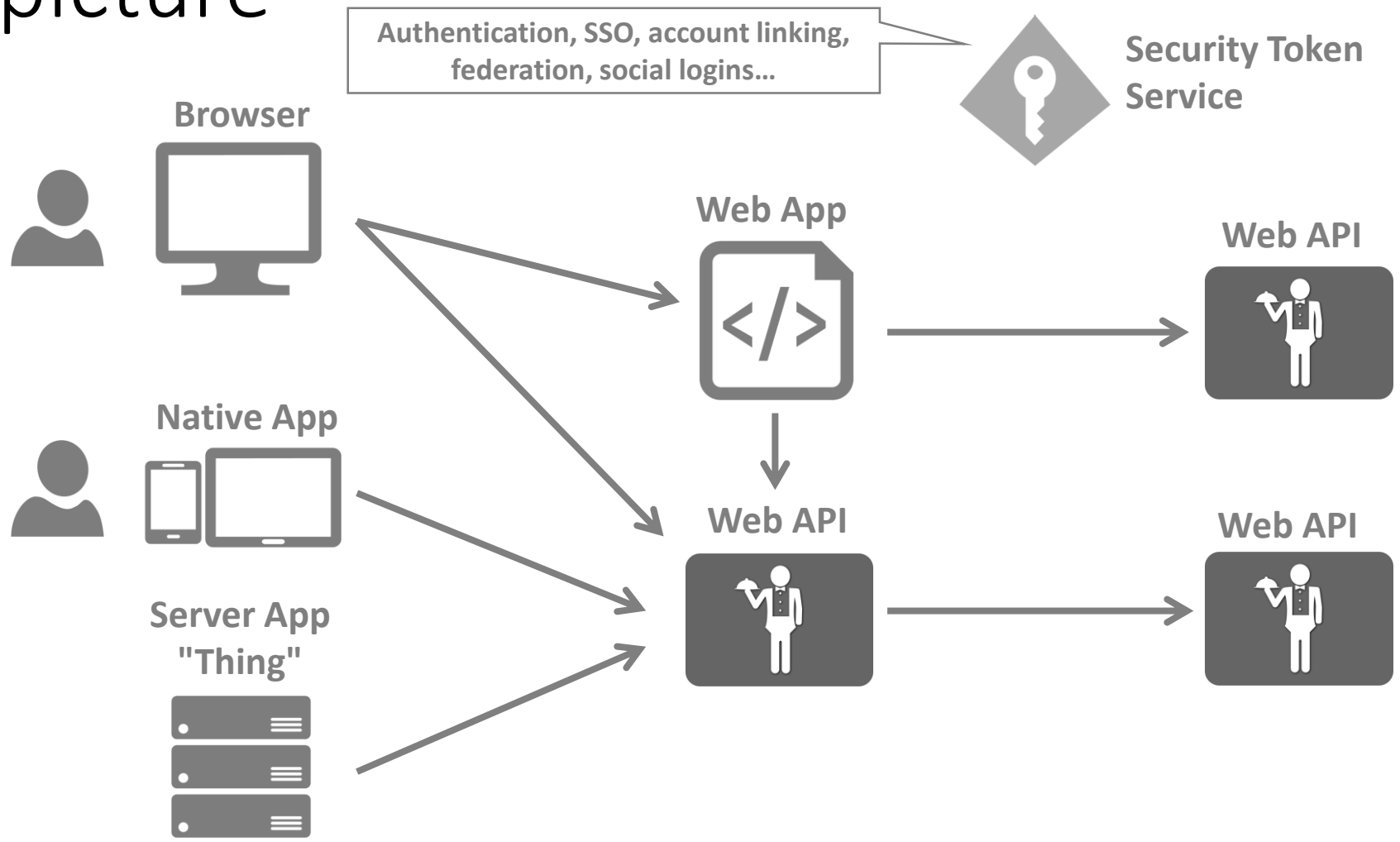
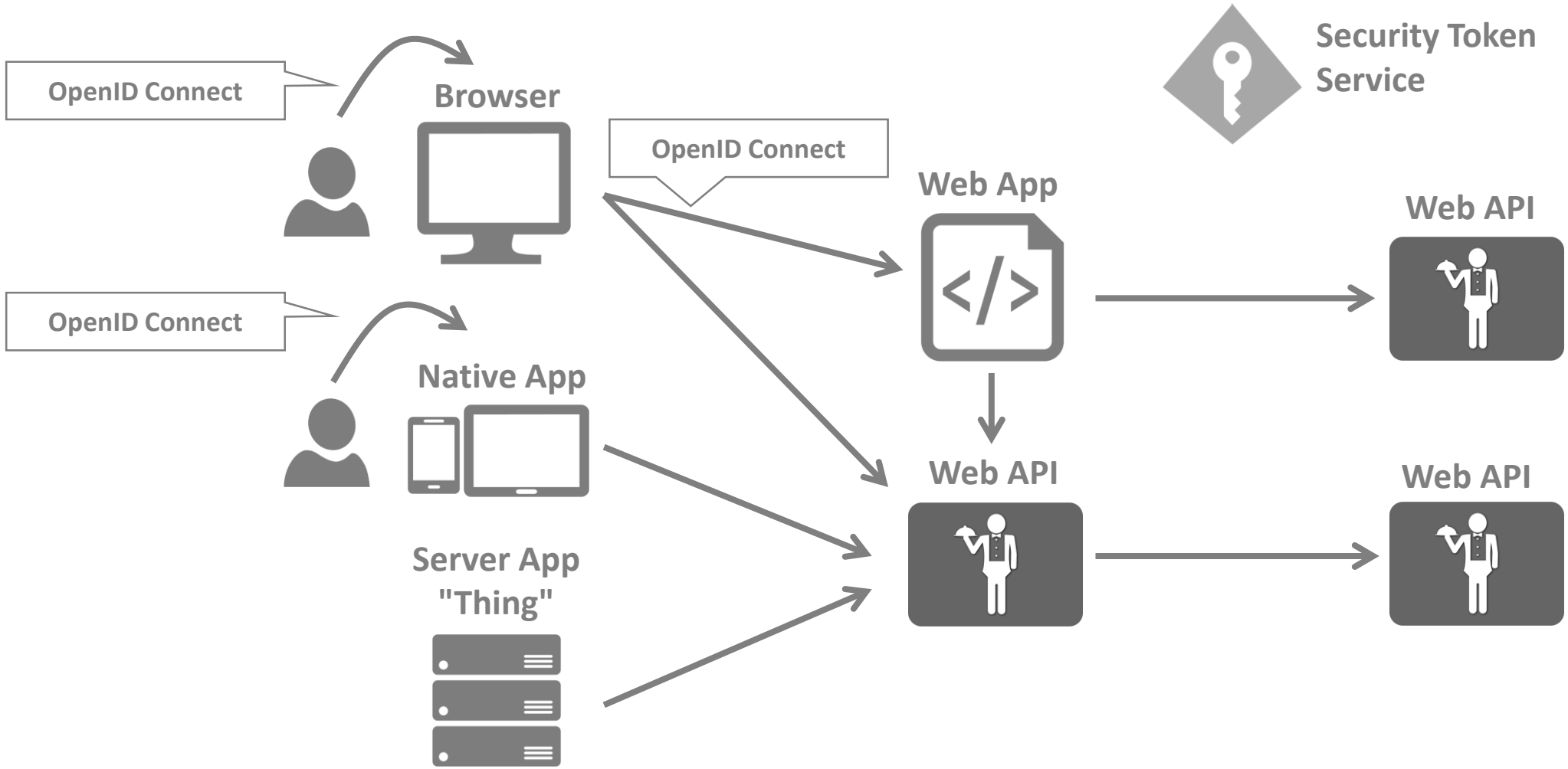
Securing APIs with ASP.NET Core 3, Visual Studio 2019, and IdentityServer4

Brock Allen
brockallen@gmail.com
http://brockallen.com
@BrockLAllen

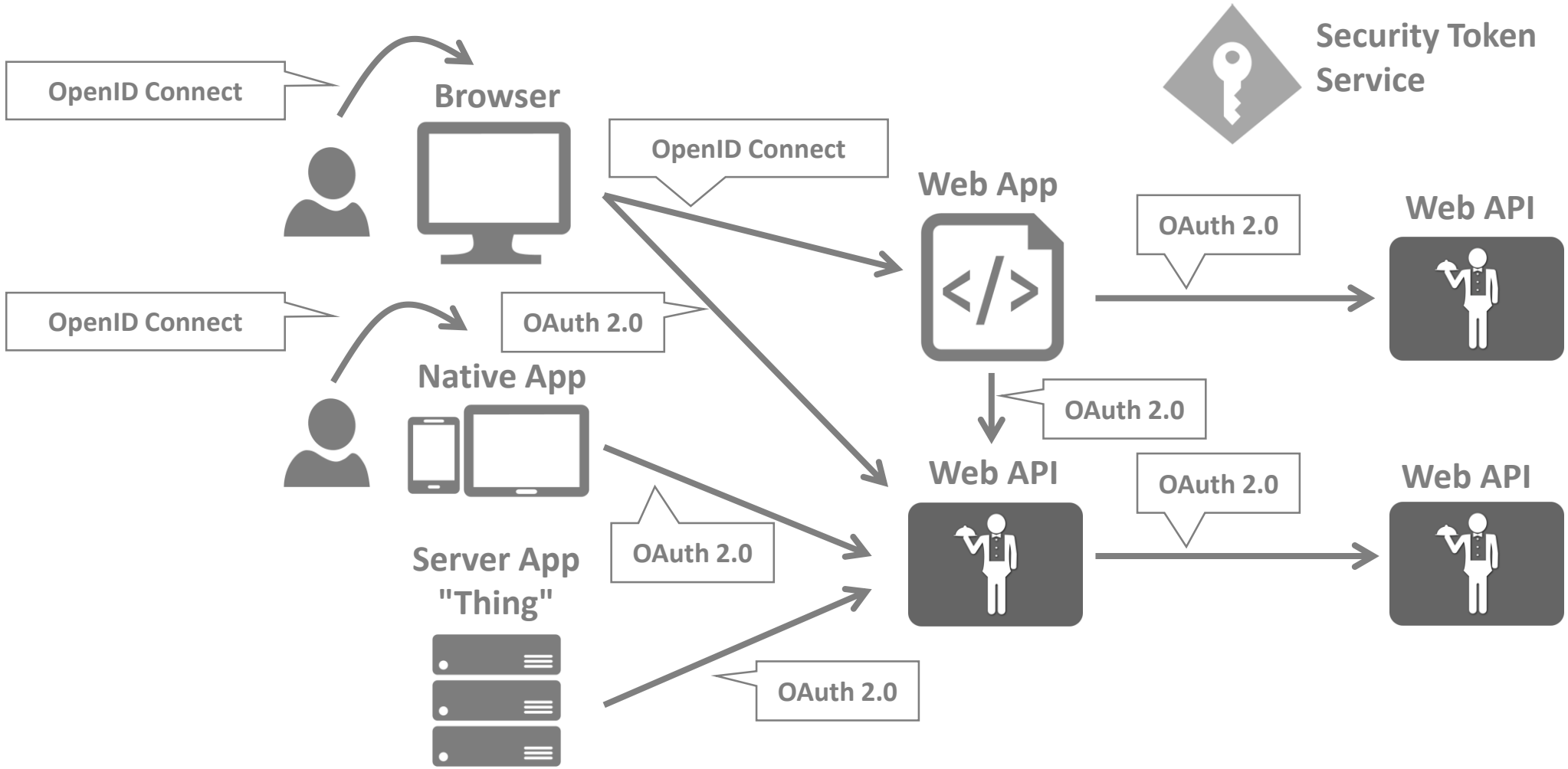
Big picture


Authentication, SSO, account linking, federation, social logins…
Security Token Service
Browser
Native App
Server App "Thing"
Web App
Web API
Web API
Web API

OIDC – Authentication & Session Management

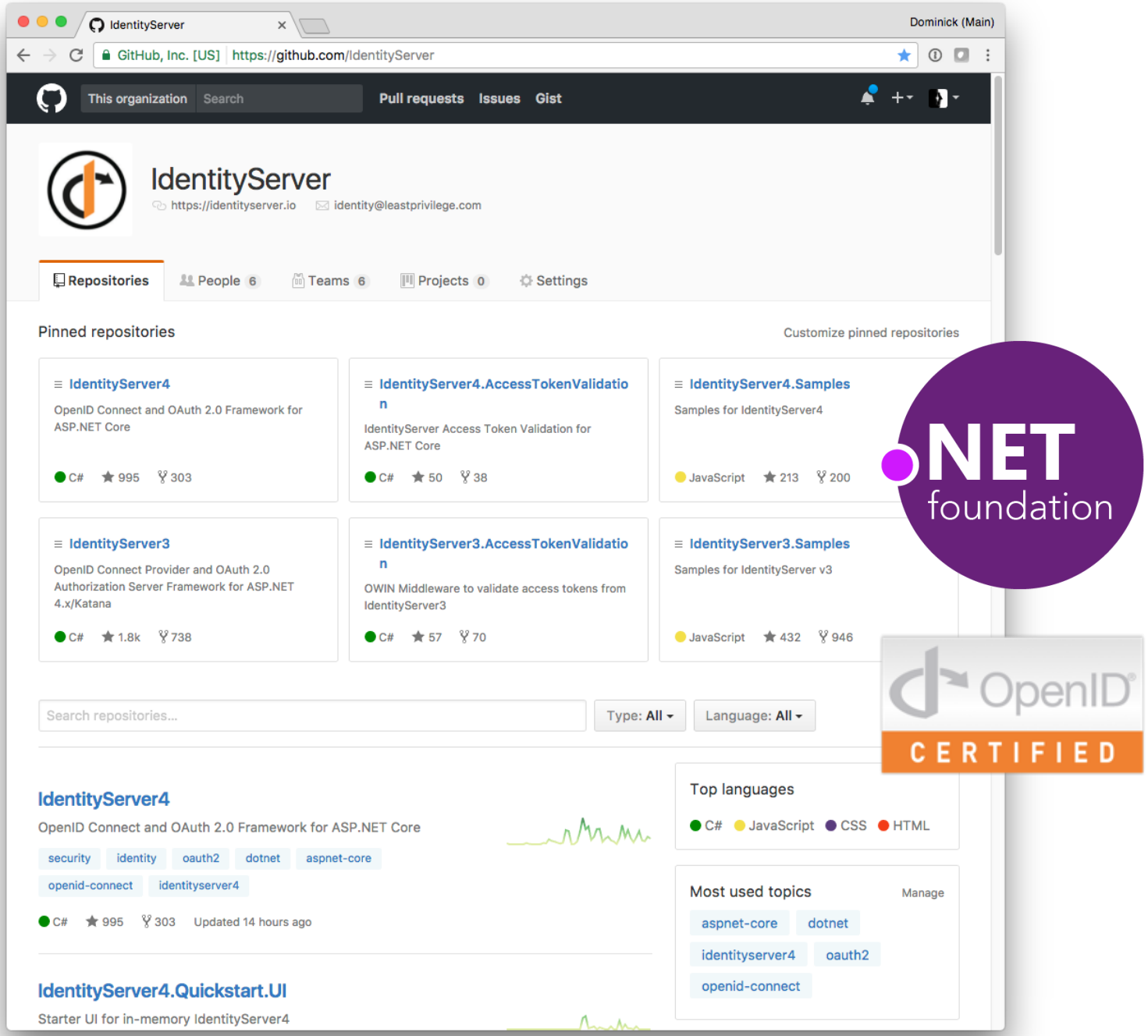


OAuth 2.0 – API Access

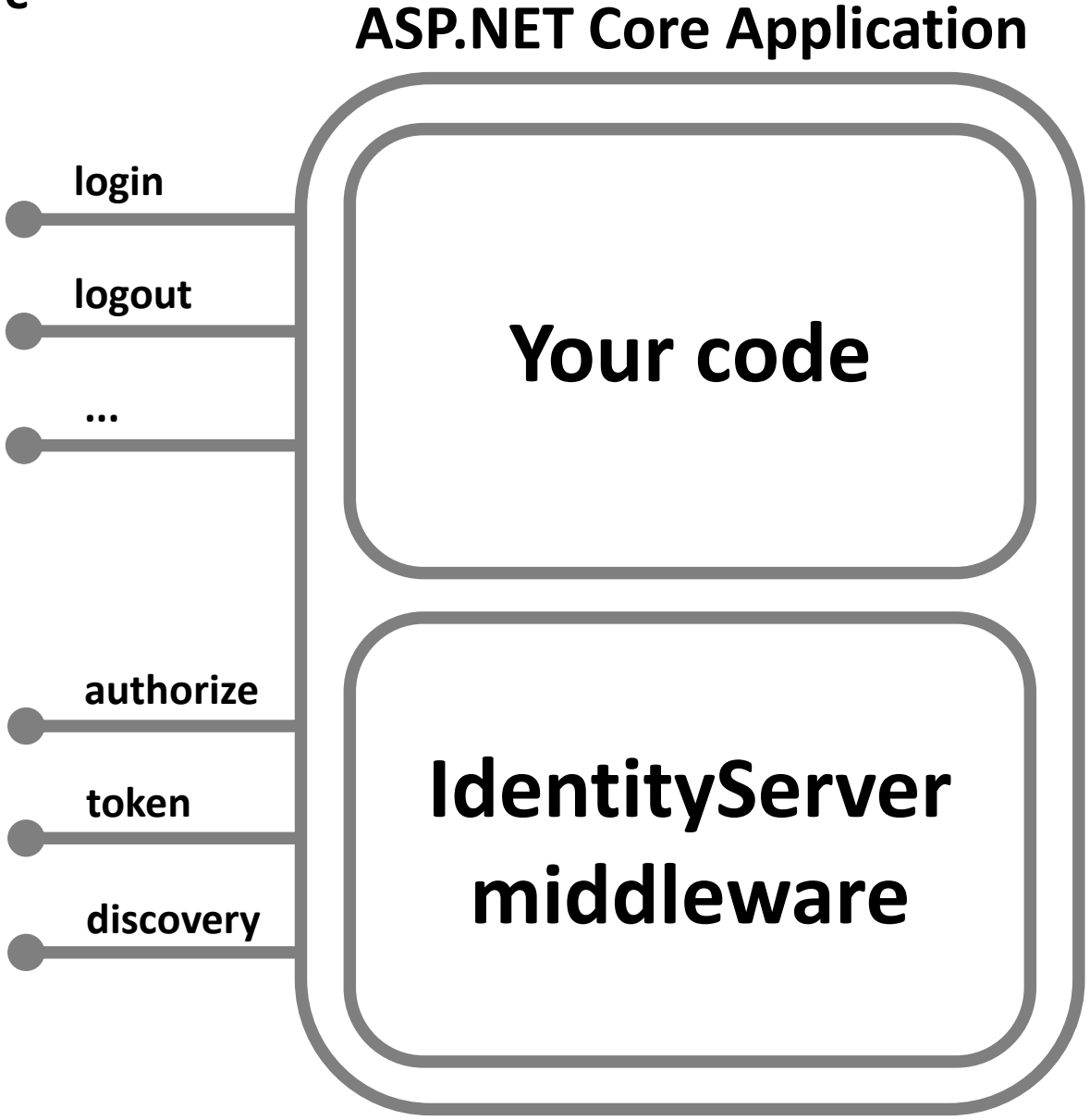


OpenID Connect and OAuth 2.0

- Protocols for obtaining and using tokens
- Allows for authentication to client application
  - With ***id_token***
- Allows for securing server APIs
  - With ***access_token***


IdentityServer
Dominick (Main)
GitHub, Inc. [US] https://github.com/IdentityServer
This organization
Search
Pull requests
Issues
Gist
IdentityServer
https://identityserver.io
identity@leastprivilege.com
Repositories
People 6
Teams 6
Projects 0
Settings
Pinned repositories
Customize pinned repositories
IdentityServer4
OpenID Connect and OAuth 2.0 Framework for ASP.NET Core
C# 995 303
IdentityServer4.AccessTokenValidation
IdentityServer Access Token Validation for ASP.NET Core
C# 50 38
IdentityServer4.Samples
Samples for IdentityServer4
JavaScript 213 200
IdentityServer3
OpenID Connect Provider and OAuth 2.0 Authorization Server Framework for ASP.NET 4.x/Katana
C# 1.8k 738
IdentityServer3.AccessTokenValidation
OWIN Middleware to validate access tokens from IdentityServer3
C# 57 70
IdentityServer3.Samples
Samples for IdentityServer v3
JavaScript 432 946
Search repositories...
Type: All
Language: All
IdentityServer4
OpenID Connect and OAuth 2.0 Framework for ASP.NET Core
security identity oauth2 dotnet aspnet-core openid-connect identityserver4
C# 995 303 Updated 14 hours ago
IdentityServer4.Quickstart.UI
Starter UI for in-memory IdentityServer4
Top languages
C# JavaScript CSS HTML
Most used topics
Manage
aspnet-core dotnet identityserver4 oauth2 openid-connect
.NET foundation
OpenID
CERTIFIED

IdentityServer Host

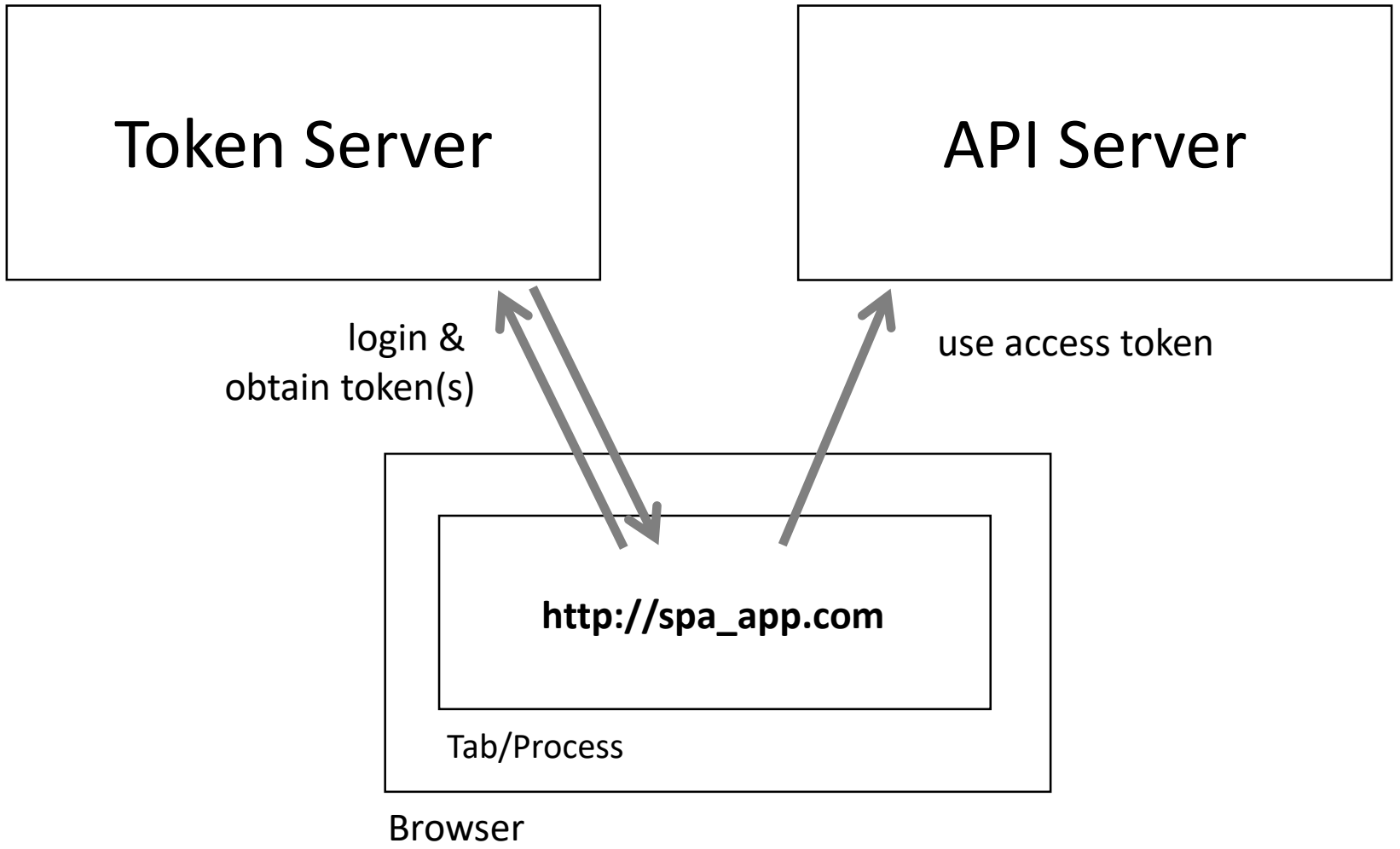


Token based API authentication

- Tokens are a form of credential
  - Typically use JSON web token (JWT)
  - Sent as “Authorization” header on every HTTP request
  - Provides a solution to CSRF
- Tokens help solve architectural issues
  - More than just browser-based apps can use tokens
  - API is client agnostic
  - No cookie management needed in API
  - Can call APIs cross-domain
  - SSO for users

JWT access token

Client, token server, and API



Protocol flows

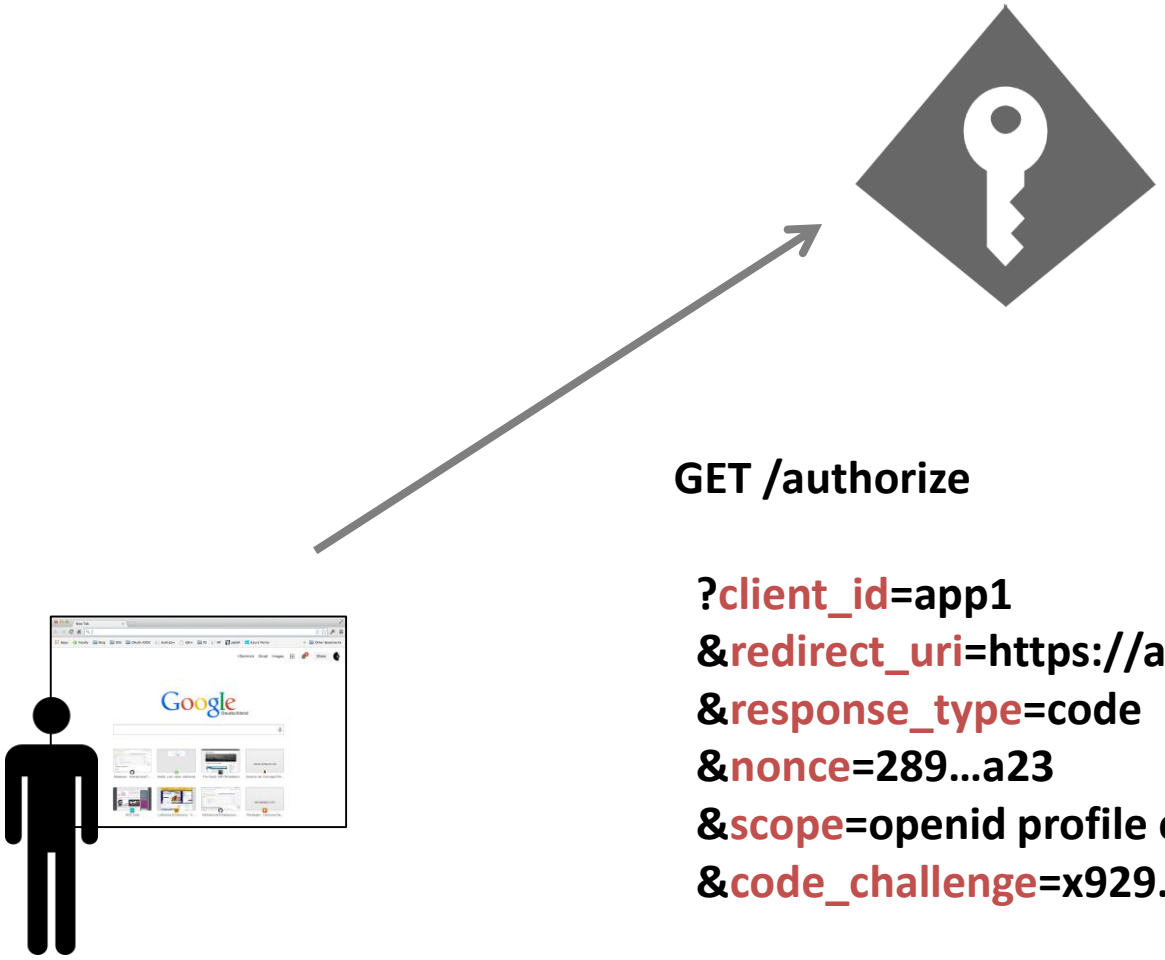
- Flows define mechanics to obtain tokens in client
  - Different based on type of app (e.g. server, mobile, SPA)
- Ongoing work in IETF to produce useful guidance
  - https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps
- SPA apps use **authorization code flow** (**with PKCE**)
  - Previous guidance was to use implicit flow

Token server endpoints

Authorize Endpoint

Token Endpoint

Authorization request



GET /authorize

?client_id=app1
&redirect_uri=https://app.com/cb.html
&response_type=code
&nonce=289…a23
&scope=openid profile email api1 api2
&code_challenge=x929…1921

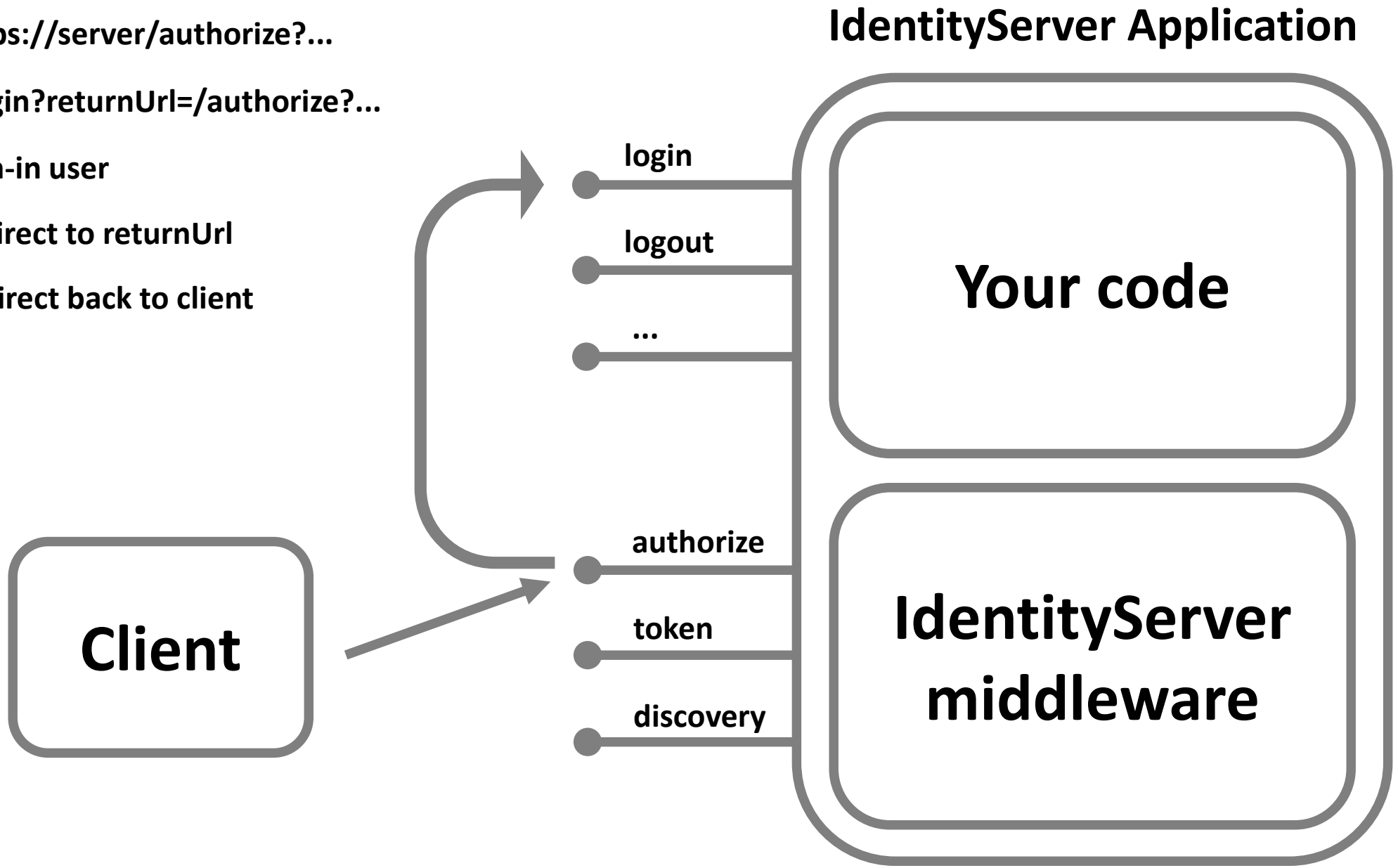
1) https://server/authorize?...

2) /login?returnUrl=/authorize?...

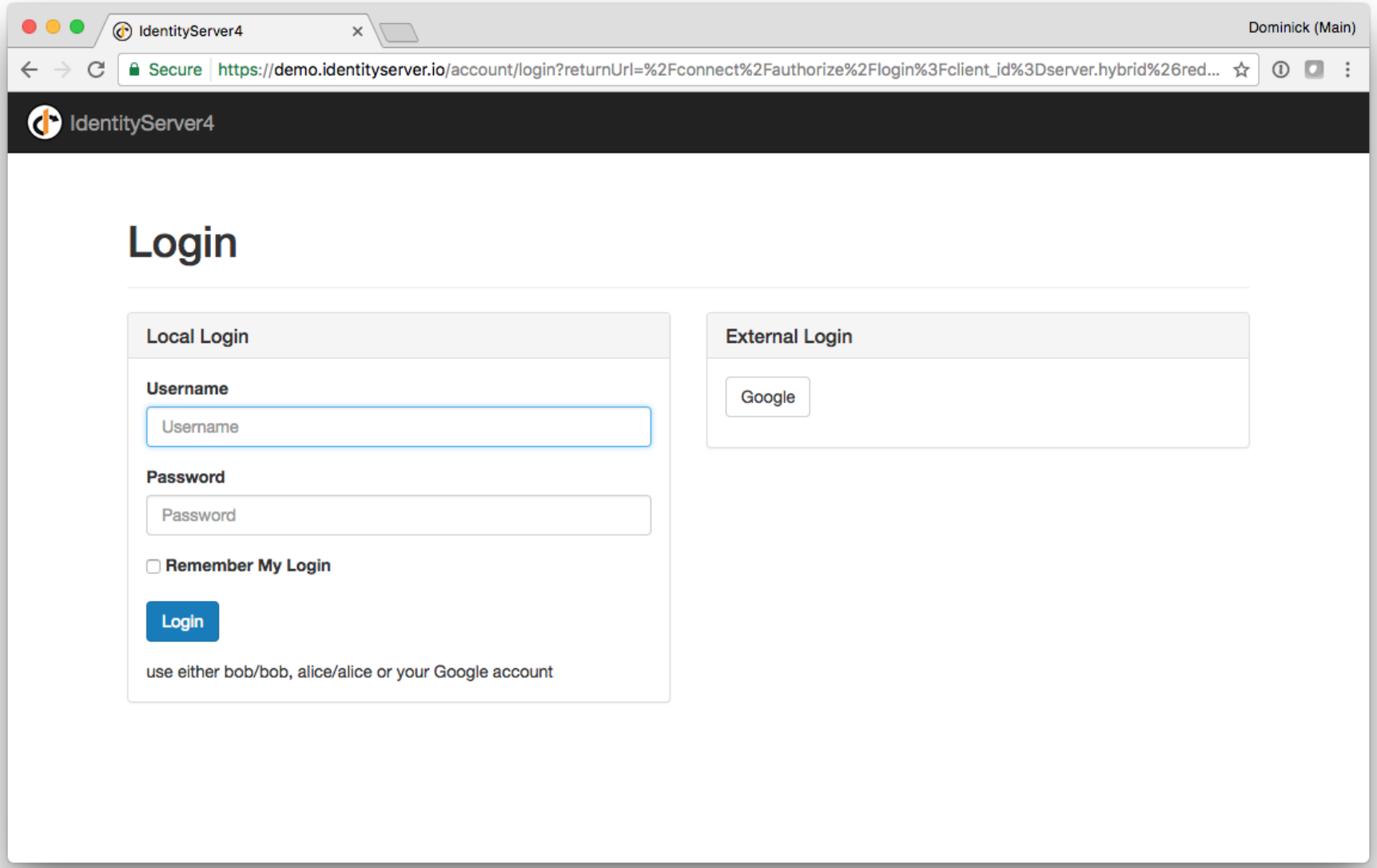
3) sign-in user

4) redirect to returnUrl

5) redirect back to client



Authentication



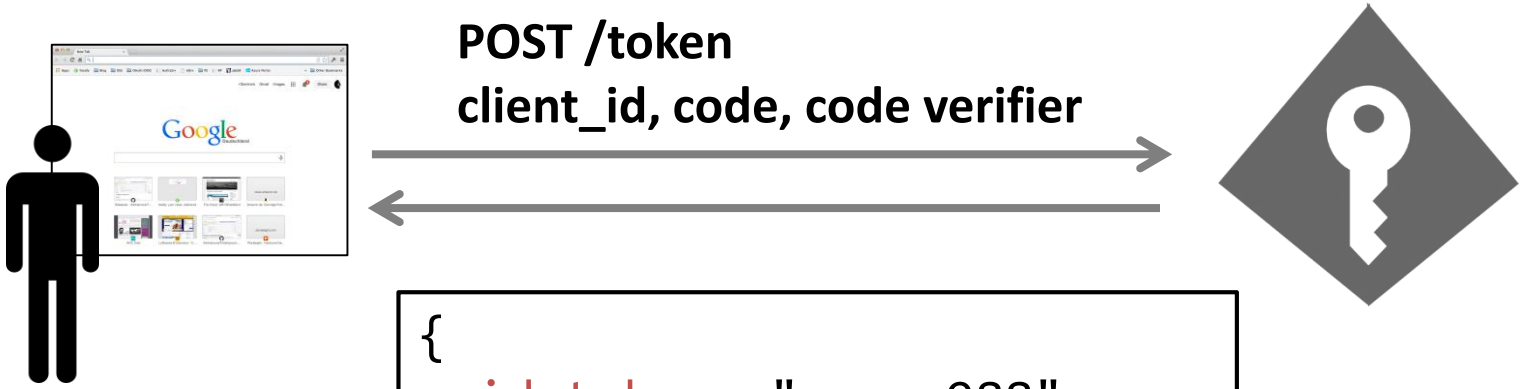
Authorization response

set SSO cookie

GET https://app.com/cb.html?code=238…823j

Token endpoint exchange

- Ajax request made to token endpoint to exchange code for tokens
  - Using client id and code verifier




```
{
  id_token: "xxae…988",
  access_token: "xyz…123",
  expires_in: 3600,
  token_type: "Bearer"
}
```

Java Script Client Library

- https://github.com/IdentityModel/oidc-client-js

```
var settings = {
    authority: 'http://localhost:5152/',
    client_id: 'spa',
    redirect_uri: 'http://localhost:5152/callback.html',
    response_type: 'code',
    scope: 'openid profile api',
};

var mgr = new Oidc.UserManager(settings);

mgr.getUser().then(function (user) {
    if (user) {
        log("logged in", user);
    }
    else {
        mgr.signinRedirect();
    }
});
```

Using access token to call APIs

- Access token passed as ***Authorization*** HTTP request header
  - Using “Bearer” scheme

```
var xhr = new XMLHttpRequest();
xhr.onload = function () {
    var user_profile = JSON.parse(xhr.response);
}

xhr.open("GET", "https://api.app.com/some_endpoint");
xhr.setRequestHeader("Authorization", "Bearer " + access_token);
xhr.send();
```

Validating access tokens at API

- ASP.NET Core provides JWT bearer authentication handler
  - Populates HttpContext.User with claims from token

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication("Bearer")
        .AddJwtBearer("Bearer", options =>
        {
            options.Authority = "https://identityserver.io";
            options.Audience = "your_api_identifier";
        });
}
```

Visual Studio 2019/ASP.NET Core 3 SPA Template

- All in-one project that contains:
  - SPA App Assets (Angular or React)
    - Provides wrapper on oidc-client-js
  - ASP.NET Identity UI
    - For user credential management
  - IdentityServer4 middleware/services
    - Simple configuration abstraction over full IdentityServer4 configuration system
  - Web API
    - Configured to trust tokens from the co-hosted IdentityServer4 instance

IdentityServer logging

- IdentityServer logs lots of information
  - You will need it while developing
- Enable in appsettings.json

```
"Logging": {
    "LogLevel": {
        "Default": "Warning",
        "IdentityServer4": "Debug"
    }
}
```

Configuring new APIs

- “Resources” section in appsettings.json
  - Key is name of API
  - “Profile” indicates:
    - “IdentityServerJwt“ co-hosted with IdentityServer host
    - “API” hosted elsewhere

```
"IdentityServer": {
    "Resources": {
        "MyApi": {
            "Profile" : "API"
        }
    }
}
```

Configuring new SPA clients

- “Clients” section in appsettings.json
  - Key is name of SPA app
  - “Profile” indicates:
    - “IdentityServerSPA“ co-hosted with IdentityServer host
    - “SPA” hosted elsewhere
      - “RedirectUri” and “LogoutUri” needed

```
"IdentityServer": {
    "Clients": {
        "MySPA": {
            "Profile" : “SPA",
            "RedirectUri": "https://localhost:5005/callback.html",
            "LogoutUri": "https://localhost:5005/index.html"
        }
    }
}
```

Configuring other clients and resources

- ASP.NET Core's abstraction targets SPAs
  - Can use IdentityServer configuration system for more control
  - Clients, Identity Resources, and API Resources

```
services.AddIdentityServer()
    .AddApiAuthorization<ApplicationUser, ApplicationDbContext>(options =>
    {
        options.Clients.Add(new Client
        {
            ClientId = "MyCustomClient",
            AllowedGrantTypes = GrantTypes.ClientCredentials,
            ClientSecrets = { new Secret("sha256_hash_of_the_real_secret") },
            AllowedScopes = { "MyAPI" }
        });
});
```

Configuring signing keys

- Signing keys required when moving to production
  - Certificate commonly used as key to sign tokens
- “Key” section in appsettings.json
  - “Type” supports:
    - “Development”, “File”, or “Store”

```
"IdentityServer": {
  "Key": {
      "Type": "File",
      "FilePath": "C:\\demos\\key.pfx",
      "Password": "password"
  }
}
```

```
"IdentityServer": {
    "Key": {
         "Type": "Store",
         "StoreLocation": "LocalMachine",
         "StoreName": "My",
         "Name":"CN=sts"
    }
}
```

Summary

- Token based security is the modern approach for apps and APIs
- OpenID Connect and OAuth 2.0 are the protocols for getting tokens
- IdentityServer4 is a framework for building a custom token server
- SPA templates in Visual Studio provide a good starting point