ENT172

Accessing OAuth 2.0 Web Services with Xamarin.Auth

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Authorize with an OAuth 2.0 server
2. Access an OAuth 2.0-secured API
3. Exchange a Refresh Token for a new Access Token

Authorize with an OAuth 2.0 server

Xamarin University

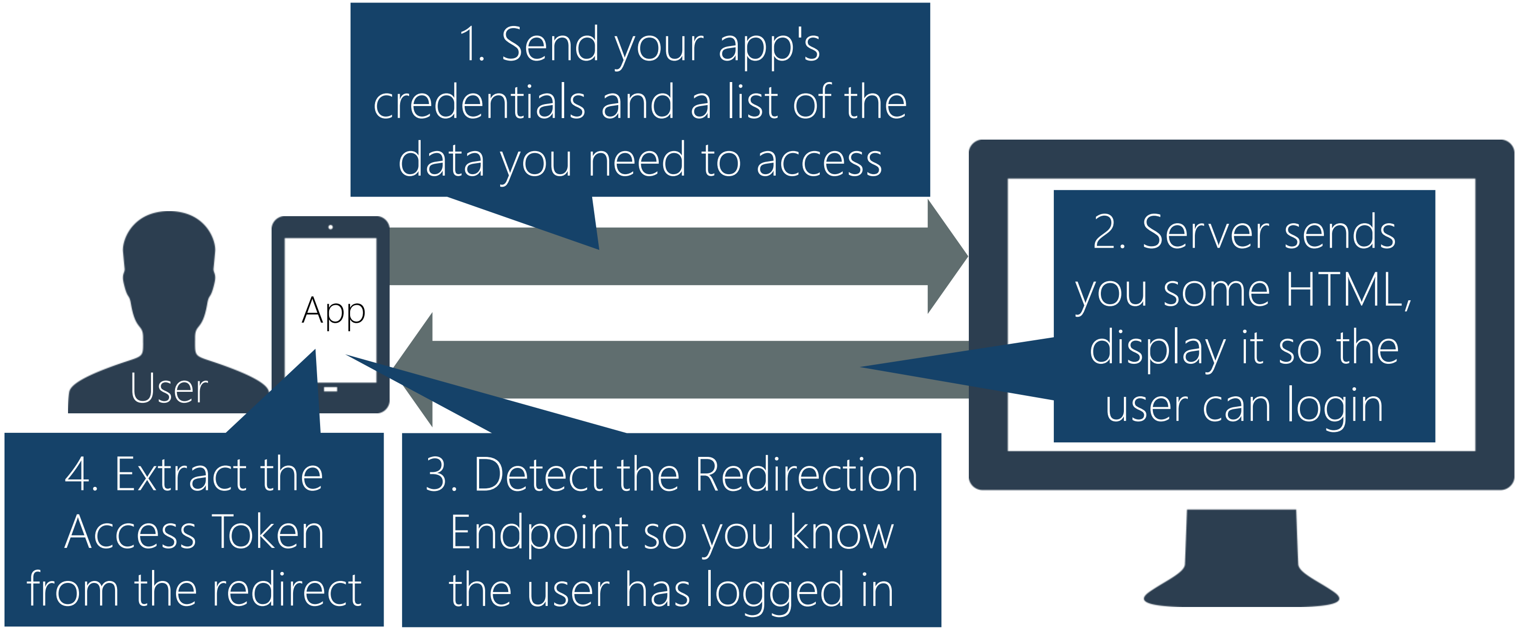
Tasks

1. Locate the values you need to send to the server for Authentication and Authorization
2. Authenticate and Authorize using Xamarin.Auth

Xamarin University

Motivation

❖ Authenticating and Authorizing a user manually is a tedious process



1. Send your app's credentials and a list of the data you need to access
2. Server sends you some HTML, display it so the user can login
User
App
4. Extract the Access Token from the redirect
3. Detect the Redirection Endpoint so you know the user has logged in

This shows the Implicit flow. The Authorization Code flow requires even more steps.

What is Xamarin.Auth?

- Xamarin.Auth provides client-side OAuth 2.0 validation using either the Implicit flow or the Authorization Code flow

Written by Xamarin

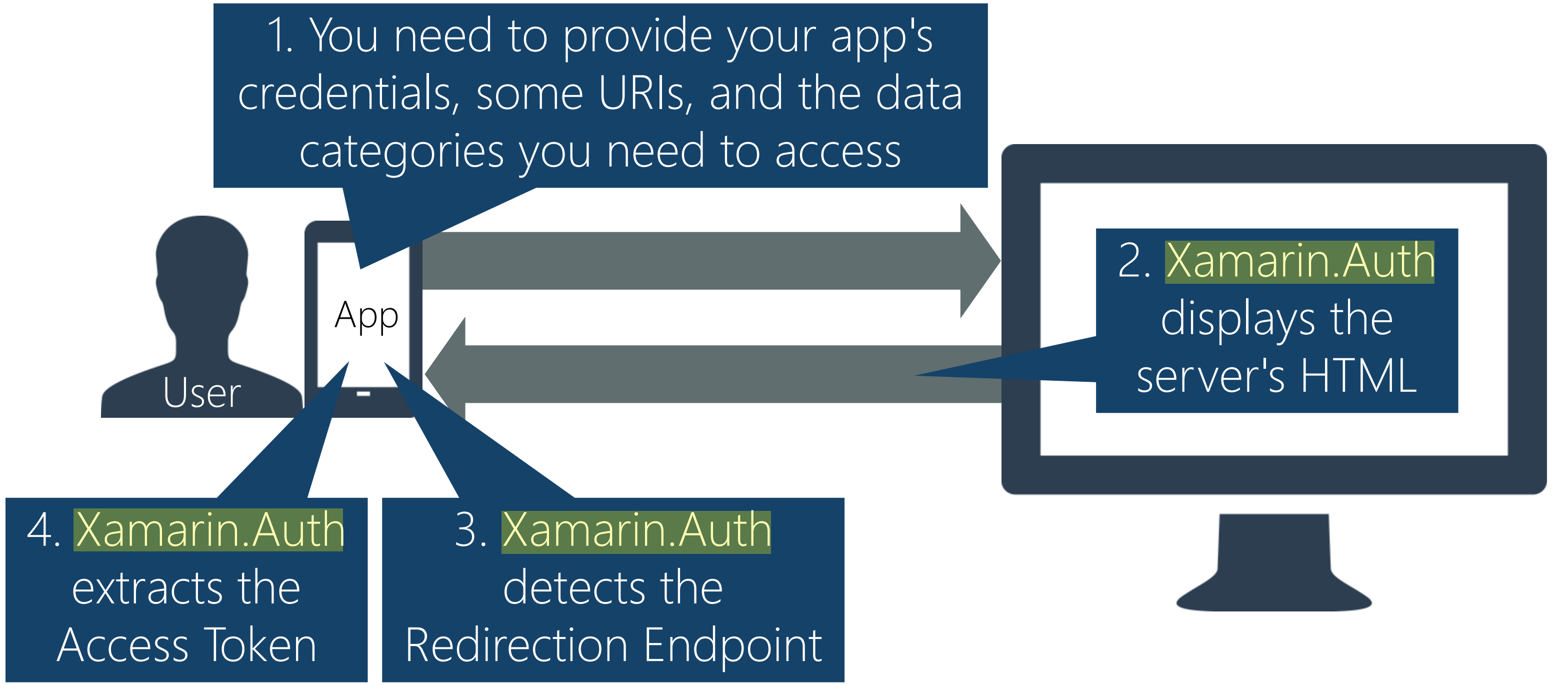
Cross platform

Available as a Component or a Nuget package

There are other OAuth plugins – choose the one that best meets your requirements

Xamarin.Auth services

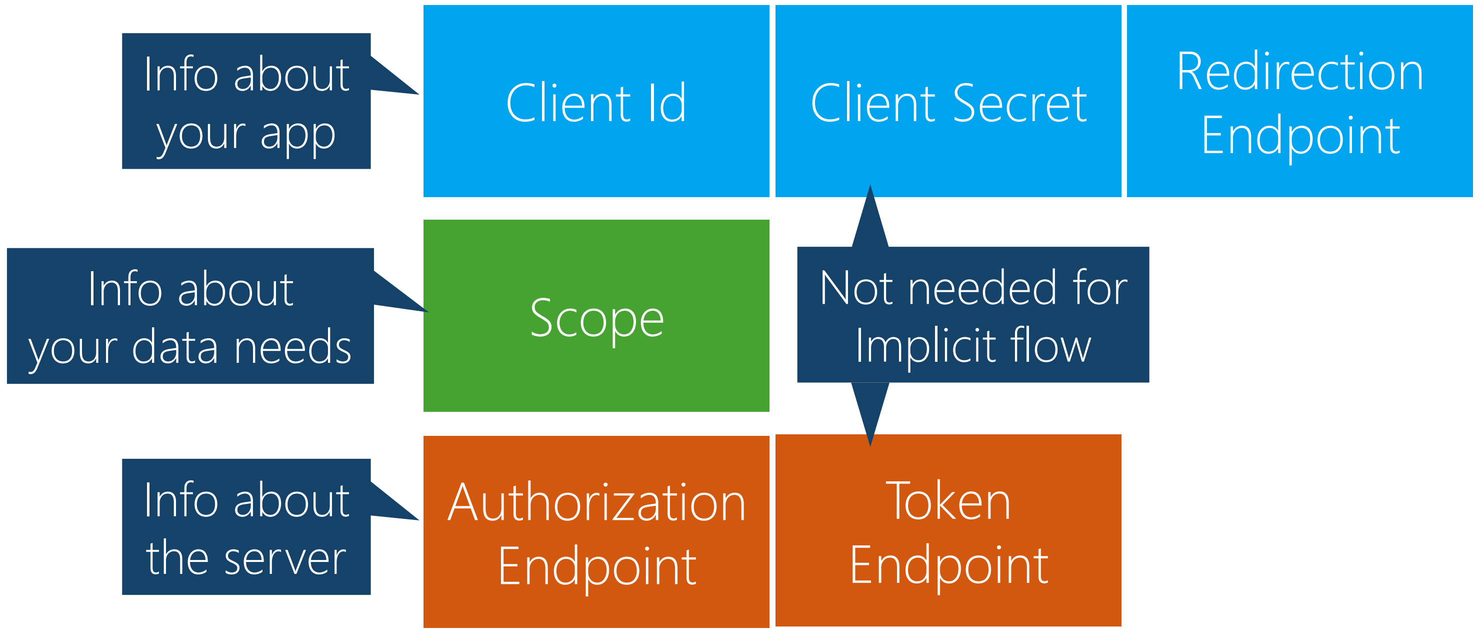
❖ Xamarin.Auth handles all the mechanics of Authentication/Authorization


1. You need to provide your app's credentials, some URIs, and the data categories you need to access
User
App
2. Xamarin.Auth displays the server's HTML
3. Xamarin.Auth detects the Redirection Endpoint
4. Xamarin.Auth extracts the Access Token

Xamarin University

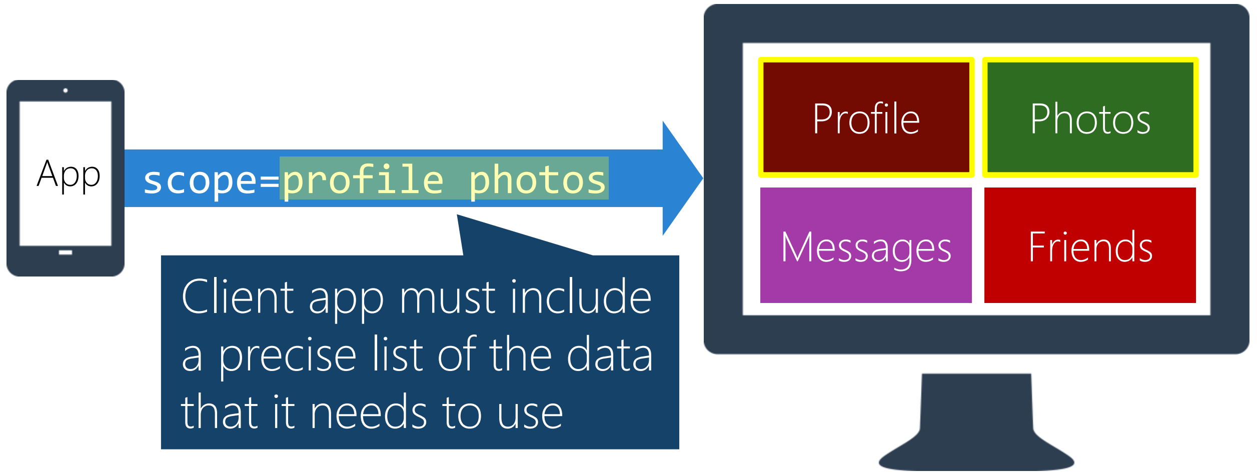
What data do you need to supply?

- You need to provide Xamarin.Auth with several pieces of information so it can perform an OAuth 2.0 flow for you



What is scope?

- The *scope* of an OAuth 2.0 authorization request describes which resources the app needs to access



Client app must include a precise list of the data that it needs to use

Xamarin University

How to determine scope

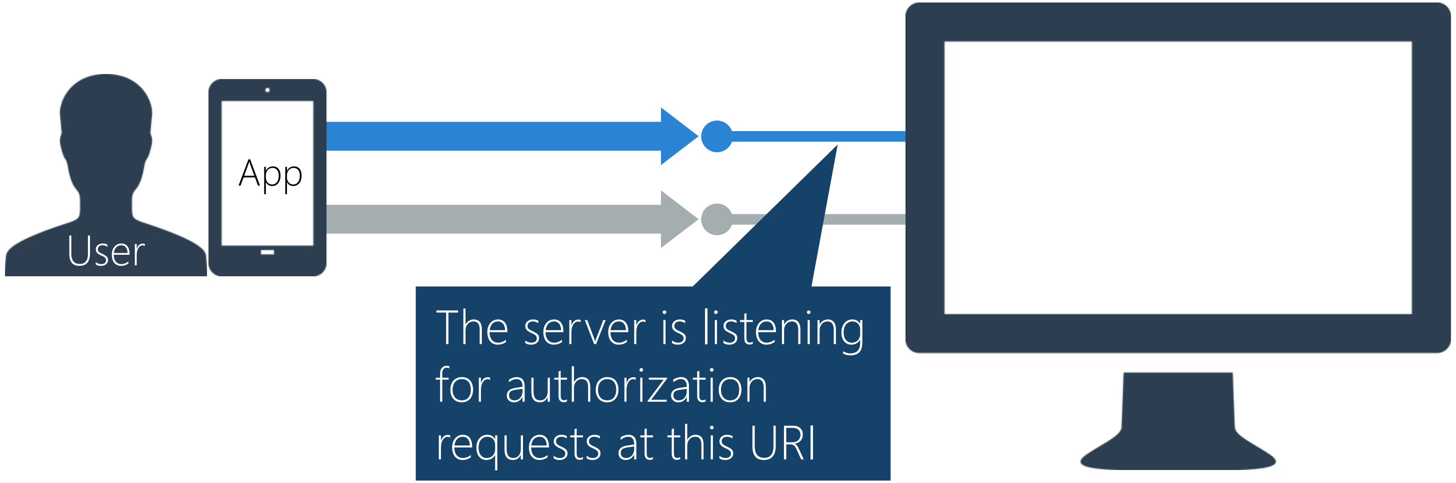
- Scope strings are not standardized by the OAuth 2.0 spec; refer to the documentation for your server for the scopes it supports

```
string Scope = "profile email";
```

The OAuth 2.0 spec says only that scope is space-delimited and case sensitive (example shows a scope for Google APIs)

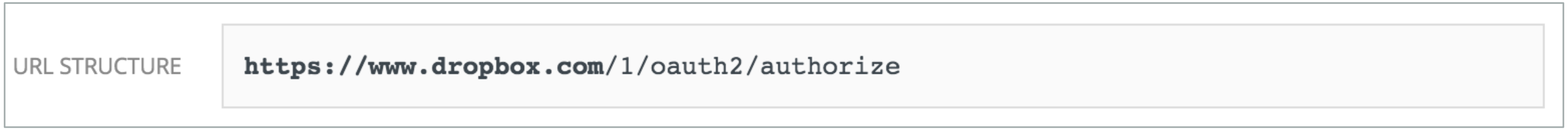
What is the Authorization Endpoint?

- The *Authorization Endpoint* is a URI defined by the server that the client app must use to authorize a user


User
App
The server is listening for authorization requests at this URI

How to find the Authorization Endpoint

- The server documentation will tell you the required Authorization Endpoint



URL STRUCTURE

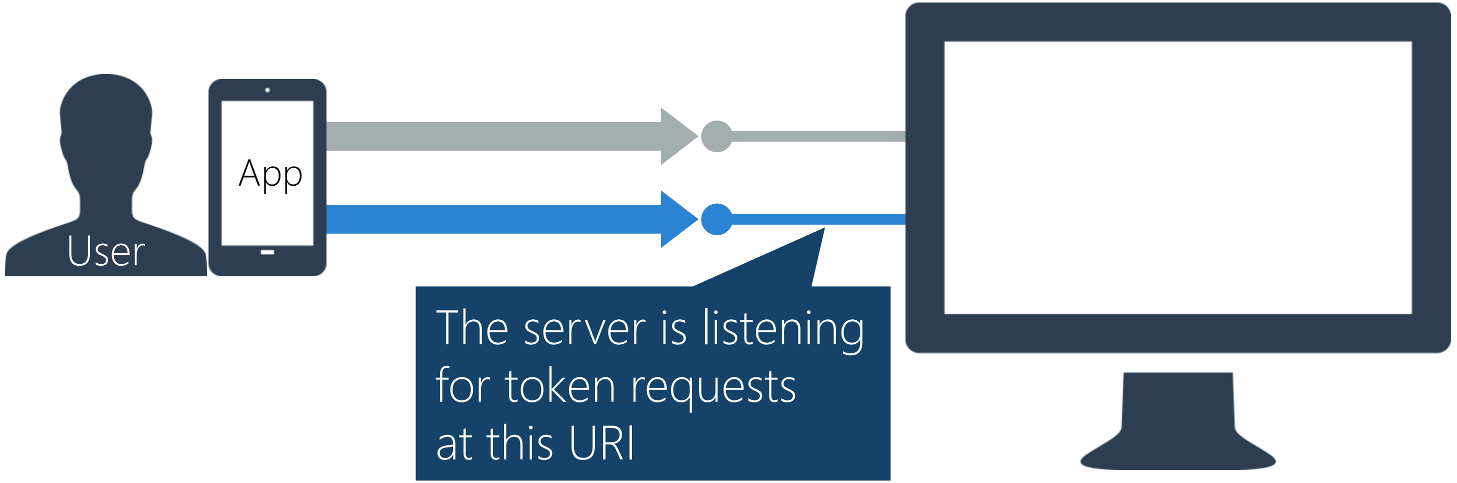
```
https://www.dropbox.com/1/oauth2/authorize
```

Dropbox's Authorization Endpoint

Source https://www.dropbox.com/developers-v1/core/docs#oa2-authorize

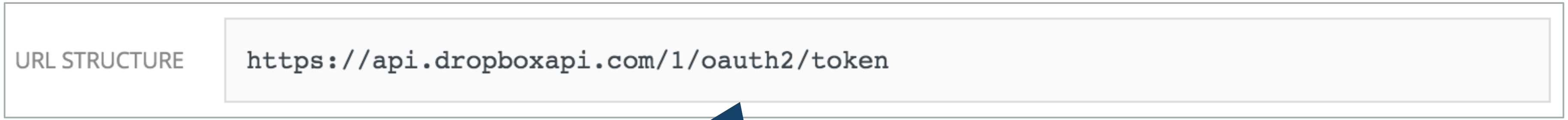
What is the Token Endpoint?

- The *Token Endpoint* is a URI defined by the server that the client app must use in most flows (e.g. to exchange an Authorization Code for an Access Token during the Authorization Code flow)



How to find the Token Endpoint

- The server documentation will tell you the required Token Endpoint



URL STRUCTURE

```
https://api.dropboxapi.com/1/oauth2/token
```

Dropbox's Token Endpoint

Xamarin University

Summary of your data

❖ You need to assemble several pieces of data before you can use Xamarin.Auth

```
string ClientId              = "<your Client Id goes here>";
string ClientSecret          = "<your Client Secret goes here>";
string Scope                 = "profile email";
Uri    AuthorizationEndpoint = new Uri("https://accounts.google.com/o/oauth2/v2/auth");
Uri    TokenEndpoint         = new Uri("https://www.googleapis.com/oauth2/v4/token");
Uri    RedirectionEndpoint   = new Uri("http://localhost");
```

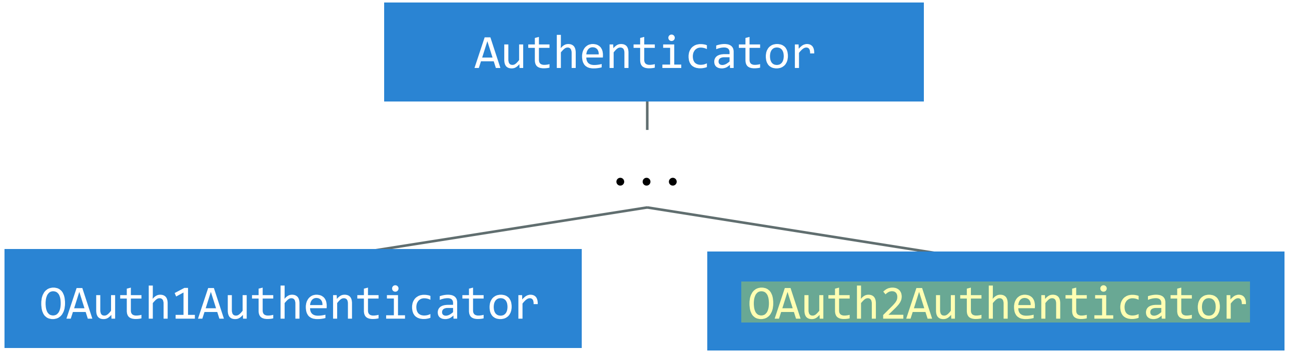
Your job is to obtain these values (example shows data to perform an Authorization Code flow against Google+)

Demonstration

Explore the Xamarin University OAuth Server

What is an Authenticator?

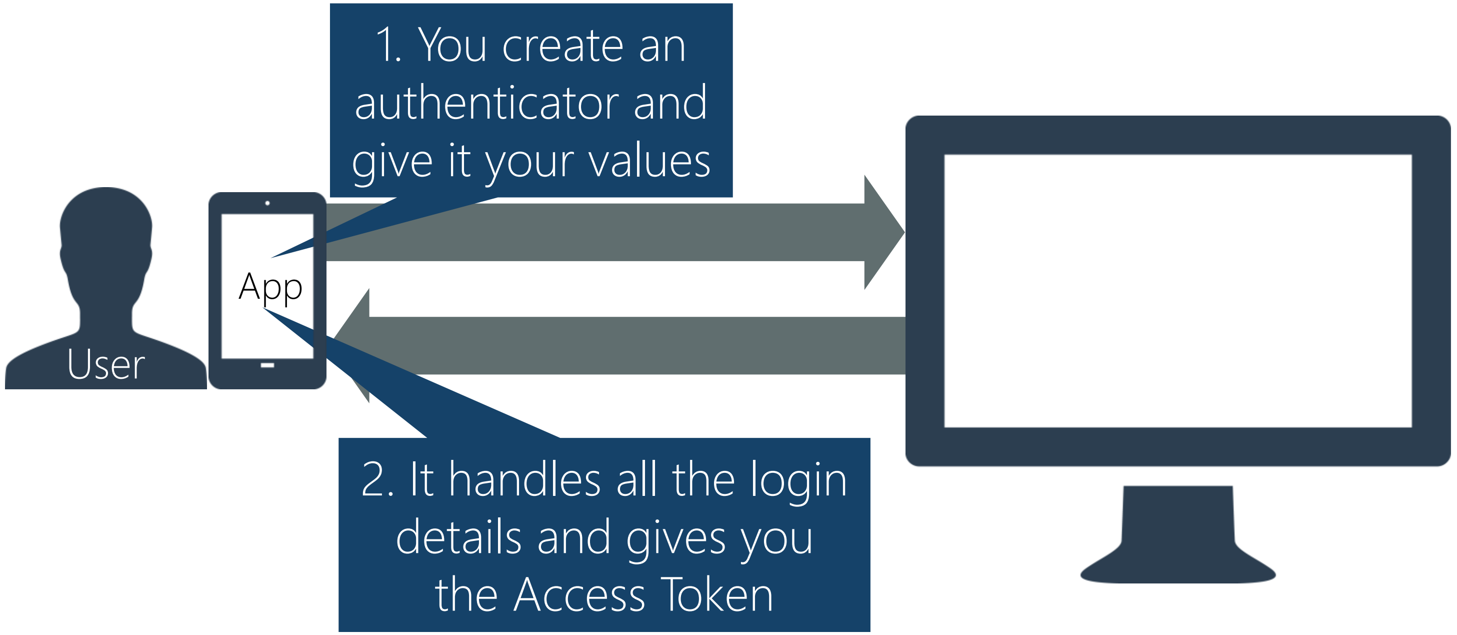
- Xamarin.Auth's authenticator types perform most of the work of authentication/authorization for you



We will only use the OAuth 2.0 version

What does OAuth2Authenticator do?

❖ `OAuth2Authenticator` handles the entire OAuth 2.0 authentication/authorization sequence for you


1. You create an authenticator and give it your values
App
User
2. It handles all the login details and gives you the Access Token

Xamarin University

How to get the Username

- You can optionally provide the authenticator with a delegate that retrieves the username (your function might parse the Id Token if the server included one, query the server, etc.)

```
Task<string> MyGetUsernameAsync(IDictionary<string,string> accountProperties)
{
   ...
}
```

You return the username you obtained, Xamarin.Auth includes it in the object it returns to you

These are the results of the OAuth2 auth process, including the Access Token you will need if you get the username from the server

How to choose a flow with authenticator

- `OAuth2Authenticator` can perform the Implicit flow or Authorization Code flow, you choose based on what you pass to the constructor

```
public OAuth2Authenticator
(
  string clientId,
  string scope,
  Uri authorizeUrl,
  Uri redirectUrl,
  GetUsernameAsyncFunc getUsernameAsync = null
);
```

Implicit flow

```
public OAuth2Authenticator
(
  string clientId,
  string clientSecret,
  string scope,
  Uri authorizeUrl,
  Uri redirectUrl,
  Uri accessTokenUrl,
  GetUsernameAsyncFunc getUsernameAsync = null
);
```

Authorization Code flow (includes Client Secret and Token endpoint)

Xamarin University

How to create an authenticator

❖ You create an **`OAuth2Authenticator`** and provide it with your app and server info

```
string ClientId              = ...;
string Scope                 = ...;
Uri    AuthorizationEndpoint = ...;
Uri    RedirectionEndpoint   = ...;

var authenticator = new OAuth2Authenticator(ClientId, Scope,
                                            AuthorizationEndpoint,
                                            RedirectionEndpoint);
```

Example code for the Implicit flow

Individual Exercise

Create an OAuth2Authenticator

Flash Quiz

Flash Quiz

① What 4 pieces of information are required for the Implicit flow

a) Client ID, Client Secret, Redirection Endpoint, Authorization Endpoint

b) Client ID, Scope, Redirection Endpoint, Authorization Endpoint

c) Client ID, Client Secret, Scope, Authorization Endpoint

d) Client ID, Scope, Redirection Endpoint, Token Endpoint

Xamarin University

Flash Quiz

① What 4 pieces of information are required for the Implicit flow

a) Client ID, Client Secret, Redirection Endpoint, Authorization Endpoint

b) <u>Client ID, Scope, Redirection Endpoint, Authorization Endpoint</u>

c) Client ID, Client Secret, Scope, Authorization Endpoint

d) Client ID, Scope, Redirection Endpoint, Token Endpoint

Flash Quiz

② True or False, Xamarin.Auth supports all standard OAuth 2.0 flows

a) True

b) False

Flash Quiz

② True or False, Xamarin.Auth supports all standard OAuth 2.0 flows

a) True

b) [✓ False]

Xamarin University

Login UI

- `Authenticator` creates a **login UI** for you, you need to display it to the user; your code will vary by platform

The return type is different on each platform

```
using AuthenticateUIType = System.Object;

public abstract partial class Authenticator
{ ...
   public AuthenticateUIType GetUI() { ... }
}
```

Xamarin.Auth source

Login web control

- Xamarin.Auth uses an embedded web control to host the server's HTML, it does not use the shared system browser

iOS

`UIWebView`

`WebView`

How to show the login UI [iOS]

- On iOS, the **`GetUI`** method returns a **`UIViewController`**

```
OAuth2Authenticator authenticator;
UIViewController rootController;
// ...
UIViewController controller = authenticator.GetUI();
var navController = new UINavigationController(controller);
rootController.PresentViewController(navController, true, null);
```

Display the view controller on iOS

Xamarin University

How to show the login UI [Android]

- On Android, the **`GetUI`** method returns an **`Intent`**

```
OAuth2Authenticator authenticator;
Activity context;
// ...
Intent intent = authenticator.GetUI(context);
context.StartActivity(intent);
```

Start the Activity on Android

Xamarin University

Authenticator Completed event

- Authenticators have a **completed event** that reports success/failure and provides an **`Account`** object

```
OAuth2Authenticator authenticator = ...;
authenticator.Completed += OnCompleted;
```

```
void OnCompleted(object sender, AuthenticatorCompletedEventArgs e)
{
  if (e.IsAuthenticated)
  {
    Account a = e.Account;
    ...
  }
}
```

Success?

Information return by the server is inside this Account object

What is an Account?

- Xamarin.Auth's **`Account`** class represents a collection of **user information**

```
public class Account
{ ...
  public virtual string                             Username   { get;         set; }
  public virtual Dictionary<string,string> Properties { get; private set; }
  public virtual CookieContainer                    Cookies    { get; private set; }
}
```

Xamarin.Auth source

`Account` is flexible in what it can store because of the **`Properties`** dictionary

Authenticator Access Token return

- The Access Token is in **`Account.Properties`** with key **`access_token`**

```
void OnCompleted(object sender, AuthenticatorCompletedEventArgs e)
{
  if (e.IsAuthenticated)
  {
    string token = e.Account.Properties["access_token"];
  }
}
```

Retrieve the token from the Account

Xamarin University

Detecting errors

❖ `OAuth2Authenticator` raises its **error event** if an error occurs

E.g. did not receive Authorization Code or Access Token as expected, Task to retrieve username was faulted, etc.

```
OAuth2Authenticator authenticator = ...;
authenticator.Error += OnError;
```

```
void OnError(object sender, AuthenticatorErrorEventArgs e)
{
   var x = e.Exception;

   var m = e.Message;
   ...
}
```

Exception will be passed along here if one occurred (might be null if no exception happened)

Message will always be non-null. Will be taken from the innermost exception if one occurred.

Token persistence

- Xamarin.Auth's **`AccountStore`** will **persist** the token securely on the device so you can use it the next time the app runs

Save

```
void OnCompleted(object sender, AuthenticatorCompletedEventArgs e)
{ ...
  AccountStore store = ... // Creation varies by platform
  store.Save(e.Account, "<Your serviceId>");
  ...
}
```

Load

```
AccountStore store = ... // Creation varies by platform
var accounts = store.FindAccountsForService("<Your serviceId>");
```

Individual Exercise

Authenticate with an OAuth 2.0 protected web service

Summary

1. Locate the values you need to send to the server for Authentication and Authorization
2. Authenticate and Authorize using Xamarin.Auth

Access an OAuth 2.0-secured API

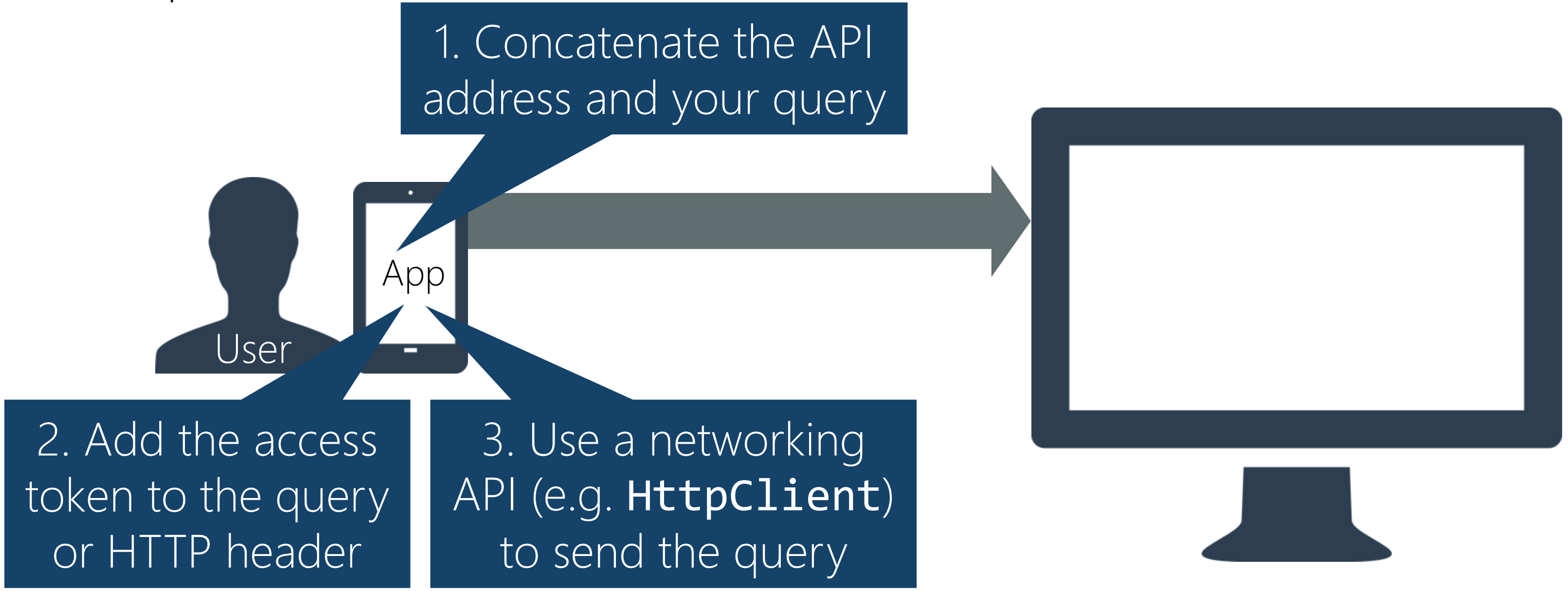
Xamarin University

Tasks

1. Use Xamarin.Auth to send an HTTP request
2. Detect and handle errors returned from your request

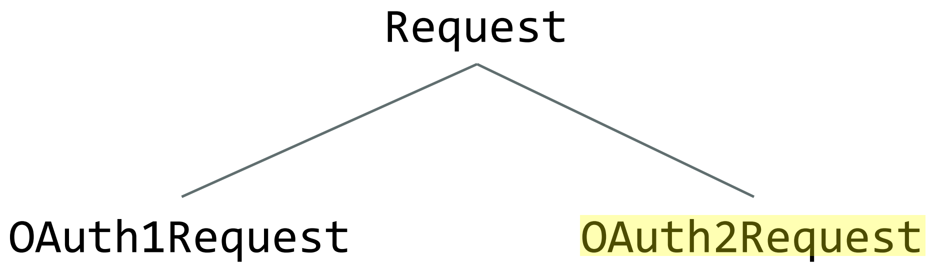
Motivation

❖ Formatting a request to an HTTP REST service manually is a tedious process


1. Concatenate the API address and your query
User
App
2. Add the access token to the query or HTTP header
3. Use a networking API (e.g. **HttpClient**) to send the query

What is an OAuth2Request?

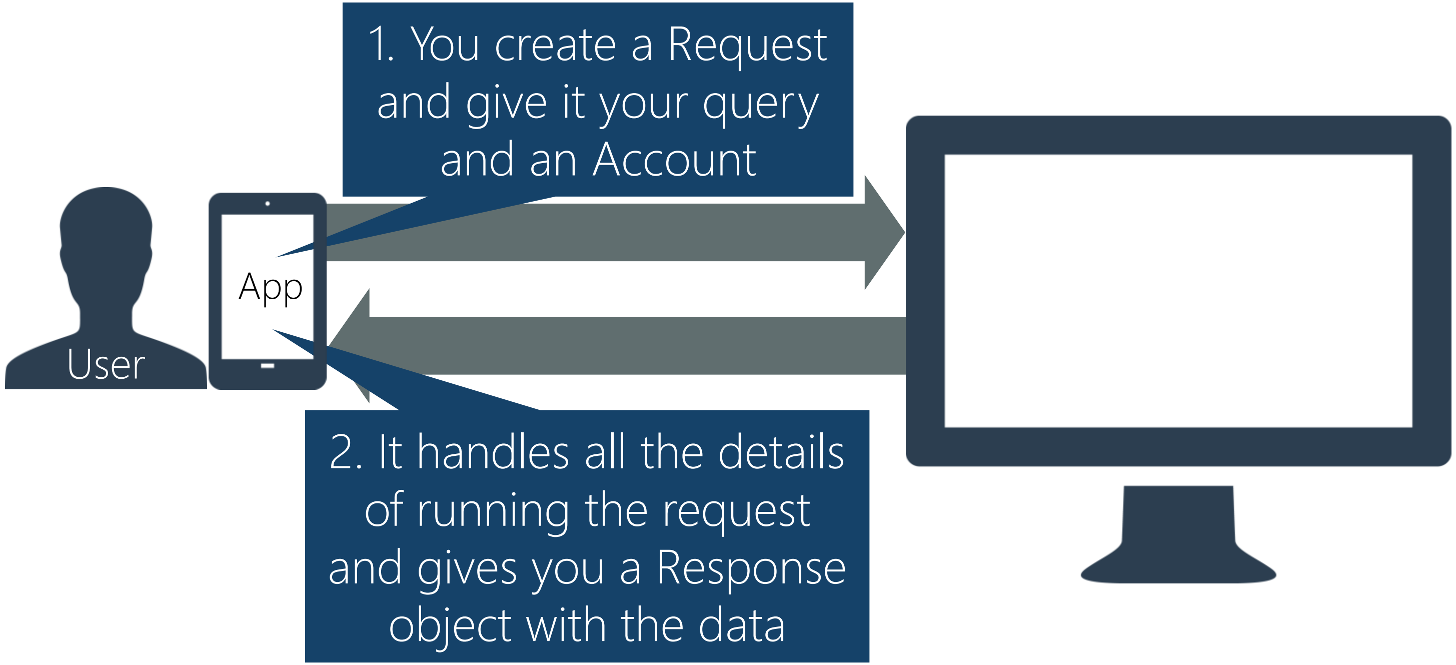
❖ Xamarin.Auth's **`OAuth2Request`** performs most of the work of executing an HTTP request for you



We will only use the OAuth 2.0 version

What does OAuth2Request do?

❖ `OAuth2Request` formats an HTTP request, adds the access token, sends it to the server, and captures the result


1. You create a Request and give it your query and an Account
User
App
2. It handles all the details of running the request and gives you a Response object with the data

Xamarin University

OAuth2Request Token availability

- **`OAuth2Request`** needs an Account object containing the Access Token

```
public class OAuth2Request : Request
{ ...
  public OAuth2Request(..., Account account);

}
```

Xamarin.Auth source

The **`Properties`** dictionary inside this Account must contain the token

Xamarin University

OAuth2Request Token formatting

❖ `OAuth2Request` adds the Access Token to the request URL, if you want to pass it in the HTTP Authorization header, you will have to format the request yourself with **little help** from Xamarin.Auth

```
public class OAuth2Request : Request
{ ...
    public static string GetAuthorizationHeader(Account account)
    { ...
        return "Bearer " + account.Properties["access_token"];
    }
}
```

Xamarin.Auth source

You must format your own request to put this in the header

OAuth2Request Token name

- Most servers use **`access_token`** as the name of the token HTTP parameter, **`OAuth2Request`** lets you specify a different name if needed

```
public class OAuth2Request : Request
{ ...
  public OAuth2Request(..., Account account);

  public string AccessTokenParameterName { get; set; }
}
```

Xamarin.Auth source

Set this property if your server requires a name other than **`access_token`** in the request.

Xamarin University

OAuth2Request query

❖ `OAuth2Request` takes the details of your query as constructor arguments

```
public class OAuth2Request : Request
{
  public OAuth2Request
  (
    string method,
    Uri url,
    IDictionary<string, string> parameters,
    Account account
  );
}
```

1. HTTP verb "GET", "POST", etc.

2. The server's API address

3. Added to the query after a "?"

Xamarin.Auth source

Xamarin University

HTTP Multipart

❖ Xamarin.Auth requests can handle HTTP Multipart which is useful for uploading files

```
public class Request
{
  void AddMultipartData(string name, string data);
  void AddMultipartData(string name, Stream data, string mimeType = "", string filename = "");
  ...
}
```

Xamarin.Auth source

Xamarin University

What is a Response?

- Xamarin.Auth's **`Response`** represents the server's response to an HTTP request

Status

Data

```
public class Response : IDisposable
{ ...
  Uri                            ResponseUri { get; protected set; }
  HttpStatusCode                 StatusCode  { get; protected set; }
  IDictionary<string, string> Headers     { get; protected set; }

  Task<string> GetResponseTextAsync  ();
  Task<Stream> GetResponseStreamAsync();
}
```

Xamarin.Auth source

How to perform HTTP GET

- You prepare an **`OAuth2Request`**, tell it to send your request to the server, then harvest the result

```
async Task<string> MyGetTextAsync(Uri url, IDictionary<string,string> properties, Account account)
{
  var request = new OAuth2Request("GET", url, properties, account);

  var response = await request.GetResponseAsync();

  string text =  await response.GetResponseTextAsync();

  return text;
}
```

1. Prepare the request

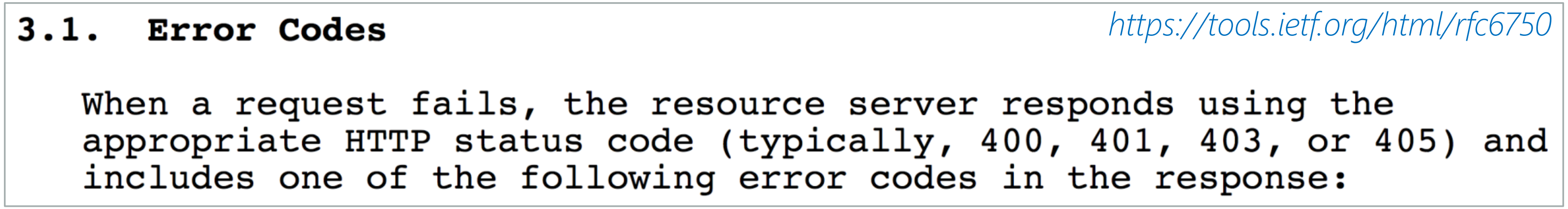
2. Send to server

3. Get result

Xamarin University

Error handling

- The OAuth2 *Bearer Token Usage specification* describes how the server should respond to invalid requests



3.1. Error Codes

https://tools.ietf.org/html/rfc6750

```
   When a request fails, the resource server responds using the
   appropriate HTTP status code (typically, 400, 401, 403, or 405) and
   includes one of the following error codes in the response:
```

The code appears in the Response's `StatusCode` property

Xamarin University

Error handling [invalid request]

- **`Response.StatusCode`** will be 400 **`BadRequest`** if your request is malformed

```
...
var request = new OAuth2Request("GET", url, properties, account);
var response = await request.GetResponseAsync();

if (response.StatusCode == HttpStatusCode.BadRequest)
{
   ...
}
```

Needed parameter was missing, invalid parameter, etc.

Error handling [invalid token]

- **`Response.StatusCode`** will be 401 **`Unauthorized`** when you need a new access token

```
...
var request = new OAuth2Request("GET", url, properties, account);
var response = await request.GetResponseAsync();

if (response.StatusCode == HttpStatusCode.Unauthorized)
{
   ...
}
```

Access token was expired, revoked, etc.

Error handling [insufficient scope]

- **`Response.StatusCode`** will be 403 **`Forbidden`** if the token does not provide access to the requested resource

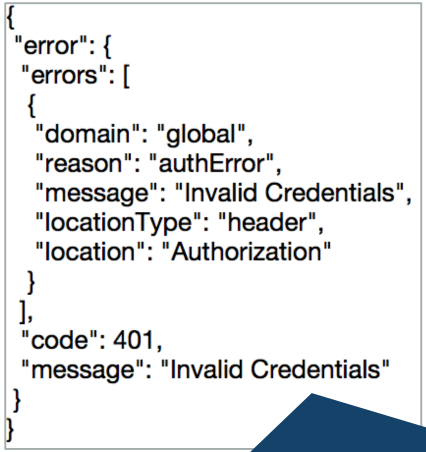
```
...
var request = new OAuth2Request("GET", url, properties, account);
var response = await request.GetResponseAsync();

if (response.StatusCode == HttpStatusCode.Forbidden)
{
   ...
}
```

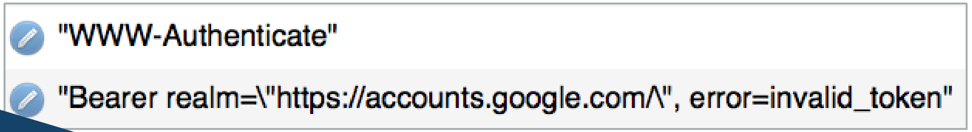
Requested scope was insufficient for app's needs

Error handling [details]

- You can check the returned text and the `WWW-Authenticate` response header for further details of the error



```
{
 "error": {
  "errors": [
   {
    "domain": "global",
    "reason": "authError",
    "message": "Invalid Credentials",
    "locationType": "header",
    "location": "Authorization"
   }
  ],
  "code": 401,
  "message": "Invalid Credentials"
 }
}
```



```
"WWW-Authenticate"
"Bearer realm=\"https://accounts.google.com/\", error=invalid_token"
```

Sample response from Google APIs using an invalid access token

Individual Exercise

Access data on an OAuth 2.0 protected service

Summary

1. Use Xamarin.Auth to send an HTTP request
2. Detect and handle errors returned from your request

Exchange a Refresh Token for a new Access Token

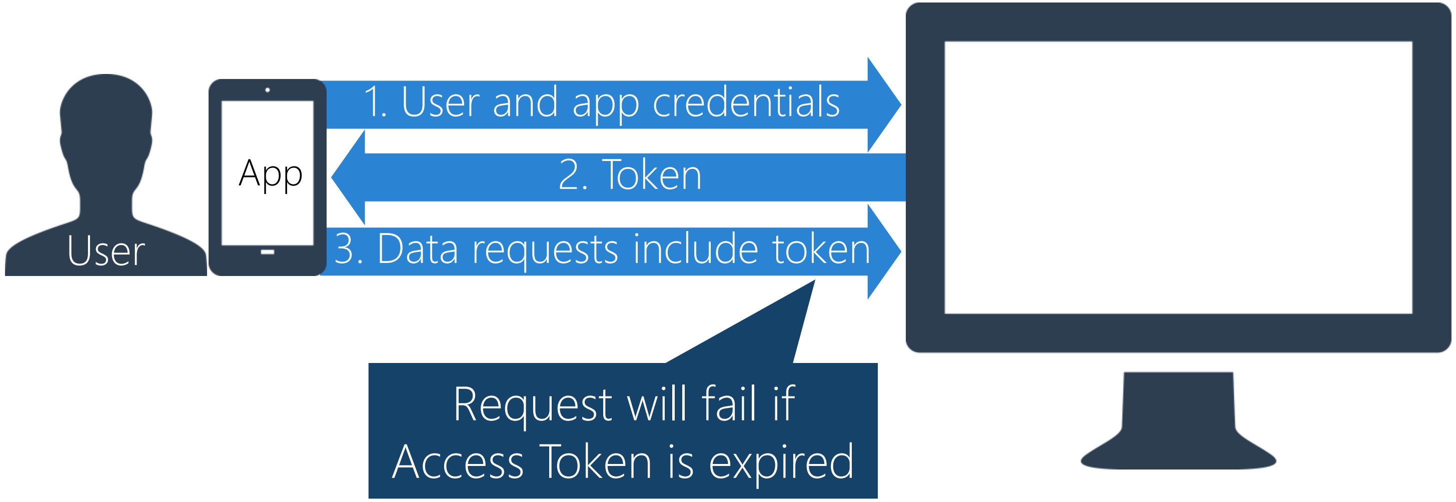
Xamarin University

Tasks

1. Determine the expiration time of an Access Token
2. Get a Refresh Token from the server
3. Exchange the Refresh Token for a new Access Token

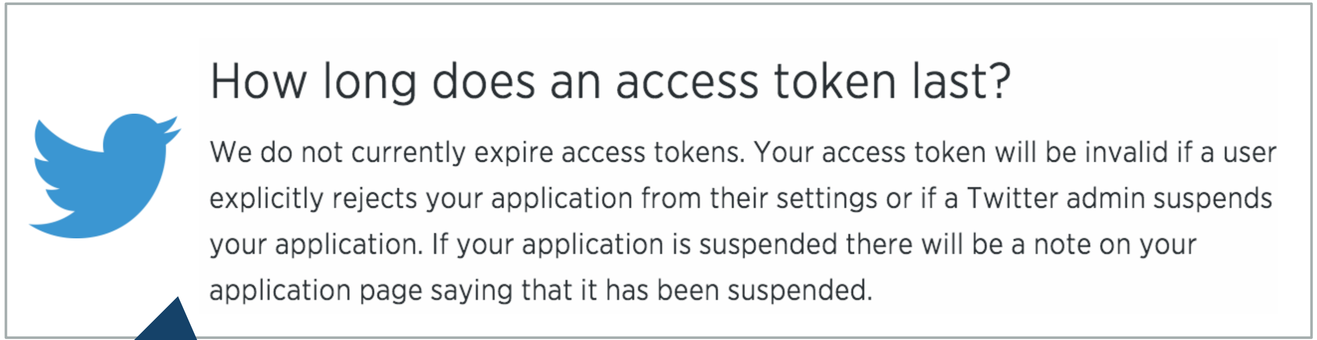
Motivation

❖ Some Access Tokens expire after a short time, it can be inconvenient to require the user to repeatedly login


User
App
1. User and app credentials
2. Token
3. Data requests include token
Request will fail if Access Token is expired

Server policy

❖ The server determines whether Access Tokens expire and if so, how long they last



How long does an access token last?

We do not currently expire access tokens. Your access token will be invalid if a user explicitly rejects your application from their settings or if a Twitter admin suspends your application. If your application is suspended there will be a note on your application page saying that it has been suspended.

Some servers never expire their Access Tokens

Others use a short expiry time (e.g. one hour)

Xamarin University

How to determine expiration?

❖ For tokens that expire, the server should send you the time-to-live with the Access Token

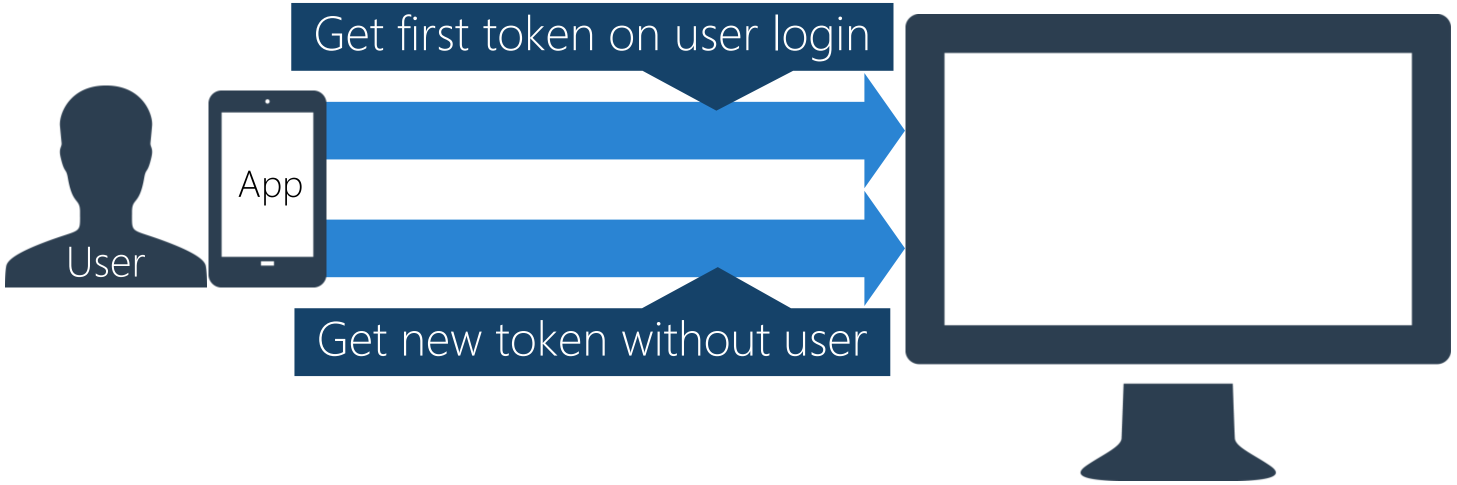
```
void OnCompleted(object sender, AuthenticatorCompletedEventArgs e)
{
   if (e.IsAuthenticated)
   {
      string duration = e.Account.Properties["expires_in"];
      ...
   }
}
```

Time in seconds until token expires

The key **expires_in** comes from the OAuth 2.0 spec

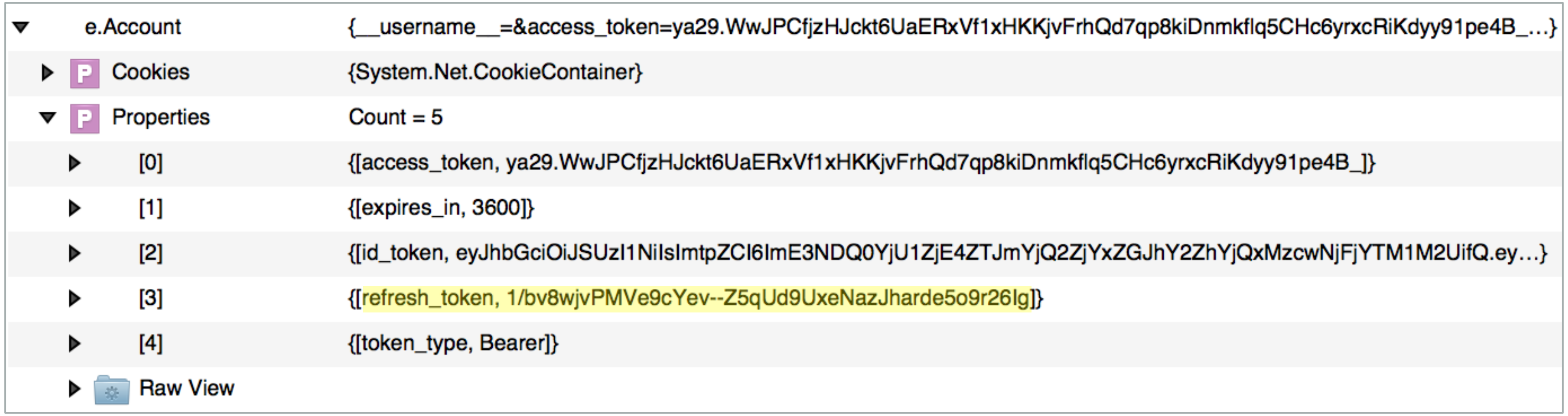
What is refresh?

❖ A server can support token *refresh* - a new Access Token can be retrieved after expiration without requiring the user to login again



What is a Refresh Token?

- A *Refresh Token* is a string the client can send to the server to get a new Access Token without requiring the user to login again



```
▼ e.Account       {__username__=&access_token=ya29.WwJPCfjzHJckt6UaERxVf1xHKKjvFrhQd7qp8kiDnmkflq5CHc6yrxcRiKdyy91pe4B_...}
  ▶ Cookies       {System.Net.CookieContainer}
  ▼ Properties    Count = 5
    ▶ [0]         {[access_token, ya29.WwJPCfjzHJckt6UaERxVf1xHKKjvFrhQd7qp8kiDnmkflq5CHc6yrxcRiKdyy91pe4B_]}
    ▶ [1]         {[expires_in, 3600]}
    ▶ [2]         {[id_token, eyJhbGciOiJSUzI1NiIsImtpZCI6ImE3NDQ0YjU1ZjE4ZTJmYjQ2ZjYxZGJhY2ZhYjQxMzcwNjFjYTM1M2UifQ.ey...}
    ▶ [3]         {[refresh_token, 1/bv8wjvPMVe9cYev--Z5qUd9UxeNazJharde5o9r26Ig]}
    ▶ [4]         {[token_type, Bearer]}
    ▶ Raw View
```

Flow requirement for refresh

- Generally, servers require that you use the Authorization Code flow if you want a Refresh Token

```
public OAuth2Authenticator
(
  string clientId,
  string scope,
  Uri authorizeUrl,
  Uri redirectUrl,
  GetUsernameAsyncFunc getUsernameAsync = null
);
```

Implicit flow will not yield Refresh Token

```
public OAuth2Authenticator
(
  string clientId,
  string clientSecret,
  string scope,
  Uri authorizeUrl,
  Uri redirectUrl,
  Uri accessTokenUrl,
  GetUsernameAsyncFunc getUsernameAsync = null
);
```

Authorization Code flow may return a Refresh Token in addition to the Access Token

Xamarin University

Parameter requirements for refresh

- Some servers require an additional parameter in your request in order to give you a Refresh Token

```
string Scope = "profile email&access_type=offline";
...
var authenticator = new OAuth2Authenticator
  (
    ClientId,
    ClientSecret,
    Scope,
    AuthorizationEndpoint,
    RedirectionEndpoint,
    TokenEndpoint
  );

authenticator.DoNotEscapeScope = true;
```

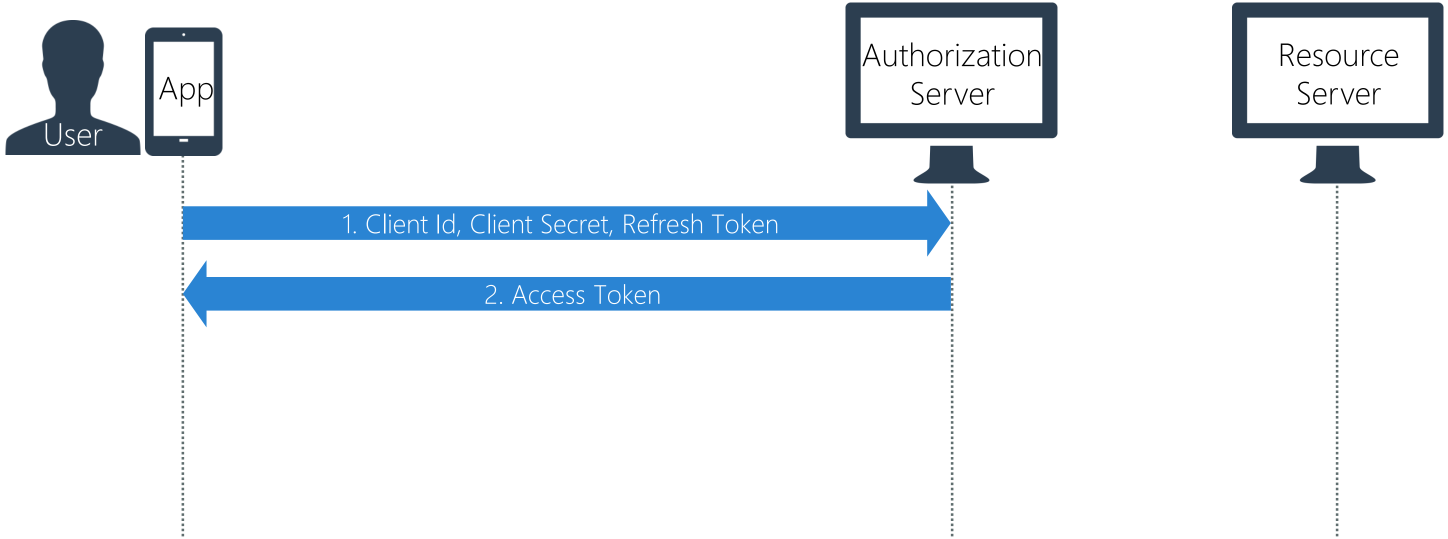
Xamarin.Auth lets you include extra parameters by adding them to your Scope and setting this property to true

Google requires this string to request a Refresh Token. Other servers use different strings and some do not require any parameter.

Xamarin University

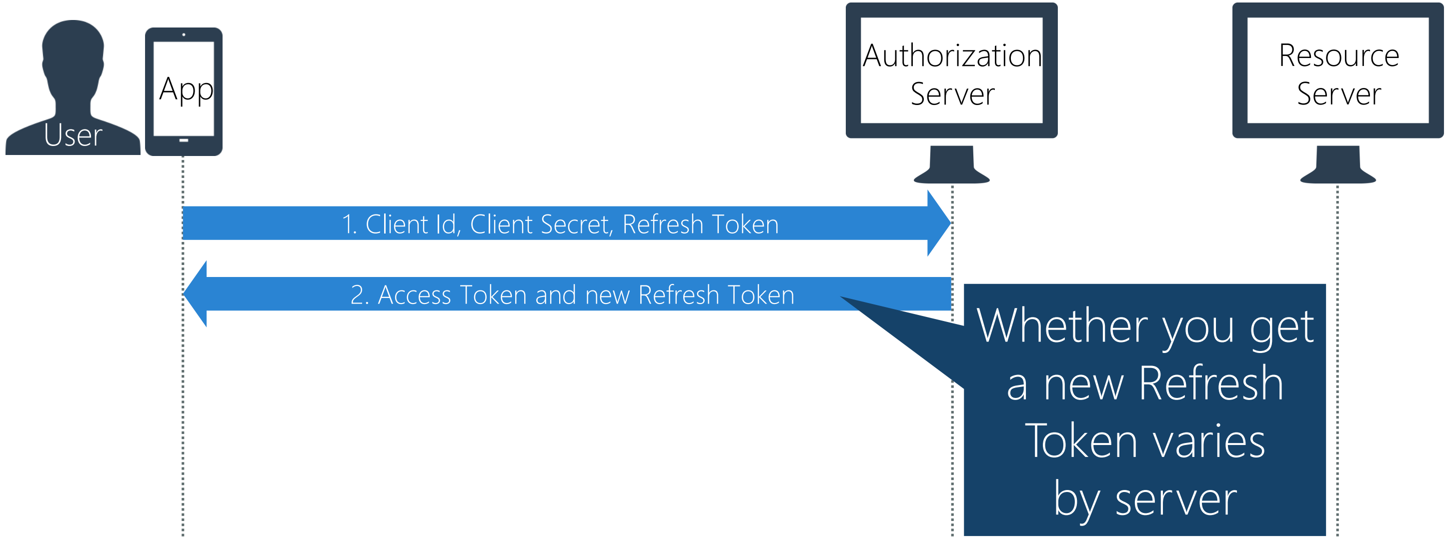
What is the Refresh flow?

- The *Refresh flow* is a single request-response to the server to exchange the Refresh Token for a new Access Token



Refresh Token duration

❖ Some servers give you a new Refresh Token every time you perform the refresh flow; others give you a single Refresh Token when the user first authenticates and your app can use that Refresh Token repeatedly


User
App
Authorization Server
Resource Server
1. Client Id, Client Secret, Refresh Token
2. Access Token and new Refresh Token
Whether you get a new Refresh Token varies by server

How to perform the Refresh flow

- We can use **`OAuth2Authenticator`**'s **`RequestAccessTokenAsync`** method to handle refresh for most servers

Create a new Authenticator

```
var authenticator = new OAuth2Authenticator
  (
    ClientId, ClientSecret, Scope, AuthorizationEndpoint,
    RedirectionEndpoint, TokenEndpoint
  );

var queryValues = new Dictionary <string, string> { ... };

var result = await authenticator.RequestAccessTokenAsync(queryValues);
```

Specify server specific query values

OAuth2Authenticator performs the refresh flow

Refresh Flow Query Values

- The query values required for the Refresh Flow vary by server but generally you're required to send the **Refresh Token**, **Client Id**, **Client Secret**, and **Grant Type**

```
var queryValues = new Dictionary<string, string>
{
  {"refresh_token", refreshToken},
  {"client_id", ServerInfo.ClientId},
  {"grant_type", "refresh_token"},
  {"client_secret", ServerInfo.ClientSecret},
};
```

Grant type is typically "refresh_token"

Refresh Flow Results

- **`RequestRefreshTokenAsync`** returns the result as a **`dictionary`** of **`string`** values

```
var result = await authenticator.RequestAccessTokenAsync(queryValues);

if (result.ContainsKey("access_token"))
    account.Properties["access_token"] = result["access_token"];

if (result.ContainsKey("refresh_token"))
    account.Properties["refresh_token"] = result["refresh_token"];
```

access tokens can by found using the keys "access_token" and "refresh_token"

Individual Exercise

Use a Refresh Token to retrieve a new Access Token

Xamarin University

Summary

1. Determine the expiration time of an Access Token
2. Get a Refresh Token from the server
3. Exchange the Refresh Token for a new Access Token

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft