Windows 10

# Лабораторный практикум

## Использование облачного сервиса Azure Mobile Apps в приложениях на платформе Windows

Октябрь, 2015 г.

Microsoft

# Описание

Azure App Service - это управляемый PaaS сервис для профессиональных разработчиков, предоставляющий широкие возможности для веб, мобильных и интеграционных сценариев.

Azure App Service предоставляет возможность развернуть любое веб-приложение на любом языке, получить готовый бэкенд для мобильных клиентов, инфраструктуру для API и готовый механизм рабочих процессов для управления потоками данных без написания кода.

Mobile Apps – мобильная служба, представляет собой функциональный модуль Azure App Service и является высоко масштабируемой, доступной из любой точки платформой для разработки мобильных приложений, которая предоставляет широкие возможности мобильным разработчикам и инструменты для быстрой и комфортной разработки корпоративных приложений.

Mobile Apps позволяет:

- **Создавать нативные и кроссплатформенные приложения.** Вы можете воспользоваться готовыми SDK службы не зависимо от того, какие приложения вы создаете: нативные iOS, Android и Windows приложения или с использованием кроссплатформенных технологий Xamarin и Apache Cordova.
- **Подключаться к системам и сервисам вашей компании**. С помощью Mobile Apps вы можете добавлять авторизацию с корпоративным аккаунтом за несколько минут и подключаться к ресурсам вашей компании, размещенными локально или в облаке.
- **Подключаться к API SaaS решений**. Более 40 коннекторов помогут вам легко интегрировать ваше приложение с SaaS решениями, которые использует ваша компания.
- **Реализовать офлайн-сценарии работы с приложениями.** Сделайте работу с вашими мобильными решениями более продуктивной, создавая приложения, которые могут работать в режиме офлайн и используют Mobile Apps для синхронизации данных в фоне при наличии соединения с любым из источников данных вашей организации или API SaaS решения.
- **Отправлять Push-уведомления миллионам пользователей за несколько секунд.** Взаимодействуйте с вашими клиентами, использующими любые устройства, с помощью технологии Push-уведомлений, персонализированными под нужды пользователей и отправленными в нужное время.

В этой лабораторной работе вы подключите универсальное приложение Windows (UWP) к мобильному бэкенду в Azure App Service. Сначала вы запустите службы мобильного бэкэнда локально на своём ПК и изучите код, для того чтобы понять, как клиентская часть мобильного приложения взаимодействует с серверной. Затем, вы добавите в клиентскую часть мобильного приложения аутентификацию через Azure Active Directory и подключите его к бэкэнду, размещённому на сервере в Microsoft Azure. Наконец, вы добавите в приложение работу с

офлайн сценарием и синхронизацией - одну из самых мощных возможностей мобильной службы Azure App Service.

## Цели

Настоящий курс научит вас:

- Запускать универсальное приложение Windows 10 (UWP), использующее бэкенд, расположенный локально и в облаке
- Подключать клиентское приложение к мобильному бэкэкнду в Azure.
- Реализовать аутентификацию в клиентском приложении через Azure Active Directory.
- Реализовать офлайн-сценарий работы приложения, с использованием локального хранилища данных SQLite и синхронизацией с мобильным бэкендом при восстановлении соединения.

---

## Системные требования

Чтобы выполнить эту лабораторную работу, вам понадобится следующее программное обсепечение:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

---

## Дополнительно

Если вы хотите выполнить все шаги лабораторной работы, включая дополнительные, вам понадобится:

- Windows 10 Mobile Emulator или телефон под управлением Windows 10 Mobile

## Настройка

Вам необходимо произвести следующие действия для подготовки своего компьютера к лабораторной работе:

1. Установите Microsoft Windows 10.
2. Установите Microsoft Visual Studio 2015. Выберите установку «Custom» и убедитесь, что в списке дополнительных функций выбран пункт «Universal Windows App Development Tools».
3. Дополнительно: Выберите пункт «Windows 10 Mobile Emulator».

---

## Упражнения

Эта лабораторная работа включает в себя следующие упражнения:

1. Запуск мобильной службы Azure App Service
2. Реализация аутентификации и подключение приложения к облаку
3. Реализация офлайн хранилища и синхронизации данных в приложении

---

Время для завершения этой лаборатории: **От 45 до 60 минут**.

# Упражнение 1: Запуск мобильной службы Azure App Service

В этом упражнении, вы воспользуетесь универсальным приложением Windows 10 (UWP), которое является трекером для записи задач (вы можете загрузить аналогичное приложение с портала Azure, при создании мобильной службы). Вы подключите приложение-клиент к мобильной службе, которую запустите локально на своём компьютере.

**Задание 1 – Знакомство с порталом Azure**

Мы начнём с описания того, как создавать мобильный бэкенд для приложения на портале Azure.

1. Вам не нужно создавать экземпляр мобильной службы в Azure App Service для этой лабораторной работы, так как мы заранее подготовили его для вас.
   Если вы хотите узнать, как самостоятельно создать мобильную службу Azure App Service , вам необходимо воспользоваться пошаговой инструкцией на портале: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-preview/.

   Посмотрим, как создать новую мобильную службу в Azure:

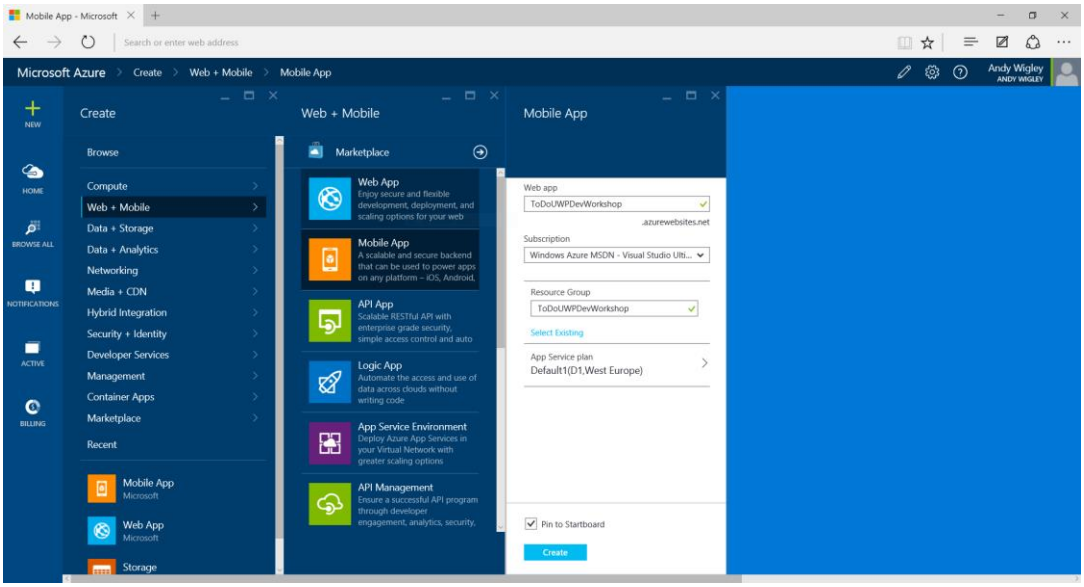


**Рисунок 1**

*Начало создания мобильной службы на портале Azure.*

2. После создания мобильной службы, вы увидите страницу с информацией:

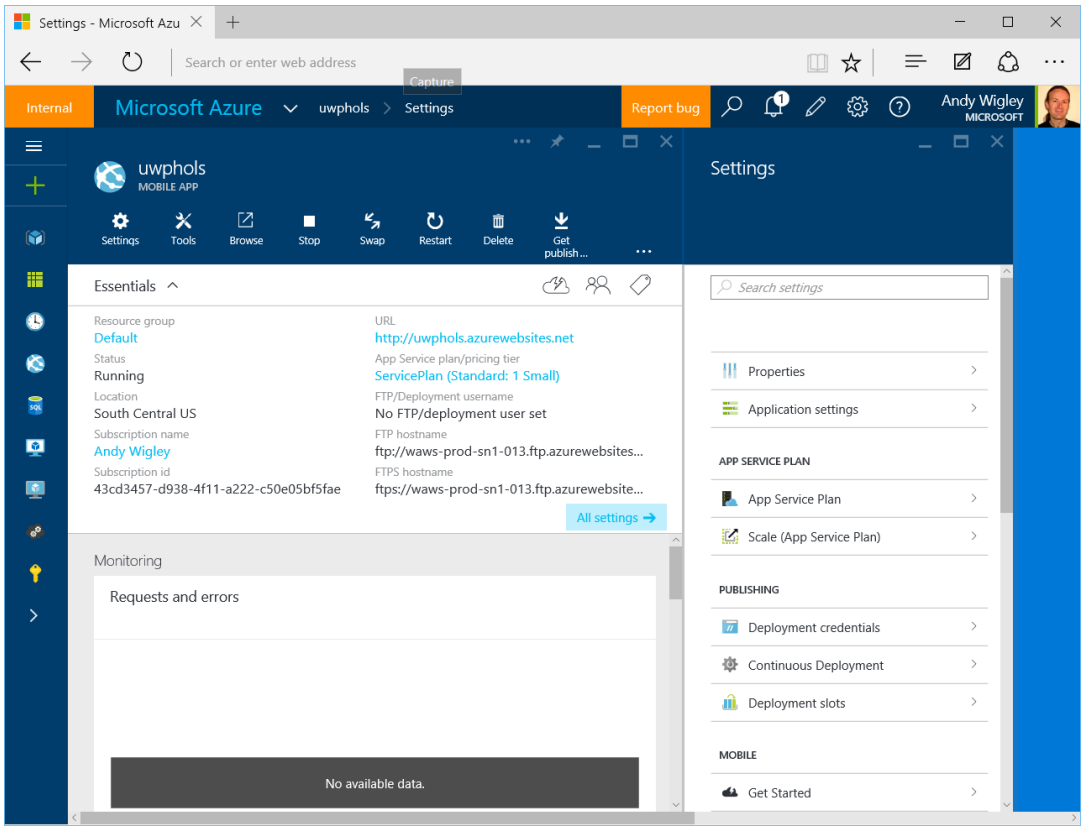


**Рисунок 2**

*Блок информации о мобильной службе на портале Azure.*

3. В блоке **Settings** вы найдете раздел **Mobile**, содержащий различные опции, включая **Get Started**. При нажатии на Get Started откроется панель Quickstart, где вы сможете получить инструкции для подключения вашего существующего приложения к мобильному бэкэнду в Azure или получите возможность загрузить примеры клиентской и серверной части мобильного приложения. Вы можете выбрать различные варианты мобильных приложений, включая Windows (C#), iOS (Objective-C), iOS (Swift), HTML/JavaScript, Xamarin.Android, Xamarin.iOS или Xamarin.Forms.

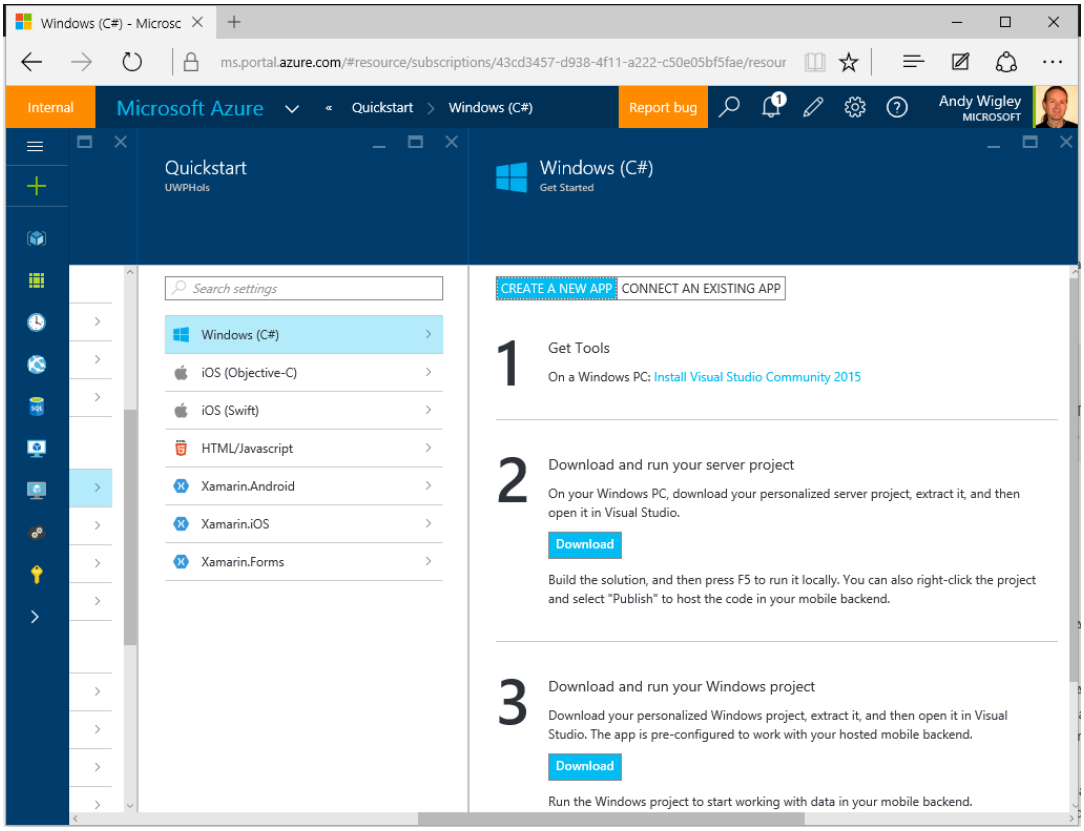


**Рисунок 3**

*Quickstart панель на портале Azure.*

4. На момент создания практикума (сентябрь 2015 года) для загрузки в качестве проекта Windows (C#) на Quickstart панели предлагается универсальное приложение Windows 8.1. Для этой лабораторной работы мы создали UWP версию этого клиента.

**Задание 2 – Запуск серверной части приложения на своем ПК**

Разберем код серверной части приложения, который можно загрузить при создании мобильного бэкенда в Azure. Вы сможете запустить его локально на своём ПК.

1. Откройте **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, в которой вы сохранили код для этой лабораторной работы, и откройте **Exercise 1\Begin-Cloud\UWPHolsService.sln**

2. Нажмите Build Solution, чтобы загрузить необходимые NuGet пакеты.

> **Примечание:** Этот проект содержит C# код для .NET бэкенда в Azure. Вы так же можете выбрать реализацию логики бэкэнда на NodeJS. Подробнее об этой возможности можно узнать на портале azure.com.

3. Этот проект содержит C# код для реализации логики бэкэнда. В папке **DataObjects** откройте файл **ToDoItem.cs**. Этот класс определяет объекты данных, которые облачный сервис сохраняет в базе данных SQL Azure:

**C#**

```
    public class TodoItem : EntityData
    {
        public string Text { get; set; }

        public bool Complete { get; set; }
    }
```

4. В папке **Controllers** откройте файл **TodoItemController.cs**. Этот класс содержит код, который определяет CRUD операции REST API облачного сервиса:

**C#**

```
     public class TodoItemController : TableController<TodoItem>
    {
        protected override void Initialize(
                                    HttpControllerContext
controllerContext)
        {
            base.Initialize(controllerContext);
            UWPHolsContext context = new UWPHolsContext();
            DomainManager =
               new EntityDomainManager<TodoItem>(context, Request);
        }

        // GET tables/TodoItem
        public IQueryable<TodoItem> GetAllTodoItems()
        {
            return Query();
        }

        // GET tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
        public SingleResult<TodoItem> GetTodoItem(string id)
        {
            return Lookup(id);
        }

        // PATCH tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
        public Task<TodoItem> PatchTodoItem(string id, Delta<TodoItem>
patch)
        {
```

```
            return UpdateAsync(id, patch);
        }

        // POST tables/TodoItem
        public async Task<IHttpActionResult> PostTodoItem(TodoItem item)
        {
            TodoItem current = await InsertAsync(item);
            return CreatedAtRoute("Tables", new { id = current.Id },
current);
        }

        // DELETE tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
        public Task DeleteTodoItem(string id)
        {
            return DeleteAsync(id);
        }
    }
```

5. В папке **App_Start** откройте файл **Startup.MobileApp.cs**. Этот файл содержит код для настройки службы, включая метод **Seed** класса **UWPHolsInitializer**. Этот метод выполняется при первом запросе к REST сервису, и добавляет два элемента, созданных внутри этого метода, в базу данных (используется только на начальном этапе):

**C#**

```
    protected override void Seed(UWPHolsContext context)
    {
        List<TodoItem> todoItems = new List<TodoItem>
        {
            new TodoItem { Id = Guid.NewGuid().ToString(),
                           Text = "First item", Complete = false },
            new TodoItem { Id = Guid.NewGuid().ToString(),
                           Text = "Second item", Complete = false },
        };

        foreach (TodoItem todoItem in todoItems)
        {
            context.Set<TodoItem>().Add(todoItem);
        }

        base.Seed(context);
    }
```

6. Для Solution Configuration выберите Debug, а для Solution Platform выберите Any CPU. Выберите Microsoft Edge из списка Debug Target справа от кнопки Start Debugging.

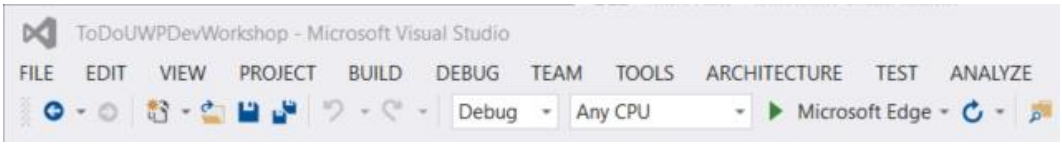


**Рисунок 4**

*Настройка веб-сайта для запуска на Локальном компьютере.*

> **Примечание:** ▶ Start Debugging – кнопка запуска отладки.

7. Выполните сборку и запуск приложения. В браузере вы увидите стандартную страницу мобильной службы Azure.

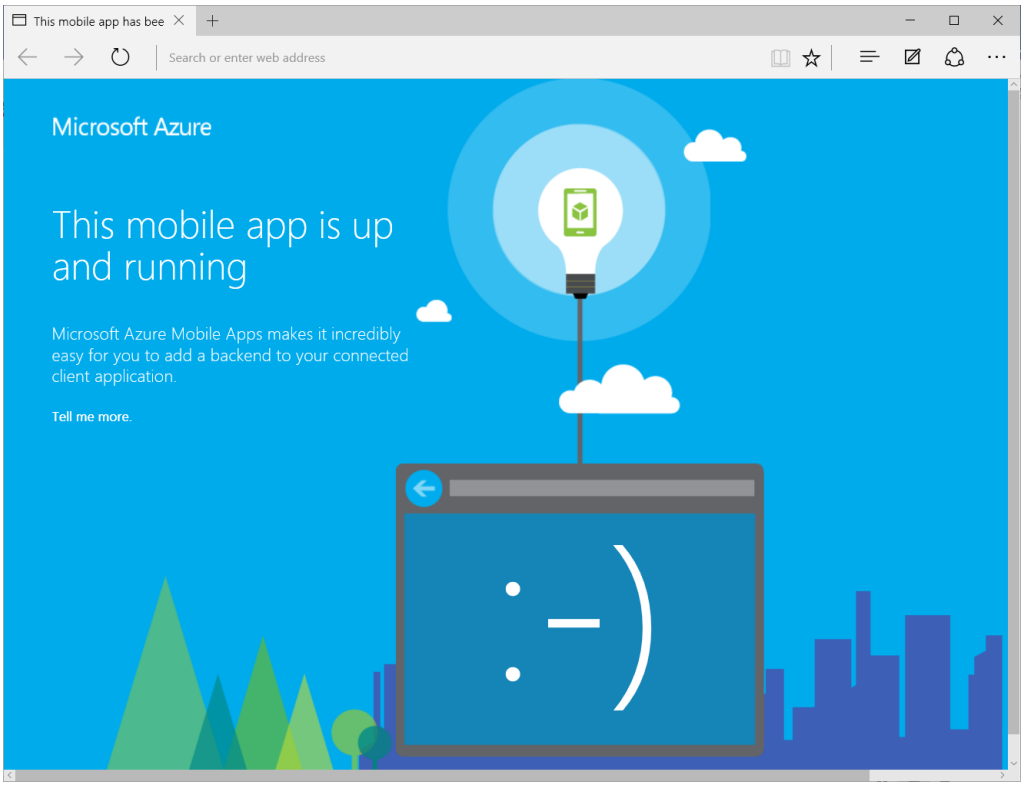


**Рисунок 5**

*Мобильный бэкенд Azure, запущенный локально на вашем ПК.*

> **Примечание:** Возможность запуска бэкэнда локально предоставляет вам еще один способ устранения неполадок в приложении.

**Задание 3 – Запуск клиентской части приложения**

Разберемся в коде UWP приложения и изменим его так, чтобы подключиться к мобильной службе, которая выполняется локально на вашем ПК.

1. Откройте второй экземпляр **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, где вы сохранили код для этой лабораторной работы, и откройте **Exercise 1\Begin-Client\ToDoUWPDevWorkshop.sln**.

2. Для Solution Configuration выберите Debug, а для Solution Platform выберите Any CPU **x86**. Выберите Local Machine из списка Debug Target справа от кнопки Start Debugging.

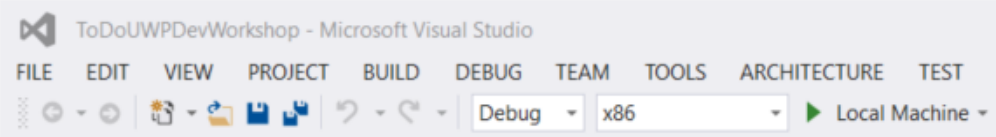


**Рисунок 6**

*Настройка клиентского приложения для запуска на локальном компьютере.*

3. Нажмите Build Solution, чтобы загрузить необходимые NuGet пакеты в проект.

4. Откройте файл **App.xaml.cs**. Обратите внимание, что в начале файла находится закомментированный код для создания экземпляра **MobileServiceClient**. Первые строчки необходимы, чтобы подключиться к облаку – оставьте этот код закомментированным. Следующий код подключает приложение к локальной версии облачного сервиса, который вы запустили в предыдущем задании.

**C#**

```
    sealed partial class App : Application
    {
        // Uncomment this code for configuring the MobileServiceClient to
        // communicate with the Azure Mobile Service and
        // Azure Gateway using the application key. You're all set to start
        // working with your Mobile Service!
        //public static MobileServiceClient MobileService =
        //    new MobileServiceClient(
        //    "https://uwpdevhols.azurewebsites.net",
        //    "",
        //    ""
        //);

        // Use this code for configuring the MobileServiceClient to
        // communicate with your local
        // test project for debugging purposes.
        public static MobileServiceClient MobileService =
            new MobileServiceClient(
            "http://localhost:50781"
        );
      }
```

5. Откройте файл **MainPage.xaml.cs**. В этом классе вы увидите, как в коде приложения реализуется обмен данными с облачным сервисом, загрузка данных из службы, добавление, обновление и удаление.

В начале файла объект **todoTable** имеет тип **IMobileServiceTable<TodoItem>** и инициализируется вызовом **App.MobileService.GetTable<TodoItem>()**. Этот объект используется по всему классу, чтобы выполнять операции для таблицы. Например, чтобы вставить новый элемент данных, вы используете метод **InsertAsync** объекта **todoTable**:

**C#**

```
    private IMobileServiceTable<TodoItem> todoTable =
         App.MobileService.GetTable<TodoItem>();

    ...

    private async Task InsertTodoItem(TodoItem todoItem)
    {
        // This code inserts a new TodoItem into the database. When the
        // operation completes and Mobile Services has assigned an Id,
        // the item is added to the CollectionView
        await todoTable.InsertAsync(todoItem);
        items.Add(todoItem);

    }
    ...
```

6. Выполните сборку и запуск приложения. Когда ваше приложение-клиент запущено, код в методе OnNavigatedTo в MainPage вызывает RefreshTodoitems, который в свою очередь вызывает службу бэкенда, чтобы извлечь все ToDo элементы, хранимые в базе данных бэкенда.

   **Примечание**: Если ваше приложение прекращает работу с ошибкой HttpRequestException, то это может происходить, потому что вы остановили отладку приложения UWPHolsService из предыдущего задания. Запустите снова проект-сервер, оставьте его в запущенном состоянии и затем запустите приложение-клиент в отдельной копии Visual Studio.

7. Так как это был первый раз, когда был осуществлён доступ к службе, она автоматически настроит базу данных и запустит метод Seed, изученный нами ранее и добавляющий два сгенерированных внутри метода элемента. Через некоторое время вы увидите два элемента, отображаемые в интерфейсе приложения-клиента.

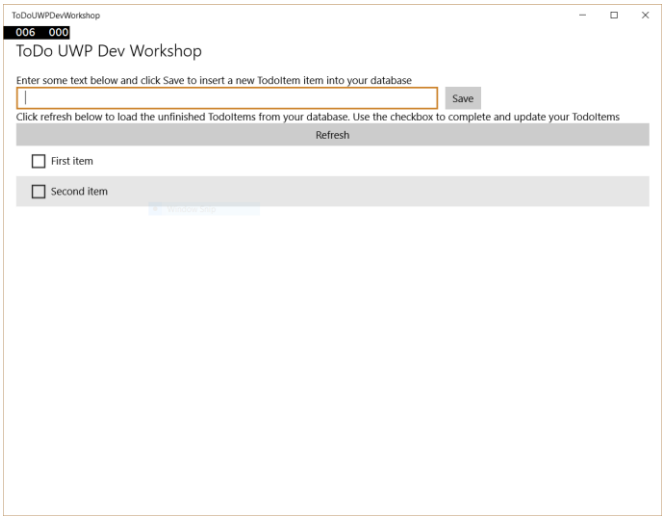


**Рисунок 7**

*Приложение отображает два 'Seed' элемента из службы бэкенда.*

8. Добавьте новые элементы к тем, которые уже есть.

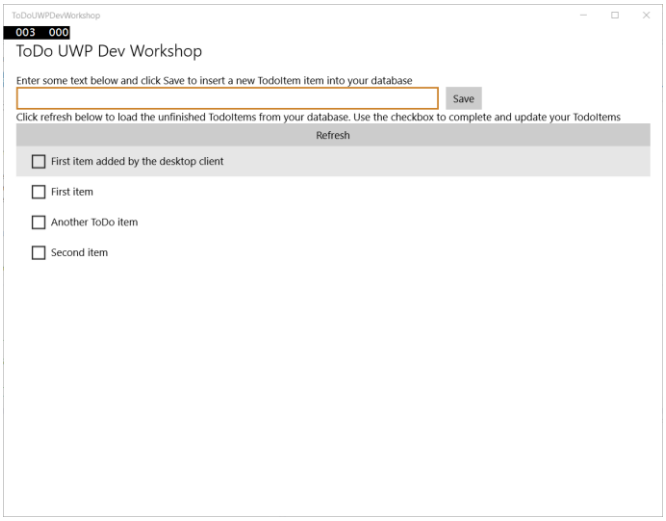


**Рисунок 8**

*Добавление новых элементов.*

9. Остановите отладку.

10. Если на вашем ПК установлен мобильный эмулятор Windows 10, выберите **Mobile Emulator** в качестве типа платформы для отладки. Если у вас подключено реальное мобильное устройство и включен режим разработчика, выберите Device. Если ни одна из этих опций недоступна, вам придётся пропустить остальную часть этого упражнения.

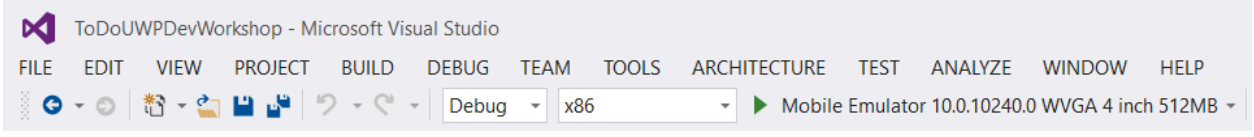


**Рисунок 9**

*Установите Mobile Emulator или Device в Debug Target, если такие опции доступны.*

11. Запустите отладку. Приложение запустится на вашем мобильном устройстве или в эмуляторе и отобразит те же элементы, которые вы видели в настольной версии к службе приложения-клиента и которые хранятся в базе данных мобильного бэкенда (в данный момент запущенного локально на вашем ПК).

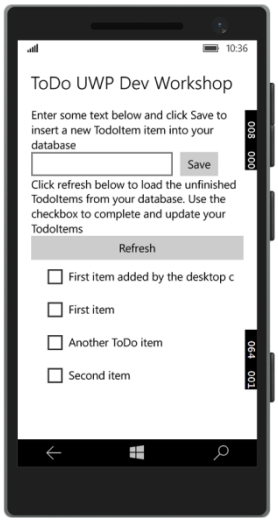


**Рисунок 10**

*Приложение на мобильном устройстве с Windows 10 использует тот же бэкенд.*

12. Отключите отладку обоих проектов: клиента и сервера.

13. Вы завершили это упражнение и запустили мобильное приложение на двух различных устройствах, которые использовали данные из одного бэкенда. Вы реализовали онлайн сценарий работы с приложением.

# Упражнение 2: Реализация аутентификации и подключение приложения к облачной службе

В этом упражнении, вы научитесь добавлять аутентификацию в UWP приложение, использующее мобильный бэкенд Azure. После успешной аутентификации и авторизации, вы увидите значение идентификатора пользователя. Вам предстоит изменить UWP приложение и подключить его к мобильной службе Azure App Service, выполняющейся в Microsoft Azure, вместо той, которая запущена локально. Мобильная служба в Azure заранее настроена и поддерживает аутентификацию через Azure Active Directory.

**Задание 1 – Добавление аутентификации в приложение**

Модифицируем UWP приложение, чтобы аутентифицировать пользователей до того, как приложение будет запрашивать данные из мобильной службы.

1. Откройте **Visual Studio 2015**. В меню **File** нажмите **Open Project/Solution**. Переместитесь в папку, в которой вы сохранили код для этой лабораторной работы, и откройте **Exercise 2\Begin-Client\ToDoUWPDevWorkshop.sln**.

2. Для Solution Configuration выберите Debug, а для Solution Platform выберите x86. Выберите Local Machine из списка Debug Target справа от кнопки Start Debugging.

3. Нажмите Build Solution, чтобы подключить необходимые NuGet пакеты.

4. Откройте файл **App.xaml.cs**. В этом упражнении код для создания экземпляра **MobileServiceClient** изменён - первые строки не закомментированы и вы используете их, чтобы подключиться к облачной службе. Следующий код подключает приложение к локальной версии облачного сервиса, которую вы использовали в предыдущем упражнении – сейчас эти строки закомментированы.

```
C#
    sealed partial class App : Application
    {

        // Uncomment this code for configuring the MobileServiceClient to
        // communicate with the Azure Mobile Service and
        // Azure Gateway using the application key. You're all set to start
        // working with your Mobile Service!
        public static MobileServiceClient MobileService =
            new MobileServiceClient(
            "https://uwpdevhols.azurewebsites.net",

"https://default43cd3457d9384f11a222c50e05bf5fae.azurewebsites.net",
            ""
```

```
        );

        // Use this code for configuring the MobileServiceClient to
        // communicate with your local
        // test project for debugging purposes.
        //public static MobileServiceClient MobileService =
        //    new MobileServiceClient(
        //    "http://localhost:59989"
        //);
    ...
```

5. В браузере пройдите по ссылке для мобильной службы Azure App Service: `https://uwpdevhols.azurewebsites.net` Вы увидите стандартную информационную страницу мобильной службы Azure.

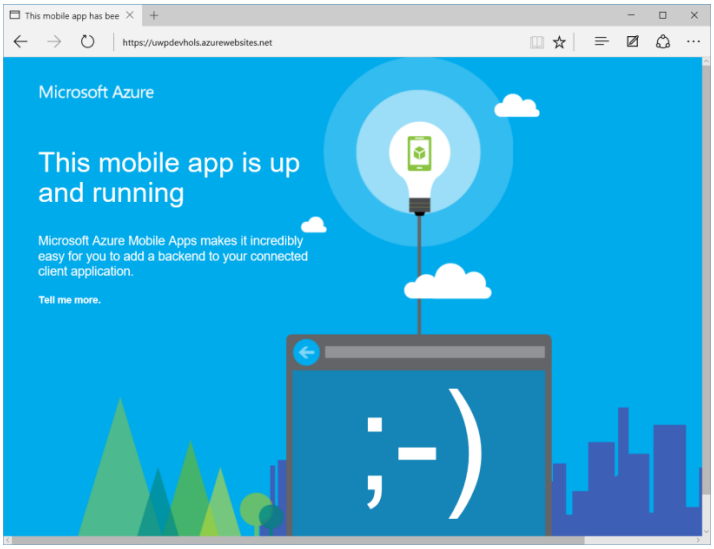


**Рисунок 11**

*Мобильная служба, запущенная в Microsoft Azure.*

6. Этот облачный сервис немного отличается от того, который был запущен локально в предыдущем упражнении. Он настроен так, чтобы доступ предоставлялся только аутентифицированным пользователям. Чтобы продемонстрировать это, запустите клиент **TodoUWPDevWorkshop**. Приложение не сможет загрузить элементы из службы, потому что пользователь приложения не был аутентифицирован через настроенного провайдера.

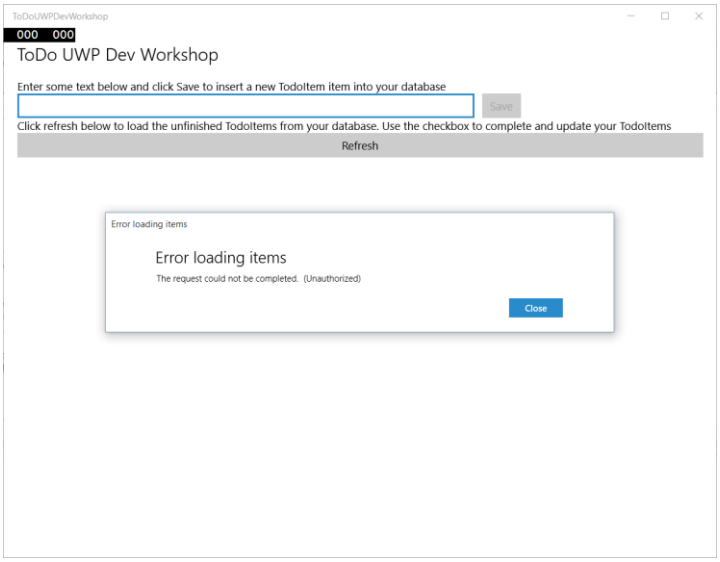


**Рисунок 12**

*Мобильная служба Azure требует пройти аутентификацию для доступа в приложение.*

> **Примечание**: Мобильная служба Azure App Service поддерживает аутентификацию через:
> - Azure Active Directory
> - Facebook
> - Google
> - Microsoft
> - Twitter
>
> Служба, к которой вы подключаетесь, настроена на поддержку аутентификации через Azure Active Directory.
>
> Подробные инструкции о том, как настроить один из таких провайдеров в мобильной службе Azure App Service, можно посмотреть в документации, в разделе **Authenticate Users**: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-users-preview/

7. Теперь вы можете добавить аутентификацию в приложение. Откройте файл **MainPage.xaml.cs**. Определите переменную для хранения зарегистрированного пользователя и способ аутентификации.

**C#**

```
// Define a member variable for storing the signed-in user.
private MobileServiceUser user;

// Define a method that performs the authentication process
// using an Azure Active Directory sign-in.
private async System.Threading.Tasks.Task AuthenticateAsync()
{
    while (user == null)
    {
        string message;
```

```
        // This sample uses the Azure Active Directory provider.
        var provider = "AAD";

        try
        {
            // Sign-in using AAD authentication.
            user = await App.MobileService.LoginAsync(provider);
            message =
                string.Format("You are now signed in - {0}", user.UserId);
        }
        catch (InvalidOperationException)
        {
            message = "You must log in. Login Required";
        }

        var dialog = new MessageDialog(message);
        dialog.Commands.Add(new UICommand("OK"));
        await dialog.ShowAsync();
    }
}
```

8. Закомментируйте или удалите вызов метода **RefreshTodoItems** в существующем переопределении метода **OnNavigatedTo**. Это предотвратит загрузку данных до аутентификации пользователя. Затем добавьте кнопку Sign in, которая будет запускать процесс аутентификации.

9. Добавьте следующий метод в класс **MainPage**:

**C#**

```
private async void ButtonLogin_Click(object sender, RoutedEventArgs e)
{
    // Login the user and then load data from the mobile app.
    await AuthenticateAsync();

    // Hide the login button and load items from the mobile app.
    this.ButtonLogin.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    await RefreshTodoItems();
}
```

10. Откройте файл **MainPage.xaml** и добавьте **Button** в элемент **StackPanel** в строчке Row 0 таблицы Grid, перед элементом **TextBlock**:

**XAML**

```
<StackPanel Grid.Row="0" Grid.ColumnSpan="2" >

    <Button Name="ButtonLogin" Click="ButtonLogin_Click"
        Visibility="Visible">Sign in</Button>
    <TextBlock Style="{StaticResource BodyTextBlockStyle}" Text="Enter some
text below and click Save to insert a new TodoItem item into your database"
TextWrapping="Wrap"/>
</StackPanel>
```

11. Прежде чем запустить приложение, вам необходимо получить имя пользователя и пароль, чтобы использовать их во время аутентификации через Azure Active Directory. Мы заранее подготовили 500 пользователей AAD специально для этих лабораторных работ. Чтобы узнать, какое имя пользователя и пароль вы можете использовать, необходимо зайти на сайт: http://uwpholsusers.azurewebsites.net/

    Вы увидите страницу, на которой будут указаны имя пользователя и пароль:

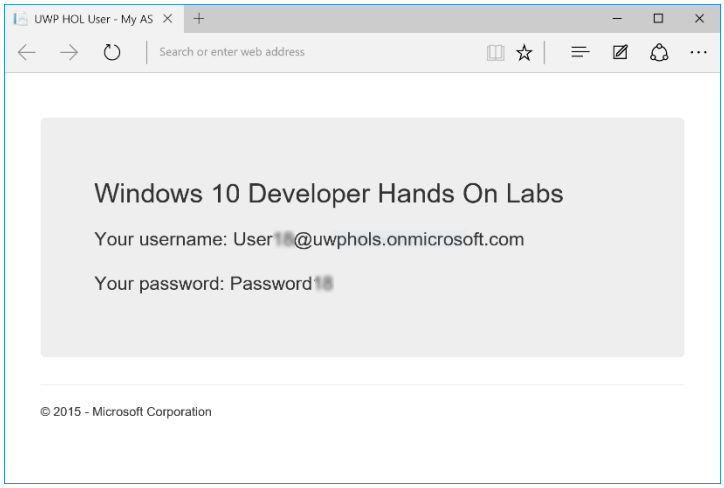


**Рисунок 13**

*Данные учетной записи в AAD.*

> **Примечание**: каждый запрос к этой странице возвращает разные имя пользователя и пароль из 500 настроенных для этой лабораторной работы. Вы должны оказаться единственным, кто использует полученные вами данные во время лабораторной работы, однако это условие не гарантируется. Поскольку идентификатор пользователя используется для разделения данных каждого пользователя в базе данных SQL Azure, то если кто-то еще будет использовать данные вашей учетной записи, вы увидите это в качестве элементов ToDo, созданных без вашего участия. Естественно, для реальных приложений, данные учетной записи не распространяются подобным образом.

12. Нажмите **F5**, чтобы запустить приложение, нажмите Sign In, и зайдите в приложение, используя полученные логин и пароль.

13. Когда вы войдете, приложение запуститься без ошибок, и вы должны будете получить доступ к вашему мобильному приложению и к возможностям добавлять ToDo элементы.

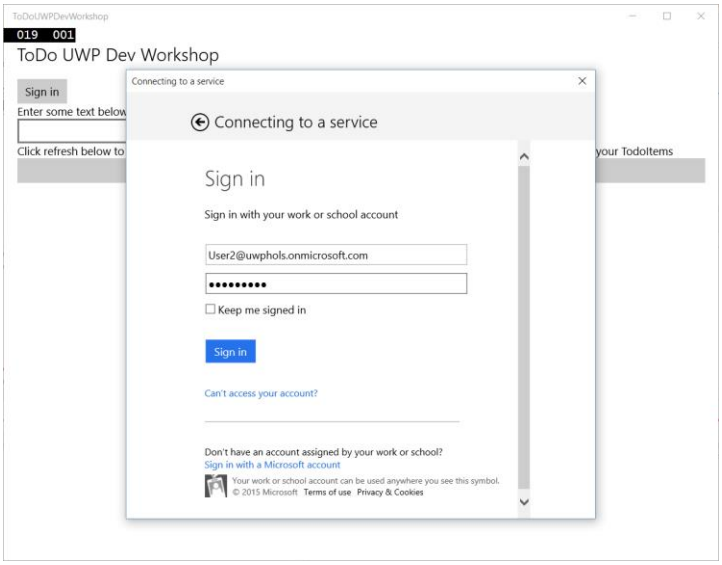


**Рисунок 4**

*Аутентификация через Azure Active Directory.*

> **Примечание:** Мобильная служба Azure App Service, к которой вы подключены, используется и другими пользователями сервиса. Все данные для пользователей хранятся в одной базе данных SQL Azure, и все пользователи имеют доступ к REST сервису в Azure.
>
> Для поддержки многопользовательского режима, необходимо было произвести некоторые изменения в коде службы запущенной в облаке, что отличает ее от службы, запущенной локально в Упражнении 1.
>
> Вы можете изучить код, выполняющийся в службе, к которой вы подключены, открыв **Exercise 2\Cloud\UwpDevHols.sln**. Ниже перечислены изменения:
>
> - В файле **DataObjects\TodoItem.cs** добавлено дополнительное поле **UserId** для хранения идентификатора пользователя для каждого элемента ToDo
> - В файле **Controllers\TodoItemController.cs** в методе **GetAllTodoItems** добавлен код, который получает идентификатор пользователя и использует его, чтобы выполнить запрос только тех ToDo элементов, которые связаны с пользователем; в методе **PostTodoItem** добавлена логика получения идентификатора пользователя и сохранения его в поле UserId каждого ToDo элемента.
> - Добавлен дополнительный атрибут **[Authorize]** в класс **TodoItemController**, который ограничивает доступ не аутентифицированным пользователям

**Задание 2 – Сохранение токен аутентификатора на клиенте**

В предыдущем задании вы реализовали стандартный вход в приложение, который требует от клиента подключаться и к провайдеру идентификатора, и к мобильному сервису каждый раз, когда запускается приложение. Этот метод не всегда эффективен. Лучше реализовать подход, когда производится кэширование токен аутентификатора, возвращенного службой, и постараться использовать его прежде чем вход через провайдера.

1. Продолжите работу с **ToDoUWPDevWorkshop.sln** из предыдущего задания.

2. В файл **MainPage.xaml.cs** подключите следующие билиотеки:

**C#**

```
using System.Linq;
using Windows.Security.Credentials;
```

3. Замените метод **AuthenticateAsync** следующим кодом:

**C#**

```
// Define a method that performs the authentication process
// using an Azure Active Directory sign-in.
private async System.Threading.Tasks.Task AuthenticateAsync()
{
    string message = string.Empty;
    // This sample uses the Azure Active Directory provider.
    var provider = "AAD";

    // Use the PasswordVault to securely store and access credentials.
    PasswordVault vault = new PasswordVault();
    PasswordCredential credential = null;

    while (credential == null)
    {
        try
        {
            // Try to get an existing credential from the vault.
            credential =
vault.FindAllByResource(provider).FirstOrDefault();
        }
        catch (Exception)
        {
            // When no matching resource an error occurs, which we ignore.
        }

        if (credential != null)
        {
            // Create a user from the stored credentials.
            user = new MobileServiceUser(credential.UserName);
            credential.RetrievePassword();
            user.MobileServiceAuthenticationТокен= credential.Password;

            // Set the user from the stored credentials.
            App.MobileService.CurrentUser = user;
```

```
            try
            {
                // Try to return an item now to determine if the
                // cached credential has expired.
                await App.MobileService.GetTable<TodoItem>().Take(1)
                      .ToListAsync();
            }
            catch (MobileServiceInvalidOperationException ex)
            {
                if (ex.Response.StatusCode ==
                        System.Net.HttpStatusCode.Unauthorized)
                {
                    // Remove the credential with the expired token.
                    vault.Remove(credential);
                    credential = null;
                    continue;
                }
            }
        }
        else
        {
            try
            {
                // Login with the identity provider.
                user = await App.MobileService
                    .LoginAsync(provider);

                // Create and store the user credentials.
                credential = new PasswordCredential(provider,
                    user.UserId, user.MobileServiceAuthenticationToken);
                vault.Add(credential);
            }
        catch (InvalidOperationException)
        {
            message = "You must log in. Login Required";
        }

        var dialog = new MessageDialog(message);
        dialog.Commands.Add(new UICommand("OK"));
        await dialog.ShowAsync();
    }
}
```

В этой версии метода **AuthenticateAsync** для доступа к мобильной службе приложение пытается использовать данные, которые хранятся в PasswordVault. Отправляется простой запрос, чтобы убедиться, что срок действие сохраненного токенне закончился. Когда возвращается код 401, осуществляется вход в систему через провайдера. Также обычный вход в систему осуществляется, когда отсутствуют сохраненные идентификационные данные.

**Примечание:** Срок действия токен может истечь и после аутентификации, когда приложение уже используется. Чтобы отслеживать такие ситуации,воспользуйтесь

информацией из материалов Caching and handling expired tokens in Azure Mobile Services managed SDK:
http://blogs.msdn.com/b/carlosfigueira/archive/2014/03/13/caching-and-handling-expired-tokens-in-azure-mobile-services-managed-sdk.aspx

4. Перезапустите приложение дважды.
   Во время первого запуска осуществите вход в систему через форму провайдера. Во время второго запуска будут использоваться сохраненные идентификационные данные и произойдет вход в систему.

5. **[Дополнительно]** Одной из возможностей PasswordVault является следующее - идентификационные данные, которые вы храните, попадают через облако во все ваши устройства, где установлено это приложение. Если у вас есть второе устройство с операционной системой Windows 10 или Windows 10 Mobile, например, телефон, планшет или ПК, и это устройство связано с тем же аккаунтом Microsoft, что и ваш ПК, вы можете протестировать эту возможность, установив то же UWP приложение на это устройство.
   Если вы подождете, пока завершится перемещение идентификационных данных, то когда вы войдете в приложение на одном из устройств, оно обнаружит хранилище идентификационных данных в **PasswordVault** и вам не придется заново заходить в систему, если только срок токена не истек.

# Упражнение 3: Реализация офлайн хранилища и синхронизации данных в приложении

В этом упражнении, вы реализуете поддержку офлайн режима в UWP приложении при помощи мобильно бэкенда в Azure. Синхронизация в офлайн сценарии позволяет использовать функции просмотра, добавления или изменения данных в приложении, даже при отсутствии соединения с сетью. Изменения сохраняются в локальной базе данных. Как только устройство вновь подучает доступ к сети, изменения синхронизируются с удаленным хранилищем на бэкенде.

В этом упражнении вы модифицируете приложение Windows, добавив поддержку офлайн возможностей с мобильной службой Azure.

**Задание 1 – Реализация офлайн сценария работы с мобильным приложением**

Офлайн сценарий с мобильной службой Azure позволяет сохранять данные и взаимодействовать с локальной базой данных. Для того чтобы воспользоваться этими возможностями в вашем приложении, необходимо инициализировать метод **MobileServiceClient.SyncContext** для локального хранилища. Затем связать таблицу через интерфейс **IMobileServiceSyncTable.** В этой лабораторной работе мы будем использовать в качестве локального хранилища SQLite.

Откройте проект ToDoUWPDevWorkshop, которой вы использовали в предыдущем упражнении.

1. Переместитесь в папку, где вы сохранили **ToDoUWPDevWorkshop** приложение-клиент из Упражнения 2. Откройте **ToDoUWPDevWorkshop.sln** в Visual Studio 2015.

2. Установите SQLite для UWP приложения.

    a. В Visual Studio, в меню **Tools** нажмите на **Extensions and Updates**

    b. В левой панели Extensions and Updates, нажмите **Online**

    c. В поле поиска, введите **SQLite**

    d. Когда отобразятся результаты поиска, прокручивайте список, пока не увидите **SQLite for Universal App Platform**. Если этот SDK еще не установлен, выберите этот элемент и нажмите на кнопку **Download**.

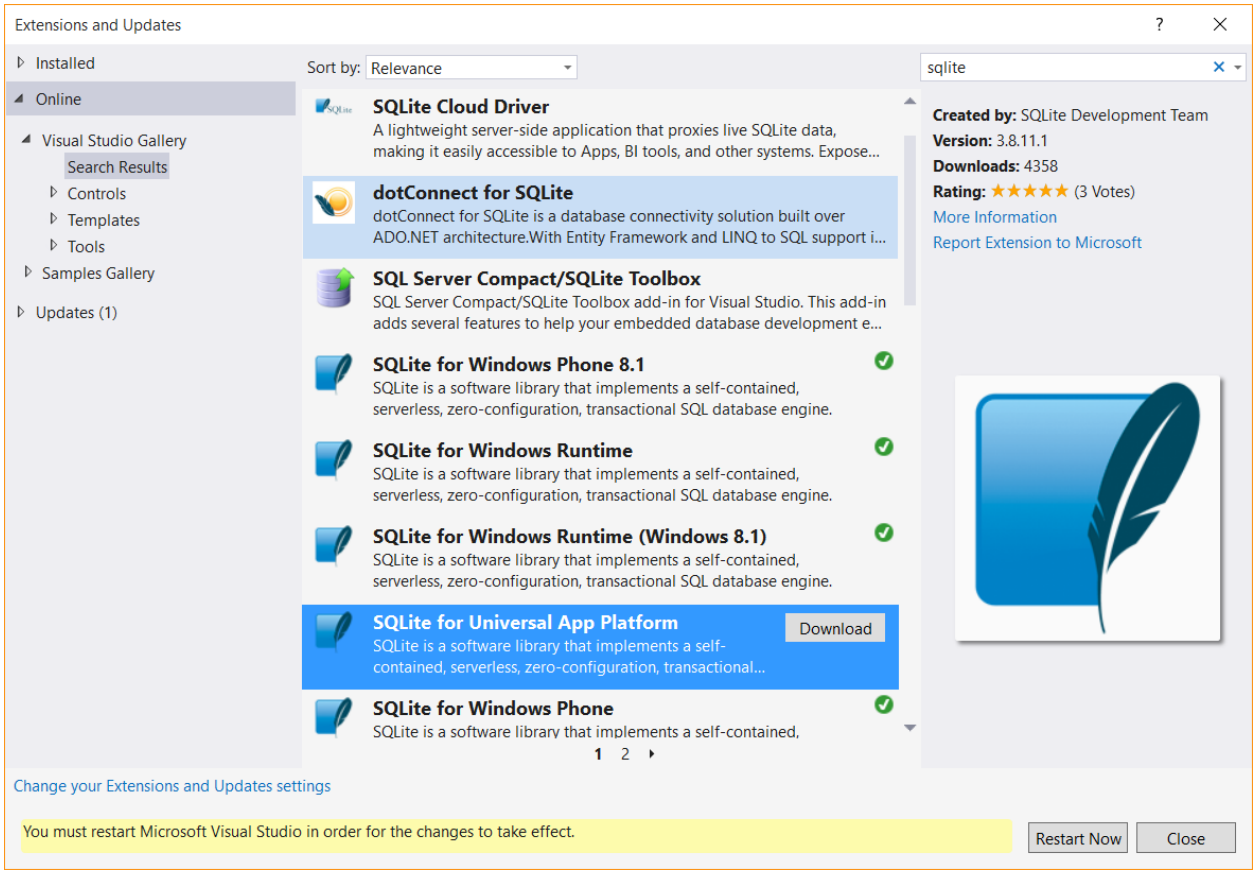


**Рисунок 15**

*Скачайте и установите SQLite для Universal App Platform SDK.*

e. Когда на экране появится UAC окно, нажмите **OK.**

f. В окне VSIX Installer нажмите **Install**. После того, как расширение установится, нажмите **Close**.

g. Нажмите на кнопку **Restart Now** в окне Extensions and Updates и подождите пока перезапустится Visual Studio 2015.

3. Добавьте ссылку на библиотеку SQLite в ваш проект.

a. В Solution Explorer нажмите правой кнопкой на References и затем выберете Add Reference, чтобы запустить Reference Manager.

b. В категории Universal Windows выберите пункт Extensions в панели слева.

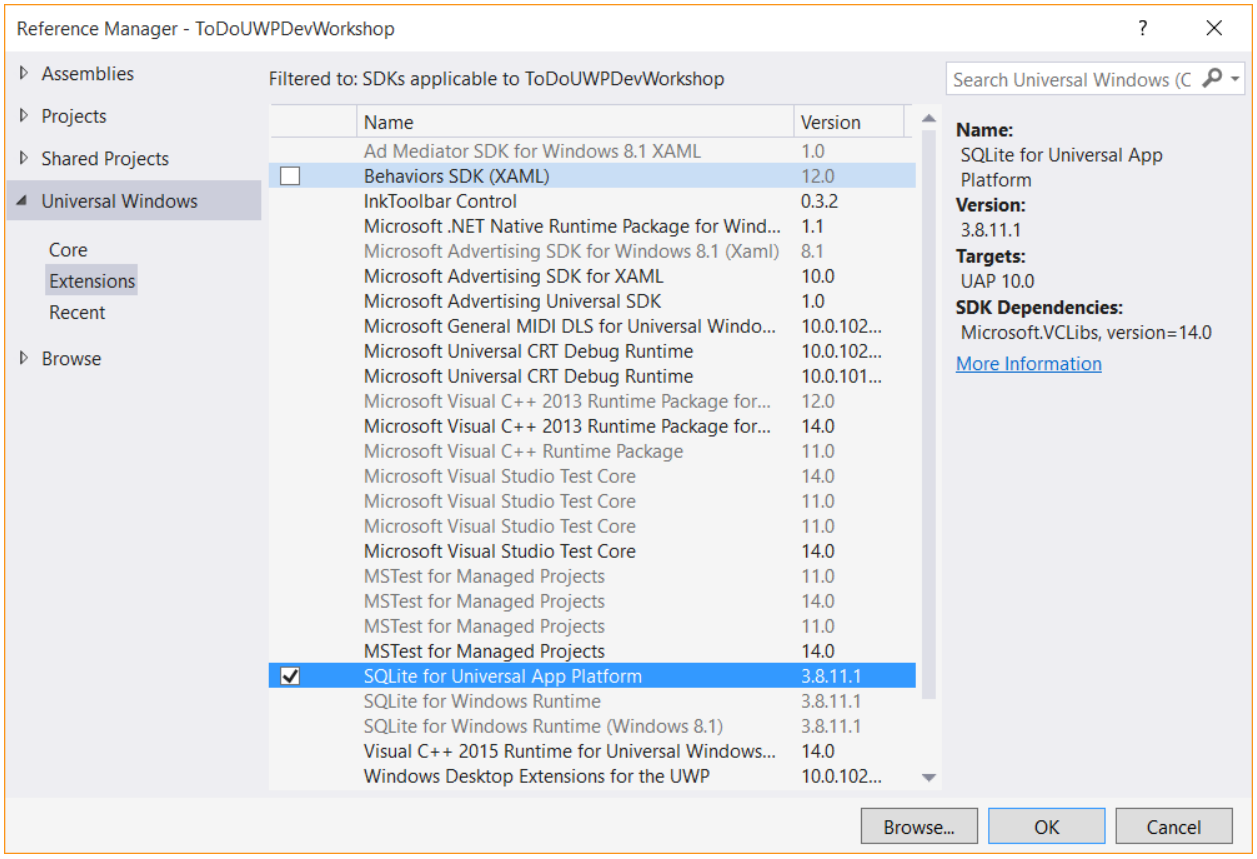


**Рисунок 14**

*Добавление ссылки на библиотеку SQLite к проекту.*

c. Выберите **SQLite для Universal App Platform** и затем **нажмите OK**.

4. Установите NuGet пакет WindowsAzure.MobileServices.SQLiteStore.

   a. В Solution Explorer правой кнопкой нажмите на проект и выберите **Manage Nuget Packages**, чтобы запустить NuGet Package Manager.

   b. На вкладке Online выберите опцию Include Prerelease. Выполните поиск **SQLiteStore**, чтобы затем выбрать **WindowsAzure.MobileServices.SQLiteStore**.

   c. Затем нажмите Install, чтобы добавить ссылку на NuGet пакет в проект.

   d. Нажмите **I Accept** в окне License Acceptance.

5. В Solution Explorer откройте **файл MainPage.cs**. Раскомментируйте следующие строки:

**C#**

```
using Microsoft.WindowsAzure.MobileServices.SQLiteStore;  // offline sync
using Microsoft.WindowsAzure.MobileServices.Sync;          // offline sync
```

6. В файле MainPage.cs закомментируйте код, который инициализирует `todoTable` через `IMobileServiceTable`. И раскомментируйте код, который инициализирует `todoTable` через `IMobileServiceSyncTable`:

**C#**

```
//private IMobileServiceTable<TodoItem> todoTable =
App.MobileService.GetTable<TodoItem>();
private IMobileServiceSyncTable<TodoItem> todoTable =
                App.MobileService.GetSyncTable<TodoItem>(); // offline sync
```

7. В файле **MainPage.cs** в разделе, отмеченном **Offline sync,** раскомментируйте методы **`InitLocalStoreAsync`** и **`SyncAsync`**. InitLocalStoreAsync инициализирует синхронизацию данных на клиенте и в SQLite. В Visual Studio вы можете выделить все закомментированные строки и использовать Ctrl+K+U, чтобы раскомментировать их.

   Обратите внимание, что в **`SyncAsync`** push операции выполняет метод `MobileServiceClient.SyncContext` вместо `IMobileServicesSyncTable`. Это необходимо для того, чтобы отслеживать изменения, произведенные на клиенте для всех таблиц. Это подходит в случаях, где таблицы связаны между собой. Дополнительная информация доступна в документации, в разделе Offline Data Sync в Azure Mobile Apps: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/

**C#**

```
private async Task InitLocalStoreAsync()
{
    if (!App.MobileService.SyncContext.IsInitialized)
    {
        var store = new MobileServiceSQLiteStore("localstore.db");
        store.DefineTable<TodoItem>();
        await App.MobileService.SyncContext.InitializeAsync(store);
    }

    await SyncAsync();
}

private async Task SyncAsync()
{
    await App.MobileService.SyncContext.PushAsync();
    await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
}
```

8. В обработчике события `OnNavigatedTo` раскомментируйте вызов `InitLocalStoreAsync`:

**C#**

```
protected override async void OnNavigatedTo(NavigationEventArgs e)
{
    await InitLocalStoreAsync(); // offline sync
    // await RefreshTodoItems();
}
```

9. Раскомментируйте 3 вызова SyncAsync в методах InsertTodoItem, UpdateCheckedTodoItem и ButtonRefresh_Click

**C#**

```
private async Task InsertTodoItem(TodoItem todoItem)
{
    await todoTable.InsertAsync(todoItem);
    items.Add(todoItem);

    await SyncAsync(); // offline sync
}

...

private async Task UpdateCheckedTodoItem(TodoItem item)
{
    await todoTable.UpdateAsync(item);
    items.Remove(item);
    ListItems.Focus(Windows.UI.Xaml.FocusState.Unfocused);

    await SyncAsync(); // offline sync
}

private async void ButtonRefresh_Click(object sender, RoutedEventArgs e)
{
    ButtonRefresh.IsEnabled = false;

    await SyncAsync(); // offline sync
    await RefreshTodoItems();

    ButtonRefresh.IsEnabled = true;
}
```

10. Измените код в SyncAsync, чтобы добавить дополнительную обработку исключений. В режиме офлайн MobileServicePushFailedException будет перехвачен с PushResult.Status == CancelledByNetworkError.

**C#**

```
private async Task SyncAsync()
{
    String errorString = null;

    try
    {
        await App.MobileService.SyncContext.PushAsync();
        // first param is query ID, used for incremental sync
        await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
    }
```

```
    catch (MobileServicePushFailedException ex)
    {
        errorString = "Push failed because of sync errors. " +
                      "You may be offine.\nMessage: " +
                      ex.Message + "\nPushResult.Status: " +
                      ex.PushResult.Status.ToString();
    }
    catch (Exception ex)
    {
        errorString = "Pull failed: " + ex.Message +
          "\n\nIf you are still in an offline scenario, " +
          "you can try your Pull again when connected with " +
          "your Mobile Service.";
    }

    if (errorString != null)
    {
        MessageDialog d = new MessageDialog(errorString);
        await d.ShowAsync();
    }
}
```

**Примечание:** MobileServicePushFailedException может произойти в обоих случаях, с pull и push методами. В операции pull это может произойти, потому что pull метод внутри себя вызывает push, для того чтобы убедиться, что таблицы согласованы между собой.

11. В Visual Studio нажмите F5, чтобы выполнить сборку и запустить приложение. Приложение-клиент будет вести себя так же, как и до добавления офлайн возможностей. Тем не менее, эти изменения позволят создать локальную базу данных, которая будет использоваться в режиме офлайн, и заполнить ее данными. Мы протестируем режим офлайн в следующем задании.

**Задание 2 – Проверка работы приложения в офлайн режиме**

В этом упражнении вы запустите приложение-клиент и проверите его работу в режиме офлайн. Когда вы добавите данные, обработчик исключений сообщит вам, что приложение работает в офлайн режиме с **PushResult.Status == CancelledByNetworkError**. Добавленные элементы будут храниться в локальном хранилище, но не будут синхронизированы с мобильным бэкендом, пока вы не восстановите соединение.

1. Добавьте ToDo элементы, чтобы убедиться, что офлайн режим приложения работает.

    a. **Вариант 1:** Для того, чтобы протестировать офлайн режим приложения для вашего ПК, нажмите в трее на иконку Notifications, для того чтобы открыть Notification Center. В панели быстрых задач включите Flight Mode.

    b. **Вариант 2:** Поменяйте тип устройства для отладки на эмулятор Windows 10 Mobile. Установите на эмулятор приложение. Когда эмулятор запустится, нажмите **кнопку >>** на панели команд, чтобы увидеть дополнительные возможности эмулятора.

В **Network** выберите **Enable Network simulation** и установите для **Network speed** значение **No network**.

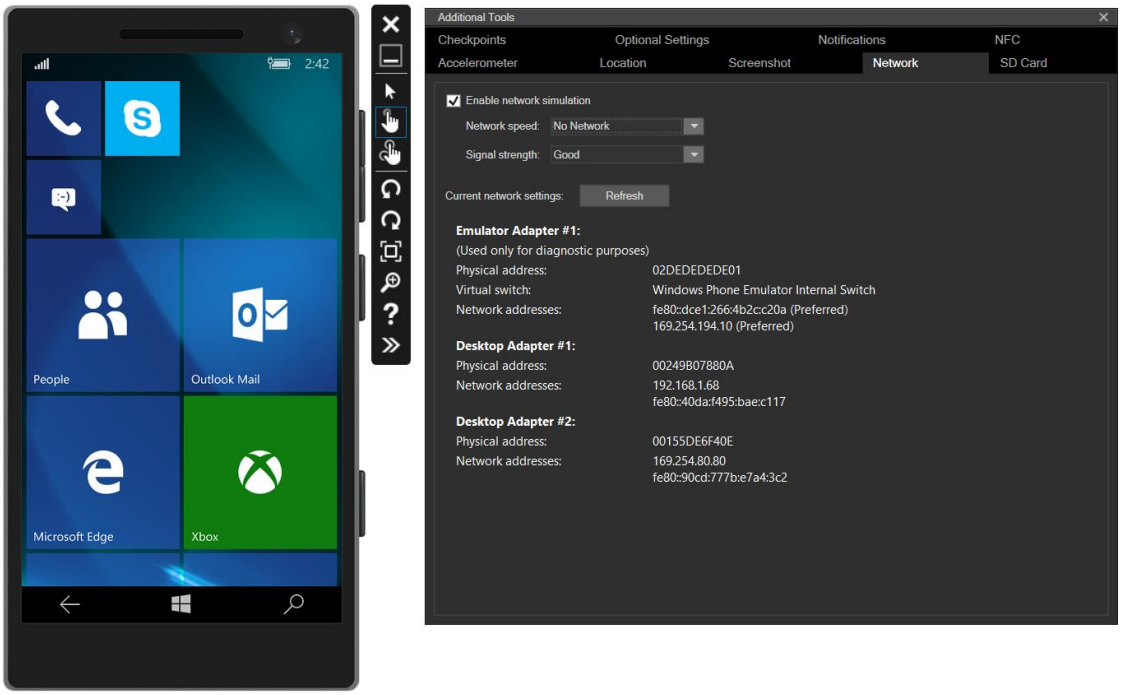


**Рисунок 17**

*Настройка эмулятора Windows 10 Mobile для работы в режиме офлайн.*

2. Нажмите F5 для создания и запуска приложения. Обратите внимание, что при запуске приложения, синхронизация не произошла.

3. Добавьте новые ToDo элементы и нажмите Save.Для каждого элемента Push метод вызовет исключение PushResult.Status=CancelledByNetworkError. Новые ToDo элементы существуют только в локальном хранилище до того момента, пока они не будут отправлены в бэкенд мобильного приложения.
   Вы можете обработать возникающие исключения **PushResult.Status=CancelledByNetworkError** таким образом, чтобы приложение –клиент вело себя так, как если бы оно было соединено с мобильной службой.

4. Закройте приложение и запустите его снова, чтобы убедиться, что элементы, которые вы создали, до сих пор отображаются в приложении и хранятся в локальном хранилище.

### Задание 3 – Повторное подключение приложения к облачной службе

В этом упражнении вы снова подключите приложение к мобильному бэкенду. Приложение перестроится из работы в режиме офлайн в режим онлайн работы. Когда вы первый раз запустите приложение, обработчик событий **OnNavigatedTo** вызовет **InitLocalStoreAsync.** Это приведет к тому, что вызовется метод **SyncAsync** для синхронизации локального хранилища с удаленным. Таким образом, приложение попытается синхронизировать данные после запуска.

1. Переведите приложение в режим онлайн.

    a. **Вариант 1:** Если вы используете десктоп версию приложения, то отключите Flight Mode и верните ваше устройство в режим онлайн.

    b. **Вариант 2:** Если вы тестируйте офлайн режим при помощи мобильного эмулятора, нажмите **>>** на панели команд, в разделе **Network** отключите **Enable network simulation**.

2. Нажмите **F5** для сборки и запуска приложения. Приложение синхронизирует локальные изменения с мобильным бэкендом в Azure с помощью операций pull и push в тот момент, когда вызывается обработчик события **OnNavigatedTo**.

3. **[Дополнительно]** Если у вас есть другое устройство с установленным приложением, запустите приложение на этом устройстве с теми же данными учетной записи, что и прежде. Проверьте, что ToDo элементы, созданные в офлайн режиме синхронизированы с мобильным бэкендом в Azure и доступны на втором устройстве.

4. В приложении поставьте галочку напротив некоторых ToDo элементов, чтобы отметить их как завершенные и сохранить эти данные в локальном хранилище.

    UpdateCheckedTodoItem вызовет SyncAsync для синхронизации информации о завершенных ToDo элементах с бэкендом. SyncAsync вызовет pull и push операции. Имейте ввиду, **что при выполнении операции pull для таблицы, в которой на клиенте произошли изменения, операция push будет производиться автоматически.** Для дополнительной информации обратитесь к документации на портале в разделе Offline Data Sync в Azure Mobile Apps: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/

# Заключение

В этой лабораторной работе вы изучили, как подключить универсальное приложение Windows (UWP) к мобильному бэкенду в Azure App Service и запустили службы мобильного бэкэнда локально на своём ПК. Вы так же добавили в клиент мобильного приложения аутентификацию через Azure Active Directory и подключили приложение к мобильному бэкэнду, размещённому на сервере в облаке Microsoft Azure. Наконец, вы реализовали сценарий работы приложения в режиме офлайн с последующей синхронизацией локального и удаленного хранилищ.