ENT410

Data Caching and Synchronization

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

1. Determine your Connectivity Strategy
2. Cache Results from a Server
3. Synchronize to a Remote Server
4. Evaluate Data Sync Tools

Determine your Connectivity Strategy

Xamarin University

Tasks

1. Make your network calls more resilient
2. Choose a suitable data strategy for your mobile application

Motivation

- Connection failure is common on phones; you must program defensively around all network-access code

Airplane mode turned on, signal lost, etc.

User expectations

- Users of modern apps expect a rich offline experience

Most standard iOS apps work offline although some offer reduced functionality

Connectivity testing

- Your app should test for connectivity before beginning any network operation

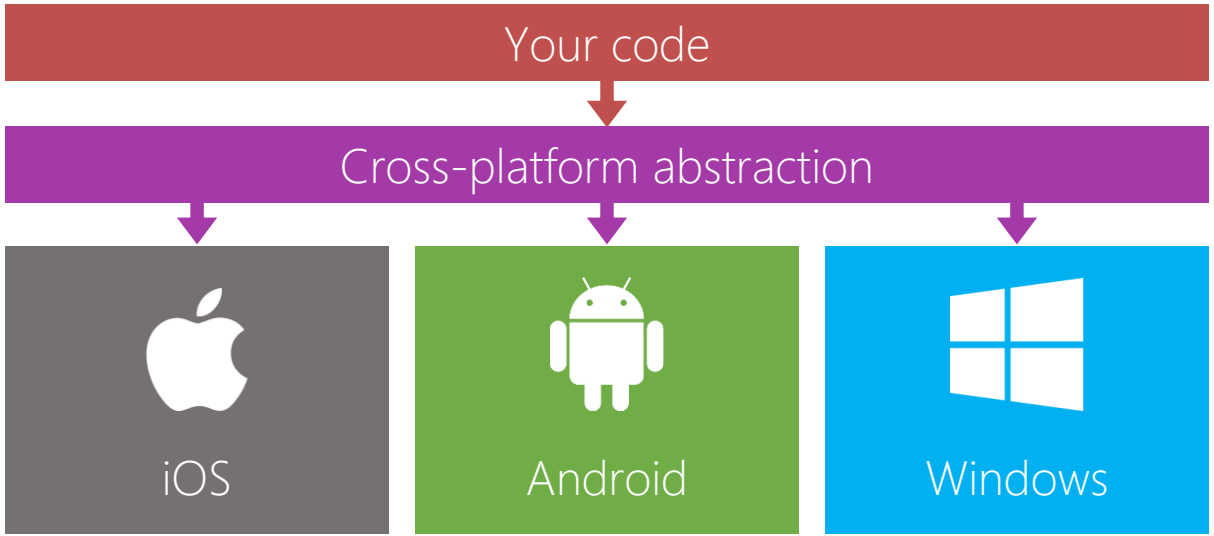
Platform connectivity APIs

- Each platform has unique APIs to detect, monitor, and work with networking hardware

You can use the platform APIs; however, it can mean writing similar code multiple times

Cross-platform connectivity APIs

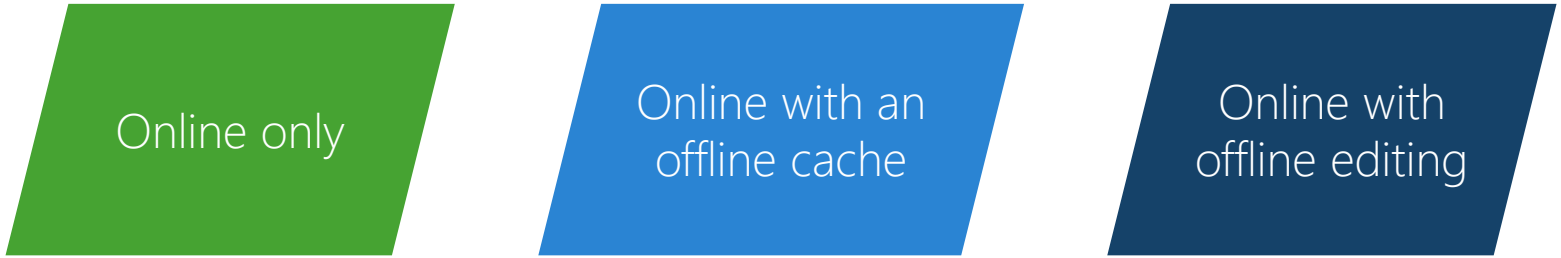
- There are cross-platform plugins that let you test connectivity from your shared code



See https://github.com/jamesmontemagno/Xamarin.Plugins/tree/master/Connectivity

Networking strategies

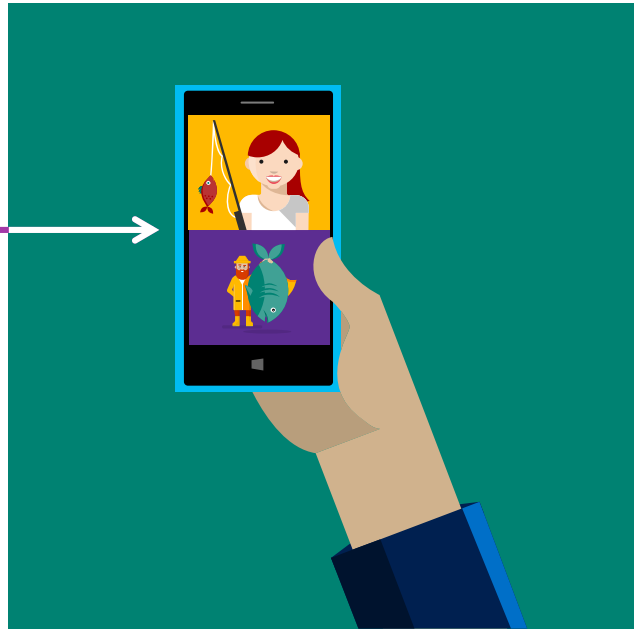
- Networked applications generally use one of three strategies for data management



Online-only apps

- Some apps can only function when an active network connection is available

Apps like messaging, VOIP, or online banking require a connection



User notification

- Inform the user when the application is offline and unable to retrieve data so they know the full functionality is unavailable

Summary

1. Make your network calls more resilient
2. Choose a suitable data strategy for your mobile application

Cache Server Results

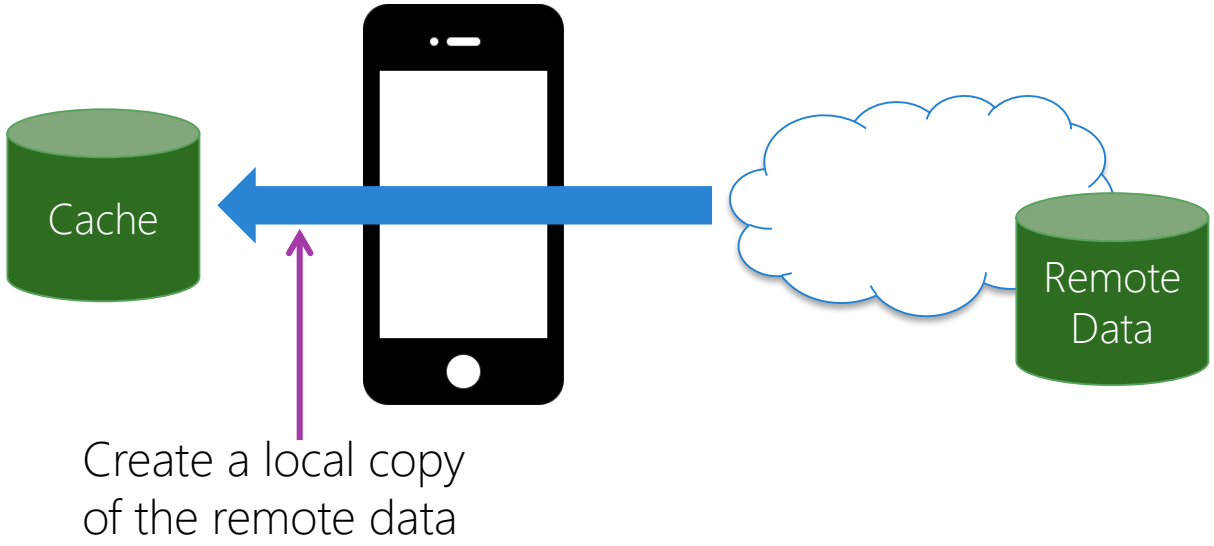
Xamarin University

Tasks

1. Cache your data by saving network requests to the device database

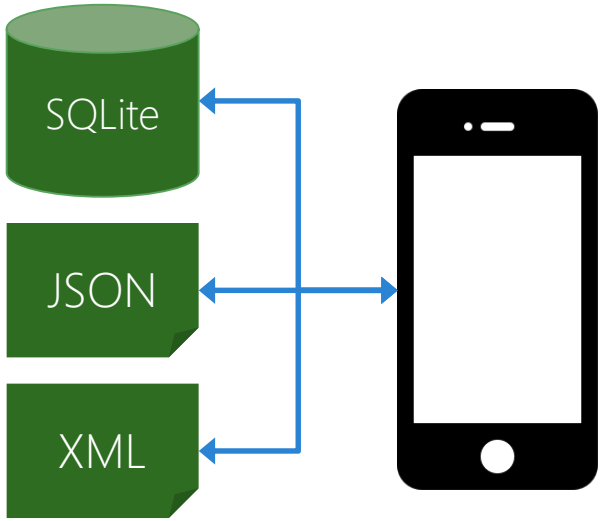
What is data caching?

- *Data caching* is storing the results of network calls locally on the device



How to implement a cache

- A local SQLite database is a common caching mechanism but other techniques such as flat files can work well



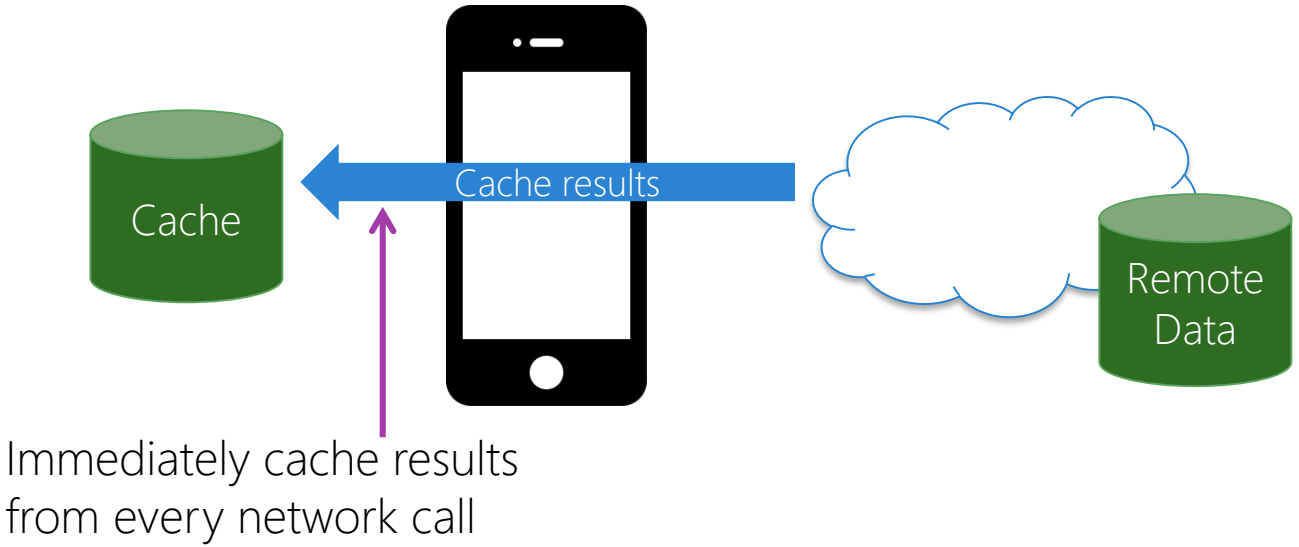
What data to cache

- You should cache data that has a long lifespan – do not cache data that changes frequently

Good	Bad
▪ Static data such as locations ▪ Information that will be accessed frequently that may have a timeout assigned	▪ Frequently changed data such as weather or bank balances ▪ Time sensitive information

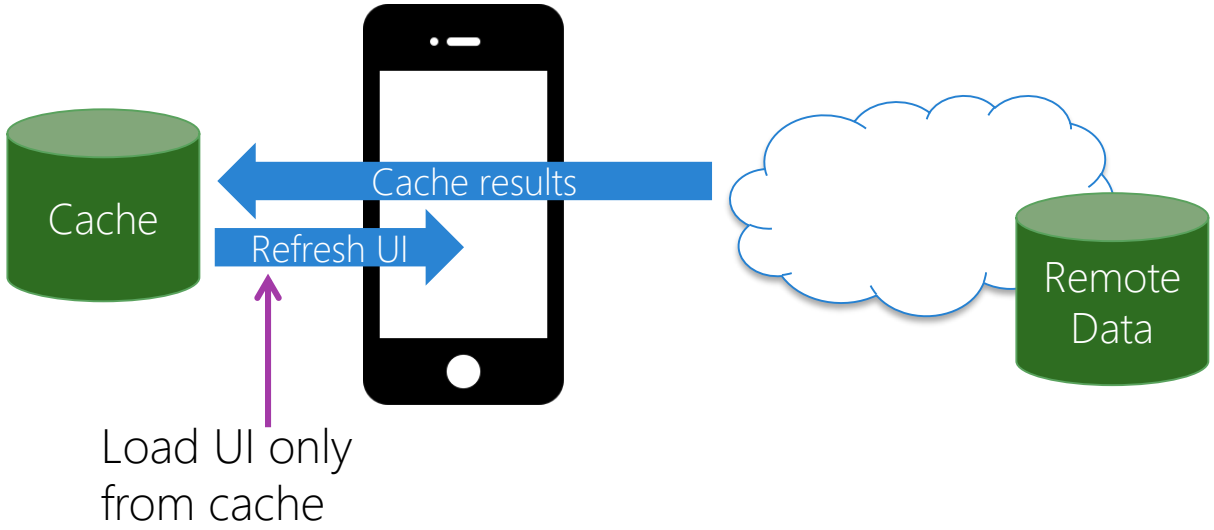
When to cache?

- Cache data as soon as you obtain it so that it is available even if you lose connectivity or the user starts your app while offline



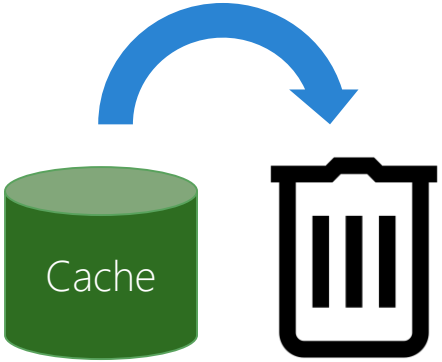
When to used cached data

- Always populate your UI from cache – this makes it simple and seamless to continue working while offline



Time-to-live policy

- You should purge data from the cache when it is no longer useful – how often to do this is highly data-dependent



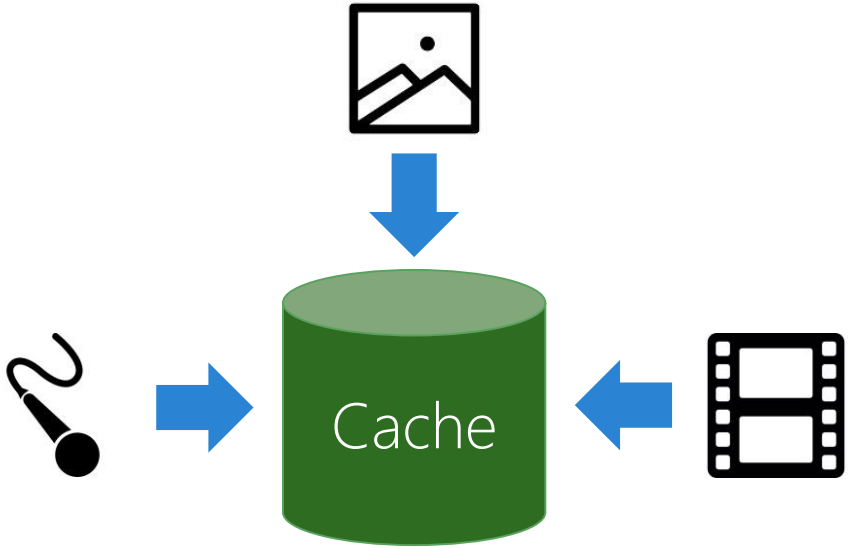
Coding offline support

- Refresh the data in the local cache from the remote server when it is available and have the UI view the information from the cache

```
async Task<IList<Job>> LoadPageFromNetwork()
{
    // Get the data from our service
    var service = new JobDataService();
    var data = await service.GetJobsForSearch(LastSearch, CurrentPage);
    ...
    // Store the jobs in the local database for online/offline use
    await App.DataManager.CoreDatabase.StoreJobs(data);
    return data;
}
```

Caching assets

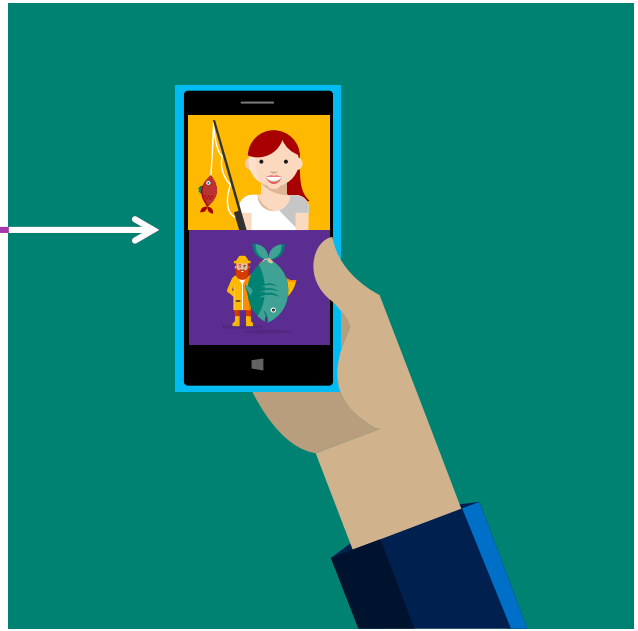
- Assets such as images, video, audio, etc. are good candidates for caching because they tend to be static and take time to download



Xamarin.Forms cache support

- Xamarin.Forms **`UriImageSource`** automatically caches images – by default images are cached for 1 day

Stored locally on device (storage location varies by platform)



Caching assets

- You can extend the default cache time for **`UriImageSource`**

```
public class ImageCacheConverter : IValueConverter
{
    public int DaysToCache { get; set; } = 30;
    public object Convert (object value, Type targetType, object parameter,
         CultureInfo culture)
    {
        return new UriImageSource {
            Uri = new Uri(value.ToString ()),
            CachingEnabled = true,
            CacheValidity = new TimeSpan(DaysToCache, 0, 0, 0, 0)
        };
    }
    ...
}
```

Store review process

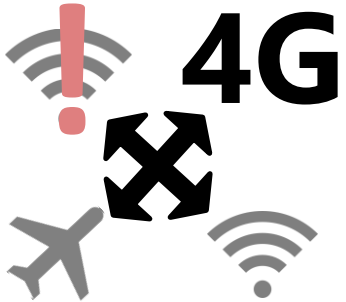
- Microsoft and Apple will test your app in airplane mode during the review process, ensure your application handles this transition gracefully

Recommended testing scenarios

- Test your applications where the network connectivity is poor or intermittent such as moving such as trains, subways, and tunnels

Unreliable network

Flight mode

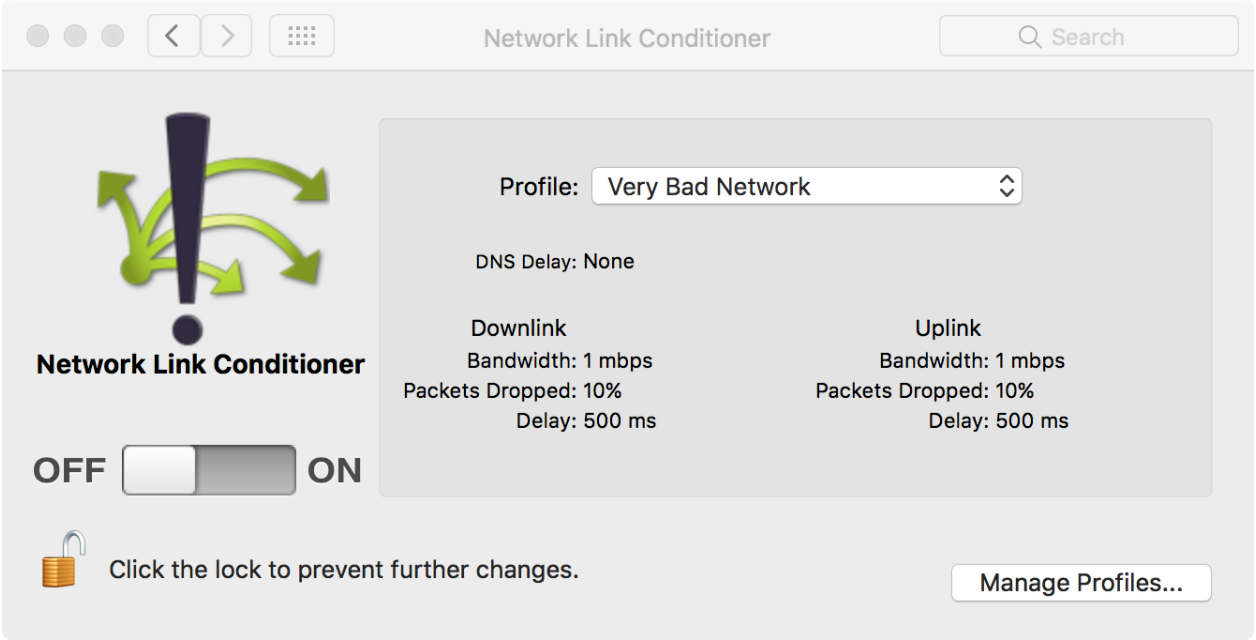


Switching between networks

Reliable networks

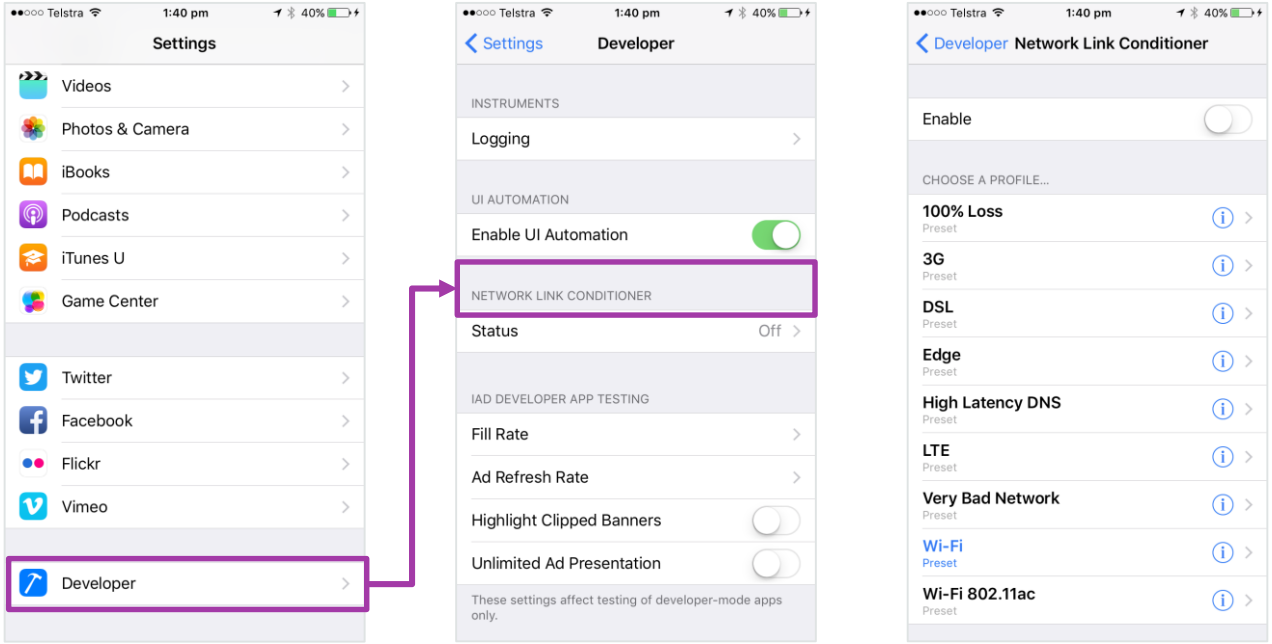
Testing on iOS simulators

- The *XCode Additional tools* let you simulate conditions such as 3G, 4G, LTE or even on a network that drops 100% of the packets



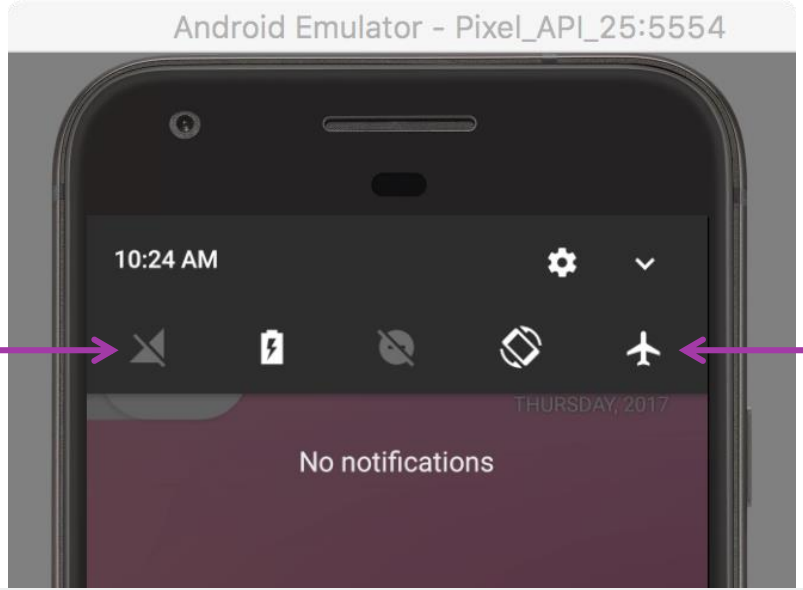
Testing on iOS devices

- Enabling developer mode on an iOS device lets you run the Network Link conditioner directly on the device



Testing on Android emulators

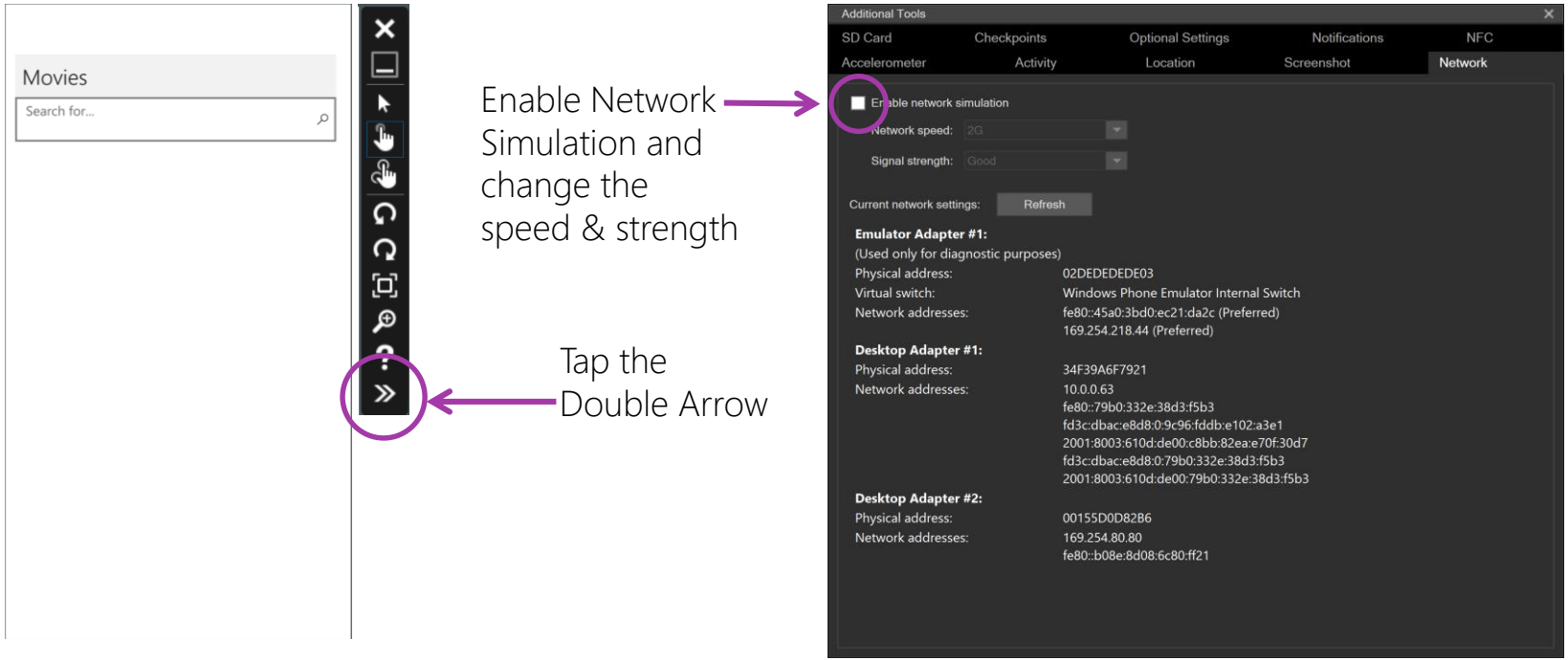
- Android emulators perform close to the native platform so you can use the regular OS features to perform several network tests



When running on the Mac, you can also use the Network Link Conditioner

Testing on Windows Phone emulators

- Windows simulator lets you toggle the network state



Individual Exercise

Caching Downloaded Data

Evaluate Data Sync Tools

Xamarin University

Tasks

1. Discuss patterns for offline editing synchronization
2. Examine challenges with Synchronization

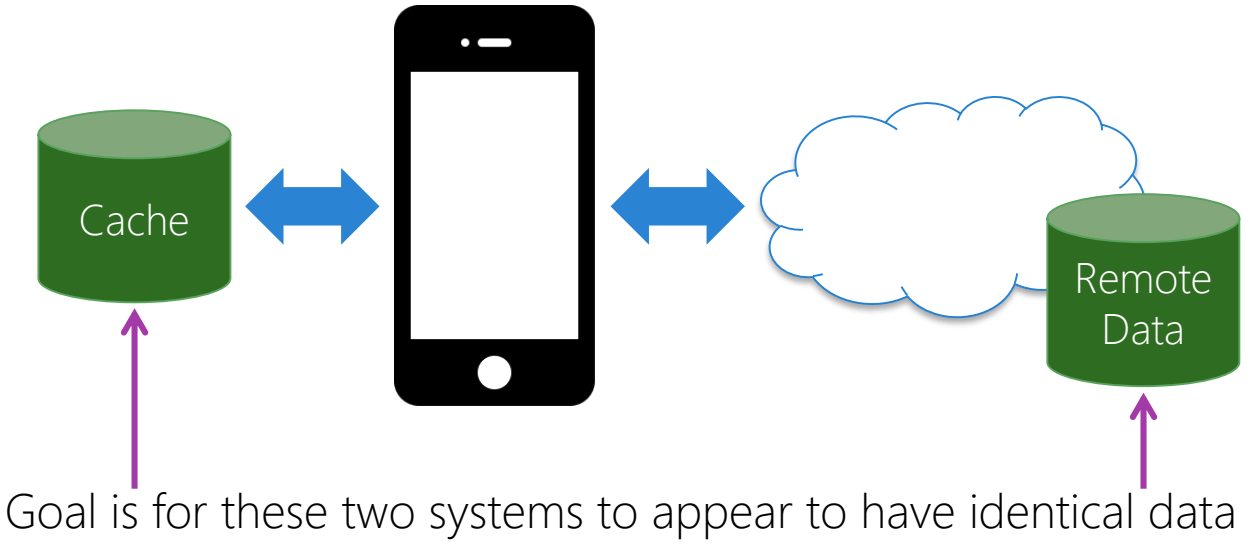
Best Practices

Motivation

- There are many situations where the ability to work offline will increase user productivity and therefore increase their satisfaction with your app

What is Data Synchronization?

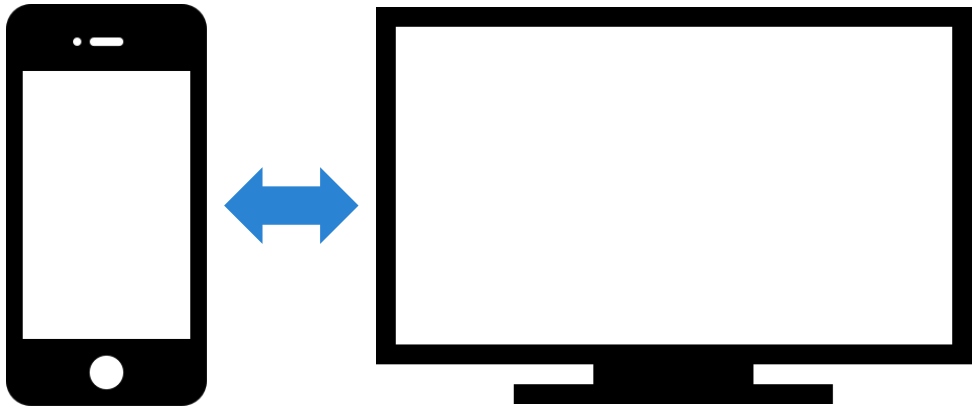
- *Data Synchronization* is the ability to make data changes on a local device and then merge the changes to a remote 'source of truth'



Sync requires client and server code

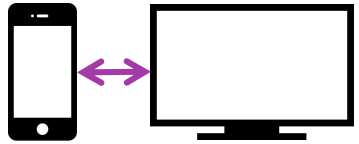
- Having control of both the client and server is needed to do sync *well* – this course will show code on both sides

Xamarin.iOS
Xamarin.Android
Windows

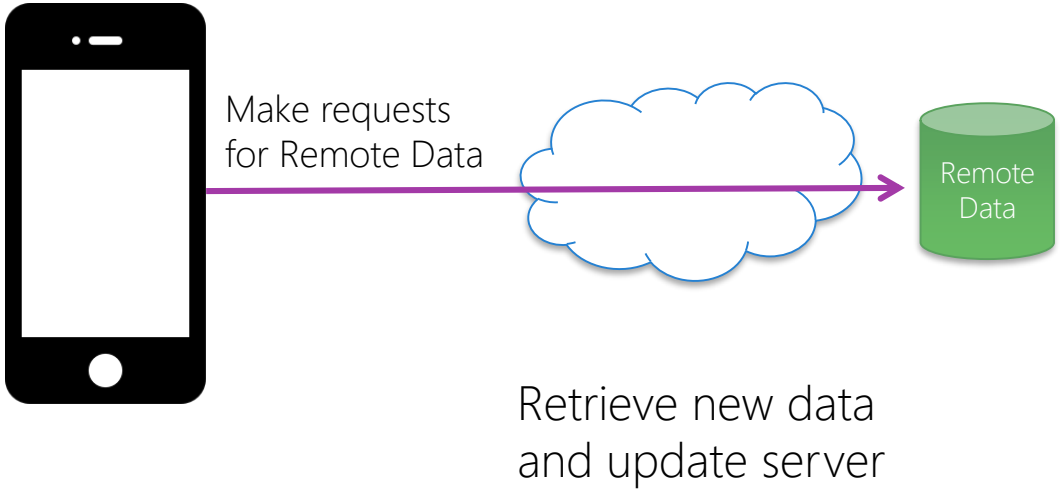


Web Api
ASMX
WCF
Service Stack

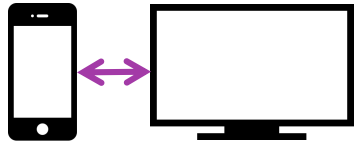
Offline editing and synchronization



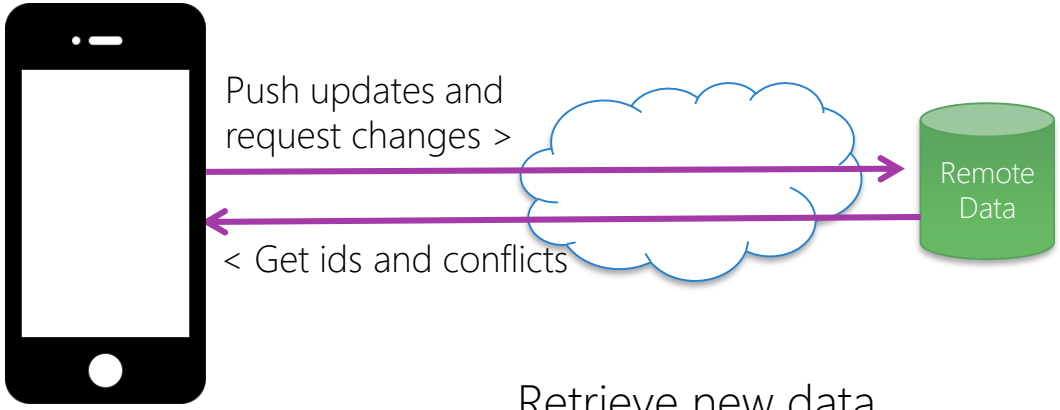
- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both



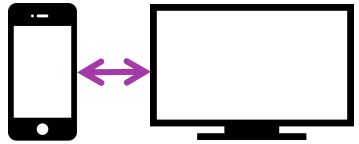
Offline editing and synchronization



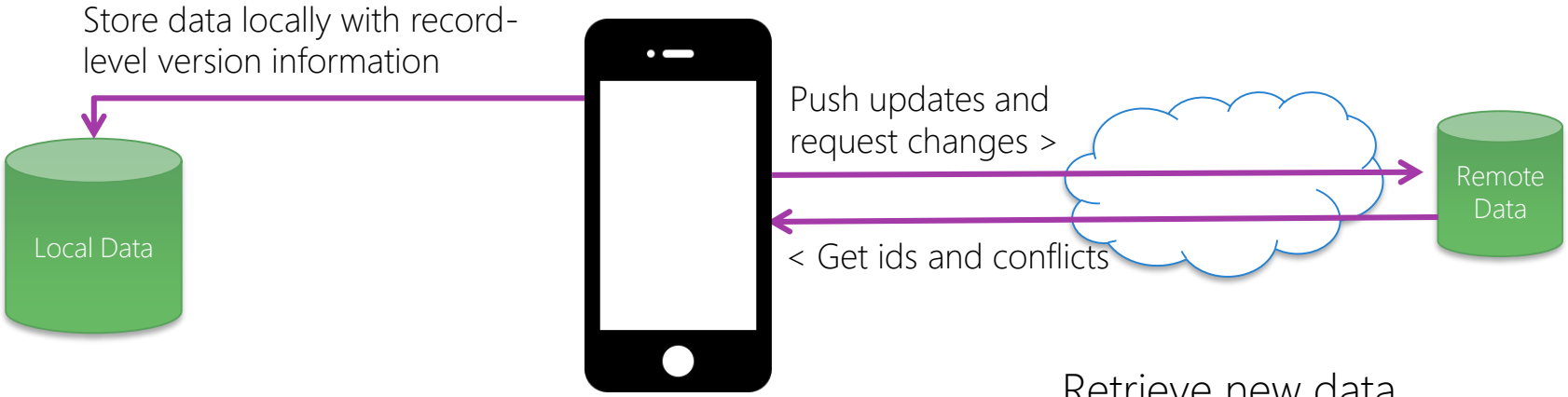
- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both



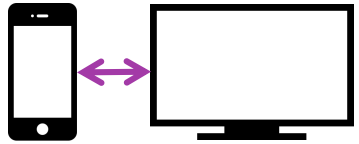
Offline editing and synchronization



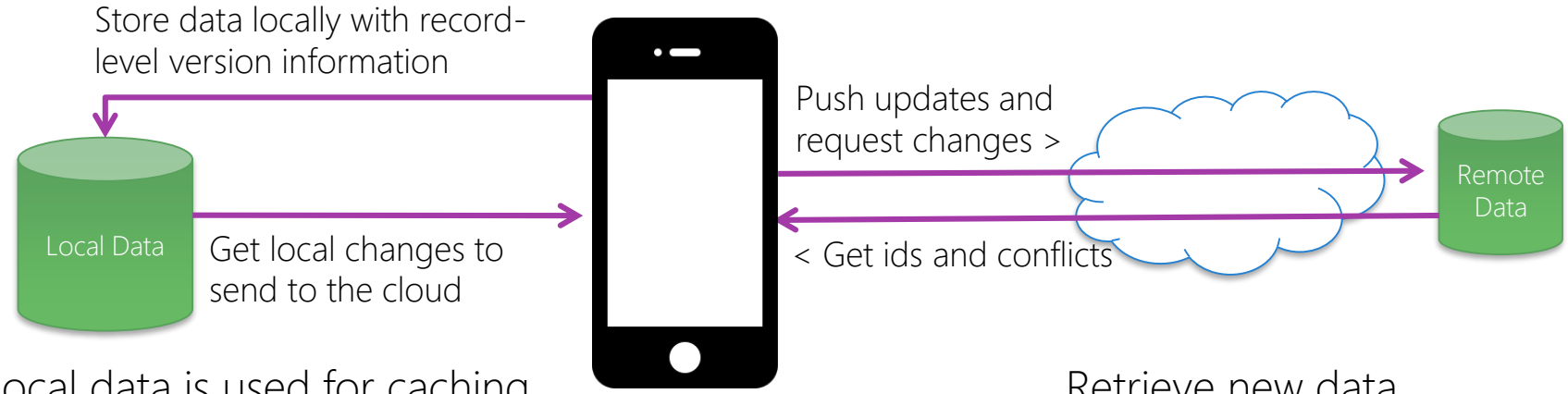
- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both



Offline editing and synchronization



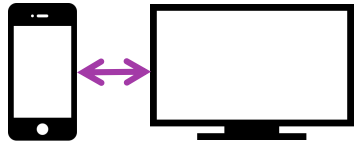
- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both



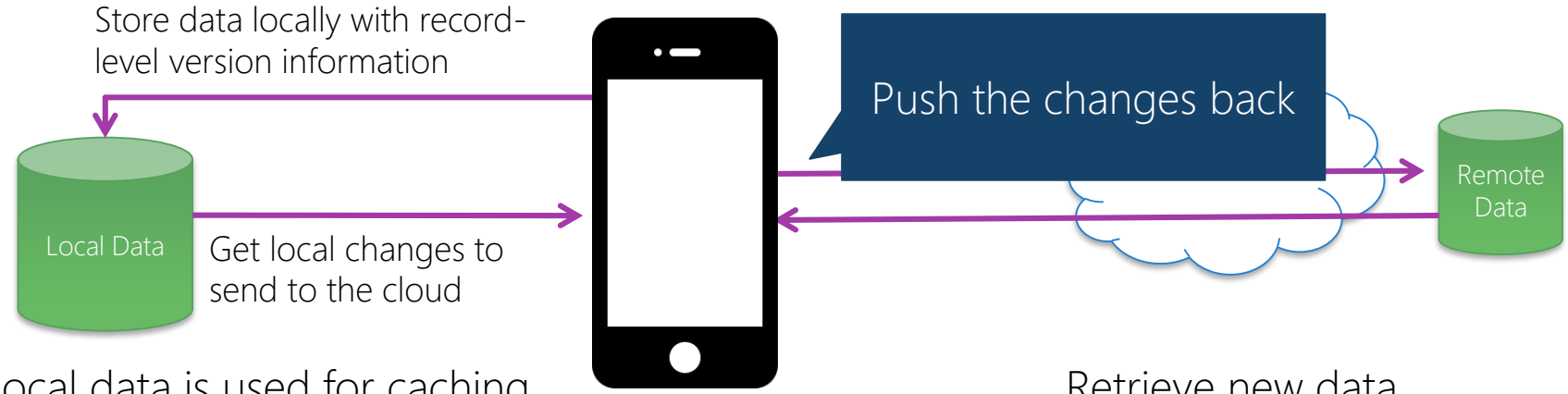
Local data is used for caching data and syncing local edits to a server

Retrieve new data and update server

Offline editing and synchronization



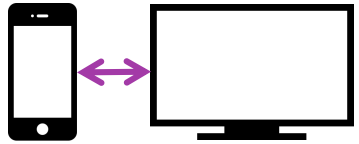
- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both



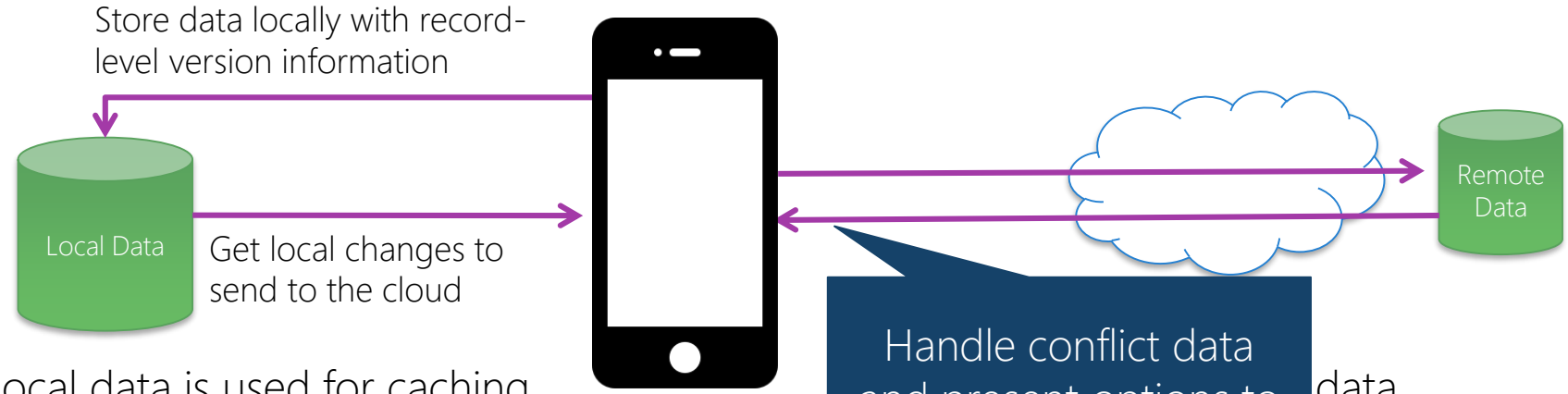
Local data is used for caching data and syncing local edits to a server

Retrieve new data and update server

Offline editing and synchronization



- Most networked application will use either a server based approach to data, a local cache of the data or the ability to work offline with both

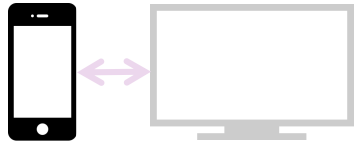


Local data is used for caching data and syncing local edits to a server

data
erver

Managing local data synching & conflict resolution significantly increases app complexity.

Creating objects to sync

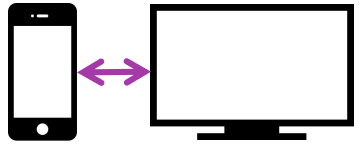


- Each business object should have its own entity - the class should represent the core information that you want to communicate between the server and the client

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public string Notes { get; set; }
    public string[] Addresses { get; set; }
}
```

Define these classes in a separate PCL so they can be shared between the mobile app and Web Server

Reuse model classes

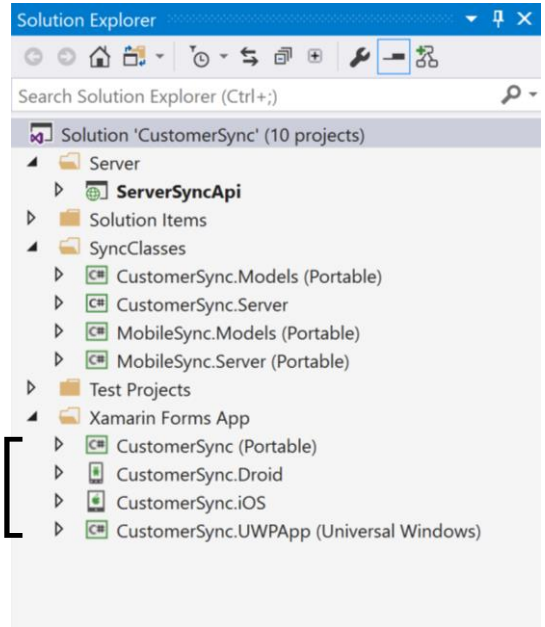


- Reuse model assemblies between the server and mobile clients to assist serialization and deserialization

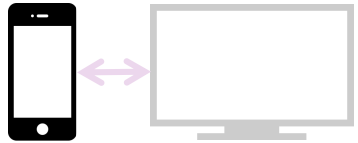
Reference the PCL from the WebApi project

Create the model classes as a PCL

Reference the PCL from the Mobile Apps



Challenge: Inserting offline records



- When the remote server is not available you'll need to store a local primary key so when the server is eventually inserted you can map the generated primary to the local copy

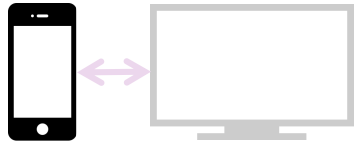
```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    ...

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

Id is the primary key on the server

CorrelationId is used for offline inserts

Challenge: Updating records



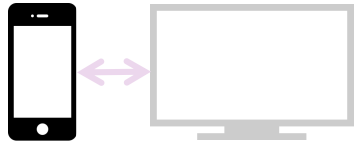
- When two devices have a local copy of the record and one applies an update to the record and the other device attempts to update the server, the second will be updating from a previous version

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    ...

    public int VersionNumber { get; set; }
}
```

Version number is used to ensure you update the version of record you have been editing or report a conflict

Challenge: Deleting records



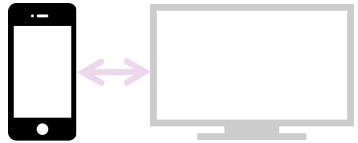
- When deleting records you should store the fact that a local record has been deleted, so synchronization will remove the record on the server

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    ...

    public int VersionNumber { get; set; }
    public bool IsDeleted { get; set; }
}
```

The local deleted record and version number is used to tell the server what to remove when syncing

Challenge: Handling conflicts



- When update or delete conflicts occur you'll need to display the information about the records in conflict and also the timing details

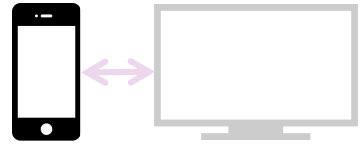
```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    ...

    public DateTime CreateDateTime { get; set; }
    public DateTime LastUpdateDateTime { get; set; }
    public DateTime DeletedDateTime { get; set; }
}
```

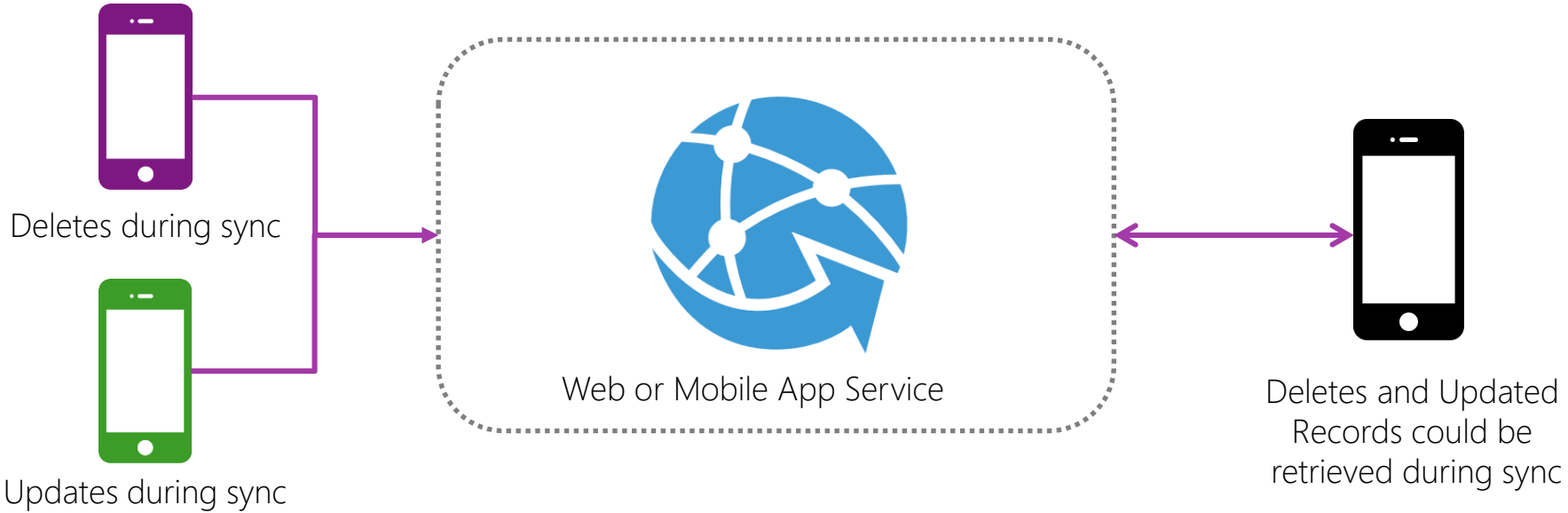
This information is used to display timing information about the conflict

Hint: use UTC for your records, so the server matches the clients

Challenge: Changes from others

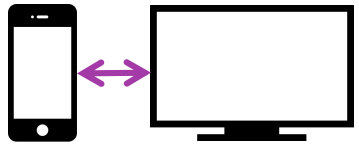


- Deletes and updates from other devices/users should be retrieved also



Look to the business rules to ensure that you need this capabilities

Structure your sync



- All this information is common to objects that will be synchronized, so it can be put in a base class

```
public class SyncObject
{
    public int Id { get; set; }
    public int VersionNumber { get; set; }

    public DateTime CreateDateTime { get; set; }
    public DateTime LastUpdateDateTime { get; set; }
    public DateTime DeletedDateTime { get; set; }

    public bool IsDeleted { get; set; }
    public string  CorrelationId { get; set; }
}
```

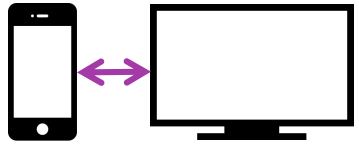
Use as a base class

Update only the version you edit

Store dates for later conflict display

CorrelationId is used for offline inserts

Structure your sync

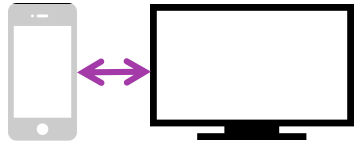


- The customer then becomes an extension of a class with synchronization capability

```
public class Customer : SyncObject
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public string Notes { get; set; }
    public string[] Addresses { get; set; }
}
```

Entity objects become simpler and focus only on their data requirements

Processing sync updates

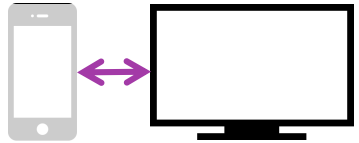


- On the server side, it is helpful to have an abstract capability to receive and process updates

```
public class BaseServerSync<T> where T : SyncObject
{
    public virtual Task<T> GetItemAsync (T item);
    public virtual Task<int> InsertAsync (T item);
    public virtual Task UpdateAsync (T item);
    public virtual Task DeleteAsync (T item);
    ...
}
```

Need the ability to receive and process single items against a data store

Processing sync updates

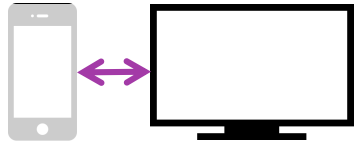


- On the server side, it is helpful to have an abstract capability to receive and process updates

```
public class BaseServerSync<T> where T : SyncObject
{
    ...
    public virtual Task AuditAsync (AuditAction action, T item);
    protected virtual Task SetupAsync ();
    protected virtual Task CommitAsync ();
    protected virtual Task RollbackAsync ();

    public virtual Task<SyncResult<T>> ProcessAsync (
        IEnumerable<T> items, bool forceChanges = false);
}
```

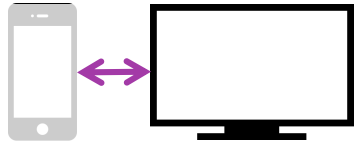
Processing sync updates



- Implement a custom subclass of sync to **`BaseServerSync`** to update an **`SyncObject`** so that changes can be applied

```
public class CustomerDataSync : BaseServerSync<Customer>
{
   ...
}
```

Processing sync updates



- Can use a WebApi method to receive changes from a client

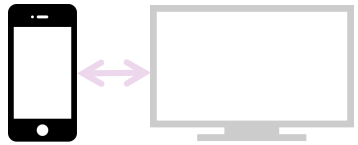
```
public class CustomersController : ApiController
{
    public CustomerDataSync _sync = new CustomerDataSync();

    public IEnumerable<Customer> Get() {
        return _sync.GetCustomers();
    }

    public SyncResult<Customer> Post([FromBody] Customer[] customers) {
        return _sync.Process(customers);
    }
    ...
```


WebApi is not a required mechanism, but it is a good mechanism for sharing the model code between the server and the mobile clients

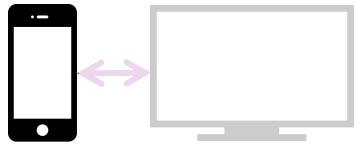
Connecting from the local data



- Send changes to the server and receive sync details

```
public async Task<SyncResult<Customer>> SyncData(List<Customer> items,
    bool forceChanges = false)
{
    using (var client = CreateRestClient())
    {
        string postBody = await JsonConvert.SerializeObjectAsync(items.ToArray());
        HttpResponseMessage getDataResponse;
        if (!forceChanges) {
            getDataResponse = await client.PostAsync ("",
                new StringContent (postBody, Encoding.UTF8, "application/json"));
        } else {
            getDataResponse = await client.PutAsync ("",
                new StringContent (postBody, Encoding.UTF8, "application/json"));
        }
```

Connecting from the local data



- Send receive changes between the server and the client.

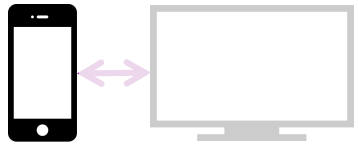
```
if (!getDataResponse.IsSuccessStatusCode)
                throw new CouldNotConnectException ();

        // Retrieve the JSON response
        jsonResponse = await getDataResponse.Content.ReadAsStringAsync()
            .ConfigureAwait(false);
    }

if (string.IsNullOrEmpty(jsonResponse))
        return null;

return await Task.Factory.StartNew(() =>
    JsonConvert.DeserializeObject<SyncResult<Customer>>(jsonResponse))
        .ConfigureAwait(false);
```

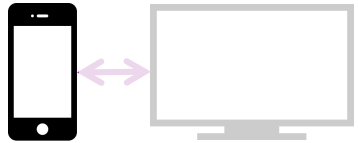
Platform-specific syncing



- For iOS 7 and above use Background Fetch mode to sync changes, even while the app is not running, can also use long running tasks

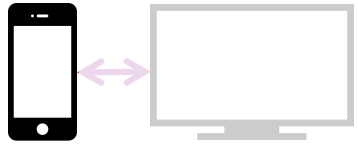
```
public override void PerformFetch (UIApplication application,
    Action<UIBackgroundFetchResult> completionHandler)
{
    try {
        var hasMoreData = await PerformSync();
        completionHandler(hasMoreData ? UIBackgroundFetchResult.NewData
            : UIBackgroundFetchResult.NoData);
    } catch {
        completionHandler(UIBackgroundFetchResult.Failed);
    }
}
```

Platform considerations

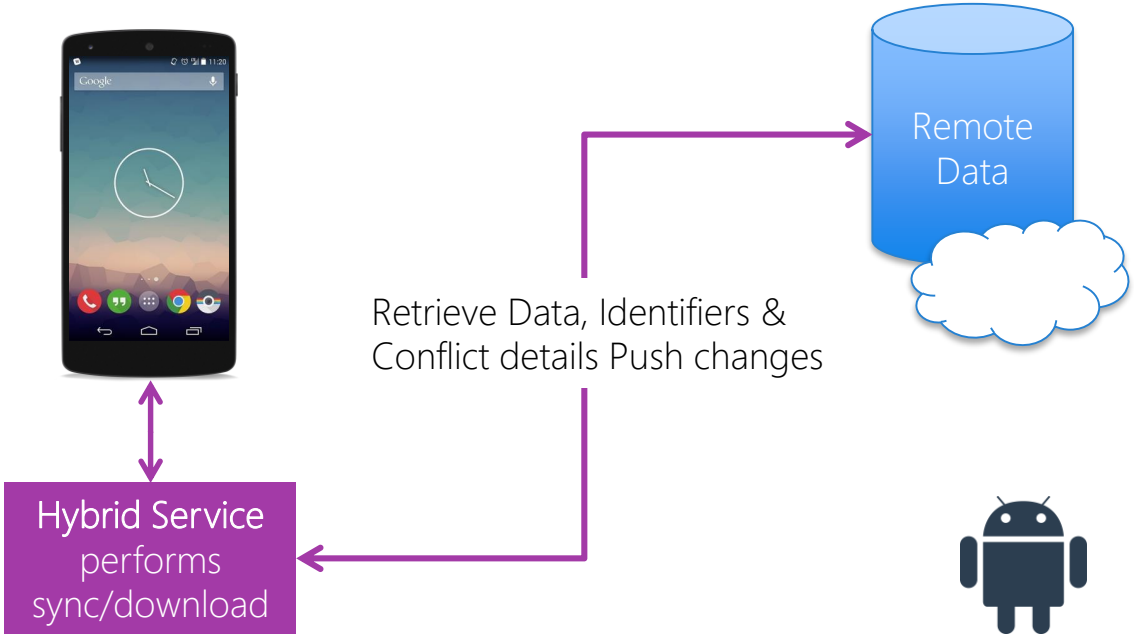


- For Android, ensure that synching is done via a Background service as the operations may take some time.

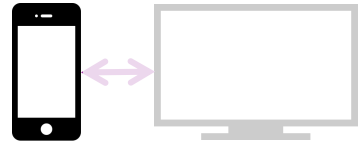
Platform Considerations



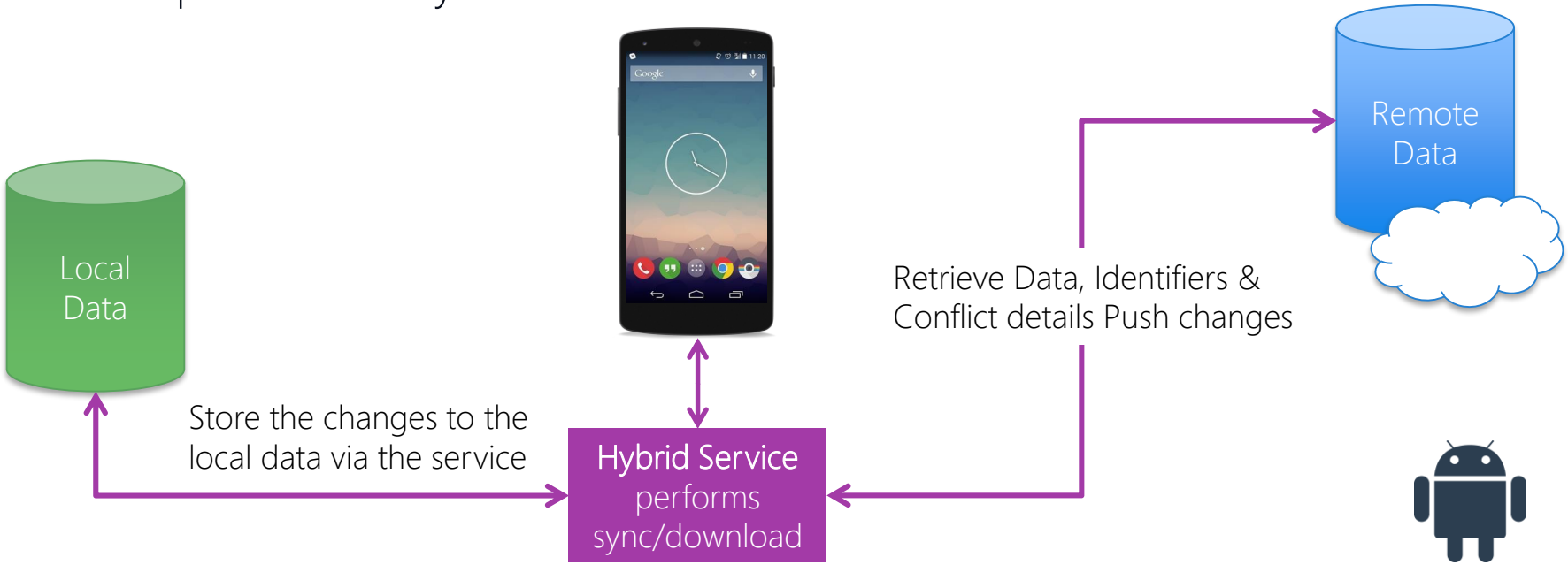
- For Android, ensure that synching is done via a Background service as the operations may take some time.



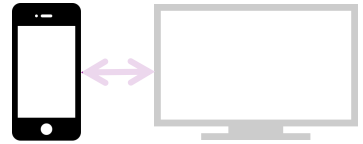
Platform Considerations



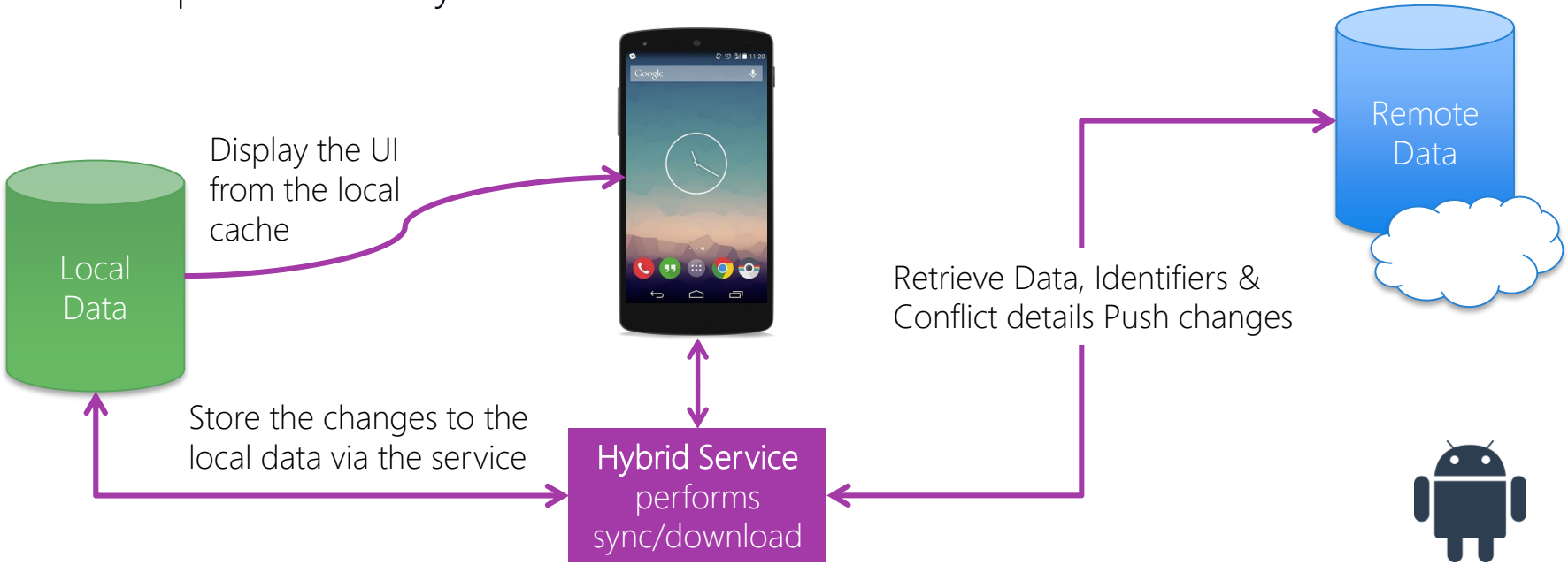
- For Android, ensure that synching is done via a Background service as the operations may take some time.



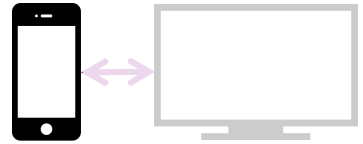
Platform-specific syncing



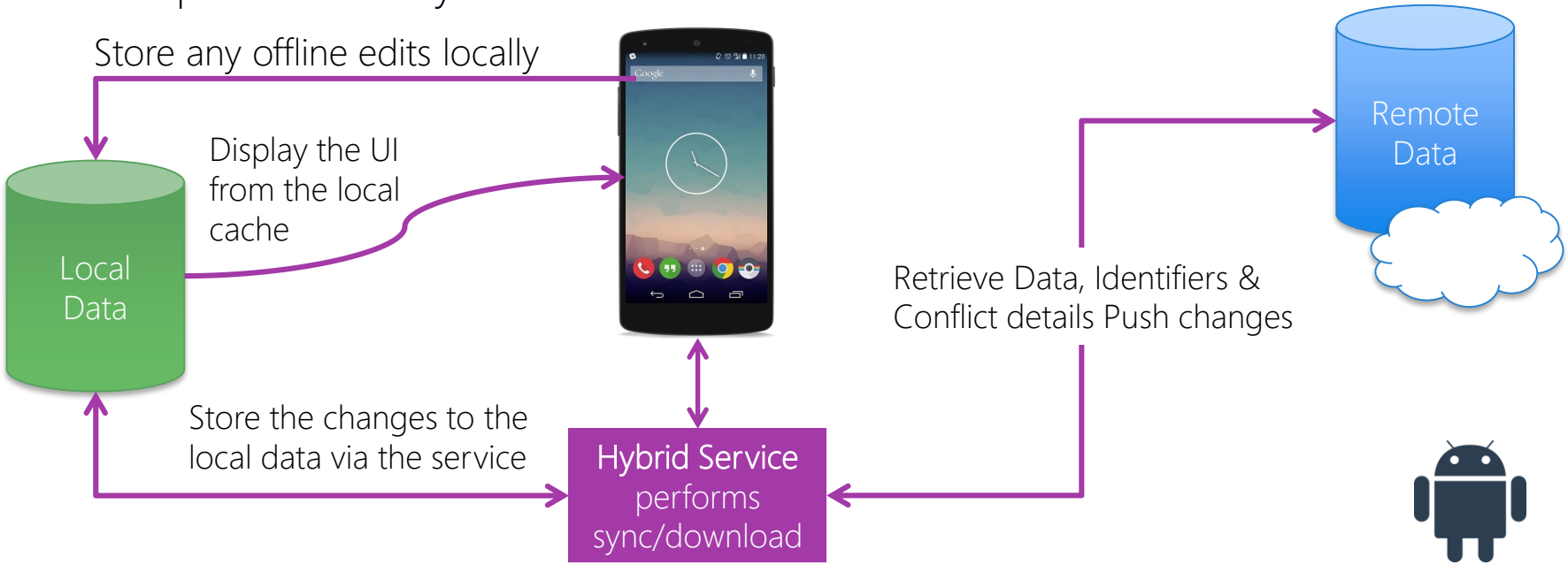
- For Android, ensure that synching is done via a Background service as the operations may take some time.



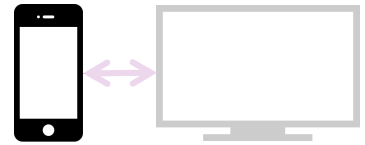
Platform-specific syncing



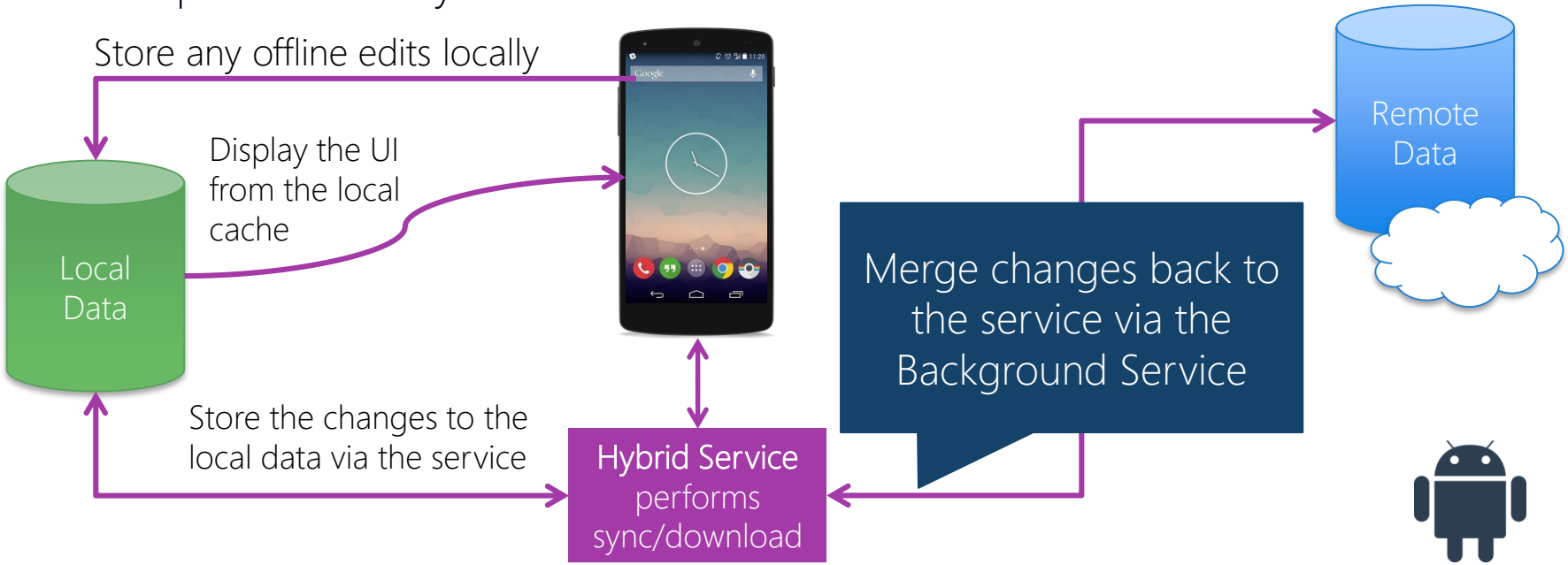
- For Android, ensure that synching is done via a Background service as the operations may take some time.



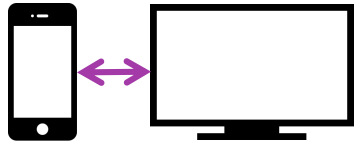
Platform-specific syncing



- For Android, ensure that synching is done via a Background service as the operations may take some time.

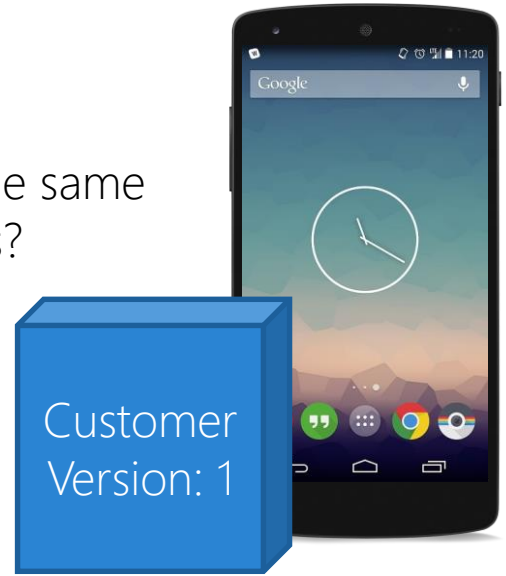


Handling conflicts

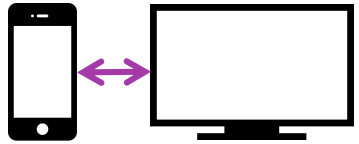


- There are two core conflicts that can occur, where users are updating the same record or where a user is updating a deleted record

When updating/deleting the same record what happens?

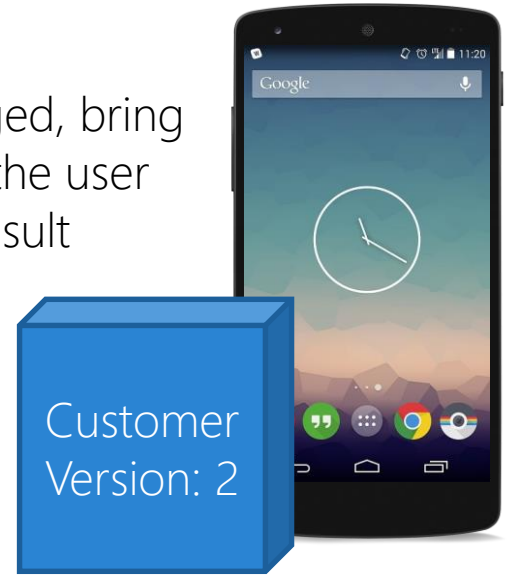


Handling update conflicts

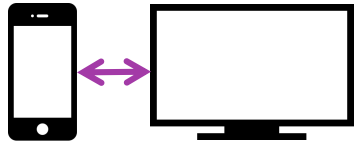


- When multiple users are editing the same record, who/what decides on which edit should be made

Where one record has changed, bring down both records and let the user decide and audit the result

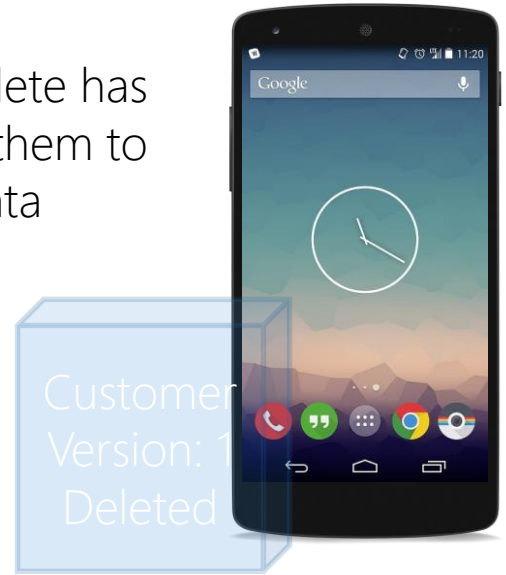


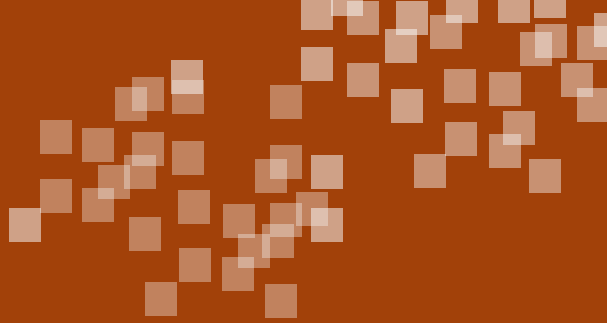
Handling delete conflicts



- When a user is updating a deleted record, what happens?

Inform the user that the delete has happened and encourage them to resynchronize their data





Demonstration

A coded offline synchronization mechanism

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz

① What are some of the challenges of synchronizing local offline data remotely (Choose all that apply)

a) Lack of network connectivity

b) No local stored procedures

c) Creating identifiers to insert

d) Handling conflicts between devices updating

e) All of the above

Flash Quiz

① What are some of the challenges of synchronizing local offline data remotely (Choose all that apply)

a) Lack of network connectivity
b) No local stored procedures
c) Creating identifiers to insert
d) Handling conflicts between devices updating
e) All of the above

Flash Quiz

② If you have offline editing capability, you don't need to warn users that there is no connection

a) True

b) False

Flash Quiz

② If you have offline editing capability, you don't need to warn users that there is no connection

a) True

b) False

Flash Quiz

③ When handling conflicts between two devices/users who have updated the record, you should:

a) Make the last update win

b) Display the two results and let the user decide

c) It depends on the business rules of the synchronization system

d) Make sure the records version numbers are the same

Flash Quiz

③ When handling conflicts between two devices/users who have updated the record, you should:

a) Make the last update win

b) Display the two results and let the user decide

c) It depends on the business rules of the synchronization system

d) Make sure the records version numbers are the same

Tasks

1. Discuss patterns for offline editing synchronization
2. Examine challenges with Synchronization

Evaluate data sync tools

Xamarin University

Tasks

1. Evaluate third-party options

Data sync code cost

- There is a significant amount of time writing, testing, deploying and managing a synchronized mechanism.

Time to develop a sync capability maybe too large

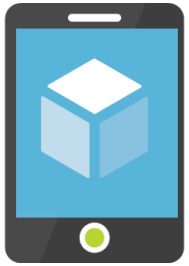
The risk of new development maybe too high

Enterprises may be affected by opportunity cost losses by being late to market

Coding your own sync gives you significantly more control.

Making pragmatic choices

- Writing Synchronization code is difficult and complex. If you can find a third-party offering you may complete your project sooner



Azure Mobile Apps

Couchbase

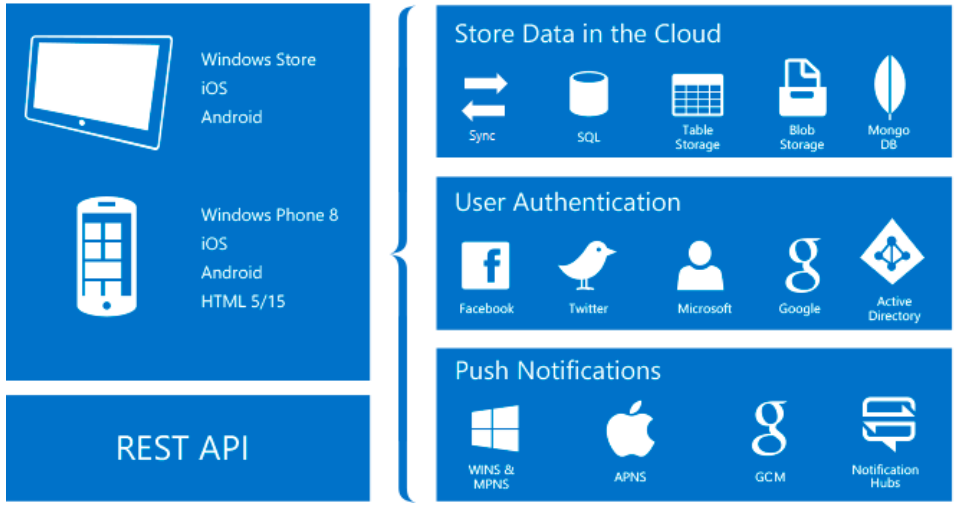
Zumero

You should always evaluate tools for suitability in *your* specific project

Azure Mobile Apps

- Azure Mobile Apps are a component of Azure App Service, which is a highly scalable mobile development platform that greatly simplify data caching and synchronization

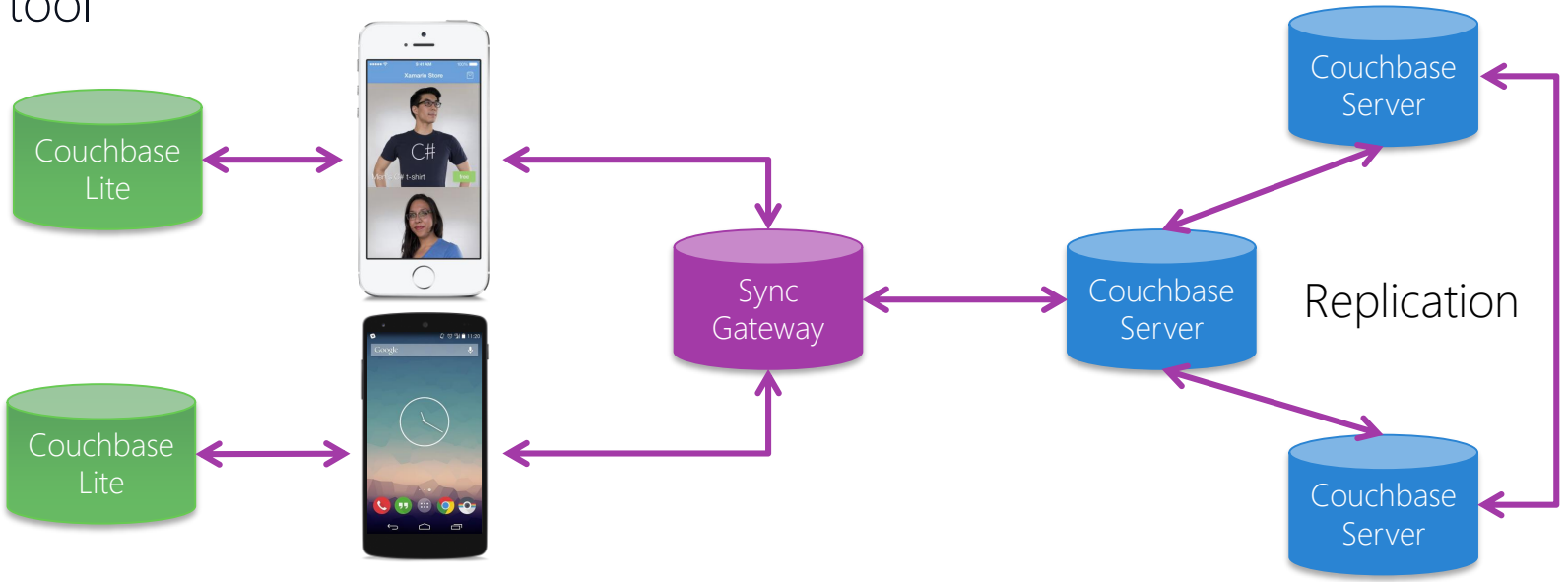
Broad mobile platform support including Windows, iOS, Android and Xamarin.Forms



For more information – checkout the Azure classes in Xamarin University

Couchbase

- Couchbase is a NoSQL-based data that by defaults supports replication between systems, including mobile devices through their SyncGateway tool



Couchbase

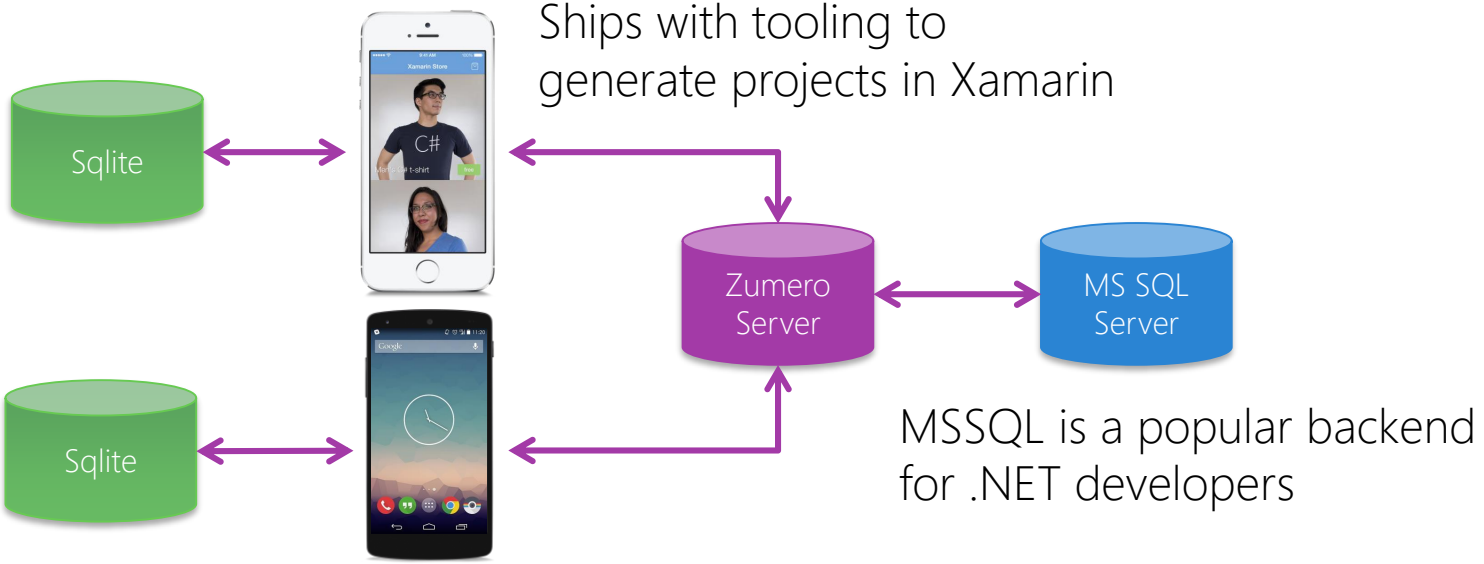
- Has very powerful replication support between the systems which almost feels like real-time communication

Pros	Cons
▪ Very quick/powerful replication ▪ LiveUpdates can be hooked into ViewModels which makes it good for Xamarin.Forms	▪ Requires a long setup process to begin working ▪ Requires understanding of NoSQL techniques such as map reduce ▪ Works best for Greenfields projects

A high-quality example of a Couchbase Xamarin.Forms app with Data Synchronization can be found at https://github.com/FireflyLogic/couchbase-connect-14

Zumero

- SQL based Synching between a mobile application and a customer system



Summary

1. Evaluate third-party options

Thank You!

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

Microsoft