WordPress for .NET (WP4.NET)

Purpose:

To construct a .NET Provider for content extracted from any WordPress Content Management System using WP REST API V2, and compatible plugins.

Design Theory:

Design one process and leverage as many reusable methods as possible and handle exceptions on an individual basis, while limiting outbound calls.
Use auto deserialization by Newtonsoft into structured classes for easy mapping.
Build custom extensions to allow for safe null checking and safe content extraction from buried model structures.
Menus are the only programmatically decorated HTML elements, and are based on WP menu standards and theme add-ons
Page scraping will need to occur for specific WP themes/content types for CSS Styles, Javascript, and Design Styles.

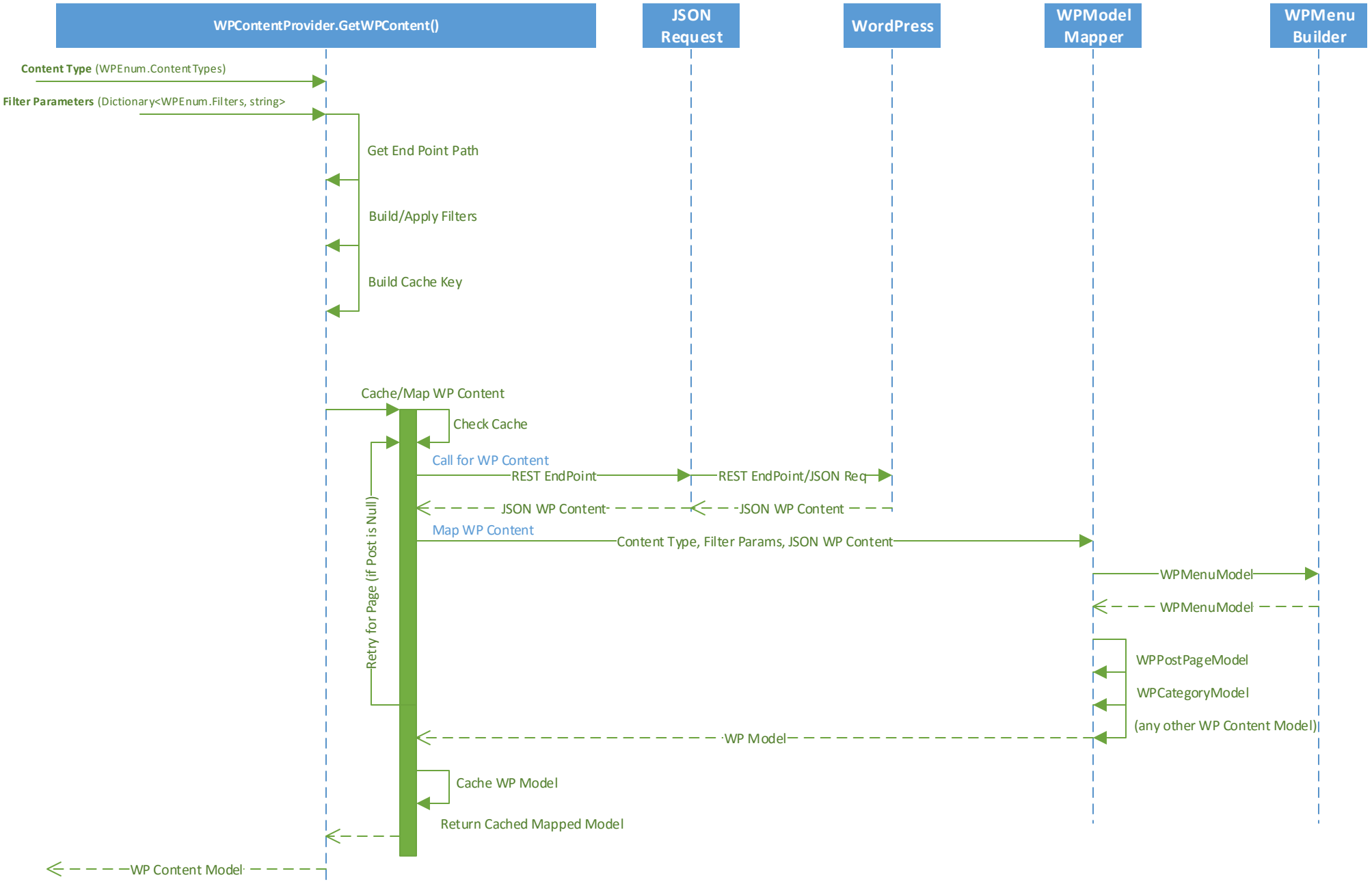
WP4.NET Artifacts:

- Sequence Diagram

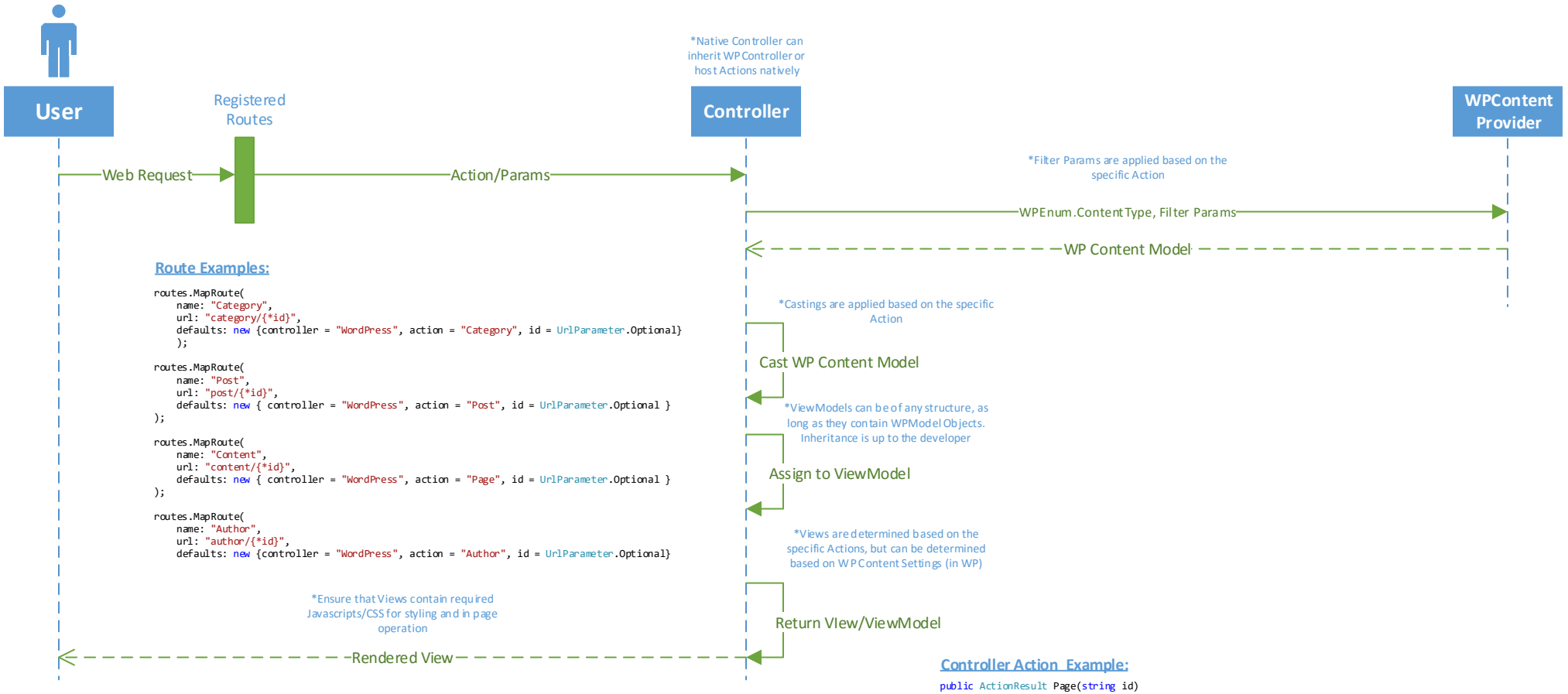
- Implementation for .NET MVC

- Notes

Sequence Diagram: WordPress Content Provider for .NET (WP4.NET)



Implementation for .NET MVC (WP4.NET)



Route Examples:

```
routes.MapRoute(
    name: "Category",
    url: "category/{*id}",
    defaults: new {controller = "WordPress", action = "Category", id = UrlParameter.Optional}
    );

routes.MapRoute(
    name: "Post",
    url: "post/{*id}",
    defaults: new { controller = "WordPress", action = "Post", id = UrlParameter.Optional }
);

routes.MapRoute(
    name: "Content",
    url: "content/{*id}",
    defaults: new { controller = "WordPress", action = "Page", id = UrlParameter.Optional }
);

routes.MapRoute(
    name: "Author",
    url: "author/{*id}",
    defaults: new {controller = "WordPress", action = "Author", id = UrlParameter.Optional}
```

View Example:

```
@using WordPress.Content.Models
@using WordPress.Content.ViewModels
@model WordPress.Content.ViewModels.WPContentViewModel

@{
    var contentType = WPEnums.ContentTypes.PAGE;
    if (ViewBag.Title == null)
    {
        ViewBag.Title = @Html.Raw(Model.GetSafeTitle(contentType));
    }
}

                    @Html.Raw(Model.PageModel.GetSafeContent())
```

*WP4.NET includes custom extensions for checking for and obtaining view model objects safely

Controller Action Example:

```
public ActionResult Page(string id)
{
    contentViewModel.PageModel =

        //detect whether the id is an int or a string
        Int32.TryParse(id, out idNumberValue)
        ? (WPPostPageModel)

            //if the id is an int, apply the "include" filter
            _wpContentSvc.GetWPContent(WPEnums.ContentTypes.PAGE, new Dictionary<WPEnums.Filters, string>()
            {
                {WPEnums.Filters.include, WPEnums.Filters.include.Apply(idNumberValue.ToString())}
            })
        : (WPPostPageModel)

            //if the id is a string, apply the "slug" filter
            _wpContentSvc.GetWPContent(WPEnums.ContentTypes.PAGE,
            new Dictionary<WPEnums.Filters, string>()
            {
                {WPEnums.Filters.slug, WPEnums.Filters.slug.Apply(id)}
            });

    //set the viewBag title for the home page, otherwise, the viewBag title is set on the View
    if (id.Contains("home"))
    {
        ViewBag.Title = "WordPress 4 .NET";
    }

    return View("~/Views/Content/WPPage.cshtml", contentViewModel);
}
```

Required AppSettings (Wed.config):

```
<!--WordPress Settings-->
<add key="TopNavMenuID" value=""/>
<add key="FootNavMenuID" value=""/>
<add key="WPEndPoint" value="http://{yourwpsite.com}"/>
<add key="UseMenuBuilder" value="false"/>
<add key="UseMVCMenuDecoration" value="true"/>
<add key="CacheExpireMins-MENU" value="1440"/>
<add key="CacheExpireMins-POST" value="1440"/>
<add key="CacheExpireMins-PAGE" value="1440"/>
<add key="CacheExpireMins-POSTCATEGORY" value="1440"/>
<add key="CacheExpireMins-MENUCOLLECTION" value="1440"/>
<!--Mock Request Settings -->
<add key="MockRequest" value="true"/>
<add key="MockPostLocation" value="C:\MockResponse\Post.txt"/>
<add key="MockPageLocation" value="C:\MockResponse\Page.txt"/>
<add key="MockCategoryLocation" value="C:\MockResponse\
Category.txt"/>
<add key="MockPostCategoryLocation" value="C:\MockResponse\
PostCategories.txt"/>
<add key="MockTopMenuLocation" value="C:\MockResponse\
TopNavMenu.txt"/>
<add key="MockFootMenuLocation" value="C:\MockResponse\
FootNavMenu.txt"/>
<add key="MockMenuCollection" value="C:\MockResponse\
MenuCollection.txt"/>
<!-- Web Proxy Settings (if needed) -->
<add key="proxyAddress" value=""/>
<add key="proxyDomain" value=""/>
<add key="proxyUsername" value=""/>
<add key="proxyPassword" value=""/>
<add key="proxyTimeout" value="20000"/>
```

Other Notes:

Current Gaps:

- Terms: not retrieving terms; requirement will be based on theme and theme usage

- Comments: not retrieving/mapping/displaying comments

- Extended plugins not included: sliders, widgets, sidebars, etc.

- Only READ operations are configured; all other CRUD operations have not been included in this project

Application Dependencies:

.NET Framework 4.5 and newer
NewtonSoft 9.0 or newer

Required WP Plugins:

WP REST API (Version 2)
https://wordpress.org/plugins/rest-api/

WP API Menus
https://wordpress.org/plugins/wp-api-menus/

WP REST API Documentation:
http://v2.wp-api.org/

**If WP REST API V2 is installed, reference can be found at the WP-JSON root:
http://{yourSite}/wp-json

Caching:

- .NET memory cache is currently being used, though other caching solutions can be used instead.

- An alternative to populating the memory cache with the entire JSON response, is to physically cache the JSON response as a flat file, then consume the flat file response, while only caching the JSON flat file path and still being able to use expiration mins

3rd Party Libraries:

- Dependency injection was not used to avoid conflicting frameworks

- Newtonsoft is the only 3rd party library used