## Oauth2.0

W rozdziale tym zostanie poruszona kwestia autoryzacji oraz autentykacji ze szczególnym naciskiem na omówienie standardu Oauth 2.0 oraz jego implementacji w technologii .Net. Pokazane zostanie również, jak taka implementacja współgra z pozostałymi modułami oraz całą architekturą aplikacji.

Oauth jest to standard autoryzacji, który pozwala uzyskać ograniczony dostęp zasobów użytkownika, poprzez delegowania operacji autoryzacji do trzeciego komponentu, przechowującego informacje o koncie użytkownika. Komponent ten po otrzymaniu odpowiednich informacji w czasie logowania, mogą to być przykładowo login i hasło, zwraca zaszyfrowany token. Następnie token ten, służy jako przepustka do wybranych przez nas zasobów. Token ma wcześniej zdefiniowany czas życia, po którego upływie wygasa i nie da się już za jego zautoryzować. Dzięki standardowi oauth można wykorzystać konta takich serwisów jak Facebook, Google czy GitHub do autentykacji w danym systemie oraz uzyskać wybrane dane, które zostały udostępnione przez te serwisy.

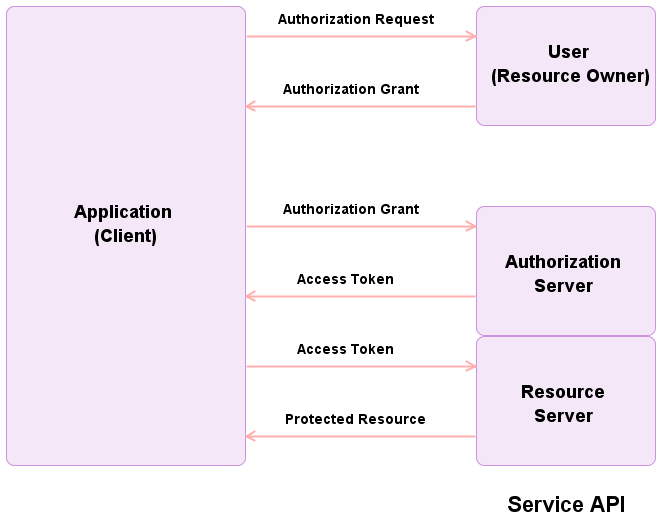
Standard ten posiada kilka ról, które odpowiadają za konkretny fragment działania całego systemu autentykacji i autoryzacji. Wyróżniamy takie pojęcia jak Resuource Owner, Client, Resource Server oraz Authorization Server. W zależności od implementacji, dany fragment aplikacji może pełnić odrębną rolę bądź kilka ról na raz. Oba podejścia są zgodną z samą ideą standardu Oauth. Poniżej zostanie przybliżone ogólne znaczenie każdej z ról.

Resource Owner, czyli właściciel zasobu, jest to użytkownik, który wysyła żądanie o dostęp do własnego konta poprzez aplikację. Zakres dostępu do zasobów danego konta jest ograniczony do zezwoleń, na jakie zdecydował się użytkownik.

Resource Server zawiera zasoby, z których dany użytkownik chce skorzystać. Mogą to być zarówno dane jak i różnorakie funkcjonalności świadczone przez aplikację. Authorization Server weryfikuje tożsamość użytkownika, a następnie przydziela tokeny dostępu do zasobów.

W przypadku omawianej aplikacji obie role, zostały zaimplementowane jako jedno api, na jednym serwerze, posiadające odpowiednie metody wystawiające zarówno zasoby jak i pozwalające na autentykacje.

Client jest aplikacją, która chce uzyskać dostęp do konta użytkownika oraz jego zasobów. Zanim będzie mogła to uczynić, musi to zostać potwierdzone przez użytkownika. To potwierdzenie musi zostać zweryfikowane poprzez api służące to autentykacji, które następnie wystawi przepustkę w postaci tokenu. Token ten pozwoli aplikacji na dostęp do danego zasobu posiadanego przez użytkownika.



Rysunek . Diagram przedstawiający przypływ danych pomiędzy rolami w OAuth

Powyższy diagram obrazuje przepływ kolejnych wiadomości w czasie pracy z Oauth2.0. Oczywiście wiele zależy od konkretnej implementacji, jednak większość z nich trzyma się ram, nakreślonych przez ten schemat. Aplikacja początkowo prosi użytkownika od dostęp do jego zasobów. Jeżeli użytkownik zezwolił na taki dostęp, w odpowiedzi odsyła Authorization Grant. Następnie aplikacja żąda wystawienia tokenu, przez serwer autoryzacyjny. W żądaniu tym znajduje się Authorazition Grant, wystawiany przez użytkownika. Serwer autoryzacji weryfikuje tożsamość użytkownika na podstawie przesłanego upoważnienia a następnie odsyła token, zezwalający na dostęp od zasobów. Od tego momentu, przy każdym żądaniu zasobów przez aplikacje kliencką przesyłany jest token. Na jego podstawie Resource Server weryfikuje, czy dany użytkownik może mieć do nich dostęp.

Implementacja Oauth2.0 w przestawianej aplikacji, składa się z kilku elementów. Każdy z nich pełni osobnę rolę w autentykacji oraz przyznawaniu dostępów do zasobów. Klasa ApplicationDbContext służy do przygotowania dostępu do bazy danych. Jako parametr generyczny przyjmuje ona klasę ApplicationUser, w której zdefiniowano informacje o użytkowniku.

Kolejnym elementem implementacji, jest ApplicationUserManger, który rozszerza klasę, udostępnioną przez jedną z bibliotek .Net.Identity. Zajmuje się ona zarządzaniem użytkownikami oraz ich właściwościami. To za jej pomocą, utworzymy nowego użytkownika, nadamy mu odpowiednią rolę systemową bądź uprawnienia. To również ona będzie weryfikowała, czy podany login i hasło są poprawne oraz czy przy ich pomocy, można wystawić token do autentykacji. Klasa ta nie powinna korzystać bezpośrednio z dostępu do bazy danych. Jej działanie powinno skupić się na innym poziomie abstrakcji, w oderwaniu od samego źródła danych. Operacje na ApplicationDbContext powinny być zaimplementowane w ApplicationUserStore, z które następnie będzie korzystał omawiany menadżer. Dzięki takiemu podejściu, w bardzo łatwy sposób można zmienić źródła danych o użytkownikach oraz sposób ich przetwarzania. Zmiana taka jednak nie będzie zmuszać programistów do modyfikacji logiki stojącej za autoryzacją oraz autentykacją. Jak widać cały czas wspierany jest wzorzec projektowy Dependency Injection (Wstrzykiwanie Zależności).



Kolejnym elementem tej układanki jest ApplicationOauthProvier. To on definiuje sposób, w jaki dany użytkownik ma potwierdzić swoją tożsamość. To tutaj przykładowo odbierane jest login i hasło po wysłaniu Athorization Request przez aplikację kliencką. Następnie wykorzystywany jest ApplicationUserManager w celu potwierdzeniu, czy dane przesłane do autoryzacji są poprawne. Jeżeli weryfikacja ta przejdzie pomyślnie, to właśnie ten komponent zwraca żądany token. Następnie w module konfiguracyjnym API, ustawiamy adres pod jakim adresem ma być wystawiony endpoint, który będzie odbierał dane do weryfikowania. Specyfikujemy również rodzaj Proividera, który taką weryfikacje ma przeprowadzić.



Rysunek . ApplicationOathProvider



Rysunek . Funkcja konfiguracyjna endpoint wykorzystywany do uzyskania tokenu