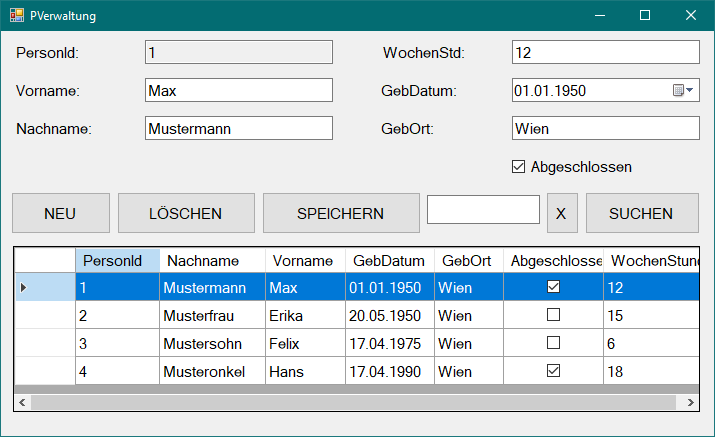
C# Schichtarchitektur – Ver 1

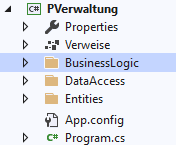
Personal Verwaltung mit WinForm und MS SQL Server- ver1

Ziel



Auflisten aller Personal-Daten in einer Tabelle, Auswahl und Bearbeiten eines Personals, Hinzufügen und Löschen eines Personals. Textsuche in den Einträgen.

Aufbau nach Schicht-Architektur



UI

BusinessLogic

DataAccess

Entities

Präsentation, Eingabe, Ausgabe

Berechtigung, Aufbereitung der Daten

Speichern und Laden von Daten

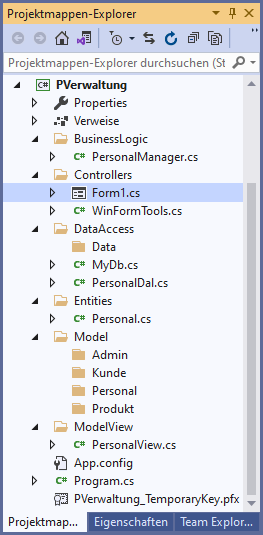
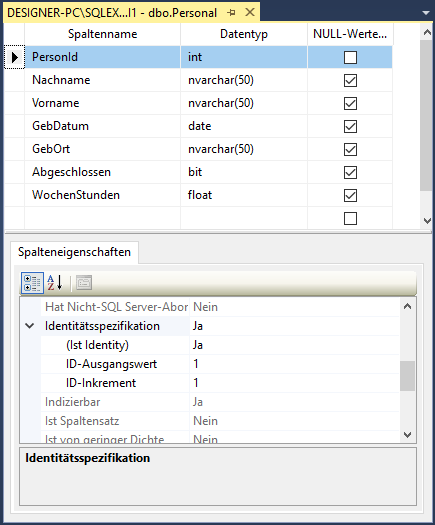
* Realisierung

**Schichten** können entweder als **Ordner** in einem Projekt,

oder als **einzelne Klassenbibliotheken** in einer verteilten Lösung

erstellt werden.

Aufbereitung

*(Mapping = Übernahme von Daten aus nötigen Felder in eine ähnliche Klasse)*

Folgende Bereiche werden im Projekt als Order angelegt.

* Entities

„Entities“ beinhaltet die Klassen, die die Datenbank Tabellen abbilden. Bsp: *Personal.cs*

Die Klassen von Entities sollen von allen Schichten (DataAccess) erreicht werden können.

* DataAccess (DAL = Data Access Layer)

Hier werden statische Klassen abgelegt, die für Datenzugriffe (Lesen und Schreiben in die DB) zuständig sind. Diese Klassen bekommen zusätzlich die Abkürzung „Dal“ als den Namen. (Bsp: Personal**Dal**.cs) Parameter und Rückgabewerte der Methoden in den DAL-Klassen sind Objekte aus dem Entities-Ebene.

* BusinnessLogic (BL = Businness Layer)

Geschäftsablaufebene ist für die Berechnung und Vorbereitung der Daten zuständig. Dies können die Auflistung, Filterung und Auswahl bestimmter Einträge sein. Auch die Ermittlung bestimmter Daten (Summe, Anzahl, Max, Min, …) geschehen in dieser Ebene.

* Controllers (UI)

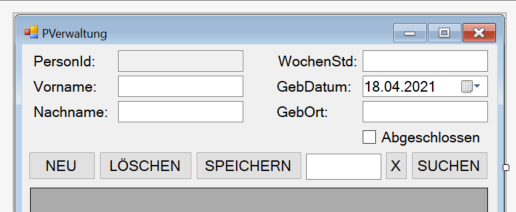
UI (User Interface) = Benutzer Schnittstelle sind die Benutzeroberflächen. Plattform abhängige Codes (wie Eingabe und Ausgabe) befinden sich in dieser Ebene.

* Model

Die Klassen, die nicht für das Abbilden der Datenbanktabellen zuständig sind, werden hier abgelegt.

* ModelView

Die Klassen, die für die Anpassung der Entity-Klassen ans UI, werden hier abgelegt. Das sind die Klassen, die für die Plattform abhängige Codes bestimmt sind. (Mapping)

Form1.cs Design

Zuerst wird eine Benutzeroberfläche unter WinForm entworfen.

Folgende WinForm-Komponente werden auf die Oberfläche positioniert:

txtPersonId txtVorname txtNachname txtWochenStd

dtpGebDatum txtGebOrt cbxAbgeschlossen btnNeu

btnLoeschen btnSpeichern txtSuchen btnXSuchen

btnSuchen dgvPersonal

Datenbank Tabelle „Personal“

Um die Benutzeroberfläche zu realisieren, werden nötige Felder mit den entsprechenden Datentypen (wie unten angeführt) am SQL Server in einer Tabelle angelegt.

PersonId int Unchecked

Nachname nvarchar(50) Checked

Vorname nvarchar(50) Checked

GebDatum date Checked

GebOrt nvarchar(50) Checked

Abgeschlossen bit Checked

WochenStunden float Checked

Entities/Personal.cs

Die Datenbanktabelle wird als eine Klasse abgebildet.

public class Personal

{

public int PersonId { get; set; }

public string Nachname { get; set; }

public string Vorname { get; set; }

public DateTime GebDatum { get; set; }

public string GebOrt { get; set; }

public bool Abgeschlossen { get; set; }

public int WochenStunden { get; set; }

}

DataAccess / MyDb.cs

Diese Klasse ist für den Zugang zur Datenbank zuständig. Zugangsdaten werden aus der Config-Datei gelesen und das SQL-Connection Objekt wird zur Verfügung gestellt.

Damit das funktioniert sollte die **System.Configuration.dll** aus der .NET Framework in den Verweisen eingetragen sein. Ein Beispiel als Verbindungsdaten ist in der Klasse als Kommentar angegeben.

**public class MyDb**

{

private static string \_MyConnectionString;

internal static string MyConnectionString

{

get { return \_MyConnectionString; }

private set { \_MyConnectionString = value; }

}

internal static string MyConnectionString2

{

get { return GetConnectionStrings(); }

}

//static constructor

**static MyDb()**

{

MyConnectionString = GetConnectionStrings();

}

**internal static SqlConnection**

**SqlConnection(string AppConfigEntry = "ConnectionString")**

{

SqlConnection con = new

SqlConnection(GetConnectionStrings(AppConfigEntry));

return con;

}

**internal static string**

**GetConnectionStrings(string AppConfigEntry = "ConnectionString")**

{

**// System.Configuration.dll vom .NET Ordner als Verweis hinzufügen**

**// C:\Program Files (x86)\Reference**

**// Assemblies\Microsoft\Framework\.NETFramework\v4.7.2\System.Configuration.dll**

string connection = "";

try

{

var aConnections =

System.Configuration.ConfigurationManager.

ConnectionStrings[AppConfigEntry];

if (aConnections != null)

{

connection = aConnections.ToString();

}

}

catch (Exception ex)

{

}

/\*

<configuration>

<connectionStrings>

<add name="ConnectionString" connectionString="Data Source=DESIGNER-PC\SQLEXPRESS;Initial Catalog=MyDb;Persist Security Info=True;User ID=user01;Password=www" providerName="System.Data.SqlClient" />

</connectionStrings>

</configuration>

\*/

return connection;

}

}

DataAccess / PersonalDal.cs

PersonalDal.cs ist für Datenzugriffe (Lesen und Schreiben) in der Tabelle „Personal“ zuständig.

internal static List<Personal> GetAll()

internal static Personal Get(int id)

internal static int Add(Personal eintrag)

internal static int Update(Personal eintrag)

internal static int Delete(Personal eintrag)

**public static class PersonalDal**

**{**

static SqlConnection con;

static PersonalDal()

{

**con = MyDb.SqlConnection("SqlServer");**

con.Open();

}

**............. /\* HIER KOMMEN ALLE METHODEN \*/**

**}**

**internal static List<Personal> GetAll()**

{

List<Personal> list = null;

**string sql = $@"SELECT \* FROM Personal";**

using (SqlCommand cmd = new SqlCommand(sql, con))

{

SqlDataReader rd = cmd.ExecuteReader();

while (rd.Read())

{

Personal eintrag = new Personal();

eintrag.PersonId = int.Parse(rd["PersonId"].ToString());

eintrag.Nachname = rd["Nachname"].ToString();

eintrag.Vorname = rd["Vorname"].ToString();

eintrag.GebDatum = DateTime.Parse(rd["GebDatum"].ToString());

eintrag.GebOrt = rd["GebOrt"].ToString();

eintrag.Abgeschlossen = bool.Parse(rd["Abgeschlossen"].ToString());

eintrag.WochenStunden = int.Parse(rd["WochenStunden"].ToString());

// Warum die Felder einzeln auslesen?

// Vorteil: Mögliche Fehlerquellen können schneller gefunden werden.

list.Add(eintrag);

}

rd.Close();

}

return list;

}

**internal static Personal Get(int id)**

{

Personal eintrag = null;

**string sql = $@"SELECT \* FROM Personal WHERE PersonId=@PersonId";**

using (SqlCommand cmd = new SqlCommand(sql, con))

{

cmd.Parameters.AddWithValue("@PersonId", id);

SqlDataReader rd = cmd.ExecuteReader();

if(rd.Read())

{

eintrag = new Personal();

eintrag.PersonId = int.Parse(rd["PersonId"].ToString());

eintrag.Nachname = rd["Nachname"].ToString();

eintrag.Vorname = rd["Vorname"].ToString();

eintrag.GebDatum = DateTime.Parse(rd["GebDatum"].ToString());

eintrag.GebOrt = rd["GebOrt"].ToString();

eintrag.Abgeschlossen = bool.Parse(rd["Abgeschlossen"].ToString());

eintrag.WochenStunden = int.Parse(rd["WochenStunden"].ToString());

}

rd.Close();

}

return eintrag;

}

**internal static int Add(Personal eintrag)**

{

int id = 0;

string sql = $@"

**INSERT INTO Personal(Vorname,Nachname,GebDatum,GebOrt,Abgeschlossen,WochenStunden)**

**output INSERTED.PersonId** **VALUES(@Vorname,@Nachname,@GebDatum,@GebOrt,@Abgeschlossen,@WochenStunden)**

";

using (SqlCommand cmd = new SqlCommand(sql, con))

{

cmd.Parameters.AddWithValue("@Vorname", eintrag.Vorname);

cmd.Parameters.AddWithValue("@Nachname", eintrag.Nachname);

cmd.Parameters.AddWithValue("@GebDatum",

eintrag.GebDatum.ToString("d"));

cmd.Parameters.AddWithValue("@GebOrt", eintrag.GebOrt);

cmd.Parameters.AddWithValue("@Abgeschlossen", eintrag.Abgeschlossen);

cmd.Parameters.AddWithValue("@WochenStunden", eintrag.WochenStunden);

id = (int)cmd.ExecuteScalar();

}

return id; // ID des letzten Eintrages

}

**internal static int Update(Personal eintrag)**

{

int anzahl = -1;

**string sql = $@"UPDATE Personal SET**

**Vorname=@Vorname, Nachname=@Nachname, GebDatum=@GebDatum, GebOrt=@GebOrt,**

**Abgeschlossen=@Abgeschlossen, WochenStunden=@WochenStunden**

**WHERE PersonId=@PersonId**

";

using (SqlCommand cmd = new SqlCommand(sql, con))

{

cmd.Parameters.AddWithValue("@PersonId", eintrag.PersonId);

cmd.Parameters.AddWithValue("@Vorname", eintrag.Vorname);

cmd.Parameters.AddWithValue("@Nachname", eintrag.Nachname);

cmd.Parameters.AddWithValue("@GebDatum",

eintrag.GebDatum.ToString("d"));

cmd.Parameters.AddWithValue("@GebOrt", eintrag.GebOrt);

cmd.Parameters.AddWithValue("@Abgeschlossen", eintrag.Abgeschlossen);

cmd.Parameters.AddWithValue("@WochenStunden", eintrag.WochenStunden);

anzahl = cmd.ExecuteNonQuery();

}

return anzahl;

}

**internal static int Delete(Personal eintrag)**

{

int anzahl = -1;

string sql = $@"DELETE Personal WHERE PersonId=@PersonId;";

using (SqlCommand cmd = new SqlCommand(sql, con))

{

cmd.Parameters.AddWithValue("@PersonId", eintrag.PersonId);

anzahl = cmd.ExecuteNonQuery();

}

return anzahl;

}

BusinessLogic / PersonalManager.cs

**public static class PersonalManager**

{

static PersonalManager()

{

}

**............. /\* HIER KOMMEN ALLE METHODEN \*/**

}

**public static List<Personal> GetAll(Expression<Func<Personal, bool>> filter = null)**

{

List<Personal> list = new List<Personal>();

list = PersonalDal.GetAll();

if (filter != null)

list = list.**AsQueryable().**Where(filter).ToList();

return list;

}

“**Expression<Func<Personal, bool>>**” ermöglicht die Übernahme der Lambda-Ausdrücke.

**public static Personal Get(int id)**

{

Personal eintrag = null;

eintrag = PersonalDal.Get(id);

return eintrag;

}

**public static int Add(Personal eintrag)**

{

int id = 0;

id = PersonalDal.Add(eintrag);

return id;

}

**public static int Update(Personal eintrag)**

{

int anzahl = -1;

anzahl = PersonalDal.Update(eintrag);

return anzahl;

}

**public static int Delete(int id)**

{

Personal eintrag = Get(id);

int anzahl = Delete(eintrag);

return anzahl;

}

**public static int Delete(Personal eintrag)**

{

int anzahl = -1;

anzahl = PersonalDal.Delete(eintrag);

return anzahl;

}

**public static List<PersonalView>**

**GetAllPersonalView(Expression<Func<PersonalView, bool>> filter = null)**

{

List<PersonalView> list = new List<PersonalView>();

List<Personal> db = PersonalDal.GetAll();

foreach (var eintrag in db)

{

PersonalView neu = new PersonalView();

neu.PersonId = eintrag.PersonId;

neu.Name = eintrag.Vorname + " " + eintrag.Nachname;

neu.GebJahr = eintrag.GebDatum.Year;

list.Add(neu);

}

if (filter != null)

list = list.AsQueryable().Where(filter).ToList();

return list;

}

Controllers / Form1.cs Codes

**public partial class Form1 : Form**

{

public Form1()

{

InitializeComponent();

}

**private void Form1\_Load(object sender, EventArgs e)**

{

// DataGird Aufbereiten

dgvPersonal.SelectionMode = DataGridViewSelectionMode.FullRowSelect;

dgvPersonal.ReadOnly = true;

dgvPersonal.SelectionChanged += DgvPersonal\_SelectionChanged;

// Suche beim Tippen

txtSuchen.KeyUp += TxtSuchen\_KeyUp;

**GridViewPersonalLaden();**

//PersonalViewLaden();

}

**............. /\* HIER KOMMEN ALLE METHODEN \*/**

}

**private void TxtSuchen\_KeyUp(object sender, KeyEventArgs e)**

{

GridViewPersonalLaden();

}

**private void DgvPersonal\_SelectionChanged(object sender, EventArgs e)**

{

int aktuelleZeilenNummer = dgvPersonal.CurrentCell.RowIndex;

DataGridViewRow aktuelleZeile = dgvPersonal.Rows[aktuelleZeilenNummer];

int id = int.Parse(aktuelleZeile.Cells[0].Value.ToString());

FormularPersonalLaden(id);

}

**private void btnSpeichern\_Click(object sender, EventArgs e)**

{

if (txtPersonId.Text != "")

{

**PersonalAktualisieren();**

}

else

{

**PersonalHinzufuegen();**

}

}

**private void btnLoeschen\_Click(object sender, EventArgs e)**

{

DialogResult dialog = new DialogResult();

dialog = MessageBox.Show("Sind Sie Sicher?", "LÖSCHEN",

MessageBoxButtons.YesNo);

if (dialog == DialogResult.Yes)

{

int id = int.Parse(txtPersonId.Text);

// Löschen

**PersonalManager.Delete(id);**

MessageBox.Show("OK. Kurs gelöscht!");

**GridViewPersonalLaden();**

}

else

{

MessageBox.Show("Nicht gelöscht");

}

}

**private void btnNeu\_Click(object sender, EventArgs e)**

{

FormularPersonalNeu();

}

**private void btnSuchen\_Click(object sender, EventArgs e)**

{

GridViewPersonalLaden();

}

**private void btnXSuchen\_Click(object sender, EventArgs e)**

{

txtSuchen.Text = "";

GridViewPersonalLaden();

}

**private void PersonalViewLaden()**

{

List<PersonalView> list = PersonalManager.GetAllPersonalView();

dgvPersonal.DataSource = list;

}

**private void GridViewPersonalLaden()**

{

List<Personal> list = null;

string suchBegrief = txtSuchen.Text;

list = PersonalManager.GetAll();

if (suchBegrief.Length > 0)

{

list = list.Where(x =>

x.Vorname.ToLower().Contains(suchBegrief.ToLower()) ||

x.Nachname.ToLower().Contains(suchBegrief.ToLower()) ||

x.GebOrt.ToLower().Contains(suchBegrief.ToLower())

).ToList();

}

dgvPersonal.DataSource = list;

}

**private void FormularPersonalLaden(int id)**

{

Personal eintrag = PersonalManager.Get(id);

if (eintrag != null)

{

txtPersonId.Text = eintrag.PersonId.ToString();

txtVorname.Text = eintrag.Vorname.ToString();

txtNachname.Text = eintrag.Nachname.ToString();

txtWochenStd.Text = eintrag.WochenStunden.ToString();

cbxAbgeschlossen.Checked = eintrag.Abgeschlossen;

dtpGebDatum.Value = eintrag.GebDatum;

txtGebOrt.Text = eintrag.GebOrt;

}

}

**private void PersonalAktualisieren()**

{

int id = int.Parse(txtPersonId.Text);

Personal eintrag = PersonalManager.Get(id);

if (eintrag!=null)

{

eintrag.Vorname = txtVorname.Text;

eintrag.Nachname = txtNachname.Text;

eintrag.GebDatum = dtpGebDatum.Value;

eintrag.GebOrt = txtGebOrt.Text;

eintrag.WochenStunden = int.Parse(txtWochenStd.Text);

eintrag.Abgeschlossen = cbxAbgeschlossen.Checked;

int anzahl = PersonalManager.Update(eintrag);

**GridViewPersonalLaden();**

**WinFormTools.GridViewZeileAuswaehlen(dgvPersonal, 0,**

**id.ToString());**

**FormularPersonalLaden(id);**

}

}

**private void PersonalHinzufuegen()**

{

Personal eintrag = new Personal();

eintrag.Vorname = txtVorname.Text;

eintrag.Nachname = txtNachname.Text;

eintrag.GebDatum = dtpGebDatum.Value;

eintrag.GebOrt = txtGebOrt.Text;

eintrag.WochenStunden = int.Parse(txtWochenStd.Text);

eintrag.Abgeschlossen = cbxAbgeschlossen.Checked;

int id = PersonalManager.Add(eintrag);

txtPersonId.Text = id.ToString();

GridViewPersonalLaden();

WinFormTools.GridViewZeileAuswaehlen(dgvPersonal, 0, id.ToString());

FormularPersonalLaden(id);

}

**private void FormularPersonalNeu()**

{

txtPersonId.Text = "";

txtVorname.Text = "";

txtNachname.Text = "";

txtWochenStd.Text = "0";

dtpGebDatum.Value = DateTime.Now.Date;

txtGebOrt.Text = "";

cbxAbgeschlossen.Checked = false;

}

Controllers / WinFormTools.cs

**public class WinFormTools**

{

**internal static void GridViewZeileAuswaehlen(DataGridView dgw,**

**int spaltenNr, string id)**

{

dgw.ClearSelection();

for (int zeileNr = 0; zeileNr < dgw.Rows.Count; zeileNr++)

{

DataGridViewRow akuelleZeile = dgw.Rows[zeileNr];

if (akuelleZeile.Cells[spaltenNr].Value.ToString() == id)

{

akuelleZeile.Selected = true;

akuelleZeile.Cells[0].Selected = true;

dgw.CurrentCell = dgw[0, zeileNr]; // Set Fokus

}

}

}

}

App.config

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

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />

</startup>

<connectionStrings>

<add name="SqlServer" connectionString="Data Source=DESIGNER-PC\SQLEXPRESS;Initial Catalog=DbBFI1;Persist Security Info=True;User ID=sa;Password=www" providerName="System.Data.SqlClient" />

</connectionStrings>

</configuration>