Web API (REST)

Table of contents

General

This is an extension on C# standards.

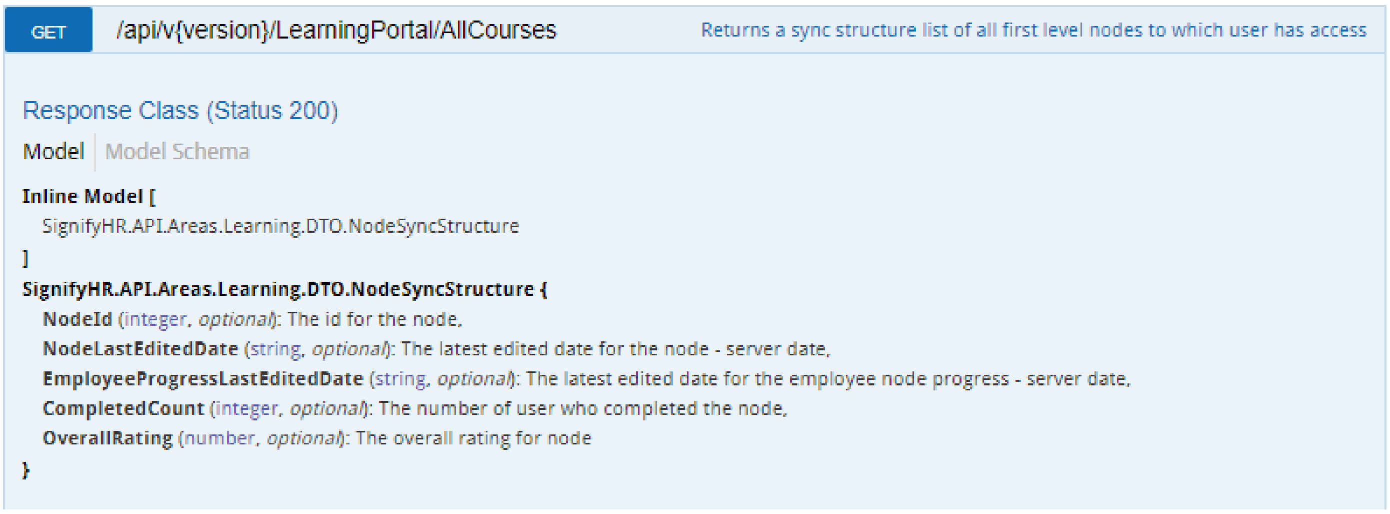
XML Documentation Commenting

Code Documenting

Ensure all publicly exposed actions, objects and properties are described using XML Documentation Comments as this is used to instruct third party users more about what an action does, what an object represents and what a property entails.

```
/// <summary>
/// Returns a sync structure list of all first level nodes to which user has access
/// </summary>
/// <returns></returns>
[ApiRoute("LearningPortal/AllCourses", AcceptedVersions = new[] { 1, 2, 3 })]
public IEnumerable<NodeSyncStructure> GetNodes()
{
    try
    {
        return NodeSyncStructure.FetchAll(SessionHandler, false);
    }
    catch (Exception ex)
    {
        throw LogException(HttpStatusCode.BadRequest, Request, ex);
    }
}
```

```
/// <summary>
/// Data Tranfer Object to transfer node comparison information from and to the API
/// </summary>
public class NodeSyncStructure
{
    /// <summary>
    /// The id for the node
    /// </summary>
    public int NodeId { get; set; }
    /// <summary>
    /// The latest edited date for the node - server date
    /// </summary>
    public DateTime NodeLastEditedDate { get; set; }
    /// <summary>
    /// The latest edited date for the employee node progress - server date
    /// </summary>
    public DateTime EmployeeProgressLastEditedDate { get; set; }
    /// <summary>
    /// The number of user who completed the node
    /// </summary>
    public int CompletedCount { get; set; }
}
```



GET /api/v{version}/LearningPortal/AllCourses Returns a sync structure list of all first level nodes to which user has access

Response Class (Status 200)

Model | Model Schema

```
Inline Model [
  SignifyHR.API.Areas.Learning.DTO.NodeSyncStructure
]
SignifyHR.API.Areas.Learning.DTO.NodeSyncStructure {
  NodeId (integer, optional): The id for the node,
  NodeLastEditedDate (string, optional): The latest edited date for the node - server date,
  EmployeeProgressLastEditedDate (string, optional): The latest edited date for the employee node progress - server date,
  CompletedCount (integer, optional): The number of user who completed the node,
  OverallRating (number, optional): The overall rating for node
}
```

Naming conventions

URI's

API URI's are based around resources. Actions are indicated with the use of HTTP Methods.

- Use nouns, not verbs
- Resources that are collections should be indicated using plural nouns
  e.g. `/pathways`
- Specific resource in collection is defined after collection (generally with *id*)
  e.g. `/pathways/{id}`
- Resources of which only one instance can exist should be indicated using singular noun
  e.g. `/pathways/{id}/favourite`
- Related/Linked resources are defined after specific resource (recursive - refer to above)
  e.g. `/pathways/{nodeId}/steps`, `/pathways/{nodeId}/steps/{stepId}`,
  `/pathways/{nodeId}/steps/{stepId}/progress`

Microsoft Documentation: Organize the API around resources.

Code

Decorate actions and controllers according to resource path and applicable version(s).

Controllers

```
[ApiRoute("TrainingRequirements")]
public class TrainingRequirementsController : ApiBaseController
{
    //code logic
}
```

Actions

```
[ApiRoute("LearningPortal/Pathways/{nodeId:int}")]
public NodeStructure GetPathway(int nodeId)
{
    //code logic
}
```

```
[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Favourite")]
public bool GetFavourite(int nodeId)
{
    //code logic
}

[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Steps/{stepId:int}", AcceptedVersions = new[] { 1, 2 })]
public StepStructure GetPathwayStep(int nodeId, int stepId)
{
    //code logic
}

[ApiRoute("LearningPortal/Pathways/{nodeId:int}/Steps/{stepId:int}", StartingVersion = 3)]
public StepStructureV3 GetPathwayStepV3(int nodeId, int stepId)
{
    //code logic
}
```

Methods

Related Documentation

Define operations in terms of HTTP methods

Conform to HTTP semantics

GET methods

Action: Retrieve resource(s).
HTTP Status Code(s): 200 (OK), 404 (Not Found)

POST methods

Action: Create resource(s) (can also be used for updates in some cases).
HTTP Status Code(s): 201 (Created), 204 (No Content), 400 (Bad Request), 200 (OK)

PUT methods

Action: Update resource(s).
HTTP Status Code(s): 200 (OK), 204 (No Content), 404 (Not Found), 409 (Conflict)

DELETE methods

Action: Delete/Remove resource(s).
HTTP Status Code(s): 204 (No Content), 404 (Not Found)

Versioning

URI versioning is utilised. Refer to below links for more information:

Versioning a RESTful web API

API Version Control

Service Versioning

Object Usage

Data Transfer Object - DTO

Data Transfer Object is the object returned by an API call.
A DTO is used to prevent exposing your database entity structure. This also prevent the exposing of data not being or to be used by the API client, such as linked entities. Data can also be returned as human readable if an API call is consumed by an UI client.

```
/// <summary>
/// Data Tranfer Object to transfer employee node rating information from and to the API
/// </summary>
public class EmployeeNodeRating
{
    /// <summary>
    /// The id of the rating
    /// </summary>
    public int Id { get; set; }
    /// <summary>
    /// The node id for pathway rated
    /// </summary>
    public int NodeId { get; set; }
    /// <summary>
    /// The rating value specified
    /// </summary>
    public int Rating { get; set; }
    /// <summary>
    /// The comment specified for ranking
    /// </summary>
    public string Comment { get; set; }
    /// <summary>
    /// The employee name for who rated node
    /// </summary>
    public string EmployeeName { get; set; }
    /// <summary>
    /// The rating date for ranking
    /// </summary>
    public string Date { get; set; }
    /// <summary>
    /// The image url for user rating
    /// </summary>
    public string ImageUrl { get; set; }
    /// <summary>
    /// The flag to indicate if comment is own comment
    /// </summary>
    public bool IsOwnComment { get; set; }

    /// <summary>
    /// Fetch the node rating details
    /// </summary>
    /// <param name="sessionHandler"></param>
    /// <param name="nodeId"></param>
    /// <returns></returns>
    public static EmployeeNodeRating Fetch(ISessionHandler sessionHandler, int nodeId)
    {
        var eagerLoadParms = new pwEmployeeRating.EagerLoadParameters
        {
            IncludeEmployees = true,
        };

        var empRating = pwEmployeeRating.TryFetchByNodeAndEmployee(sessionHandler, nodeId,
sessionHandler.EmployeeId.Value, eagerLoadParms);

        if (empRating == null)
            return null;
        else
            return new EmployeeNodeRating
            {
                Id = empRating.Id,
                NodeId = empRating.NodeId,
                EmployeeName = empRating.prsEmployee.NameSurname,
                Comment = empRating.Comment,
                Date = empRating.CreatedDate.ToShortDateString(),
                ImageUrl = String.Format("~/api/thumbnail/AspectRatioEmployee?id={0}&width=null&height=96",
empRating.prsEmployee.EmployeeNumber),
                IsOwnComment = true,
                Rating = empRating.Rating
            };
    }
}
```

Value/View Object - VO

A Value/View Object is generally the object received by an API call.
This is similar to a DTO in the sense that the underlying database entity is not exposed. A VO generally only contains properties for all the applicable, editable values to be submitted by the client; validation are also performed on these values prior to submitting data to data storage / database.

```
/// <summary>
/// View Object to transfer employee node rating information from the API
/// </summary>
public class EmployeeNodeRating
{
    /// <summary>
    /// The id for node
    /// </summary>
    public int NodeId { get; set; }
    /// <summary>
    /// The rating value specified
    /// </summary>
    public int Rating { get; set; }
    /// <summary>
    /// The comment specified for ranking
    /// </summary>
    public string Comment { get; set; }
    /// <summary>
    /// The flag to exclude comment from updating
    /// </summary>
    public bool IgnoreCommentChange { get; set; }

    /// <summary>
    /// Log the node rating for employee
    /// </summary>
    /// <param name="sessionHandler"></param>
    /// <returns></returns>
    public void LogRating(ISessionHandler sessionHandler)
    {
        var empRating = pwEmployeeRating.TryFetchByNodeAndEmployee(sessionHandler, this.NodeId,
sessionHandler.EmployeeId.Value);
        if (empRating == null)
        {
            empRating = new pwEmployeeRating
            {
                NodeId = this.NodeId,
                EmployeeId = sessionHandler.EmployeeId.Value,
                Rating = this.Rating,
                Comment = this.IgnoreCommentChange ? String.Empty : this.Comment.Trim(),
            };

            empRating.Create(sessionHandler);
        }
        else
        {
            empRating.Rating = this.Rating;
            empRating.Comment = this.IgnoreCommentChange ? empRating.Comment : this.Comment.Trim();
            empRating.Update(sessionHandler);
        }
    }
}
```