

Practical Lecture 3 Building an Admin GUI

Practical Session Structure

1. Introduction
2. Building a business component
- 3. Building an admin GUI**
4. Introducing .NET remoting
5. Creating a web service and client website
6. Developing a Java client

2

Overview

- In order to start this session, you need to have completed all of the practical lecture 2
- In this lecture we will implement a graphical user interface (GUI) that provides admin functionality to a project supervisor, making use of the business component built previously

3

Learning Objectives

- Create a graphical user interface
- Make use of an existing business component deployed as a DLL

4

Introduction

- In this practical session we will:
 - Implement a GUI application to test the business component developed in the previous practical lecture
 - Make use of a number of GUI controls (widgets) to build the application
 - Use the façade class for the Admin role in order to extract/insert data into the database

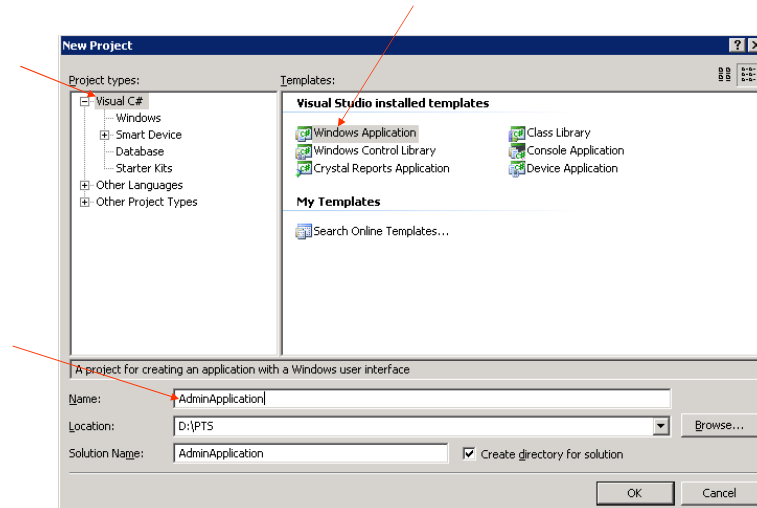
5

Creating the Project

- We will create the Admin GUI as a *Windows Application* in a new solution
- Open Visual Studio 2005
- Go to *File -> New Project*
- Select *Visual C#* as the project type and then select *Windows Application* as the template
- Name the project *AdminApplication* and save it in a suitable location

6

Creating the Project /2



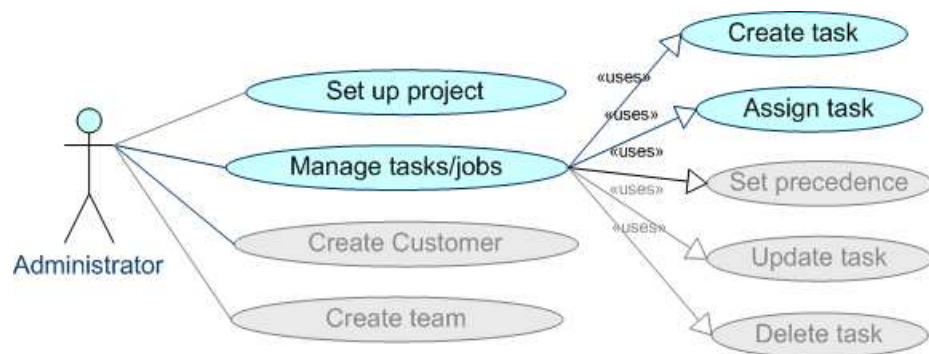
7

Administrator Role

- We need to build a user interface to support the administrator's functionality:
 - setting up the project
 - dividing the project into tasks
 - assign tasks to teams
 - create new teams if necessary
 - update any changes to the project
 - track overall project progress

8

What are we building?

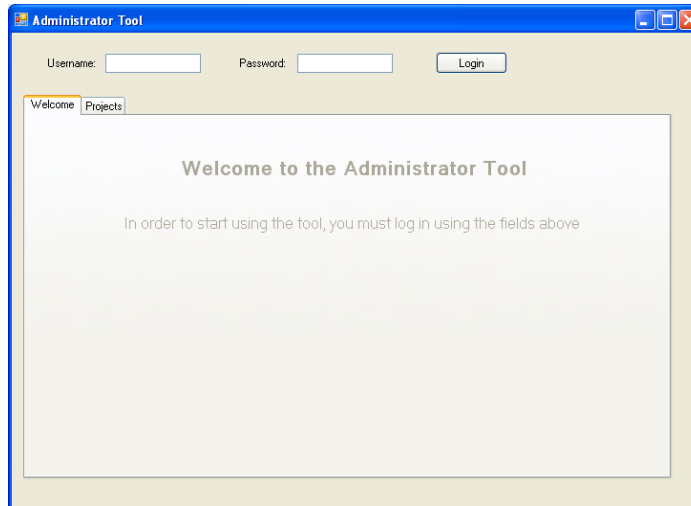


9

What are we building?

10

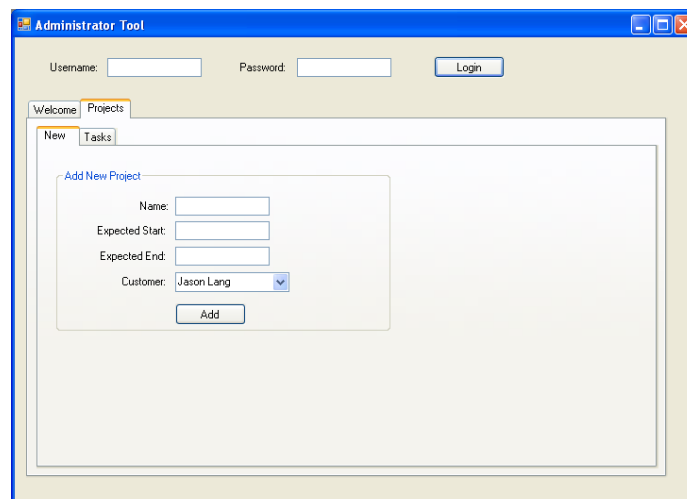
What are we building?



A screenshot of a web application window titled "Administrator Tool". At the top, there are two input fields labeled "Username:" and "Password:", followed by a "Login" button. Below these fields are two tabs: "Welcome" (which is selected) and "Projects". The main content area of the "Welcome" tab displays the text "Welcome to the Administrator Tool" and a message: "In order to start using the tool, you must log in using the fields above".

11

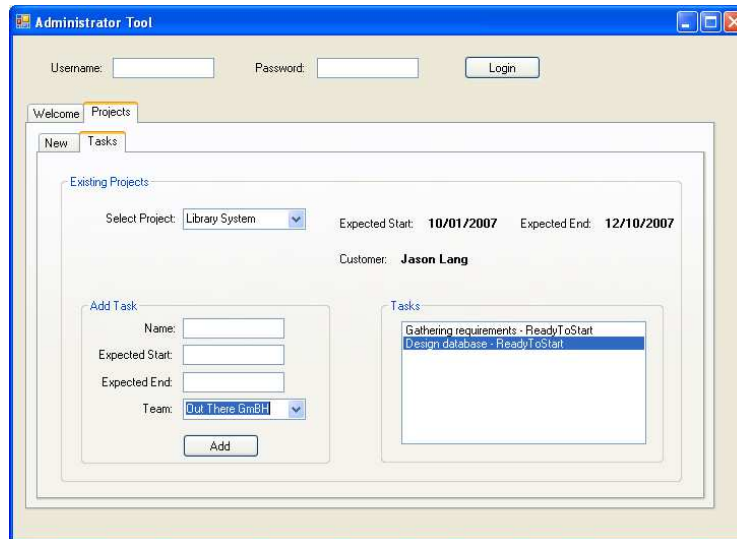
What are we building?



A screenshot of the same "Administrator Tool" window, but with the "Projects" tab selected. Within the "Projects" tab, there are two sub-tabs: "New" (selected) and "Tasks". The "New" sub-tab contains a form titled "Add New Project". This form has four input fields: "Name:", "Expected Start:", "Expected End:", and "Customer:". The "Customer:" field is a dropdown menu currently showing "Jason Lang". Below these fields is an "Add" button.

12

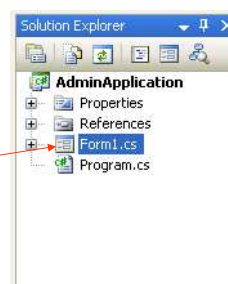
What are we building?



13

frmAdmin

- By creating a windows application, Visual Studio will automatically create a windows form
- Right-click and rename it to frmAdmin.cs



14

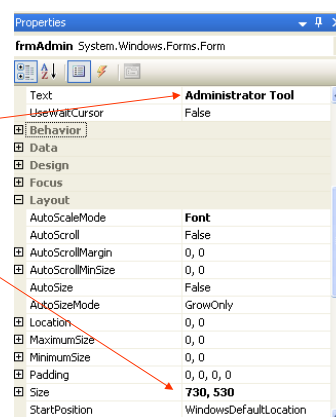
Partial Classes

- When creating forms in Visual Studio 2005, it will create two files for each form
- Separation of logic and user interface design and layout code
- When you add controls to a form the code for positioning, customising, adding events, etc. will be placed in a xxx.Designer.cs file
- Thus, the code for the logic is not cluttered

15

Form Properties

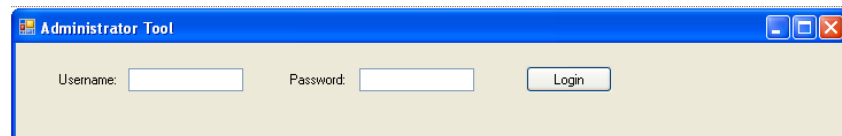
- In the properties of the form:
 - Change “Text” to *Administrator Tool*
 - Resize the form to 730 by 530



16

Controls for Login

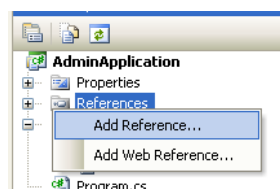
- Add 2 textboxes and change the Properties (from the Toolbox pane):
 - Name: txtUsername, txtPassword
- Add associated labels to each textbox with the text set to Username and Password.
- Add a button named btnLogin and Text set to Login



17

Using the PTSLibrary

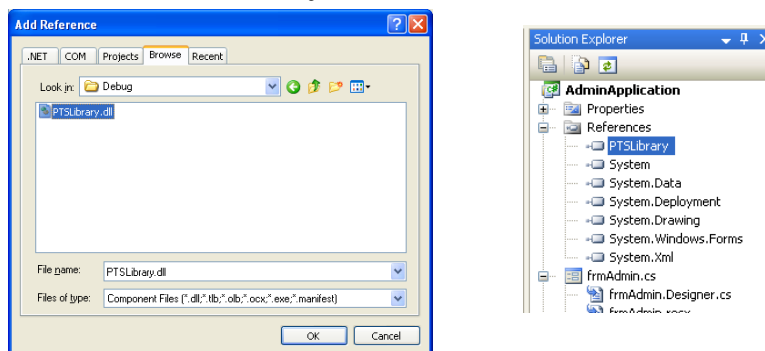
- The admin tool user interface we are building will need to communicate with the PTSLibrary
 - For this we need to add the PTSLibrary component as a reference to our project
 - Right-click on *References* in the Solution Explorer and then select *Add Reference...*



18

Adding a Reference

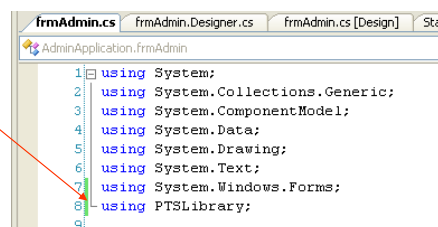
- On the Add Reference dialog select the *Browse* tab and browse to the PTSLibrary project (subfolder PTSLibrary > bin > Debug) and add the PTSLibrary.dll as a reference



19

Adding a Reference /2

- By adding a reference to the PTSLibrary, we provide our project with visibility of our component
 - But in order to make calls on the PTSAdminFacade from the code in our frmAdmin form, we still need to add a using directive



20

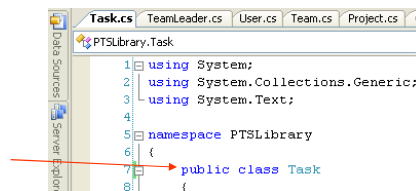
Using a Component in a Different Namespace

- Calls are made from our admin tool (AdminApplication namespace) to our business component (PTSLibrary namespace)
 - Visibility needs to be considered
 - Classes in PTSLibrary were created with default visibility
 - All classes that will need to be accessed externally will need to be made public (including classes used as return types)

21

Making Classes Public

- Open the PTSLibrary project created in the previous session and change the visibility of the following classes to *public*:
 - PTSSuperFacade
 - PTSAAdminFacade
 - PTSCustomerFacade
 - PTSClientFacade
 - Customer
 - Project
 - User
 - Team
 - TeamLeader
 - Task
 - PTSSuperDAO



22

Instantiating Façade

- Open the code of the frmAdmin file and add code to declare/instantiate an object of PTSAdminFacade and an integer called adminId

```

4  using System.Data;
5  using System.Drawing;
6  using System.Text;
7  using System.Windows.Forms;
8  using PTSLibrary;
9
10 namespace AdminApplication
11 {
12     public partial class frmAdmin : Form
13     {
14         private PTSAdminFacade facade;
15         private int adminId;
16
17         public frmAdmin()
18         {
19             InitializeComponent();
20             facade = new PTSAdminFacade();
21             adminId = 0;
22         }
23     }

```

23

Adding Event Handling

- Visual Studio makes it easy to add event handling code
- To add code that is executed when the btnLogin button is clicked, just double click on the button on the designer
 - An event handler is automatically attached
 - Skeleton code is created for a handling method

24

Login Code

- Add code to the skeleton method to take the details from the text fields and attempt authentication on PTSLibrary

```
private void btnLogin_Click(object sender, EventArgs e)
{
    try
    {
        adminId = facade.Authenticate(this.txtUsername.Text, this.txtPassword.Text);
        if (adminId != 0)
        {
            this.txtUsername.Text = "";
            this.txtPassword.Text = "";
            MessageBox.Show("Successfully logged in");
        }
        else
        {
            MessageBox.Show("Wrong login details");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Extracts the text from the text field

Displays a popup

25


Add Administrator to DB

- In order to test the login, we will need to add an administrator to our database
 - Ensure that SQL Server is running
 - Open your database
 - Open the Person table and add an administrator (make sure *IsAdministrator* is set to true)

Table - dbo.Person		Summary					
UserId	Name	Username	Password	Email	TelephoneNo	IsAdministrator	
1	Markus	Admin	Admin	NULL	NULL	True	
*	NULL	NULL	NULL	NULL	NULL	NULL	

26

Test Login

- Run the form by clicking on , selecting *Start Debugging* from the *Debug* menu or press F5
 - If all goes well, you should see a popup as shown below, if not you need to find and correct the problem

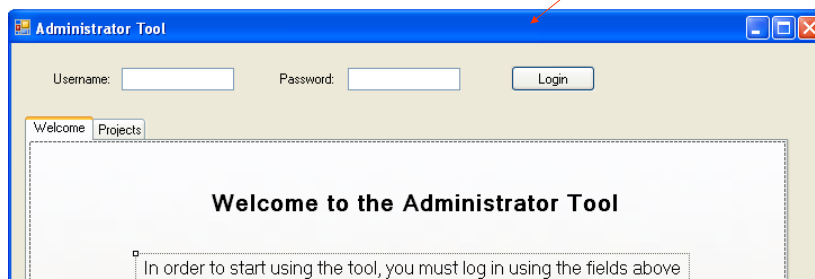


27

Add a TabControl

- Drag a TabControl from the Toolbox onto the form and change the text on the tab pages to *Welcome* and *Projects*
- Add two labels to the first tab page

Should now look like this



28

Add Another Tab Control

- On the second tab page (Projects), add another tab control with two tabs (*New* and *Tasks*)
- On the first page of the second tab control add:
 - 1 GroupBox
 - 4 Labels
 - 3 TextBoxes (txtProjectName, txtProjectStart, txtProjectEnd)
 - 1 ComboBox (cbCustomer)
 - 1 Button (btnAddProject)
- Have a look at the next slide to see what it should look like when you're finished

29

Add Another Tab Control /2

The screenshot shows a Windows application titled "Administrator Tool". At the top, there are fields for "Username:" and "Password:" with a "Login" button. Below this is a tab control with "Welcome" and "Projects" tabs. The "Projects" tab is active and contains a sub-tab control with "New" and "Tasks" tabs. The "New" tab is active and displays a form titled "Add New Project". This form includes four labels: "Name:", "Expected Start:", "Expected End:", and "Customer:". Each label is followed by a text input field, except for "Customer:" which is followed by a dropdown menu. At the bottom of the form is an "Add" button.

30

Adding More Controls

- Go to the second tab of the second tab control (Tasks) and add:
 - 3 GroupBoxes
 - 8 Labels - normal
 - 3 Labels – set to bold (lblStartDate, lblEndDate, lblCustomer)
 - 3 TextBoxes (txtTaskName, txtTaskStart, txtTaskEnd)
 - 2 ComboBoxes (cbProjects, cbTeams)
 - 1 Button (btnAddTask)
 - 1 ListBox (lbTasks)

31

Adding More Controls /2

- Your form should now look like this:

The screenshot shows the 'Administrator Tool' application window. At the top, there are fields for 'Username:' and 'Password:' with a 'Login' button. Below this is a tab control with 'Welcome' and 'Projects' tabs. The 'Projects' tab is active, and it contains a sub-tab control with 'New' and 'Tasks' tabs. The 'Tasks' sub-tab is active, displaying a form with the following elements:

- Existing Projects:** A section containing a 'Select Project:' dropdown menu, and labels for 'Expected Start:' (with bold text 'lblStartDate'), 'Expected End:' (with bold text 'lblEndDate'), and 'Customer:' (with bold text 'lblCustomer').
- Add Task:** A section containing three text boxes for 'Name:', 'Expected Start:', and 'Expected End:', a 'Team:' dropdown menu, and an 'Add' button.
- Tasks:** A section containing a large empty rectangular box labeled 'lbTasks'.

32

Restricting Access

- When a user starts the application, we want to restrict access to anything on the second page of the first tab control until he/she has logged in successfully
- Select the *tabControl1* and set its *Enabled* property to *False*
- If you run the application now, you will see that the tab control is greyed out

33

Add Required Variables

- Declare the variables that will be needed in our tool:

```

11 {
12     public partial class frmAdmin : Form
13     {
14         private PTSAdminFacade facade;
15         private int adminId;
16         private Customer[] customers;
17         private Project[] projects;
18         private Team[] teams;
19         private Project selectedProject;
20         private Task[] tasks;
21     }
22     public frmAdmin()
23     {
24     }

```

34

Adjust Login Code

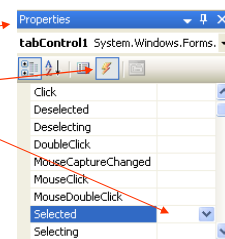
- Add code to the *btnLogin_Click* method to ensure that the tab control is enabled and user taken to the next tab on successful login

```
private void btnLogin_Click(object sender, EventArgs e)
{
    try
    {
        adminId = facade.Authenticate(this.txtUsername.Text, this.txtPassword.Text);
        if (adminId != 0)
        {
            this.txtUsername.Text = "";
            this.txtPassword.Text = "";
            MessageBox.Show("Successfully logged in");
            {tabControl1.SelectTab(1);
            tabControl1.Enabled = true;
            }
        }
        else
        {
            {tabControl1.SelectTab(0);
            tabControl1.Enabled = false;
            MessageBox.Show("Wrong login details");
            }
        }
    }
}
```

35

Load Customers

- When the *Projects* tab of *tabControl1* is selected, the customers should be loaded into the *cbCustomer* combo box
 - Need to add a method to react to the event of the control being selected
 - Select *tabControl1*
 - Go to the properties
 - Change to Events
 - Double-click in the field next to *Selected*
 - Skeleton method is automatically created



36

Load Customers /2

- If the second page is selected, the customers will be obtained from the façade and the combo box is populated
- Add the following code to the `tabControl1_Selected` method:

Call to the PTSLibrary

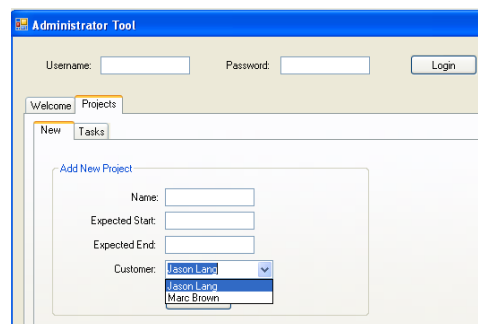
Note how the combo box is bound to an array

```
private void tabControl1_Selected(object sender, TabControlEventArgs e)
{
    if (tabControl1.SelectedIndex == 1)
    {
        customers = facade.GetListOfCustomers();
        cbCustomer.DataSource = customers;
        cbCustomer.DisplayMember = "Name";
        cbCustomer.ValueMember = "id";
    }
}
```

37

Load Customers /3

- Before you can test the new code, make sure that you enter some entries into the Customer table of the database
- If all goes well you will see the customers you entered listed in the combo box



38

Add New Project

```
private void btnAddProject_Click(object sender, EventArgs e)
{
    DateTime startDate;
    DateTime endDate;

    if (txtProjectName.Text == "")
    {
        MessageBox.Show("You need to fill in the name field");
        return;
    }

    try
    {
        startDate = DateTime.Parse(txtProjectStart.Text);
        endDate = DateTime.Parse(txtProjectEnd.Text);
    }
    catch (Exception)
    {
        MessageBox.Show("The date(s) are in the wrong format");
        return;
    }

    facade.CreateProject(txtProjectName.Text, startDate, endDate, (int)cbCustomer.SelectedValue, adminId);
    txtProjectName.Text = "";
    txtProjectStart.Text = "";
    txtProjectEnd.Text = "";
    cbCustomer.SelectedIndex = 0;
    MessageBox.Show("Project successfully created");
    tabControl2.SelectTab(1);
}
```

- Add code to be executed when the *Add* button for a project is pressed

39

Loading Projects and Teams

- When the *Tasks* tab of *tabControl2* is selected, the projects and teams should be loaded into the *cbProjects* and *cbTeams* combo boxes
- Similar to customer combo box add an event handling method

```
private void tabControl2_Selected(object sender, TabControlEventArgs e)
{
    if (tabControl2.SelectedIndex == 1)
    {
        projects = facade.GetListOfProjects(adminId);
        cbProjects.DataSource = projects;
        cbProjects.DisplayMember = "Name";
        cbProjects.ValueMember = "ProjectId";
        setProjectDetails();

        teams = facade.GetListOfTeams();
        cbTeams.DataSource = teams;
        cbTeams.DisplayMember = "Name";
        cbTeams.ValueMember = "TeamId";
    }
}
```

Again, note how the combo boxes are bound to arrays of objects

We need to define this method

40

SetProjectDetails Method

- Whenever the projects are loaded or a different project is selected from *cbProjects*, the details shown should be updated – this is what this method does

```
private void setProjectDetails()
{
    selectedProject = projects[cbProjects.SelectedIndex];
    lblStartDate.Text = selectedProject.ExpectedStartDate.ToShortDateString();
    lblEndDate.Text = selectedProject.ExpectedEndDate.ToShortDateString();
    lblCustomer.Text = ((Customer)selectedProject.TheCustomer).Name;
    UpdateTasks();
}
```

Sets the *selectedProject* variable to the currently selected project

We need to define this method

Values are extracted straight from the *Project* and *Customer* objects

41

UpdateTasks Method

- Whenever the selected project changes, the tasks for that particular project should be loaded – this is what this method is for

```
private void UpdateTasks()
{
    tasks = facade.GetListOfTasks(selectedProject.ProjectId);
    lbTasks.DataSource = tasks;
    lbTasks.DisplayMember = "NameAndStatus";
    lbTasks.ValueMember = "TaskId";
}
```

There is no such property in the Task class. This needs to be introduced.

42

NameAndStatus Property

- In order to display the name and status of each task in the list box, it is necessary to add a property which returns this in the Task object
- Open the PTSLibrary project and add the following code to the Task class. Then rebuild the project

```
public string NameAndStatus
{
    get { return name + " - " + status; }
}
```

43

Reload Project Details

- We have a method for reloading project details – *setProjectDetails*
- When the selected index of *cbProjects* changes, this method should be called
- Double-click on the *cbProjects* combo box
 - This automatically adds an event handler and writes a skeleton method
 - Add a method call to *setProjectDetails*

```
private void cbProjects_SelectedIndexChanged(object sender, EventArgs e)
{
    setProjectDetails();
}
```

44

Add New Task

```
private void btnAddTask_Click(object sender, EventArgs e)
{
    DateTime startDate;
    DateTime endDate;

    if (txtTaskName.Text == "")
    {
        MessageBox.Show("You need to fill in the name field");
        return;
    }

    try
    {
        startDate = DateTime.Parse(txtTaskStart.Text);
        endDate = DateTime.Parse(txtTaskEnd.Text);
    }
    catch (Exception)
    {
        MessageBox.Show("The date(s) are in the wrong format");
        return;
    }

    facade.CreateTask(txtTaskName.Text, startDate, endDate, (int)cbTeams.SelectedValue, selectedProject.ProjectId);
    txtTaskName.Text = "";
    txtTaskStart.Text = "";
    txtTaskEnd.Text = "";
    cbTeams.SelectedIndex = 0;
    MessageBox.Show("Task successfully created");
    UpdateTasks();
}
```

- Add code to be executed when the *Add* button for a task is pressed

45

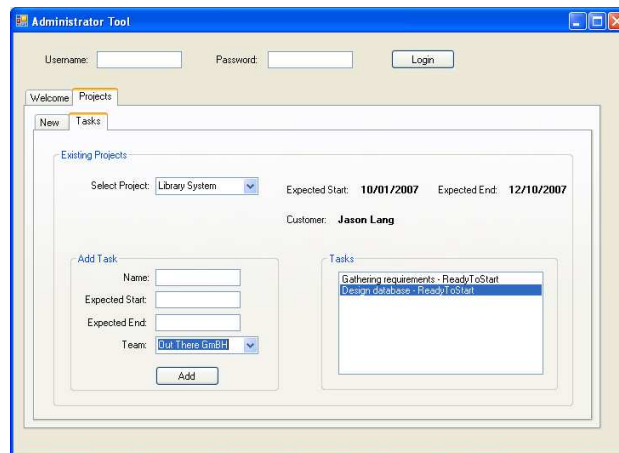
Test the Application

- Now that you have built a prototype of the Admin tool, compile and run it
- Test all the features implemented
 - Log in
 - Add project
 - Add task
 - Change selected project
- Fix any problems that you might find

46

Test the Application /2

- Your application should look somewhat like this:



47

Complete Code listing /2

- Please see the notes of this slide for the complete code listing of the *frmAdmin.Designer.cs* file

48

Summary

- In this session we have built a graphical user interface application for an administrator
- This application uses the PTSLibrary component implemented in the previous lecture
- In the next session we will take both projects and make them communicate using .NET Remoting

49