgang til og ikke har adgang til. Dette er nøglefunktionaliteten i OAuth.

Autentifikation er en forudsætning for autorisation og måden at håndtere dette har udviklet sig meget gennem tiden. Overordne set kan webbaseret autentifikation opdeles i 3 overordnede kategorier:

Simpel HTTP autentifikation

Dette er den mest basale type af autentifikation og er defineret i HTTP standarten med specifikationen RFC 7235 [8]. Websider (endpoints) konfigureres på serveren med autentifikationstypen *Basic* eller *Digest* og brugere-registeret vedligeholdes typisk i en krypteret tekstfil på serveren.

Hvis en klient sender en forespørgsel på en af de beskyttede sider, svarer serveren med en *401 Unauthorized response* der åbner en login-promt i browseren. Efter brugeren har logget ind, sendes brugernavn og password med WWW-Authenticate attribute i headeren, der verificeres mod tekstfilen på serveren inden den ønskede side returneres.

Udfordringer ved denne type autentificering er at http-protekollen er stateless, og brugernavn og password skal derfor sendes med hver forespørgsel, med en potentiel risiko for eksponering overfor uvedkommende. Brugernavn og password er kun beskyttet af basal kryptering (SSL) med høj risiko for identitetstyveri. Desuden giver metoden ikke mulighed for en differentieret brugestyring og adgangskontrol [4], [6].

Autentifikation med database, sessioner og cookies

Moderne monolitiske webapplikationer bliver traditionelt udviklet med autentifikation mod et lokalt brugere- og rolle-register i applikationsdatabasen, samt brug af sessioner og cookies. Herved undgår man at brugernavn og password sendes med alle forespørgsler og samtidig flyttes autentifikationen og brugerstyringslogikken delvist fra serveren til applikationen, hvorved en differentieret brugerstyring med større grad af kontrol over brugeren lettere kan implementeres. Denne autentifikations-model indbefatter desuden at brugeren kun skal logge ind i applikationen én gang, hvorefter der oprettes en session til brugeren sammen med en session-cookie, der identificere brugeren under sessionens varighed [18].

Modellen er som udgangspunkt mere sikker end HTTP autentifikation, fordi brugernavn og password kun overføres én gang. Herefter er det kun session-cookien, der sendes med i klientens forespørgsler og sessions-cookien slettes når brugeren afslutter sessionen, så denne ikke kan udnyttes af en evt. angriber. Modellen giver dog stadig sikkerhedsmæssige udfordringer i form af f.eks. *man-in-the-middle-attacks*, *cross-site request forgery* og *SQL-injections* og al sikkerheden skal implementeres i applikationen, og kræver således en høj grad af viden om internetsikkerhed hos udvikleren [19].

Tokenbaseret authetifikation

Med en mere moderne arkitektur, der er baserede på distribuerede systemer og microservices, giver sessionsbaserede autentifikation bl.a. en udfordringer med *cross-origin resource sharing (CORS).* Når autentifikation sker vha. en session, der lagres på applikationsserveren, kan autentificeringen ikke bruges, hvis brugeren forsøger at tilgå ressourcer på en anden service / endpoint. Fordi deling af ressourcer på tværs af servere og domæner bliver mere og mere almindeligt i nutidens distribuerede applikations-arkitektur på internettet, er tokenbaseret autentifisering udviklet som en løsning på dette.

Med tokenbaseret autentifikation, genereres en krypteret tekststreng ved login, der returneres til klienten og som indeholder alle nødvendige informationer om brugeren - herunder brugerens identitet og tilladelser til andre services. Denne token kan derefter sendes med i efterfølgende http-forespørgsler, og kan identificere og legitimere brugeren på tværs af servere og domæner. Metoden er således *stateless* idet adgangen ikke baseres på lagrede informationer på den enkelte server og autentifikationen kan dermed deles af flere forskellige services på tværs af domæner [19].

Tokenbaseret autentifikation giver således også mulighed for en centraliseret / distribueret autentifikation, hvor udvikleren kan lade en 3. part f.eks. Microsoft eller Google varetage autentificeringen og al sikkerheden forbundet med dette. Identitetstyveri sikres typisk med både kryptering og digital signering med f.eks. *public/private key*, således af ægteheden af informationerne kan verificeres. Et eksempel på en tokenbaseret autentifikationsprotekol er *Open Identity.*

2.2 OAUTH 2.0 og Open Identity

OAUTH 2.0 er idag den mest udbredte industristandart for implementering af distribueret, tokenbaseret adgangskontrol i web-, mobil- og skrivebordsapplikationer, og standarten er beskrevet i RFC 6749 [5]. OAuth 2.0 kan implementeres sammen med forskellige autentifikations-protekoller bl.a. Open Identity. Fordelene ved OAuth 2.0 er en høj grad af

sikkerhed, fordi token-strengen beskyttes af tovejskryptering og brugernavn og adgangskode kendes ikke af klient-applikationen, og dårlig sikkerhed i klient-applikationen ikke fører til identitets-tyveri.

OAuth er blevet udviklet for at afhjælpe de udfordringer der opstod med de første forsøg på at implementerer *Federated Identity*, hvilket bestod i at lade brugere oplyse brugernavn og password til 3. part f.eks. Facebook i klientapplikationen. Klientapplikationen kunne dermed logge på brugerens facebook-konto, og kontrollere at brugerens identitet. Problemet var dog at klienten dermed samtidig fik fri adgang til alting på brugerens Facebook-konto uden brugerens eksplicitte samtykke - herunder alle brugerens kontaktpersoner og også adgang til f.eks. at slå indhold op på brugerens væg. Dette problem kaldes *The password anti-pattern* [3], og ses stadig brugt nogen steder i dag, men det må kraftigt frarådes.

Terminologi i OAuth 2.0

Fra [10], [14] og [2]

User: En bruger, der ønsker at tilgå en eller flere ressourcer, der beskyttes af OAuth-serveren, og som derfor har brug for at identificere og legitimere sig over for OAuth-serveren. OAuth-serveren holder ikke nødvendigvis selv identitetsoplysninger om brugeren, men sørger for at brugeren, som ejer informationerne kan give applikationer og webservices adgang til at anvende hvis det kræves.

Claims: En brugers krav / rettigheder til ressourcer - både identitetsoplysninger som f.eks. brugerens navn, emailadresse m.m. samt hvilke ressourcer brugeren har ret til at anvende. De informationer som brugeren selv ejer, kan brugeren stille til rådighed for applikationer og webservices, som brugeren ønsker at tilgå. De rettigheder til ressourcer, som brugeren har fået tildelt, bruges til at styre brugeren adgang til ressourcerne.

Client: Klienten er den applikation, som brugeren ønsker at benytter og som derfor har brug for at have adgang til brugerens identitet og brugerens *Claims*. Klienten skal også ofte kunne tilgå andre ressourcer f.eks. webservices for at hente data, og skal derfor kunne identificerer sig over for OAuth-serveren med ClientId og Client-secret. For at en bruger kan tilgå en webservice (ressource) via en klient, kræver det at både klienten og brugeren har rettigheder til denne webservice.

Resources: En ressource er nogen man ønsker at beskytte og give adgang til til udvalgte brugere og/eller klienter - det kan enten være brugerens identitetsdata eller data fra en webservice. I OAuth 2.0 har alle ressourcer et unikt navn, som identificerer ressourcen og som klienter kan anvende for at tilgå ressourcen.

Resource owner: Den bruger, der ejer en service eller noget information, og som kan give andre adgang / tilladelse til at tilgå ressourcen. brugeren selv ejer f.eks. identitetsoplysninger om sig selv, som tidligere beskrevet, mens systemejeren ejer de ressourcer, som udstilles via en webservice.

Autorisations service: OAuth-serveren, der sikrer ressourcer i forhold til autentificerede brugere og registrerede klienter. Ressource-serveren giver ressource-ejeren adgang til at administrerer adgang til disse ressourcer.

Autentifikations service: En service der kan identificere og autentificere en bruger. Denne service kan implementeres som en separat services, men vil ofte blive implementeret

sammen med OAuth servicen f.eks. med en OpenID Connect server, og bliver da også ofte fejlagtigt opfattet som samme service og som en del af OAuth standarten.

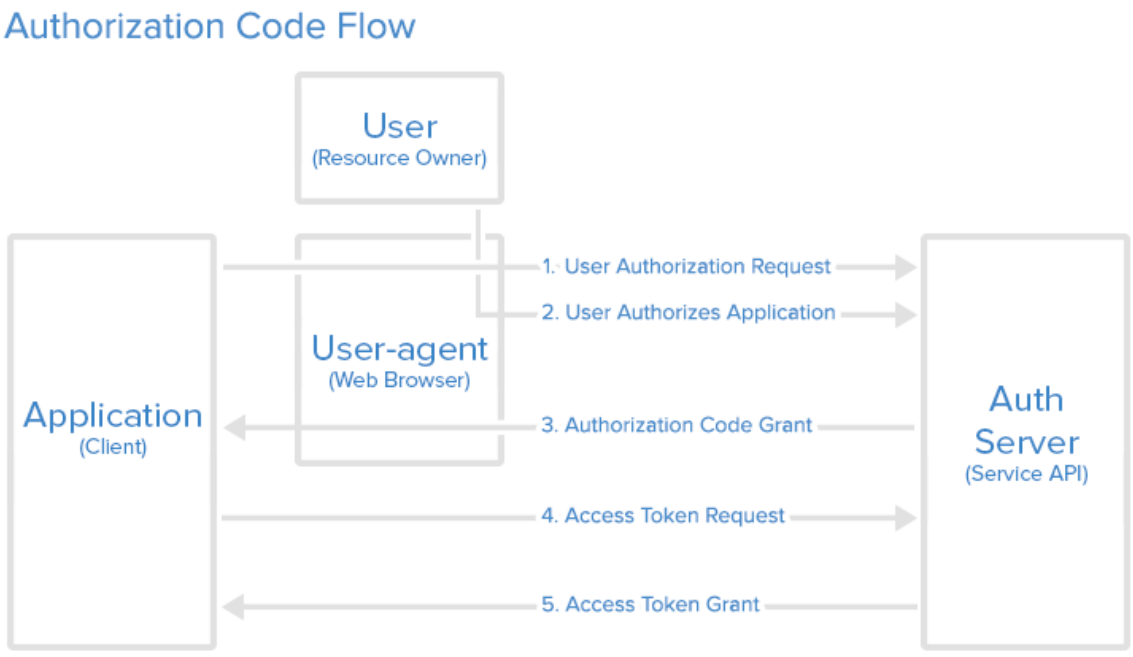
2.3 Autentifikationsflow i OAUTH 2.0

Authentifikationstyper, der understøttes af OAuth 2.0 ([2], [13] og [14])

Authorisation Code flow – det mest anvendte autentifikations-flow, som typisk bruges i forbindelse med serverbaserede web- og mobilapplikationer. Flowet består i at browseren omdirrigeres til autentificerins-servicen, hvor brugen logger ind samt giver tilladelse til at klienten må anvende brugerens identitetsdata. Herefter omdirigeres browseren tilbage til klientapplikationen, sammen med en identitets-token, der overføres via en back-channel til klientapplikationens server. Dette flow giver den bedst mulige sikkerhed, fordi identitets-tokenen ikke er tilgængelig i selve browseren og klientens autorisation samtidig bliver valideret (se figur 1).

Implicit flow – anvendes primærrt til SPA applikationer, der udelukkende afvikles i browseren med JavaScript og ikke har en server backend. Identitets-tokenen overføres direkte til browseren i forbindelse med omdirigeringen fra autentificeringsserveren.

Hybrid flow – bliver sjældent brugt, men tillader en kombination af Authorisation Code flow og Implicit flow.



Figur 1: Principskitse for Authorisation Code flow (fra [13])

3. Analyse og design

Formålet med projektet er at arbejde med en authentificerings-service, der kan bruges til centraliseret autentificering og autorisation af brugere og klienter i en servicebaseret arkitektur. Projektet skal opfattes som POC for et evt. senere udviklingsprojekt, og skal dermed udelukkende kunne demonstrerer principperne for centraliseret autentificering og autorisation, med udgangspunkt i OAuth og Open Identity som beskrevet i kapitel 2.

Nedenstående analyse tager udgangspunkt i udvikling den endelige authentificerings-service, men omfanget af sådan et udviklingsprojekt gør, at målet med dette projekt afgrænses til at være et indledende POC projekt. Rent metodemæssigt ville dette under alle omstændigheder være første skridt i et større projekt, hvor man arbejder efter den agile projektform, og hvor erfaringerne fra dette projekt ville kunne bruges til at detailplanlægge de næste faser i det overordnede projekt.

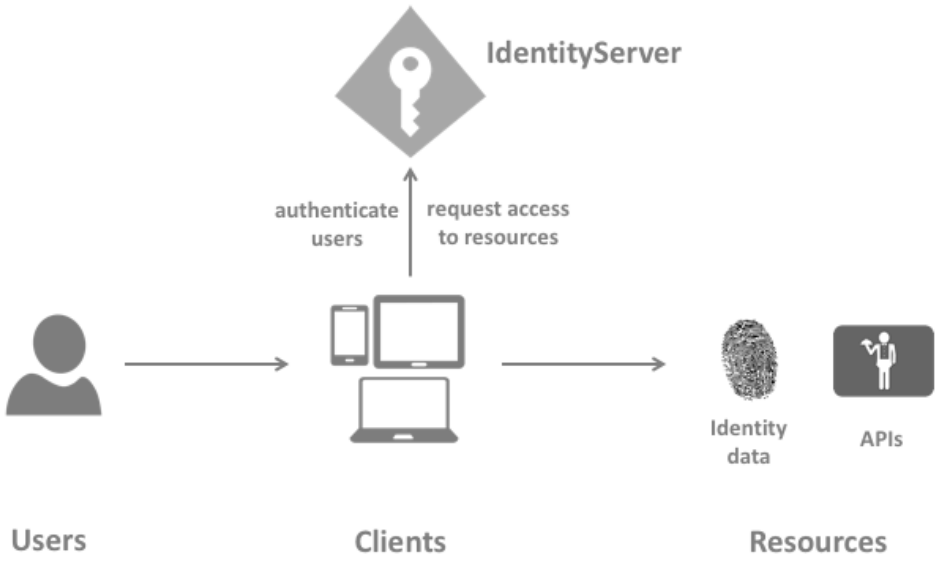
3.1 Overordnet koncept og system arkitektur

Systemarkitekturen er tænkt som en række uafhængige klientapplikationer, som brugere kan anvende til at tilgå en række uafhængige webservices (ressourcer). Centralt findes autentificeringsservicen (Identityserver), hvor brugere kan identificere sig (logge på med brugernavn og password) for således at opnå adgang til de ønskede ressourcer via klientapplikationerne. Systemadministratorer kan styre adgangen til ressourcerne - både brugernes adgang til ressourcerne samt hvilke klient-applikationer, der har adgang til hvilke ressourcer.

En principskitse for systemarkitekturen er vist på figur 2. På figuren fremgår det at "Identity data"opfattes som en ressource. Identitets data vil i dette POC projekt være koblet til autentificerings-serveren, og ressourceejeren vil være brugeren selv, hvor de øvrige ressourcer vil bestå af f.eks. uafhængige webservices.

3.2 Kravspecifikation

Der skal udvikles en autentificerings- og autoriseringsservice - herefter kaldet **Bouncer** (*Dørmand*) - der skal implementerer OAuth 2.0 og OpenID Connect som autentificerings- og autoriseringsprotekoller. Servicen skal kunne autentificere brugere og kontrollere adgangen til services, som udstilles via en klient-applikation. Servicen skal have et administrationsinterface, hvorigennem administration af brugere og klienter samt disses rettigheder til ressourcer kan styres.
Arkitekturen på min arbejdsplads er baseret på Miscrosoft platformen og servicen skal derfor programmeres i ASP.NET Core. Persistent datalagring skal implementeres hva. EntityFramework Core med datalagring i Microsoft SQL Server.



Figur 2: Principskitse for systemarkitektur ved centraliseret autentifceringsservice (fra [10])

Core-versionen af .NET framework'et er under kraftig udvikling og virker stadig ret ustabil, med hyppige ændringer og mange fejlretninger. Visse centrale koncepter er endda ændret væsentligt i forhold til hovedversionen af .NET, men Core versionen er valgt, da denne version vurderes at være den mest fremtidssikre løsning, som Microsoft vil satse på fremadrettet.

3.3 Domænemodel

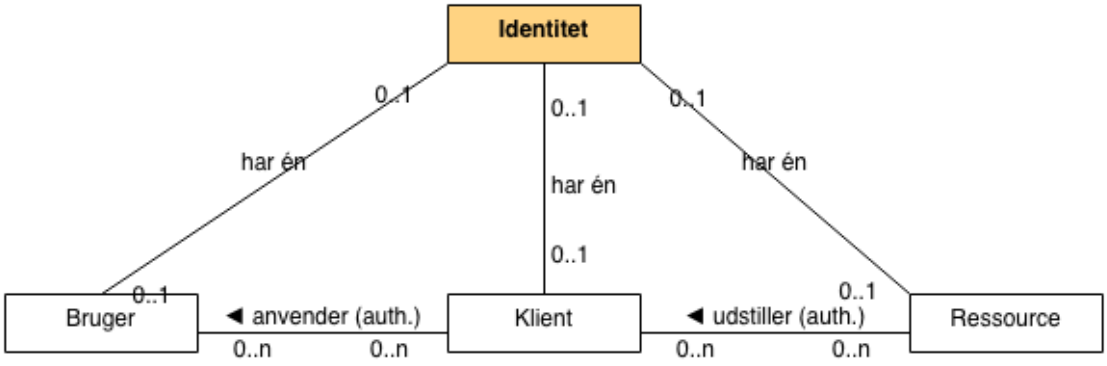
Domænemodellen er vist på figur 3.

Ud fra studierne i afsnit 2 og figur 2, kan domænemodellen betragtes som 3 konceptuelle klasser / entiteter - en bruger, en klient og en ressource, samt en fjerde klasse / entitet, som repræsenterer identitets- og OAuth-servicen. Hver af de 3 førstnævnte klasser har en identitet, som de kan hente fra OAuth servicen, og derved identificerer sig selv over for de øvrige klasser / entiteter i systemet.

Brugere kan anvende klienter f.eks. via en webbrowser eller en mobiltelefon, og via klienten kan brugeren tilgå data fra en ressource (f.eks. en webservice), hvis denne udstilles via klienten. Det er dog en forudsætning, at både klienten og brugeren er autoriseret til at tilgå ressourcen.

3.4 Use Cases

Nedenfor beskrives 5 usecases, som beskriver de vigtigste funktioner i autentifikationsservicen. Usecase 2,3,4 og 5 skal opfattes som CRUD-funktionalitet, men er for overblikkets skyld her kun beskrever som CREATE-funktionalitet. Projektet her sigter efter at implementerer og demonstrerer Usecase 1 som POC, men i et system, hvor fundamentet for resten af funktionaliteten er tilstede. Usecase-diagram er vist på figur 4



Figur 3: Domænemodel

Usecase 1: Kendt bruger tilgår data fra en ressource via en klientapplikation

Brugeren åbner en klient-applikation via sin browser, og navigere til en side i klient applikationen, der vis data fra en webservice, der er beskyttet af Bouncer. Browseren omdirigeres nu til Bouncers loginside, hvor brugeren logger ind med brugernavn og password. browseren omdirigeres nu til en bekræftelsesside, hvor brugeren skal godkende at klientapplikationen må anvende brugerens identitetsdata.

Brugeren klikker på *Godkend* og browseren bliver derefter omdirigeret tilbage til den ønskede siden i klient-applikation. På siden kan brugeren nu se data fra webservicen.

Usecase 2: Ukendt bruger registrerer sig via en klientapplikation

Brugeren åbner en klient-applikation via sin browser, og navigere til en side i klient applikationen, der vis data fra en webservice, der er beskyttet af Bouncer. Browseren omdirigeres nu til Bouncers loginside, hvor brugeren kan logge ind med brugernavn og password.

Brugeren klikker på linket Registrer ny bruger, hvorved der fremkommer en formular til brugerregistrering. Brugeren udfylder data og klikker på knappen *Send*. Brugeren omdirigeres nu til en side, der bekræfter at registreringen er send og afventer godkendelse fra administrator.

Brugeren klikker på knappen *Tilbage* og browseren bliver nu omdirigeret tilbage til klientapplikationens startside.

Usecase 3: Administrator autoriserer bruger til ressource via administrationsmodul

Administratoren åbner Bouncer og logger ind med brugernavn og password. I bouncer åbnes menuen Administration, og undermenuen Brugere. Her vises nu en liste mad alle registrerede brugere. Administratoren klikker på på Mangler godkendelse, hvorved listen med brugere reduceres til dem der afventer godkeldelse.

Administratoren klikker på en af brugerne, hvorved en formular åbnes. I formularen vises en liste med claims for den pågældende bruger, herunder den klientapplikation, som brugeren ønsker adgang til. Administratoren sætter et flueben ud for den ønskede klien-

tapplikation, og klikker *Gem*, hvorved browseren omdirigeres tilbage til listen med brugere.

Usecase 4: Administrator registrerer ny ressource via administrationsmodul

Administratoren åbner Bouncer og logger ind med brugernavn og password. I bouncer åbnes menuen Administration, og undermenuen Ressourcer. Her vises nu en liste mad alle registrerede ressourcer. Administratoren klikker på på Tilføj ny ressource, hvorved en formular åbnes.

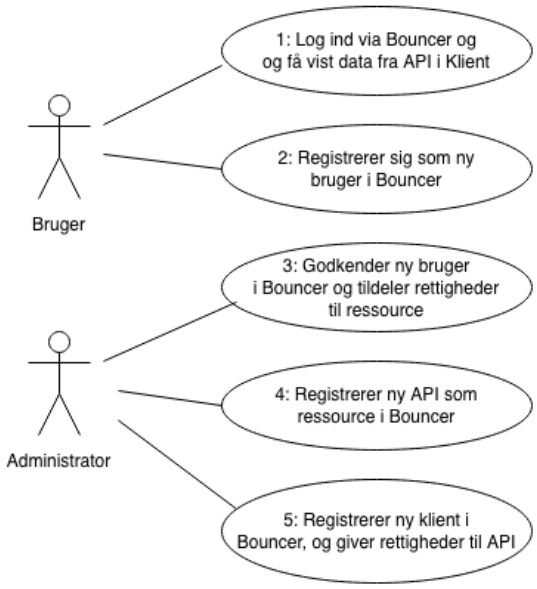
Administratoren udfylder et sigende navn for ressourcen, samt hvilken basis-url at ressourcen befinder sig på. administratoren klikker på *Gem*, hvorved der genereres en unik id til ressourcen samt en unik nøgle, der skal indsættes i klientapplikationen

Usecase 5: Administrator registrerer ny klientapplikation via administrationsmodul og autoriserer denne til at tilgå ressource

Administratoren åbner Bouncer og logger ind med brugernavn og password. I Bouncer åbnes menuen Administration, og undermenuen Klienter. Her vises nu en liste mad alle registrerede klienter. Administratoren klikker på Tilføj ny klient, hvorved en formular åbnes.

Administratoren udfylder et sigende navn for klienten, indtaster hvilke url der skal omdirigeres til når brugeren logge ind og logge ud samt tilføjer "api1"i listen over ressourcer, der skal gives adgang til.

Administrator klikker på knappen *Gem*, hvorved der genereres en unik id til klienten samt en unik nøgle, der skal indsættes i klientapplikationen.



Figur 4: Use Case diagram

3.5 IdentityServer4

IdentityServer4 er et gratis, open source framework, der bruges til at implementerer OpenID Connect og OAuth 2.0 i en ASP.NET Core 2 applikation. Frameworket er udviklet af Brock Allen og Dominick Baier, er licenseret under *Apache 2* og er en del af .NET Foundation. IdentityServer4 er en officielt certificeret implementering af OpenID Connect standarten [10].

IdentityServer4 stiller følgende funktionalitet til rådighed:

Autentificering som en Service : centraliseret login og bruger-godkendelses for mange applikationstyper, herunder (web, native, mobile, services), herunder delt *Single Sign-on / Sign-out* for alle organisationens applikationer.

Adgangskontrol for webservices / API'er : leverer *access tokens* til API'er fra klienter, der skal kunne tilgå og udstille data, både *server-to-server*, web-applikationer, *Single Page Applications ("SPA")* og native/mobile apps.

Federation Gateway : Understøtter eksterne identiteter som f.eks. *Azure Active Directory*, *Google*, *Facebook* m.m.

IdentityServer4 stiller endvidere en række demonstrations-projekter til rådighed, som demonstrerer hvordan framwork'et kan anvendes sammen med Microsofts *.NET Core Identity* og *EntityFramework Core*. IdentityServer4 lægger vægt på fleksibilitet og muligheden for at kunne konfigurerer servicen efter egne behov, og vurdere derfor at være optimal til et POC projekt og evt. den videre udvikling.

3.6 Udviklingsplan

Den overordnede udviklingsplan er følgende:

1. POC: Der skal programmeres tre applikationer / services, der kan demonstrerer et autentificerings-flow mellem bruger, klient og ressource, som vist i den konceptuelle model på figur 2:

    - Bouncer: en autentificerings- og autoriseringsservicen der laves med udgangspunkt i de tilgængelige demoprojekter fra IdentityServer4. Servicen skal både implementerer basal brugergodkendelse med log ind og log ud, samt have persistent datalagring i en SQL Server, for dermed at demonstrere de grundlæggende forudsætninger for at kunne programmere det endelige system.
    - Api1: En REST API webservice, der udstiller et simpelt datasæt, og som beskyttes af Bouncer
    - GUIClient: en klient-applikation, der udstiller data fra Api1 som en beskyttet ressource, og som lader Bouncer autentificere brugere, der ønsker at tilgå beskyttede ressourcer.

2. Med udgangspunkt i erfaringerne fra POC projektet, laves en detaljeret analyse af organisationens behov og rammer i form af platform og intern sikkerhedspolitik. Herudfra designes den endelige service, med kravspecifikation og usecases.

3. Omprogrammering af Bouncer på baggrund af analysen. Bouncer skal her kun opnå samme basis-funktionalitet som der blev opnået i POC projektet, men skal være tilrettet de endelige kravspecifikationer og usecases. Test af basalt autentifikationsflow der opfylder organisationen behov.

4. Programmering af administrations-interface, hvor administratorer kan oprette, opliste, opdaterer og slette brugere, klienter og ressourcer i systemet, samt administrerer rettigheder mellem brugere, klienter og ressourcer.

5. Test af systemet, herunder accepttest samt evt. tilretning af funktionalitet.

6. Implementering

Udviklingen skal gennemføres som et agilt projekt, hvor der for hver trin i projektplanen planlægges en række sprint, tests og iterationer, der kan lede projektet mod målet. Løbende brugerinvolvering, delmål og tests skal sikre, at projektet styres i den rigtige retning.

der laves en overordnet forventelig tidsplan efter POC projektet, men disse tilrettes løbende som et led i den agile projektstyring. Der laves kun bindende detailtidsplaner for det aktuelle sprint.

Detailplan for POC projektet

Mine forudsætninge for at gennemføre POC-projektet er, at jeg har arbejdet med .NET og C# på mine 2 foregående kurser, men jeg har ikke tidligere arbejdet med ASP.NET og heller ikke med .NET Core. En del af POC-projektet vil derfor også være at blive fortrolig med ASP.NET Core.

Trin1: Basal MVC applikation Der opbygges en basal ASP.NET Core MVC applikation med udgangspunkt i en simpel konsol-applikation. Formålet med trin 1 er at forstå hvordan de basale funktioner i en ASP.NET Core web applikation implementeres og fungerer.

Målet med trin 1 er at have en fungerende MVC applikation, der i funktionalitet svare til dotnet's MVC projektskabelon.

Trin 2: Persisten datalagring med EntityFramework I trin 2 arbejdes med Entity Framework Core (EFC), lagring af data i en SQL Server database og basal adgangskontrol. EFC implementeres i en ny MVC projektskabelon og der konfigureres, så den understøtter adgangskontrol. Der opbygges en objektmodel til at håndterer brugere. Formålet med trin 2 er at lære at implementerer basal CRUD via EFC, samt forså hvordan ASP.NET Core understøtter basal adgangskontrol. Der arbejdes med "code first"tilgangen.

Målet er, at jeg efter trin 2 har en fungerende ASP.NET Core MVC applikation, der kan oprette, læse, opdatere og slette brugere i en database, samt styre disses adgang til forskellige dele af applikationen.

Trin 3: IdentityServer4 I trin 3 implementeres IdentityServer4 (IS4) i en ASP.NET Core MVC applikation med udgangspunkt i den basale Quickstarts på IS4's hjemmeside, med "in-memory"datalagring. Gennem de valgte Quickstart-guider skal forretningslogikken bag IS4, objektmodellen samt IS4's plads i en moderne applikationsarkitektur undersøges.

Formålet med trin 3 er at få en forståelse for, hvordan Autentificering med OAUTH 2.0 og Open ID fungerer og hvordan IS4 implementerer dette.

Trin 4: Opstart af Bouncer Opstart af applikationen Bouncer. Der tages udgangspunkt i erfaringer fra Sprint 1, 2 og 3, således at der oprettes en ny ASP.NET Core MVC applikation, og heri implementeres og konfigureres IS4. Der etableres brugerstyringsfunktinonalitet med udgangspunkt i trin 2 vha. ASP.NET Core Identity middleware,

og persistent datalagring i en SQL Server. Bouncer konfugureres med en klient og en api-ressource, som klienten har rettigheder (claims) til. Klient og ressouce bibeholdes som "In-memory"objekter.

Formålet med trin 4 er at etablerer fundametet for Bouncer og at starte på at implementerer den ønskede forretningslogik samt identificerer evt. problematikker i forhold til dette. Efter sprint 4 er den fulde datamodel tilgængelig på SQL serveren, men kun systemets bruger-objekterne er koblet på denne.

Trin 5: Autentificerings-flow I dette trin programmeres en ressource (*Api1*) og en klient-applikation (*GUIClient*) med udgangspunkt i .NET's projektskabeloner for en webapplikation og en webapi. OpenID Connect implementeres som autentificeringsmekanisme i disse programmer og i klient-applikationen oprettes en controller, der henter data fra Api1, og viser dem i et view. Denne controller konfigureres så den kræver autentificering af brugeren.

Trin 6: Hele objekt-modellen flyttes til i SQL server Implementering af fuld persisten datalagring i SQL serveren, hvor både brugere, klienter og ressourcer kan lagres i databasen. Der oprettes en *Seed*-klasse, der kan oprette brugere, klienter og ressourcer i en tom database, og der oprettes basale CRUD-klasser for klienter og ressourcer. Test af ekstension af bruger-, klient. og ressource-klasserne, så udvidelse af den basale objekt-model opnås.

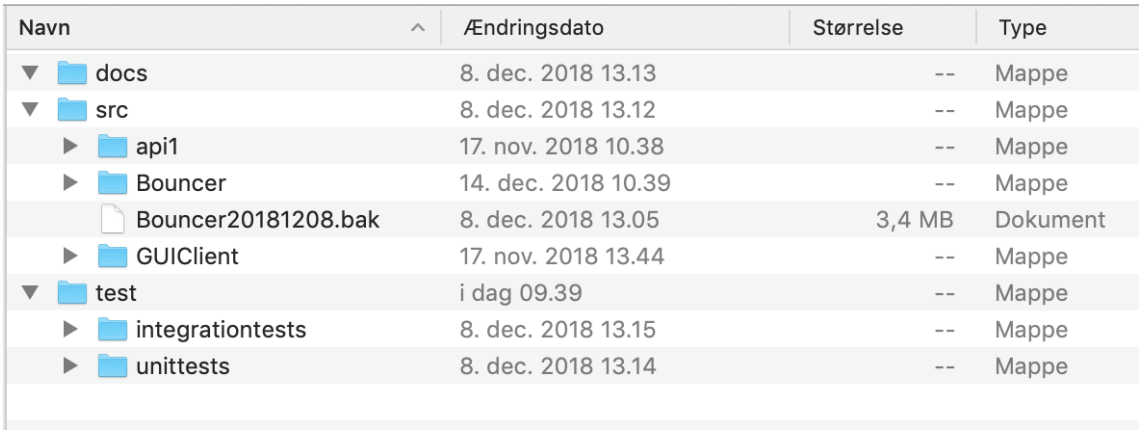
4. Bemærkninger til kodens indhold og opbygning

I POC-projektet, har jeg gennemført trin 1 - 5 (se afsnit 3, mens trin 6 ikke kunne nås inden for den tilgængelige tid. Med trin 5 er det lykkedes at få et autentificerings-flow mellem de 3 applikationer / services til at fungere. Samtidig er der opnået delvis persistent datalagring, ved at brugerdata lagres i SQL-serveren.

4.1 Projektstrukturen

Koden er vedlagt i afleveringen sammen med denne rapport, og brugervejledning / dokumentation findes i bilag B. Projektetstrukturen bestå af 3 mapper i roden af projektmappen: docs/ (beregnet til dokumentation), src/ (indeholder kildekoden) og test/ (beregnet til unit- og integrationstest)

Mappen src/ indeholder projektmapper til de 3 applikationer (Bouncer, Api1, og GUIClient) (se figur 5), samt en backup af SQL Server databasen, som kan restores og hvor datamodellen for IdentityServer4 kan undersøges nærmere (Se bilag C).



Figur 5: Mappestruktur i projektmappen

4.2 Bouncer

I bilag D er udvalgte klasser fra Bouncer vist. Udgangspunktet for applikationen er en MVC projekt-skabelon, hvor jeg med NuGet installere IdentityServer4 (IS4). Måden at IS4 bliver implementeret i applikationen er, at man i klassen *Startup* tilføjer IS4 til applikationens services samt konfigurere IS4 i metoden *ConfigureServices*, og i metoden *Confugure* tilføjer IS4 til applikationens *HTTP request pipeline*.

I bilag D i klassen Startup på linje 35-45, kan det ses at IS4 konfigureres med AddAspNetIdentity, som er de autentificerede brugere, og at disse hentes fra klassen *ApplicationUser*, der er en ekstension af klasse IdentityUser. Ressourcer og klienter konfigureres som in-memory objekter, der injeceres med metoderne *GetApis()* og *GetClients()* fra klassen Config.

Det ses også på linje 26 - 31 at applikationen konfugureres med EntityFramework samt ASP.NET Core Identity, og at Identity-servicesn baseres på ApplicationUser samt persistent lagring vha. EntityFramework gennem klassen *ApplicationDbContext*. Herved bliver standart brugestyring med bl.a. login og logout tilgængelig i applikationen.

Klassen *Config* er også vist i bilag D.

4.3 Api1

I bilag E er udvalgte klasser fra Api1 vist. Udgangspunktet for Api1 er en projekt-skabelon for en REST API. Skabelonen kan som udgangspunkt returnerer nogle "dummy-værdier, når der forspørges på adressen /api/values, og det er disse dummy-værdier jeg har brugt i POC-projektet. Servicen er ikke udviklet yderligere, men overload-metoden, hvor der forspørges på adressen /api/values/[int] er dog ændret, så denne også returnerer et json-objekt.

Konfigurationen af Api1 sker i klassen *Sartup* på linje 32 - 39, hvor der i metoden *ConfigureServices* tilføjes *AddIdentityServerAuthentication()* til Authentication-servicen. Her sættes Bouncer som "authority"(localhost:5000), og Api1 identificerer sig selv som "api1". Denne identifikation kan genfindes i Bouncer's Config-klasse.

Api1's ValueController klasse er også vist i bilag E, og her ses det at Controlleren er beskyttet med nogleordet [Authorize], hvilket betyder at brugere og klineter skal være godkendt af Bouncer, for at kunne tilgå ressourcen.

4.4 GUIClient

I bilag F er udvalgte klasser fra GUIClient vist. Der er taget udgangspunkt i en MVC skabelon, der også er konfugureret til at anvende autentificering, men konfugurationen her er lidt mere omfattene (se indsatte kommentarer i klassen *Startup*). Applikationen konfigureres til at anvende OpenID Connect som autentificerings-metode, og at anvende en cookie for at opnå "statefull"tilstand.

Applikationen konfigureres også til at anvende Bouncer som "autoritet"(linje 46) og identifikation af sig selv vha. et KlientId og et password (linje 50 og 51). Endelig sætte Api1 som Scope, hvorved Applikationen oplyser Bouncer at den ønsker adgang til denne ressource.

I Klientapplikaitonen har jeg tilføjet en ny Controller (ApiController), der har en metode / route *Values()* samt en overloadet metode *Values(id)*, der begge laver et GET request til Api1, og sender det returnerede json-objekt fra Api1 ind i et View og returnere Viewet. Som det ses er begge metoder beskyttet vha. nøgleordet [Authorize], hvilket betyder at brugere skal være logget ind for at kunne tilgå metoden / viewet.

5. Test

5.1 Test af applikationer og autentificerings-flow

Udviklingsplan for POC-projektet er beskrevet i afsnit 3, og hvert trin i planen er gennemført som sprint-forløb i weekender. Hele ideen med dette har været, at nedbryde projektet i små veldefinerede "bidder", med et specifikt mål for øje.

Som led i de enkelte sprints, har der derfor været gennemført accept-tests, med henblik på at kunne vurderer, om målet for et trin / sprint er nået. Der har i POC-projektet ikke været arbejdet med unit- og integrationstests, fordi den ønskede funktion i høj grad er opnået gennem konfiguration af den valgte middleware, og jeg ikke selv har skulle programmere særlig meget af den centrale funktionalitet.

At basere funktionaliteten i sin applikation på middleware udelukker ikke unit- og integrationstests, og i det videre udviklingsforløb forventer jeg da også at disse testmetoder skal inddrages. Mappestrukturen i min projektmappe, som er uploadet sammen med rapporten, er derfor også klargjort til tests (se figur 5). Men da POC-projektet primært har fokuseret på at demonstrerer et koncept som læringsmål, og ikke et system der skal sættes i drift, er unit- og integrationstest altså fravalgt her.

Inspektion af autentificerings-flow med Wireshark

Som supplement til de gennemførte accept-tests, har jeg undersøgt kommunikationen i forbindelse med det opnåede autentificerings-flow mellem de 3 systemer med Wireshark (se bilag G) - dels med henblik på at kunne forstå hvad der sker, dels med henblik på at sikre / dokumenterer, at alle systemerne er involveret i flowet.

Bilag G viser et capture på *Loopback interfacet (127.0.0.1)*, hvor der er sat filter på port 5000 (Bouncer), 5001 (Api1) og 5002 (GUIClient). Kommunikation med port 1433 (SQL Server) er ikke medtaget, selv om dette også er en del af systemet og findes i capture-file. Capture-filen er også vedlagt i projektmappen under mappen test/

Autentificerings-flowet svarer til det der er beskrevet under brugervejledningen i bilag B og starter med et GET request på http://localhost:5002/values, som er beskyttet. Centrale elementer af autentificerings-flowet kan identificeres på følgende linjer i capture-filen:

- Linje 47: brugeren sender en GET request til klient-applikationen på http://localhost:5002/values.
- Linje 89: klientapplikationen svarer med statuskode 302, som er en URL redirect til Bouncer (http://localhost:5000/connect/authorize)
- (I den mellemliggende kommunikationssekvens kan jeg ikke finde selve login eventet, hvor der sendes en POST request fra browseren til Bouncer med brugernavn og password)

- Linje 783: brugeren er nu logget ind / godkendt og er blevet ført tilbage til klient-applikationens /signin-oicd end-point med en OpenIdConnect token, som stemples ned i en cookie.
- Linje 836: sker en ny omdirrigering tilbage til klientapplikationens http://localhost:5002/values, som der oprindelig blev lavet en GET request på
- Linje 838: en ny GET request sendes til http://localhost:5002/values
- Linje 844: klientapplikationen sender et GET request til API'et på http://localhost:5001/api/values. Request'et indeholder nu en token-streng, som bruger og klient kan valideres med. Token-strengen kan ses i wiresharks detaljevisning.
- Linje 872: Da både bruger og klient-applikationen er autentificeret og autoriseret via token, sender API'et et OK response (statuskode 200) tilbage klient-applikationen, indeholdende værdierne som json-objekt.
- Linje 876: Klientapplikationen sender OK response til browseren, der returneren html-side, der bl.a. indeholder værdierne.
- Linje 1048: Der sendes et nyt GET request med token fra browser til klient-applikationen på URL http://localhost:5002/values/5.
- Linje 1054: Klient-applikationen sender en GET Request til API'et på http://localhost:5001/api/values/
- Linje 1058: API'et sender OK response til klientapplikationen indeholdende et json-objekt
- Linje 1061: Klient-applikationen sender OK response til browseren indeholdende en html-side bl.a. med værdierne fra API'et

Det man kan bemærke er, at ved den indledende autentificering, skete der rigtig meget kommunikation til / fra Bouncer (linjerne mellem 89 og 783), men efterfølgende - når brugeren er autentificeret og token-strengen med brugerens identitet kan sendes med i et request - kræves der relativ lidt kommunikation. Når klient-applikationen førstegang kontakter API'et, sker der lidt kommunikation med Bouncer, hvor klienten modtager et access-token til API'et, men når klient-applikationen anden gang sender et request til API'et på linje 1054, bliver bouncer (port 5000) ikke kontaktet, for token-strengen viser API'et at bruger og klient allerede er autoriseret.

Tokenstrengen som kan findes i capture-filen med wiresharks detaljevisning, og som kan inspiceres f.eks. med debugger-værktøjet på https://jwt.io:

****
****
****
****
****
****
****
****
****
****
DH5y **********
****

6. Sammenfatning og konklusion

6.1 Sammenfatning

Jeg har i dette projekt arbejdet med centraliseret autentificering og autorisation i en service-baseret arkitektur. Jeg har gennemført et POC-projekt, hvor jeg har programmeret en autentificerings- og autorisationsservice kaldet *Bouncer*, der vha. IdentityServer4 implementerer OAuth 2.0 og OpenID Connect standarterne. Jeg har desuden programmeret en klient-applikation og en REST API, som er konfigureret til at anvende Bouncer til adgangskontrol, og har med disse tre applikationer demonstreret et basalt autentificerings-flow.

Alle tre applikationer er programmeret i ASP.NET Core, og jeg har fået etableret konsistent datalagring for dele af objektmodellen vha. Entity Framework Core. På grund af manglende tid, er dele af objektmodellen i Bouncer stadig konfigureret som in-memory data, men hele objektmodellen er dog koblet til databasen gennem EntityFramework. Applikationerne er i høj grad baseret på de standart-skabeloner, som følger med .NET Core, ASP.NET Identity og IdentityServer4. Jeg har således ikke haft fokus på at programmere applikationer og funktionalitet fra bunden, men har i stedet valgt at fokusere på at arbejde med konfigurationen af de 3 applikationer, så de kan "spille"sammen og derved demonstrerer et autentificerings-flow.

Jeg har dog gennem projektet arbejdet en del med ASP.NET Core, og fået en god indsigt i hvordan dette framework bruges, herunder hvordan ASP.NET MVC, Razor-pages og EntityFramework Core grundlæggende fungerer. Jeg føler derfor at jeg med projektet har fået et godt fundament for at komme videre med udvikling af moderne web-applikationer og REST API'er. Dette har for mig også været et vigtigt læringsmål med dette projekt.

6.2 Konklusion og perspektivering

Jeg arbejder i Region Hovedstaden, og min afdeling er i øjeblikket i gang med at flytte dele vores systemer til Azure. Dette vil give os mulighed for i højere grad at dele data, applikationer og services med eksterne brugere f.eks. rådgivere der løser opgaver for os. Sikkerheden i Azure er som udgangspunkt baseret på Azure AD (OAuth), men dette kommer til at være styret centralt, og decentralt kommer vi ikke til at få adgang til at kunne oprette eksterne bruger i Azure AD. Projektet her har derfor givet mig en vigtig viden og erfaring i forhold til at finde en alternativ løsning på adgangskontrol til afdelingens kommende web-applikationer og -ressourcer.

Projektet her har vist, at det - med en relativ lille indsats - er muligt at programmere en autentificerings-service i ASP.NET Core, der hva. IdentityServer4 implementere OAuth 2.0 og OpenID Connect. Da IdentityServer4 også understøtter *Federation Gateway*, vil det let kunne spille sammen med f.eks. Azure AD. Ved at benytte sig af middleware komponen-

ter som f.eks. IdentityServer4 og ASP.NET Identity, kan man som udvikler implementerer kompleks funktionalitet på en sikker måde i sine applikationer, idet denne funktionalitet er udviklet af eksperter og som derfor garanterer "best practice"på området.

Det er dog vigtigt at være opmærksom på, at middleware-løsninger ikke må bliver en sovepude i forhold til sikkerhed. Typisk giver disse middleware-komponenter mulighed for omfattende konfiguration, og man bør som udvikler have en grundlæggende forståelse for den teknologi man ønsker at implementerer, og samtidig kunne forholde sig til hvilken betydning konfigurationsmulighederne har i forhold til sikkerhed.

Litteratur

[1] Microsoft docs: Getting started with ef core on asp.net core. https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/, Year = 2018.

[2] Oauth 2.0 - oauth. https://oauth.net/2/.

[3] Oauth and oauth wrap: defeating the password anti-pattern. https://arstechnica.com/information-technology/2010/01/oauth-and-oauth-wrap-defeating-the-password-anti-pattern/, 2010.

[4] The http www-authenticate response header. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/WWW-Authenticate, 2012.

[5] RFC 6749: The oauth 2.0 authorization framework. https://www.rfc-editor.org/info/rfc6749, 2012.

[6] Web authentication methods explained. https://blog.risingstack.com/web-authentication-methods-explained, 2012.

[7] Openid connect core 1.0 specification. https://openid.net/specs/openid-connect-core-1_0.html, 2014.

[8] RFC 7235: Hypertext transfer protocol (http/1.1): Authentication. https://www.rfc-editor.org/info/rfc7235, 2014.

[9] Entity framework (ef) documentation. Microsoft Developer Network, 2016.

[10] Identityserver4 documentation. http://docs.identityserver.io/en/latest/index.html, 2016.

[11] Openid connect explained. https://connect2id.com/learn/openid-connect, 2016.

[12] Umletino. a free online uml tool for fast uml diagrams, 2018.

[13] Mitchell Anicas. An introduction to oauth 2. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2, 2014.

[14] Matthias Biehl. *OAuth 2.0: Getting Started in API Security.* API-University Press, 2015.

[15] Adam Freeman. *Pro ASP.NET Core MVC.* Apress, sixth edition edition, 2016.

[16] James F. Kuroso and Keith W. Ross. *Computer Networking. A Top-Down Approach.* Pearsons, seventh edition edition, 2017.

[17] Julia Miller, Rowan ; Lerman. *Programming Entity Framework: Code First.* O'Reilly Media, Inc., 2011.

[18] Robin Nixon. *Learning PHP, MySQL and JavaScript With JQuery, CSS and HTML5*. O'Reilly Media, fourth edition edition, 2015.

[19] Chris Sevilleja. The ins and outs of token based authentication. `https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication`, 2015.

[20] Andrew Troelsen and Philip Japikse. *C# 6.0 and the .NET 4.6 Framework*. Apress, 7th edition, 2015.

Bilag

A. Anvendte værktøjer

Applikationerne / systemet er udviklet ved brug af følgende værktøjer:

- .NET Core version 2.1 med middleware:
  - Entity Framework Core
  - ASP.NET Core Identity
  - IdentityServer4
- Visual Studio Code version 1.29
- SQL Operatons Studio version 0.32.9. NB! er ændret til *Azure Data Studio* (se `https://docs.microsoft.com/en-us/sql/azure-data-studio/what-is?view=sql-server-2017`)
- Docker for Mac version 18.09.0, build 4d60db4
  - SQL Server container image `https://hub.docker.com/r/microsoft/mssql-server/`
- UMLetino, A free online UML tool for fast UML diagrams [12].

Rapporten er skrevet med LaTeX med håndtering af referencer i BibTeX.

B. Brugervejledning

Systemkrav

Koden er udviklet i ASP.NET Core og kræver at .NET Core version 2.1 er installeret på computeren. Programmerne er udviklet som *proofe of concept (POC)* og er kun blevet testet i et udviklingsmiljø baseret på Microsofts *Kestrel*-server, som er standart HTTP server for ASP.NET Core.

Koden er udviklet og testet på en Macbook Pro 64 bit, men da .NET Core er cross-platform versionen af .NET, burde koden kunne køres på både Windows- og Linux-systemer, hvis disse har .NET Core version 2.1 installeret.

Persistent lagring af brugerdata sker i en Microsoft SQL Server 2017. Da udviklingen er sket på en Macbook, er har jeg anvendt Docker version 18.09.0 til at køre SQL server som en container.

Opsætning af database i Docker

Forudsætning for dette trin er, at Docker er installeret og er startet (se evt. online vejledning for hjælp til dette. I en bash-konsol eksekveres kommandoen:

```
sudo docker pull mcr.microsoft.com/mssql/server:2017-latest
```

... hvorved det nyeste docker image for Microsoft SQL Server hentes ned fra nettet. Eksekver nu kommandoen:

```
sudo docker run -e 'ACCEPT\_EULA=Y' -e 'SA_PASSWORD=<YourStrongPassw0rd>'
-p 1433:1433 --name mssql1 -d mcr.microsoft.com/mssql/server:2017-latest
```

... hvor <**YourStrongPassw0rd**> er udskiftet med et valgt administrator password til SQL Serveren.

Hvis containeren startes uden fejl, kan man nu logge på SQL Serveren via sit foretrukne RDBMS-værktøj med login: **sa** og det valgte administrator password. I dette projekt har jeg anvendt programmet *SQL Operations Studio* som RDBMS. Når der er opnået forbindelse til serveren, oprettes en ny database med navnet **Bouncer** og der oprettes en bruger i databasen, med login **Bouncer** og password **Bouncer_123**. Denne bruger tildeles database-rollen **db_owner**.

I en bash-konsol navigeres til mappen /src/Bouncer, og følgende kommando eksekveres

```
dotnet ef database update
```

Herved oprettes datamodellen i databasen og en testbruger oprettes.

Opstart af services / applikationer

Systemet består af 3 services / applikationer:

- Bouncer, der er OAuth- og authentifikationsserver. Applikationen har indbygget login / logout funktionalitet, der bruges til autentifikation af brugeren, og som andre systemer kan redirregere til, når deres brugere skal logge ind og identificere sig. Bouncer er sat op til at kører på localhost port 5000 (se figur 6).
- API1, der er en rest-api, som udstiller data. API1 er baseret på en almindelig ASP.NET Core MVC applikation, som udstiller rest-api'et vha. routing. API'et har kun én metode tilgængelig, og er blevet konfigureret, så denne metode er beskyttet af Bouncer. API1 er sat op til at køre på localhost port 5001 (se figur 7).
- GUIClient, der er klientapplikationen, hvorigennem brugeren ønsker at tilgå data i API1. Dette er også en ASP.NET Core MVC applikation, som vha. routing udstiller sider for brugeren. Siden *Kontakt* er blevet konfigureret, så den er beskyttet af Bouncer. Samtidig viser denne side data, som hentes fra API1. GUIClient er sat op til at køre på localhost port 5002 (Se figur 8).

Gå ind i hver af de 3 mapper under /src mappen fra kommandolinjen, og anvend kommandoen:

```
dotnet run
```

... hvilket starter hver af de 3 applikationer op. Start med *Bouncer*, hvorefter *Api1* startes og tilsidst startes *GUIClient*.

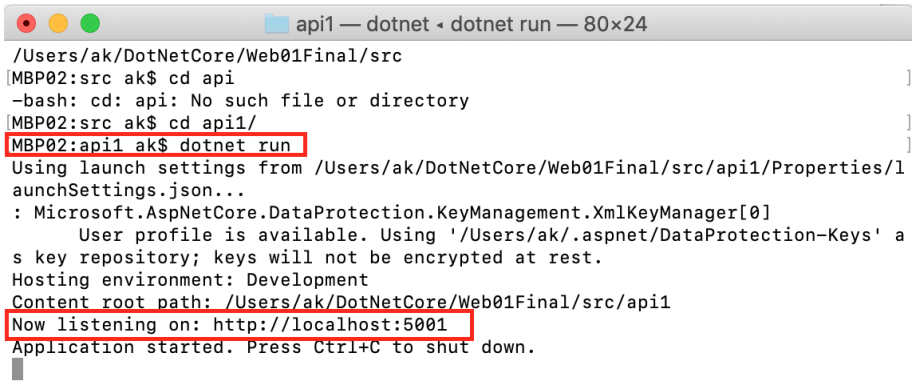
Test af om OAuth servicen kører

Åben en browser og naviger til `http://localhost:5000/.well-known/openid-configuration`. Dette er Bouncers *discovery endpoint*, som stilles til rådighed gennem IdentityServer4, og kan bruges af andre applikationer til at hente metadata om servicen. Skærmbilledet på figur 9 burde ses.

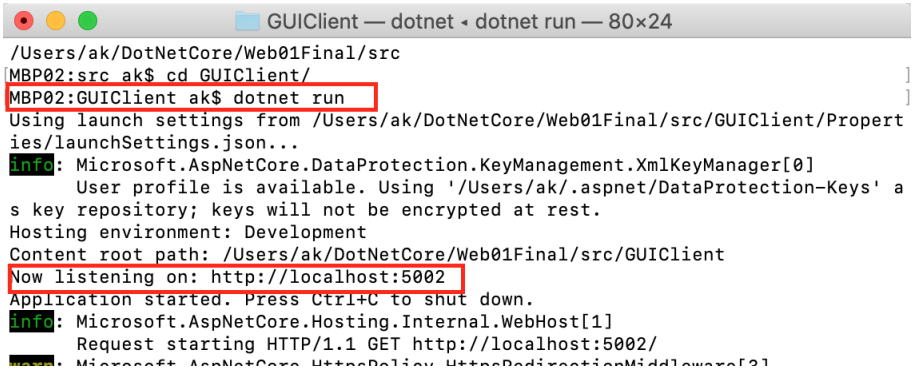
Test af autentifikations-flow

1. Når alle tre systemer er startet og det er testet om Bouncer kører, åbnes en browser og der navigeres til adressen `http://localhost:5002`. Klientapplikationens startbillede skal nu ses i browseren (se figur 10).
2. Naviger nu til menupunktet Hent værdier fra API , som er kræver at brugeren er logget ind. Browserens adresse bliver nu omdirigeret til Bouncer (endpoint: `http://localhost:5000`), og bliver ført til *Bouncers* login side.
3. Log ind med brugernavn: **test@mail.test** og password: ******* (se figur 11).
4. Der vises nu en *"consent"*-side, hvor brugere skal give tilladelse til at klient-applikationen må anvende de oplistede data. Dette foregår stadig i Bouncer (se figur 12). Accepter dette, hvorefter browserens adresse omdirigeres tilbage til `http://localhost:5002/values`.

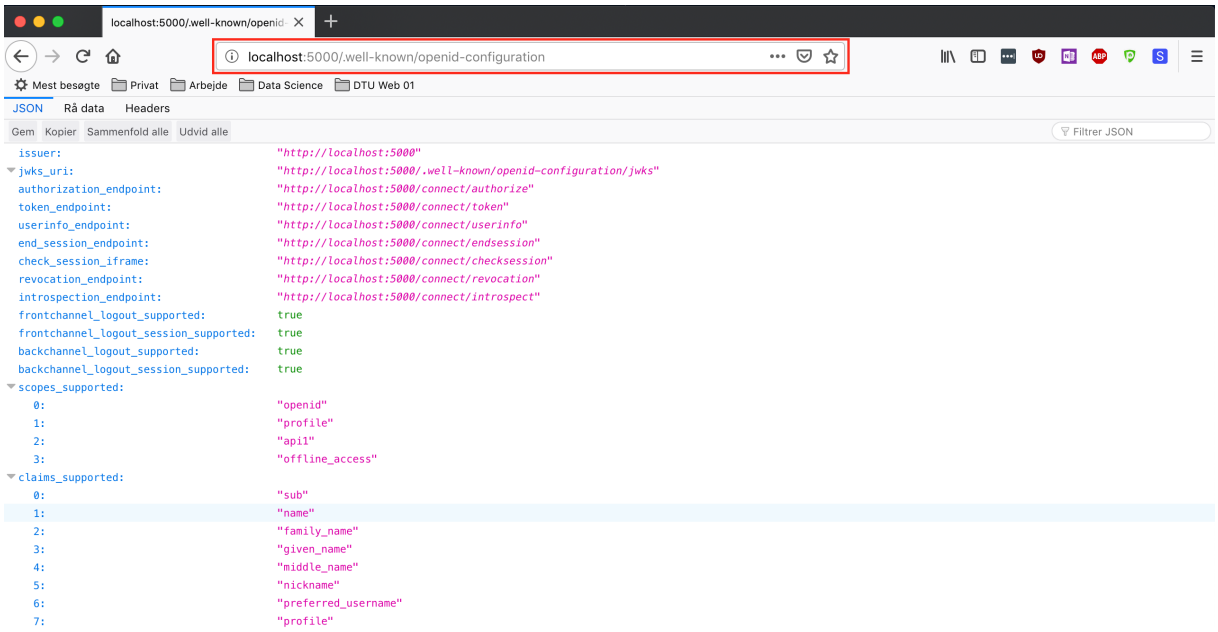
Figur 6: Opstart af Bouncer



Figur 7: Opstart af webservicen Api1



Figur 8: Opstart af klient-applikationen

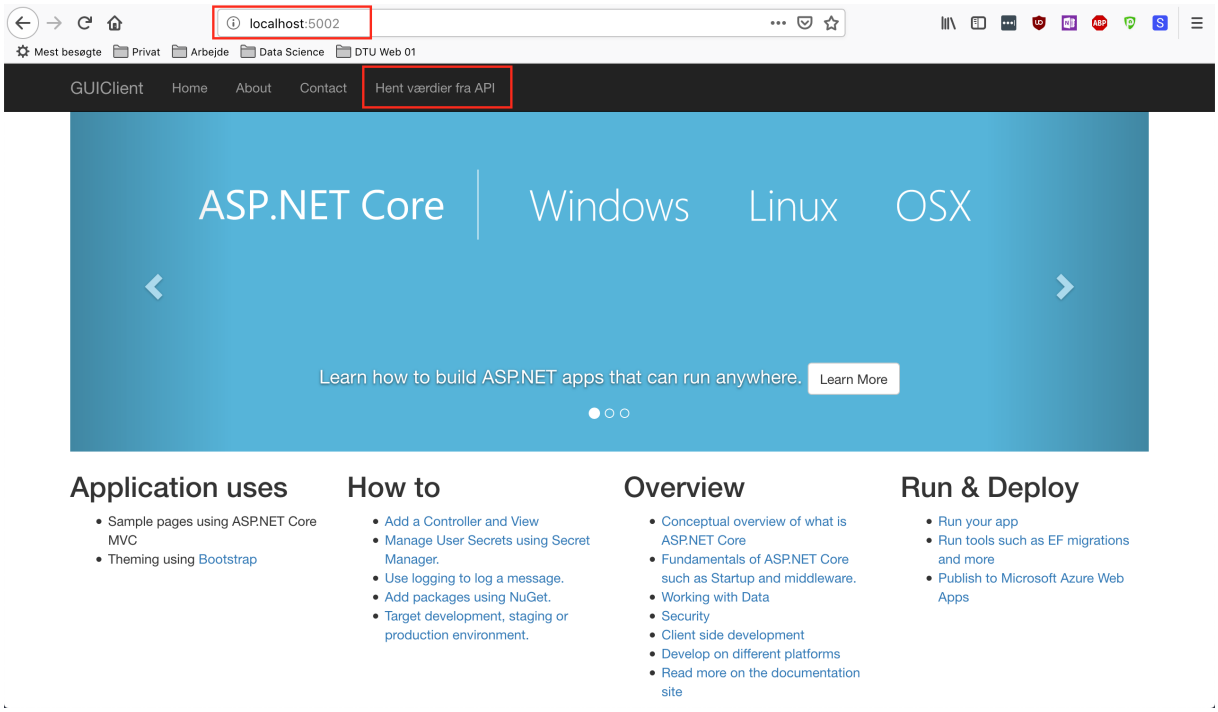


Figur 9: Test af om Bouncer kører

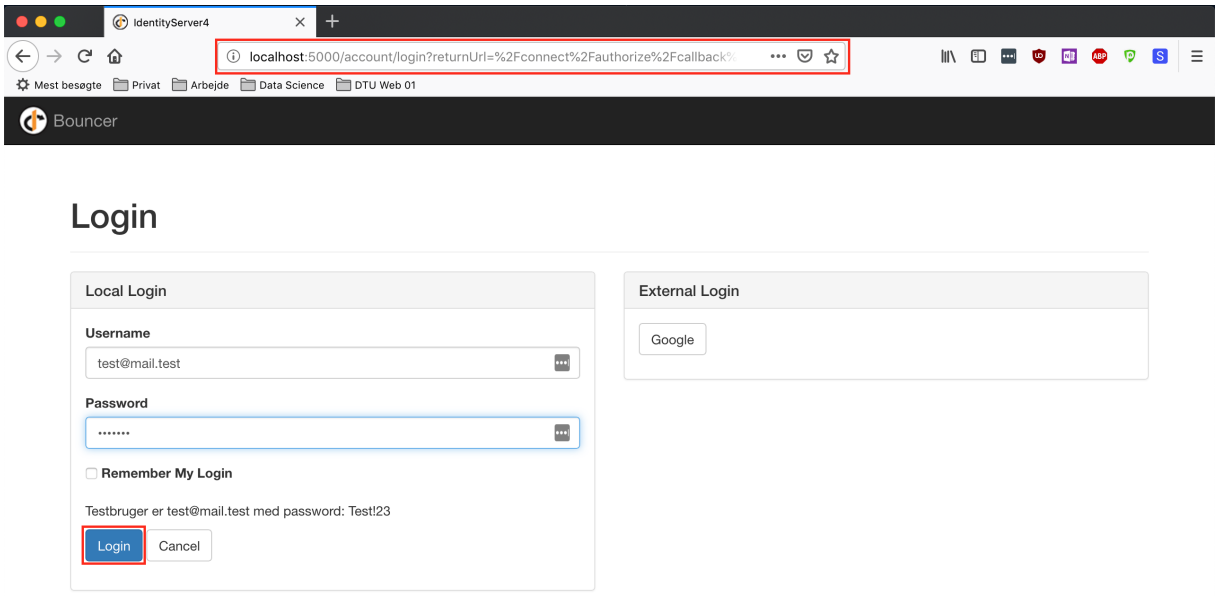
5. I browseren vises nu siden *Vis værdier fra API'et* på GUIClient, med json-data hentet fra API1 (["Value01","Value02"]). Herunder vises oplysninger om brugeren, som som stammer fra Bouncer og som brugeren har giver tilladelse til må anvendes i klient-applikationen. Disse data er overført via den identitets-token, som også bliver brugt til autorisation i API'et (se figur 13).

6. I klientapplikationen er der også implementere en metode, der henter en specifik værdi fra API'et. Indsæt et nummer i url'en (f.eks. `http://localhost:5002/values/5`), hvorved dette kald returneres til view'et (se figur 14).

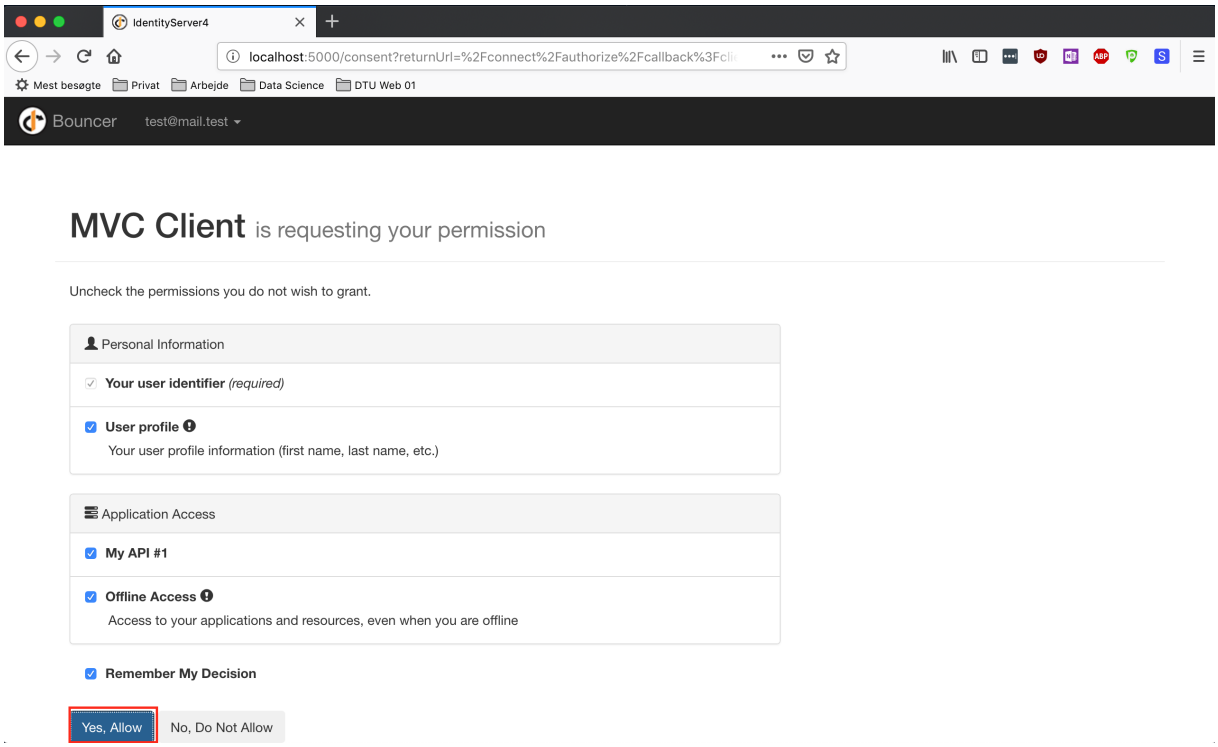
7. Forsøger man at tilgå API'et direkte uden om klienten, returneres der ikke nogen værdier (se figur 15). Dette skyldes Bouncer beskytter API'et, både i forhold til hvilke brugere der har adgang til API'et og hvilke klienter der har adgang til API'et. Tilgås API'et direkte via en browser, sker dette ikke via en godkendt klient, og der returneres ikke nogen værdier.



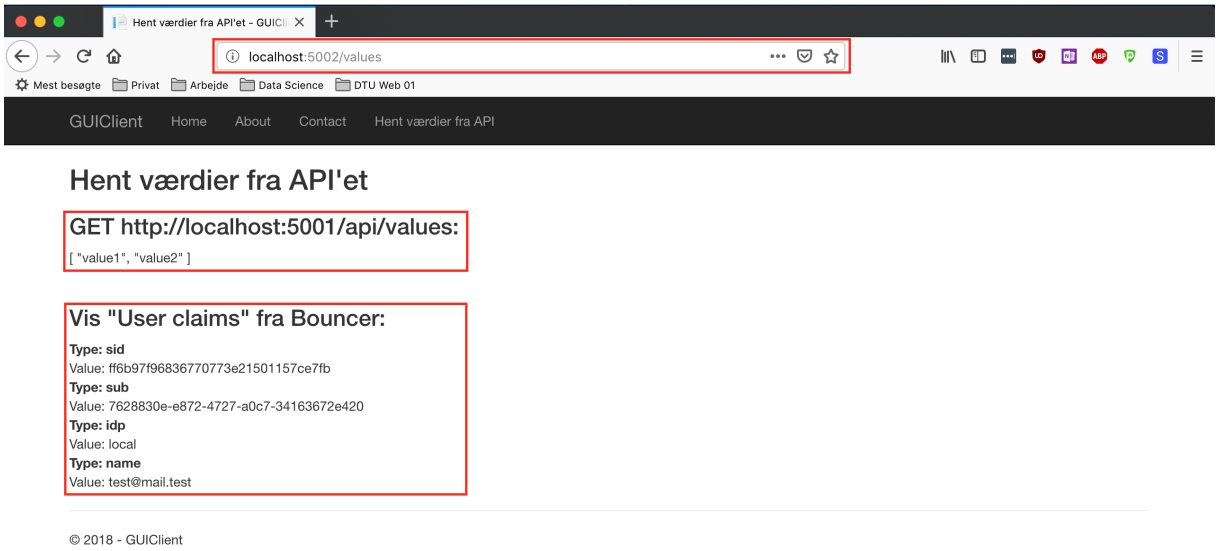
Figur 10: Startbillede i klient-applikationen



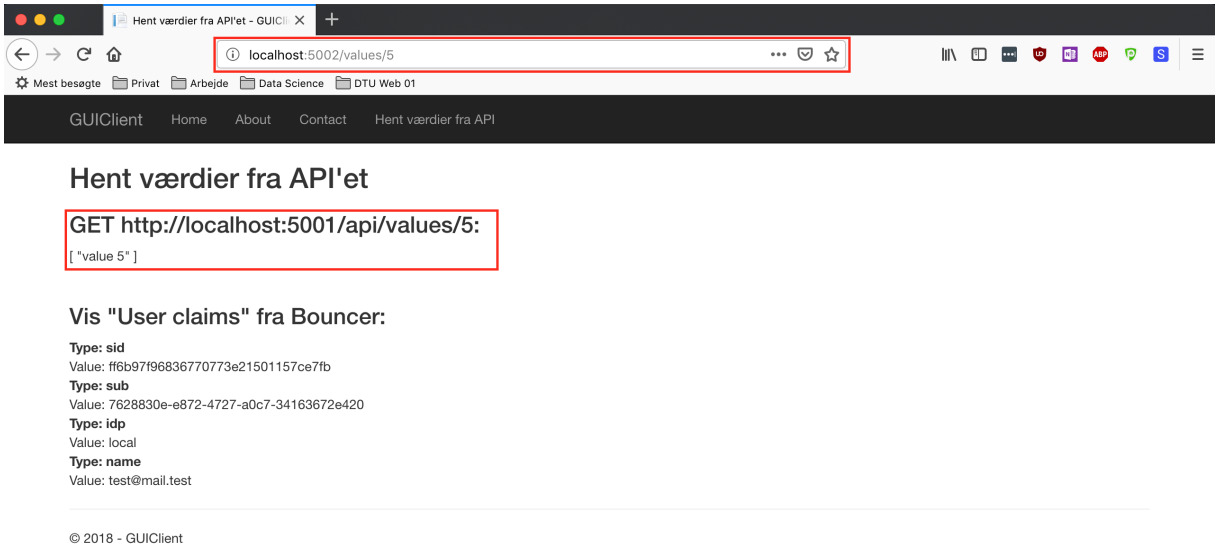
Figur 11: Omdirrigering til Bouncers loginside



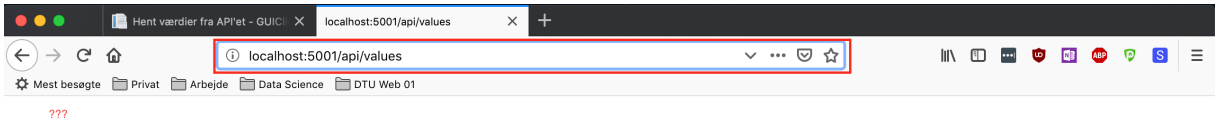
Figur 12: Brugeren godkender at klient applikationen må få adgang til brugerens informationer



Figur 13: Returnering til klient applikationen efter login. Værdier fra API'et er hentet og vises på siden



Figur 14: En anden metode fra API'et, som også er implementeret i klient applikationen



Figur 15: Adgang til API'et kan kun ske vie an klient der er godkendt via Bouncer.

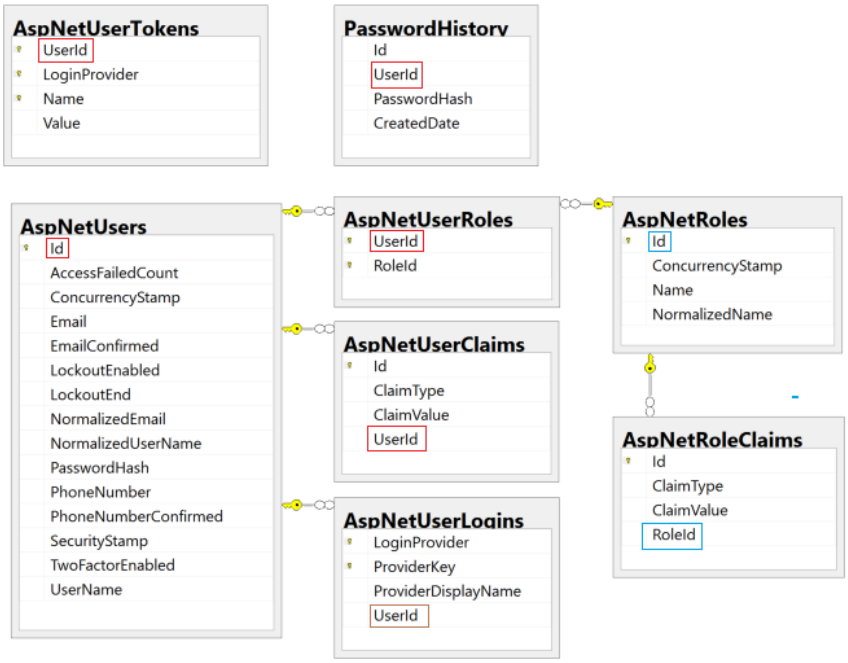
C. ER-diagrammer

Nedenfor er datamodellen for IdentityServer4 beskrevet og vist i form af ER-diagrammer. Diagrammerne stammer fra IdentityServer4 dokumentationen [10].

Datamodel for ASP.NET Identity

På figur 16 er datamodellen for ASP.NET Identity vist. Overordnet set består modellen af en tabel med brugere (AapNetUsers) og en tabel med roller (AspNetRoles). Til disse tabeller findes der så en række relaterede tabeller, der implementerer features for brugere og roller og skaber en relation mellem brugere og roller.

Ved at anvende / integrerer datamodellen for ASP.NET Identity i IdentityServer4, kan standart-funktionaliteten til brugerstyring i ASP.NET udnyttes, herunder brugerregistrering, login / logout, 2 faktor godkendlelse og password recovery.

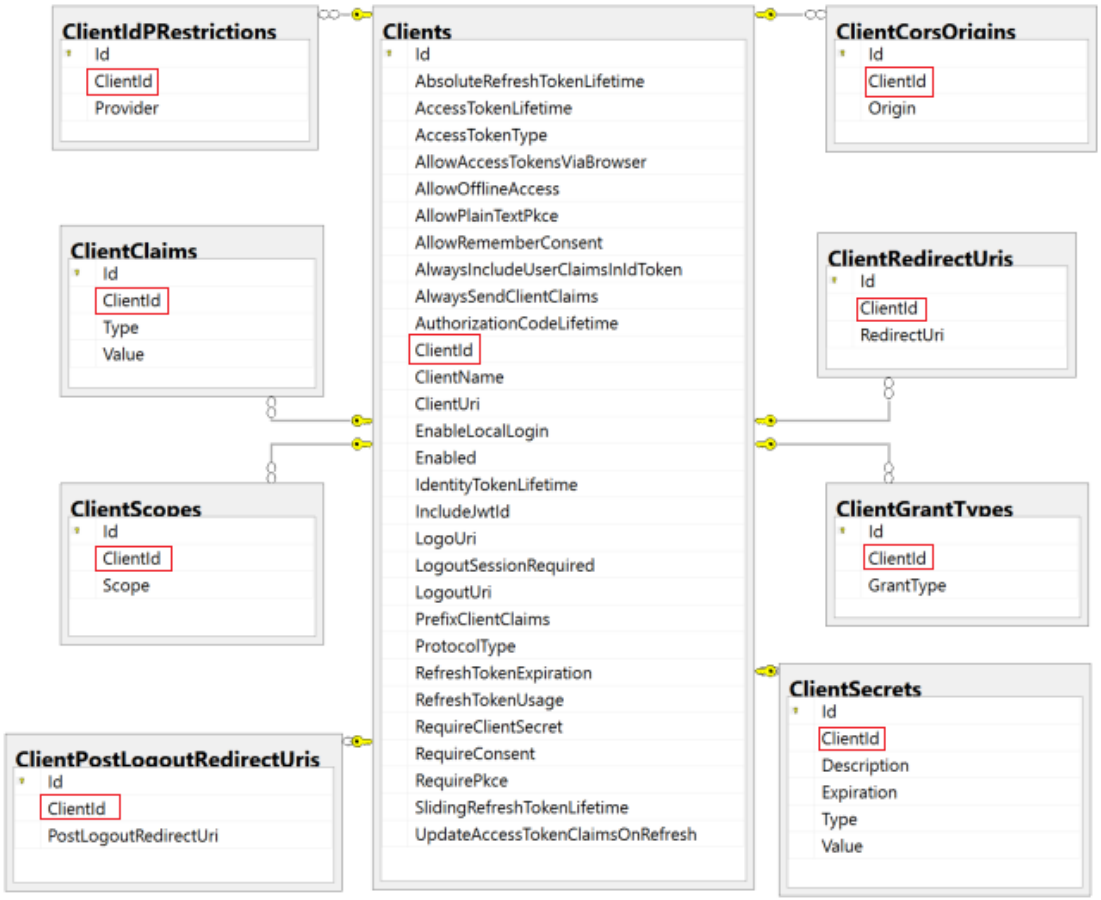


Figur 16: ER-diagram for ASP.NET Identity

Datamodel for klienter og api-ressourcer i IdentityServer4

På figur 17 er datamodellen for Klienter vist. Den centrale tabel i datamodellen er *Clients*, hvor klienterne registreres. Relaterede tabeller giver mulighed for bl.a. at registrerer *Claims* f.eks. hvilke api-ressourcer klienten skal have adgang til og hvilken adresse browsere skal føres tilbage til, efter autorisationen er gennemført.

På figur 18 er datamodellen for api-ressourcer vist. Tabellerne bruges til at registrere og identificerer api-ressourcer, herunder registreret passwords/nøgler og *Claims* for api-ressourcerne i Identityserver4



Figur 17: ER-diagram for klienter i IdentityServer4

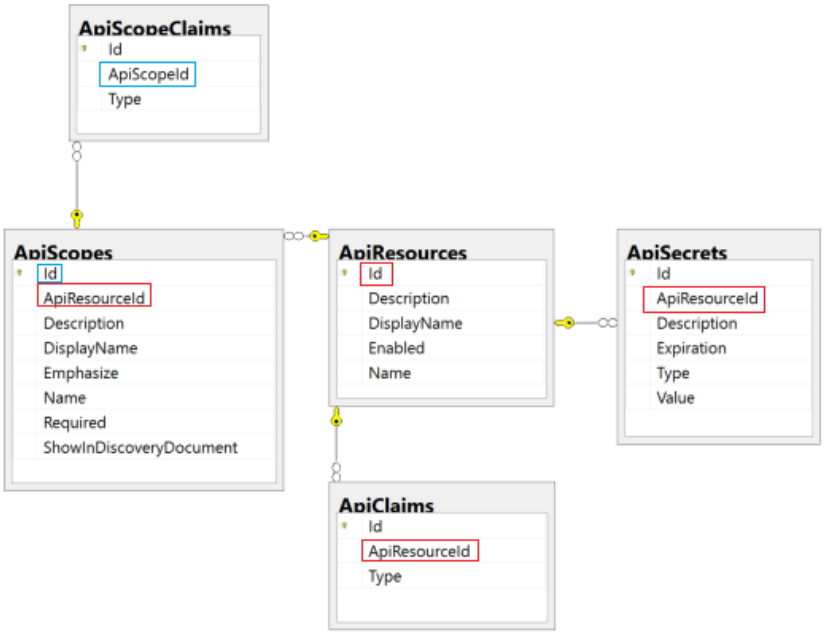
Datamodel for "IdentityResources"i IdentityServer4

På figur 19 er datamodellen for identitets ressourcer vist. IdentityRessource bruges til at registrere brugere og bruger-data, hvis brugerstyring ikke er implementeret med ASP.NET Identity.

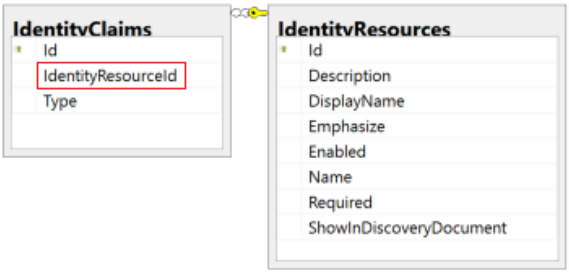
Tabel til "Persisted Grant"

På figur 20 er tabellen til registrering af "Persistet Grant"vist. Et "Persisted Grant"repræsenterer en identitets- eller access-token, som genereres i forbindelse med bruger- og klient-godkendelse.

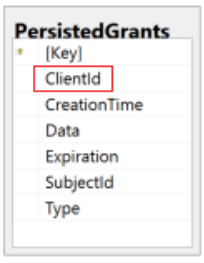
Token-strengen registreres i tabellen sammen med gyldighed / varighed.



Figur 18: ER-diagram for api-ressourcer i IdentityServer4



Figur 19: ER-diagram for "IdentityResources"i IdentityServer4



Figur 20: Tabel til registrering af "Persisted Grant"

D. Udskrift af koden - Bouncer

```
using Bouncer.Data;
using Bouncer.Models;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;

namespace Bouncer
{
    public class Startup
    {
        public IConfiguration Configuration { get; }
        public IHostingEnvironment Environment { get; }

        public Startup(IConfiguration configuration, IHostingEnvironment
environment)
        {
            Configuration = configuration;
            Environment = environment;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options =>

 options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")))
;

            services.AddIdentity<ApplicationUser, IdentityRole>()
```

```
30              .AddEntityFrameworkStores<ApplicationDbContext>()
31              .AddDefaultTokenProviders();
32
33          services.AddMvc();
34
35          var builder = services.AddIdentityServer(options =>
36          {
37              options.Events.RaiseErrorEvents = true;
38              options.Events.RaiseInformationEvents = true;
39              options.Events.RaiseFailureEvents = true;
40              options.Events.RaiseSuccessEvents = true;
41          })
42              .AddInMemoryIdentityResources(Config.GetIdentityResources())
43              .AddInMemoryApiResources(Config.GetApis())
44              .AddInMemoryClients(Config.GetClients())
45              .AddAspNetIdentity<ApplicationUser>();
46
47          if (Environment.IsDevelopment())
48          {
49              builder.AddDeveloperSigningCredential();
50          }
51          else
52          {
53              throw new Exception("need to configure key material");
54          }
55      }
56
57      public void Configure(IApplicationBuilder app)
58      {
59          if (Environment.IsDevelopment())
60          {
61              app.UseDeveloperExceptionPage();
```

```
62                  app.UseDatabaseErrorPage();
63              }
64              else
65              {
66                  app.UseExceptionHandler("/Home/Error");
67              }
68
69              app.UseStaticFiles();
70              app.UseIdentityServer();
71              app.UseMvcWithDefaultRoute();
72          }
73      }
74  }
75
```

```
using IdentityServer4;
using IdentityServer4.Models;
using System.Collections.Generic;

namespace Bouncer
{
    public static class Config
    {
        public static IEnumerable<IdentityResource> GetIdentityResources()
        {
            return new IdentityResource[]
            {
                new IdentityResources.OpenId(),
                new IdentityResources.Profile(),
            };
        }

        public static IEnumerable<ApiResource> GetApis()
        {
            return new ApiResource[]
            {
                new ApiResource("api1", "My API #1")
            };
        }

        public static IEnumerable<Client> GetClients()
        {
            return new[]
            {
                new Client
                {
                    ClientId = "mvc",
```

```
33                      ClientName = "MVC Client",
34                      AllowedGrantTypes = GrantTypes.HybridAndClientCredentials,
35
36                      ClientSecrets =
37                      {
38                          new Secret("511536EF-F270-4058-80CA-
    1C89C192F69A".Sha256())
39                      },
40
41                      // where to redirect to after login
42                      RedirectUris = { "http://localhost:5002/signin-oidc" },
43
44                      // where to redirect to after logout
45                      PostLogoutRedirectUris = { "http://localhost:5002/signout-
    callback-oidc" },
46
47                      AllowedScopes = new List<string>
48                      {
49                          IdentityServerConstants.StandardScopes.OpenId,
50                          IdentityServerConstants.StandardScopes.Profile,
51                          "api1"
52                      },
53                      AllowOfflineAccess = true
54                  }
55              };
56          }
57      }
58  }
59
```

E. Udskrift af koden: api1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

namespace api1
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvcCore()
                .AddAuthorization()
                .AddJsonFormatters();

```

```
32              services.AddAuthentication("Bearer")
33                  .AddIdentityServerAuthentication(options =>
34                  {
35                      options.Authority = "http://localhost:5000";
36                      options.RequireHttpsMetadata = false;
37
38                      options.ApiName = "api1";
39                  });
40          }
41
42          // This method gets called by the runtime. Use this method to
    configure the HTTP request pipeline.
43          public void Configure(IApplicationBuilder app, IHostingEnvironment
    env)
44          {
45              if (env.IsDevelopment())
46              {
47                  app.UseDeveloperExceptionPage();
48              }
49              else
50              {
51                  app.UseHsts();
52              }
53
54              app.UseHttpsRedirection();
55              app.UseAuthentication();
56              app.UseMvc();
57          }
58      }
59  }
60
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace api1.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        public ActionResult<IEnumerable<string>> Get(int id)
        {
            return new string[] {$"value {id.ToString()}"};
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
```

```
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}

```

F. Udskrift af koden: GUIClient

```
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace GUIClient
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
services to the container.
        public void ConfigureServices(IServiceCollection services)
        {

 services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

```

```
31              JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
32
33              // NB! HER KONFIGURERES KLIENTENS AUTENTIFIKATIONSMETODE
34              services.AddAuthentication(options =>
35                  {
36                      options.DefaultScheme = "IS4Cookies"; // navngivning af
    cookie
37                      options.DefaultChallengeScheme = "oidc"; // anvend openid
    connect
38                  })
39                  .AddCookie("IS4Cookies") // tilføj den pågældende cookie
40                  // anvend og konfiguration af OpenIdconnect service
41                  .AddOpenIdConnect("oidc", options =>
42                  {
43                      options.SignInScheme = "IS4Cookies"; // brug den samme
    cookie
44
45                      // sæt adressen til Bouncer som autorisations service
46                      options.Authority = "http://localhost:5000";
47                      options.RequireHttpsMetadata = false;
48
49                      // Klientens identitit i Bouncer - dette er konfigureret i
    Bouncer
50                      options.ClientId = "mvc"; // klientens identitet
51                      options.ClientSecret = "511536EF-F270-4058-80CA-
    1C89C192F69A"; // klientens "password"
52                      options.ResponseType = "code id_token";
53
54                      options.SaveTokens = true;
55                      options.GetClaimsFromUserInfoEndpoint = true;
56
57                      // Identificer API'en, som der skal trækkes data fra
```

```
58                      options.Scope.Add("api1");
59                      options.Scope.Add("offline_access");
60                      options.ClaimActions.MapJsonKey("website", "website");
61                  });
62          }
63
64          public void Configure(IApplicationBuilder app, IHostingEnvironment
    env)
65          {
66              if (env.IsDevelopment())
67              {
68                  app.UseDeveloperExceptionPage();
69              }
70              else
71              {
72                  app.UseExceptionHandler("/Home/Error");
73                  app.UseHsts();
74              }
75
76              app.UseHttpsRedirection();
77              app.UseAuthentication();
78              app.UseStaticFiles();
79              //app.UseCookiePolicy();
80
81              app.UseMvcWithDefaultRoute();
82          }
83      }
84  }
85
```

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using GUIClient.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication;
using System.Net.Http;
using System.Net.Http.Headers;
using Newtonsoft.Json.Linq;

namespace GUIClient.Controllers
{
    public class ApiController : Controller
    {
        [HttpGet]
        [Route("values")]
        [Authorize]
        public async Task<IActionResult> Values()
        {
            var accessToken = await HttpContext.GetTokenAsync("access_token");

            var client = new HttpClient();
            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", accessToken);
            var content = await
client.GetStringAsync("http://localhost:5001/api/values");

            ViewBag.Json = JArray.Parse(content).ToString();
            ViewData["Message"] = "GET http://localhost:5001/api/values:";
```

```
            return View();
        }

        [HttpGet]
        [Route("values/{id:int}")]
        [Authorize]
        public async Task<IActionResult> Values(int id)
        {
            var accessToken = await HttpContext.GetTokenAsync("access_token");

            var client = new HttpClient();
            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", accessToken);
            var content = await
client.GetStringAsync($"http://localhost:5001/api/values/{id.ToString()}");

            ViewBag.Json = JArray.Parse(content).ToString();
            ViewData["Message"] = $"GET
http://localhost:5001/api/values/{id.ToString()}:";
            return View();
        }


    }
}
```

G. Capture fra Wireshark

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
No.     Time           Protocol Length Source Port Destination Port Info
     25 1.741657       TCP      64     49991       5002             49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
     26 1.741665       TCP      64     49990       5002             49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
     27 1.741689       TCP      76     5002        49991            [TCP ACKed unseen segment] 5002 → 49991 [ACK] Seq=1 Ack=2 Win=6365
Len=0 TSval=116231853 TSecr=116211853
     28 1.741701       TCP      76     5002        49990            [TCP ACKed unseen segment] 5002 → 49990 [ACK] Seq=1 Ack=2 Win=6365
Len=0 TSval=116231853 TSecr=116211853
     29 1.756061       TCP      64     49988       5002             49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
     30 1.756141       TCP      76     5002        49988            [TCP ACKed unseen segment] 5002 → 49988 [ACK] Seq=1 Ack=2 Win=6359
Len=0 TSval=116231867 TSecr=116211866
     31 1.757254       TCP      64     49989       5002             49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
     32 1.757285       TCP      76     5002        49989            [TCP ACKed unseen segment] 5002 → 49989 [ACK] Seq=1 Ack=2 Win=6359
Len=0 TSval=116231868 TSecr=116211868
     45 2.373848       TCP      64     49984       5002             49984 → 5002 [ACK] Seq=1 Ack=1 Win=5471 Len=0
     46 2.373946       TCP      76     5002        49984            [TCP ACKed unseen segment] 5002 → 49984 [ACK] Seq=1 Ack=2 Win=6341
Len=0 TSval=116232481 TSecr=116212480
     47 2.547546       HTTP     460    49984       5002             [TCP Previous segment not captured] GET /values HTTP/1.1
     48 2.547577       TCP      76     5002        49984            [TCP ACKed unseen segment] 5002 → 49984 [ACK] Seq=1 Ack=386 Win=6335
Len=0 TSval=116232653 TSecr=116232653
     49 2.668218       TCP      88     50025       5000             50025 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116232771 TSecr=0 SACK_PERM=1
     50 2.668354       TCP      88     5000        50025            5000 → 50025 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116232771 TSecr=116232771 SACK_PERM=1
     51 2.668372       TCP      76     50025       5000             50025 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116232771
TSecr=116232771
     52 2.668392       TCP      76     5000        50025            [TCP Window Update] 5000 → 50025 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116232771 TSecr=116232771
     53 2.702923       IPA      206    50025       5000             unknown 0x54
     54 2.702953       TCP      76     5000        50025            5000 → 50025 [ACK] Seq=1 Ack=131 Win=407616 Len=0 TSval=116232804
TSecr=116232804
     67 3.783600       IPA      1718   5000        50025            unknown 0x54
     68 3.783726       TCP      76     50025       5000             50025 → 5000 [ACK] Seq=131 Ack=1643 Win=406144 Len=0 TSval=116233882
TSecr=116233882
     69 3.786662       RSL      81     5000        50025            UNIT DATA REQuest [Packet size limited during capture]
     70 3.786711       TCP      76     50025       5000             50025 → 5000 [ACK] Seq=131 Ack=1648 Win=406144 Len=0 TSval=116233885
TSecr=116233885
     71 3.934734       IPA      211    50025       5000             unknown 0x54
     72 3.934843       TCP      76     5000        50025            5000 → 50025 [ACK] Seq=1648 Ack=266 Win=407488 Len=0 TSval=116234032
TSecr=116234032
     85 4.033720       IPA      682    5000        50025            unknown 0x54
     86 4.033922       RSL      81     5000        50025            UNIT DATA REQuest [Packet size limited during capture]
     87 4.033953       TCP      76     50025       5000             50025 → 5000 [ACK] Seq=266 Ack=2254 Win=405504 Len=0 TSval=116234130
TSecr=116234130
     88 4.034042       TCP      76     50025       5000             50025 → 5000 [ACK] Seq=266 Ack=2259 Win=405504 Len=0 TSval=116234130
TSecr=116234130
     89 4.126447       HTTP     1477   5002        49984            HTTP/1.1 302 Found
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
     90 4.126478       TCP      76     49984       5002            49984 → 5002 [ACK] Seq=386 Ack=1402 Win=5449 Len=0 TSval=116234222
TSecr=116234222
     91 4.158224       TCP      88     50028       5000            50028 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116234253 TSecr=0 SACK_PERM=1
     92 4.158305       TCP      88     5000        50028           5000 → 50028 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116234253 TSecr=116234253 SACK_PERM=1
     93 4.158323       TCP      76     50028       5000            50028 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116234253
TSecr=116234253
     94 4.158337       TCP      76     5000        50028           [TCP Window Update] 5000 → 50028 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116234253 TSecr=116234253
     95 4.158692       IPA      1171   50028       5000            unknown 0x54
     96 4.158717       TCP      76     5000        50028           5000 → 50028 [ACK] Seq=1 Ack=1096 Win=406656 Len=0 TSval=116234253
TSecr=116234253
     97 4.402560       IPA      1016   5000        50028           unknown 0x54
     98 4.402591       TCP      76     50028       5000            50028 → 5000 [ACK] Seq=1096 Ack=941 Win=406848 Len=0 TSval=116234497
TSecr=116234497
     99 4.419936       IPA      1265   50028       5000            unknown 0x54
    100 4.419971       TCP      76     5000        50028           5000 → 50028 [ACK] Seq=941 Ack=2285 Win=405504 Len=0 TSval=116234513
TSecr=116234513
    185 11.782422      TCP      64     49991       5002            [TCP Dup ACK 25#1] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    186 11.782431      TCP      64     49990       5002            [TCP Dup ACK 26#1] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    187 11.782452      TCP      76     5002        49991           [TCP Dup ACK 27#1] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116241853 TSecr=116211853
    188 11.782459      TCP      76     5002        49990           [TCP Dup ACK 28#1] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116241853 TSecr=116211853
    189 11.796781      TCP      64     49988       5002            [TCP Dup ACK 29#1] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    190 11.796898      TCP      76     5002        49988           [TCP Dup ACK 30#1] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116241867 TSecr=116211866
    191 11.798033      TCP      64     49989       5002            [TCP Dup ACK 31#1] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    192 11.798104      TCP      76     5002        49989           [TCP Dup ACK 32#1] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116241868 TSecr=116211868
    229 14.158454      TCP      64     49984       5002            [TCP Keep-Alive] 49984 → 5002 [ACK] Seq=385 Ack=1402 Win=5449 Len=0
    230 14.158613      TCP      76     5002        49984           [TCP Keep-Alive ACK] 5002 → 49984 [ACK] Seq=1402 Ack=386 Win=6335
Len=0 TSval=116244222 TSecr=116234222
    231 14.449760      TCP      64     50028       5000            [TCP Keep-Alive] 50028 → 5000 [ACK] Seq=2284 Ack=941 Win=406848
Len=0
    232 14.449790      TCP      76     5000        50028           [TCP Keep-Alive ACK] 5000 → 50028 [ACK] Seq=941 Ack=2285 Win=405504
Len=0 TSval=116244513 TSecr=116234513
    257 16.368632      IPA      6085   5000        50028           unknown 0x54
    258 16.368670      TCP      76     50028       5000            50028 → 5000 [ACK] Seq=2285 Ack=6950 Win=400832 Len=0
TSval=116246426 TSecr=116246426
    259 16.386150      RSL      81     5000        50028           UNIT DATA REQuest [Packet size limited during capture]
    260 16.386202      TCP      76     50028       5000            50028 → 5000 [ACK] Seq=2285 Ack=6955 Win=400832 Len=0
TSval=116246443 TSecr=116246443
    261 16.551614      IPA      662    50028       5000            unknown 0x54
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    262 16.551656      TCP      76     5000        50028           5000 → 50028 [ACK] Seq=6955 Ack=2871 Win=404928 Len=0
TSval=116246608 TSecr=116246608
    263 16.554729      TCP      88     50041       5000            50041 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246610 TSecr=0 SACK_PERM=1
    264 16.554831      TCP      88     5000        50041           5000 → 50041 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246610 TSecr=116246610 SACK_PERM=1
    265 16.554853      TCP      76     50041       5000            50041 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116246610
TSecr=116246610
    266 16.554869      TCP      76     5000        50041           [TCP Window Update] 5000 → 50041 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116246610 TSecr=116246610
    267 16.559843      IPA      643    50041       5000            unknown 0x54
    268 16.559923      TCP      76     5000        50041           5000 → 50041 [ACK] Seq=1 Ack=568 Win=407232 Len=0 TSval=116246614
TSecr=116246614
    269 16.588559      TCP      88     50042       5000            50042 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246641 TSecr=0 SACK_PERM=1
    270 16.588668      TCP      88     5000        50042           5000 → 50042 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246641 TSecr=116246641 SACK_PERM=1
    271 16.588689      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116246641
TSecr=116246641
    272 16.588705      TCP      76     5000        50042           [TCP Window Update] 5000 → 50042 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116246641 TSecr=116246641
    273 16.600106      IPA      636    50042       5000            unknown 0x54
    274 16.600151      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=1 Ack=561 Win=407232 Len=0 TSval=116246652
TSecr=116246652
    275 16.611042      TCP      88     50043       5000            50043 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246662 TSecr=0 SACK_PERM=1
    276 16.611290      TCP      88     5000        50043           5000 → 50043 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116246662 TSecr=116246662 SACK_PERM=1
    277 16.611315      TCP      76     50043       5000            50043 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116246662
TSecr=116246662
    278 16.611339      TCP      76     5000        50043           [TCP Window Update] 5000 → 50043 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116246662 TSecr=116246662
    279 16.612361      IPA      645    50043       5000            unknown 0x54
    280 16.612504      TCP      76     5000        50043           5000 → 50043 [ACK] Seq=1 Ack=570 Win=407168 Len=0 TSval=116246663
TSecr=116246663
    281 16.668343      IPA      290    5000        50043           unknown 0x54
    282 16.668344      IPA      276    5000        50028           unknown 0x54
    283 16.668377      TCP      76     50043       5000            50043 → 5000 [ACK] Seq=570 Ack=215 Win=407552 Len=0 TSval=116246716
TSecr=116246716
    284 16.668390      TCP      76     50028       5000            50028 → 5000 [ACK] Seq=2871 Ack=7155 Win=400640 Len=0
TSval=116246716 TSecr=116246716
    285 16.668418      IPA      276    5000        50041           unknown 0x54
    286 16.668432      TCP      76     50041       5000            50041 → 5000 [ACK] Seq=568 Ack=201 Win=407552 Len=0 TSval=116246716
TSecr=116246716
    287 16.668495      IPA      290    5000        50042           unknown 0x54
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    288 16.668514      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=561 Ack=215 Win=407552 Len=0 TSval=116246716
TSecr=116246716
    289 16.669145      IPA      540    50028       5000            unknown 0x54
    290 16.669177      TCP      76     5000        50028           5000 → 50028 [ACK] Seq=7155 Ack=3335 Win=404416 Len=0
TSval=116246716 TSecr=116246716
    291 16.736470      IPA      16388  5000        50028           unknown 0x54
    292 16.736481      IPA      4774   5000        50028           unknown 0x20
    293 16.736522      TCP      76     50028       5000            50028 → 5000 [ACK] Seq=3335 Ack=28165 Win=379584 Len=0
TSval=116246782 TSecr=116246782
    294 16.976699      IPA      603    50028       5000            unknown 0x54
    295 16.976918      TCP      76     5000        50028           5000 → 50028 [ACK] Seq=28165 Ack=3862 Win=403904 Len=0
TSval=116247020 TSecr=116247020
    296 16.982913      TCP      76     50028       5000            50028 → 5000 [FIN, ACK] Seq=3862 Ack=28165 Win=379584 Len=0
TSval=116247025 TSecr=116247020
    297 16.983034      TCP      76     5000        50028           5000 → 50028 [ACK] Seq=28165 Ack=3863 Win=403904 Len=0
TSval=116247025 TSecr=116247025
    298 16.983308      IPA      1442   5000        50028           unknown 0x54
    299 16.983354      TCP      64     50028       5000            50028 → 5000 [RST] Seq=3863 Win=0 Len=0
    360 21.828881      TCP      64     49991       5002            [TCP Dup ACK 25#2] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    361 21.828895      TCP      64     49990       5002            [TCP Dup ACK 26#2] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    362 21.828955      TCP      76     5002        49991           [TCP Dup ACK 27#2] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116251853 TSecr=116211853
    363 21.828969      TCP      76     5002        49990           [TCP Dup ACK 28#2] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116251853 TSecr=116211853
    364 21.844108      TCP      64     49989       5002            [TCP Dup ACK 31#2] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    365 21.844122      TCP      64     49988       5002            [TCP Dup ACK 29#2] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    366 21.844168      TCP      76     5002        49989           [TCP Dup ACK 32#2] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116251868 TSecr=116211868
    367 21.844181      TCP      76     5002        49988           [TCP Dup ACK 30#2] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116251868 TSecr=116211866
    406 24.206192      TCP      64     49984       5002            [TCP Keep-Alive] 49984 → 5002 [ACK] Seq=385 Ack=1402 Win=5449 Len=0
    407 24.206245      TCP      76     5002        49984           [TCP Keep-Alive ACK] 5002 → 49984 [ACK] Seq=1402 Ack=386 Win=6335
Len=0 TSval=116254222 TSecr=116234222
    432 26.706104      TCP      64     50043       5000            [TCP Keep-Alive] 50043 → 5000 [ACK] Seq=569 Ack=215 Win=407552 Len=0
    433 26.706125      TCP      64     50042       5000            [TCP Keep-Alive] 50042 → 5000 [ACK] Seq=560 Ack=215 Win=407552 Len=0
    434 26.706133      TCP      64     50041       5000            [TCP Keep-Alive] 50041 → 5000 [ACK] Seq=567 Ack=201 Win=407552 Len=0
    435 26.706166      TCP      76     5000        50043           [TCP Keep-Alive ACK] 5000 → 50043 [ACK] Seq=215 Ack=570 Win=407168
Len=0 TSval=116256716 TSecr=116246716
    436 26.706184      TCP      76     5000        50042           [TCP Keep-Alive ACK] 5000 → 50042 [ACK] Seq=215 Ack=561 Win=407232
Len=0 TSval=116256716 TSecr=116246716
    437 26.706192      TCP      76     5000        50041           [TCP Keep-Alive ACK] 5000 → 50041 [ACK] Seq=201 Ack=568 Win=407232
Len=0 TSval=116256716 TSecr=116246716
    498 31.864828      TCP      64     49991       5002            [TCP Dup ACK 25#3] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    499 31.864847      TCP      64     49990       5002            [TCP Dup ACK 26#3] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    500 31.864898      TCP      76     5002        49991           [TCP Dup ACK 27#3] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116261853 TSecr=116211853
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    501 31.864917      TCP      76     5002        49990           [TCP Dup ACK 28#3] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116261853 TSecr=116211853
    502 31.881008      TCP      64     49989       5002            [TCP Dup ACK 31#3] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    503 31.881020      TCP      64     49988       5002            [TCP Dup ACK 29#3] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    504 31.881062      TCP      76     5002        49989           [TCP Dup ACK 32#3] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116261869 TSecr=116211868
    505 31.881072      TCP      76     5002        49988           [TCP Dup ACK 30#3] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116261869 TSecr=116211866
    518 32.802935      IPA      2563   50043       5000            unknown 0x53
    519 32.803076      TCP      76     5000        50043           5000 → 50043 [ACK] Seq=215 Ack=3057 Win=404736 Len=0 TSval=116262786
TSecr=116262786
    544 34.246383      TCP      64     49984       5002            [TCP Keep-Alive] 49984 → 5002 [ACK] Seq=385 Ack=1402 Win=5449 Len=0
    545 34.246492      TCP      76     5002        49984           [TCP Keep-Alive ACK] 5002 → 49984 [ACK] Seq=1402 Ack=386 Win=6335
Len=0 TSval=116264223 TSecr=116234222
    590 36.906441      TCP      64     50042       5000            [TCP Keep-Alive] 50042 → 5000 [ACK] Seq=560 Ack=215 Win=407552 Len=0
    591 36.906452      TCP      64     50041       5000            [TCP Keep-Alive] 50041 → 5000 [ACK] Seq=567 Ack=201 Win=407552 Len=0
    592 36.906475      TCP      76     5000        50042           [TCP Keep-Alive ACK] 5000 → 50042 [ACK] Seq=215 Ack=561 Win=407232
Len=0 TSval=116266716 TSecr=116246716
    593 36.906485      TCP      76     5000        50041           [TCP Keep-Alive ACK] 5000 → 50041 [ACK] Seq=201 Ack=568 Win=407232
Len=0 TSval=116266716 TSecr=116246716
    652 39.842642      IPA      2171   5000        50043           unknown 0x54
    653 39.842673      TCP      76     50043       5000            50043 → 5000 [ACK] Seq=3057 Ack=2310 Win=405440 Len=0
TSval=116269610 TSecr=116269610
    654 39.858488      IPA      2503   50041       5000            unknown 0x54
    655 39.858519      TCP      76     5000        50041           5000 → 50041 [ACK] Seq=201 Ack=2995 Win=404800 Len=0 TSval=116269625
TSecr=116269625
    660 40.010010      IPA      1010   5000        50041           unknown 0x54
    661 40.010046      TCP      76     50041       5000            50041 → 5000 [ACK] Seq=2995 Ack=1135 Win=406656 Len=0
TSval=116269775 TSecr=116269775
    662 40.026678      IPA      2582   50042       5000            unknown 0x54
    663 40.026715      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=215 Ack=3067 Win=404672 Len=0 TSval=116269790
TSecr=116269790
    690 41.432743      IPA      7869   5000        50042           unknown 0x54
    691 41.432848      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=3067 Ack=8008 Win=399744 Len=0
TSval=116271189 TSecr=116271189
    692 41.432875      RSL      81     5000        50042           UNIT DATA REQuest [Packet size limited during capture]
    693 41.432885      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=3067 Ack=8013 Win=399744 Len=0
TSval=116271189 TSecr=116271189
    694 41.779370      IPA      1779   50042       5000            unknown 0x54
    695 41.779421      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=8013 Ack=4770 Win=403008 Len=0
TSval=116271534 TSecr=116271534
    696 41.780047      IPA      277    5000        50042           unknown 0x54
    697 41.780076      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=4770 Ack=8214 Win=399552 Len=0
TSval=116271534 TSecr=116271534
    700 42.100692      TCP      64     49991       5002            [TCP Dup ACK 25#4] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    701 42.100705      TCP      64     49990       5002            [TCP Dup ACK 26#4] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    702 42.100773      TCP      76     5002        49991           [TCP Dup ACK 27#4] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116271853 TSecr=116211853
    703 42.100798      TCP      76     5002        49990           [TCP Dup ACK 28#4] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116271853 TSecr=116211853
    704 42.116800      TCP      64     49989       5002            [TCP Dup ACK 31#4] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    705 42.116828      TCP      64     49988       5002            [TCP Dup ACK 29#4] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    706 42.116993      TCP      76     5002        49989           [TCP Dup ACK 32#4] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116271869 TSecr=116211868
    707 42.117003      TCP      76     5002        49988           [TCP Dup ACK 30#4] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116271869 TSecr=116211866
    720 42.260537      IPA      1799   50042       5000            unknown 0x54
    721 42.260569      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=8214 Ack=6493 Win=401280 Len=0
TSval=116272011 TSecr=116272011
    722 42.261485      IPA      16388  5000        50042           unknown 0x54
    723 42.261491      IPA      2007   5000        50042           unknown 0xfb
    724 42.261523      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=6493 Ack=26457 Win=381312 Len=0
TSval=116272011 TSecr=116272011
    753 44.481009      TCP      64     49984       5002            [TCP Keep-Alive] 49984 → 5002 [ACK] Seq=385 Ack=1402 Win=5449 Len=0
    754 44.481063      TCP      76     5002        49984           [TCP Keep-Alive ACK] 5002 → 49984 [ACK] Seq=1402 Ack=386 Win=6335
Len=0 TSval=116274224 TSecr=116234222
    769 45.253386      IPA      3032   50042       5000            unknown 0x53
    770 45.253417      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=26457 Ack=9449 Win=398336 Len=0
TSval=116274994 TSecr=116274994
    771 45.335345      IPA      1320   5000        50042           unknown 0x54
    772 45.335429      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=9449 Ack=27701 Win=380096 Len=0
TSval=116275075 TSecr=116275075
    773 45.356130      IPA      2869   50042       5000            unknown 0x54
    774 45.356199      TCP      76     5000        50042           5000 → 50042 [ACK] Seq=27701 Ack=12242 Win=395520 Len=0
TSval=116275095 TSecr=116275095
    779 45.821067      IPA      3804   5000        50042           unknown 0x54
    780 45.821199      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=12242 Ack=31429 Win=376320 Len=0
TSval=116275555 TSecr=116275555
    781 45.821267      RSL      81     5000        50042           UNIT DATA REQuest [Packet size limited during capture]
    782 45.821334      TCP      76     50042       5000            50042 → 5000 [ACK] Seq=12242 Ack=31434 Win=376320 Len=0
TSval=116275555 TSecr=116275555
    783 46.098616      HTTP     3799   49984       5002            POST /signin-oidc HTTP/1.1  (application/x-www-form-urlencoded)
    784 46.098652      TCP      76     5002        49984           5002 → 49984 [ACK] Seq=1402 Ack=4109 Win=6277 Len=0 TSval=116275828
TSecr=116275828
    798 46.312645      IPA      479    50025       5000            unknown 0x53
    799 46.312687      TCP      76     5000        50025           5000 → 50025 [ACK] Seq=2259 Ack=669 Win=407104 Len=0 TSval=116276040
TSecr=116276040
    814 46.681068      IPA      2219   5000        50025           unknown 0x54
    815 46.681111      RSL      81     5000        50025           UNIT DATA REQuest [Packet size limited during capture]
    816 46.681111      TCP      76     50025       5000            50025 → 5000 [ACK] Seq=669 Ack=4402 Win=403392 Len=0 TSval=116276399
TSecr=116276399
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    817 46.681125      TCP      76     50025       5000            50025 → 5000 [ACK] Seq=669 Ack=4407 Win=403392 Len=0 TSval=116276399
TSecr=116276399
    818 46.700751      IPA      1024   50025       5000            unknown 0x54
    819 46.700794      TCP      76     5000        50025           5000 → 50025 [ACK] Seq=4407 Ack=1617 Win=406144 Len=0
TSval=116276418 TSecr=116276418
    832 46.921418      IPA      402    5000        50025           unknown 0x54
    833 46.921495      RSL      81     5000        50025           UNIT DATA REQuest [Packet size limited during capture]
    834 46.921517      TCP      76     50025       5000            50025 → 5000 [ACK] Seq=1617 Ack=4733 Win=403008 Len=0
TSval=116276635 TSecr=116276635
    835 46.921537      TCP      76     50025       5000            50025 → 5000 [ACK] Seq=1617 Ack=4738 Win=403008 Len=0
TSval=116276635 TSecr=116276635
    836 46.961791      HTTP     4548   5002        49984           HTTP/1.1 302 Found
    837 46.961827      TCP      76     49984       5002            49984 → 5002 [ACK] Seq=4109 Ack=5874 Win=5379 Len=0 TSval=116276674
TSecr=116276674
    838 46.979109      HTTP     5462   49984       5002            GET /values HTTP/1.1
    839 46.979137      TCP      76     5002        49984           5002 → 49984 [ACK] Seq=5874 Ack=9495 Win=6193 Len=0 TSval=116276690
TSecr=116276690
    840 47.027187      TCP      88     50091       5001            50091 → 5001 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116276736 TSecr=0 SACK_PERM=1
    841 47.027251      TCP      88     5001        50091           5001 → 50091 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116276736 TSecr=116276736 SACK_PERM=1
    842 47.027264      TCP      76     50091       5001            50091 → 5001 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116276736
TSecr=116276736
    843 47.027276      TCP      76     5001        50091           [TCP Window Update] 5001 → 50091 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116276736 TSecr=116276736
    844 47.027523      HTTP     960    50091       5001            GET /api/values HTTP/1.1
    845 47.027543      TCP      76     5001        50091           5001 → 50091 [ACK] Seq=1 Ack=885 Win=406912 Len=0 TSval=116276736
TSecr=116276736
    858 47.322916      TCP      88     50093       5000            50093 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116277027 TSecr=0 SACK_PERM=1
    859 47.323159      TCP      88     5000        50093           5000 → 50093 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116277027 TSecr=116277027 SACK_PERM=1
    860 47.323187      TCP      76     50093       5000            50093 → 5000 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116277027
TSecr=116277027
    861 47.323210      TCP      76     5000        50093           [TCP Window Update] 5000 → 50093 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116277027 TSecr=116277027
    862 47.357955      IPA      148    50093       5000            unknown 0x54
    863 47.358139      TCP      76     5000        50093           5000 → 50093 [ACK] Seq=1 Ack=73 Win=407680 Len=0 TSval=116277061
TSecr=116277061
    864 47.362369      IPA      1723   5000        50093           unknown 0x54
    865 47.362406      TCP      76     50093       5000            50093 → 5000 [ACK] Seq=73 Ack=1648 Win=406144 Len=0 TSval=116277065
TSecr=116277065
    866 47.513820      IPA      153    50093       5000            unknown 0x54
    867 47.513860      TCP      76     5000        50093           5000 → 50093 [ACK] Seq=1648 Ack=150 Win=407616 Len=0 TSval=116277214
TSecr=116277214
    868 47.515552      IPA      687    5000        50093           unknown 0x54
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
    869 47.515588      TCP      76     50093       5000            50093 → 5000 [ACK] Seq=150 Ack=2259 Win=405504 Len=0 TSval=116277215
TSecr=116277215
    870 47.898552      TCP      249    5001        50091           5001 → 50091 [PSH, ACK] Seq=1 Ack=885 Win=406912 Len=173
TSval=116277596 TSecr=116276736 [TCP segment of a reassembled PDU]
    871 47.898587      TCP      76     50091       5001            50091 → 5001 [ACK] Seq=885 Ack=174 Win=407616 Len=0 TSval=116277596
TSecr=116277596
    872 47.901206      HTTP     81     5001        50091           HTTP/1.1 200 OK  (application/json)
    873 47.901253      TCP      76     50091       5001            50091 → 5001 [ACK] Seq=885 Ack=179 Win=407616 Len=0 TSval=116277598
TSecr=116277598
    876 48.149374      HTTP     2571   5002        49984           HTTP/1.1 200 OK  (text/html)
    877 48.149435      TCP      76     49984       5002            49984 → 5002 [ACK] Seq=9495 Ack=8369 Win=5340 Len=0 TSval=116277844
TSecr=116277844
    904 49.921032      TCP      64     50043       5000            [TCP Keep-Alive] 50043 → 5000 [ACK] Seq=3056 Ack=2310 Win=405440
Len=0
    905 49.921087      TCP      76     5000        50043           [TCP Keep-Alive ACK] 5000 → 50043 [ACK] Seq=2310 Ack=3057 Win=404736
Len=0 TSval=116279610 TSecr=116269610
    908 50.087382      TCP      64     50041       5000            [TCP Keep-Alive] 50041 → 5000 [ACK] Seq=2994 Ack=1135 Win=406656
Len=0
    909 50.087438      TCP      76     5000        50041           [TCP Keep-Alive ACK] 5000 → 50041 [ACK] Seq=1135 Ack=2995 Win=404800
Len=0 TSval=116279776 TSecr=116269775
    938 52.172674      TCP      64     49991       5002            [TCP Dup ACK 25#5] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    939 52.172684      TCP      64     49990       5002            [TCP Dup ACK 26#5] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
    940 52.172732      TCP      76     5002        49991           [TCP Dup ACK 27#5] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116281853 TSecr=116211853
    941 52.172745      TCP      76     5002        49990           [TCP Dup ACK 28#5] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116281853 TSecr=116211853
    942 52.189828      TCP      64     49989       5002            [TCP Dup ACK 31#5] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    943 52.189843      TCP      64     49988       5002            [TCP Dup ACK 29#5] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
    944 52.189889      TCP      76     5002        49989           [TCP Dup ACK 32#5] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116281870 TSecr=116211868
    945 52.189903      TCP      76     5002        49988           [TCP Dup ACK 30#5] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116281870 TSecr=116211866
   1002 55.889257      TCP      64     50042       5000            [TCP Keep-Alive] 50042 → 5000 [ACK] Seq=12241 Ack=31434 Win=376320
Len=0
   1003 55.889314      TCP      76     5000        50042           [TCP Keep-Alive ACK] 5000 → 50042 [ACK] Seq=31434 Ack=12242
Win=395520 Len=0 TSval=116285555 TSecr=116275555
   1034 58.188554      TCP      64     49984       5002            [TCP Keep-Alive] 49984 → 5002 [ACK] Seq=9494 Ack=8369 Win=5340 Len=0
   1035 58.188598      TCP      76     5002        49984           [TCP Keep-Alive ACK] 5002 → 49984 [ACK] Seq=8369 Ack=9495 Win=6193
Len=0 TSval=116287844 TSecr=116277844
   1048 58.454449      HTTP     5464   49984       5002            GET /values/5 HTTP/1.1
   1049 58.454488      TCP      76     5002        49984           5002 → 49984 [ACK] Seq=8369 Ack=14883 Win=6109 Len=0 TSval=116288109
TSecr=116288109
   1050 58.514796      TCP      88     50109       5001            50109 → 5001 [SYN] Seq=0 Win=65535 Len=0 MSS=16324 WS=64
TSval=116288168 TSecr=0 SACK_PERM=1
   1051 58.515095      TCP      88     5001        50109           5001 → 50109 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16324 WS=64
TSval=116288168 TSecr=116288168 SACK_PERM=1
```

/Users/ak/DotNetCore/Web01Final/test/Wireshark.pcapng 1203 total packets, 256 shown

```
   1052 58.515153      TCP      76     50109       5001            50109 → 5001 [ACK] Seq=1 Ack=1 Win=407744 Len=0 TSval=116288168
TSecr=116288168
   1053 58.515207      TCP      76     5001        50109           [TCP Window Update] 5001 → 50109 [ACK] Seq=1 Ack=1 Win=407744 Len=0
TSval=116288168 TSecr=116288168
   1054 58.516357      HTTP     962    50109       5001            GET /api/values/5 HTTP/1.1
   1055 58.516438      TCP      76     5001        50109           5001 → 50109 [ACK] Seq=1 Ack=887 Win=406912 Len=0 TSval=116288169
TSecr=116288169
   1056 58.564109      TCP      240    5001        50109           5001 → 50109 [PSH, ACK] Seq=1 Ack=887 Win=406912 Len=164
TSval=116288217 TSecr=116288169 [TCP segment of a reassembled PDU]
   1057 58.564152      TCP      76     50109       5001            50109 → 5001 [ACK] Seq=887 Ack=165 Win=407616 Len=0 TSval=116288217
TSecr=116288217
   1058 58.564517      HTTP     81     5001        50109           HTTP/1.1 200 OK  (application/json)
   1059 58.564559      TCP      76     50109       5001            50109 → 5001 [ACK] Seq=887 Ack=170 Win=407616 Len=0 TSval=116288217
TSecr=116288217
   1060 58.573091      TCP      2545   5002        49984           5002 → 49984 [PSH, ACK] Seq=8369 Ack=14883 Win=6109 Len=2469
TSval=116288225 TSecr=116288109 [TCP segment of a reassembled PDU]
   1061 58.573121      HTTP     81     5002        49984           HTTP/1.1 200 OK  (text/html)
   1062 58.573130      TCP      76     49984       5002            49984 → 5002 [ACK] Seq=14883 Ack=10838 Win=5301 Len=0
TSval=116288225 TSecr=116288225
   1063 58.573141      TCP      76     49984       5002            49984 → 5002 [ACK] Seq=14883 Ack=10843 Win=5301 Len=0
TSval=116288225 TSecr=116288225
   1078 59.962693      TCP      64     50043       5000            [TCP Keep-Alive] 50043 → 5000 [ACK] Seq=3056 Ack=2310 Win=405440
Len=0
   1079 59.962763      TCP      76     5000        50043           [TCP Keep-Alive ACK] 5000 → 50043 [ACK] Seq=2310 Ack=3057 Win=404736
Len=0 TSval=116289610 TSecr=116269610
   1082 60.130611      TCP      64     50041       5000            [TCP Keep-Alive] 50041 → 5000 [ACK] Seq=2994 Ack=1135 Win=406656
Len=0
   1083 60.130659      TCP      76     5000        50041           [TCP Keep-Alive ACK] 5000 → 50041 [ACK] Seq=1135 Ack=2995 Win=404800
Len=0 TSval=116289776 TSecr=116269775
   1112 62.284514      TCP      64     49991       5002            [TCP Dup ACK 25#6] 49991 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
   1113 62.284529      TCP      64     49990       5002            [TCP Dup ACK 26#6] 49990 → 5002 [ACK] Seq=1 Ack=1 Win=6368 Len=0
   1114 62.284534      TCP      64     49989       5002            [TCP Dup ACK 31#6] 49989 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
   1115 62.284540      TCP      64     49988       5002            [TCP Dup ACK 29#6] 49988 → 5002 [ACK] Seq=1 Ack=1 Win=6365 Len=0
   1116 62.284595      TCP      76     5002        49991           [TCP Dup ACK 27#6] [TCP ACKed unseen segment] 5002 → 49991 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116291922 TSecr=116211853
   1117 62.284608      TCP      76     5002        49990           [TCP Dup ACK 28#6] [TCP ACKed unseen segment] 5002 → 49990 [ACK]
Seq=1 Ack=2 Win=6365 Len=0 TSval=116291922 TSecr=116211853
   1118 62.284616      TCP      76     5002        49989           [TCP Dup ACK 32#6] [TCP ACKed unseen segment] 5002 → 49989 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116291922 TSecr=116211868
   1119 62.284624      TCP      76     5002        49988           [TCP Dup ACK 30#6] [TCP ACKed unseen segment] 5002 → 49988 [ACK]
Seq=1 Ack=2 Win=6359 Len=0 TSval=116291922 TSecr=116211866
   1174 66.010293      TCP      64     50042       5000            [TCP Keep-Alive] 50042 → 5000 [ACK] Seq=12241 Ack=31434 Win=376320
Len=0
   1175 66.010348      TCP      76     5000        50042           [TCP Keep-Alive ACK] 5000 → 50042 [ACK] Seq=31434 Ack=12242
Win=395520 Len=0 TSval=116295635 TSecr=116275555
```