Here's a suggested project structure for implementing the CQS pattern with Entity Framework Core. This structure assumes a simple application with separate layers for data access, business logic, and presentation.

```

MyApp

│ .gitignore

│ README.md

│ MyApp.sln

│

├───MyApp.Data

│ │ MyApp.Data.csproj

│ │

│ ├───Entities

│ │ Drivers.cs

│ │ // other entity classes

│ │

│ └───Migrations

│ // EF Core migration files

│

├───MyApp.Domain

│ │ MyApp.Domain.csproj

│ │

│ ├───Commands

│ │ CreateDriverCommand.cs

│ │ UpdateDriverCommand.cs

│ │ DeleteDriverCommand.cs

│ │ // other command classes

│ │

│ ├───DTOs

│ │ DriversDto.cs

│ │ // other DTO classes

│ │

│ ├───Handlers

│ │ ├───CommandHandlers

│ │ │ CreateDriverCommandHandler.cs

│ │ │ UpdateDriverCommandHandler.cs

│ │ │ DeleteDriverCommandHandler.cs

│ │ │ // other command handler classes

│ │ │

│ │ └───QueryHandlers

│ │ GetDriverByIdQueryHandler.cs

│ │ GetAllDriversQueryHandler.cs

│ │ // other query handler classes

│ │

│ └───Queries

│ GetDriverByIdQuery.cs

│ GetAllDriversQuery.cs

│ // other query classes

│

└───MyApp.Web

│ MyApp.Web.csproj

│

├───Controllers

│ DriversController.cs

│ // other controller classes

│

├───Views

│ │ // view files

│ │

│ └───Shared

│ \_Layout.cshtml

│ // other shared view files

│

└───wwwroot

│ // static files, such as CSS, JavaScript, images, etc.

```

In this structure:

1. `MyApp.Data`: Contains the data access layer, including the Entity Framework Core DbContext, entity classes, and migrations.

2. `MyApp.Domain`: Contains the business logic layer, including command and query classes, DTO classes, and command/query handlers.

3. `MyApp.Web`: Contains the presentation layer, including controllers, views, and static files.

This structure helps to separate concerns and makes it easier to maintain and test your application.