**DPS907 WSA500 Assignment 4**

Hypermedia representations.

Read/skim all of this document before you begin work.

This document is being edited.  
Some “how to” content for the code-generated section must be added.  
This notice will be removed when the edits are complete.

**Due date**

Sunday, October 28, 2018, at 11:00pm ET

Grade value: 10% of your final course grade

*If you wish to submit the assignment before the due date and time, you can do that.*

**Overview and purpose**

After learning about REST constraints and hypermedia-driven design, you need some experience in implementing the concepts and techniques. Your work will include 1) manually-generated controls, and 2) code-generated controls (using an enhanced media type formatter).

**Specifications overview and work plan**

Here’s a brief list of specifications that you must implement:

* Follows best practices
* Implements the recommended system design guidance
* Customized appearance on the landing web page
* Uses Entity Framework and Code First technology
* Includes a Fiddler log file that shows complete coverage of tests
* Fully-implemented documentation components (code comments, XML code comments, annotations, Web API Help Page, entry/root URI)
* Manually-generated hypermedia representation for the Invoice entity, *get all* and *get one*
* Code-generated hypermedia representation for the Track entity, *get some* (for a specific album)

Students in DPS907 must complete these additional specifications (use cases):

* Deliver hypermedia representations from PUT (command) and DELETE requests

During the class/session, your professor will help you *get started* and *make progress* on this assignment.

Every week, in the computer-lab class/session, it is possible that your professor will record a grade when you complete a specific small portion of the assignment. We call this “*in-class grading*“. The *in-class grading* will be announced in-class by your professor.

**Getting started**

Create a new web service, named Assignment4. It must use the “Web service project v1” project template.

Build/compile, and run (without debugging), to ensure that the app’s home > index view loads in a browser. Then, customize the index view (with your name etc.) before continuing.

**Add the recently-learned code assets**

Recently, you learned about several code assets that add value to a web service. Locate these assets, and add them to your project. Yes, you have permission to add, activate, modify (where necessary), and use these code assets in your assignment work.

* HandleError.cs – error handling
* CreateErrorController.cs – to enable you to test HandleError.cs
* ByteFormatter.cs – handle non-text internet media types
* HRFormatterICT.cs – from the Week 6 folder – classes that support hypermedia representations

Make sure that you *activate* or *configure* each one, as you have recently learned. Build/compile and run, to ensure that it still works without errors.

Now, continue by working on the specifications (use cases).

**Fully-implemented documentation components**

The goal of this task is to fully implement a range of documentation components (code comments, XML code comments, annotations, Web API Help Page, entry/root URI).

For all components, you will have to iteratively work on them, as you work on the other specifications (use cases). In other words, it is unreasonable to expect that you’ll be able to fully complete this task before writing code for controllers and resource models.

Please note:  
Do this work now, before starting work on the other specifications (use cases). Do not leave this section of work until after, because the work done later depends on the this section’s work.

Add and configure a controller to handle requests to the entry point.

Which entity (or entities) should you focus on? In this assignment, we will be working with the *Invoice* and *Track* entities. Plan on making (eventually) controllers for Invoice, Track, and Album. That fact suggests that we will need resource model classes for those entities too. In addition, there will be properties from other associated entities that we’ll work with too.

Enable the Web API Help Page. As you have learned, a number of tasks are required.

**Manually-generated hypermedia representation for the Invoice entity**

In this section, we will support *get all* and *get one*, for the Invoice entity. The hypermedia representation will be manually-generated.

Please note:  
There are many invoices. Maybe your manager method for get all should take and deliver only some (maybe 10 or 20). That will make it easier to test and configure.

Create a controller to handle *Invoice* requests. As noted above, it will need methods to handle only the *get all* and *get one* requests. Obviously, this task requires resource model classes to be created, and manager methods.

The resource model class for the Invoice entity must include these specific additional properties from associated objects:

* Number (count) of invoice line items
* From the associated customer… company name, first name, and last name

The delivery of hypermedia representations (HR) will be done by writing code in the controller, as you learned in the Week 5 code example. The *get all* HR will include a link to self, and each item in the collection will include the recommended links. The *get one* HR will include the recommended links to self and to the collection.

As you make progress on this task, use both Fiddler and the Web API Help Page to test your work.

**Code-generated hypermedia representation for the Track entity**

In this section, we will support *get some*, for the *Track* entity. Some? We will actually do this by working with an *Album* controller, and attribute routing. The hypermedia representation will be code-generated, after you add new code to the *HRFormatterICT* class.

Why are we working with the *Album* controller to do *Track*-related requests? So that we can support URI segments like the following:

* api/albums/{id}/tracks
* api/albums/{id}/addtrack
* api/albums (to support POST)
* api/albums/{id} (to support PUT and DELETE requests)

More to come…   
Some “how to” content for the code-generated section must be added.

**Testing your work**

Use Fiddler.

Test all scenarios (use cases).

This time, add some requests that you know will cause HTTP response errors (400s and/or 500s), so that we know that your app is responding with the correct status code etc.

Again, remember the tip from the notes:  
Create a plain text file in your project root to store entity bodies that can be used in copy/paste.

**Saving – “exporting” – your tests**

On the left side list of requests, you can delete items that you do not want included in the export.

When you’re ready to save, choose File > Export Sessions > All Sessions…

The export format will be “HTTPArchive v1.2”. Click the Next button to choose a save location (your project’s root, in the same folder level as the “packages” and “assign1” folder) and specify a filename. Name the file by using the project name (e.g. “*whatever*.har”).

(You can test whether the export was successful. How? First, close then re-open Fiddler. Choose File > Import Sessions. Select “HTTPArchive” as the import format. Navigate to the folder that holds the “.har” file, and select it. Finally, browse through the request-response sessions.)

**Reminder about academic honesty**

You must comply with the College’s academic honesty policy.

Although you may interact and collaborate with others, *you must submit your own work*.

When you are ready to submit your work, you will copy some of the code in your project to plain text files, so that the My.Seneca/Blackboard “SafeAssign” tool can do its job. The next section will tell you which files to copy.

From the [Blackboard web site](https://www.blackboard.com/safeassign/index.html):  
SafeAssign compares submitted assignments against a set of academic papers to identify areas of overlap between the submitted assignment and existing works.

**Submitting your work**

Here’s how to submit your work, before the due date and time:

1. Locate the folder that holds your solution files. In Solution Explorer, right-click the “Solution” item, and choose “Open Folder in File Explorer”. It has three  (or more) items: a Visual Studio Solution file, a folder that has your project’s source code, and a “packages” folder. Go UP one level.

2. Make a copy of the folder. (You will be uploading a zipped version of the copy.)

3. Add a new folder named “MyCode” to the root of the copied folder.

4. Remove the “packages” folder from the copied folder; also, remove the “bin” and “obj” folders.

5. Copy these source code files from your project to the “MyCode” folder:

* Manager.cs
* HRFormatterICT.cs
* Invoice\_vm.cs
* InvoicesController.cs
* Track\_vm.cs
* Album\_vm.cs
* AlbumsController.cs

For each of those files, change the “cs” file name extension to “txt”.

6. Compress/zip the copied folder. The zip file size SHOULD probably be less than 10MB (and definitely less than 50MB). If it isn’t, you haven’t followed the instructions properly.

7. Login to My.Seneca/Blackboard. Open the Web Services course area. Click the “Assignments” link on the left-side navigator. Follow the link for this assignment. Submit/upload your zip file. The page will accept three submissions, so if you upload, then decide to fix something and upload again, you can do so.