Windows 10

Лабораторный практикум

Веб-приложения в Windows

Октябрь 2015 г.

Microsoft

Overview

Следя за релизом Windows 10, набор инструментов «Windows Bridge» и, в частности, Project Westminster открывают доступ к UWP-платформе широкому спектру разработчиков с разным опытом за плечами, включая iOS-разработчиков, разработчиков классических Windows-приложений и веб-разработчиков. Мост Windows для веба позволяет вам легко перейти от вашего кода в вебе в пространство приложений, фактически опубликовав ваш отзывчивый сайт в Магазине Windows. Веб-приложения и хостящиеся веб-приложения имеют доступ к вызову UWP API напрямую из JavaScript, что позволяет интегрироваться с такими возможностями, как живые плитки, активные уведомления, контакты, голосовые команды для Кортаны и встраиваемыми покупками Магазина Windows.

Хостящиеся веб-приложения немедленно отображают изменения, сделанные в коде веб-сайта, позволяя вам легко поддерживать контент в актуальном состоянии.

После создания хостящегося веб-приложения для Windows 10 вы можете заинтересоваться расширением на другие платформы. ManifoldJS – это новый фреймворк с открытым кодом, который генерирует хостящиеся веб-приложения для популярных платформ.

Цели

Данная лабораторная научит вас:

- Создавать хостящиеся веб-приложения
- Отправлять уведомления из хостящегося веб-приложения
- Получать доступ к камере из хостящегося веб-приложения
- Задавать собственные изображения для плиток и экрана загрузки
- Обновлять живые плитки из хостящегося веб-приложения
- Генерировать хостящиеся веб-приложения для других платформ с помощью ManifoldJS

Системные требования

Для выполнения лабораторной работы вам потребуются:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Опциональные требования

Если вы хотите также сделать опциональные задачи, вым потребуются:

- The Node Package Manager (npm)
- ManifoldJS

Установка

Вы должны осуществить следующие шаги для подготовки своего компьютера к данной лабораторной работе:

1. Установите Microsoft Windows 10.
2. Установите Microsoft Visual Studio 2015. Выберите опцию настройки установки (custom install) и убедитесь, что инструменты разработки универсальных Windows-приложений выбраны в списке опций.
3. Опционально: установите npm.
4. Опционально: установите ManifoldJS

Инструкции и ссылки на установку npm и ManifoldJS приведены в Упражнении 3: Задаче 1.

Упражнения

Данный практикум включает следующие упражнения:

1. Создание хостящегося веб-приложения
2. Поддержка дополнительных платформ и устройств с помощью ManifoldJS (опционально)

Расчетное время для завершения курса: **От 30 до 45 минут**.

Упражнение 1: Создание хостящегося веб-приложения

Имея уже размещенный в сети отзывчивый веб-сайт, вы можете создать хостящееся веб-приложение для Магазина Windows за считанные минуты. В данном упражнении мы создадим хостящееся веб-приложение, используя сайт Codepen.io в качестве примера. Codepen позволяет вам вводить и выполнять собственный код на JavaScript, CSS и HTML. Как только вы дадите доступ к Windows Runtime для вашего приложения, Codepen будет для вас отличным способом познакомиться с интеграцией с API без необходимости размещать специальный сайт на сервере. Мы вызовем уведомление из хостящегося приложения и добавим возможность сделать снимок через камеру.

Задача 1 – Создание пустого универсального Windows-приложения на JavaScript

Мы начнем с создания проекта на основе шаблона UWP Blank App JavaScript.

1. Откройте Visual Studio 2015, используйте **File > New> Project**, чтобы открыть диалог создания нового проекта. Перейдите в **Installed > Templates > JavaScript** и выберите шаблон **Blank App (Universal Windows)**.

2. Назовите ваш проект **HostedWebApp** и выберите место для сохранения вашего проекта практикум. Мы создали папку на диске **C:**, названную **HOL –** вы увидите ее на снимках экрана по ходу лабораторной работы.

3. Оставьте без изменения опцию **Create directory for solution**. Вы можете снять выделение с опции **Add to source control** если не собирайтесь управлять версиями проекта. Нажмите **OK** для создания проекта.

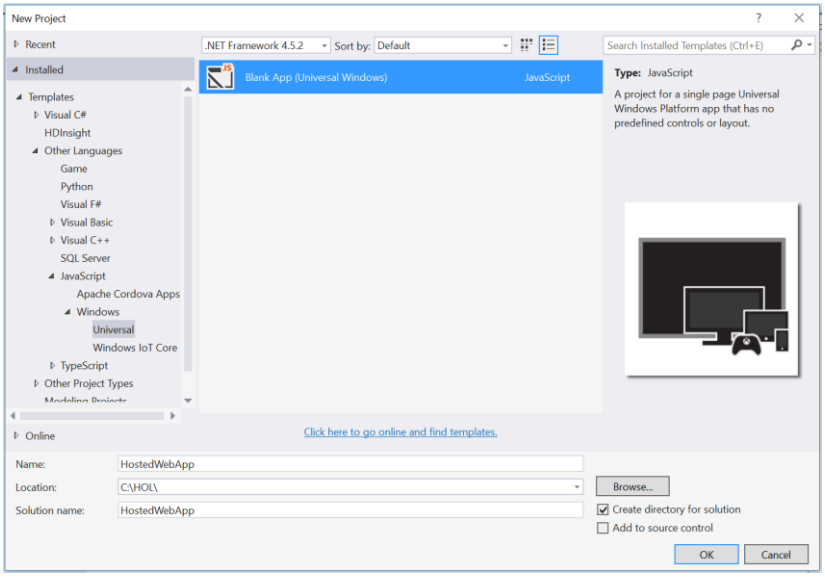

Рисунок 1

Создание нового пустого проекта в Visual Studio 2015.

4. Установите конфигурацию решения в **Debug** (Отладка) и целевую и **x86**. Выберите **Local Machine** (локальная машина) из выпадающего меню Debug Target справа от кнопки начала отладки.

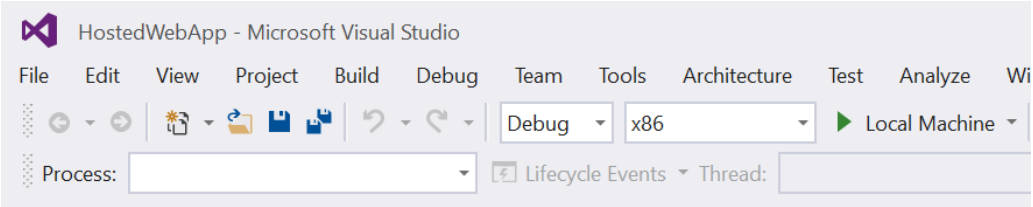

Рисунок 2

Настройте запуск приложения на локальной машине.

Примечание: -- это кнопка начала отладки.

5. Используйте кнопку начала отладки, чтобы собрать и запустить приложение. Вы увидите черный фон приложения с надписью “Content goes here.”

Рисунок 3

Пустое универсальное приложение на JavaScript, запущенное в десктопном режиме.

6. Остановите отладку и вернитесь Visual Studio.

Задача 2 – Добавление Codepen в веб-приложении

Хостящееся веб-приложение представляет собой обертку вокруг веб-сайта, предоставляющего контент, она использует движок Edge для рендеринга. Вы можете ограничить набор просматриваемых страниц конкретным сайтом или сайтами и формировать переход по остальным ссылкам во внешний браузер по умолчанию. В данной задаче мы отобразим сайт Codepen в виде хостящегося веб-приложения и запустим всплывающее уведомление из приложения.

1. Удалите папки **css**, **js**, и **WinJS**, а также файл **default.html** из проекта **HostedWebApp**.

> **Замечание:** при создании веб-приложения с исключительно хостящимся контентом вы можете удалить папки **css**, **js** и **WinJS** а также файл **default.html**. Мы удалили данные файлы, но вы можете решить их сохранить и использовать для реализации оффлайн-страниц в вашем приложении.
>
> Мы оставили папку **images**, так как в ней размещены изображения приложения, например, экран загрузки и логотип для магазина, которые по-прежнему имеют смысл в контексте хостящегося приложения.

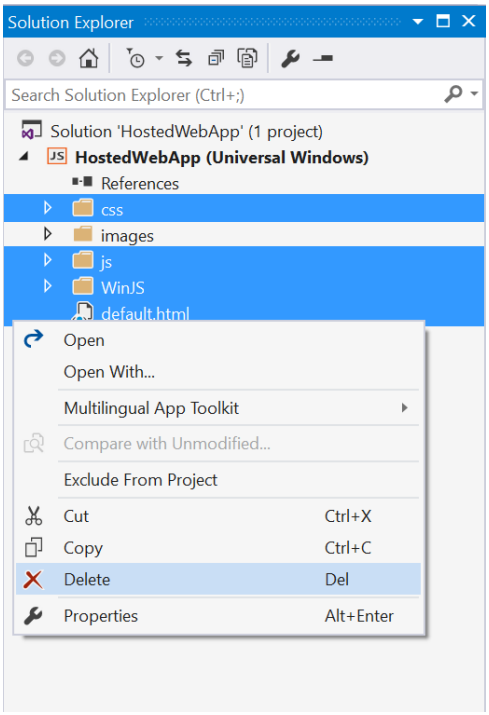

Рисунок 4

Удалите папки и файлы, которые не нужны в хостящемся приложении.

2. Откройте файл **package.appxmanifest** из вашего проекта **HostedWebApp** в редакторе манифеста.

3. На вкладке **Application** измените **Start page** на **http://codepen.io/seksenov/pen/wBbVyb/?editors=101**.

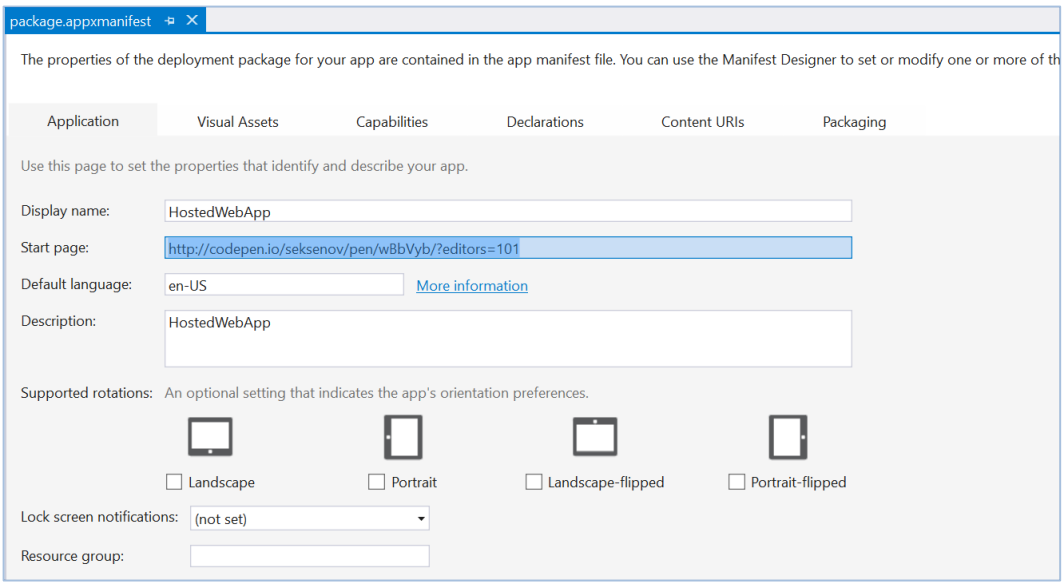

Рисунок 5

Установка Codepen в качестве стартовой страницы

4. Перейдите на вкладку **Content URIs** и добавьте **http://codepen.io/seksenov/pen/wBbVyb/?editors=101** в поле URI. Оставьте **Rule** установленным в **include** и поставьте **WinRT Access** в значение **All**.

Примечание: Правила для ссылок на контент приложения (ACURS, Application Content URI Rules) указывают страницы, которые хостятся или разрешены в вашем приложении. В частности, вы можете решить, что пользователи могут ходить по сайту Codepen внутри вашего приложения, но внешние ссылки должны открывать в браузере. Подобные включения и исключения позволяют вам контролировать границы вашего приложения и предотвращают скатывание к сценарию, напоминающему стандартный веб-браузер. Контентные ссылки также дают возможность включить или выключить доступ к Windows Runtime для различных частей приложения и решить, какой уровень доступа нужно дать: **None**, **All**, или **Allow for web only**.

Чтобы указать внешнюю ссылку используете протокол **http(s)://**. Чтобы задать локальную ссылку, используйте протокол **ms-appx-web:///**.

5. Добавьте **http://*.codepen.io/** как дополнительную контентную ссылку с теми же настройками для **Rule** и **WinRT Access**, что и в первой ссылке. Звездочка указывает покрытие всех возможных поддоменов.

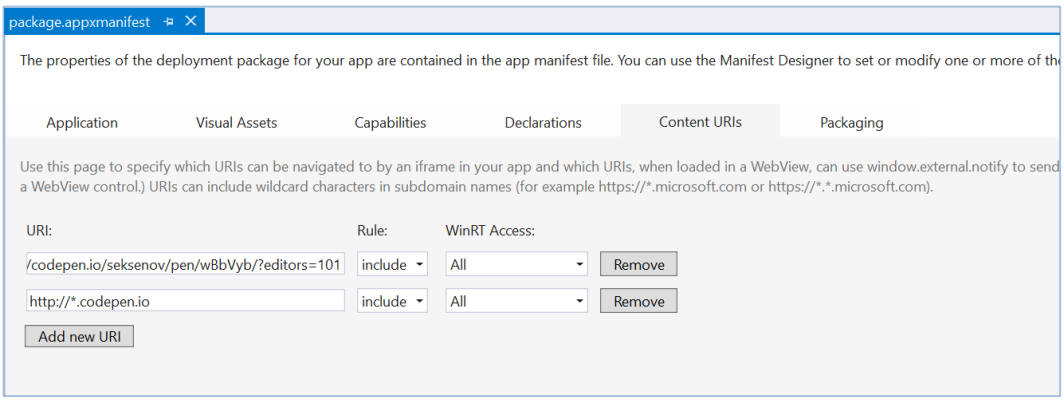

Рисунок 6

Добавление контентных ссылок для Codepen

6. Соберите и выполните приложение. Вы увидите, что сайт Codepen появился внутри окна приложения уже с некоторым кодом и контентом на HTML и JavaScript. Код генерации уведомления уже присутствует в приложении.

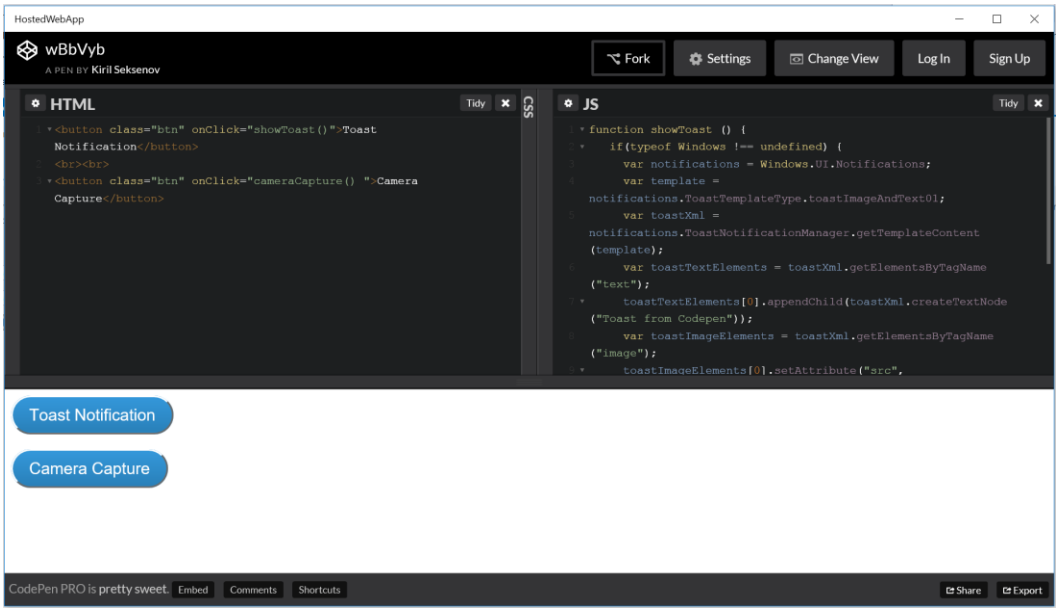

Рисунок 7

Codepen в качестве хостящегося веб-приложения.

7. Используйте кнопку «Toast Notification» для отправки уведомления в вашу систему. Мы установили свойство **WinRT Access** в значение **All** для **Codepen**, поэтому код на JavaScript в панели имеет права на доступ к Windows API, включая возможность отправки уведомлений.

Примечание: Мы используем Codepen в качестве удобного инструмента для вставки налету собственного кода на JavaScript с возможностью сохранить на удаленном сервере и немедленного его отображения для тестирования в приложении. В реальном сценарии метод **showToast()** будет работать как часть скрипта внутри веб-проекта, который вы

размещаете на некотором сервере, чтобы он начал работать. Вы можете создать отдельный проект для хостящегося веб-приложения и использовать контентные ссылки для отображения сайта в приложении.

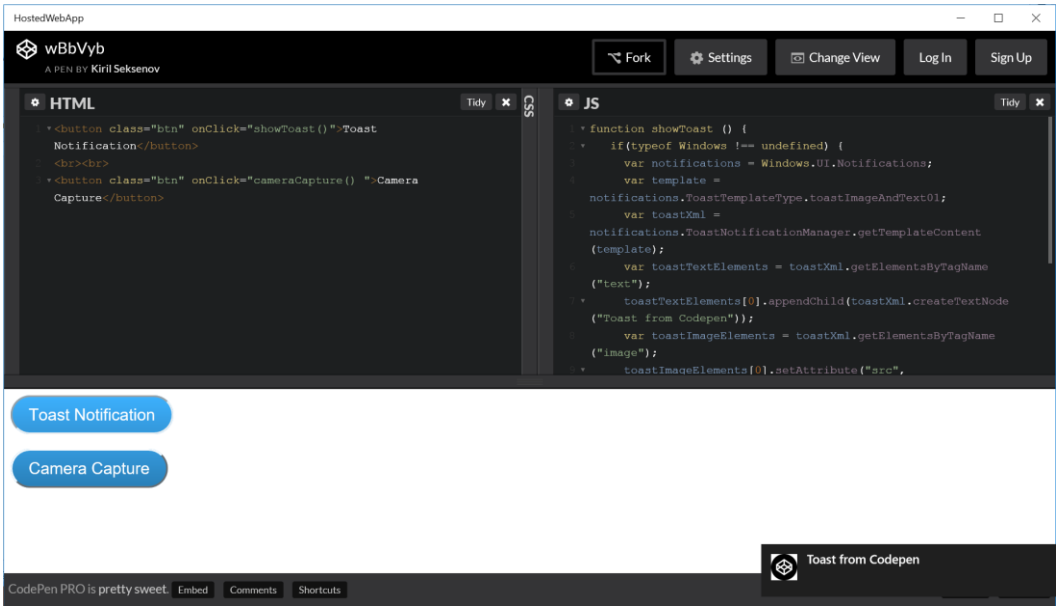

Рисунок 8

Уведомление, сгенерированное хостящимся веб-приложением

Задача 3 – Включение съемки камерой

Хотя кнопка «Camera Capture» уже определена в разметке HTML, сама съемка камерой еще не реализована. Чтобы это сделать мы добавим немного кода.

1. Во все еще работающем приложении добавьте новую функцию после метода systemAlertCommandInvokedHandler() в панели **JS**, чтобы отработать съемку камерой:

JavaScript

```
function cameraCapture() {
    if (typeof Windows != 'undefined')
    {
        var captureUI = new Windows.Media.Capture.CameraCaptureUI();
        //Set the format of the picture to be captured (.png, .jpg, ...)
        captureUI.photoSettings.format =
            Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;
        //Pop up the camera UI to take a picture
        captureUI.captureFileAsync(
            Windows.Media.Capture.CameraCaptureUIMode.photo).then(
            function(capturedItem) {
            // Do something with the picture
        });
  }
}
```

2. Используйте кнопку **Camera Capture** чтобы открыть интерфейс камеры. Если система запросит разрешение на доступ к камере, нажмите **Yes**. После открытия интерфейса камеры вы можете снять кадр с помощью кнопки камеры с правой стороны окна.

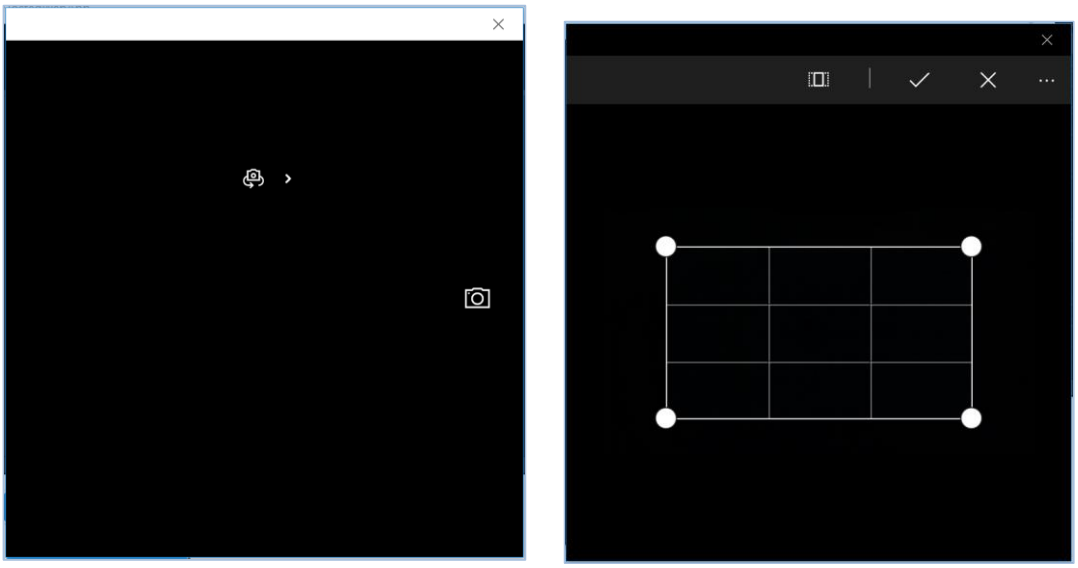

Рисунок 9

Снятие изображения с помощью открытой камеры, режим редактирования фотографии.

Примечание: мы не сохраняем фотографию внутри блока **then**, поэтому любая съемка носит временный характер.

3. Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Добавление живых плиток

Также, как и с другими UWP-приложениями, вы можете определить изображения для хостящихся веб-приложений. В рамках данной задачи мы добавим изображения для экрана загрузки, средней плитки и меню Пуск. После настройки вида плитки по умолчанию мы создадим и обновим живую плитку из приложения.

1. Откройте **package.appxmanifest** в редакторе манифеста и выберите вкладку **Visual Assets**.

2. Используйте многоточие под **Square 71x71 logo** в масштабе **Scale 200**, чтобы открыть диалог выбора изображения. Перейдите в папку **Lab Assets** в директории практикума и выберите файл **Square71x71Logo.scale-200.png**. Нажмите **Open**, чтобы заменить картинку по умолчанию выбранным логотипом. Если нужно, подтвердите, что хотите перезаписать файл в проекте.

3. Повторите шаг 2 с **Square150x150Logo** в масштабе **Scale 200**, **Square44x44Logo** в масштабе **Scale 200**, и **Splash screen logo** в масштабе **Scale 200** в соответствии с пунктами в манифесте.

Примечание: мы добавили изображения логотипов в демонстрационных целях. Для более глубокого погружения и дополнительных рекомендаций рекомендуем посмотреть лабораторную работу про работу с живыми плитками и уведомлениями.

4. Выберите раздел **All Image Assets,** измените поля **Background color** для плитки и экрана загрузки на значение **deepSkyBlue**. Промотайте вниз, чтобы убедиться, что новые изображения отображаются в манифесте.

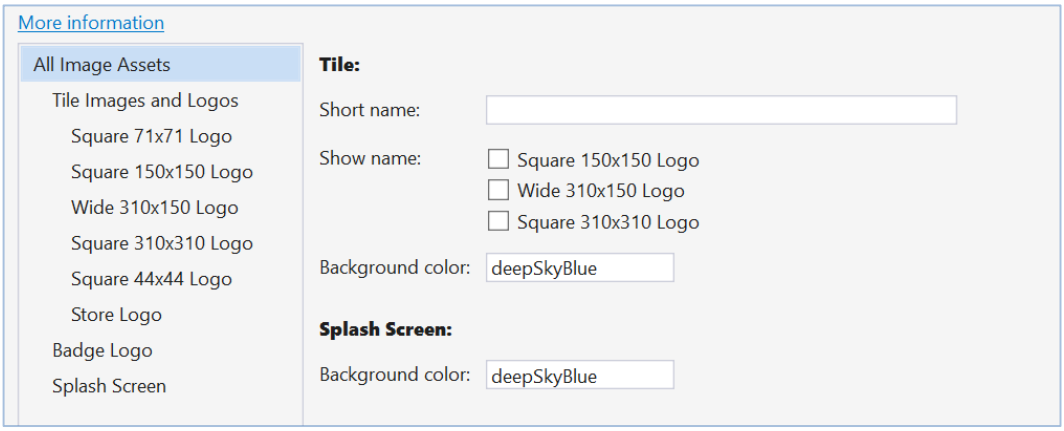

Рисунок 10

Установка цвета фона для плиток и экрана загрузки.

5. Соберите и запустите приложение. При загрузке приложения вы увидите новый экран загрузки с голубым фоном и белым логотипом практикума по центру.

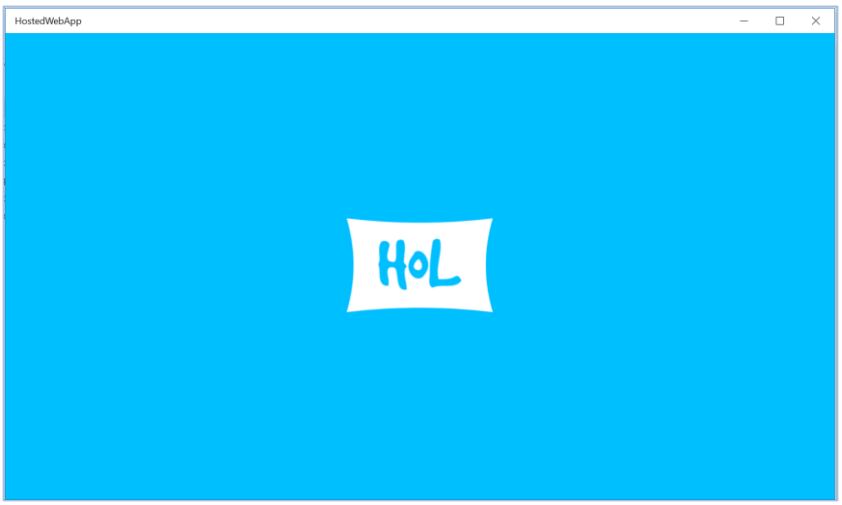

Рисунок 11

Новый собственный экран загрузки для хостящегося веб-приложения.

6. Пока приложение работает, найдите приложение **HostedWebApp** в списке всех приложений в меню Пуск. Щелкните правой кнопкой мыши по имени приложения и закрепите его в меню Пуск. Если плитка приложения стала любого другого размера, кроме среднего, поменяйте размер на средний. Вы увидите голубой фон с белым логотипом HoL.

Рисунок 12

Средняя плитка по умолчанию

7. Вернитесь к запущенному приложению. В панели HTML добавьте кнопку, которая вызывает метод **updateTile()** по событию click.

HTML

```
<button class="btn" onClick="cameraCapture() ">Camera Capture</button>
<br><br>
<button class="btn" onClick="updateTile() ">Update Tile</button>
```

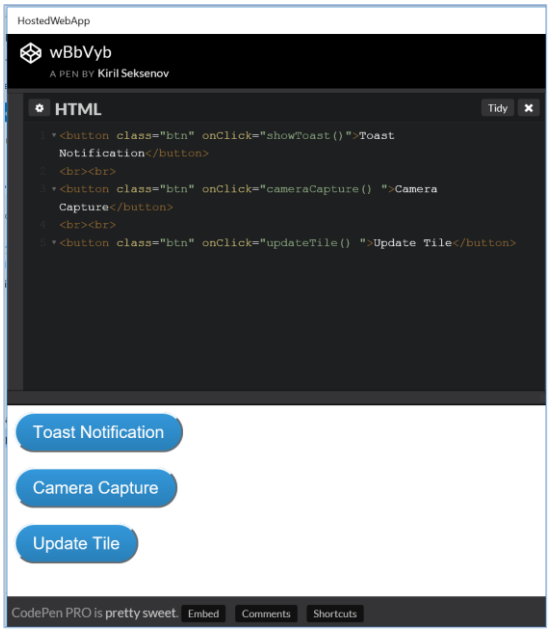

Рисунок 13

Кнопка обновления плитки в панели HTML.

8. Добавьте функцию **updateTile()** в панели **JS** после функции cameraCapture().

JavaScript

```
function updateTile() {
    if (typeof Windows !== 'undefined' && typeof Windows.UI !== 'undefined'
&&
        typeof Windows.UI.Notifications !== 'undefined')
    {
    console.log('Attempting to update the tile');
    var notifications = Windows.UI.Notifications,
        tile =
notifications.TileTemplateType.tileSquare150x150PeekImageAndText01,
        tileContent = notifications.TileUpdateManager.getTemplateContent(
            tile),
        tileText = tileContent.getElementsByTagName('text'),
        tileImage = tileContent.getElementsByTagName('image');

        tileText[0].appendChild(tileContent.createTextNode('Demo
Message'));
        tileImage[0].setAttribute('src',
            'http://unsplash.it/150/150/?random');
        tileImage[0].setAttribute('alt', 'Random demo image');

        var tileNotification = new
            notifications.TileNotification(tileContent);
        var currentTime = new Date();
        tileNotification.expirationTime = new Date(currentTime.getTime() +
20
            * 1000);

        notifications.TileUpdateManager.createTileUpdaterForApplication(
            ).update( tileNotification);

  }
  else {
    //alternate behavior
  }
}
```

Примечание: каждый раз, когда в рамках хостящегося контента мы добавляем метод, требующий доступ к API платформы, мы также проверяем, если тип (**typeof**) **Windows** определен. Это условие равно true, если сайт работает в формате хостящегося веб-приложения на устройстве с Windows 10. Дополнительно метод updateTile() также проверяет доступность **Windows.UI** и **Windows.UI.Notifications** прежде, чем к ним обращаться.

Для простоты мы используем старый шаблон для создания плитки. О настройке адаптивного шаблона можно узнать в лабораторной работе про плитки и уведомления.

9. Нажмите на кнопку **Update Tile**, чтобы вызвать обновление живой плитки. Когда вы откроете меню Пуск, вы увидите, что изображение сменяется и ваша плитка становится живой. Через несколько секунд поверх картинки появится текст.

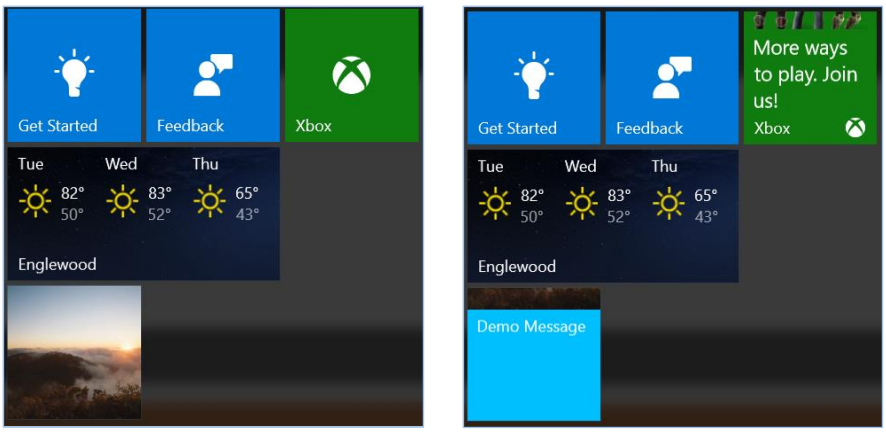

Рисунок 14

Контент живой плитки

10. Подождите 20 секунд, наблюдая за плиткой. По истечении времени плитка вернется в изначальное состояние.

> **Применчание:** внутри метода updateTile() мы отправляем сообщение через console.log. Если что-то пошло не так, посмотрите окно JavaScript Console в Visual Studio, чтобы проверить сообщение там. Если в консоли написано **Attempting to update the tile**, возможно, у вас синтаксическая ошибка в методе.

11. Остановите отладку и вернитесь в Visual Studio.

Задача 5 – Дополнительные возможности

Помимо сценариев, которые мы посмотрели в лабораторной работе, хостящиеся приложения могут интегрироваться со множеством других возможностей из богатого набора Windows 10. Хотя мы не сможем осветить подробно их все в лабораторной, есть несколько сценариев, которые могут вас заинтересовать. Дополнительная информация доступна по ссылке: http://microsoftedge.github.io/WebAppsDocs/en-US/win10/HWAfeatures.htm.

Голосовые команды для Кортаны

Вы можете интегрировать хостящееся веб-приложение с Кортаной, указав файл Voice Command Definition (VCD) в вашем html-коде, используя мета-тег. После регистрации, вы можете использовать VCD для голосовой активации приложения, запуска фоновых сервисов и реализации других способов взаимодействия с Кортаной.

Гибридные приложения

Если вы хотите, чтобы пользователи могли иметь доступ к вашем приложению в оффлайн-режиме, вы можете создать гибридное приложение. Гибридное приложение может отдавать как хостящийся контент, так и локальный, размещенный внутри пакета или локального хранилища.

Брокер веб-аутентификации

Выше хостящееся веб-приложение может использовать преимущества брокера веб-аутентификации для обработки входя пользователей, если вы опираетесь на такие интернет-протоколы, как OpenID или Oauth. URI для веб-аутентификации может быть также определен через мета-тег внутри любой html-страницы приложения.

Декларация свойств приложения

Для предоставления доступа к микрофону и другим возможностям устройства и ресурсам, вы должны объявить соответствующие возможности в манифесте приложения. Это можно сделать через редактор или вручную через XML. Подробности доступны в документации: https://msdn.microsoft.com/en-us/library/windows/apps/br211477.aspx.

Упражнение 2: Поддержка дополнительных платформ и устройств с ManifoldJS (опциональное)

Хостящиеся веб-приложения – это отличный способ быстро перенести существующий веб-проект с отзывчивым дизайном на новые платформы. ManifoldJS – это инструмент, который использует метаданные с вашего сайта для генерации нативных хостящихся приложений для разных платформ, включая iOS, Android, Windows 10, Chrome OS и Firefox OS. Для платформ, которые не поддерживают хостящиеся веб-приложения напрямую, ManifoldJS использует Cordova.

Манифест, генерируемый ManifoldJS следуют стандарту W3C для манифеста веб-приложения и включает такие метаданные, как стартовая страница сайта, белый список URL, имя сайта, цвет темы и изображения для приложения.

Примечание: Свежая версия и новости про ManifoldJS доступны на сайте http://www.manifoldjs.com/. Подробнее о манифесте W3C для веб-приложений можно узнать на сайте https://w3c.github.io/manifest/.

Задача 1 – Установка ManifoldJS и создание манифеста

1. Откройте командную строку из-под администратора. С помощью установленного npm используйте команду **npm install –g manifoldjs**, чтобы поставить ManifoldJS в глобальном режиме на вашей рабочей машине.

Command Prompt

```
> npm install –g manifoldjs
```

Примечание: Посетите сайт https://nodejs.org/, чтобы скачать и поставить пакетный менеджер node (npm).

2. Сгенерируйте манифест на сайте http://www.manifoldjs.com/generator. Вы также можете загрузить готовый манифест, чтобы генератор его поправил и сообщил вам о возможных пробелах.

Примечание: Если ваш сайт не имеет манифеста, ManifoldJS может сгенерировать его для вас. Однако вы, наверняка, захотите создать собственный, чтобы воспользоваться всеми возможностями брендирования.

3. Загрузите манифест в корень вашего сайта на сервере. Обычно манифест размещается там же, где и файл index.html.

Задача 2 – Генерация хостящегося веб-приложения

В рамках данной задачи мы создадим хостящиеся веб-приложения для вашего сайта под разные платформы.

4. Вернитесь к лоальной машине. Создайте папку для хранения хостящихся веб-приложений. Перейдите в командную строку. Передайте адрес сайта в manifoldjs для генерации манифеста. Мы используем Bing в качестве примеры. Вы можете также добавить параметр **-l debug** для словесного вывода.

Command Prompt

```
> manifoldjs http://www.bing.com/
```

5. Проверьте код, сгенерированный утилитой ManifoldJS в вашей папке.

6. Чтобы поставить и запустить сгенерированное Windows 10 приложение, запустите следующую команду из папки, созданной ManifoldJS:

Command Prompt

```
> manifoldjs run windows
```

7. Ваше приложение будет сгенерировано и запушено.

Резюме

Хостящиеся веб-приложения – это мощный способ интегрировать опыт существующего веб-проекта с возможностями Магазина Windows и API платформы. В рамках данной лабораторной работы мы создали хостящееся веб-приложение с собственными плитками, которое может отправлять уведомления, обновлять плитки и запускать камеру устройтсва. Мы также познакомились с тем, как генерировать веб-приложения для различных других платформ.