

Практическая­ лаборатория

Курс: Подсоединение клиента универсальной платформы Windows к мобильным приложениям Azure Mobile Apps

Сентябрь, 2015 г.

Содержание

**No table of contents entries found.**

Краткое описание

* 1. Azure App Service это платформа, управляемая PaaS, предлагающая профессиональным разработчикам, которые работают над широким набором возможностей для веба, мобильные и интеграционные возможности. Мобильные приложения Azure App Service предлагают крупномасштабную, глобально доступную платформу разработки мобильных приложений для Разработчиков Предприятия и Интеграторов Систем, которая предлагает широкий набор возможностей мобильным разработчикам.
  2. С мобильными приложениями вы можете:
* Создавать аппаратные и кроссплатформенные приложения, вне зависимости от того, создаете ли Вы аппаратную платфору для iOS, Android и Windows или кроссплатформенное приложение Xamarin или Cordova (Phonegap), вы можете воспользоваться Службой Приложений, использующей аппаратные SDK.
* Подключаться к системам вашей компании – с помощью мобильных приложений вы можете добавлять корпоративную регистрацию за несколько минут и подключиться к ресурсам вашей компании локально или через облако.
* Подключайтесь с легкостью к SaaS API - с помощью более 40 коннекторов SaaS API вы легко можете интегрировать своё приложение в SaaS API , которое ваша компания использует сегодня.
* Готовые оффлайн приложения с синхронизацией делают ваши трудовые ресурсы продуктивными, создавая приложения, которые работают оффлайн, и используют мобильные Приложения для синхронизации данных на фоне при наличии соединения с любым из источников данных вашей компании или SaaS API.
* Push-уведомления миллионам за несколько секунд привлекают ваших клиентов моментальными уведомлениями на любом устройстве, персонализированном для его нужд, отправленными в нужное время.

В этой лаборатории вы подключите приложение универсальной платформы Windows к серверному компоненту Службы Приложений Azure.Вы сначала запустите сервисы серверной части локально на своём ПК и исследуете код, чтобы понять основы того, как взаимодействуют клиентское приложение и служба серверных компонентов мобильного приложения. Вы измените клиентское приложение, чтобы отличаться от Azure Active Directory и подключить его к мобильному серверному компоненту, размещённому на сервере в Microsoft Azure. Наконец, вы измените клиентское приложение, чтобы реализовать автономную возможность синхронизации, включая одну из самых мощных возможностей, предложенных мобильной службой Azure App Service.

# Цели

* 1. Эта лаборатория покажет вам как:
  + Запускать приложение Windows 10 UWP, которое предназначено для синхронизации данных с сервисом серверной части мобильного приложения, запускающее службы серверных компонентов локально на вашем ПК.
  + Подключать клиентское приложение к серверному мобильному компоненту, выполняющемуся в Azure.
  + Выполнять аутентификацию в отличие от Azure Active Directory в клиентском приложении.
  + Менять клиентское приложение для подключения независимого режима, локального хранения данных в SQLite и затем их синхронизации с серверным компонентом мобильного, когда соединение восстановлено.

# Системные требования

* 1. Чтобы выполнить эту установку, необходимо обладать следующим:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Дополнительные надстройки

* 1. Если вы захотите совершить дополнительные действия, вам понадобится:
  + Windows 10 Mobile Emulator или телефон, запускающий Windows 10 Mobile

# Настройка

* 1. Вы должны осуществить следующие шаги для подготовки своего компьютера:
  2. Установить Microsoft Windows 10.
  3. Установить Microsoft Visual Studio 2015. Выберите пользовательскую установку и убедитесь, что выбраны универсальные инструменты разработки приложений Windows из списка дополнительных функций.
  4. Дополнительно: Установите Windows 10 Mobile Emulator.

# Упражнения

* 1. Эта Практическая Лаборатория включает следующие упражнения:
  2. Запуск мобильных приложений Azure Mobile Apps
  3. Активировать функцию аутентификации для своего приложения и подключиться к Облаку
  4. Активировать синхронизацию автономных режимов для своего приложения
  5. Расчётное время для завершения этой лаборатории: **От 45 до 60 минут**.

Упражнение 1: Запуск мобильных приложений Azure Mobile Apps

В этом упражнении вы запустите приложение UWP Windows 10, которое подобно приложению ToDo, которое вы можете загрузить из Портала Azure, когда вы создадите серверный компонент мобильного приложения Azure. Вы подключите клиентское приложение к серверному компоненту мобильного на Azure, которое вы запустите локально на своём ПК в среде разработки Azure. Вы подключитесь к действительному серверному компоненту Azure, запущенному в облаке в следующем упражнении.

Задача 1 – Разобраться в основах Azure Портал

Мы начнём с понимания, как создать сервис серверного компонента мобильного приложения Azure на Портале Azure.

1. Вам не нужно создавать сервис серверного компонента мобильного приложения Azure для этой практической лаборатории, так как мы заранее создали его для вас.  
   Если вы хотите научиться, как создавать новый сервис серверного компонента мобильного приложения Azure, вам необходимо следовать указаниям в разделе, **Создание нового сервиса серверного компонента мобильного приложения на Azure** на <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-preview/>.  
     
   Это учебное пособие покажет вас как создать новый сервис серверного компонента мобильного приложения на Azure:

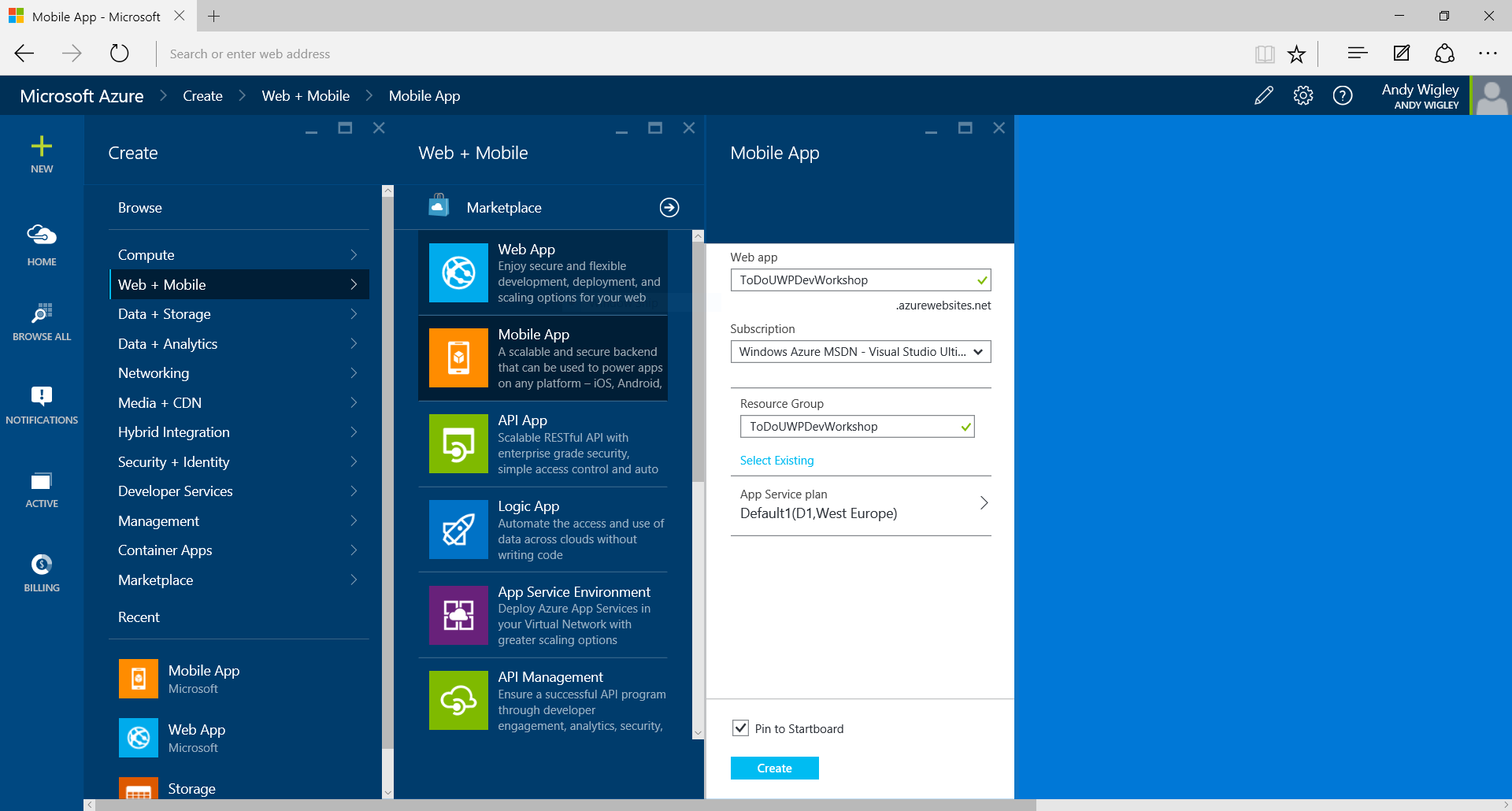


Рисунок 1

Начало создания нового серверного компонента мобильного приложения Azure в Azure Портал.

1. Если вы создали свой собственный серверный компонент, вы увидите лезвие на Портале Azure с информацией о своём серверном компоненте мобильного приложения, подобном этому:

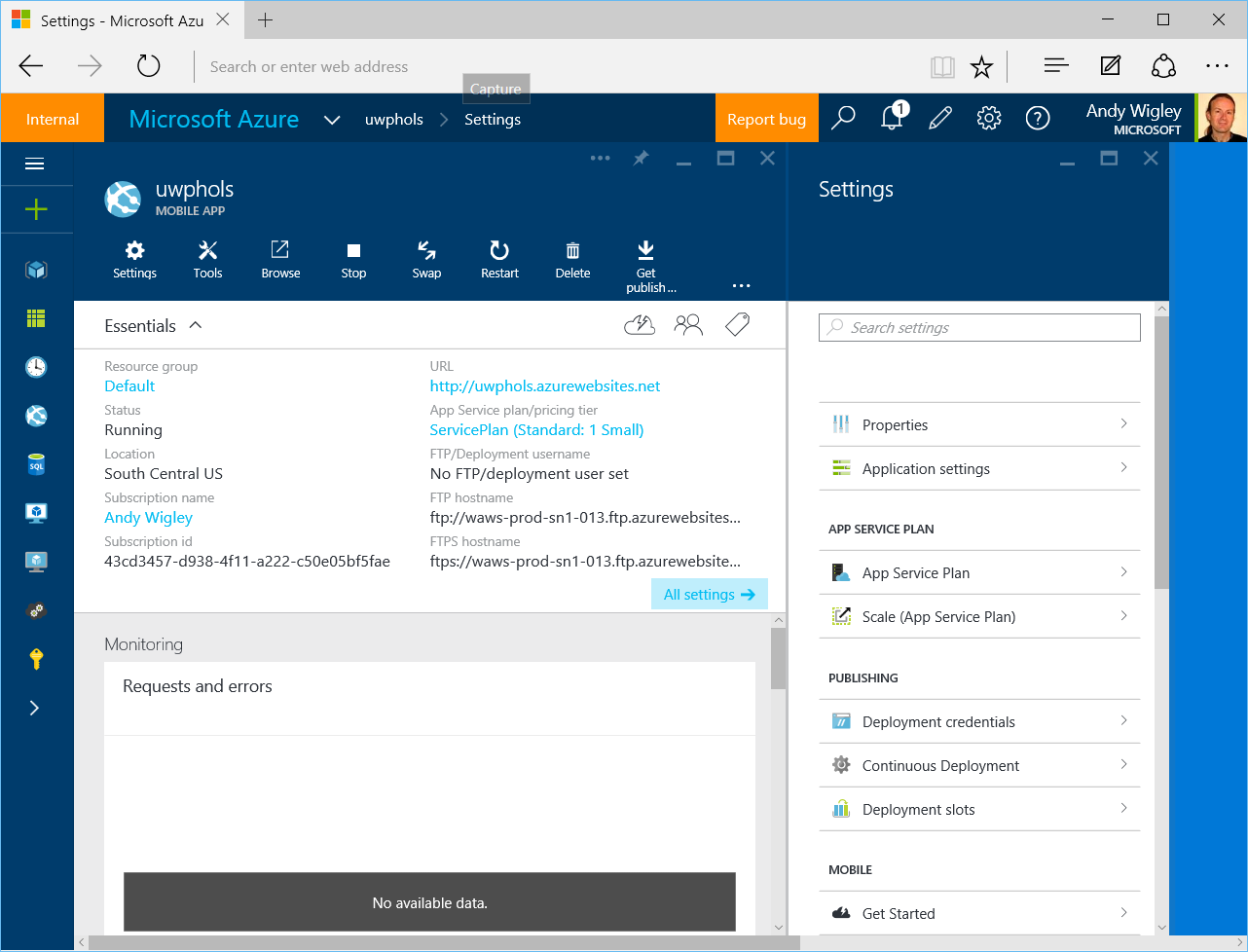


Рисунок 2

Лезвие для серверного компонента мобильного приложения на Azure Portal.

1. В**Настройках** вы найдете раздел Мобильный, содержащий некоторое количество опций, включая **Запустить**. Если вы кликните здесь, то вы откроете **Быстрый старт**, где вы можете получить инструкции по подключению существующего приложения к своему серверному компоненту мобильного приложения на Azure, или вы можете загрузить проекты на сервер, и пример – клиентское мобильное приложение. Вы можете выбрать между некоторым количеством различных клиентов, включая Windows (C#), iOS (Objective-C), iOS (Swift), HTML/JavaScript, Xamarin.Android, Xamarin.iOS или Xamarin.Forms.

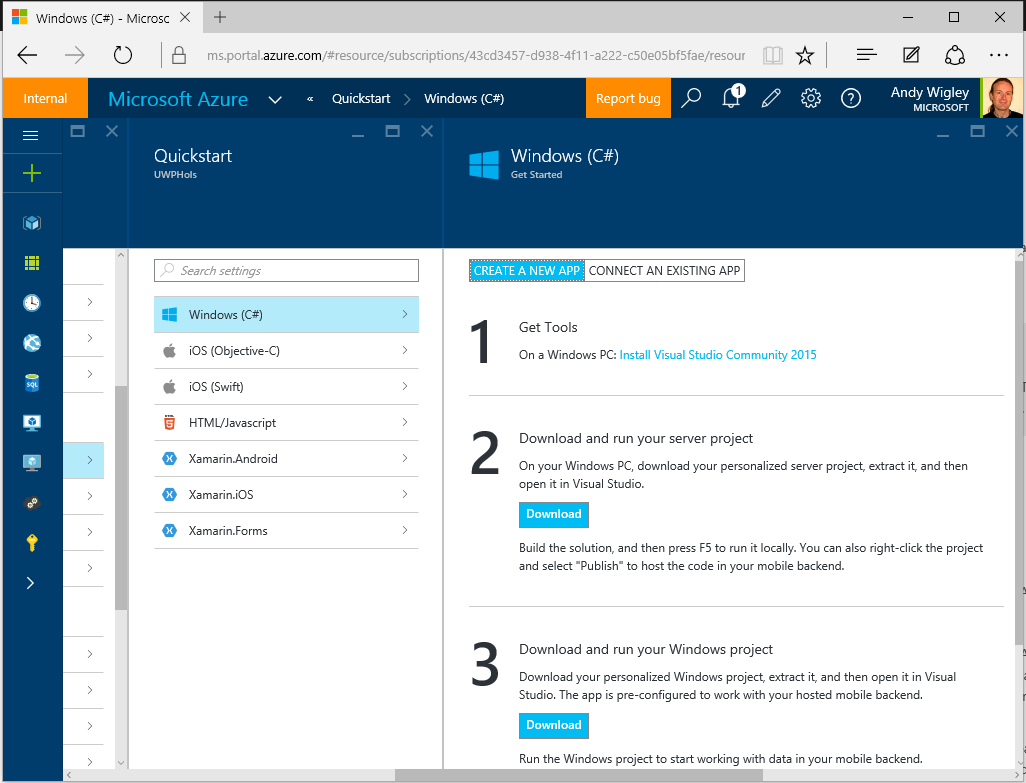


Рисунок 3

Опции мобильного приложения Quickstart в Azure Portal.

1. Во время написания этой инструкции (сентябрь 2015 года) загрузка Quickstart для Windows (C#) является проектом универсального приложения Windows 8.1. В целях этой лаборатории мы создали версию UWP этого клиента.

Задача 2 – Запустить проект сервера на своем ПК

Мы рассмотрим код сервера, который был загружён, когда был создан серверный компонент-приложение мобильного на Azure, который вы будете использовать позже в этой лаборатории. Вы затем запустите её локально на своём ПК.

1. Откройте **Visual Studio 2015**. В меню **Файл** кликните **Начать Проект/Решение**. Перемещайтесь в папку, в которой вы сохранили код для этой лаборатории, и откройте **Упражнение 1\Begin-Cloud\UWPHolsService.sln**.
2. Создайте проект, чтобы восстановить пакеты NuGet.

**Примечание:** Этот проект содержит код C# для серверного компонента .NET для вашего серверного компонента мобильного приложения на Azure. Вместо этого вы можете выбрать свою логику программы серверных компонентов, осуществлённую в NodeJS. См. документацию мобильных приложений на Azure для дополнительной информации.

1. Этот проект содержит код C# для логики программы серверных компонентов. В папке **Объекты данных** откройте **ToDoItem.cs**. Этот класс определяет объекты данных, которые облако сохраняет в базе данных SQL Azure:
   * 1. C#
   1. открытый классTodoItem: Модель данных
   2. {
   3. Текстовый фрагментТекст{ получить установить}
   4. Ключевое слово Завершить { получитьустановить }
   5. }
2. В папке **Контроллеры** откройте **TodoItemController.cs**. Этот класс содержит код, который выполняет операции CRUD, предложенные сервисом облака REST API:
   * 1. C#
   1. открытый классTodoItemController:TodoItemController TableController<TodoItem>
   2. {
   3. Защищённое override void Инициализация(
   4. HttpControllerContext controllerContext)
   5. {
   6. Основа.Инициализация (controllerContext);
   7. UWPHolsContextконтекст = новоеUWPHolsContext();
   8. Администратор домена =   
       новое EntityDomainManager<TodoItem -> (контекст, запрос);
   9. }
   10. // Таблицы GET/TodoItem
   11. Открытый IQueryable<TodoItem> GetAllTodoItems()
   12. {
   13. возврат запрос ();
   14. }
   15. // Таблицы GET/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   16. Открытый SingleResult<TodoItem> GetTodoItem (строковый ID)
   17. {
   18. возврат Возврат (ID);
   19. }
   20. // PATCH tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   21. открытый Задача <TodoItem> (Patch <TodoItem> в строковом ID, )
   22. {
   23. возврат UpdateAsync (ID, патч);
   24. }
   25. // POST tables/TodoItem
   26. Открытая синхронизация Задача<IHttpActionResult> PostTodoItem ( TodoItemэлемент)
   27. {
   28. TodoItem исходное состояние ожидание InsertAsync (элемент);
   29. Возврат CreatedAtRoute("создаваемое Таблицы" ID = исходному состояниюID }, исходное состояние);
   30. }
   31. // Удалить Таблицы/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
   32. открытая часть Задача удалить TodoItem (строковое ID)
   33. {
   34. возврат DeleteAsync (ID);
   35. }
   36. }
3. В папке **App\_Start** откройте файл **Startup.MobileApp.cs**. Этот файл содержит код конфигурации для службы, включая метод Начального **Значения UWPHolsInitializer**класс. Этот метод выполняется первый раз, когда запрос задан службе REST, и вставляет два элемента данных, созданные в этом методе, в базу данных (в целях Quickstart):

**C#**

Защищенное override void Значение (контекст UWPHolsContext)

{

Список< todoItem >TodoItems = НовоеСписок<TodoItem>

{

Новое TodoItem { Id = Guid.NewGuid().ToString(),

Текст = "Первый элемент", Завершенныйложный },

Новое TodoItem { Id = Guid.NewGuid().ToString(),

Текст "Второй элемент" Завершенныйложный }, .

};

foreach: TodoItem todoItem в todoItems

{

Контекст.Установка<TodoItem>().Добавление (todoItem);

}

Основа.Исходное значение (контекст);

}

1. Настройте свою Конфигурацию Решения на устранение неполадок и свою Платформу Решений на любой ЦП. Выберите Microsoft Edge из выпадающей Debug Target возле Кнопки Пусковая Отладка.

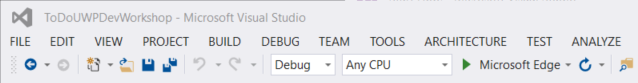


Рисунок 4

* + 1. Сконфигурируйте свой веб-сайт для работы на Локальном компьютере.
  1. **Примечание:**  Пусковая кнопка устранения неполадок.

1. Создайте и запустите своё приложение. Вы увидите окно веб-браузера, показывающее экран ‘запуска’ информации по умолчанию для серверного компонента мобильного на Azure приложении.

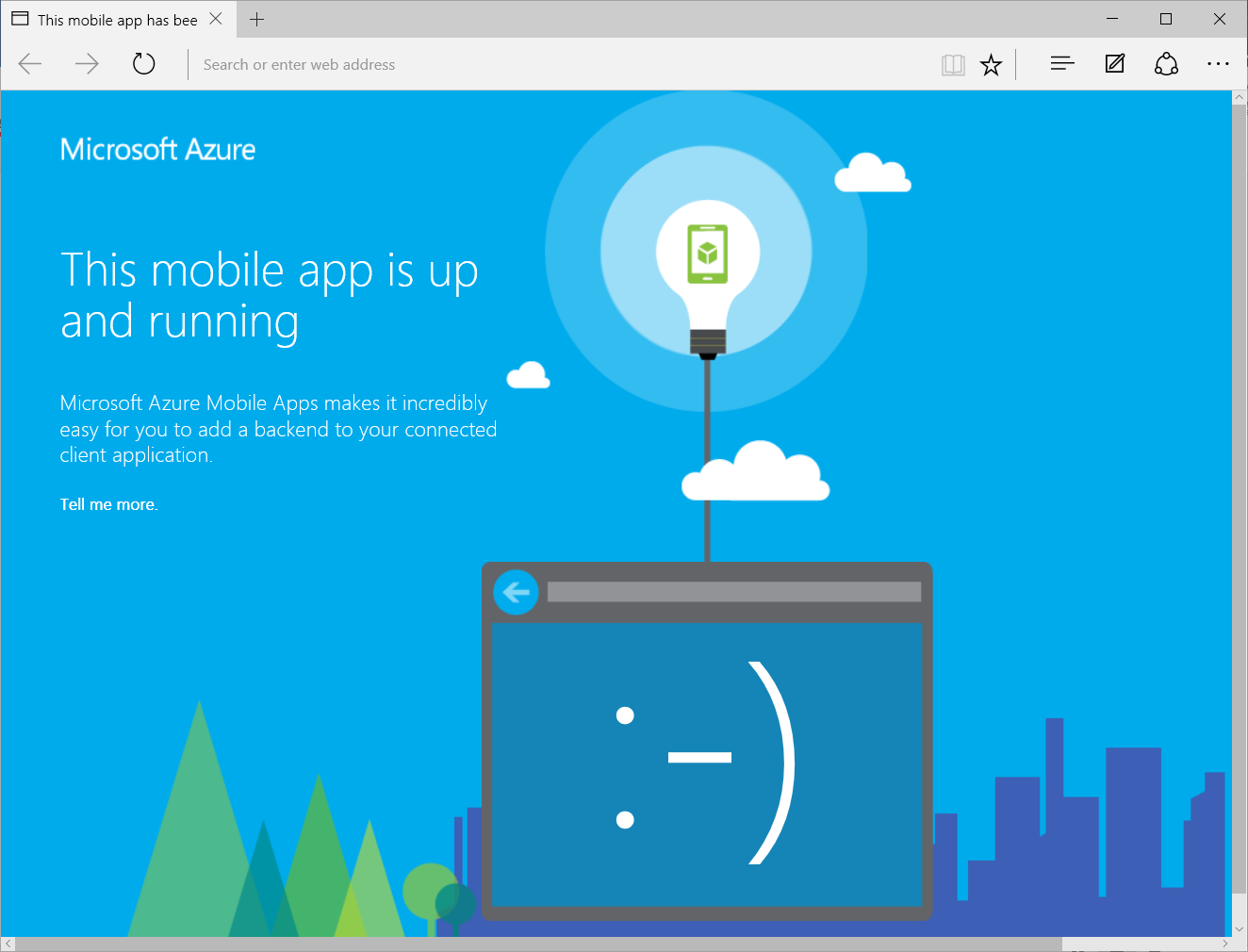


Рисунок 5

Серверный компонент мобильного приложения на Azure, работающий локально на вашем ПК.

* 1. **Примечание:** Локальный запуск вашей службы серверных компонентов даёт вам один способ устранения неполадок с вашим приложением.

Задача 3 – Открыть и запустить проект приложения клиента

Мы рассмотрим код приложения Windows UWP, который соединяется с серверным компонентом мобильного приложения на Azure и выполнит её конфигурацию, чтобы соединиться со службой серверных компонентов, которая локально выполняется на вашем ПК.

1. Откройте второй экземпляр **Visual Studio 2015**. В меню **Файл** кликните **Начать Проект/Решение**. Перемещайтесь в папку, в которой вы сохранили код для этой лаборатории, и откройте **Упражнение 1\Begin-Client\ToDoUWPDevWorkshop.sln**.
2. Настройте свою Конфигурацию Решения на **устранение неполадок** и свою Платформу Решений в **x86**. Выберите **Локальный** **компьютер** из Цели Отладки, выпадающей возле Кнопки Пуск устранения неполадок.

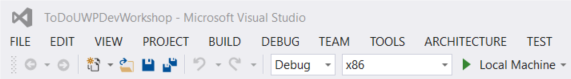


Рисунок 6

* + 1. Сконфигурируйте своё клиентское приложение, чтобы запустить на локальном компьютере.

1. Создайте проект, чтобы восстановить пакеты NuGet.
2. Откройте **App.xaml.cs**. Обратите внимание, что на вершине этого класса есть код для инстанцирования объекта **MobileServiceClient**. Первое раздел объявлений является формой, которую вы используете, чтобы подключиться к облаку – оставьте это закомментированным. Второе незакомментированное объявление является формой, которая подключается к локальной версии облачного сервиса, который вы просто запустили в предыдущих шагах.

**C#**

Частичный класс Приложение: Приложение

{

// Этот код раскомментирован для конфигурации MobileServiceClient к

// Обменивайтесь данными со Службой Мобильного сервиса на Azure и

// Шлюз Azure, использующий ключ приложения. Вы обладаете всем необходимым для запуска

// Работа с вашей Службой Мобильного!

//Открытый статический MobileServiceClient MobileService =

// Новый MobileServiceClient(

// "https://uwpdevhols.azurewebsites.net",

// "",

// ""

//);

// Используйте этот код для конфигурации MobileServiceClient

// Обменивайтесь данными с локальным объектом

// Протестируйте проект с целью устранения неполадок.

Открытый статический MobileServiceClient MobileService =

Новый MobileServiceClient(

"http://localhost:50781"

);

...

1. Откройте **MainPage.xaml.cs**. В этом классе вы можете видеть, как ваш код приложения обменивается данными с облачным сервисом серверных компонентов и загружает объекты из службы и вставляет, обновляет и удаляет объекты.  
   На верху класса, поле **todoTable** типа **IMobileServiceTable<TodoItem>** инициализировано вызовом **App.MobileService.GetTable<TodoItem>()**. Этот объект используется по всему классу, чтобы выполнить операции по типизированным данным для таблицы.  
   Например, чтобы вставить новый элемент данных, вы используете метод **InsertAsync**этого объекта **todoTable**:

**C#**

Закрытая часть IMobileServiceTable<TodoItem> todoTable =

App.MobileService.GetTable<TodoItem>();

...

Закрытая асинхронная передача Задача InsertTodoItem(TodoItem todoItem)

{

// Этот код вставляет новый TodoItem в базу данных. Когда

// операция завершена, Службе Мобильного присваивается ID,

// Элемент добавлен в CollectionView

Подождите todoTable.InsertAsync(todoItem);

items.Add(todoItem);

}

...

1. Создайте и запустите своё приложение. Когда ваше клиентское приложение запущено, код в методе MainPage OnNavigatedTo вызывает RefreshTodoitems, который вызывает вашу службу серверных компонентов, чтобы извлечь элементы любого ToDo, хранимые во внутренней базе данных.
   1. **Примечание**: Если ваше приложение дает сбой с HttpRequestException, вероятно,потому что вы прекратили устранение неполадок UWPHolsService в конце предыдущей задачи. Запустите снова проект службы серверных компонентов, оставьте его в запущенном состоянии и затем запустите клиентское приложение в отдельной копии Visual Studio.
2. Так как это первый раз, когда был осуществлён доступ к службе, она автоматически сконфигурирует базу данных и запустит метод Исходного Значения, который вы рассмотрели ранее и который вставляет два элемента. Через некоторое время вы увидите два элемента, отображенные в клиентском приложении UI.

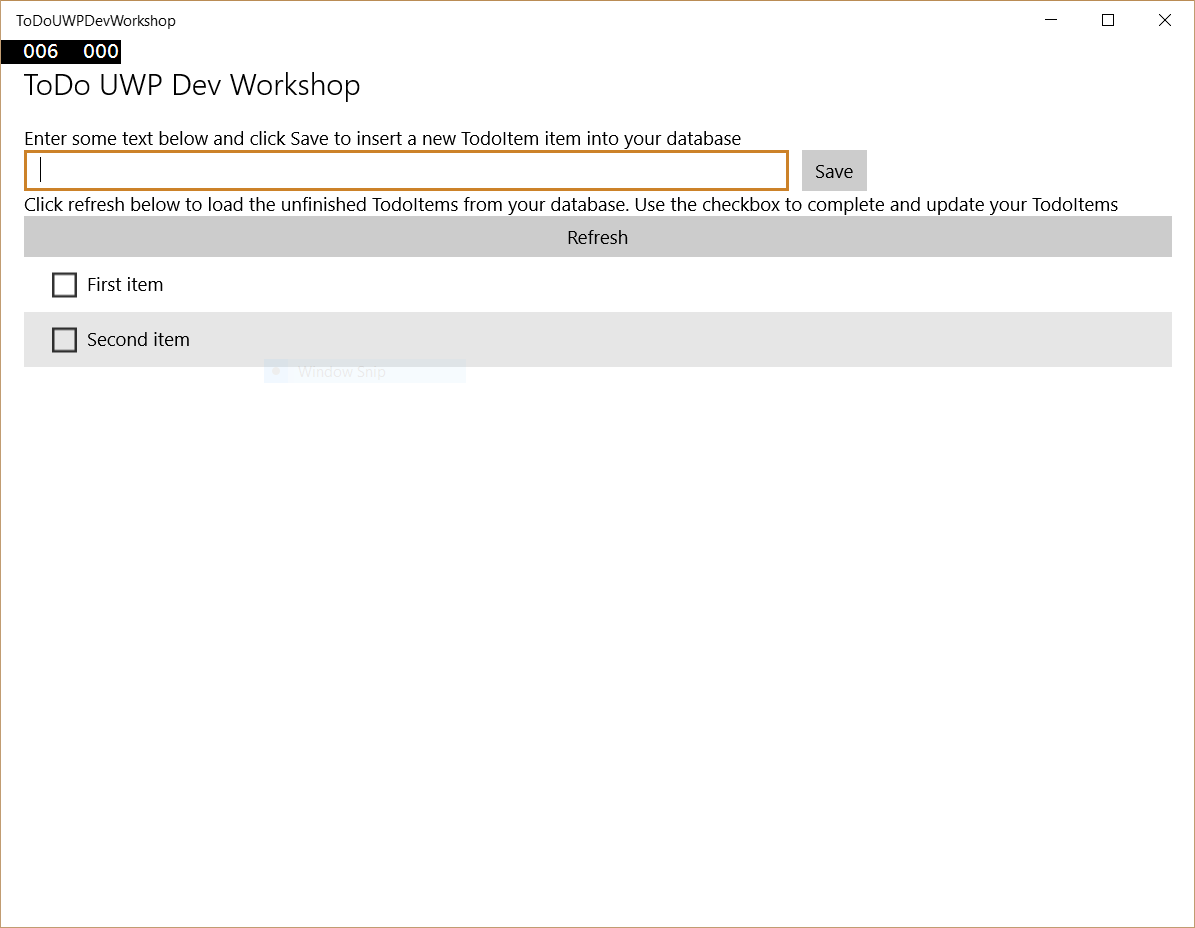


Рисунок 7

Клиентское приложение извлекает два элемента данных в "Исходном Значении" из службы серверных компонентов.

1. Вставьте новые элементы в дополнение к двум элементам, которые были добавлены методом Исходного Значения в службе.

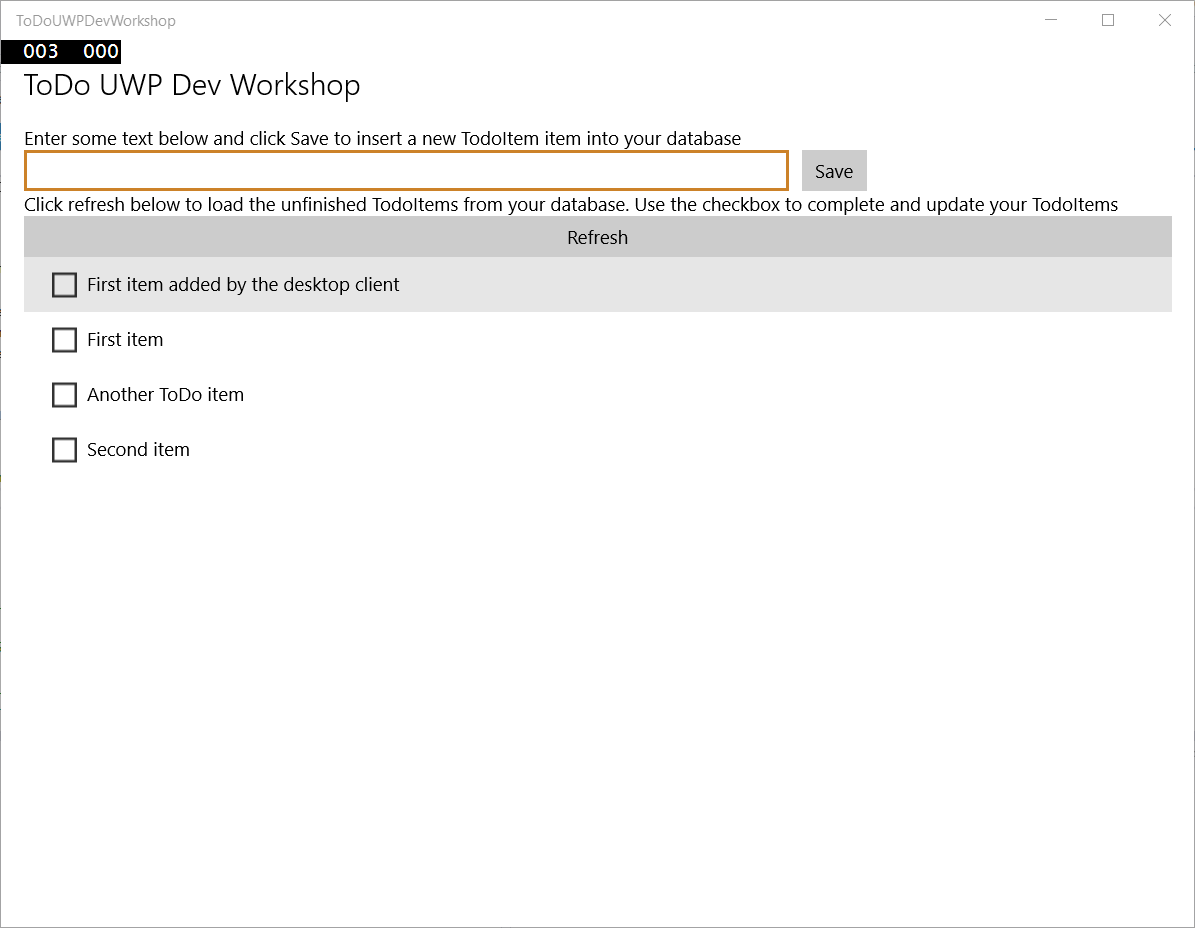


Рисунок 8

Добавление элементов новых данных.

1. Отключение устранения неполадок.
2. Если на вашем ПК установлен мобильный эмулятор разработки Windows 10, выберите  **Мобильный эмулятор…** выпадающий возле Кнопки Пуск устранение неполадок. Если вы имеете реальное устройство, подключённое и включённое для Режима Разработчиков устройство-телефон, выберите **Устройство** из выпадающей строки целевой объект отладки. Если ни одна из этих недоступна, вам придётся пропустить остальную часть этого упражнения.

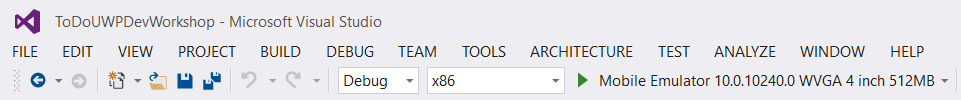


Рисунок 9

Сконфигурируйте целевой объект отладки на мобильный эмулятор или реальное устройство, если они доступны.

1. Запустите отладку. Это приложение запустится на вашем мобильном устройстве или эмуляторе и отобразит те же элементы, которые ввели на рабочий стол клиента и которые вы храните в базе данных серверного компонента (который запущен локально на вашем ПК в данный момент).



Рисунок 10

Клиентское приложение, выполняющееся мобильным устройством на Windows 10, использует службу тех же серверных компонентов.

1. Отключите отладку и в проекте клиентского приложения, и в проекте службы.
2. Вы завершили это упражнение и запустили приложение мобильного через службу серверных компонентов мобильного по двум различным устройствам, которые совместно используют данные. Вы создали *опыт использования подключённых мобильных* для пользователей этого приложения.

Упражнение 2: Активировать функцию аутентификации для своего приложения и подключиться к Облаку

Это упражнение покажет вам, как добавить объединенную аутентификацию в приложение UWP, которое подключено к серверному компоненту Мобильного приложения на Azure. После того, как оно было успешно аутентифицировано и авторизовано вашим мобильным Приложением, будет отображено значение идентификатора пользователя. Вы модифицируете приложение UWP, чтобы подключиться к Службе мобильных Приложений Azure , выполняющуюся в Microsoft Azure вместо той, которая запущена локально. Служба мобильных Приложений на Azure уже была сконфигурирована, чтобы поддержать аутентификацию в отличие Azure Active Directory.

Задача 1 – Добавить аутентификацию в приложение

Мы модифицируем приложение UWP, чтобы аутентифицировать пользователей до того, как мы запросить ресурсы из вашей Службы Приложений.

1. Откройте **Visual Studio 2015**. В меню **Файл** кликните **Начать Проект/Решение**. Перемещайтесь в папку, в которой вы сохранили код для этой лаборатории, и откройте **Упражнение 2\Begin-Client\ToDoUWPDevWorkshop.sln**.
2. Создайте проект, чтобы восстановить пакеты NuGet.
3. Откройте **App.xaml.cs**. В этом упражнении код для инстанцирования объекта **MobileServiceClient** был изменён, чтобы первое объявление не комментировалось – это – форма, которую вы используете, чтобы подключиться к облаку. Второе объявление является формой, которая подключается к локальной версии облачного сервиса, которую вы использовали в предыдущем упражнении – сейчас это закомментировано.

**C#**

Частичный класс Приложение: Приложение

{

// Этот код раскомментирован для конфигурации MobileServiceClient к

// Обменивайтесь данными со Службой Мобильного сервиса на Azure и

// Шлюз Azure, использующий ключ приложения. Вы обладаете всем необходимым для запуска

// Работа с вашей Службой Мобильного!

Открытый статическийMobileServiceClient MobileService =

Новый MobileServiceClient(

"https://uwpdevhols.azurewebsites.net",

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

""

);

// Используйте этот код для конфигурации MobileServiceClient

// Обменивайтесь данными с локальным объектом

// Протестируйте проект с целью устранения неполадок.

//Открытый статический MobileServiceClient MobileService =

// Новый MobileServiceClient(

// "http://localhost:59989"

//);

...

1. В браузере идите по ссылке мобильного приложения Azure App Service: https://uwpdevhols.azurewebsites.net Должна появиться стандартная страница загрузки, указывая, что сервис запущен в облаке.

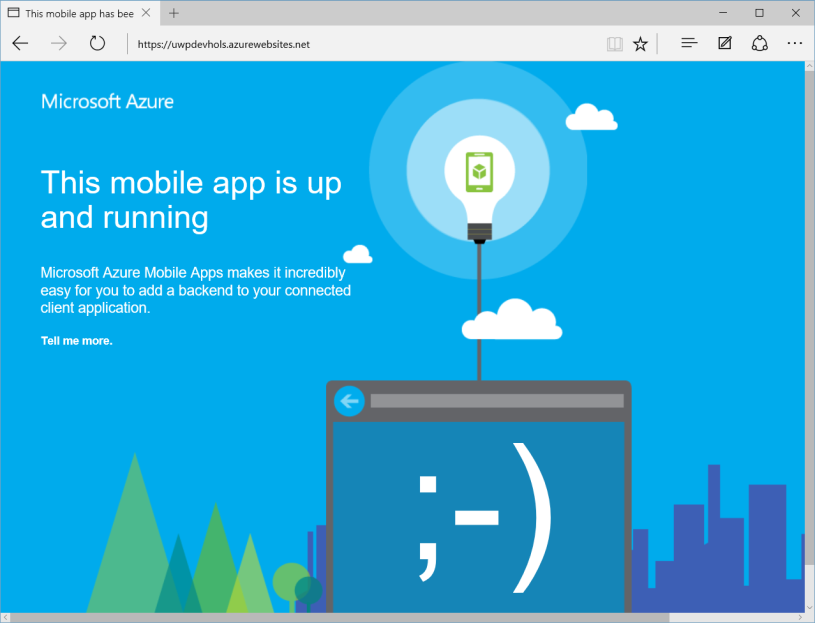


Рисунок 11

Мобильное приложение The Azure запущено в Microsoft Azure.

1. Этот сервис облака слегка отличается от того, который был запущен в предыдущем упражнении. Он был настроен на запрос, чтобы доступ предоставлялся через подтвержденных пользователей. Чтобы продемонстрировать это, запустите **TodoUWPDevWorkshop** приложение клиента Приложение не сможет запустить элементы сервиса, потому что пользователь приложения не был аутенфицирован в отличие от провайдера.

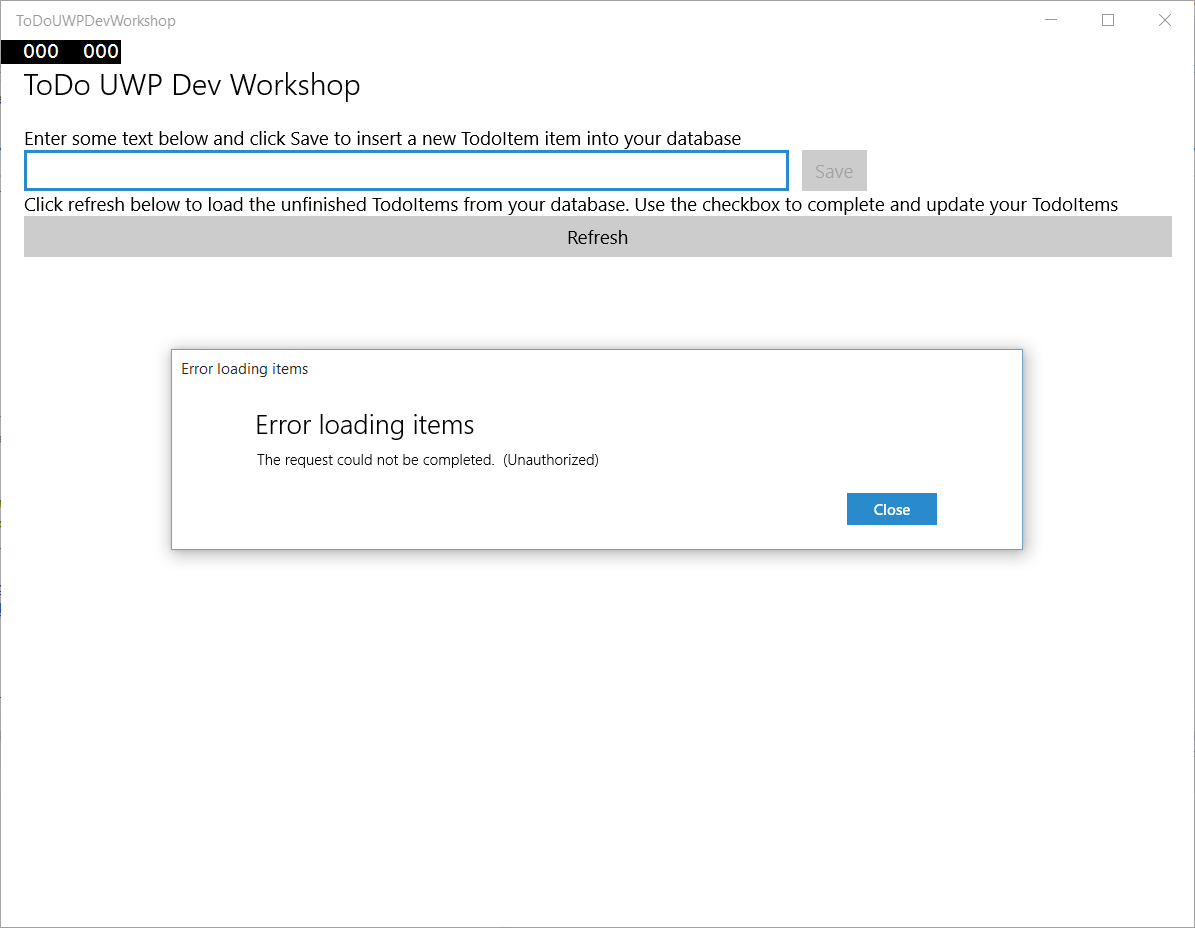


Рисунок 12

Мобильное приложение The Azure требует аутенфицированный доступ пользователя.

* 1. **Примечание**: Мобильное приложение The Azure поддерживает аутентификацию в отличие
* Azure Active Directory
* Facebook
* Google
* Microsoft
* Twitter

Сервис, к которму вы подключаетесь, был настроен на поддержку аутентификации в отличие от Azure Active Directory.

Подробные шаги и инструкции, как настроить один из таких провайдеров в мобильном приложении Azure App Service, смотрите на <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-dotnet-backend-windows-store-dotnet-get-started-users-preview/>, topic**Authenticate Users**

1. Теперь вы добавите аутентификацию в приложение Откройте **MainPage.xaml.cs**. Определите переменную для хранения зарегистрированных пользователей и способ осуществления процесс аутентификации.

**C#**

//Определите переменную для хранения зарегистрированных пользователей.

закрытая частьMobileServiceUser пользователь

//Определить способ. осуществляющий процесс аутентификации

// использование Azure Active Directory

закрытая асинхронизацияSystem.Threading.Tasks.задачаAuthenticateAsync()

{

В то время как (пользователь==null)

{

Строкасообщение

// В этом примере использовался провайдер Azure Active Directory

var пользователь="AAD";

try

{

//войти с помощью аутентификации AAD

пользователь=awaitприложение.MobileService.LoginAsync(пользователь);

Сообщение=

строкаформат(Теперь вы зарегистрированы {0}", пользователь UserId);

}

catch(InvalidOperationException )

{

сообщение= Вы должны войти в систему Необходимо зарегистрироваться;

}

vardialog =new MessageDialog (сообщение);

dialog.Commands.Add(new UICommand("OK"));

await dialog.ShowAsync();

}

}

1. Закомментировать или удалить **RefreshTodoItems** способ в существующем **OnNavigatedTo**method override. Это помогает предотвратить загрузку данных до аутентификации пользователя. Затем вы добавите кнопку Войти в приложение, которое запустит аутентификацию.
2. Добавьте следующий метод на **главную страницу** класс:

**C#**

private async void ButtonLogin\_Click(object sender, RoutedEventArgs e)

{

//Зарегистрируйте пользователя и загрузите данные из мобильного приложения.

await AuthenticateAsync();

// Скрыть кнопку входа в систему и загрузить элементы из мобильного приложения.

этаButtonLogin.Visibility = Windows.UI.Xaml.Visibility.Collapsed;

await RefreshTodoItems();

}

1. Откройте **MainPage.xam** файл проект и добавьте следующее **кнопка** элемент в **StackPanel**in Row 0 of the Grid, перед **TextBlock** элемент:

**XAML**

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

<название кнопки="ButtonLogin" Click="ButtonLogin\_Click"

Visibility="Visible">Sign in</Button>

<TextBlock Style ="{StaticResource BodyTextBlockStyle}" текст ="Введите текст и кликните Сохранить, чтобы вставить TodoItem в базу данных" TextWrapping="Wrap"/>

</StackPanel>

1. Прежде чем запустить приложение, вам необходимо выяснить имя пользователя и пароль, чтобы использовать во время аутентификации в отличие от Azure Active Directory. Были созданы пользовательские AAD только для этих лабораторных, которые были обеспечены 500 пользователями. Чтобы узнать, какое имя пользователя и пароль использовать, необходимо зайти на сайт: **TBC**  
     
    TEMPORARY: Выберите любого пользователя из [User1@uwphols.onmicrosoft.com](mailto:User1@uwphols.onmicrosoft.com) через[User499@uwphols.onmicrosoft.com](mailto:User499@uwphols.onmicrosoft.com) пароль это Пароль ***n***, где ***n*** это то же самое число, которое было использовано в пользователе ***n***.
2. Нажмите **F5**, чтобы запустить приложение, кликните **Войти**, и зайдите в приложение, используя свои данные.
3. Когда вы войдете в систему, приложение должно запуститься без ошибок и вы должны будете сделать поисковый запрос вашего мобильного приложения и обновить данные.

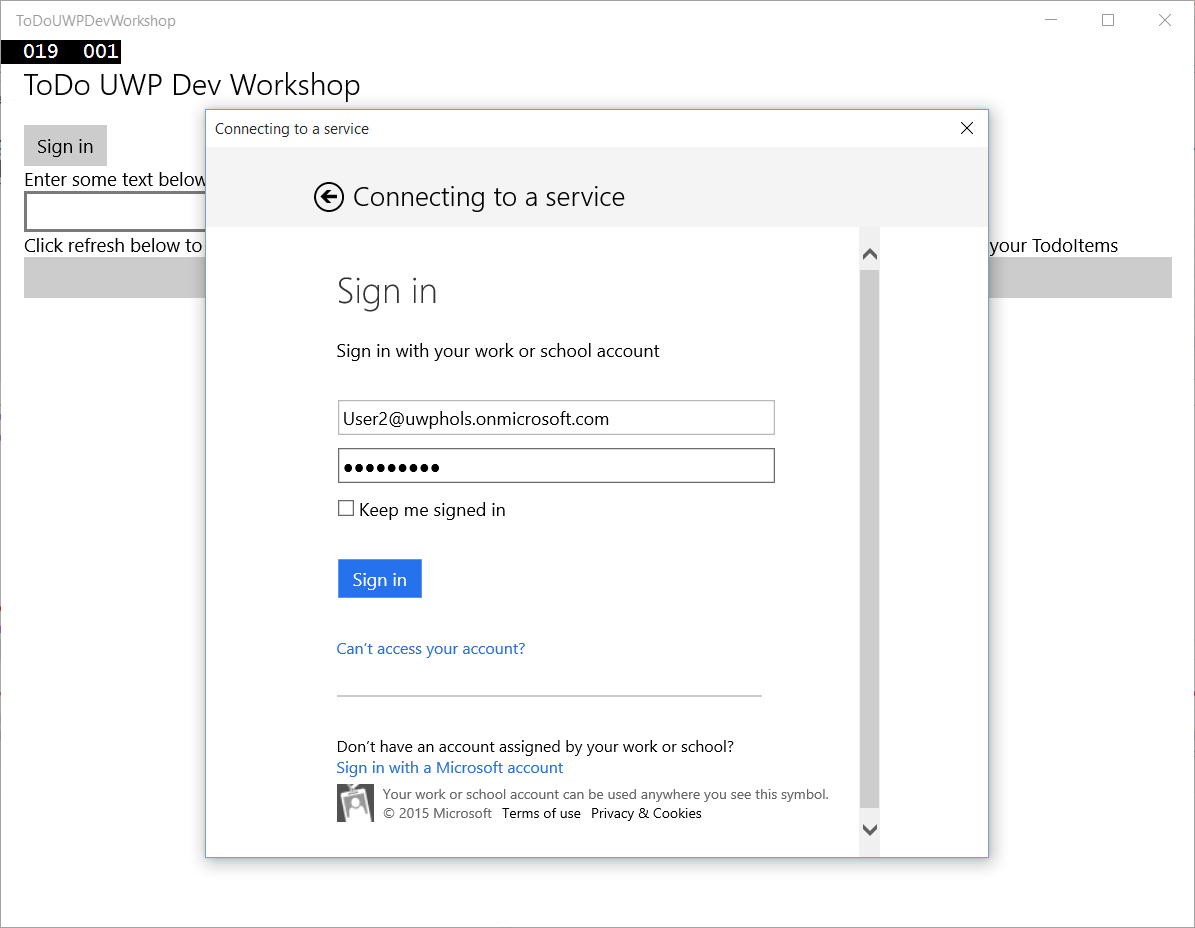


Рисунок 13

Аутентификация в отличие от Azure Active Directory.

* 1. **Примечание:** Мобильное приложение The Azure App Service, к которому вы подключены, используется и другими пользователями сервиса. Все данные для пользователей хранятся в одной базе данных SQL Azure, и все пользователи имеют доступ к сервису REST на Azure.
  2. Чтобы поддержать многочисленных пользователей, некоторые изменения были необходимы для кода в облаке в отличие от локального сервиса, запущенного в упражнении 1.
  3. Вы можете изучить код, запущенный в сервисе, к которому вы подключены, открыв **Exercise 2\Cloud\UwpDevHols.sln**. Вот эти изменения:
* В **DataObjects\TodoItem.cs** дополнительное **UserId** поле для хранения идентификационных данных пользователей в отличие от элементов ToDo
* В **Controllers\TodoItemController.cs** в методе  **GetAllTodoItems** код, который получает пользователь и использует его, чтобы изменить запрос на элементы Todo, которые ассоциируются с пользователем; в методе **PostTodoItem**, получить идентификацию пользователя и сохранить его в папке UserId
* В дополнение **[Authorize]**  присвоить класс **TodoItemController**, который ограничивает доступ до только аутентифицированных пользователей

Задача 2 – Сохранить опознавательный признак клиента

Предыдущее упражнение показал стандартный вход в систему, который требует от клиента подключаться и к провайдеру идентификатора, и к сервису приложения каждый раз, когда запускается приложение. Этот метод не только неэффективный. Вы можете столкнуться с такими вопросами как, стоит ли нескольким пользователям запускать приложение в одно и то же время. Лучший подход это произвести кэширование базового элемента, возвращенный вашим приложением и постараться использовать первый элемент прежде чем использовать вход в систему на основе провайдера.

1. Продолжайте с **ToDoUWPDevWorkshop.sln** из предыдущего упражнения.
2. В **MainPage.xaml.cs** добавьте следующее, **используя** инструкции

**C#**

using System.Linq;

using Windows.Security.Credentials;

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

**C#**

//Определить способ. осуществляющий процесс аутентификации

// использование Azure Active Directory

закрытая асинхронизацияSystem.Threading.Tasks.задачаAuthenticateAsync()

{

Строкасообщение

// В этом примере использовался провайдер Azure Active Directory

var provider = "AAD";

//Используйте PasswordVault для безопасного хранения и доступа к данным.

PasswordVault vault = Новый PasswordVault ();

PasswordCredential credential = null;

В то время как (credential == null)

{

try

{

//Постарайтесь использовать данные из электронного хранилища

credential = vault.FindAllByResource(provider).FirstOrDefault();

}

Обработчик исключения (исключение)

{

// При несовпадающих ресурсах возникает ошибка, которую мы игнорируем.

}

если(credential != null)

{

// Создайте пользователя из сохраненых идентификационных данных.

user = new MobileServiceUser (credential.UserName);

credential.RetrievePassword();

user.MobileServiceAuthenticationToken = credential.Password;

// Установите пользователя из сохраненных идентификационных данных.

App.MobileService.CurrentUser = user;

try

{

//Попробуйте вернуть элемент, чтобы определить

Закончилось ли функционирование кэшированных данных.

await App.MobileService.GetTable<TodoItem>().Take(1)

ToListAsync();

}

catch (MobileServiceInvalidOperationException ex)

{

если (ex.Response.StatusCode ==

System.Net.HttpStatusCode.Unauthorized

{

// Удалите идентификационные данные с истекшим сроком базового элемента

vault.Remove(credential);

credential = null;

Продолжайте ;

}

}

}

Другое

{

try

{

// Войдите в систему с идентификатором провайдера

user = await App.MobileService

.LoginAsync(provider);

// Создайте и храните идентификационные данные пользователя.

credential = new PasswordCredential (provider,

user.UserId, user.MobileServiceAuthenticationToken);

vault.Add(credential);

}

catch(InvalidOperationException )

{

сообщение= Вы должны войти в систему Необходимо зарегистрироваться;

}

vardialog =new MessageDialog (сообщение);

dialog.Commands.Add(new UICommand("OK"));

await dialog.ShowAsync();

}

}

В этой версии **асинхронной аутентификации** приложение пытается использовать данные, которые хранятся в PasswordVault, для доступа к мобильному приложению. Отправлен простой запрос, чтобы убедиться, что действие базового элемента не закончилось. Когда 401 возвращен, осуществляется попытка входа в систему обычного провайдера. Также обычный вход в систему осуществляется, когда отсутствуют хранимые идентификационные данные.

* 1. **Примечание:** Это приложение тестирует истекший базовый элемент во время входа в систему, но это может произойти и после аутентификации, когда используется приложение. Чтобы решить проблемы с истекшим сроком, обратитесь к разделу  [Кэширование и истекший срок базового элемента в Azure Mobile Services, управляемой SDK](http://blogs.msdn.com/b/carlosfigueira/archive/2014/03/13/caching-and-handling-expired-tokens-in-azure-mobile-services-managed-sdk.aspx).

1. Перезапустите приложение 2 раза.  
   Во время первого запуска осуществите вход в систему с помощью провайдера. Во время второго запуска потребуются кэшированные идентификационные данные.
2. **На выбор** **Характеристика PasswordVault** это то, что идентификационные данные, которые вы храните, перемещаются через облако по всем устройствам, где установлено это приложение. Если у вас есть второе устройство, где запущено Windows 10 или Windows 10 Mobile, такие как телефон, планшет или ПК, и это устройство привязано к тому же аккаунту Microsoft, что и разработчик ПК, вы можете попробовать установить то же приложение UWP на это устройство.   
   Если вы немного подождете, когда завершится перемещение идентификационных данных, когда вы войдете в приложение на одном из устройств, он обнаружит то же хранилище идентификационных данных в **PasswordVault** и вам не придется заново заходить в систему если только срок не истек.,

Упражнение 3: Активировать синхронизацию автономных режимов для своего приложения

Это упражнение покажет, как в режиме оффлайн добавлять саппорт к приложению UWP на Azure Mobile App. Синхронизация оффлайн позволяет использовать функцию просмотра, добавления или модификации приложения, даже если нет соединения с сетью. Изменения сохраняются в локальной базе данных. Как только устройство вновь онлайн, изменения синхронизируются с серверным приложением.

В этом упражнении вы обновите приложение Windows, чтобы поддержать характеристики оффлайн приложения Azure.

Задача 1 – Обновить клиентское приложение, чтобы поддержать особенности автономного режима

* 1. Оффлайн характеристики мобильного приложения Azure позволяют взаимодействовать с локальной базой данных. Чтобы воспользоваться этим в вашем приложении, вам необходимо инициализировать **MobileServiceClient.SyncContext** в локально хранилище. Затем предоставьте ссылку через интерфейс **IMobileServiceSyncTable** В этом пособии мы используем SQLite для локального хранения.
  2. Первое задание - открыть ToDoUWPDevWorkshop, которое вы создали в первом упражнении.

1. Перемещайтесь в файл, где вы сохранили **ToDoUWPDevWorkshop** в Упражнении 2. Откройте **ToDoUWPDevWorkshop.sln** в Visual Studio 2015.
2. Установите SQLite для Universal Windows Platform.
   1. В Visual Studio, в **Tools** кликните **Extensions and Updates**
   2. Слева от Extensions and Updates, кликните **Online**
   3. В строке поиска справа сверху, введите **SQLite**
   4. Когда отобразится результат поиска. прокрутите вниз, пока не увидите **SQLite for Universal App Platform**. Если диск SDK еще не установлен, выберите этот элемент, затем кликните на кнопку **Download**.

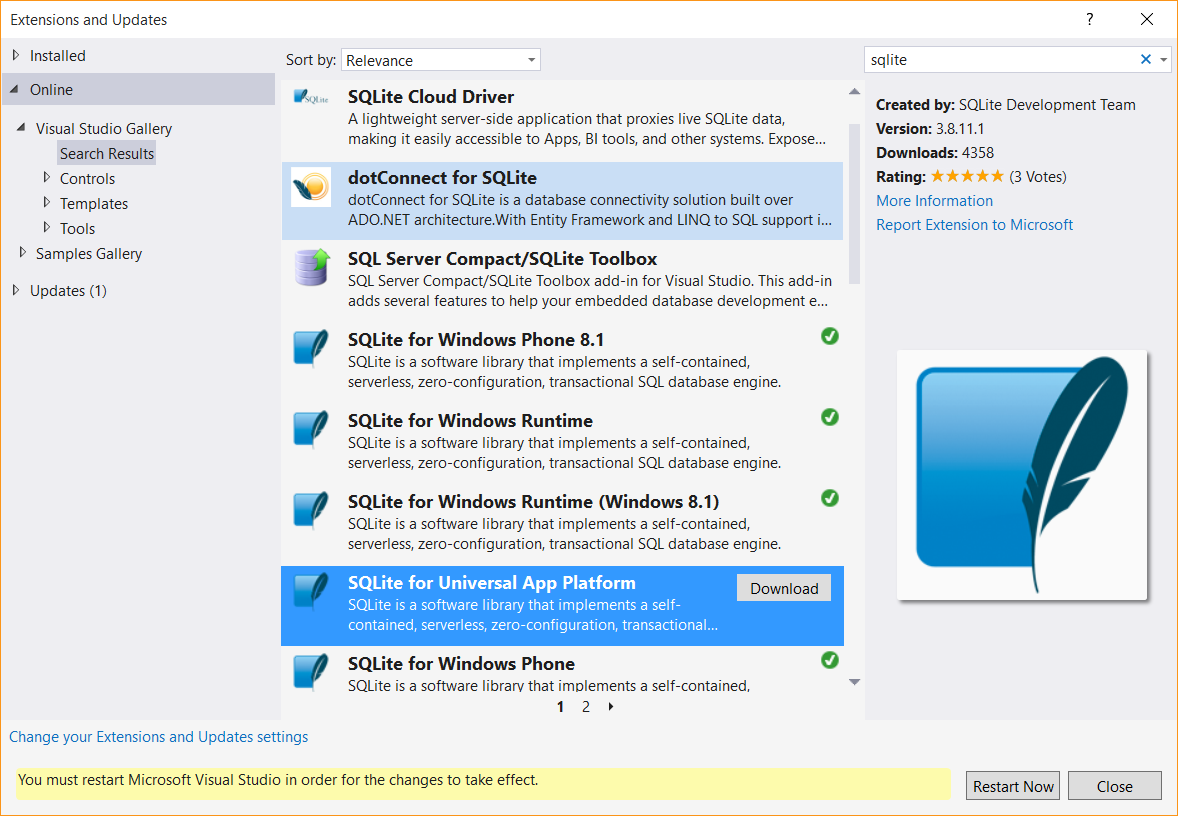


Рисунок 14

* + 1. Скачайте и установите SQLite для Universal App Platform SDK.
  1. Когда на экране появится приглашение ввода UAC, кликните **OK.**
  2. В VSIX Installer window кликните **Install**. После программного расширения кликните **закрыть**
  3. Кликните на кнопку **запустить сейчас** в окне Extensions and Updates и подождите пока запустится Visual Studio 2015.

1. Добавить ссылку в SQLite runtime dll к вашему проекту.
   1. В Solution Explorer кликните References node в проекте и затем кликните **добавить ссылку**, чтобы запустить Reference Manager.
   2. В категории "Universal Windows" выберите опцию "Extensions" в навигационной панели слева.

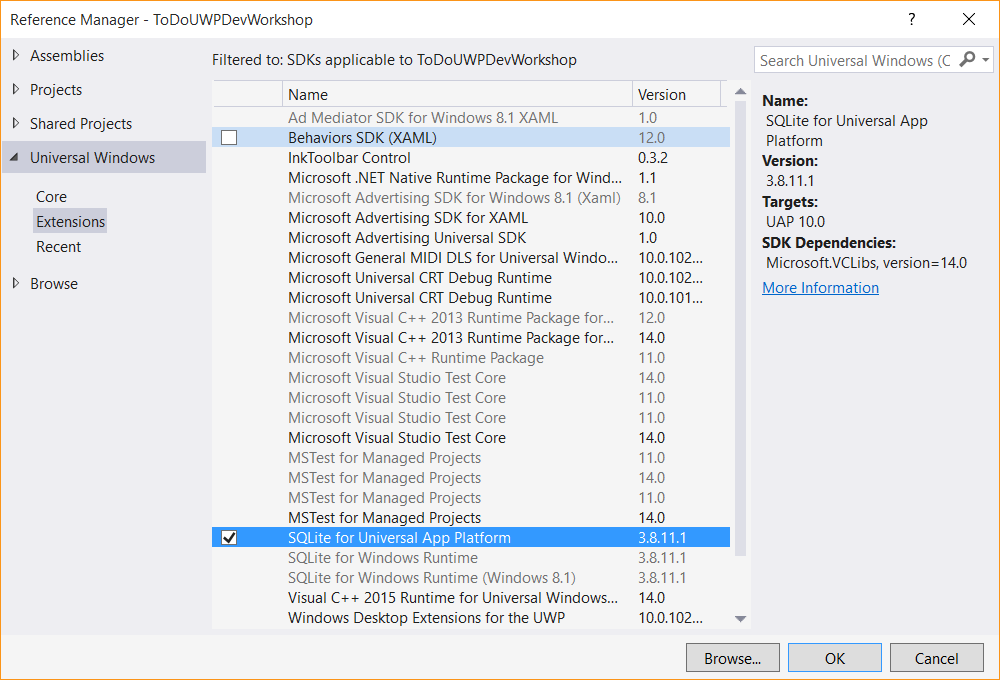


Рисунок 15

* + 1. Добавить ссылку в SQLite runtime dll к вашему проекту.
  1. Выберите**SQLite для Universal App Platform** и затем кликните **ок**.

1. Установите WindowsAzure.MobileServices.SQLiteStore NuGet package.
   1. В Solution Explorer кликните на проект, а затем на **Manage Nuget Packages**, чтобы запустить NuGet Package Manager.
   2. В онлайн табулатуре выберите опцию "Include Prerelease" в ниспадающей строке сверху. Начните поиск **SQLiteStore**, чтобы поместить 2.0.0-beta**WindowsAzure.MobileServices.SQLiteStore**.
   3. Затем кликните **установить**, чтобы добавить ссылку на проект the NuGet.
   4. Кликните **Я принимаю** в окне принятия лицензии.
2. В Solution Explorer откройте  **файл MainPage.cs**. Разкомментируйте следующее используя инструкции:
   * 1. C#
   1. используяMicrosoft.WindowsAzure.MobileServices.SQLiteStore; синхронизация оффлайн
   2. используяMicrosoft.WindowsAzure.MobileServices.Sync; синхронизация оффлайн
3. В MainPage.cs прокомментируйте код, который инициализирует todoTable как IMobileServiceTable. Разкомментируйте код, который инициализирует todoTable как IMobileServiceSyncTable:

C#

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

1. В **MainPage.cs** в разделе с пометкой **синхронизация оффлайн** разкомментируйте методы**InitLocalStoreAsync** и **SyncAsync**. Способ InitLocalStoreAsync инициализирует контекст синхронизации клиента с SQLite. В Visual Studio вы можете выбрать все откомментированные строки и использовать Ctrl+K+U, чтобы разкомментировать.

Обратите внимание, что в **SyncAsync** push операции выполняются MobileServiceClient.SyncContext вместо IMobileServicesSyncTable. Это происходит потому что контекст отслеживает изменения. произведенные клиентом. Это подходит в случаях, где таблицы связаны между собой. За дополнительной информацией обратитесь [Offline Data Sync в Azure Mobile Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/).

C#

* 1. private async Task InitLocalStoreAsync()
  2. {
  3. Если(!App.MobileService.SyncContext.IsInitialized)
  4. {
  5. var store = new MobileServiceSQLiteStore("localstore.db");
  6. store.DefineTable<TodoItem>();
  7. await App.MobileService.SyncContext.InitializeAsync(store);
  8. }
  9. await SyncAsync();
  10. }
  11. private async Task SyncAsync()
  12. {
  13. await App.MobileService.SyncContext.PushAsync();
  14. await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
  15. }

1. **Примечание:** В этом примере PullAsync мы извлечем все записи в todoTable, но также возможна фильтрация записей, отправив запрос. Первый параметр PullAsync это ID запроса, который используется в инкрементальной синхронизации, которая использует временную отметку UpdatedAt, чтобы использовать записи, измененные после последней синхронизации. ID запроса должен быть объяснительной строкой, которая является уникальной для каждого запроса в вашем приложении клиента. Чтобы отказать от инкрементальной синхронизации, установите ноль в качестве ID запроса. Это извлечет все записи операции, которые потенциально неэффективны.
2. В OnNavigatedTo раскомментируйте InitLocalStoreAsync:

C#

* 1. Защищенная асинхронизация OnNavigatedTo(NavigationEventArgs e)
  2. {
  3. await InitLocalStoreAsync(); // offline sync
  4. // await RefreshTodoItems();
  5. }

1. Раскомментируйте SyncAsync в методах InsertTodoItem ,UpdateCheckedTodoItem , и ButtonRefresh Кликните

C#

* 1. Закрытая асинхронная передача Задача InsertTodoItem(TodoItem todoItem)
  2. {
  3. Подождите todoTable.InsertAsync(todoItem);
  4. items.Add(todoItem);
  5. await SyncAsync(); // offline sync
  6. }

1. ...
   1. private async Task UpdateCheckedTodoItem(TodoItem item)
   2. {
   3. await todoTable.UpdateAsync(item);
   4. items.Remove(item);
   5. ListItems.Focus(Windows.UI.Xaml.FocusState.Unfocused);
   6. await SyncAsync(); // offline sync
   7. }
   8. private async void ButtonRefresh\_Click(object sender, RoutedEventArgs e)
   9. {
   10. ButtonRefresh.IsEnabled = false;
   11. await SyncAsync(); // offline sync
   12. await RefreshTodoItems();
   13. ButtonRefresh.IsEnabled = true;
   14. }
2. Измените код в SyncAsync, чтобы добавить дополнительные средства. В режиме оффлайн MobileServicePushFailedException будет с PushResult.Status == CancelledByNetworkError.

C#

* 1. private async Task SyncAsync()
  2. {
  3. String errorString = null
  4. try
  5. {
  6. await App.MobileService.SyncContext.PushAsync();
  7. // first param is query ID, used for incremental sync  
      await todoTable.PullAsync("todoItems", todoTable.CreateQuery());   
     }
  8. catch (MobileServicePushFailedException ex)
  9. {
  10. errorString = "Push failed because of sync errors. " +
  11. Режим оффлайн \ сообщение " +
  12. ex.Message + "\nPushResult.Status: " +
  13. ex.PushResult.Status.ToString();
  14. }
  15. catch (Exception ex)
  16. {
  17. errorString = "Pull failed: " + ex.Message +
  18. "\n\nсценарий для онлафн режима, " +
  19. "Вы можете попробовать снова, когда подключитесь"+
  20. "мобильный сервис";
  21. }
  22. if (errorString != null)
  23. {
  24. MessageDialog d = new MessageDialog(errorString);
  25. await d.ShowAsync();
  26. }
  27. }

**Примечание:** MobileServicePushFailedException может произойти в обоих случаях - операция извлечения и push операции. В случае операции извлечения это происходит, чтобы убедиться, что все таблицы согласованы между собой.

1. В Visual Studio нажмите F5, чтобы заново созжать и запустить клиентское приложение. Приложение будет вести себя так же, как и до изменений синхронизации оффлайн, так как он выполняет операции на вставку, обновление. Тем не менее, это заполнит локальную базу данных, которая может быть использована в режиме оффлайн. Мы протестируем режим оффлайн в следующем разделе теперь когда локальная база данных заполнена.

Задача 2 – Проверка режима работы клиентского приложения в автономном режиме

* 1. В этом упражнении вы запустите клиентское приложение в режиме оффлайн. Когда вы добавите данные, обработчик исключений сообщит вам, что приложение работает в режиме оффлайн с **PushResult.Status == CancelledByNetworkError**. Добавленные элементы будут храниться в локальном хранилище, но не будут синхронизированы с мобильным приложением, пока вы не будете онлайн .

1. Произведите изменения, чтобы убедиться, что приложение работает в режиме оффлайн.
   1. **Опция 1** Обработайте **App.xaml.cs**. Раскомментируйте инициализацию MobileServiceClient и добавьте следующие строки, которые используют неисправный URL

C#

* 1. public static MobileServiceClient MobileService = new MobileServiceClient(
  2. "<https://uwpdevhols.azurewebsites.fail>",
  3. "<https://default43cd3457d9384f11a222c50e05bf5fae.azurewebsites.fail>",
  4. ""
  5. );
  6. **Опция 2** Измените цель отладки на эмулятор Windows 10 Mobile. Установите приложение на эмулятор. Когда эмулятор запущен, кликните **кнопку >>** на панели команд, чтобы показать дополнительные характеристики эмулятора.  
     В **Network** проверьте **Активировать сетевое моделирование** и установите  **скорость работы сети** в **No network**.

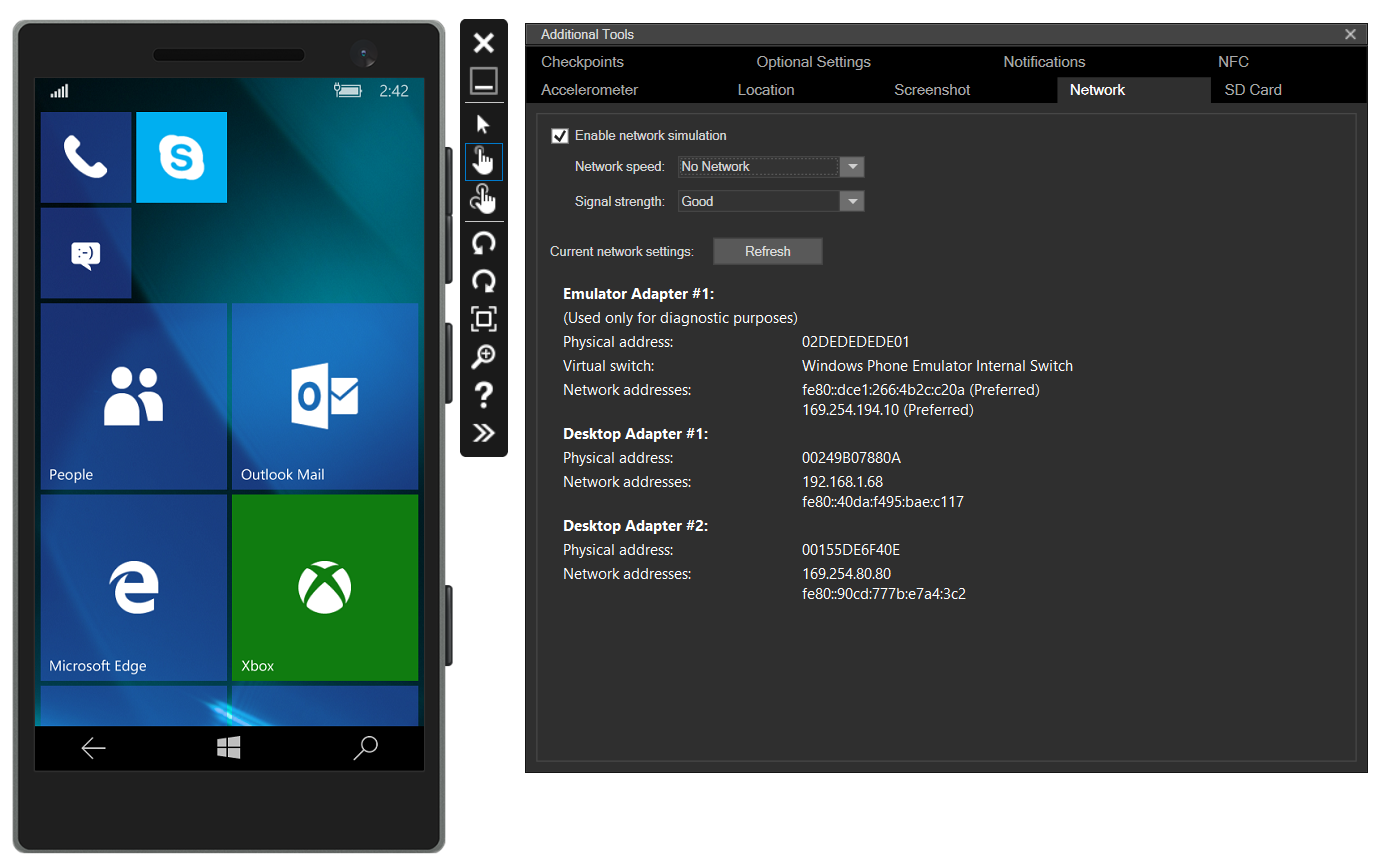


Рисунок 16

* + 1. Установка эмулятора Windows 10 Mobile для работы в режиме оффлайн.

1. Нажмите F5 для создания и запуска приложения. Обратите внимание, что синхронизация не произошла, когда вы запустили приложение.
2. Введите новые todo items и кликните **сохранить** Push не получится для каждого PushResult.Status=CancelledByNetworkError Новые todo items есть только в локальном хранилище, пока они не будут перемещены в мобильное приложение.   
   Вы можете предотвратить диалог исключений для **PushResult.Status=CancelledByNetworkError**, затем клиентское приложение будет вести себя как если бы оно было соединено с мобильным приложением серверной поддержки для операций создания, чтения, удаления (CRUD).
3. Закройте приложение и запустите его снова, чтобы убедиться, что элементы, которые вы создали, до сих пор видимы в приложении и сохранены в локальном хранилище.

Задача 3 – Повторно подключить клиентское приложение к серверному компоненту мобильного приложения

В этом упражнении вы заново подключите приложение к серверному компоненту. Приложение перемещается из режима оффлайн в режим онлайн с серверным компонентом. Когда вы первый раз запустите приложение, обработчик событий **OnNavigatedTo** обратится к**InitLocalStoreAsync** В свою очередь это приведет к тому, что **SyncAsync** синхронизируется с локальным хранилищем серверного компонента. Таким образом, приложение попытается синхронизировать после запуска приложения.

1. Переведите приложение в режим онлайн.
   1. **Опция 1** Исправьте **App.xaml.cs** Раскомментируйте вашу предыдущую инициализацию MobileServiceClient, чтобы использовать верную URL мобильного приложения.
   2. **Опция 2** Если вы тестируйте режим оффлайн мобильного эмулятора, кликните **>>** на панели команд, чтобы показать дополнительные характеристики.  
      В **Network** снимите пометку **Enable network simulation**.
2. Нажмите **F5** для создания и запуска приложения. Приложение синхронизирует локальные изменения с серверным компонентом Azure Mobile App с помощью оперций извлечения и push, как только запускается обработчик событий **OnNavigatedTo**.
3. **На выбор** Если у вас есть другое устройство с установленным приложением, запустите приложение на этом устройстве с теми же данными, что и прежде. Имейте ввиду, что Todo items. созданные прежде, сейчас синхронизированы с серверным компонентом Azure Mobile App и сейчас доступны на втором устройстве.
4. В приложении поставьте галочку напротив некоторых элементов, чтобы завершить их в локальном хранилище.

UpdateCheckedTodoItem и SyncAsync синхронизируются в серверный компонент мобильного приложения. SyncAsync выполняет и операции извлечения и push. Тем не менее, имейте ввиду, **что когда вы выполняете операцию извлечения, в которой клиент сделал изменения, операция push будет производиться автоматически.** Это происходит с целью убедиться, что все таблицы в локальном хранилище остаются совместимыми. В этом случае мы могли бы не обращаться к PushAsync, так как она осуществляется автоматически. Это может привести к неожиданной push операции, если вы не в курсе. За дополнительной информацией обратитесь [Offline Data Sync в Azure Mobile Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-offline-data-sync-preview/).

Summary

* 1. Мобильное приложение Azure App Service В этом приложении вы развили пустое приложение, созданное из шаблона Hello World app, которое отображает специфическую информацию на всех устройствах Windows 10 devices. Вы также узнали, как использовать данные в Blend для быстрого создания и визуализирования вашего приложения. В следующей лаборатории вы узнаете, как перемещаться в приложении UWP, использовать ручную навигацию и осуществлять контроль элементов навигации.