Lab 5:
Charlotte's WebAPI

Due at the start of the final exam.

Overview

In this lab, you will extend the "Scheduling" WebAPI demo from lecture to provide an API endpoint to access course sections for a particular semester, then incorporate that data into the WPF student registration demo.

Getting Started

Copy the WebApi/Registration solution from the course notes repository. You will modify this solution to meet the lab's goals.

Goals

1. There are currently 3 controller classes in this API: StudentsController, giving access to Student objects; CoursesController, which lists catalog courses (but not course sections); and RegistrationController, which allows students to enroll in a course section. Briefly review the three controllers; take note of the four `Dto` classes, too.

2. Add a new C# Class to the Controllers directory (don't use the built-in **Controller** option when adding the new item) for a `ScheduleController`. This controller will give information on the schedule of classes for a given semester term. Base this controller on the CoursesController class, and note how that class uses a `CatalogContext` and various routing mechanisms to execute its logic.

3. Write these API methods in your `ScheduleController`:

   (a) `GET /api/schedule/{year}/{term}`: returns a list of all `CourseSectionDto` objects offered in the given semester. The semester's year and term name is in the URL path, as in `/api/schedule/2017/fall`. Your controller method should take these two route parameters and construct the corresponding semester term name, e.g., "Fall 2017", then find the `SemesterTerm` object in the database with that name. If it is found, return all its course sections. Otherwise, throw a 404 Not Found exception.

   (b) `GET /api/schedule/{id:int}`: returns a list of all `CourseSectionDto` objects offered in the given semester, identified by the semester's ID. You should architect a means of sharing the code you wrote in the previous method with this, as they both do a similar thing: given the means to identify a semester, load that semester, and return its list of course sections.

   (c) `GET /api/schedule/terms`: returns a list of all semester terms in the system. You will need to write a new `Dto` for the `SemesterTerm` class, consisting solely of the term's ID and name.

4. Launch your WebAPI project and load the `/api/schedule/terms` endpoint in your browser. Select and copy **one** of the JSON objects in the result. Go to Visual Studio to the RegistrationViewModel.cs file. **Place your cursor prior to** the `public class RegistrationViewModel` line, then use the Edit -> Paste Special -> Paste JSON as Class menu. Rename the new class `SemesterTermDto`, and add a `ToString()` method that returns the name of the term. Do the same procedure for one of the course section objects returned from `/api/schedule/2017/fall`, naming the class `CourseSectionDto`, and providing a `ToString()` that returns a string of the form "CECS 174-01", i.e., "*DEPARTMENT NUMBER-SECTION*".

5. Your view model now needs properties for a list of `SemesterTermDto` objects, and one for a list of `CourseSectionDto` objects. Add these.

6. Recall how the registration app works: the user enters their name, and then types the full name of a course section to register for, e.g., "CECS 174-01". It is assumed that the semester is Fall 2017. We are going to change this:
   (a) Add a new row in the grid below Student's name, with the label "Semester:" and an initially empty ComboBox. Bind the ComboBox's `ItemsSource` to the view model's list of semester terms.
   (b) Change the "Course section" TextBox to an initially empty ComboBox. Bind the ComboBox's `ItemsSource` to the view model's list of course sections.
   (c) Register a new event handler for the `Loaded` event of your `MainWindow` itself (in the `<Window>` tag in the XAML). Make your event handler `async`. In the handler, construct a `RestClient` object and issue a request to `GET /api/schedule/terms`. Parse the resulting array as a list of `SemesterTermDto` objects, then set your view model's property with that list.
   (d) Register a new `async` event handler for the `SelectionChanged` event of the ComboBox of semester terms. In this handler, you will issue a `GET /api/schedule/{id}` request with the ID of the combo box's `SelectedItem`. (Hint: the combo box itself will be the `sender`.) Update your view model so that the combo box of course sections shows the list you just downloaded.
   (e) **Finally**, the `mRegisterBtn_Click` handler will now be broken because we changed some of the UI elements it relied on. Instead of pulling the course name from a text box, you will use the selected item of the course sections combo box to build the JSON body sent in the registration request. You will also not hard-code the semester ID, instead using the semester combo box for that information. Update this handler, as well as the async version.
   (f) To test your code, have the student named "`A B`" register for the course CECS 228-01. You should see 0 (success) as the response. Run the request again, and you should see 3 (already enrolled) as the response. If at any point you want to start the database over, you will have to delete it from Visual Studio's SQL Object Explorer and then re-run the Registration Test console app from the EntityFramework folder in the notes repository.

Important Notes

1. This lab assumes that you have Visual Studio 2017 installed. VS2017 installs a local database engine based on Microsoft SQL Server called **MSSQLLocalDB**.
2. Before you begin, you should verify that you have such a SQL Server instance installed. To do so, open Visual Studio 2017 and go to View->SQL Server Object Explorer. Expand the SQL Server node that appears.
   (a) If you do not see **(localdb)\MSSQLLocalDB**, then click the "Add SQL Server" button (with the green plus sign). In the window that appears, expand Local, then select MSSQLLocalDB and click Connect.
3. Once MSSQLLocalDB appears in your SQL Server Object Explorer, expand its node, then expand Databases. This is where you will see the **CsulbCatalog** database show up once the application is running. You may need to occasionally right-click on Databases and select **Refresh** to see the database after it is created.
4. **BEFORE YOU CAN START THIS LAB,** you must first instantiate your local database with the CSULB catalog information. To do this, load the EntityFramework\Registration solution from the course repository in Visual Studio. Run the "Test" project and use Option 1 to instantiate the database. **Make sure you have retrieved the latest version of the notes repository before you do this**.

Deliverables

Turn in the following when the lab is due:

1. A printed copy of:

(a) `ScheduleController.cs`

(b) `RegistrationViewModel.cs`

(c) `MainWindow.xaml.cs`