Building a Mobile Application for DotNetNuke

Bruce Chapman

Director, iFinity Software

7th November 2012

Presentation Agenda

- Brief Introduction to Mobile Applications + DotNetNuke
- Demonstrate how to build a Mobile Application that uses Service Layer of DotNetNuke to interact
  - » Understanding the Service Layer
  - » Building Service Endpoints
- Example Application : “Dnn Dash” – allows access to the Dashboard of a DotNetNuke site
  - » Building Mobile Application

Mobile Applications

- Purpose built (device specific) Mobile Applications are a superior way to deliver a mobile experience
- Html 5 can provide a rich experience, but cannot match purpose built at this point
- The theme of Mobile App vs Mobile Site is currently a hot topic of debate

HTML 5 VS NATIVE APPS

Html 5

- Write onc
- Server-sic changed
- Sandboxe features
- No App S
- Speed dep connection
- Direct monetization more difficult

Native App

- Monetization by app or in-app purchase easy

Conclusion

•Choose according to requirements

•Services layer may be largely the same in either case

Mobile Applications and DotNetNuke

- DotNetNuke 6.2 introduced the Services Layer, which exposes key DNN API features
- DotNetNuke 7.0 redefines the Services Layer, which is built on top of the WebAPI services
- DNN Services layer is easily extended for module specific purposes
- An entire new field of DotNetNuke application development has been created

Mobile Application Types

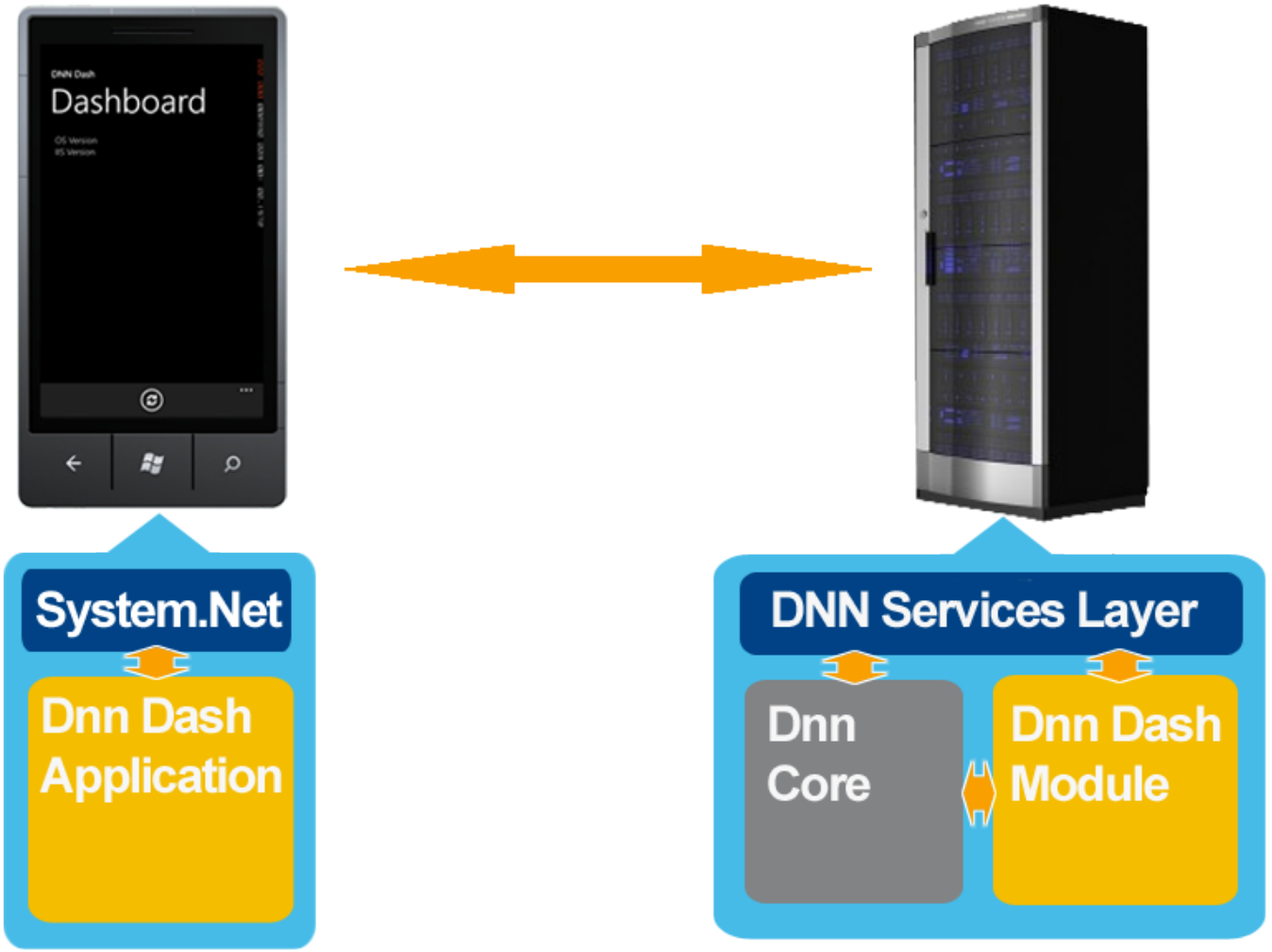
- Corporate development : customising apps for in-house use on mobile + tablet platforms
- Commercial development : new applications that provider web + mobile experience
- Open Source development : providing applications + back end processes

A Natural Fit : Mobile + Cloud

Mobile
- UX
- Features
- Always on
- Portability

Cloud
- Storage
- Processing
- Authentication

Building a DotNetNuke Mobile Application



Understanding the Service Layer

- 6.2 – Used the MVC Service Layer
- 7.0 – Uses the Web API for the Services Layer
- Two aspects to layer:
  - » Built-in DNN API
  - » Extension for Third-party modules
- Provides authentication based on DotNetNuke user accounts and profiles
- Provides simple pattern to design service endpoints
- Flexibility in format (Json/Xml/others)

Extending the Service Layer

- 2 Basic Components to API Extensions
  1. Controller
     - Definition of Methods and layer on top of business logic
     - Inherits from *DnnController*
  2. RouteMapper
     - Defines what Urls map to which Controller methods
     - Inherits from *IServiceRouteMapper*

Example Service Layer Call

```
namespace DnnDash.SimpleDashboard.Services
{
    public class DashController : DnnApiController
    {
        [AllowAnonymous]
        public string Ping()
        {
            return "Dnn Dash Version 01.00.00";
        }
        [DnnAuthorize(StaticRoles="Administrators")]
        public string PingAdmin()
        {
            return "Dnn Dash Version (Admin) 01.00.00";
        }
        [RequireHost]
        public string PingHost()
```

- Url to call:

http://example.com/DesktopModules/DnnDash/API/Dash/Ping

Deconstructing Services Layer Url

http://example.com/DesktopModules/DnnDash/API/Dash/Ping

- http://example.com => domain name
- /DesktopModules => fixed part of Url, denotes module Url
- /DnnDash => specifies module
- /API => fixed, denotes services layer
- /Dash => specifies controller [DashController]
- /Ping => method name [Public string Ping()]

- NOTE: /DesktopModules/DnnDash doesn't necessarily have to exist

RouteMapper registers Url Routes

```
namespace DnnDash.SimpleDashboard.Services
{
    public class DashRouteMapper : IServiceRouteMapper
    {
        public void RegisterRoutes(IMapRoute routeManager)
        {
            routeManager.MapHttpRoute("DnnDash_SimpleDashboard", "default"
                  , "{controller}/{action}", new[] { "DnnDash.SimpleDashboard.Services" });
        }
    }
}
```

- MapRoute configures what Urls will work:
  - "DnnDash" → /DesktopModules/DnnDash
  - "{controller}/{action}" → Class/Method
    - /API/Dash/Ping → DashController.Ping()

- No other configuration required

Services Layer Demonstration

Services Authentication

- Authentication:
  - » Identifying & Authorising User accounts
  - » Preventing Unauthorised Access
  - » Avoiding opening exploitable holes
  - » Providing ‘context’ for current user within service calls
- Web-browser based services call use cookie-based authentication (ie AJAX calls)
- Mobile devices use digest authentication to provide authentication
  - » Username/password pair provided with each call

Services Authentication cont.

- Services methods access level can be marked with:
  - AllowAnonymous – no authentication
  - RequiresHost – if true, must be SuperUser
  - StaticRoles – named DotNetNuke roles
- All ‘regular’ DNN objects are available in context for service calls:
  - Portal Settings (current portal, alias, settings)
  - UserInfo (authenticated user)

Tips for Writing Services

- Use RESTful method / parameter naming principles
- Stick to common return format in project (Json/Xml/String – whatever)
- Plan for infrequent service calls in Application Design
- Be frugal with data going backwards and forwards – this may require custom serializing of objects

```
<dotnetnuke type="Package" version="5.0">
  <packages>
    <package name="DnnDash Simple Dashboard" type="Module" version="01.01.00">
      <friendlyName>Dnn Dash Simple Dashboard API</friendlyName>
      <description>The Dnn Dash Simple Dashboard API allows authenticated clients to read Dashboard information.</description>
      <owner>
        <name></name>
        <organization></organization>
        <url><![CDATA[<a href="http://dnndash.com" target="_new">dnndash.com</a>]]></url>
        <email><![CDATA[]]></email>
      </owner>
            </releaseNotes>
            <dependencies>
              <dependency type="CoreVersion">07.00.00</dependency>
            </dependencies>
            <components>
              <component type="Assembly">
                <assemblies>
                  <assembly>
                    <path>bin</path>
                    <name>DnnDash.SimpleDashboard.dll</name>
                    <sourceFileName>DnnDash.SimpleDashboard.dll</sourceFileName>
                  </assembly>
                </assemblies>
              </component>
      </component>
      </components>
    </package>
  </packages>
</dotnetnuke>
```


Writing a Mobile Application - Example

- 'DnnDash' application on Windows Phone
- Reads installed DotNetNuke dashboard components for display on mobile devices
- Uses DnnDash services layer to send back Xml of dashboard information
- Host-level access required to run
- DnnDash module/phone app available from dnndash.com
- http://dnndashservice.codeplex.com/ for source code

Accessing Services from Windows Phone

- Uses System.Net.HttpWebRequest
  - » Asynchronous method
  - » Check Http status code to monitor correct authentication (401 returned when not authenticated)
- Each call from the device provides user authentication username/password pair
- Uses Xml from Service call to create UI

DNN Dash Phone App Demo

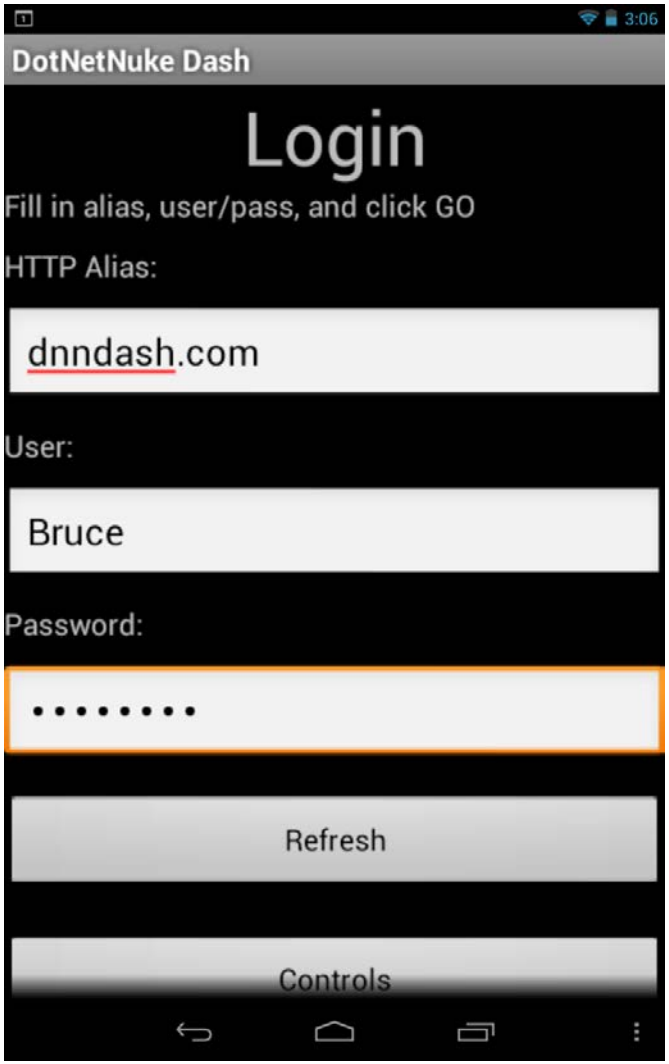
- https://play.google.com/store/apps/details?id=com.rm.dnndash&feature=search_result
- Search 'DnnDash' on the Google store

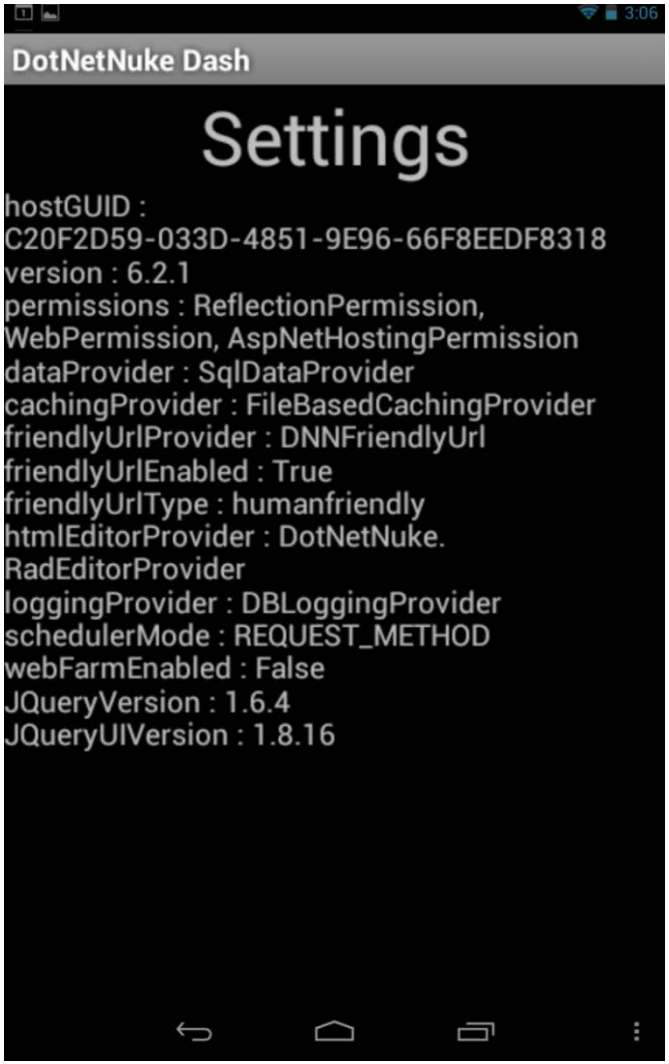
- Windows Phone 7 version still in Approval, but should be on the Windows Store soon

- Connect to dnndash.com
- U : mobiledemo / P : dnn2012

DNN Dash Phone App Demo

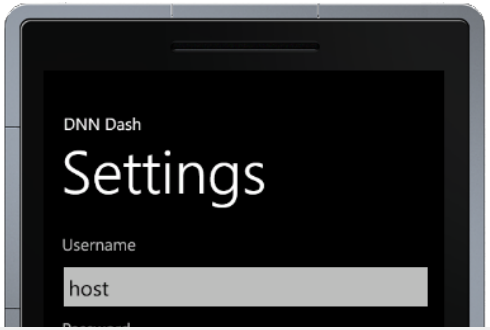
Android Version Screenshots

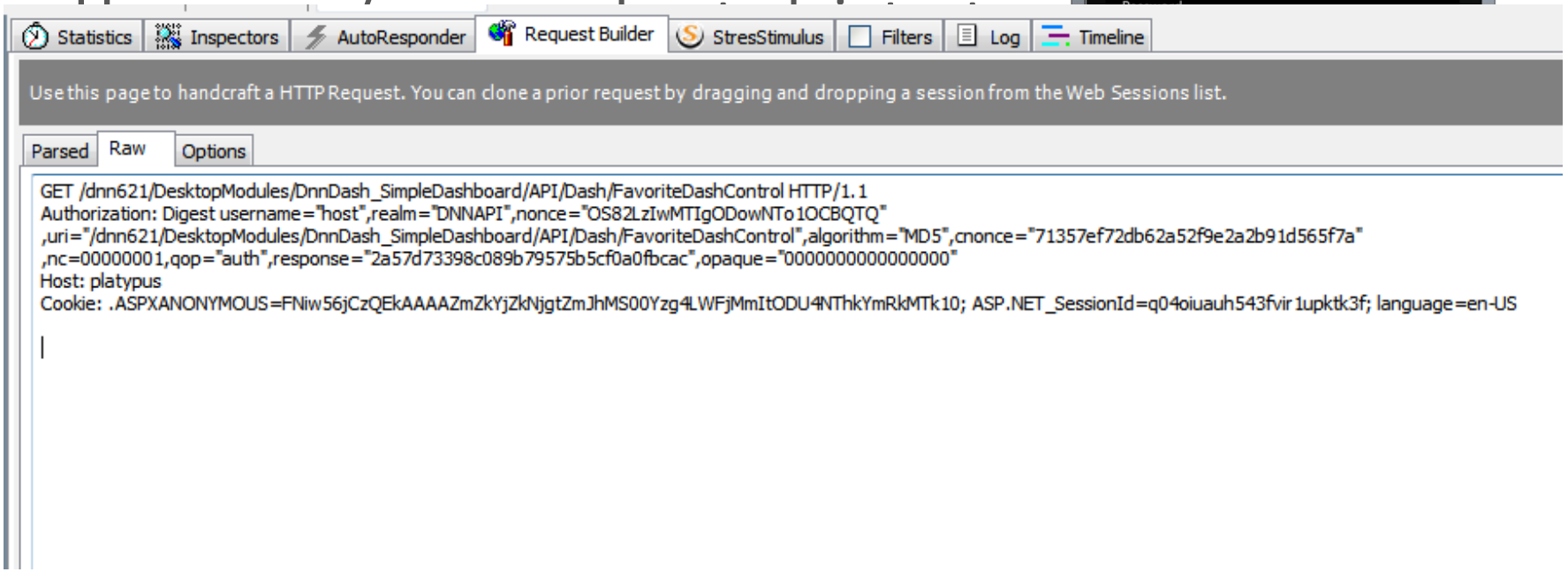




Authenticating Phone User

- Username/Password supplied with each call
- Defaults to Digest Authentication –





Requesting Data from Service

```
private void HandleResponse(IAsyncResult asyncResult)
{
    try
    {
        // get the state information
        RequestResponseState asyncState = (RequestResponseState)asyncResult.AsyncState;
        HttpWebRequest dashboardRequest = (HttpWebRequest)asyncState.AsyncRequest;

        // end the async request
        asyncState.AsyncResponse = (HttpWebResponse)dashboardRequest.EndGetResponse(asyncResult);

        // get the response stream
        Stream streamResult = asyncState.AsyncResponse.GetResponseStream();

        // if OK, load the data
        dataStatusCode = asyncState.AsyncResponse.StatusCode;
        if (dataStatusCode == HttpStatusCode.OK)
        {
            XElement xmlResult = XElement.Load(streamResult);
            PopulateControls(xmlResult);

            // time stamp this Response
            LastResponse = DateTime.Now;
        }
        else
        {
            // something is not right
            ShowErrorMessage(string.Format("Error in API Call \n\n Status Code : {0} \n\n Error : {1}", dataStatusCode, "Response Not OK"));
        }
    }
    catch (System.Net.WebException ex)
    {
        HttpWebResponse errResponse = (HttpWebResponse)ex.Response;
        dataStatusCode = errResponse.StatusCode;
        ShowErrorMessage(ex.Message);
    }
```

Posting Data to a service

- Posting is much the same as retrieving data
- Post generally means changing server state
- Posts should be idempotent
- Failure should allow for re-submit without losing client state

Considerations for Mobile App Design

- Store-posted apps can have lead-times for changes in design
- DotNetNuke modules can be changed + patched immediately
- Therefore, introduce flexibility into design that is data-driven, side-stepping requirements to re-release mobile clients

Conclusions

- Mobile Internet usage will soon overtake fixed line internet, if it hasn't already
- Hardware sales growth in mobile devices outstrips desktop computing
- DotNetNuke provides a perfect bridge between mobile devices and your data, especially if you want to put it in the cloud
- Building modules for the services layer is simple and fast
- Mobile Apps are the next big thing in DotNetNuke

QUESTIONS

- Dnn Dash application source code available from http://dnndashservice.codeplex.com/
  - » Dnn Service Layer
  - » Dnn Dash Desktop App
  - » Dnn Dash Windows Phone App
  - » Android Phone App

- Slides available at http://www.slideshare.net/brchapman

- Follow me on twitter : @brucerchapman