# Purpose

In this lab you will implement the database version of the Time Entry Repository using .NET Core Entity Framework.

# Discussion points

* DB Testing: real DB vs InMemory or SQLite
* Steeltoe Connectors

# Get started

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

git cherry-pick database-start

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

# Add the dependencies

1. We will be using the Entity Framework Core backed by a MySQL database for our persistence layer. Add the following packages to the project:
2. dotnet add src/PalTracker package Microsoft.EntityFrameworkCore --version 2.1.4
3. dotnet add src/PalTracker package Pomelo.EntityFrameworkCore.MySql --version 2.1.2
4. On Cloud Foundry, connection information for the bound MySQL database is provided inside the **VCAP\_SERVICES**environment variable. The Connector package helps our application consume this information. Add the Steeltoe.CloudFoundry.Connector.EFCore package.
5. dotnet add src/PalTracker package Steeltoe.CloudFoundry.Connector.EFCore --version 2.1.1

Connectors are part of the [Steeltoe](http://steeltoe.io/) project, which provides a number of client libraries that ease microservice development. We will leverage these tools in later labs.

# Create the repository

1. In the PalTracker project, create a class called **TimeEntryRecord**, which will represent a single time entry row in the database. This abstraction layer allows us to decouple the domain-level representation of a time entry from how it's modeled in the database.

Make sure the class has an empty no-argument constructor. Use Data Annotations to map the class to the **time\_entries**[table](https://docs.microsoft.com/en-us/ef/core/modeling/relational/tables) and map its [fields](https://docs.microsoft.com/en-us/ef/core/modeling/relational/columns) to their respective columns in the database.

[Hide TimeEntryRecord.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/interact-with-database/index.html" \l "pal-tracker87267d76-97ab-4a9d-8eda-b474be5bb1e7)

pal-tracker/src/PalTracker/TimeEntryRecord.cs

**using** System;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** **PalTracker**

{

[**Table("time\_entries")**]

**public** **class** **TimeEntryRecord**

{

[**Column("id")**]

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

[**Column("project\_id")**]

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

[**Column("user\_id")**]

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

[**Column("date")**]

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

[**Column("hours")**]

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

**public** **TimeEntryRecord**()

{

}

}

}

1. In Entity Framework, context is the primary mechanism for interacting with data as objects. Create a class called **TimeEntryContext** that:
   * Derives from **[DbContext](https://docs.microsoft.com/en-us/ef/core/api/microsoft.entityframeworkcore.dbcontext)**.
   * Takes a constructor argument of type **[DbContextOptions](https://docs.microsoft.com/en-us/ef/core/miscellaneous/configuring-dbcontext)**.
   * Has a single property called **TimeEntryRecords** of type **DbSet<TimeEntryRecord>**.

[Hide TimeEntryContext.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/interact-with-database/index.html" \l "pal-tracker96fe4546-1ac0-4ff4-a0f2-f0dc272ebbb7)

pal-tracker/src/PalTracker/TimeEntryContext.cs

**using** Microsoft.EntityFrameworkCore;

**namespace** **PalTracker**

{

**public** **class** **TimeEntryContext** : **DbContext**

{

**public** **TimeEntryContext**(DbContextOptions options) : **base**(options)

{

}

**public** DbSet<TimeEntryRecord> TimeEntryRecords { **get**; **set**; }

}

}

1. Create some mapping extension methods to help us convert between **TimeEntry** and **TimeEntryRecord**:

[Hide MappingExtensions.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/interact-with-database/index.html" \l "pal-trackerff5e8d4b-17dc-4d31-8c8e-1592a5eda6b7)

pal-tracker/src/PalTracker/MappingExtensions.cs

**namespace** **PalTracker**

{

**public** **static** **class** **MappingExtensions**

{

**public** **static** TimeEntry **ToEntity**(**this** TimeEntryRecord record) => **new** TimeEntry

{

Id = record.Id,

ProjectId = record.ProjectId,

UserId = record.UserId,

Date = record.Date,

Hours = record.Hours

};

**public** **static** TimeEntryRecord **ToRecord**(**this** TimeEntry entity) => **new** TimeEntryRecord

{

Id = entity.Id,

ProjectId = entity.ProjectId,

UserId = entity.UserId,

Date = entity.Date,

Hours = entity.Hours

};

}

}

1. Create a new class called **MySqlTimeEntryRepository** that:
   * Implements the **ITimeEntryRepository** interface.
   * Takes **TimeEntryContext** as a constructor argument.

Implement the repository, using **MySqlTimeEntryRepositoryTest** as your guide. Notice that the test intentionally leverages ADO.NET and raw sql rather than the Entity Framework to verify that data is properly persisted to the database.

Hint: You may want to use [No-track queries](https://docs.microsoft.com/en-us/ef/core/querying/tracking) for read-only operations.

Take a look at our solution if you need help:

[Hide MySqlTimeEntryRepository.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/interact-with-database/index.html" \l "pal-tracker5d4a06a3-18d1-4957-85d4-534c83e109d8)

pal-tracker/src/PalTracker/MySqlTimeEntryRepository.cs

**using** System.Collections.Generic;

**using** System.Linq;

**using** Microsoft.EntityFrameworkCore;

**namespace** **PalTracker**

{

**public** **class** **MySqlTimeEntryRepository** : **ITimeEntryRepository**

{

**private** **readonly** TimeEntryContext \_context;

**public** **MySqlTimeEntryRepository**(TimeEntryContext context)

{

\_context = context;

}

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

{

**var** recordToCreate = timeEntry.ToRecord();

\_context.TimeEntryRecords.Add(recordToCreate);

\_context.SaveChanges();

**return** Find(recordToCreate.Id.Value);

}

**public** TimeEntry **Find**(**long** id) => FindRecord(id).ToEntity();

**public** **bool** **Contains**(**long** id) =>

\_context.TimeEntryRecords.AsNoTracking().Any(t => t.Id == id);

**public** IEnumerable<TimeEntry> **List**() =>

\_context.TimeEntryRecords.AsNoTracking().Select(t => t.ToEntity());

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

{

**var** recordToUpdate = timeEntry.ToRecord();

recordToUpdate.Id = id;

\_context.Update(recordToUpdate);

\_context.SaveChanges();

**return** Find(id);

}

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

{

\_context.TimeEntryRecords.Remove(FindRecord(id));

\_context.SaveChanges();

}

**private** TimeEntryRecord **FindRecord**(**long** id) =>

\_context.TimeEntryRecords.AsNoTracking().Single(t => t.Id == id);

}

}

# Configure the service

1. In **Program.cs** add Cloud Foundry as a configuration provider. This allows our application to connect to the MySQL database that is bound to it in Cloud Foundry.
2. + using Steeltoe.Extensions.Configuration.CloudFoundry;
3. namespace PalTracker
4. {
5. public class Program
6. {
7. // ...
8. public static IWebHostBuilder WebHostBuilder(string[] args) =>
9. WebHost.CreateDefaultBuilder(args)
10. + .ConfigureAppConfiguration(config => config.AddCloudFoundry())
11. .UseStartup<Startup>();
12. }
13. }
14. In Startup.cs, register **TimeEntryContext** with the [dependency injection container](https://docs.microsoft.com/en-us/ef/core/miscellaneous/configuring-dbcontext#using-dbcontext-with-dependency-injection), configuring it to use MySQL.
15. + using Steeltoe.CloudFoundry.Connector.MySql.EFCore;
16. namespace PalTracker
17. {
18. public class Startup
19. {
20. // ...
21. public void ConfigureServices(IServiceCollection services)
22. {
23. // ...
24. + services.AddDbContext<TimeEntryContext>(options => options.UseMySql(Configuration));
25. }
26. }
27. }
28. We are now ready to replace the in-memory repository with a "real" implementation. In the **ConfigureServices** method of Startup.cs, replace **InMemoryTimeEntryRepository** with **MySqlTimeEntryRepository**.
29. - services.AddSingleton<ITimeEntryRepository, InMemoryTimeEntryRepository>();
30. + services.AddScoped<ITimeEntryRepository, MySqlTimeEntryRepository>();

Be sure to use the [**Scoped**](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection#service-lifetimes-and-registration-options) instead of **Singleton** lifetime.

This guideline comes from the [Registering services](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.0#registering-services) section of the Dependency Injection documentation.

Take a look at our solution if you need help.

[Hide Startup.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/interact-with-database/index.html" \l "pal-tracker59547090-5358-4c9a-ae8a-038feed2b78b)

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;

**using** Steeltoe.CloudFoundry.Connector.MySql.EFCore;

**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.AddScoped<ITimeEntryRepository, MySqlTimeEntryRepository>();

services.AddDbContext<TimeEntryContext>(options => options.UseMySql(Configuration));

}

*// 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();

}

}

}

1. Add the following to the beginning of TimeEntryIntegrationTest to allow it to connect to the database and clear the database between runs:
2. public TimeEntryIntegrationTest()
3. {
4. + Environment.SetEnvironmentVariable("MYSQL\_\_CLIENT\_\_CONNECTIONSTRING", DbTestSupport.TestDbConnectionString);
5. + DbTestSupport.ExecuteSql("TRUNCATE TABLE time\_entries");
6. // ...
7. }
8. Run the tests to make sure everything still passes.
9. dotnet **test** **test**/PalTrackerTests
10. Point the application at your development database by adding to the end of the appsettings.json file and run the app to confirm it works correctly.
11. "Default": "Warning"
12. }
13. },
14. + "mysql": {
15. + "client": {
16. + "ConnectionString": "Server=localhost;Database=tracker\_dotnet\_dev;Uid=tracker\_dotnet;Pwd=password;"
17. + }
18. + },
19. "AllowedHosts": "\*"
20. }
21. dotnet run --project src/PalTracker

# Update CI

At this point the tests should run successfully on your local machine, but not on CircleCI. This is because the tests assume a database is present. However, the database does not exist yet in the testing container.

1. Update your pipeline to install MySQL and Flyway as well as run the migrations against the local testing database.
2. - checkout
3. + - run:
4. + name: Install MySQL
5. + command: |
6. + apt-get update
7. + apt-get -y install mysql-server
8. + service mysql start
9. +
10. + # Wait for MySQL to start
11. + sleep 5
12. +
13. + - run:
14. + name: Install flyway
15. + command: |
16. + curl https://repo1.maven.org/maven2/org/flywaydb/flyway-commandline/5.1.1/flyway-commandline-5.1.1-linux-x64.tar.gz | tar xvz
17. +
18. + - run:
19. + name: Create test database and migrate
20. + command: |
21. + mysql -uroot < databases/tracker/create\_databases.sql
22. + flyway-\*/flyway -url="jdbc:mysql://localhost:3306/tracker\_dotnet\_test" -locations=filesystem:databases/tracker -user=tracker\_dotnet -password=password clean migrate
23. +
24. - run:
25. name: Test and build
26. Commit and push your code to GitHub, then let your pipeline deploy the new application. Trigger a production deploy once you are confident that everything is working.

# Assignment submission

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

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

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