**Purpose**

In this lab, you will build a Time Entry service which will expose a [RESTful](https://en.wikipedia.org/wiki/Representational_state_transfer) API for time entries. The lab will introduce the fundamentals of ASP.NET Core for building web services.

**Discussion points**

* ASP.NET Core overview
* [ApiController vs Controller](https://stackify.com/asp-net-core-web-api-guide/)
* [Controller vs ControllerBase](https://docs.microsoft.com/en-us/aspnet/core/web-api)
* .NET Core [Dependency Injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection)
* Service Lifetimes (Transient, Scoped, Singleton)
* Discussion of curl/postman/etc.

**Get started**

Before starting the lab, pull in some failing tests using Git:

git cherry-pick mvc-start

The tests we pull in require *Moq*, so add the following dependency to the *PalTrackerTests* project.

dotnet add **test**/PalTrackerTests package Moq --version 4.10.0

Our goal is to get our test suite passing by the end of the lab.

**Time Entries CRUD**

You will build a service that can do CRUD operations on time entries by first defining its API. Begin by implementing create, read, update and delete actions on time entries and persisting them with a repository. At first, the repository will be backed by a Dictionary. It will be replaced by a database in a later lab.

**Data layer**

1. Create a TimeEntry **struct**:

[Hide TimeEntry.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/rest-endpoints/index.html" \l "pal-tracker353723b4-e134-46ee-88a6-bc853a41a1f8)

pal-tracker/src/PalTracker/TimeEntry.cs

**using** System;

**namespace** **PalTracker**

{

**public** **struct** TimeEntry

{

**public** **long**? Id { **get**; **set**; }

**public** **long** ProjectId { **get**; **set**; }

**public** **long** UserId { **get**; **set**; }

**public** DateTime Date { **get**; **set**; }

**public** **int** Hours { **get**; **set**; }

**public** **TimeEntry**(**long** id, **long** projectId, **long** userId, DateTime date, **int** hours)

{

Id = id;

ProjectId = projectId;

UserId = userId;

Date = date;

Hours = hours;

}

**public** **TimeEntry**(**long** projectId, **long** userId, DateTime date, **int** hours)

{

Id = null;

ProjectId = projectId;

UserId = userId;

Date = date;

Hours = hours;

}

}

}

1. Create the following interface:

[Hide ITimeEntryRepository.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/rest-endpoints/index.html" \l "pal-tracker916930f4-c609-4b09-8809-0d4a8cc63d8c)

pal-tracker/src/PalTracker/ITimeEntryRepository.cs

**using** System.Collections.Generic;

**namespace** **PalTracker**

{

**public** **interface** **ITimeEntryRepository**

{

TimeEntry **Create**(TimeEntry timeEntry);

TimeEntry **Find**(**long** id);

**bool** **Contains**(**long** id);

IEnumerable<TimeEntry> **List**();

TimeEntry **Update**(**long** id, TimeEntry timeEntry);

**void** **Delete**(**long** id);

}

}

1. The tests that were cherry-picked reference a class that does not exist. This means the test project does not yet compile.

Stub out TimeEntryController using throw new NotImplementedException() to fix the compiler errors. We will provide the implementation in a later step.

1. Implement the **ITimeEntryRepository** interface with a class called **InMemoryTimeEntryRepository**. Use the **InMemoryTimeEntryRepositoryTest** to guide your implementation.

You can [run specific tests](https://docs.microsoft.com/en-us/dotnet/core/testing/selective-unit-tests#xunit) by executing the following command from the **test/PalTrackerTest** directory.

dotnet **test** **test**/PalTrackerTests --filter PalTrackerTests.InMemoryTimeEntryRepositoryTest

Take a look at our solution if you need help:

[Hide InMemoryTimeEntryRepository.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/rest-endpoints/index.html" \l "pal-tracker1ef1d09f-a34b-42c8-873b-f82b5ecaf10a)

pal-tracker/src/PalTracker/InMemoryTimeEntryRepository.cs

**using** System.Collections.Generic;

**using** System.Linq;

**namespace** **PalTracker**

{

**public** **class** **InMemoryTimeEntryRepository** : **ITimeEntryRepository**

{

**private** **readonly** IDictionary<**long**, TimeEntry> \_timeEntries = **new** Dictionary<**long**, TimeEntry>();

**public** TimeEntry **Create**(TimeEntry timeEntry)

{

**var** id = \_timeEntries.Count + 1;

timeEntry.Id = id;

\_timeEntries.Add(id, timeEntry);

**return** timeEntry;

}

**public** TimeEntry **Find**(**long** id) => \_timeEntries[id];

**public** **bool** **Contains**(**long** id) => \_timeEntries.ContainsKey(id);

**public** IEnumerable<TimeEntry> **List**() => \_timeEntries.Values.ToList();

**public** TimeEntry **Update**(**long** id, TimeEntry timeEntry)

{

timeEntry.Id = id;

\_timeEntries[id] = timeEntry;

**return** timeEntry;

}

**public** **void** **Delete**(**long** id)

{

\_timeEntries.Remove(id);

}

}

}

1. Using the **ConfigureServices** method of the **Startup** class, register the new repository with the dependency injection container. Be sure to use the [**Singleton** lifetime](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection#service-lifetimes-and-registration-options). Take a look at our solution if you need a hand.

[Hide Startup.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/rest-endpoints/index.html" \l "pal-tracker6efc3e3c-a2c1-4286-b0a0-163309f7f0df)

pal-tracker/src/PalTracker/Startup.cs

﻿**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Threading.Tasks;

**using** Microsoft.AspNetCore.Builder;

**using** Microsoft.AspNetCore.Hosting;

**using** Microsoft.AspNetCore.HttpsPolicy;

**using** Microsoft.AspNetCore.Mvc;

**using** Microsoft.Extensions.Configuration;

**using** Microsoft.Extensions.DependencyInjection;

**using** Microsoft.Extensions.Logging;

**using** Microsoft.Extensions.Options;

**namespace** **PalTracker**

{

**public** **class** **Startup**

{

**public** **Startup**(IConfiguration configuration)

{

Configuration = configuration;

}

**public** IConfiguration Configuration { **get**; }

*// This method gets called by the runtime. Use this method to add services to the container.*

**public** **void** **ConfigureServices**(IServiceCollection services)

{

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

services.AddSingleton(sp => **new** WelcomeMessage(

Configuration.GetValue<**string**>("WELCOME\_MESSAGE", "WELCOME\_MESSAGE not configured.")

));

services.AddSingleton(sp => **new** CloudFoundryInfo(

Configuration.GetValue<**string**>("PORT"),

Configuration.GetValue<**string**>("MEMORY\_LIMIT"),

Configuration.GetValue<**string**>("CF\_INSTANCE\_INDEX"),

Configuration.GetValue<**string**>("CF\_INSTANCE\_ADDR")

));

services.AddSingleton<ITimeEntryRepository, InMemoryTimeEntryRepository>();

}

*// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.*

**public** **void** **Configure**(IApplicationBuilder app, IHostingEnvironment env)

{

**if** (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

**else**

{

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseMvc();

}

}

}

**REST controller**

1. Create a **TimeEntryController** class that inherits from **Microsoft.AspNetCore.Mvc.ControllerBase** and uses [Constructor Injection](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection#constructor-injection) to inject the **ITimeEntryRepository** dependency.
2. Create [actions](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions#defining-actions) that correspond to CRUD operations. Be sure to use the [ASP.NET Core attribute](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing#attribute-routing-with-httpverb-attributes) appropriate to the REST operation. For example, use **[HttpGet]** for reads.

Action methods must return the appropriate implementation of an **[IActionResult](https://docs.microsoft.com/en-us/dotnet/api/Microsoft.AspNetCore.Mvc.IActionResult)** interface such as **[NotFoundResult](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.notfoundresult)**,[**CreatedResult**](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.createdresult), etc.

Actions that operate on existing time entries (read, update, delete) should confirm that the provided time entry id is valid prior to performing the operation. If the provided id is invalid, the endpoint should return a status of 404 (NOT FOUND).

For endpoints that receive a payload, use the **[FromBody]** attribute to [bind the payload](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding#customize-model-binding-behavior-with-attributes) to the TimeEntry model.

1. Use the **TimeEntryControllerTest** to guide you to build the **TimeEntryController**. Once the tests pass you are done.

Take a look at our solution if you get stuck.

[Hide TimeEntryController.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/rest-endpoints/index.html" \l "pal-trackerff8cdfd6-b28b-4d2f-8f92-c67324498847)

pal-tracker/src/PalTracker/TimeEntryController.cs

**using** Microsoft.AspNetCore.Mvc;

**namespace** **PalTracker**

{

[**Route("/time-entries")**]

**public** **class** **TimeEntryController** : **ControllerBase**

{

**private** **readonly** ITimeEntryRepository \_repository;

**public** **TimeEntryController**(ITimeEntryRepository repository)

{

\_repository = repository;

}

[**HttpPost**]

**public** IActionResult **Create**([FromBody] TimeEntry timeEntry)

{

**var** createdTimeEntry = \_repository.Create(timeEntry);

**return** CreatedAtRoute("GetTimeEntry", **new** {id = createdTimeEntry.Id}, createdTimeEntry);

}

[**HttpGet("{id}", Name = "GetTimeEntry")**]

**public** IActionResult **Read**(**long** id)

{

**return** \_repository.Contains(id) ? (IActionResult) Ok(\_repository.Find(id)) : NotFound();

}

[**HttpGet**]

**public** IActionResult **List**()

{

**return** Ok(\_repository.List());

}

[**HttpPut("{id}")**]

**public** IActionResult **Update**(**long** id, [FromBody] TimeEntry timeEntry)

{

**return** \_repository.Contains(id) ? (IActionResult) Ok(\_repository.Update(id, timeEntry)) : NotFound();

}

[**HttpDelete("{id}")**]

**public** IActionResult **Delete**(**long** id)

{

**if** (!\_repository.Contains(id))

{

**return** NotFound();

}

\_repository.Delete(id);

**return** NoContent();

}

}

}

1. Run the full test suite with:
2. dotnet **test** **test**/PalTrackerTests
3. Once everything is passing, push your changes to Github and watch them run through CI.

**Assignment submission**

Submit the assignment using the **cloudNativeDeveloperRest** gradle task. It requires you to provide the URL of your application. For example:

**cd** ~/workspace/assignment-submission

./gradlew cloudNativeDeveloperRest -PserverUrl=https://[prod-app-url]