**Exercise 1 (For all exercises, feel free to look at the Week 4 example app in the class GitHub repository)**

[**X**] Create a new MVC application like you have done before in the previous labs

[**X**] Add the EntityFramework NuGet package to the newly created project.

[**X**] Create a project folder named "Data" and then a folder under "Data" named "Entities".

[**X**] Create a "User" entity under the "Entities" folder with an int property named "Id" that has an attribute of [Key].  
What does the [Key] attribute tell EntityFramework?

The [Key] attribute is used to uniquely identify an entity by the specified property.

[**X**] Add a class under your "Data" folder that inherits from DbContext.  (Look at the example app for help)

[**X**] Add a DbSet collection to the new DbContext class for the "User" entity.   (Look at the example app for help)

[**X**] Configure a connection string for your database in web.config.  (Look at the example app for help)  
What are connection strings used for?

Connection strings are to connect your application to your database.

[**X**] Create actions and corresponding views for listing and creating Users as you did in the last lab.

[**X**] Use your DbContext class to retrieve and add Users in the actions.  (Look at the example app for help)

[**X**] Run the application and make sure it functions as expected.  Add Users and view them in the list.

[**X**] Open "Server Explorer" in Visual Studio and view the "User" table created by the EntityFramework.

[**X**] Add another property to your User entity

String FavoriteColor

[**X**] Rerun the application.  Look for the newly created property in the "User" table using "Server Explorer".

[**X**] The Users you added in step 10 are now gone.  Why did they disappear?

Since we changed properties within our User entity (which is the database representation of the user object), the entities needed to be rebuilt to reflect the changes in the entity itself since there was a new database field (the property we added to the entity). Since the previous users didn’t have the updated entity properties, they had to be flushed to keep the database. Also note that the views “create” and “list” needed to be regenerated as well to reflect the changes.

**Exercise 2**

[**X**] Add a UserViewModel.  Use the view model in place of the User entity model in your User actions and views.  
Why would we use view models rather than the entity models in the controllers and views?

I’m going out on a limb here and I think we would use “viewmodels” rather than the entity models in controllers and views because the views for the “viewmodel” won’t need to be regenerated when changes are made to the properties which will keep the database representation intact and up to date.

[**X**] Feel free to copy the "Map" utility functions from the Week 4 example app.

[**X**] Run the application to make sure it is still functioning as expected.

[**X**] Implement the Details, Delete and Edit functions that are accessible at the end of each User in the User list.  (Look at the example app for help)

**Exercise 3 (Look at the example app for help)**

[**X**] Add another entity to your application that is "related" to the User entity.  Remember to add a DbSet collection for your new entity to your DbContext class.  
  
 **Added Pets class that is related to the User entity.**

[**X**] In the User list view, add another action for viewing a list of the new entity you added.  
  
 **Added a new action link to Views🡪 User 🡪 List to display a link of pets associated with the ID**

[**X**] Create actions and corresponding views for listing and creating your new entity.  Use view models rather then interacting in directly with the new entity model.

[**X**] Run the application:

[**X**] Test to see if you can access the list for your new entity from the User list.

[**X**] Test to see if you can create an instance of your new entity from the list for your new entity.

**Exercise 4 (Extra credit)**

1. Implement the Details, Delete and Edit function for your new entity.