ENT302

Using the Xamarin Salesforce Component

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Implement an external app and authenticate a user
2. Perform CRUD operations on a Standard Object
3. Create a custom SObject
4. Perform a Search

Implement an external app and authenticate a user

Xamarin University

Tasks

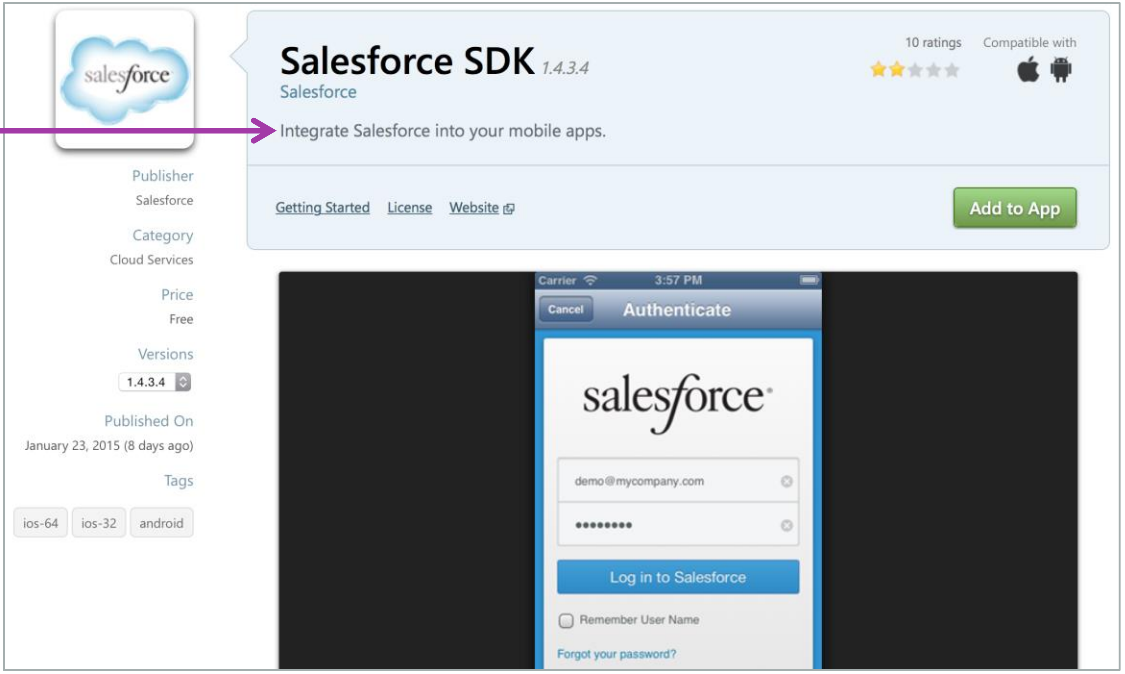
1. Add the Salesforce Component
2. Create a `SalesforceClient`
3. Display the login UI

What is the Salesforce Component?

- The Salesforce Component is a library that wraps the Salesforce APIs

Gives convenient access to Salesforce from Xamarin projects

Supports iOS and Android



Salesforce SDK 1.4.3.4
Salesforce
Integrate Salesforce into your mobile apps.

10 ratings
Compatible with

Getting Started License Website

Publisher
Salesforce

Category
Cloud Services

Price
Free

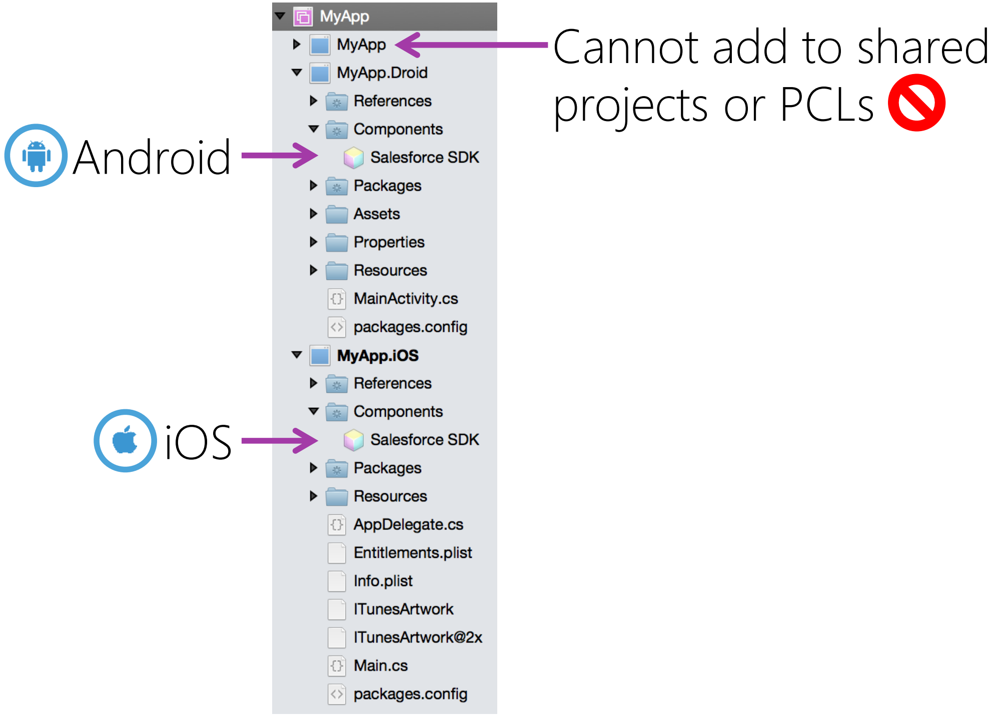
Versions
1.4.3.4

Published On
January 23, 2015 (8 days ago)

Tags
ios-64 ios-32 android

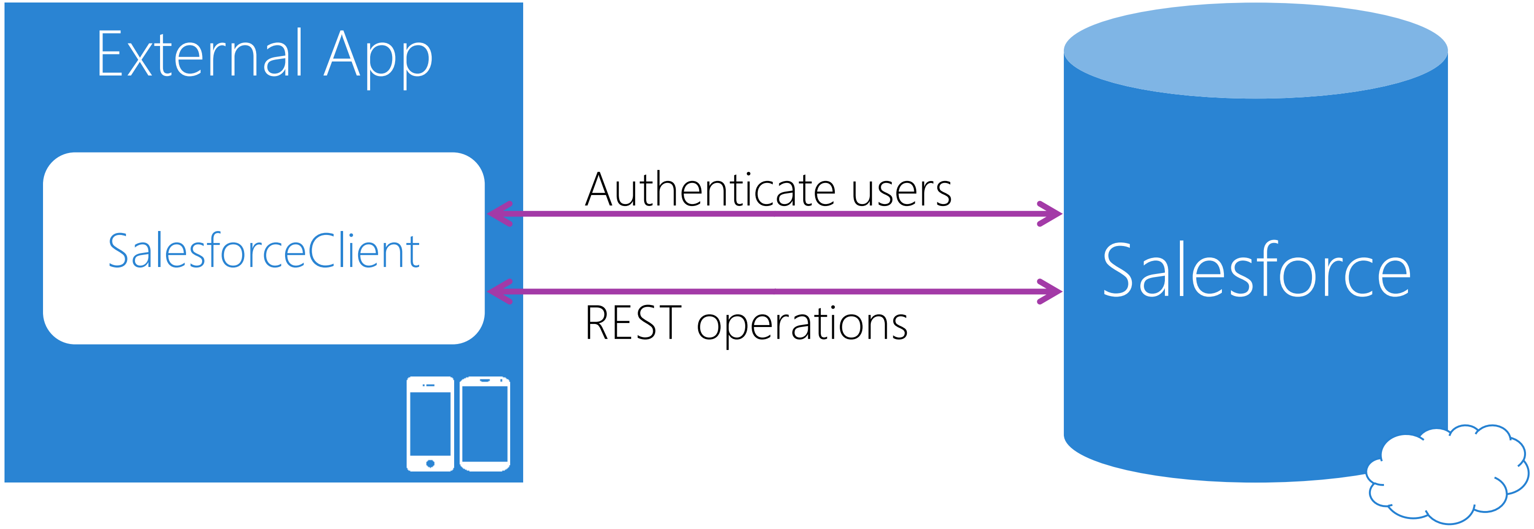
How to add the Salesforce Component

❖ Add the Salesforce Component to your iOS and Android projects



What is SalesforceClient?

- `SalesforceClient` is a class in the Salesforce Component that wraps the Salesforce Authentication and REST APIs



SalesforceClient constructor

- **`SalesforceClient`** performs the OAuth 2.0 User-Agent Flow for you so it needs your app's identity and callback

```
string ClientKey    = "...";
string ClientSecret = "...";
Uri    CallbackUrl  = new Uri("...");

var client = new SalesforceClient(ClientKey, ClientSecret, CallbackUrl);
```

$\uparrow$ $\uparrow$ $\uparrow$

The values used here must match the values stored on the Salesforce server for your Connected App, ClientKey and CallbackUrl are used for initial authentication, ClientSecret is used for refresh

Xamarin University

SalesforceClient login UI

- **SalesforceClient** creates a **login UI** for you, your code needs to display it to the user

Returns an **Intent** on Android

```
Activity         context;
SalesforceClient client;
// ...
var intent = (Intent)client.GetLoginInterface();
context.StartActivity(intent);
```

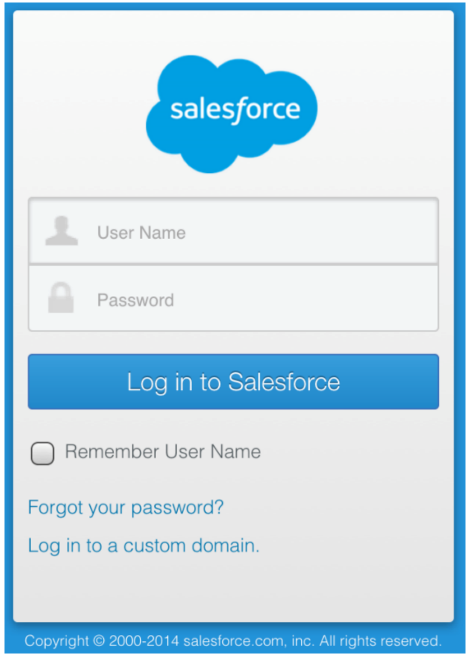
Returns a **ViewController** on iOS

```
UIViewController rootController;
SalesforceClient client;
// ...
var controller = (UIViewController)client.GetLoginInterface();
var navController = new UINavigationController(controller);
rootController.PresentViewController(navController, true, null);
```

SalesforceClient OAuth UI

- Displaying the login UI begins the OAuth sequence

User credentials sent directly to Salesforce and are not available to your app


salesforce
User Name
Password
Log in to Salesforce
Remember User Name
Forgot your password?
Log in to a custom domain.
Copyright © 2000-2014 salesforce.com, inc. All rights reserved.

SalesforceClient Complete event

- `SalesforceClient` raises an **event** when the user has completed authentication, the event args contain the user's account info

```
SalesforceClient client;
// ...
client.AuthenticationComplete += OnComplete;
```

```
void OnComplete(object sender, AuthenticatorCompletedEventArgs e)
{
  if (e.IsAuthenticated)
  {
    ISalesforceUser user = e.Account;
    // ...
  }
}
```

Success? → `if (e.IsAuthenticated)`

User info → `ISalesforceUser user = e.Account;`

Xamarin University

What is ISalesforceUser?

- `ISalesforceUser` represents an authenticated user

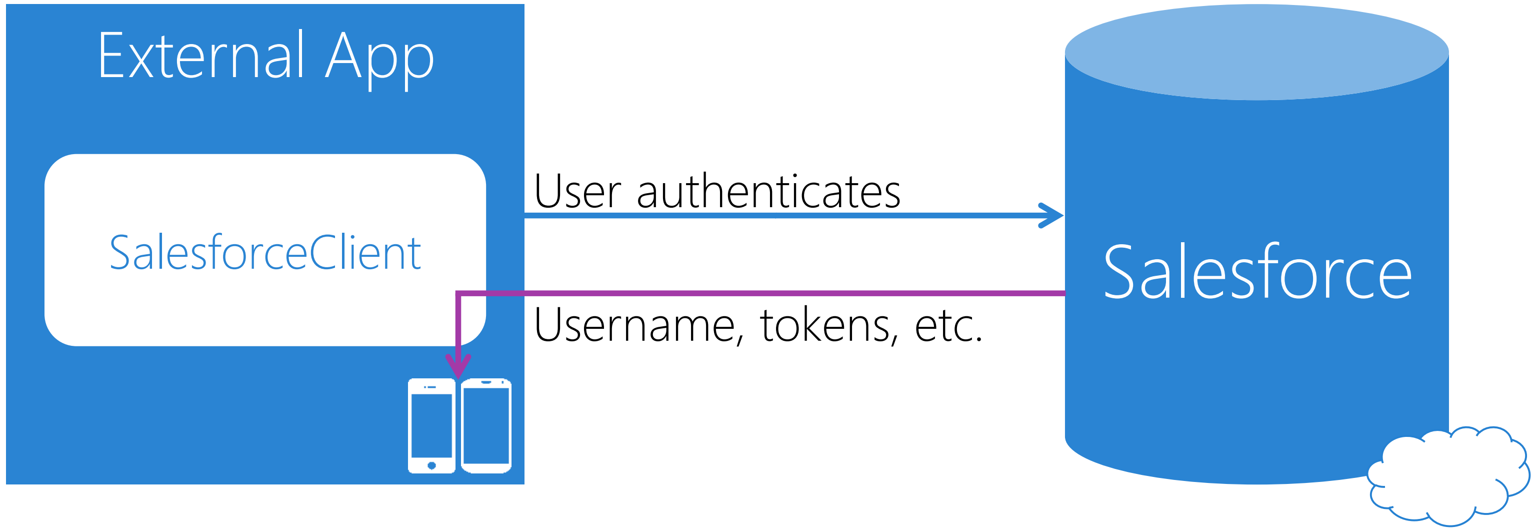
```
public interface ISalesforceUser
{
  string                             Username   { get; set; }
  Dictionary<string, string> Properties { get; }
  ...
}
```

Contains the OAuth access token, the OAuth scopes, the URL of the Salesforce server to use with REST calls, etc.

Xamarin University

User caching

- `SalesforceClient` automatically stores a user's `ISalesforceUser` info on the device and reloads it in its constructor



User caching API

- **SalesforceClient** exposes an **API** to let you save/load users

Save one user →
Get all saved users →
Currently active user →

```
void Save(ISalesforceUser account);

IEnumerable<ISalesforceUser> LoadUsers();

ISalesforceUser CurrentUser { get; set; }
```

Note: you cannot disable the auto save/load, but you can overrule it by setting **CurrentUser** to **null** or to a user of your choice from among those saved

How to test if a user was loaded?

- Examine **`SalesforceClient.CurrentUser`** to determine if a saved user was successfully loaded

Auto load success? →

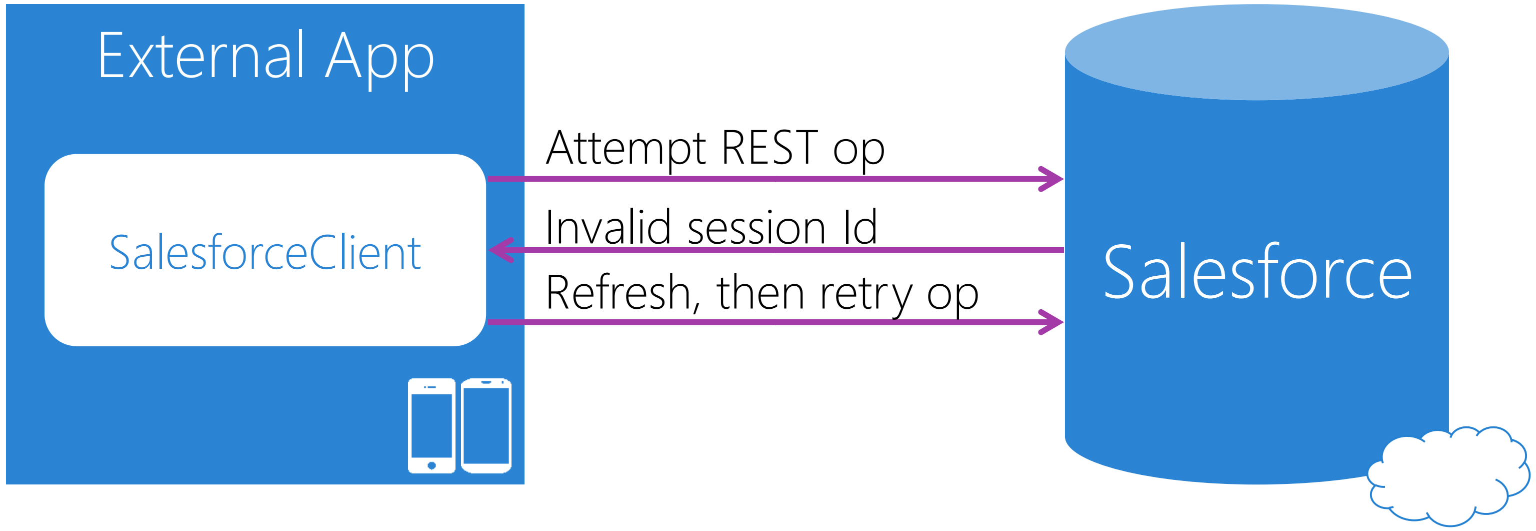
```
...
var client = new SalesforceClient(...);

if (client.CurrentUser == null)
{
   // No user was loaded, display login UI
   // ...
}
```

Xamarin University

Session refresh

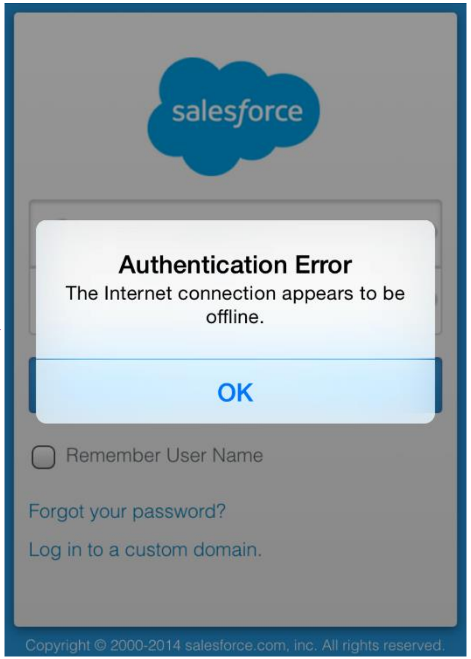
❖ `SalesforceClient` will attempt to refresh the access token as needed, it throws an exception if the refresh fails

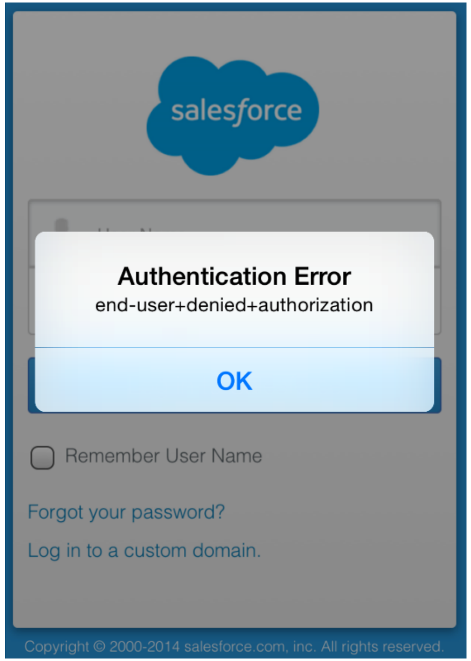


Error Reporting

- `SalesforceClient` displays some problems directly to the user

Network errors or security issues (e.g. suspected forgery) are shown as alerts →


Authentication Error
The Internet connection appears to be offline.
salesforce
OK
Remember User Name
Forgot your password?
Log in to a custom domain.
Copyright © 2000-2014 salesforce.com, inc. All rights reserved.


Authentication Error
end-user+denied+authorization
salesforce
OK
Remember User Name
Forgot your password?
Log in to a custom domain.
Copyright © 2000-2014 salesforce.com, inc. All rights reserved.

Group Exercise

Create a SalesforceClient and authenticate a user

Flash Quiz

Xamarin University

Flash Quiz

① Which OAuth 2.0 flow does the Salesforce Component use?

a) Web server

b) User agent

c) Username and password

Flash Quiz

① Which OAuth 2.0 flow does the Salesforce Component use?

a) Web server

b) <u>User agent</u>

c) Username and password

Flash Quiz

② The user has to login again when their access_token expires?

a) True

b) False

Flash Quiz

② The user has to login again when their access_token expires?

a) True

b) <u>False</u>

Summary

1. Add the Salesforce Component
2. Create a `SalesforceClient`
3. Display the login UI

Perform CRUD operations on a Standard Object

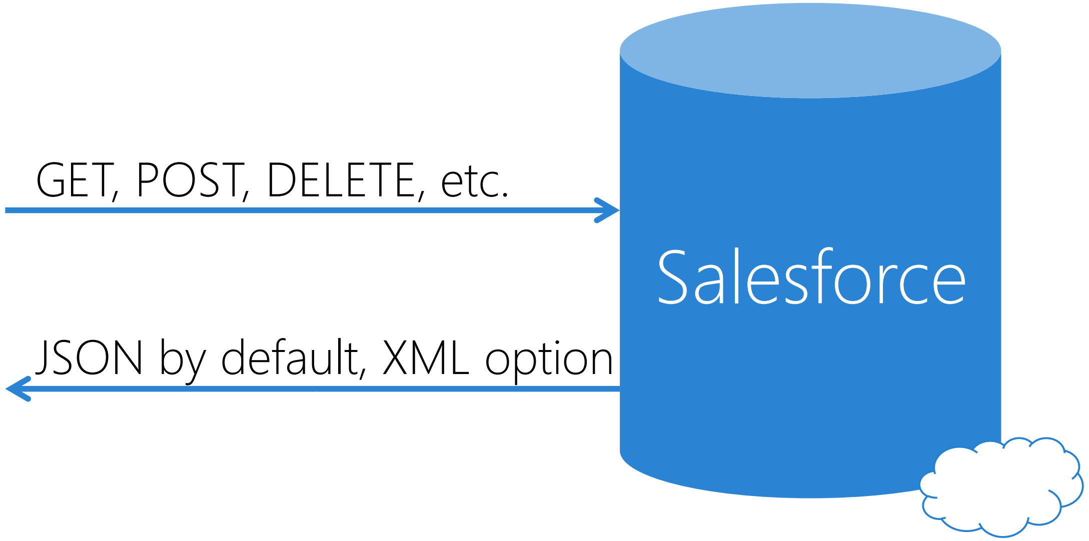
Xamarin University

Tasks

1. Write a SOQL query
2. Execute a query
3. Create a new record
4. Update an existing record
5. Delete a record

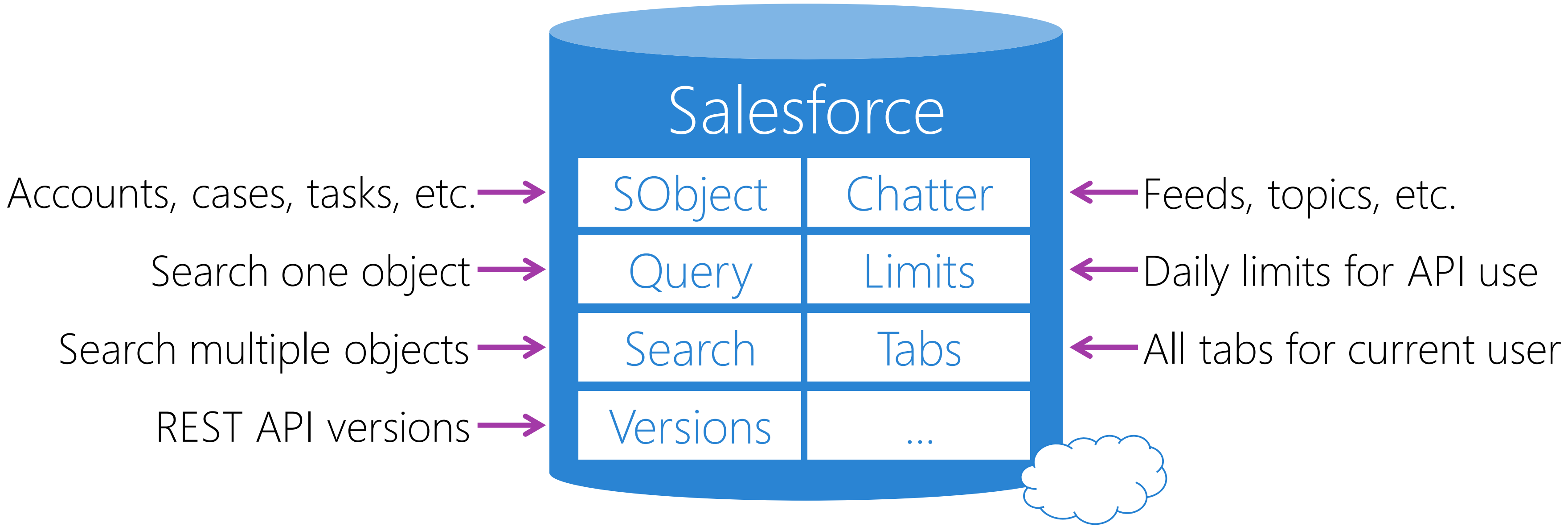
What is the Salesforce REST API?

- The Salesforce REST API provides access to Salesforce data using standard HTTP verbs



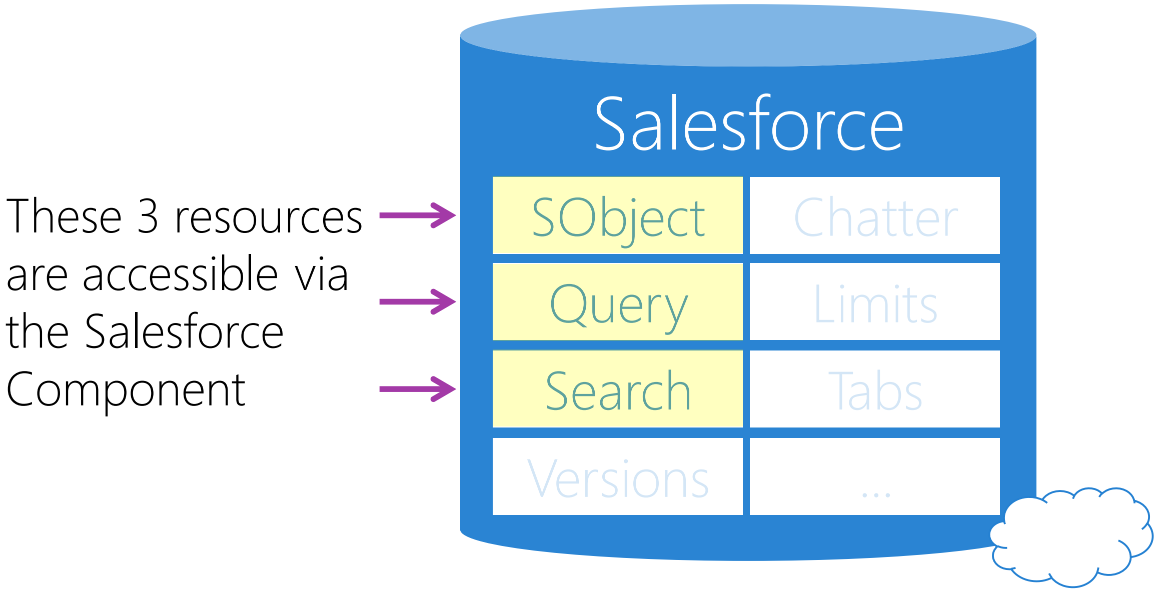
What is a Salesforce REST Resource?

❖ A Salesforce *REST Resource* is a piece of Salesforce data exposed via the Salesforce REST API



Salesforce Component REST access

- The Salesforce Component provides access to **some** of the resources exposed by the Salesforce REST API


These 3 resources are accessible via the Salesforce Component
Salesforce
SObject
Chatter
Query
Limits
Search
Tabs
Versions
...

What is an SObject?

- SObject is a class defined in the Salesforce Component that represents a record from a Salesforce Object (i.e. **`SObject`** represents a table row)

Id has a dedicated entry, all other fields go in the Options dictionary

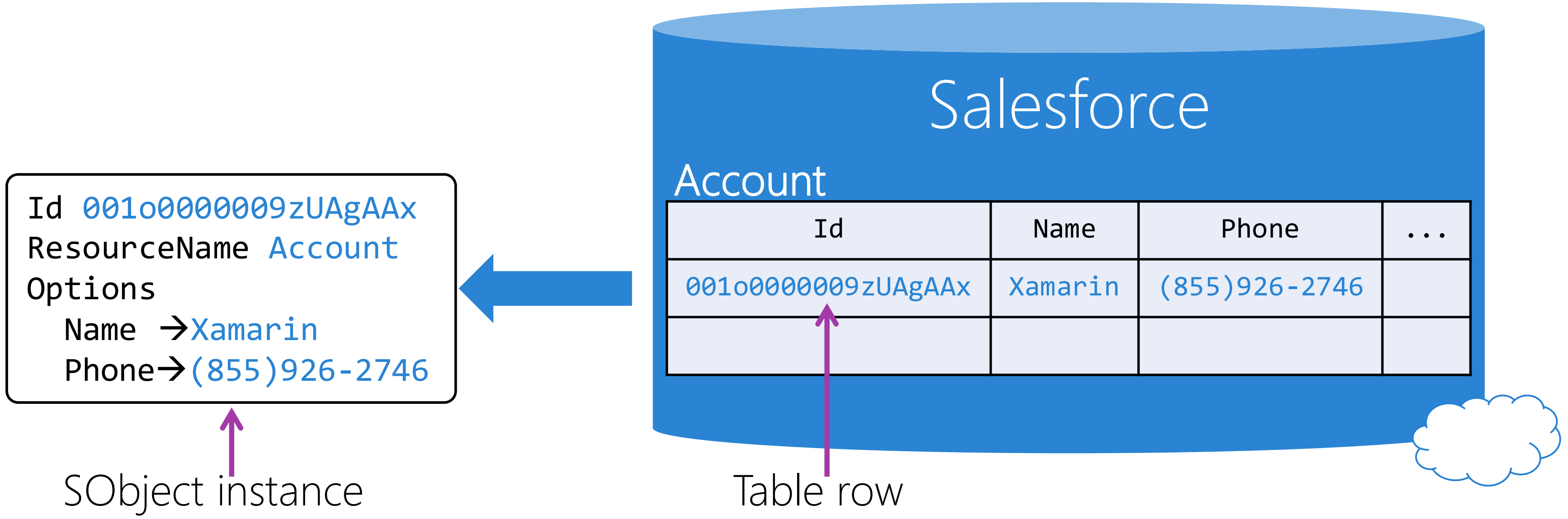
```
public class SObject : ISalesforceResource
{ ...
  public string                             Id      { get; }
  public IDictionary<string, JsonValue> Options { get; }

  public virtual string ResourceName { get; set; }
}
```

Object name (i.e. table name)

Mapping to SObjects

- The Salesforce Component creates SObject instances for you during query operations



```
Id 001o0000009zUAgAAx
ResourceName Account
Options
  Name →Xamarin
  Phone→(855)926-2746
```

Id	Name	Phone	...
001o0000009zUAgAAx	Xamarin	(855)926-2746	

CRUD methods

- **SalesforceClient** extension methods in the Salesforce Component provides CRUD operations

```
Task<string>                     CreateAsync(SObject sobject) ...
Task<IEnumerable<SObject>> QueryAsync (string  query  ) ...
Task                             UpdateAsync(SObject sobject) ...
Task<bool>                       DeleteAsync(SObject sobject) ...
```

Provide a simple interface that uses **SObject** and string and hide the details of the Salesforce REST API

Note: there are also synchronous versions of these methods that block the calling thread so are rarely needed.

What is SOQL?

- The *Salesforce Object Query Language* (SOQL) is a language for writing SELECT statements against a Salesforce Object (i.e. select from a table)

```
SELECT Id,Name FROM Account WHERE BillingState = 'CA' ORDER BY AnnualRevenue
```

Fields to select must include Id. Wildcard not supported.

Supports HAVING to include standard function calls like COUNT, SUM, MIN, etc.

Can order by ASC or DESC

Supports LIMIT and OFFSET for paging, and GROUP BY for grouping

Xamarin University

How to Query

- Use **`QueryAsync`** to execute a SOQL query

SOQL query →

Execute →

Id has its own property, other values are in Options →

```
SaleforceClient client;
...
var query = "SELECT Id,Name,Phone FROM Account WHERE Name LIKE 'X%'";

var sObjects = await client.QueryAsync(query);

foreach (var account in sObjects)
{
  string i = account.Id;
  string n = account.Options["Name"];
  string p = account.Options["Phone"];
  // ...
}
```

How to Create

❖ Use **`CreateAsync`** to create a new record

```
SaleforceClient client;
...
var xam = new SObject();

xam.ResourceName = "Account";

xam.Options["Name"] = "Xamarin";

string id = await client.CreateAsync(xam);
```

Specify table to create in →

Set required → field values

↑ Returns the Id of the new record if successful or null

Xamarin University

How to prepare an update

- SObject **event** gives you an opportunity to prepare your data before it is sent to Salesforce as an update

```
public class SObject : ISalesforceResource
{ ...
  public event EventHandler<UpdateRequestEventArgs> PreparingUpdateRequest;
}
```

Contains a dictionary of all the field names and values in the SObject's Options dictionary, the most common thing to do is remove read-only fields so they are not sent to Salesforce

How to Update

- Use **UpdateAsync** to update an existing record

```
SaleforceClient client;

async Task UpdateEmployeeCountAsync(string id, string count)
{
  string q = "SELECT Id,Name,NumberOfEmployees,LastModifiedDate FROM Account WHERE Id='" + id +"'";
  SObject a = (await client.QueryAsync(q)).First(); // retrieve the record to update

  a.Options["NumberOfEmployees"] = count; // modify local data

  a.PreparingUpdateRequest += (sender, args) =>
    {
       args.UpdateData.Remove("LastModifiedDate"); // remove a read-only field
    };

  await client.UpdateAsync(a);
}
```

How to Delete

- Use **`DeleteAsync`** to delete an existing record

```
SaleforceClient client;

async Task DeleteAccountAsync(string id)
{
  var a = new SObject();

  a.Id           = id;
  a.ResourceName = "Account";

  bool wasDeleted = await client.DeleteAsync(a);
}
```

Specify Id and table →

↑ bool indicates success/failure

Individual Exercise

Query a Standard Object

Summary

1. Write a SOQL query
2. Execute a query
3. Create a new record
4. Update an existing record
5. Delete a record

Create a custom SObject

Tasks

1. Code a derived class of SObject
2. Override ResourceName
3. Write a property for each field
4. Handle updates
5. Use the supplied type converter

Xamarin University

Motivation [problem]

- Using SObject to store your client-side data is awkward

Could forget to set table name →

Might misspell field names →

Must handle update on each instance →

```
var a = new SObject();

a.ResourceName = "Account";

a.Options["NumberOfEmployees"] = "200";

a.PreparingUpdateRequest += (s, e) =>
  {
     ...
  };
```

Xamarin University

Motivation [solution]

❖ Using a **custom SObject-derived type** is simpler and safer

```
var a = new SObject();

a.ResourceName = "Account";

a.Options["NumberOfEmployees"] = "200";

a.PreparingUpdateRequest += (s, e) =>
  {
     ...
  };
```

```
var a = new MyAccount();

a.NumberOfEmployees = 200;
```

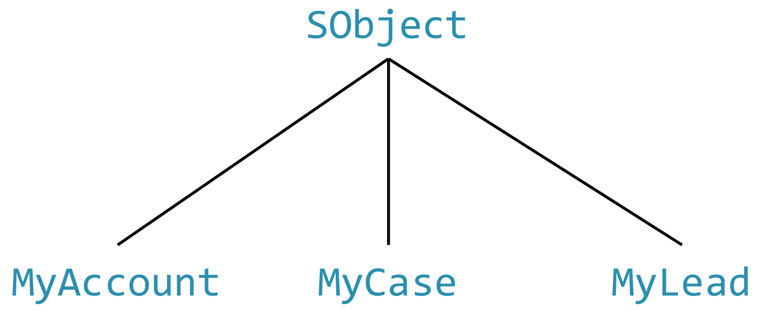
Offers a named property with type conversion. **`ResourceName`** and **`PreparingUpdateRequest`** are handled internally.

Xamarin University

Custom SObject

- You can code derived classes of SObject that provide a simpler and safer interface

Typical to write one subclass for each Salesforce Object you access →



SObject services

- SObject has built-in **support** for custom derived types

```
public class SObject : ...
{ ...
  public IDictionary<string, JsonValue> Options { get; protected set; }

  protected JsonValue GetOption   (string key, string defaultValue = "") ...
  protected void      SetOption<T>(string key, T value, Func<T, JsonValue> convertFunc = null) ...

  public virtual string ResourceName { get; set; }

  public event EventHandler<UpdateRequestEventArgs> PreparingUpdateRequest;
}
```

You write properties that store data here

You override Resource Name

You subscribe and then handle updates in your derived class

How to code an SObject [steps]

- Steps to implement a custom SObject type:
  1. Code a derived class of SObject
  2. Override ResourceName
  3. Write a property for each field
  4. Handle updates

Xamarin University

How to code an SObject [step 1]

❖ Code a derived class of **SObject**

```
public class MyAccount : SObject
{
   ...
}
```

Class names typically mirror the Salesforce Object names, the “My” pattern is used here to emphasize that this is code you would write

How to code an SObject [step 2]

- Override **`ResourceName`**

Hardcode the Resource Name (this is the Salesforce Object name, i.e. the table name) →

```
public class MyAccount : SObject
{
  public override string ResourceName
  {
    get { return "Account"; }
    set { }
  }
  ...
}
```

How to code an SObject [step 3]

❖ Write a **property** for each field

```
public class MyAccount : SObject
{ ...
  public string Name
  {
    get { return GetOption("Name"); }
    set { SetOption("Name", value); }
  }

  public int NumberOfEmployees
  {
    get { return ToInt(GetOption("NumberOfEmployees")); }
    set { SetOption("NumberOfEmployees", value.ToString()); }
  }

  public string LastModifiedDate
  {
    get { return GetOption("LastModifiedDate"); }
    set { SetOption("LastModifiedDate", value); }
  }
}
```

Supply properties for all the fields that your app needs to access

```
static int ToInt(JsonValue value)
{
  int result;

  if (int.TryParse(value.ToString(), out result))
    return result;
  else
    return 0;
}
```

How to code an SObject [step 4]

❖ Handle updates

Subscribe →

Prepare data as needed →

```
public class MyAccount : SObject
{ ...
  public MyAccount()
  {
    base.PreparingUpdateRequest += OnUpdate;
  }

  void OnUpdate(object sender, UpdateRequestEventArgs args)
  {
     args.UpdateData.Remove("LastModifiedDate");
  }
}
```

Type converter [motivation]

❖ Queries return SObjects and not instances of your custom derived type

```
var query = "SELECT Id,Name,NumberOfEmployees,LastModifiedDate FROM Account";

var sObjects = await client.QueryAsync(query);
```

↑

Returns `IEnumerable<SObject>`

Type converter [provided]

- SObject provides a generic **type converter** from SObject to your custom derived type

```
public class SObject : ...
{ ...
  public T As<T>() where T : SObject, new()
  {
    var result = new T();

    result.SetInner(this);

    return result;
  }
}
```

Create an instance of your derived type → `var result = new T();`

Avoid copying data by wrapping the new object around the old one → `result.SetInner(this);`

Type converter [use]

- You need to manually **apply the type converter** to create instances of your derived class

Convert one at a time →

```
var query = "...";
var sObjects = await client.QueryAsync(query);

var accounts = new List<MyAccount>();
foreach (var sObject in sObjects)
{
  var account = sObject.As<MyAccount>();
  accounts.Add(account);
}
```

Convert all at once using LINQ →

```
var accounts = sObjects.Select(s => s.As<MyAccount>()).ToList();
```

Individual Exercise

Create a custom SObject

Flash Quiz

Xamarin University

Flash Quiz

① Which operations are generally easier when using a custom SObject-derived type vs. using SObject directly?

a) Get/set field values

b) Preparing an update

c) Setting the `ResourceName`

d) All of the above

Flash Quiz

① Which operations are generally easier when using a custom SObject-derived type vs. using SObject directly?

a) Get/set field values

b) Preparing an update

c) Setting the `ResourceName`

d) All of the above

Flash Quiz

② The **`SObject.As<T>`** method is inefficient since it copies the fields?

a) True

b) False

Flash Quiz

② The **`SObject.As<T>`** method is inefficient since it copies the fields?

a) True

b) [✓ False]

Summary

1. Code a derived class of SObject
2. Override ResourceName
3. Write a property for each field
4. Handle updates
5. Use the supplied type converter

Perform a Search

Xamarin University

Tasks

1. Write a SOSL search
2. Execute a search

Xamarin University

Motivation

- SOQL queries search only a single Salesforce Object, sometimes you need to search across a larger area

```
SELECT Id,Name FROM Account WHERE Name='Xamarin'
```

↑

This query will only search Account,
it will not search not search Lead,
Case, Feed, Idea, Order, etc.

What is SOSL?

- The *Salesforce Object Search Language* (SOSL) is a language for writing text-search expressions across multiple Salesforce Objects

```
FIND {corp* OR inc*} IN NAME FIELDS RETURNING Account(Id,Name), Lead
```

Search term(s). Wildcards */? and AND/OR are supported

Fields to search: ALL FIELDS, EMAIL FIELDS, NAME FIELDS, PHONE FIELDS, SIDEBAR FIELDS

Can omit the RETURNING clause to search all Objects

Object to search and fields to retrieve

Object to search, retrieve only the Id field

Automatically included info

- SOSL results automatically include the **record type and record URL**

```
FIND {corp*} IN NAME FIELDS RETURNING Account(Id,Name)
```

```
...
{
  "attributes":
  {
    "type": "Account",
    "url": "/services/data/v28.0/sobjects/Account/001o000000KvXbkAAF"
  },
  "Id": "001o000000KvXbkAAF",
  "Name": "Xamarin Corporation"
}
...
```

Record type

Record URL

What is a SearchResult?

- The Salesforce Component returns instances of **`SearchResult`**

```
public class SearchResult
{
  public SearchResult(JsonValue jv) { ... }

  public string Type { get; set; }

  public string Url  { get; set; }

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

Record type (e.g. Account) → `public string Type { get; set; }`

URL of the matching record → `public string Url  { get; set; }`

Id of the matching record → `public string Id   { get; set; }`

Mapping results to SearchResult

- Searches using the Salesforce Component prune the response down to the properties available in **`SearchResult`**

SOSL result from Salesforce

```
{
  "attributes":
  {
    "type": "Account",
    "url": "/services/data/v28.0/sobjects/Account/001o000000KvXbkAAF"
  },
  "Id": "001o000000KvXbkAAF",
  "Name": "Xamarin Corporation"
}
```

All data except Type, Url, and Id is discarded

`SearchResult` instance

```
Type Account
Url  /services/data/v28.0/sobjects/Account/001o000000KvXbkAAF
Id   001o000000KvXbkAAF
```

Recommended SOSL style

- Typically do not include field selections in RETURNING clause so only the Id is returned (**`Id`** is required in the results and other fields would be discarded in the mapping to SearchResult)

```
FIND {corp*} OR {inc*} IN NAME FIELDS RETURNING Account, Lead
```

Should list Objects to search as shown here, this will select only the Id field from each matching record

Xamarin University

Search method

- `SalesforceClient` extension method provides Search support

```
Task<IEnumerable<SearchResult>> SearchAsync(string search) ...
```

$\uparrow$ Results

$\uparrow$ SOSL search

Note: there is also a synchronous version of this method that blocks the calling thread so is rarely needed.

Xamarin University

How to Search

❖ Use **SearchAsync** to execute a SOSL search

SOSL →

Execute →

Process results e.g. retrieve by Id and display →

```
SaleforceClient client;
...
var search = "FIND {corp*} IN NAME FIELDS RETURNING Account, Lead";

var results = await client.SearchAsync(search);

foreach (var result in results)
{
  string id           = result.Id;
  string resourceName = result.Type;
  // ...
}
```

Individual Exercise

Perform a search

Xamarin University

Summary

1. Write a SOSL search
2. Execute a search

Thank You!

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

Microsoft