SPA OAuth Demo

AngularJS + WebApi

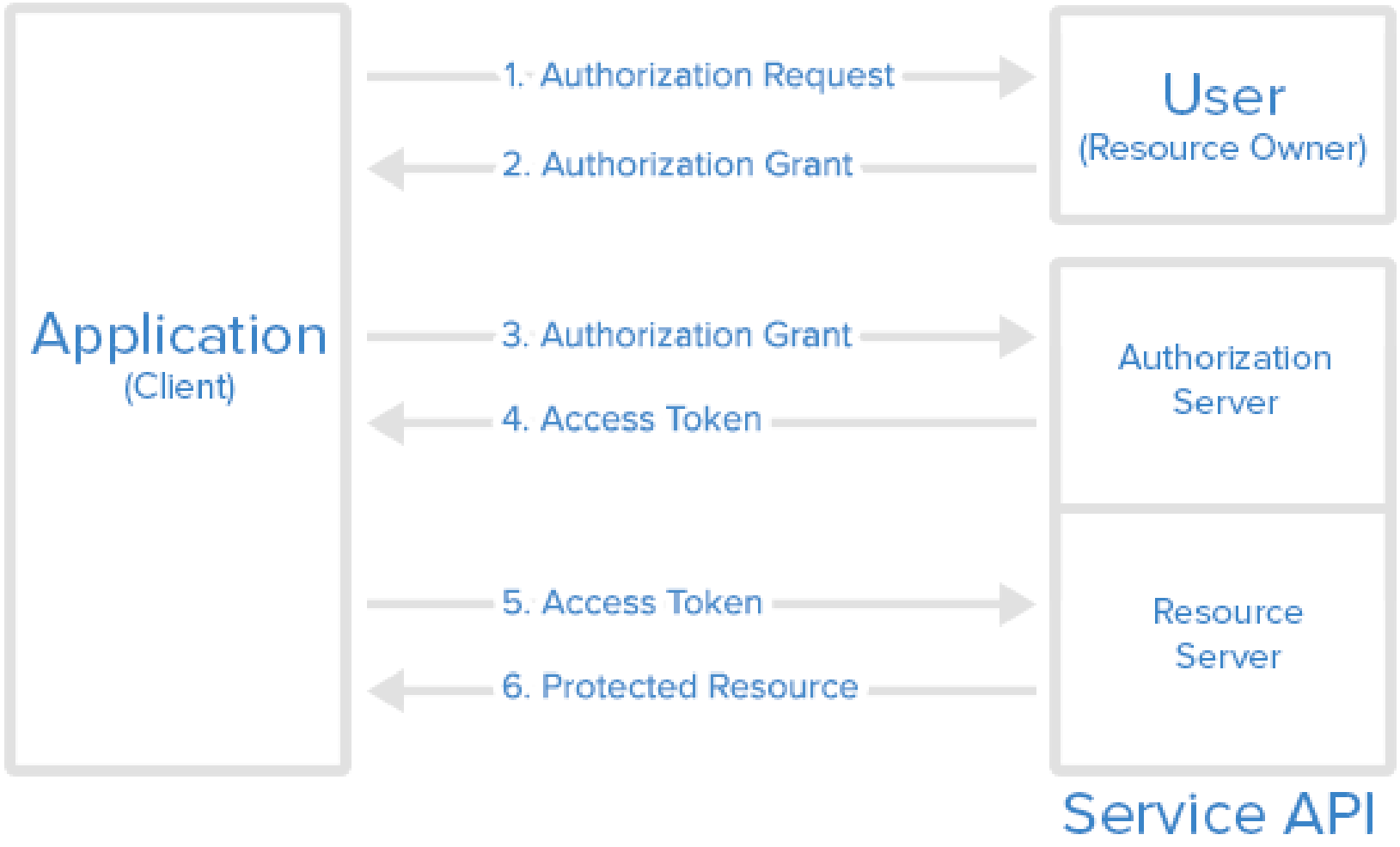
- **SPA**: Single Page Application
- **OAuth**: OAuth is an open standard for authorization. OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials.

OAuth Flows

There is one OAuth spec describing several flows

This demo covers a password grant flow which assumes a trust relationship between the client, auth server, and api (resource server), meaning both the client and server are within the same domain (not your typical oauth scenario)

Abstract Protocol Flow



OAuth 2.0 (Implicit Flow)

"The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly"

Resource Owner Password Credentials Flow

"The resource owner password credentials (i.e., username and password) can be used directly as an authorization grant to obtain an access token. The credentials should only be used when there is a high degree of trust between the resource owner and the client (e.g., the client is part of the device operating system or a highly privileged application), and when other authorization grant types are not available (such as an authorization code). Even though this grant type requires direct client access to the resource owner credentials, the resource owner credentials are used for a single request and are exchanged for an access token. This grant type can eliminate the need for the client to store the resource owner credentials for future use, by exchanging the credentials with a long-lived access token or refresh token."

Access Tokens

"Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server."

Refresh Tokens

"Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner)."

OWIN

- ***OWIN*** *(*Open Web Server Interface for .NET) *defines a standard interface between .NET web servers and web applications. The goal of the OWIN interface is to decouple server and application, encourage the development of simple modules for .NET web development, and, by being an open standard, stimulate the open source ecosystem of .NET web development tools.*
- **Katana** is the Microsoft implementation of the OWIN specs, and provides all the layers, sometimes in more than one flavor, specified by OWIN. In addition to implementing hosts and servers, Katana provides a series of APIs to facilitate the development of OWIN applications, including some functional components like authentication, diagnostics, static files serving, and bindings for ASP.NET Web API and SignalR. To avoid confusion, remember that Katana is not a full-fledged web server, but just the “glue” between the OWIN world and IIS.

OWIN

- Meant to be OS independent and can self-host. With OWIN, your code is not related to the OS (specifically to System.Web, the “huge” monolithic library that lies behind the execution of ASP.NET). This means that you can use whatever you want instead of IIS (i.e. Katana or Nowin) and update it when necessary, instead of updating the OS. Moreover, if you need it, you can build your custom host and insert whatever you want in the HTTP request processing pipeline (i.e. your custom authentication logic).
- OAuth is an OWIN middleware component

Setting up ASP.NET Web API with Owin

- Create a new **ASP.NET Web Application**, choose the **Empty** template, and tick the **Web API** option under "Add folders and core references for": this will install all the Nuget packages needed for a Web API project, and will setup the folder structure;
- Install the Owin packages and the Owin-Web API "bridge": by installing the **Microsoft.AspNet.WebApi.Owin** you'll get everything you need;
- Install **Microsoft.Owin.Host.SystemWeb** to run the within IIS.
- Configure the Owin Startup class to fire up Web API: just add a **OWIN Startup class** from Visual Studio contextual menu and add to the Configuration method the right configuration for Web API.

Setting Up OAuth Authorization Server

- Register OWIN OAuth middleware

```
app.UseOAuthAuthorizationServer
app.UseOAuthBearerAuthentication
```

- Define options for token format, expiration
- Setup endpoint to receive authorization grant

Installing Angular

- bower install angular (preferably Angular 1.4+)

Authentication Flow

- Both the API (resource server) and authorization server are owned by the same company and are trusted.
- Authentication over HTTPS to public client
- Upon login, user/password and client id is sent to auth server and access token is returned. Access token in HTML5 local storage.
- Client id is used to validate the user.
- Requests are made to the API with the access token in the header.
- When an access token expires, a new one is generated.

OAuth Access Tokens

- Encrypt Bearer Tokens
- Bearer Tokens must be short lived (several hours to days)
- Don't pass in urls, put in header
- Refresh tokens periodically
- Validate SSL Certs

Third Party OAuth Solutions

- **IdentityServer3**: https://github.com/IdentityServer/IdentityServer3
- **Auth0**: https://auth0.com

Tips and Tricks

- Prevent hot linking of sensitive images by returning a Base64 string and placing it has a background-image on a div. Authorize the request using [AuthorizeAttribute].
- If writing your own auth server
  - validate requests for access tokens and refresh tokens on a database
  - Include the option to disable client ids or users if compromised.
  - Use strong encryptions such as rijndael. Asynmetric keys are also possible but difficult to do it, just like writing your own auth server.

Useful Links

- **OAuth 2.0 specs**: https://tools.ietf.org/html/rfc6749
- **Oauth Security:** http://www.oauthsecurity.com
- **Oauth Bible**: http://authbible.com
- **Persisting Refresh Token**: http://timney.net/persisting-your-refresh-tokens
- **OAuth Resource Password Flow Refresh Token with Web Api**: http://timney.net/oauth-resource-password-flow-refresh-token-with-web-api
- **OAuth Resource Password Flow with Web Api**: http://timney.net/oauth-resource-password-flow-with-web-api
- **OAuth 2.0 Threat Model**: http://tools.ietf.org/html/rfc6819
- **Beginner's Guide to OAuth**: http://oauth.net/documentation/getting-started
- **Intro to OAuth2**: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2