Lab Answer Key: Module 10: Improving Application Performance and Responsiveness

Lab: Improving the Responsiveness and Performance of the Application

Exercise 1: Ensuring That the UI Remains Responsive When Retrieving Teacher Data

Task 1: Build and run the application

1. Start the MSL-TMG1 virtual machine if it is not already running.
2. Start the 20483B-SEA-DEV11 virtual machine.
3. Log on to Windows® 8 as **Student** with the password **Pa$$w0rd**. If necessary, click **Switch User** to display the list of users.
4. Switch to the Windows 8 **Start** window and then type Explorer.
5. In the **Apps** list, click **File Explorer**.
6. Navigate to the **E:\Mod10\Labfiles\Databases** folder, and then double-click **SetupSchoolGradesDB.cmd**.
7. Close File Explorer.
8. Switch to the Windows 8 **Start** window.
9. Click **Visual Studio 2012**.
10. In Microsoft® Visual Studio®, on the **File** menu, point to **Open**, and then click **Project/Solution**.
11. In the **Open Project** dialog box, browse to **E:\Mod10\Labfiles\Starter\Exercise 1**, click **Grades.sln**, and then click **Open**.

12. In Solution Explorer, right-click **Solutions 'Grades'**, and then click **Properties**.

13. On the **Startup Project** page, click **Multiple startup projects**. Set **Grades.Web** and **Grades.WPF** to **Start without debugging**, and then click **OK**.

14. On the **Build** menu, click **Build Solution**.

15. On the **Debug** menu, click **Start Without Debugging**.

16. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.

17. Notice that the UI briefly freezes while fetching the list of students for Esther Valle (try moving the application window after logging on but before the list of students appears).

18. Close the application window.

Task 2: Modify the code that retrieves teacher data to run asynchronously

1. On the **View** menu, click **Task List**.

2. In the **Task List** window, in the **Categories** list, select **Comments**.

3. Double-click the **TODO: Exercise 1: Task 2a: Convert GetTeacher into an async method that returns a Task<Teacher>** task.

4. In the code editor, delete the following line of code:

```
public Teacher GetTeacher(string userName)
```

5. In the blank line below the comment, type the following code:

```
public async Task<Teacher> GetTeacher(string userName)
```

6. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2b: Perform the LINQ query to fetch Teacher information asynchronously** task.

7. In the code editor, modify the statement below the comment as shown in bold below:

```
var teacher = await Task.Run(() =>
(from t in DBContext.Teachers
where t.User.UserName == userName
select t).FirstOrDefault());
```

8. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2c: Mark MainWindow.Refresh as an asynchronous method** task.

9. In the code editor, modify the statement below the comment as shown in bold below:

```
public async void Refresh()
```

10. In the **Task List** window, double-click the **TODO: Exercise 1: Task 2d: Call GetTeacher asynchronously** task.

11. In the code editor, modify the statement below the comment as shown in bold below:

```
var teacher = await utils.GetTeacher(SessionContext.UserName);
```

Task 3: Modify the code that retrieves and displays the list of students for a teacher to run asynchronously

1. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3a: Mark StudentsPage.Refresh as an asynchronous method** task.

2. In the code editor, modify the statement below the comment as shown in bold below:

```
public async void Refresh()
```

3. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3b: Implement the OnGetStudentsByTeacherComplete callback to display the students for a teacher here** task.

4. In the blank line below the comment, type the following code:

```
private void
OnGetStudentsByTeacherComplete(IEnumerable<Student> students)
{
}
```

5. In the **Task List** window, double-click the **Exercise 1: Task 3b: Relocate the remaining code in this method to create the OnGetStudentsByTeacherComplete callback (in the Callbacks region)** task.

6. In the code editor, move all of the code between the comment and the end of the **Refresh** method to the Clipboard.

7. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3b: Implement the OnGetStudentsByTeacherComplete callback to display the students for a teacher here** task.

8. Click in the blank line between the curly braces and paste the code from the Clipboard.

9. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3c: Use a Dispatcher object to update the UI** task.

10. In the code editor, click at the end of the comment line, press Enter, and then type the following code:

```
this.Dispatcher.Invoke(() => {
```

11. Immediately after the last line of code in the method, type the following code:

```
});
```

12. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3d: Convert GetStudentsByTeacher into an async method that invokes a callback** task.
13. In the code editor, delete the following line of code:

```
public List<Student> GetStudentsByTeacher(string teacherName)
```

14. In the blank line below the comment, type the following code:

```
public async Task GetStudentsByTeacher(string teacherName,
Action<IEnumerable<Student>> callback)
```

15. In the code editor, modify the return statement below the **if(!IsConnected())** line to return without passing a value to the caller:

```
    return;
```

16. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3e: Perform the LINQ query to fetch Student data asynchronously** task.
17. In the code editor, modify the statement below the comment as shown in bold below:

```
var students = await Task.Run(() =>
(from s in DBContext.Students
where s.Teacher.User.UserName == teacherName
select s).OrderBy(s => s.LastName).ToList());
```

18. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3f: Run the callback by using a new task rather than returning a list of students** task.
19. In the code editor, delete the following code:

```
return students;
```

20. In the blank line below the comment, type the following code:

```
await Task.Run(() => callback(students));
```

21. In the **Task List** window, double-click the **TODO: Exercise 1: Task 3g: Invoke GetStudentsByTeacher asynchronously and pass the OnGetStudentsByTeacherComplete callback as the second argument** task.
22. In the code editor, modify the statement below the comment as shown in bold below:

```
await
utils.GetStudentsByTeacher(SessionContext.UserName,OnGetStudent
sByTeacherComplete);
```

Task 4: Build and test the application

1. On the **Build** menu, click **Build Solution**.

2. On the **Debug** menu, click **Start Without Debugging**.

3. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.

4. Verify that the application is more responsive than before while fetching the list of students for Esther Valle, and then close the application window.

5. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, you should have updated the Grades application to retrieve data asynchronously.

Exercise 2: Providing Visual Feedback During Long-Running Operations

Task 1: Create the BusyIndicator user control

1. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

2. In the **Open Project** dialog box, browse to **E:\Mod10\Labfiles\Starter\Exercise 2**, click **Grades.sln**, and then click **Open**.

3. In Solution Explorer, right-click **Solutions ‘Grades’**, and then click **Properties**.

4. On the **Startup Project** page, click **Multiple startup projects**. Set **Grades.Web** and **Grades.WPF** to **Start without debugging**, and then click **OK**.

5. On the **Build** menu, click **Build Solution**.

6. In Solution Explorer, right-click **Grades.WPF**, point to **Add**, and then click **UserControl**.

7. In the **Name** box, type **BusyIndicator.xaml**, and then click **Add**.

8. In Solution Explorer, expand **Grades.WPF**, and then drag **BusyIndicator.xaml** into the Controls folder.

> **Note:** It is better to create the user control at the project level and then move it into the Controls folder when it is created. This ensures that the user control is created in the same namespace as other project resources.

9. In the BusyIndicator.xaml file, in the **UserControl** element, delete the following attributes:

```
d:DesignWidth="300" d:DesignHeight="300"
```

10. Modify the **Grid** element to include a **Background** attribute, as the following markup shows:

```
<Grid Background="#99000000">
</Grid>
```

11. Type the following markup between the opening and closing **Grid** tags:

```
<Border CornerRadius="6"
        HorizontalAlignment="Center"
        VerticalAlignment="Center">
    <Border.Background>
        <LinearGradientBrush>
            <GradientStop
                Color="LightGray"
                Offset="0" />
            <GradientStop
                Color="DarkGray"
                Offset="1" />
        </LinearGradientBrush>
```

```
        </Border.Background>
        <Border.Effect>
            <DropShadowEffect
                Opacity="0.75" />
        </Border.Effect>
    </Border>
```

12. On the blank line before the closing **Border** tag, type the following code:

```
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />


            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
    </Grid>
```

13. On the blank line before the closing **Grid** tag, type the following code:

```
    <ProgressBar x:Name="progress"
        IsIndeterminate="True"
        Width="200"
        Height="25" Margin="20" />
```

14. Click after the end of the **ProgressBar** element, and then press Enter.
15. In the new line, type the following code:

```
    <TextBlock x:Name="txtMessage"
        Grid.Row="1" FontSize="14"
        FontFamily="Verdana"
        Text="Please Wait..."
        TextAlignment="Center" />
```

16. On the **File** menu, click **Save All**.

17. In Solution Explorer, expand **Grades.WPF**, and then double-click **MainWindow.xaml**.

18. Towards the bottom of the MainWindow.xaml file, locate the **TODO: Exercise 2: Task 1b: Add the BusyIndicator control to MainWindow** comment.

19. Click at the end of the comment, press Enter, and then type the following code:

```
<y:BusyIndicator
   x:Name="busyIndicator"
   Margin="0"
   Visibility="Collapsed" />
```

20. On the **Build** menu, click **Build Solution**.

Task 2: Add StartBusy and EndBusy event handler methods

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2a: Implement the StartBusy event handler** task.

2. In the blank line below the comment, type the following code:

```
private void StartBusy(object sender, EventArgs e)
{
   busyIndicator.Visibility = Visibility.Visible;
}
```

3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2b: Implement the EndBusy event handler** task.

4. In the blank line below the comment, type the following code:

```
private void EndBusy(object sender, EventArgs e)
{
   busyIndicator.Visibility = Visibility.Hidden;
}
```


Task 3: Raise the StartBusy and EndBusy events

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3a: Add the StartBusy public event** task.
2. In the blank line below the comment, type the following code:

```
public event EventHandler StartBusy;
```


3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3b: Add the EndBusy public event** task.
4. In the blank line below the comment, type the following code:

```
public event EventHandler EndBusy;
```


5. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3c: Implement the StartBusyEvent method to raise the StartBusy event** task.
6. In the blank line below the comment, type the following code:

```
private void StartBusyEvent()
{
   if (StartBusy != null)
      StartBusy(this, new EventArgs());
}
```

7. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3d: Implement the EndBusyEvent method to raise the EndBusy event** task.

8. In the blank line below the comment, type the following code:

```
private void EndBusyEvent()
{
   if (EndBusy != null)
      EndBusy(this, new EventArgs());
}
```

9. In Solution Explorer, double-click **MainWindow.xaml**.

10. In the MainWindow.xaml file, locate the **TODO: Exercise 2: Task 3e: Wire up the StartBusy and EndBusy event handlers for the StudentsPage view** comment.

11. Immediately below the comment, modify the **StudentsPage** element to include **StartBusy** and **EndBusy** attributes, as the following code shows:

```
<y:StudentsPage x:Name="studentsPage" StartBusy="StartBusy"
EndBusy="EndBusy"
StudentSelected="studentsPage_StudentSelected"
Visibility="Collapsed" />
```

12. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3f: Raise the StartBusy event** task.

13. In the blank line below the comment, type the following code:

```
StartBusyEvent();
```

14. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3g: Raise**

the EndBusy event task.

15. In the blank line below the comment, type the following code:

```
EndBusyEvent();
```


Task 4: Build and test the application

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.
4. Verify that the application displays the busy indicator while waiting for the list of students to load, and then close the application window.
5. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, you should have updated the Grades application to display a progress indicator while the application is retrieving data.