**Preparation**

Exercises will refer to my week 8 GitHub repo which is at [**https://github.com/oit-gaden/Web-Development-2020-Winter/tree/master/Week8**](https://github.com/oit-gaden/Web-Development-2020-Winter/tree/master/Week8)

Copy the contents of your week 7 lab folder to your week 8 lab folder.

**Exercise 1 - Create a repository for students**

1. Create a "**Repositories**" folder under **webapi**.
2. Create a **repository** class and associated interface for the **student** entity you have defined. You can follow the code I created for the **product** repository. You only have to implement the "**get all"** method.  You can also leave out any code related to logging.

**Exercise 2 - Create view model and service for students**

1. Create a ViewModels folder under webapi.
2. Create a StudentViewModel class with the student id and email address from the student entity.  Add a boolean property called **"Special"**.  See the ProductViewModel in my repo for an example.
3. Create a "**Services**" folder under webapi.
4. Create a **service** class and associated interface for the StudentViewModel you defined in step 2. You can follow the code I created for the **product** service. Again, you only have to implement the "**get all"** method. You can leave out any code related to logging.
5. Determine some "**business rule**" for determining when to set the "**special**" property when the "**get all method**" is called. You'll be adding unit tests later for testing the application of this business rule. It doesn't matter how you implement the business rule. Just that you add code for "calculating" the "special" property. You can follow the pattern in my BusinessRules.cs if you want.

**Exercise 3 - Modify student controller**

1. Replace the constructor injection of DbContext with the injection of the student service interface. See my product controller for an example.
2. Retrieve the students using the injected student service instead of the DbContext in the **GetAllStudents**action method. You will now be returning the **StudentViewModel** class rather than the Student entity class. You can leave out any code related to logging.
3. Implement the dependency injection configuration in **Startup.cs**. Add a call to **RegisterApplicationServices** and its implementation where you add the dependency injection configuration for both the student repository and the student service.
4. Test the webapi with Postman as you did in week 7 to make sure it still operates as expected. Put data in the database that will trigger the setting of the "special" property to both true and false;

**Exercise 4 - Add unit tests**

1. Create a folder called **webapi.tests** under the week8 lab folder.
2. Open a cmd/terminal window to the **webapi.tests** folder and run the following command:  
     
   **dotnet new nunit**
3. Run the sample test by running the following command:  
     
   **dotnet test**You should see one test succeed.
4. Install test frameworks by running the following commands:  
     
   **dotnet add package fakeiteasy**  
   **dotnet add package fluentassertions**
5. Reference the **project under test** (webapi) by running the following command:  
     
   **dotnet add reference ..\webapi\webapi.csproj**
6. Start up VSCode and add the new webapi.test project to the workspace.
7. Remove the sample **UnitTest1.cs** file.
8. Add **StudentServiceTests.cs** (see my webapi.tests/ProductServiceTests.cs for an example)Create a mock student repository in the Setup method  
     
   Create an instance of the class under test in the Setup method. In this case it is the StudentService.cs class.Add a test for the scenario where the **"get all"** method returns one or more "special" students.  
     
   Add a test for the scenario where the **"get all"** method returns no "special" students.  
     
   Use **FluentAssertion** or **NUnit Assert** for the test assertions.
9. Run the tests by running the following command:  
     
   **dotnet test**  
     
   You should see two tests pass.
10. Go into the **StudentService**class and change the criteria for determining a special student. Rerun the tests and you should see one test pass and one test fail. Go back and "correct" the **StudentService** class by returning to the original "business rule". Rerun the tests and you should now see both tests passing again.
11. Push your Docker images to Docker Hub and push your code to GitHub.

**Extra Credit (20 points) - Use "special" property in the student list display**

Add code to your webapp to modify the display of students in some way if they are determined to be **"special"** by your business rule from exercise 2.  See my webapp for an example.