3-tier Architecture

Step by step Exercises

Hans-Petter Halvorsen

https://www.halvorsen.blog
Software Architecture

3-Tier: A way to structure your code into logical parts. Different devices or software modules can share the same code.

Web Services: A standard way to get data over a network/Internet using standard Web protocols (HTTP, etc.)

APIs: Application Programming Interface. Different devices or software modules can share the same code. Code once, use it many times
The database-centric style. Typically, the clients communicate directly with the database.

A three-tier style, in which clients do not connect directly to the database.

Web Services, etc.
3-tier/layer Architecture

Note! The different layers can be on the same computer (Logic Layers) or on different Computers in a network (Physical Layers)
Why 3-Tier (N-Tier Architecture?)

• Flexible applications
• Reusable code
  – Code once, use many times
• Modularized
  – You need only to change part of the code
  – You can deploy only one part
  – You can Test only one part
  – Multiple Developers
• Different parts (Tiers) can be stored on different computers
• Different Platforms and Languages can be used
• etc.
Presentation tier
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

http://en.wikipedia.org/wiki/Multitier_architecture
3-tier/layer Architecture

**Presentation Tier**
- This is the topmost level of the application.
- The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents.
- It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network.
- In simple terms it is a layer which users can access directly such as a web page, or an operating systems GUI.

**Application tier (business logic, logic tier, data access tier, or middle tier)**
- The logical tier is pulled out from the presentation tier and, as its own layer.
- It controls an application’s functionality by performing detailed processing.

**Data tier**
- This tier consists of database servers. Here information is stored and retrieved.
- This tier keeps data neutral and independent from application servers or business logic.
- Giving data its own tier also improves scalability and performance.

3-tier Architecture

Presentation Tier

Business Logic Tier

Data Access Tier

Stored Procedures

Database

Data Tier

Logic Tier
3-tier Architecture

Different Devices can share the same Business and Data Access Code

The different Tiers can be physical or logical
3-tier + WebService Architecture - Example

Server-side

- Server(s)
- Web Server
- Web Services
- Business/Data Logic Tier
- Data Source
- Stored Procedures

Client-side

- Presentation Tier
- Data Tier
- Client-side devices (e.g., smartphone, tablet, laptop)
3-tier + WebService Architecture - Example

Team Foundation Server

Web Server

Installed on one or more Windows Servers in your LAN or in the Cloud

Web Services

Presentation Tier

Business/Data Logic Tier

Stored Procedures

Data Source

TFS Client

Visual Studio

Team Foundation Server

MS SQL Server
3-tier Architecture Scenarios

Presentation Layer

Business Logic

Data Access Logic

Client

Internet

Presentation Layer

Web Service

Web Server

Local Network (LAN)

Database

Stored Procedures

Firewall

Server

Client
Exercises

1. Create **Data Tier** (Database)
2. Create **Logic Tier** (Database Communication Logic)

Create **Presentation Tier** (User Interface Logic):

3. **WebApp**: Using ASP.NET Web Forms (WS normally not needed)
4. **Desktop App**: Using WinForms
   A. Without Web Services (We assume the App will be used only in the LAN and that we have direct access to the Database)
   B. With Web Services (We assume the App should be used on Internet outside the Firewall without direct DB access)
Note! The different Tiers can be on the same Computer (Logic Layers) or on different Computers in a network (Physical Layers)

Device Tiers

- Presentation Tiers (e.g., WinForms, Mobile App)
- Business/Logic Tier
- Data Access Tier (e.g., ADO, ADO.NET)
- Data Tier (Tables, Views, Stored Procedures)

Network Scenarios

- Local Network
- Internet
- Firewall

Device App Share the same Business/Logic Tier and APIs

Separate Presentation Tier for each Device App

Web App, Desktop App, Mobile App

Clients, Presentation Tier, Data Tier, Business Tier, Logic Tier, Web Service, Web Server, Internet, Local Network, IP, etc.
Visual Studio Projects

Solution with all Projects (Logic Tier, Web Service, Desktop App, Web App, Mobile App)

Solution with Projects used by Web App (Logic Tier, Web App)
We are going to create the Database / Data Layer/Tier, including:

1. Tables
2. Views
3. Stored Procedures
4. Triggers
5. Script for some “Dummy” Data

Download Zip Files with Tables, Views, Stored Procedures and Triggers in order to create the Data Tier in SQL Server (The ZIP File is located on the same place as this File)

Note! Install them in this order
Execute the different Scripts inside SQL Server Management Studio
You are finished with the Exercise
Create Logic Tier

Purpose:
- All the Apps should/could share the same Logic Tier
- To make your Apps easier to maintain and extend
- etc.

ASP.NET Web Forms
- Presentation Tier

WinForms
- Presentation Tier

Windows Store App
- Presentation Tier

Visual Studio
- Logic Tier

Data Tier
- Database
Create an Empty (Blank) **Solution** in Visual Studio
Add **Project** for Logic Tier (Data Access)

Select a "**Class Library**" Project

"LogicTier"
Add a New **Class** to the Project ("StudentData.cs")
Create the **Code**, e.g., like this (“StudentData.cs”):

Create your own **Namespace**

A View that collects data from several tables

**Improvements:** Use Try... Catch ...
You should test the SQL Query in the SQL Server Management Studio first
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Data;

namespace Tuc.School.LogicTier
{
    public class StudentData
    {
        public DataSet GetStudentDB(string connectionString)
        {
            string selectSQL = "select StudentName, StudentNumber, SchoolName, ClassName,
            Grade from StudentData order by StudentName";

            // Define the ADO.NET objects.
            SqlConnection con = new SqlConnection(connectionString);

            SqlDataAdapter da = new SqlDataAdapter(selectSQL, con);

            DataSet ds = new DataSet();
            da.Fill(ds);

            return ds;
        }
    }
}
Create a proper name for the **Assembly** (.dll File)

This will be the Assembly for your Logic Tier, that can be imported and used in other projects. Create once – use it many times!!
You are finished with the Exercise
Presentation Layer
Web App: ASP.NET WebForms

We will create a WebForm like this where the data comes from our Logic Tier
Add Project for Presentation Tier (ASP.NET WebForm)
Add a New **Class** ("Student.cs")
Add Code (“Student.cs”)

Note! This is our Logic Tier

Add a Reference to the Assembly in the Logic Tier
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data;
using Tuc.School.LogicTier;

namespace Tuc.School.WebApp
{
    public class Student
    {
        public DataSet GetStudent(string connectionString)
        {
            StudentData studentData = new StudentData();
            return studentData.GetStudentDB(connectionString);
        }
    }
}
Add a New **WebForm** *(StudentInformation.aspx)*
Create WebForm Page ("StudentInformation.aspx")

GridView (Drag & Drop from Toolbox or create code in .aspx file)
using System.Web.Configuration;
using Tuc.School.WebApp;

namespace WebApp
{
    public partial class StudentInformation : System.Web.UI.Page
    {
        private string connectionString =
            WebConfigurationManager.ConnectionStrings["SCHOOLConnectionString"].ConnectionString;

        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                FillStudentGrid();
            }
        }

        private void FillStudentGrid()
        {
            DataSet ds = new DataSet();
            Student studentList = new Student();
            ds = studentList.GetStudent(connectionString);
            gridStudentData.DataSource = ds;
            gridStudentData.DataBind();
        }
    }
}
Store the “ConnectionString” for your Database in “Web.Config”

```xml
<configuration>
    <connectionStrings>
        <add name="SCHOOLConnectionString" connectionString="Data Source=macwin8;Initial Catalog=SCHOOL;Persist Security Info=True;User ID=sa;Password=xxxxxx"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>
```

Then you can easily switch Database without changing the Code!!
Test your Web App

Note! We have used a “View” in order to get data from several tables
Additional Exercise:

Update GridView with New Data from Database

Goto New WebForm in order to Add another Student
You are finished with the Exercise
Part A: Without Web Services (we assume the App will be used only in the local LAN (or local on the same computer where the database is located) and that we have direct access to the Database)
Add a WinForm Project
Add a New **Class** ("StudentWinForm.cs")
Add **Code** in Class

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using Tuc.School.LogicTier;

namespace Tuc.School.WinFormsApp
{
    class StudentWinForm
    {
        public DataSet GetStudent(string connectionString)
        {
            StudentData studentData = new StudentData();
            return studentData.GetStudentDB(connectionString);
        }
    }
}
```

Add a Reference to the Assembly in the Logic Tier
using System.Data;
using Tuc.School.LogicTier;

namespace Tuc.School.WinFormApp
{
    class StudentWinForm
    {
        public DataSet GetStudent(string connectionString)
        {
            StudentData studentData = new StudentData();

            return studentData.GetStudentDB(connectionString);
        }
    }
}
Create Form

Student Information

DataGridView
Create Form Code

```csharp
namespace WinformApp
{
    public partial class Form1 : Form
    {
        private string connectionString = ConfigurationManager.ConnectionStrings["SCHOOL_ConnectionString"].ConnectionString;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            FillStudentGrid();
        }

        private void FillStudentGrid()
        {
            DataSet ds = new DataSet();
            StudentList studentList = new StudentListForm();
            ds = studentList.GetStudent(connectionString);
            dataGridView1.DataSource = ds.Tables[0];
        }
    }
}
```
using System.Configuration;
using Tuc.School.WinFormsApp;

namespace WinFormApp
{
    public partial class Form1 : Form
    {
        private string connectionString = ConfigurationManager.ConnectionStrings["SCHOOLConnectionString"].ConnectionString;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            FillStudentGrid();
        }

        private void FillStudentGrid()
        {
            DataSet ds = new DataSet();
            StudentWinForm studentList = new StudentWinForm();
            ds = studentList.GetStudent(connectionString);
            dataGridViewStudentInformation.DataSource = ds.Tables[0];
        }
    }
}
Note! Add System.Configuration Reference
Create DB ConnectionString in **App.config**

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>

  <connectionStrings>
    <add name="SCHOOLConnectionString" connectionString="Data Source=macwin8;Initial Catalog=SCHOOL;Persist Security Info=True;User ID=sa;Password=xxxxxx"
         providerName="System.Data.SqlClient" />
  </connectionStrings>

</configuration>
```
Test it

It works!!!
You are finished with the Exercise
Part B: Using **Web Services** (we assume the App should be used on Internet outside the Firewall)
Step 1: Create Web Service

Create an ASP.NET Project:

Add Web Service:

“SchoolWS”

“SchoolWS.asmx”
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

using System.Data;
using System.Web.Configuration;
using Tuc.School.LogicTier;

namespace SchoolWS
{
    // Summary description for SchoolWS
    {
        private string connectionString = WebConfigurationManager.ConnectionStrings["SCHOOLConnectionString"].ConnectionString;

        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }

        [WebMethod]
        public DataSet GetStudent()
        {
            StudentData studentData = new StudentData();
            return studentData.GetStudentDB(connectionString);
        }
    }
}

Web Service Code

Database ConnectionString is located in Web.config

Web Service Method
Database ConnectionString is located in `Web.config`
Test Web Service

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company’s Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute’s Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#?

```csharp
[WebService(Namespace="http://microsoft.com/webservices/*")]
public class MyWebService {
    // implementation
```
Deploy/Publish Web Service to IIS

Copy Web Service Files (Project) to default IIS Directory: `C:\inetpub\wwwroot`
Test if WS working:

http://localhost/SchoolWS
Step 2: Use Web Service in WinForm

Create New WinForm Project:

“WinFormAppWSClient”
Add Web Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: [http://localhost/SchoolWS/](http://localhost/SchoolWS/)  

Our Web Service Methods

- GetStudent
- HelloWorld

1 service(s) found at address 'http://localhost/SchoolWS/'.

Namespace: SchoolWSReference

OK  Cancel
Create GUI

Student Information

DataGridView

Label
Create Code

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormAppWSClient
{
    public partial class FormWSClient : Form
    {

        public FormWSClient()
        {
            InitializeComponent();
        }

        private void FormWSClient_Load(object sender, EventArgs e)
        {
            FillStudentGrid();
        }

        private void FillStudentGrid()
        {
            DataSet ds = new DataSet();

            SchoolWSReference.SchoolWSSoapClient schoolWs = new SchoolWSReference.SchoolWSSoapClient();
            ds = schoolWs.GetStudent();

            dataGridViewStudentInformation.DataSource = ds.Tables[0];
        }
    }
}
using System.Windows.Forms;

namespace WinFormAppWSClient
{
    public partial class FormWSClient : Form
    {
        public FormWSClient()
        {
            InitializeComponent();
        }

        private void FormWSClient_Load(object sender, EventArgs e)
        {
            FillStudentGrid();
        }

        private void FillStudentGrid()
        {
            DataSet ds = new DataSet();

            SchoolWSReference.SchoolWSSoapClient schoolWs = new SchoolWSReference.SchoolWSSoapClient();
            ds = schoolWs.GetStudent();

            dataGridViewStudentInformation.DataSource = ds.Tables[0];
        }
    }
}
Test it:

It works!!!
You are finished with the Exercise
Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no
Web: https://www.halvorsen.blog