T2000 – Praxis II

Conception and implementation of a web based time tracking solution

Studiengang Angewandte Informatik

Dualen Hochschule Baden Württemberg Stuttgart

von

Matthias Bidlingmeyer

Matrikelnummer:	1613581
Ausbildungsfirma:	Hewlett-Packard Enterprise
Betreuer der Ausbildungsfirma:	Sven Henselmann
	sven.henselmann@hpe.com

Abstract

This paper describes the process of the conception and implementation of a web based time tracking solution.

Therefore previous approaches to the problem are being analyzed. As a result, ASP.NET Web Forms is found to be most suitable for the implementation. Because this web development framework strongly defines the structure of the application, this technique is being explained thoroughly.

The next part takes a deeper look at the underlying business processes of the tool and reduces it to strictly defined use cases. In the last step before the implementation, a database schema is generated out of the use cases.

Finally, a strategy for migrating to the new tool is presented, followed by a summary and a lookout into the future.

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Falls sowohl eine gedruckte als auch elektronische Fassung abgegeben wurde, versichere ich zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt

Ort, Datum

Unterschrift Matthias Bidlingmeyer

Contents

Acronyms

ACID	Atomicity Consistency Isolation Durability
API	Application Programming Interface
ASP	Active Server Pages
CR	Change Request
CRUD	Create Read Update Delete
DBMS	Database Management System
GUI	Graphical User Interface
HPE	Hewlett Packard Enterprise
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IT	Information Technology
ITIL	Information Technology Infrastructure Library
JSON	JavaScript Object Notation
MEAN	MongoDB Express Angular.js Node.js
MVC	Model View Controller
RAD	Rapid Application Development
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
VB	Visual Basic
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

List of Figures

List of Tables

List of Code

1 Introduction

Nowadays, practically every company is dependent on some kind of IT, be it for the firm's internal infrastructure, as tools for working or as means of communication. The development, installation and maintenance of such systems is not only expensive, it also requires a certain amount of know-how.

That's the reason why most companies outsource their required IT. This is the business segment the department for which the time tracking project is being developed for focuses on.

The IT business is never a static one, so a product has to be continually maintained and adapted to changes in order to keep working. That's why most contracts with the customer run over an extended period of time. In the scope of the ITIL there exist so called Change Requests (short CR – also termed projects in this paper) which regulate this dynamic nature of the business.

In order to invoice the efforts performed on these Change Requests to the customer, detailed performance records have to be created. These records contain a detailed listing of how much time has been worked by which employee on which project. With many projects being worked on by many employees, the number of performance records which have to be issued per month is rather high.

A manual writing of these records entails two major problems. Firstly, the process involves a huge amount of data processing, which inevitably leads to making mistakes. Secondly, the process is particularly time consuming, i.e. expensive.

That's why HPE has a keen interest in automating this routine as far as possible.

One of the approaches to this, named *Project Tandem* is presented in this paper.

Project Tandem is a tool which allows to manage the Change Requests, enables employees to track the amount of time that is being worked on these CRs, and provides an automatic generation of the performance records.

2 History

There have been two previous attempts in bringing a complete solution to this problem. The first one was built on the base of Microsoft SharePoint. For various reasons which will be elaborated later, a second attempt was being made, this time using a combination of an online SharePoint and a desktop application developed in C#. Again, this approach has been found unsuitable, and therefore the development was canceled. As a conclusion to these failed efforts, a third attempt in finding a proper solution was made. It is this third attempt that this paper covers. But first, the different aspects of the two previous solutions and why they were found unfitting will be covered.

2.1 SharePoint solution

In order to understand the problems that come with the SharePoint solution, a quick overview over its capabilities is necessary. The purpose of SharePoint is to facilitate the creation of a customizable and flexible system for storing and analyzing information. The core to this is made up of a combination of lists, much comparable to a Database Management System (DBMS). These lists have many options to expand their functionality, which often times resemble the range of functions found in Microsoft Excel. Furthermore, SharePoints can be easily combined and integrated with many other technologies by Microsoft.

It is this versatility and the user-friendliness, which makes it such a popular tool for many companies, including Hewlett-Packard Enterprise. By using this software, the wide variety of different departments can easily build their own administration framework. (Noel & Spence, 2010)

However, HPE does not provide the complete functionality of SharePoint to its employees. This might be due to security or license issues. These extended functions include SharePoint Apps, SharePoint AddIns and JavaScript support.

This is why HPE's SharePoint often reaches its limits when trying to project a use case to it. This then often results in a lack of functionality, where SharePoint only handles some parts of a desired spectrum of features. This also leads to many weak workarounds, increasing the complexity of the SharePoint and thereby decreasing the consistency and integrity of the data.

In the case of project Tandem, this leads to the main problems, which are decentralized, redundant data, which has to be processed in several manual processes, resulting in work intensive and most notably unreliable output.

2.2 Desktop application

In an attempt to enhance the functionality of the SharePoint solution and eradicate the error-proneness, a desktop application was developed in the .NET-Framework with the programming language C#.

The centralized Database, out of the lack of an own server, has been implemented as a new SharePoint. This is not a good practice.

According to Microsoft, “if your business logic requires transactions, storing data in a database is preferable to using [SharePoint] lists.” (Microsoft, 2016) The issue at hand is that accessing and manipulating a SharePoint list via its API does not guarantee the ACID (Atomicity, Consistency, Isolation, and Durability) properties for this transaction.

Another problem arises with increasing complexity of the relations between different tables which, as we will see later in this paper, are indeed rather complex for project Tandem. “SharePoint lists are meant to store simple data structures. If you require a complex data model with intricate relationships, a database is more appropriate.” (Microsoft, 2016) As a fact, relationships can’t be modeled, and transactions can’t be queried by SQL. This makes it necessary for the desktop application to take care of complex queries, again resulting in an unreliable application.

Nevertheless, the desktop application has been implemented in large parts, with the main functionality working. However, a new problem arose when the developer had to be exchanged. While the code of the application was perfectly solid, it didn’t follow any established programming patterns, nor was it developed in a modular way. In an analysis conducted by the overtaking developer, it was concluded that the expense of developing a completely new application equaled the effort of finishing the existing approach.

This conclusion basically reset the whole project to zero.

2.3 Web application

While effectively all progress had been lost, this opened the opportunity to reassess the whole application in every aspect. It was also important to learn from the mistakes and problems in the previous approaches and technologies used. In the end, it was settled on a web application to be developed in the .NET framework, using web forms and VB.Net in combination with Microsoft SQL Server.

There are several reasons for making this decision.

The big advantage of web applications on desktop apps is the centralized nature of the web app. Not having to deal with the user's personal computer setup makes the solution much more reliable by removing the following problems.

Firstly, the web app does not require any specific libraries or other prerequisites installed on the pc. Moreover, it is not even bound to a specific operating system, and technically can be used from any device with internet access including smartphones and tablets.

Another big issue that comes with keeping the app running concerns the distribution of updates. While changes to a website can simply be updated on the server, being instantly available to all users, a desktop application requires a sophisticated updating procedure. Not only is this process of updating a very critical one, so is the task of handing out the software in the first place.

As a conclusion, it has been decided to develop project Tandem as a web application.

The next step is to choose the technology which should be used.

The Wikipedia article on the "comparison of notable web frameworks" currently lists over 100 different frameworks. (Wikipedia, 2016)

However this wide selection narrows down really quickly once the prospects of long term support inside the department is considered. The small team that is responsible for any technical development is specifically focused on one technology only, which is the Microsoft .NET Framework using Web Forms and SQL Server.

Although this is a relatively old technology, it is nevertheless vital to have a team which can provide support for this application in the future. After all, there can be no doubt that software support is at least as important as software development, as change is a fundamental part of today's information technology.

So in order to derive a solution that can be supported by the team to its full extent, these technologies have been agreed on. Even more, the website will use the same data access pattern which has been established by said team.

3 ASP.NET Web Forms

The first step towards implementing any application is to understand the technologies at hand. The term ASP.NET Web Forms can be split into three parts:

.NET

Microsoft's .NET framework provides a platform for Multilanguage development on Windows.

ASP.NET

The sub framework ASP.NET allows to run these .NET applications on an IIS Server. This enables theses apps to handle HTTP requests and responses.

ASP.NET Web Forms

Web Forms provides a specific toolset for developing a web application such as UI components, a stateful behavior and object-oriented programming.

This technology has been first introduced in 2002. At that time, the web was just accelerating in popularity. In order to allow desktop developers to easily switch to web programming, the ASP.NET Web Forms framework has been created. 'In effect, Web Forms is a giant abstraction layer aimed to deliver a classic event-driven GUI over the Web […] to make web development feel just the same as Windows Forms development.' (Sanderson, 2011)

3.1 The big picture

In order to understand the different aspects of the ASP.NET Web Forms technology, it's necessary to outline how these parts interact with each other, and which role they play in the bigger picture.

At the most basic level, there are two networked machines: the server which provides the website, and the client who views the site in his browser. As an underlying protocol, HTTP (Hypertext Transfer Protocol) is used, enabling the two parties to communicate on a common level.

3.1.1 The HTTP Request-Response Cycle

When a client navigates to a specific URL in his Browser, the URL is converted to an IP address by the Domain Name Service (DNS). Knowing the address of the server, the browser opens a connection to port 80 of the server and sends an HTTP request to it. The server receives this request, possibly containing parameters such as a textbox value etc., and builds an appropriate HTTP response.

This is where ASP.NET comes into play. Among other technologies such as PHP, Java Server Pages or the MEAN Stack, this server side software evaluates the request and serves a response accordingly.

On the client side, the browser then renders the HTTP, completing the one HTTP Request-Response Cycle. Figure 1 illustrates this process.

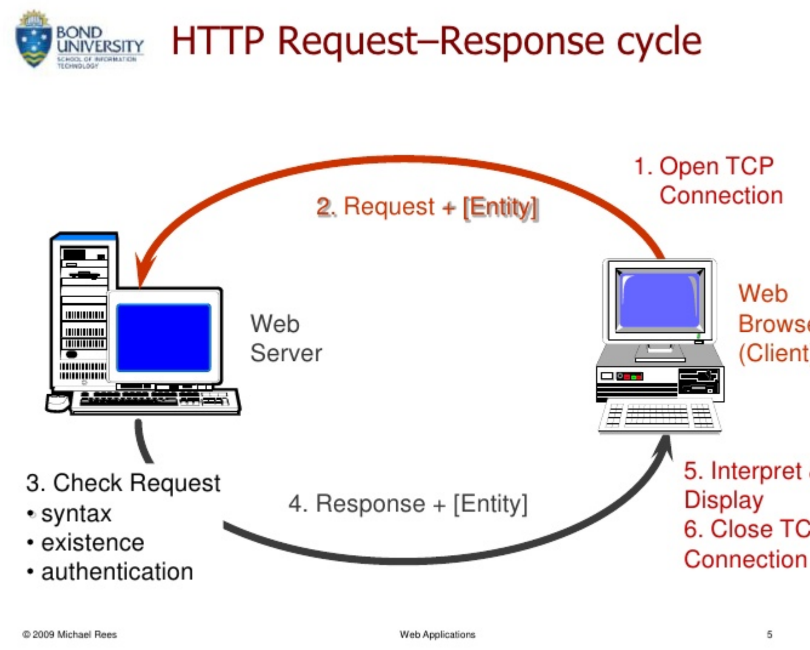


Figure 1. The HTTP Request-Response Cycle (Rees, 2009)

3.1.2 Regular Postbacks

The Request-Response Cycle is not only triggered by navigating to an URL in a browser, it is also being initiated by various interactions between the user and the site content.

Usually such behavior is achieved by using the HTML ‘form’ tag.

```
<form id="exampleForm"
   action="http://localhost/Example/MyExampleAspNetPage.aspx" method="GET">
   <input id="PostBackButton" type="submit" value="This is a GET request"/>
   ...
</form>
```

Code 1. HTML Form tag for postbacks

Code 1 is a simple demonstration of the form tag. The action attribute defines the target on the server for the request. The method attribute can be set to either GET or POST. The main difference here is that while with a GET request all parameters are parsed and appended to the URL, with a POST request the data is being sent in the background, invisible to the client. Furthermore, many browsers have a limited length for the GET request. In this case, the form data will be submitted by clicking the input of type submit. The server can than tailor a HTML site and send it as a response to this specific request.

3.1.3 Postbacks in Web Forms

When using ASP.NET, postbacks are handled differently. The framework adds an extra layer on top of the HTML protocol. This layer enables the developer to use an event-driven approach to handle user interaction. That way, any Web Form control can be associated with a piece of server side code while the ASP.NET runtime takes care of the request-response cycle. The backend can be programmed using any .NET language, such as C#, VB.NET or F#. This paper will focus on VB.Net, as it is the language used in project Tandem. (Troelsen & Japikse, 2015)

3.2 The Web Forms API

Now that the connection between client and server, HTML and VB.Net have been outlined, a more detailed explanation of the Web Forms API can be conducted.

API Overview

- Presentation logic and business logic are separated by a model called *code-behind*
- Any .NET language can be used for the *code-behind*. For faster execution, the code is compiled into .dll files
- There are various UI components called Web Forms controls. This model is very similar to ASP.NET's approach to desktop applications
- Master pages allows subpages to have a common UI frame
- The backend is completely object-oriented
- Partial postbacks enable an Ajax like updating of different parts of a site
- The *web.config* file can be used to configure the web application

There are many more facets to the Web Forms API, however, this paper will be restricted to the topics relevant for project Tandem. The following chapters will examine the listed aspects and their implications further.

3.2.1 Code-behind

A Web Forms page consists of two separated parts, the presentation logic and the business logic. This model is termed *code-behind*. In most cases this separation is achieved by having an .aspx file for the UI and a VB.Net file for the logic linked to it.

As will be elaborated in chapter 3.4, this separation is only superficial and leads to many problems.

It is important to note that this does not mean that the presentation logic is executed on the client nor the business logic on the server. Moreover, the next chapter will show that quite the opposite is true. (Troelsen & Japikse, 2015)

3.2.2 Web Form controls

One of the features provided by the ASP.NET framework are the Web Form controls. Their syntax combines the markup nature of HTML and the semantics of the Windows Presentation Foundation (WPF). These are used to build the user interface.

Web Form controls are used as HTML elements in the presentation logic of a web page. However, the important characteristic of these controls is that they are executed on the server. The following example illustrates the underlying mechanism:

Code 2 shows a simple Web Forms control, a button.

```
<asp:Button ID="btnMyButton" runat="server" Text="Button" BorderColor="Blue"
   BorderStyle="Solid" BorderWidth="5px" />
```

Code 2. Button Web Forms control

When the site containing this button is requested, the server generates the following HTML code and sends it to the client:

```
<input type="submit" name="btnMyButton" value="Button" id="btnMyButton"
   style="border-color:Blue; border-width:5px; border-style:Solid;" />
```

Code 3. Generated HTML to the ASP Button

For more complex UI elements, this strategy simplifies the syntax significantly as compared to HTML. More importantly, it allows for a native correlation between the control and the code-behind. (Troelsen & Japikse, 2015)

3.2.3 Managing states in Web Forms

The state in the context of applications refers to the information that is constantly exchanged between the user and the machine. For example, this information includes shopping cart items, user information, preferences etc.

3.2.3.1 HTML, a stateless protocol

An important property of the HTML protocol is its stateless nature. Recalling the request-response cycle of the protocol, every time a response is being sent from the server to the client, the previous interaction is completely forgotten about. This is a huge difference when compared to a desktop application, where the state is usually dragged through the whole lifetime of the application. (Troelsen & Japikse, 2015)

3.2.3.2 The Web Forms ViewState

With HTML being a stateless protocol, it is up to the developer to adopt a stateful design. The ASP.NET framework provides several options for this:

- Use ASP.NET ViewState
- Use ASP.NET control state
- Define application-level data
- Define session-level data
- Use the cache object
- Define cookie data

The most important ones will be explained in the following chapters.

3.2.3.3 ASP.NET ViewState

When submitting data into a form on a web page, the server would have to manually repopulate the fields when building the response in order for the user not to lose his input. The ASP.NET ViewState provides a remedy. Using a hidden input field, the state is contained throughout the HTML request-respnse cycle. This process is taken care of entirely by the framework, without the developer having anything to do.

In the background, as mentioned, a hidden input field is used to hold the information as a Base64 encoded String:

```
<input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
        value="/wEdAAQx3DSib26RI/NSNPQTRbIffuWr/p/8eyHPp3MiaQ8WHO8=" />
```

Code 4. The Web Forms ViewState in action

3.2.3.4 Application-level and Session-level data

The *HttpApplicationState* class provided by ASP.NET is responsible for holding information relevant to the entirety of the application, shared by all users. When this information changes at any point, every user will be supplied with the new state after their next postback.

The application state is held in the memory as long as the application is running on the server.

On the other hand, there is the *HttpSessionState*, which holds data for the current user only. (Troelsen & Japikse, 2015)

Figure 2 illustrates this behavior.

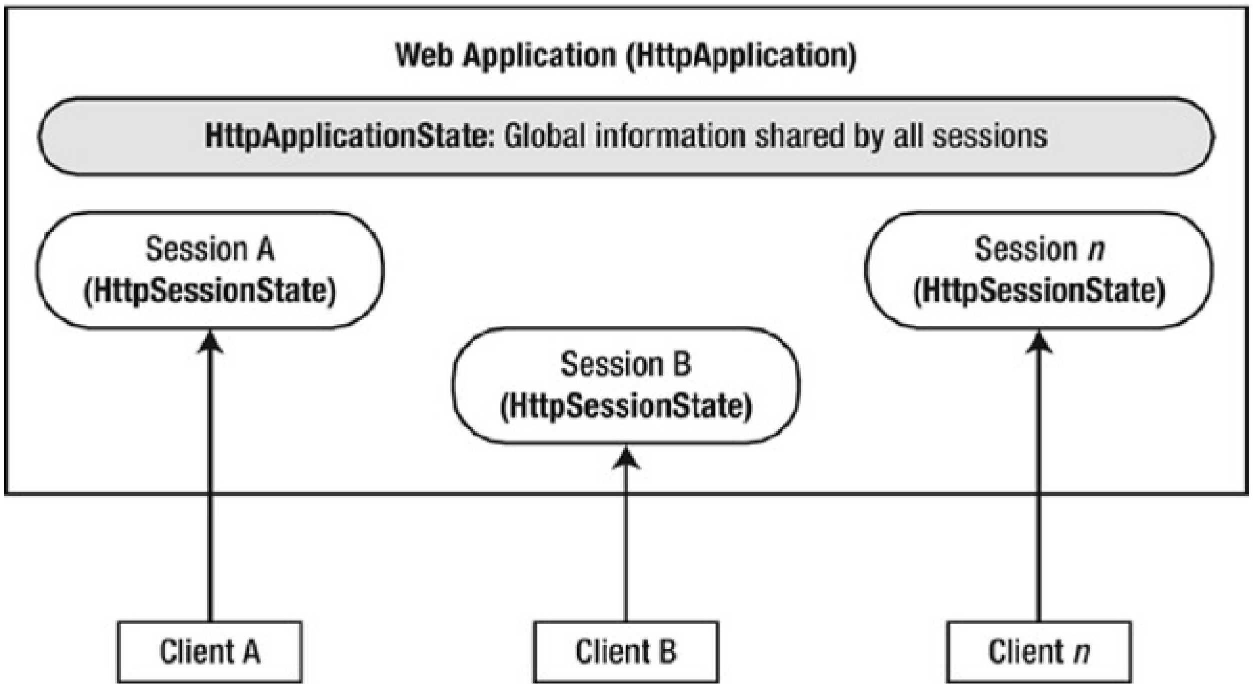

Web Application (HttpApplication)
HttpApplicationState: Global information shared by all sessions
Session A (HttpSessionState)
Session B (HttpSessionState)
Session n (HttpSessionState)
Client A
Client B
Client n

Figure 2. Difference between application and session state

The usage of these different types of states is fairly easy:

```
Application["AvailableContact"] = "Justus";
Application["SalesActive"] = False;
Application["GlobalHighscore"] = 900392;

Session["UserName"] = "Justin";

Session["LocalHighscore"] = 16278;
```

Code 5.Usage of the application and the session state

3.2.3.5 The Web Forms Cache

Another way to handle application-level data comes in form of the Web Forms cache. This cache allows to store data for a limited period of time, as opposed to the lifetime of the *HttpApplicationState*.

3.2.3.6 Cookies

The last state technique that will be discussed here are cookies. Unlike the previous attempts in preserving information, cookies are stored on the client and not on the server. Cookies are small text files containing information which belongs to a specific website. When the user navigates to this website, the browser appends the cookie to the request. The server can than read this data and act accordingly. The server can also change the cookie's content or send a new one to the browser, which then handles the storing of them.

ASP.NET knows two different kinds of cookies. Persisting and temporary cookies. While persisting cookies are stored to the client's hard-drive, temporary cookies are only cached by the browser until it is being closed by the user.

The *HttpCookie* class provides a server side representation of a cookie. That is a key/value pair, which can be added to *Response.Cookies* collection. It is this collection that will be sent to the browser via the HTML header. (Troelsen & Japikse, 2015)

A temporary cookie can be created as follows:

```
HttpCookie myCookie = new
     HttpCookie(“CookieName”,”SomeValue”);
Response.Cookies.Add(myCookie);
```

Code 6. Adding a temporary cookie to the HTML response

In order to create a persistant cookie, an expire date has to be supplied:

```
HttpCookie myCookie = new
     HttpCookie(“CookieName”,”SomeValue”);
theCookie.Expires = DateTime.Now.AddMonths(5);
Response.Cookies.Add(myCookie);
```

Code 7. Adding a persitant cookie to the HTML response

3.2.4 The Web Forms Page Life Cycle

Now that the state handling has been discussed, a deeper look into the business logic will follow. In order to understand how the VB.Net code is structured, and how the framework’s underlying logic is built, the so called Web Forms Page Life Cycle has to be considered.

Every time a postback occurs, this life cycle is being triggered. Understanding this cycle allows the developer to handle everything in the right place at the right time. In general terms, the page goes through the stages outlined in the following table

Table 1. Web Forms Page Life Cycle

Stage	**Description**
Page request	The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page.
Start	In the start stage, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. The page also sets the UICulture property.
Initialization	During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.
Load	During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.

Postback event handling	If the request is a postback, control event handlers are called. After that, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.)
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed.

These stages in the life cycle raise numerous events which can be handled by the developer’s code. In order to register functions to the events, they have to be referenced as an attribute in the Web Form controls. Some notable events are listed below:

Table 2. Web Forms Life Cycle Events

Page Event	**Typical Use**
PreInit	Raised after the start stage is complete and before the initialization stage begins.

	Check the IsPostBack property to determine whether this is the first time the page is being processed.
Init	Raised after all controls have been initialized. The Init event of individual controls occurs before the Init event of the page. Use this event to read or initialize control properties.
Load	The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page. Use the OnLoad event method to set properties in controls and to establish database connections.
Control events	Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
PreRender	Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls. Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.

SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.
Unload	Raised for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections. For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.

(Microsoft, 2016)

3.3 ASP.NET Web Forms, a conclusion

Many facets of the ASP.NET Web Forms framework have been discussed so far. As already mentioned, the framework has been created in order to facilitate desktop developers the switch to web development. That's why many features have been implemented to resemble the desktop's approach (e.g. the Web Forms controls which mimic the controls used in WPF). Another concept that Microsoft had in mind is the Rapid Application Development (RAD), which in the case of Web Forms allows to easily create many pages using a GUI builder, and having the logic strapped right into each page.

So while the framework has definitely achieved its intentions, after 15 years of web development, the techniques used are outdated. That's the reason why the newest version of the ASP.NET framework comes without Web Forms.

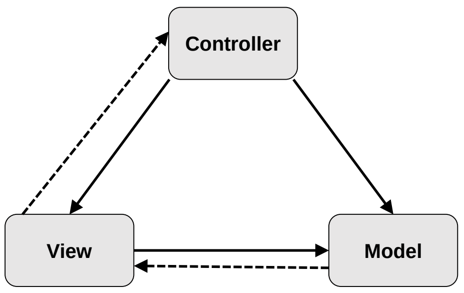
In order to understand what's unfavorable in Web Forms, it is best to show how other techniques differ to it. That's what the following chapters are all about. (Ciliberti, 2013)

3.4 Web Forms vs MVC

The successor to Web Forms in the ASP.NET framework is MVC, which stands for Model-View-Controller. The MVC model is a widely spread approach to web development, used by many other frameworks as well.

3.4.1 Model-View-Controller

An illustration of the MVC approach will help explain the technique at hand.



Code 8. MVC model

- Model: The models represent data. This can be achieved in many ways, and mostly involves a connection the database.
- View: A view is the visual representation of a model. The view always represents the state of the model, displaying the information and allowing the user to modify the data.
- Controller: The controller is responsible for linking the user input to the correct view and model.

An example process flow of an MVC application goes as follows:

- The user interacts with a view
- The view forwards the input to its controller
- The controller processes the user input
- The controller passes data to a model
- The model processes the data
- The controller invokes a new view
- The new view acquires its data from the model
- The view is presented to the user

The big advantage of MVC is the separation of concerns. The UI (view), the business logic (controller) and the data access (model) are completely separated and therefore interchangeable.

With a basic understanding of MVC in mind, a comparison to Web Forms can be drawn, exposing many weaknesses of Web Forms.

3.4.2 View based vs Action based solution

Web Forms pursues a view based approach for its web pages. This means that the user invokes a request to a view, which then triggers the page life cycle. The whole life cycle is executed and the appropriate events invoked. Eventually, a result HTML will be constructed and sent back as the response.

However when reconsidering the HTTP protocol, it becomes clear that the protocol has a completely different underlying logic. An action (GET, POST, PUT, DELETE, etc.) is sent to the server and a view is expected as response. This exactly the behavior that is represented by MVC. (koirala, 2014)

In MVC, the HTML request is analyzed by the controller, which then invokes the appropriate view with the appropriate data (model).

Figure 3 illustrates these two different behaviors.

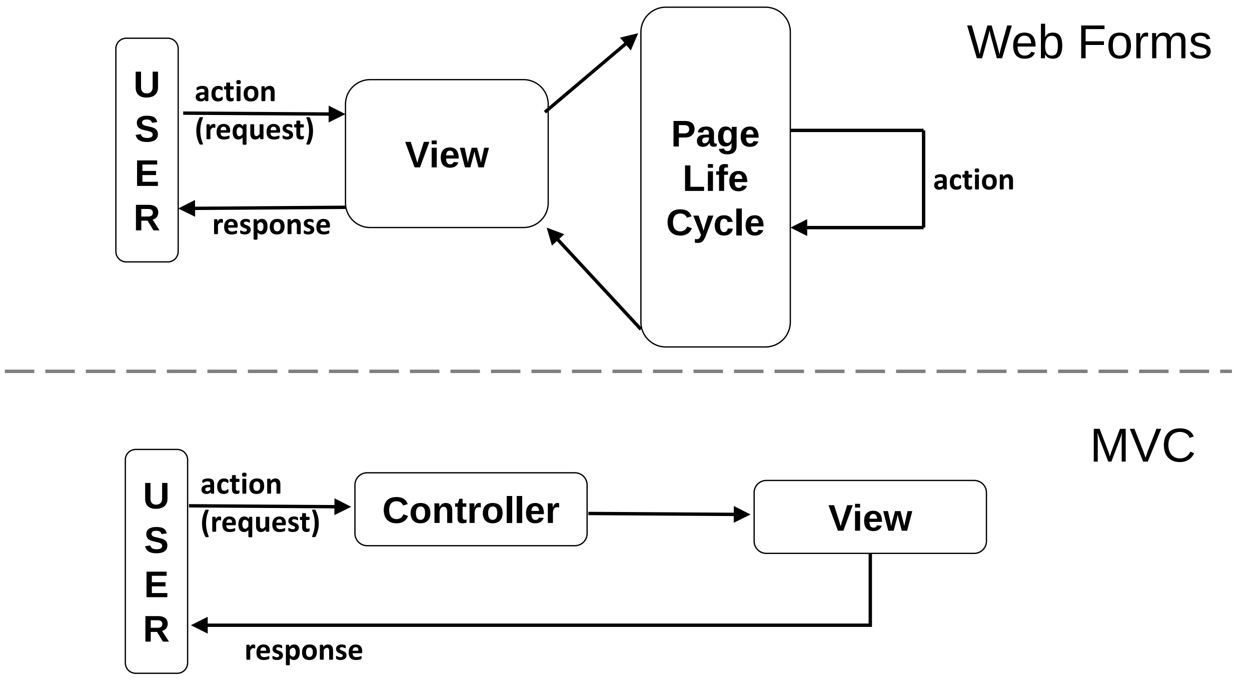


Figure 3. Web Forms View based vs MVC Action based

3.4.3 Tight coupling between view and business logic

Another issue that comes alongside with the architecture of the Web Forms framework is the tight coupling between the view and the business logic.

As explained in the chapter about the code-behind model, every view file (.aspx) is linked directly to its logic file (.aspx.vb). This bond is so tight that the code can be used by the respective view only. This is due to the fact that the file contains all the event handlers belonging to a specific view.

Moreover, this leads to the code file being very large when compared to other layers of the project. Therefore the code is rather unreadable and expensive to maintain and support.

Looking at MVC, a whole new range of options presents itself. The separation of concerns allows for views to be completely detached from their logic. This enables to implement a popular use case, when there should be a different view for mobile clients than for desktop users. While the logic stays the same in the controller and the model, the controller simply refers to another view. (koirala, 2014)

How could this be achieved with Web Forms code-behind?

3.4.4 Response types in Web Forms

Also owed to the coupling between HTML and VB.NET, it becomes difficult to send responses different than HTML. Web Forms simply isn’t built for sending JSON or XML objects.

Again, the controller in MVC allows to send any available response type, depending on the input. (koirala, 2014)

3.4.5 Combining Views and Models

Using MVC, the controller can combine any view with any model. In Web Forms the view is responsible for making this decision and referencing to some other view. This results in an unstructured application with complex cross-references and scattered logic. (koirala, 2014)

3.4.6 Unit testing of a Web Forms page

The fact that the *System.Web.UI.Page* class, from which the code-behind has to inherit, has a lot of dependencies which are resolved at runtime, makes it nearly impossible to instantiate the class for the purpose of unit testing.

MVC doesn’t encounter this problem at all since the logic is coded into basic classes.

(koirala, 2014)

3.4.7 Collaborating with Web Forms

When developing a project using the MVC-model, it is easy to have different developers working on completely different aspects of the application. While a web designer can fully concentrate on drafting the UI, a programmer can take care of the code. Similarly, MVC facilitates the collaboration for whole teams. This is because all parts are strictly encapsulated so that the different parties can develop on top of defined interfaces.

3.4.8 Conclusion

Summed up in one sentence, Web Forms offers a RAD framework to quickly and easily build a website. MVC on the other hand provides a robust and sophisticated framework for developing a sustaining solution.

In the year 2016 however, there is practically no need for the concept of Web Forms. Other technologies such as Microsoft Sharepoint and LightSwitch enable people to build rather complex applications without the need for much expertise. Websites that exceed the capabilities of these systems are better off using newer technologies such as MVC.

For most teams, including the team that will support project Tandem, the hardest threshold to switching technologies is the substantial learning curve. In the business of software development, the only thing a customer knows for certain is the deadline. That’s why many teams prefer to continue using an old RAD technology. In the short run this might seem reasonable, looking ahead a few years usually renders this decision void. (Ciliberti, 2013)

4 Data access layer – GMT Framework

In project Tandem, ASP.NET provides the framework which supports the web layer of the application. The second part to the project is the access layer to the database.

Before the object oriented approach was widely adopted, programs were structured procedurally, ‘top to bottom’. Similarly, databases evolved alongside to this technique. So while the object oriented has developed, the databases stayed the same. This produced a growing gap between code and database, creating a need for some kind of interaction layer. Consequently, countless frameworks have been developed in order to close the gap.

For this purpose, the team which will support project Tandem has developed a proprietary framework, titled GMT Framework. The GMT framework delivers a complete abstraction layer between the Database and the business logic. This chapter will explain the inner workings of this framework and its role in the application.

4.1 GMT functionalities

The framework provides the following functionalities to the developer:

- Database tables and views are represented by VB.Net classes
- Manager classes provide the CRUD (Create, Read, Update, Delete) operations
- Allows for an object oriented approach to access data
- Web Forms views and code-behind for a complete administration of the database

4.2 GMT usage

Using the GMT framework is fairly easy. Tables and Views are represented by classes with the columns as attributes. The Manager classes are used for writing and reading from and to the database.

The following code illustrates the usage of the four CRUD operations with the framework.

Code 9. Retrieving an entry from database via the GMT framework

```
Private Function getEmployee() As cl_Employee
        'Create an instance of the Employee Manager class
        Dim myEmployeeManager As New cl_EmployeeManager

        'Specify the conditions for the employee
        Dim params As New Hashtable
        params.Add("employee_ID", 5)

        'Retrieve employee #5 from the database
        Dim myEmployee As cl_Employee = myEmployeeManager.ReadOne(params)

        Return myEmployee
End Function
```

The *params* Hash table specifies the conditions that the database entry has to fulfill. Passing this Hash table to the manager object's *ReadOne* function returns the database entry as object.

Code 10. Adding and deleting an entry in the database via the GMT framework

```
Private Sub addDeleteEmployee(name, surname)
        'Create an instance of the Employee Manager class
        Dim myEmployeeManager As New cl_EmployeeManager

        'Create the new employee as object
        Dim myNewEmployee As New cl_Employee
        myNewEmployee.name = name
        myNewEmployee.surname = surname

        'Write the object to the database using the manager object
        myEmployeeManager.Update(myNewEmployee)
        myEmployeeManager.Delete(myNewEmployee)
End Sub
```

By simply passing an employee object to the *Update* function of the manager, a database entry can be updated or created. Passing the object to the *Delete* function deletes the entry from the database.

4.3 GMT inner workings

The usage of the GMT framework is remarkably easy. This chapter will clarify how the framework functions internally. The generated administration view will be omitted because it is not essential to the framework. Figure 4 outlines the structure of the framework.

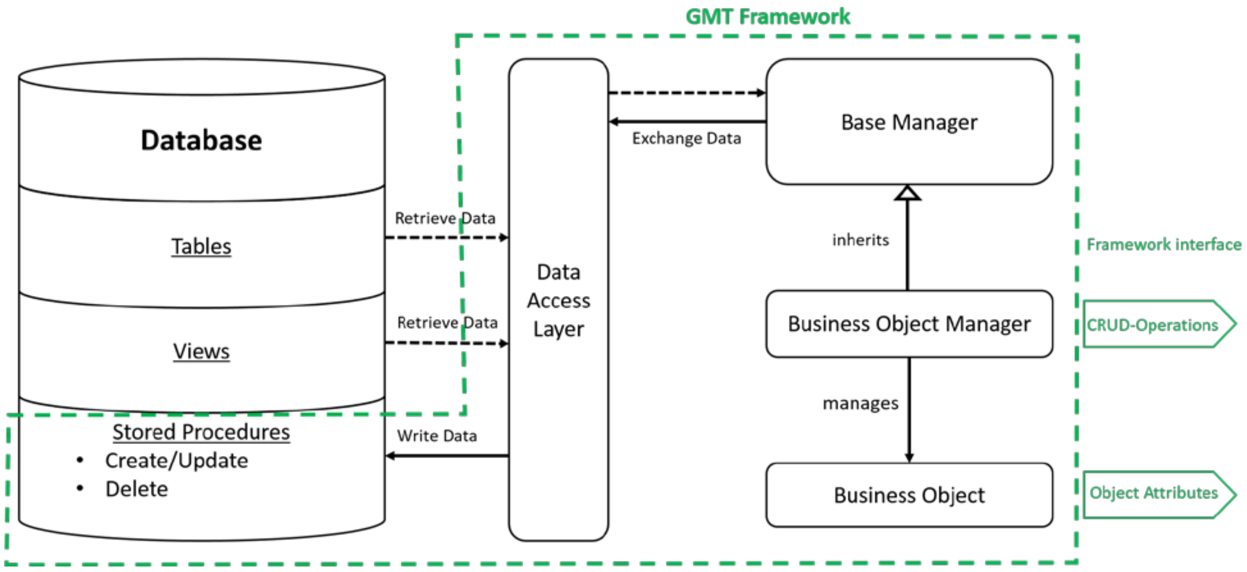


Figure 4. GMT framework structure

As the dotted green line in the above figure suggests, the GMT framework acts as an abstraction layer between the database and the application, such that only a small interface has to be considered by the developer (see previous chapter about the usage of the framework).

As can be seen, there are five different components to the framework:

- **Stored Procedures**: These procedures are responsible for the SQL logic of creating, updating and deleting entries.
- **The DataAccessLayer class**: This layer handles the communication with the database and provides interfaces to the Manager class.
- **The BaseManager class**: The Base Manager is a prototype class for the Business Object Manager. It defines which functions a manager class has to have. Additionally, it implements the communication to the Data Access Layer.
- **The business object manager class:** For every table and view a Manager class is generated to provide the CRUD operations to the developer.
- **The business object class:** For every table and view a Business Object is generated to allow an object oriented handling of database entries.

4.4 Conclusion

The GMT Framework offers a simple to use framework for translating the database schema into an object driven model – nothing more and nothing less.

Consequently, any transformation required on the data such as filtering, sorting or combining it have to be handled by the developer. That’s why an important part when working with the framework is to create views to take care of most of these transformations. For multifaceted applications however, this results in a huge number of views. The problem with this is that the developer quickly loses track of the views, often resulting in redundant or even unused views. Also, the reusability of those views is restricted.

A similar issue is the framework’s use of stored procedures.

These both issues belong to the same root problem: The framework is deeply intertwined with the database.

Eventually this means that a change to the database schema is accompanied by an extensive manual change to all the views and the stored procedures.

5 Implementation of Project Tandem

This chapter covers the implementation of the application. In particular the use cases and the database design will be presented. These two aspects have been deducted from the business process. In turn, supported by explanations, the reader can hereby reason the work flow in the department. The reason for this advance is that the use cases and the database provide the solid base for the implementation of the application, which is in the focus of this chapter.

5.1 Use Cases

The first step towards implementing any application is to analyze its use cases.

In the case of project Tandem, as has been shown, there have been previous attempts in bringing up a solution. That's why there already exist a number of use cases. Nevertheless, it is indispensable to reexamine the situation as it is very likely that requirements have changed or previous analysis are flawed.

First of all, the enquiry has brought up the following four user roles:

- Admin: is responsible for administrating the basic data in the tool
- Manager: of one or more projects has the functionality required to handle his employees/trackers
- Performance Record Creator: is allowed to create the performance records (for the customer)
- Tracker: is the employee who tracks his worked time

Furthermore, before the use cases can be understood, a few processes have to be explained separately:

User registration and administration

In order to distribute the workload that comes with adding a new user to the system, the registration process has been largely outsourced to the user. On the Login view, the user can request a new account by entering all the required data for an employee, including email, password and name. In the next step, the Admin can approve this account.

Budget calculations and agios

The Change Requests that are handled by the application have two budget dimensions to it.

On the one side, there is a specific price that the customer is willing to pay for the hours worked on a Change Request by an employee with a certain level of expertise. On the other hand, HPE has specific costs for these employees per hour.

These prices and costs vary depending on which day of the week and or time of the day efforts have been performed. In project Tandem, this variation of costs is called agio.

For a project, there is an overall cost and price budget, with their difference being the actual profit for HPE.

HPE uses outsourcing as well

HPE, just like its customers, is using outsourced resources. This means that when this paper talks about employees in the context of project Tandem, this is not exactly accurate. The people that work on the Change Requests can be either employed by HPE or by some external company.

5.1.1 Admin Role

The first important function of the Admin is his part on the user administration. He can approve a requested account and has to allocate a user role to it. Furthermore he can cause a user's password to be reset to a random one which will then be sent to the user's email.

The second responsibility of the Admin is to take care of the basic data in the system. This basic data includes employees, their suppliers and customers.

The Admin's third functionality is to connect the employee with their supplier.

It is being assumed that all these tasks don't have to be executed on a frequent level. Hence the tasks are assigned to the Admin.

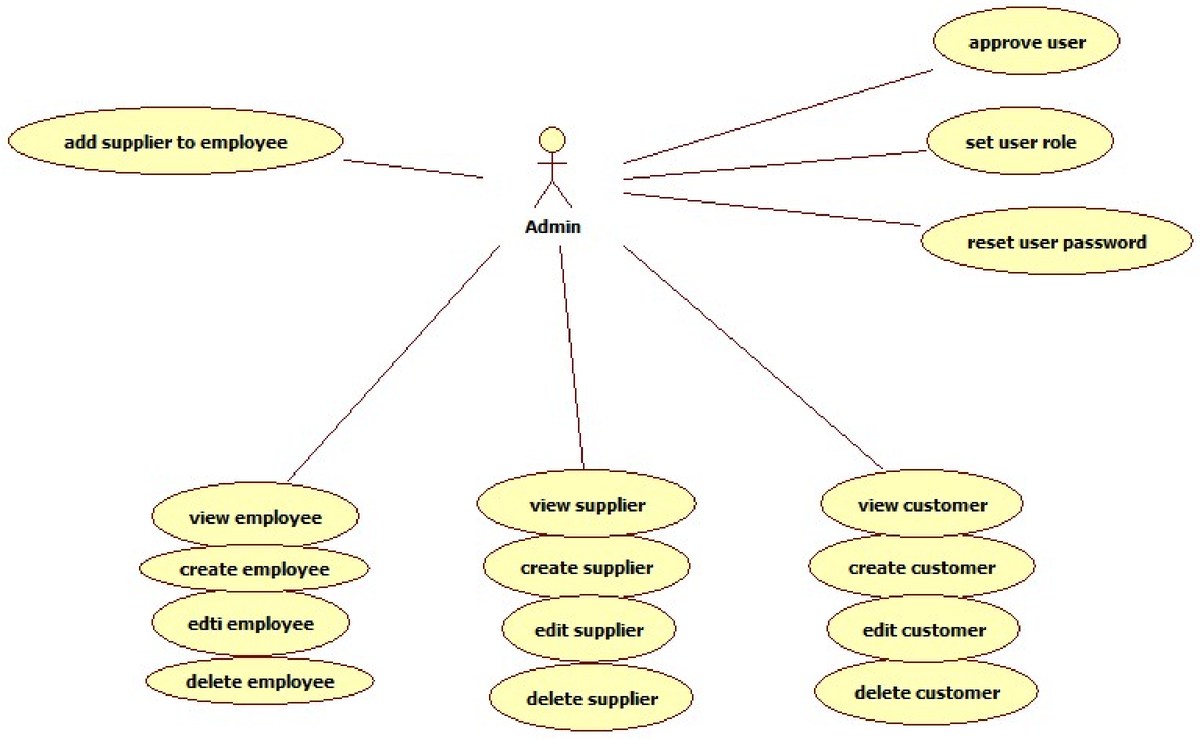


Figure 5. Use Case Diagram for the Admin role

5.1.2 Manager Role

First of all, a Manager can create new projects and assign a customer to it. This implicitly assigns him as the project manager. Furthermore, the Manager can manage the agios, i.e. defines which times of a day are assigned more costs/prices.

Apart from this, what a Manager basically does is to linking existing data together to provide a framework for the employee to track his worked time. Thereby he transforms the simple data added by the Admin into a sensible net of information. There are five fundamental actions to this:

Add Tracker

The core functionality of Tandem is to allow employees to track their worked hours. Therefore, the first and foremost thing a Manager does, alongside to creating a project, is to assign his subordinates to it. This relation between employee and project is called *Tracker.*

Add cost agio / price agio to Tracker

For every Tracker relation (employee to project), the two required agios (cost, price) have to be assigned.

View budget analyses

The Manager should be able to see some analyses of the budget. Taking into account the tracked hours for the project, the respective agios and the overall budgets for the project, some statistics and projections can be presented to the Manager.

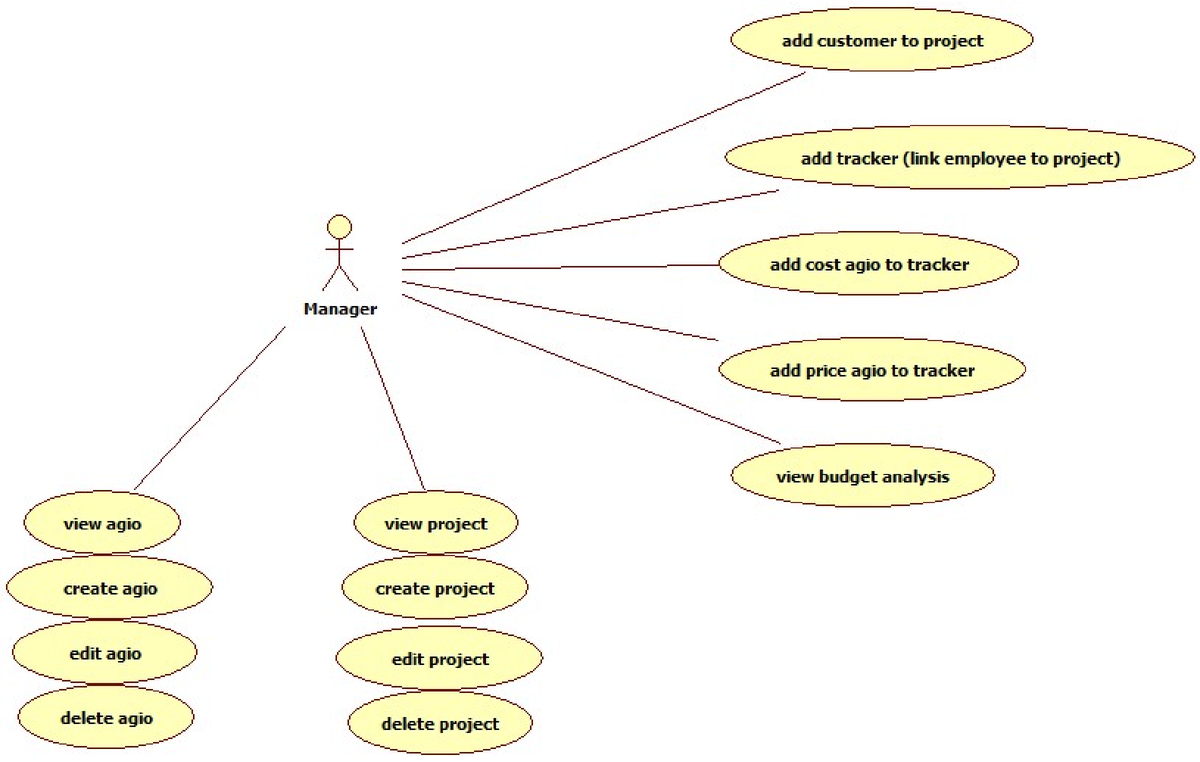


Figure 6. Use Case Diagram for the Manager role

5.1.3 Tracker and Performance Record Creators

The last two roles, the Trackers and the Performance Record Creators, while having simple use cases, the implementation is quite complex (see Database Design).

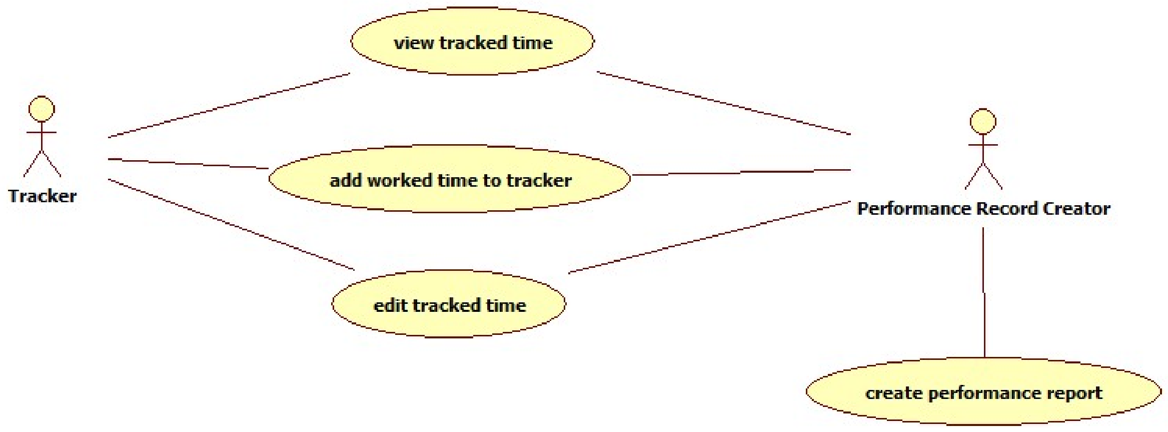
Tracker

The Tracker is limited to tracking his working hours to his projects. This not only includes the hours, but also a description, travel costs and more. These kind of datasets are called *Trackings* in this paper.

Performance Record Creator

The people belonging to this user role usually are working in the finance sector of HPE. They can have the application create a performance record to be sent to the customer.

Additionally they can manipulate the Trackings of the Trackers, as they often need to correct flawed data.


view tracked time
Tracker
add worked time to tracker
Performance Record Creator
edit tracked time
create performance report

5.2 Database Design

Project Tandem is a highly information based application. Not only does it store administrative information (projects, employees, customers, suppliers) and keeps track of worked hours, the system is also responsible for managing the complex relations between this data. This is where the relational database comes into play.

As the front end and its business logic have a very stiff structure (see Chapter 3.2), it is really the database schema that shapes the application. This chapter will explain the schema and its implications for the whole project.

Note 1: for reasons of clarity and comprehensibility some details like small tables and unimportant attributes have been excluded from this chapter. A complete version of the database schema can be found in the appendix.

Note 2: all following relations between tables are One-to-Many relationships. In the diagrams the 'One' side is represented by a key symbol, and the 'Many' side by two white dots.

5.2.1 Basic Data

First of all, there are the four basic actuators the form the basic data of the application:

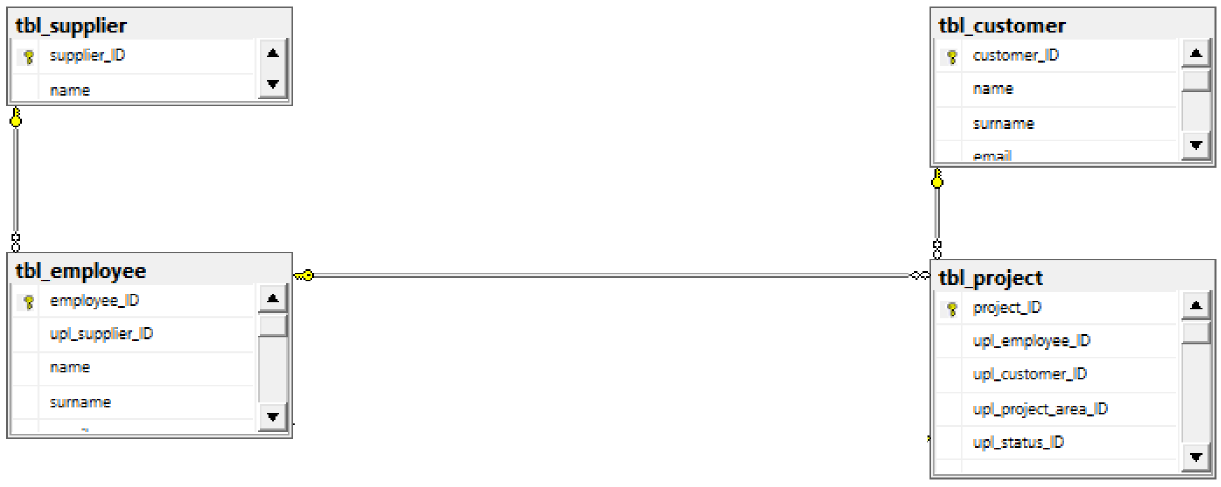
- Employees
- Suppliers

- Projects
- Customers

There are three relations between these four tables:

- An employee has a supplier
- A project has a customer
- A project is managed by an employee

Figure 7. The four basic data tables and their relations



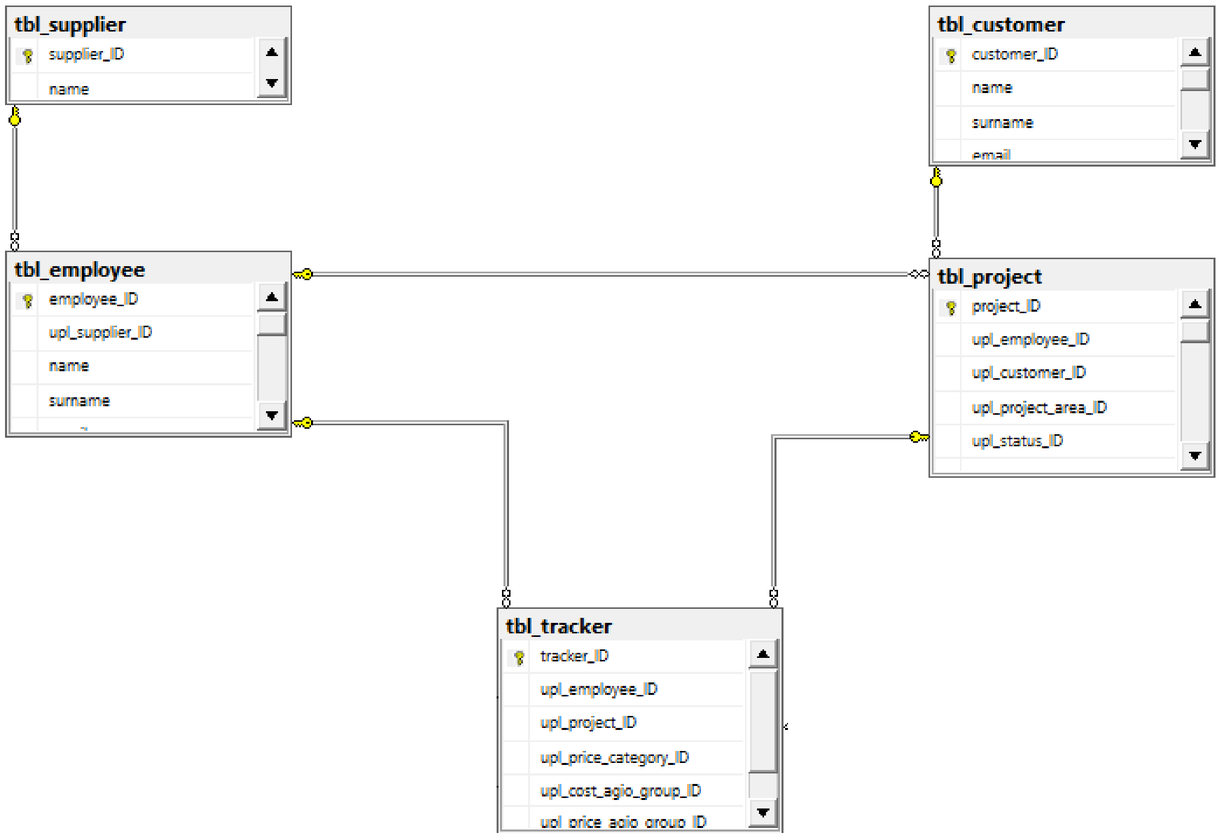
5.2.2 Time Tracking

This part will cover the database tables created for the time tracking part, which is the core functionality of the application.

The first Relation that is required for this is called 'Tracker', which links an employee to a project. This enables him to separately track his work for each respective project. It is a many-to-many relationship, as an employee can track for several projects, just as a project is being tracked on by multiple employees.

By normalizing this relationship, a new table is introduced into the schema called 'Tracker'. It is connected to the 'Employee' table and the 'Project' table by two one-to-many relations.

Figure 8. The Tracker relation in context to the basic data



The Tracker relation table defines that an employee is tracking on a specific project. What is missing is the functionality to store specific time spans to that relation. Thus two new tables are added:

Tracking

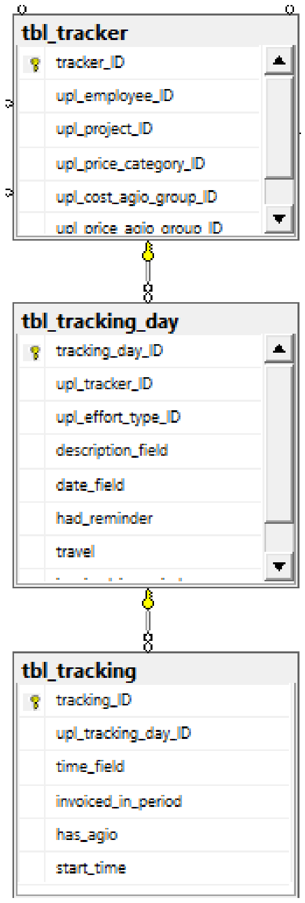
A Tracking represents one unit of worked time, e.g. four hours of continuous work for the same project. A tracking will usually only contain one property, the worked time. However, if the employee has worked outside the normal working hours (overtime, weekend, and holidays), the start time of the tracking will be saved as well. This has to do with the agios and will be further explained later.

Tracking Day

These Trackings itself do not contain the date on which the work has been tracked. Rather, it is assigned to another table called Tracking Day. This is due to the fact that on the same day, an employee can work at different times of the day. Another reason for this is that a Tracking Day has additional properties like a description and travel costs.

It is these Tracking Days that are then linked to the Tracker relation:

Figure 9. Connection between Tracker, Tracking Day and Tracking



5.2.3 Agios

Another important component of project Tandem is the calculation of different budgets. The calculation consists of four inputs:

The costs for an employee

The basic costs per hour that HPE has per employee are stored directly inside the Employee table.

The price for an employee

There are different categories under which HPE 'sells' its employees, e.g. as developer, consultant or project manager. For each Tracker relation, a category can be assigned to it, holding the basic costs per hour to the customer.

The time of the day when the work has been performed

Like explained above, a Tracking usually only holds a worked amount of time. In this case the basic costs / prices are taken into account. When the period of work does not fall under regular work time however, the time of the day is taken into account, where the surcharge matrix is used to calculate the additional costs.

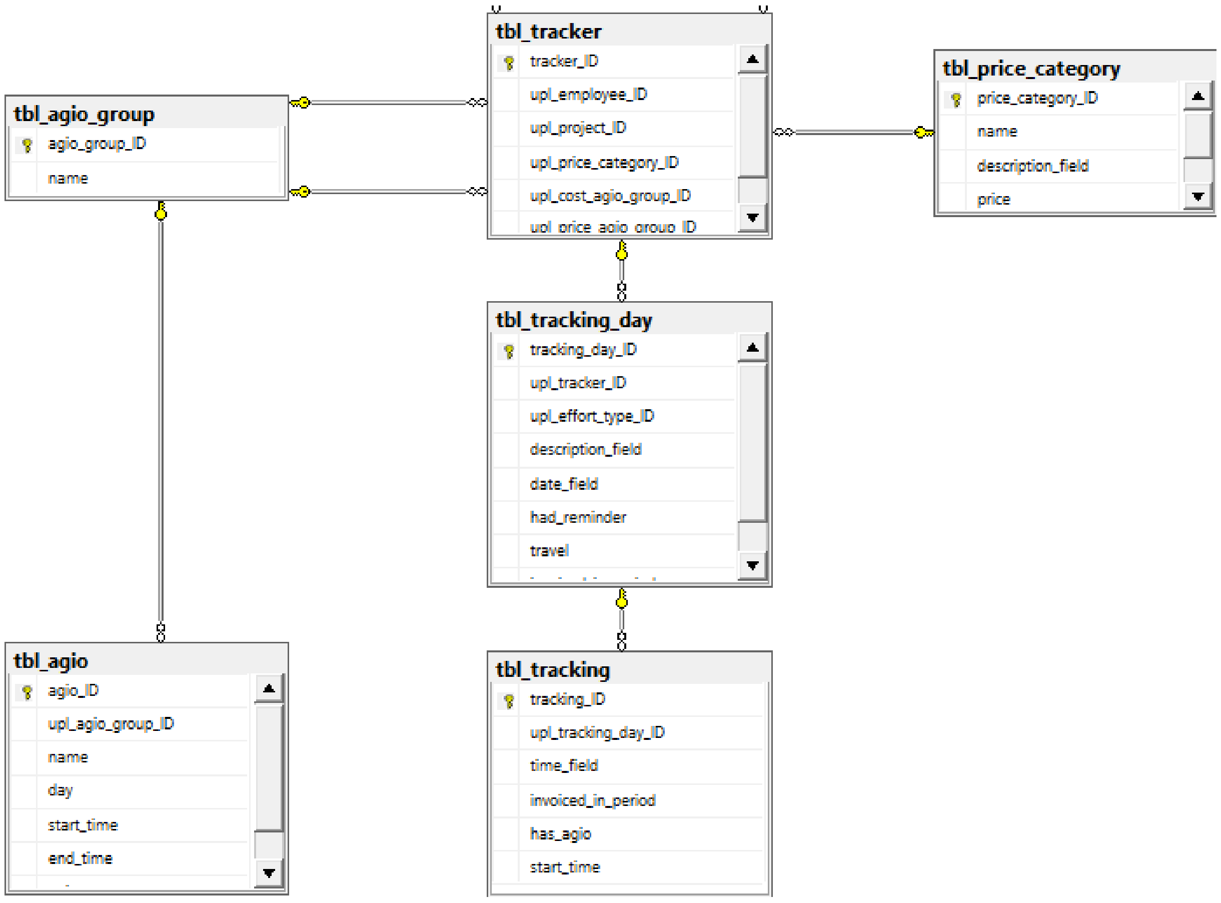
The Agio Matrix

The purpose of the Agio Matrix is to map a specific surcharge to the different times of the days in a week. The Tracker relation always has to specify two of these Agio Matrixes, one that is applied to the costs, and another one that is applied to the price.

The Agio Matrix is realized by two tables. Each record in the Agio table contains the day, the start and end time and the corresponding surcharge in the form of a percentage multiplier. The Agio Group table then compiles multiple Agio rows together.

The following excerpt from the database schema illustrates these relations:

Figure 10. The incorporation of the budget calculation tables into the schema



5.3 Implementation in ASP.NET Web Forms

Having understood the technologies used for this project, having examined the use cases of the application and with the database schema designed, the actual programming can take place.

However, this paper will not go into the details of the code. This has the simple reason that with the acquired knowledge on the Web Forms framework and the GMT framework, and with a complete database design at hand, there is practically nothing left to explain. The nature of Web Forms prohibits any sophisticated structure to be applied to the application.

Also, the code-behind model doesn't allow different pages to use the same code. That's why when explaining the exact implementation, it would be necessary to explain every view on its own. This would go beyond the scope of this paper.

Therefore a short listing of the required views should be sufficient to covering the implementation:

- Administrative views for managing Employees, Projects, Customers, Suppliers and Agios
- User Profile view
- View for assigning Employees to a Project
- Time Tracking view
- Dashboard for showing various statistics
- Interface for generating performance records

When taking the different user roles and subpages into account, a total of roughly 30 different views has to be implemented. As of the time finishing this paper, ca. 80% of the views are finished.

6 Testing and Migration to the new Tool

Before using an application in the daily business, it is advisable to first test it. As project Tandem is not ready yet for a testing phase, this chapter will shortly discuss some possible approaches.

The first step is a one month testing phase, in which a selected group of employees tracks on a number of projects. This is sufficient to affect every aspect of the application, from the registration of a user to entering the administrative data to generating time trackings and eventually to creating the performance records.

During and after this phase, critical flaws in the application can be fixed. Additionally some adaptions to the user interface can be made based on the feedback of the testers.

The next part is to migrate from the old solution to the new one. There are two strategies that can be pursued.

One approach is to keep the old tool running, while managing new projects in the new tool. This has the advantage that the administrative overhead is kept to a minimum. Also it allows to fix upcoming problems without having too much affected data. On the other hand two solutions have to be maintained at the same time. Furthermore, the employees have to use two different applications for the same goal.

The other option is to transfer all the data from the old tool to the new one. This creates a huge one time effort but creates a clean cut.

7 Summary and Lookout

The scope of Project Tandem covers many complex aspects. Grasping the underlying business and agreeing on a technology are only the first step towards the implementation. Taking a closer look at the previous approaches to the problem helped in that understanding. Generating the use cases required a vast amount of investigation on the status quo. It was important to merge the information acquired from the different perspectives of managers, employees and developers. Representing this knowledge in the form of a relational database was the next challenge. Using the Rapid Application Development framework ASP.NET Web Forms, these preparations eventually allowed to quickly design and implement the different views.

However, one thing that has been made clear over the course of this paper, is that the Web Forms technology is not suitable to this project. The sole reason for deciding on it was to ensure the long term support.

One issue that using Web Forms ensues which hasn't been mentioned yet is the user experience. With Web Forms it is not possible to provide the same interface that can be achieved using modern techniques. But when it comes to introducing a new software, the power over the success of the application solely lies in the hand of the end user. If he fails to be satisfied and rejects to adopt the new system, the new tool will fail to be established.

This and the points made in chapter 3 are the reason why regarding the future of the project, changing the technology to ASP.NET MVC is suggested.

8 Bibliography

Ciliberti, J. (2013). *ASP.NET MVC 4 Recipes: A Problem-Solution Approach.* New York: Apress.

koirala, S. (2014, 09 27). *Code Project*. Retrieved September 10, 2016, from Webforms vs MVC and Why MVC is better ?.http://www.codeproject.com/Articles/821275/Webforms-vs-MVC-and-Why-MVC-is-better

Microsoft. (2016, August 18). *Microsoft Developer Network*. Retrieved September 10, 2016, from ASP.NET Page Life Cycle Overview: https://msdn.microsoft.com/en-us/library/ms178472.aspx

Microsoft. (2016). *MSDN*. Retrieved September 10, 2016, from Using SharePoint Lists vs. Database Tables: https://msdn.microsoft.com/en-us/library/ff647105.aspx

Noel, M., & Spence, C. (2010). *Microsoft SharePoint 2010 Unleashed.* Pearson Education.

Rees, M. (2009). Retrieved September 10, 2016, from http://image.slidesharecdn.com/inft132-09303webconcepts-090920164402-phpapp02/95/inft132-093-03-web-concepts-5-728.jpg?cb=1253465082

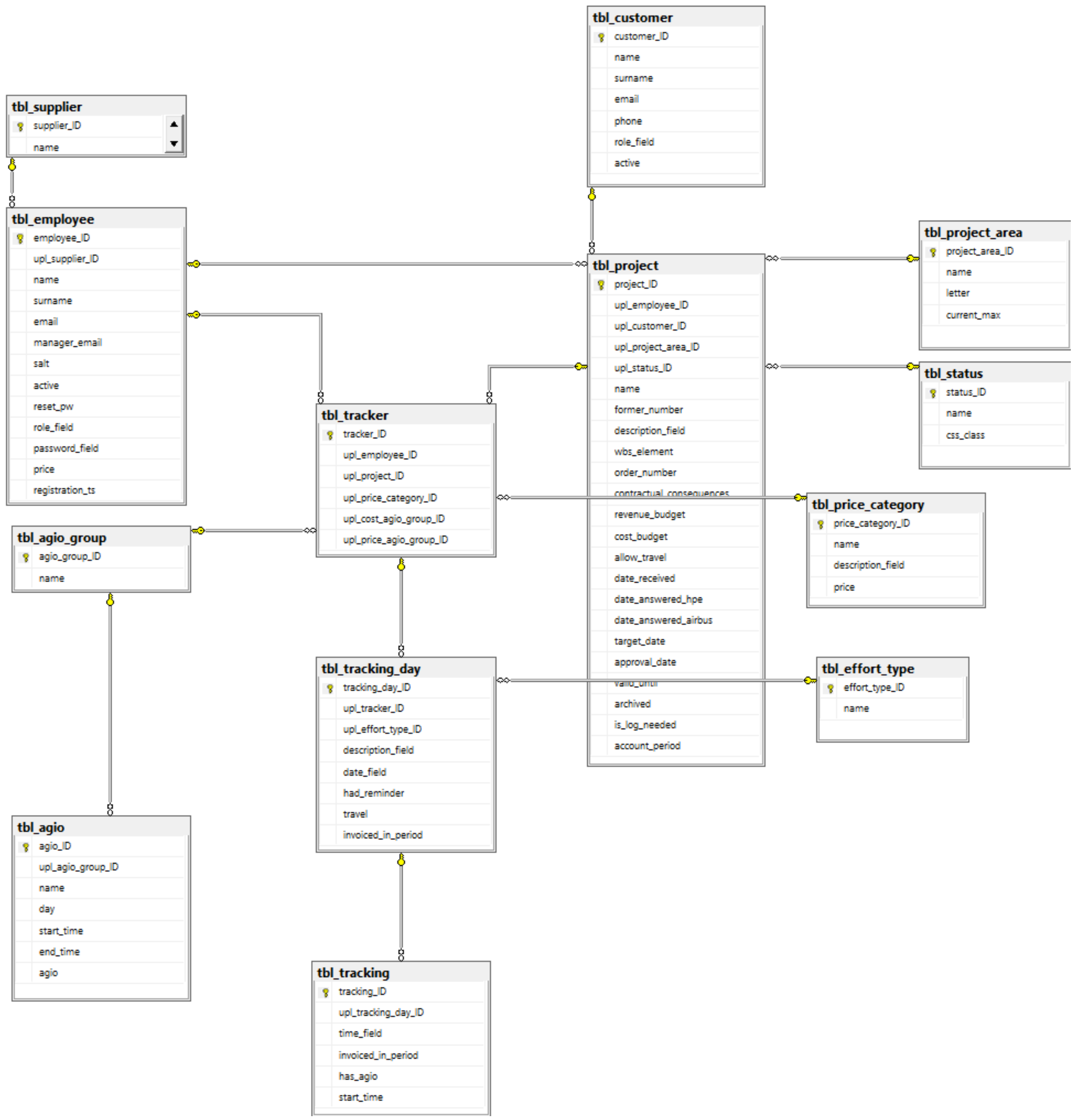
Sanderson, S. (2011). *Pro ASP.NET MVC 2 Framework.* Apress.

Troelsen, A., & Japikse, P. (2015). *C# 6.0 and the .NET 4.6 Framework* (7th ed.). New York: Apress.

Wikipedia. (2016). *Wikipedia*. Retrieved September 10, 2016, from Comparison of web frameworks: https://en.wikipedia.org/wiki/Comparison_of_web_frameworks

9 Appendix

Complete Database Schema


tbl_customer
customer_ID
name
surname
email
phone
role_field
active
tbl_supplier
supplier_ID
name
tbl_employee
employee_ID
upl_supplier_ID
name
surname
email
manager_email
salt
active
reset_pw
role_field
password_field
price
registration_ts
tbl_project
project_ID
upl_employee_ID
upl_customer_ID
upl_project_area_ID
upl_status_ID
name
former_number
description_field
wbs_element
order_number
contractual_consequences
revenue_budget
cost_budget
allow_travel
date_received
date_answered_hpe
date_answered_airbus
target_date
approval_date
valid_until
archived
is_log_needed
account_period
tbl_project_area
project_area_ID
name
letter
current_max
tbl_status
status_ID
name
css_class
tbl_tracker
tracker_ID
upl_employee_ID
upl_project_ID
upl_price_category_ID
upl_cost_agio_group_ID
upl_price_agio_group_ID
tbl_price_category
price_category_ID
name
description_field
price
tbl_agio_group
agio_group_ID
name
tbl_tracking_day
tracking_day_ID
upl_tracker_ID
upl_effort_type_ID
description_field
date_field
had_reminder
travel
invoiced_in_period
tbl_effort_type
effort_type_ID
name
tbl_agio
agio_ID
upl_agio_group_ID
name
day
start_time
end_time
agio
tbl_tracking
tracking_ID
upl_tracking_day_ID
time_field
invoiced_in_period
has_agio
start_time