FACULTY OF SCIENCE

UNIVERSITY OF SOUTH BOHEMIA IN ČESKÉ BUDĚJOVICE

Design and implementation of modules “Phases” and “External app connectivity” for real enterprise project “Time Project” using Angular2 and ASP.NET WEB API frameworks

Master thesis

Bc. Michal Kortan

Supervisor: Václav Pustějovský, Ing

Guarantor: Marta Vohnoutová, Ing

České Budějovice 2017

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL MAGISTERSKÉ PRÁCE

Student: Michal Kortan, Bc.

Obor – zaměření studia: Aplikovaná informatika

Katedra/ústav, kde bude práce vypracovávána: Ústav aplikované informatiky

Školitel: Ing. Václav Pustějovský
Baader Computer spol. s r.o.
Thámova 11-13
186 00 Praha 8 – Karlín
Tel.: +420 226 044 100
Fax.: +420 226 044 199
Email: vaclav.pustejovsky@bcpraha.com

Garant z PřF: Marta Vohnoutová, Ing, Aplikovaná informatika

Školitel – specialista, konzultant: Marian Benčat, Ing. Michal Merta

Téma magisterské práce: Návrh a vývoj modulu „Phases", „External app connectivity" pro reálný enterprise projekt „Time Project" (chytrá evidence času) pomocí frameworků Angular 2 a ASP.NET WebAPI

Cíle práce :

Systém „Time Project" (dále jen TP) slouží jako docházkový systém k podrobné evidenci času zaměstnance. TP je rozdělen do několika modulů, které umožňují zaměstnavateli provádět různé operace nad denními záznamy (dále jen DN), strukturovat DN do projektů/paketů, provádět statistiky apod. Předmětem této práce je TP rozšířit o níže uvedené moduly a to konkrétně modul „Phases" tj. administrace fází, přiřazení k jednotlivým projektům, přiřazení fází k DN. "External app connectivity" tj. napojení TP na rozhraní externí aplikace a umožnit přiřazení externí reference k dennímu záznamu. V našem případě se jedná o napojení na issue tracker systém JIRA. Nicméně práce by měla řešit obecné napojení na různé typy podobných systémů pomocí různých způsobů.

Úkolem studenta je dále projít celým vývojem softwaru od návrhu řešení, definici databázových struktur, nakódování frontendu i backendu a naučit se frontendový framework Angular 2 popř. backendový ASP.NET WebAPI.

Závěrem této práce bude naimplementovaný, funkční a zdokumentovaný modul „Phases", "External app connectivity" včetně navržených způsobů komunikace s externími systémy.

Základní doporučená literatura :

Coury, Felipe, et al. 2015-2016. *NG-Book 2.* s.l. : Google, 2015-2016.
framework, Entity. 2016. Entity framework tutorial. [Online] 2016. [Cited: 10 19, 2016.] http://www.entityframeworktutorial.net/.
Freeman, Adam. 2013. *Pro ASP.NET MVC 5.* s.l. : Apress, 2013. ISBN13: 978-1-4302-6529-0.
Google. 2015. Rangle's Angular 2 Training Book. [Online] Google, 2015. [Cited: 10 19, 2016.] https://angular-2-training-book.rangle.io/.
Uurlu, Ali, Zeitler, Alexander and Kheyrollahi, Ali. 2013. *Pro ASP.NET Web API.* New York : Apress, 2013. ISBN-13: 978-1430247258.

Financování práce :.....

Vedoucí práce : Ing. Václav Pustějovský.....podpis : .....

U externích vedoucích fakultní garant práce: Marta Vohnoutová, Ing.....podpis : .....

Garant oboru mag. studia .....podpis : .....

Vedoucí katedry: RNDr Libor Dostálek.....podpis : .....

Případný souhlas vedoucího ústavu AV .....podpis : .....

V Českých Budějovicích dne .....podpis studenta : .....

Bibliography

Kortan, M., 2017: Design and implementation of modules “Phases” and “External app connectivity” for real enterprise project “Time Project” using Angular2 and ASP.NET WEB API frameworks. Mgr, Thesis, in English. – 95 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Annotation:

The thesis deals with module extension in currently developing project called Time Project 5. Specifically, the Phases, Notification and External Interface module. The thesis is divided into four major chapters. The first chapter introduces theoretical background consisting of backend and frontend technology description. The second chapter serves as determination of customer’s expectations in form of functional and nonfunctional requirements gathering. The major focus of the third chapter is on application architecture design of both server and client side together with subcomponents design. Additionally, this chapter is complemented by the data model and Time Project 5 application design. The last, and at the same time, major chapter reveals the actual implementation of mentioned subcomponents including further necessary support features.

DECLARATION

I hereby declare under oath that the submitted Master's degree thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited. The submitted document here present is identical to the electronically submitted text document.

I hereby declare that, in accordance with Article 47b of Act No. 111/1998 in the valid wording, I agree with the publication of my master thesis, in full form to be kept in the Faculty of Science archive, in electronic form in publicly accessible part of the STAG database operated by the University of South Bohemia in České Budějovice accessible through its web pages. Further, I agree to the electronic publication of the comments of my supervisor and thesis opponents and the record of the proceedings and results of the thesis defense in accordance with aforementioned Act No. 111/1998. I also agree to the comparison of the text of my thesis with the Theses.cz thesis database operated by the National Registry of University Theses and a plagiarism detection system.

V Českých Budějovicích, Dne 10.4.2017

Podpis autora……

Acknowledgement

It is my pleasure to acknowledge the roles of several individuals who were instrumental for completion of my Master Thesis.

First of all, I would like to express my gratitude to Václav Pustějovský, Ing, whose skillful guidance and ideas helped to shape the thesis. I truly enjoyed working under his leadership.

Second of all, I would like to acknowledge helpful suggestions and guidance, especially in implementation phase, from specialists/consultants Michal Merta, Ing and Marian Benčat.

And Finally, my deepest appreciation belongs to my family for their patience and understanding during my studies at University of South Bohemia in České Budějovice.

Contents

Chapter 1

Introduction

In the constantly growing world of modern web technologies it is very difficult to choose the right system that can help run businesses more efficiently in terms of time evidence. Among sophisticated and specialized enterprise solutions for project based time recordings belongs Time Project. This solution can even compete with giants such as ZEP[1] or SAP[2] in this field.

Time Project 5 (hereafter TP) is a re-implementation and improvement of TP 4.3, a smart client server time effort documentation system based on Borland Delphi Pascal of 2005. The new system is a web application based on the modern Microsoft .NET programming languages and includes sophisticated application frameworks like Google's Angular 2 and the powerful Kendo UI components.

The thesis deals with necessary theory introduction, design and implementation of system subcomponents/modules- Phases, Notifications and External connectivity. Phases module is used to organize the effort in certain phases. This is appropriate for billing requirements used by architects in Germany, who must document their services in the context of certain phases in an immobile construction process. The phase module than allows to assign every item of a daily effort report to a certain phase. The module for Notifications was implemented to present system messages from different incidents in a modern way by using Angular 2 user interface integration. The last module deals with several connections to TP. The major focus is on Jira to TP connection that dulcify filling in a daily record.

[1] Zeiterfassung für Projekte- Web solution for project oriented work
[2] Systeme, Anwendungen und Produkte in der Datenverarbeitung (Systems, Applications & Products in Data Processing)- Multinational software corporation

1.1 Thesis objectives

- Introduction of Time Project 5
- Theory description
- Requirements analysis
- Data model, system architecture and subcomponents design
- Implementation part

1.2 Time Project 4.3

Latest version of Time Project (TP) has reached a certain age of 15 years and is already considered as outdated. TP was designed and implemented as desktop application (client and server) in Delphi by company Kirchhoff Datensysteme Software. This company was established in 1986 under the name “ASSDOS” with specialization in house service systems. The company Baader computer raises at the beginning of the nineties with rapidly growing area of software development. Later on, in order to meet the various business priorities, the company transformed into holding in 2003 and German companies have been renamed "Kirchhoff Datensysteme Service" and "Kirchhoff Datensysteme Software (Kirchhoff Datensysteme Software , 2014).

TP 4.3 is an application for time evidence of employees in connection to projects, activities, phases etc. All this information can be provided not only for internal and external purposes, but also for responsible project managers or administrators. Collecting these information leads to significant increase of time estimations on specific tasks called Project management and Corporate controlling. TP is suitable for all companies from various fields such as marketing, commercial agencies, architects, consultants or software developers which are using evidence of time to calculate their expenses. Already existing clients prove the diversity of TP.

- Graffinity Pharmaceuticals AG
- CONTEAM
- Budde Industrie Design GmbH
- CONCENTIS and many others

TP offers a great amount of options for specific business needs from evidence of attendance to highly specified project management. The most important functions and features are:

1) Easy handling
2) Flexibility and customization
3) Extensive evaluation possibilities
4) Project budget planning
5) Reliability and security
6) Easy evaluation of working hours for customer

In addition, TP includes the possibility to customize access rights according to specified needs or early warning system (i.e. warning before budget is exceeded) (KG, 2008).

As it was implied, TP 4.3 was created over a decade ago that is why it is based on Borland Delphi Turbo Pascal, a programming language that is generally not in use any more. Also, the design is focused on Windows 95 and has never been improved (Figure 1: Time Project 4.3). These are the major obstacles for further improvements and additional feature developments. Therefore, a proposal has been made to develop new version of TP 5 which is based on modern technologies.

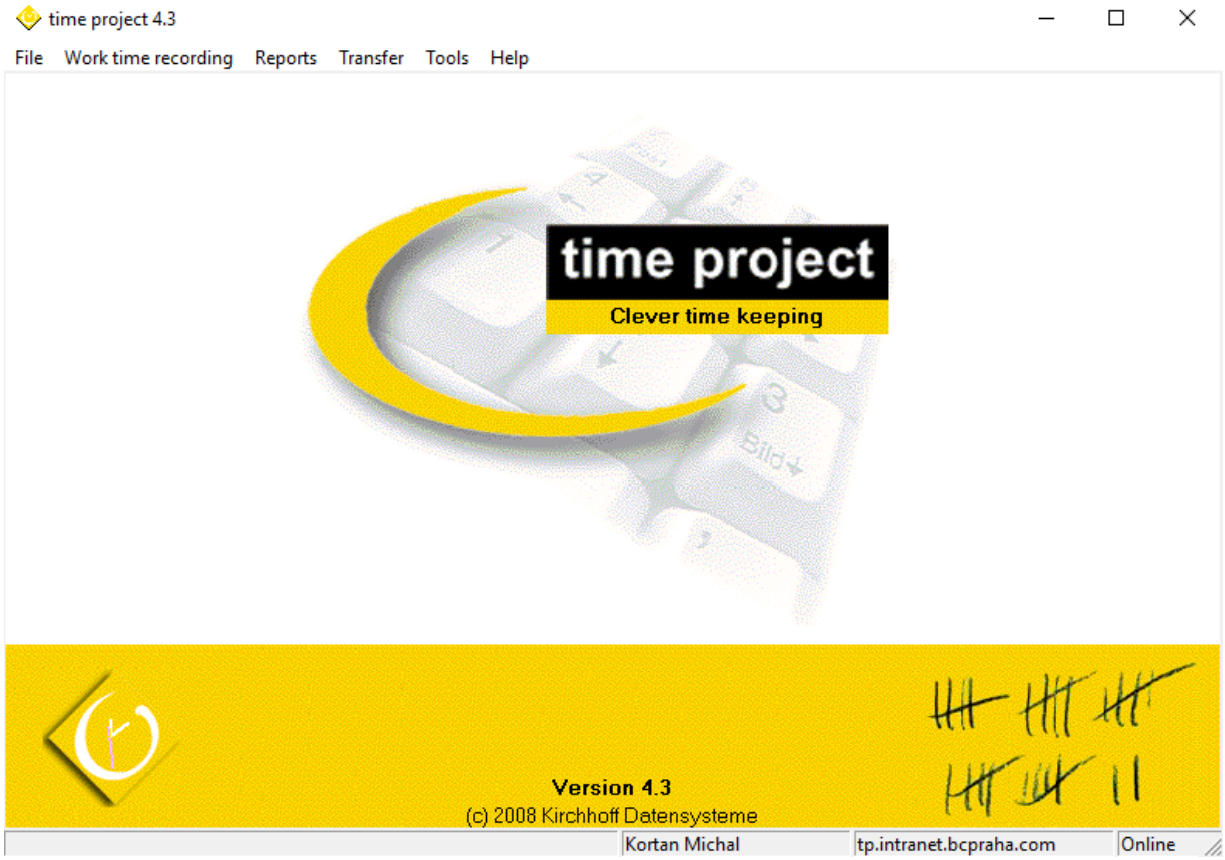


Figure 1: Time Project 4.3

1.3 Time project 5

Decision has been made for new TP 5 to be implemented as a web application. A web application is a program that is stored on a remote server and delivered through the Internet using a web browser. The idea of replacing old TP 4.3 occurred when a major customer (City of Cologne) requested new improvements. It was, in general, not possible to enhance features of the old TP at all. The new TP 5 covers new modern design based on responsive template/layout to support all devices with different resolutions. TP 5 targets mainly current customers like:

- Public service authorities (City of Cologne)
- Public sector companies
- Freelancers like architects
- Small and medium enterprises in the service sector (like Baader Computer)

To be able to develop the best possible solution, the research of competitors has been made. This research includes functional requirements as well as very detailed product pricing. TP does not only compete with other software applications (specialized in time recording or as part of a large enterprise resource planning solution) but also with self-tinkered substitutions like MS Excel or MS Access. As far as SAP (comes from abbreviation of "Systeme, Anwendungen, Produkte in der Datenverarbeitung") solutions are concerned, TP cannot be compared to build-in solution specifically designed to its needs. The users of Excel and Access will at some point crash against their boundaries. So, the real competition would be with existing sophisticated and specialized solutions for project based on time recordings. Because TP is targeted for German companies, local market needs to be researched. Among the strongest competitors in German market is ZEP. Based on collected data, new version of TP had to be differentiated. Other products in the field of Time recording focus mainly on project which stands in the center of the application whereas TP focuses on user and his daily duties.

At the beginning of the thesis, some of the used technology was predefined by the choices Baader computer made before the development of the TP 5 started. Namely, RDBMS operated by MS SQL Server and ASP.NET Web API as a major framework. To develop TP 5, the thesis must cover number of other technologies that must be studied and understood. These include N-Tiered architecture, client and server side frameworks and techniques, implementation technologies such as not only relational databases and ASP.NET WEB API, but also Entity Framework and Angular 2.

Chapter 2

Theory

The purpose of this chapter is to briefly describe necessary theory about developed modules. All this theory is practically used in implementation part of the thesis.

2.1 Development approach

In today's fast-developing world, the key is to choose the most suitable method for software development. Among the most used and widespread SDLCs[3] belong for instance agile software development, spiral model and waterfall model (Jamsheer, 2017). The whole process of Phases module, Notification module and External app connectivity can be divided into two main parts. First part consists of analysis, requirements collection and design. Second parts' major focus is on implementation phase. From the above points, it can be observed that for the first part the most suitable method is waterfall. For the second part, however, agile methodology with weekly sprints is used mainly because of the ability to constantly inform customer about new implemented features.

2.1.1 Agile development

This incremental and iterative approach provides opportunities to estimate the direction of a project throughout the software development lifecycle. The basic idea is that at the end of the so-called sprints (weekly, monthly) the development team must present potentially shippable product. Agile paradigm emphasizes on continual revision of all software development aspects namely analysis, design, implementation and testing (McLaughlin, 2013).

[3] Software/System Development Life Cycle

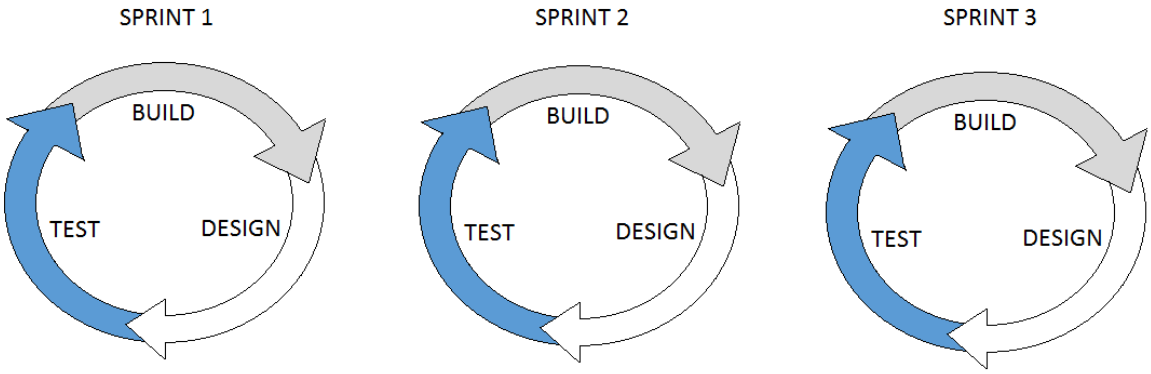


Figure 2: Agile development

2.2 Database

The word database is a compound of two words, data and base. Data can be explained as known facts which can be reported and have specific meaning. Another definition states that they are expressions for detail used for phenomenon description or observable property of object. Base can be described as the set which allows to set the coordinates to specified space. Based on these definitions, it can be claimed that database is a partly ordered set of data. In addition to the definition, database represents logically linked set of real aspects of the world (Daux, et al., 2002).

To be able to operate with databases DBMS[4] must be present. It is an aggregate of programs, procedures and methods which allow users to create and maintain databases. Among the major functions and abilities of DBMS belong:

1) Database definition - data type, structure and integral restriction definition
2) Database construction - data storage process
3) Database administration - access, data manipulation- includes query and search functions

[4] Database Management System

There exist several types of DBMS such as navigational, relational, object-oriented and so on. Next paragraph focuses on relational database in more detail because it has been predefined by Baader computer for TP 5. The paragraph, however, serves only as a very brief introduction to relational database creation.

One of the recommended ways of how to develop RDM[5] is to create ERM[6]. It is basically a model, based on analysis of requirements, for abstract and conceptual data representation. For that it uses entities, relationships, identifiers, attributes and instances. Entities represents an object in real work such as (user, car etc.) and is also a set of instances. Attributes are properties used for closer entity specification. Several types of relations exists between entities - 1:1, 1:N, M:N. When entity-relational model is finished, the relational model can be created. The table below describes the transformation between these two types (Daux, et al., 2002).

ERM	RDM
Entity	Table
Identifier	Primary key
Attribute	Column
Instance	Row

Table 1: ERM to RDM

Microsoft SQL Server Management Studio 2016 program has been used to maintain relational database. As the name might suggest it uses SQL[7] for querying and maintaining the database.

[5] Relational Data Model
[6] Entity-Relationship Model
[7] Structured Query Language

2.3 Entity framework

Entity Framework (hereafter EF) is an ORM[8] framework that enables working with relational data using domain-specific objects. It eliminates the need for most of the data-access code that usually needs to be written. EF allows to create a model by writing code (Code first approach) or using boxes and lines in the EF Designer (Database first approach or Model first). Both approaches can be used to target an existing database or create a new database (Microsoft, 2016). EF is an enhancement to ADO.NET[9] that has an automated build-in mechanism for storing and accessing data. Querying these data is then achieved via LINQ[10], then retrieved and handled as strongly typed objects. One of the biggest advantages is that ORM allows to keep database design separate from domain class (maintainability and extensibility). In addition, it automates standard CRUD[11] operations. It also provides services such as lazy loading or change tracking, however, these services are described in implementation part of the thesis (EntityFrameworkTutorial, 2016). Finally, it is important to mention that TP 5 is built on 6.1.3 version of EF.

[8] Object-Relational Mapping
[9] Data Access technology- communication between relational and non-relational systems
[10] Language Integrated Query
[11] Create, Read, Update, Delete

2.4 ASP.NET Web API 2

ASP.NET is built on the .NET framework, as a server-side web application framework designed to produce dynamic web pages. In contrast with WCF[12] service, ASP.NET WEB API 2, that TP 5 uses, just needs a REST[13] URL, a set of inbound arguments, and a response JSON or XML message (Figure 3). The biggest advantage of this framework is that it is REST (Representational State Transfer) by default which makes it interoperable across all platforms capable of making HTTP[14] requests. It can be inferred that every device capable of making HTTP requests to a URL[15] is RESTful. The same applies to JSON[16] and straight XML[17] data. JSON has become favorite mainly due to its simplistic format in comparison with the size of XML/SOAP[18] data (Anderson, et al., 2012).

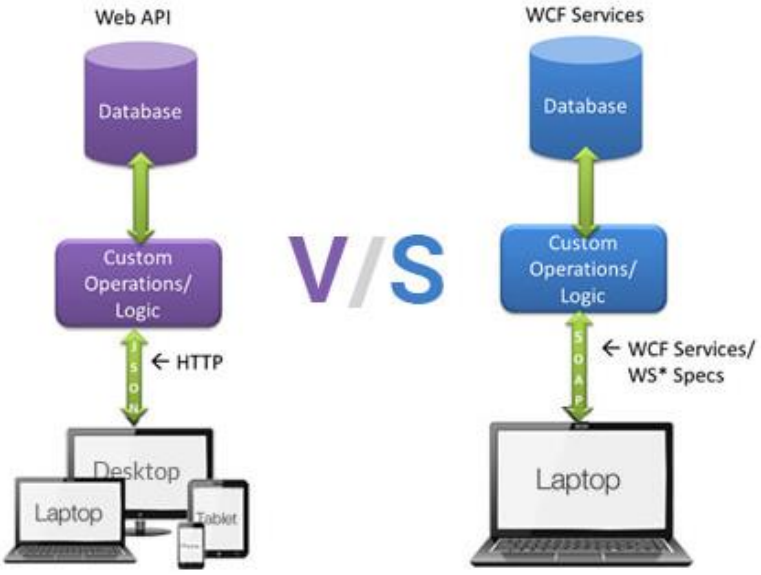


Figure 3: Web API vs WCF Services (Dee, 2014)

[12] Windows Communication Foundation- a set of APIs for building connected, service- oriented applications
[13] Representational State Transfer- provides interoperability between systems on the internet
[14] Hypertext Transfer Protocol- application protocol for transferring files on the World Wide Web
[15] Uniform Resource Locator= web address
[16] JavaScript Object Notation- open-standard format for transmitting data objects
[17] Extensible Markup Language
[18] Simple Object Access Protocol- messaging protocol

2.5 Angular 2

On September 14th 2016, after two and half years of development, the final version of Angular 2 was released. It is a framework for building client applications in HTML and either JavaScript or TypeScript that compiles to JavaScript. Writing Angular applications is made by composing HTML templates that are managed by component classes. An application logic is then run by services which are together with mentioned components boxed in modules (Google, 2016). So far, three of eight main building blocks of the Angular application have been mentioned. Those and the other blocks are described in more detail in the Design chapter.

Chapter 3

Analysis

Analysis is a process of determining customer's expectations and needs for an application. This process uses two types of requirements:

1) Functional- functions and behavior
2) Nonfunctional- system specification

Nonfunctional requirements are covered in chapter Design, specifically Architecture design of server and client side. Functional requirements apply the use case analysis for better understanding and description (Martin & Martin, 2006). However, since the use case diagram provides a graphical overview above users and their actions/functionalities they want to achieve based on gathered requirements, following chapters take into consideration functional requirements only.

3.1 Core features

As implied in previous chapters, the whole system concept partly follows the former version of TP 4.3 functionality. Database model has been completely reengineered and transformed into MS SQL database where all data produced by following modules is stored: Work time, Employee, Cost, Activities, Projects, Customers, Phases, Notifications and others. For the purposes of the thesis, module phases and notifications together with others are the most relevant to focus on. Next chapter describes all modules in more detail.

3.2 Phases module

Some TP customers are using the extra Phase module to organize their effort in certain phases. This is appropriate to billing requirements for architects in Germany, they must document their services in the context of certain phases in an immobile construction process. (In German these phases have the certain name “Gewerke”, there are phases for the foundation of a house or for the timber work to construct the basis of a roof.) The phase module allows to assign every item of a daily effort report to a certain phase, when this is applied to an appropriate selected project. Thus, the total effort can be reported in a sorted way divided by different phases.

3.3 Notification Module

The old TP 4.3 included a basic notification system that generated dialogue based messages for warnings about budget exceeds. The new TP should include a more useful and common solution. Thus, the module for Notifications was implemented to present system messages from different incidents in a modern way. This is fully integrated in the Angular 2 user interface. As in other enterprise web applications the notification feature appears as a bell icon in a header menu. This icon indicates the existence of unread messages by the number of these items. After a click on the bell, icon opens the notification browse list that allows to select and to open the appropriated notification message that the user wants to read and react to.

Phase Requirements:

Read phase- Phases browse grid

Create phase- Phases detail view

Edit phase- Phases detail view

Delete phase

Filter and sort phase grid

Export to excel

Data localization into EN, DE, CZ

Last changes logging

Assigned phase to project and otherwise

Assigned project in phases browse grid

Set the duration of assigned project in phase

Autocomplete field for assigned phases for project

Autocomplete field for phases in the daily record item form

Possibility to turn usage of Phase module off

Possibility to record daily item without phases

Implement a design according to Mockup

UI localization

*Notify the user when new phase is created, edited, deleted and exceeded

***Notification requirements:**

Indicator of unread notifications

Read notification- notification browse grid

Notification detail view

Notification must be stored in all languages

Read notification template- notification template browse grid

Edit notification template in all languages- notification template detail view

Notification generation source

Localizable

More detailed specification of these requirements has been gathered from issue/project tracking system called Jira[19] where the most tasks are listed. In case of further specification, TP project manager has been contacted. Complete documentation can be found on attached CD.

[19] Software development tool used by agile teams

3.4 External connectivity

The thesis deals with several connections with TP. One of them is known as Jira connection to TP. The idea behind this connection is the possibility to dulcify filling in a daily record. To successfully create a daily record a project name together with subproject must be filled in. However, Jira connection to TP enables filling in so called Jira issue key which causes automatic completion of project and subproject name. To be able to achieve such behavior the following requirements need to be accomplished.

Requirements:

- Create New custom field in Jira
- Establish a connection between Jira server and TP database
- Establish a connection between TP server and Jira server
    - Using Basic authentication
    - Prepare for using OAuth
- Create appropriate area in TP system settings to define connection to Jira server
- Jira autocomplete inside daily report item

Chapter 4

Design

Design phase is where the application architecture, which defines components, their interfaces and behavior, is established. This phase starts when all requirements from analysis phase are gathered and subsequently mapped into an application architecture. Whilst analysis phase focuses on doing the “right” thing, design phase focuses on doing “things” right (in other words “how” phase). Thus, design phase determines which programming languages, application architectures, architecture layering, data structures and many others to use (Burback, 1998).

Following chapters represent data model design and architecture of both, client and server side application with detailed description. Further, it details application design together with color scheme.

4.1 Data model design

Conceptual representation of Phases data structure was designed with the help of Microsoft Visio 2016 Crow’s foot database notation template. Instead of showing all conceptual models and their appropriate relations, the thesis displays only one conceptual model (Figure 4). Other necessary models (presented in Implementation Chapter) are relational only.

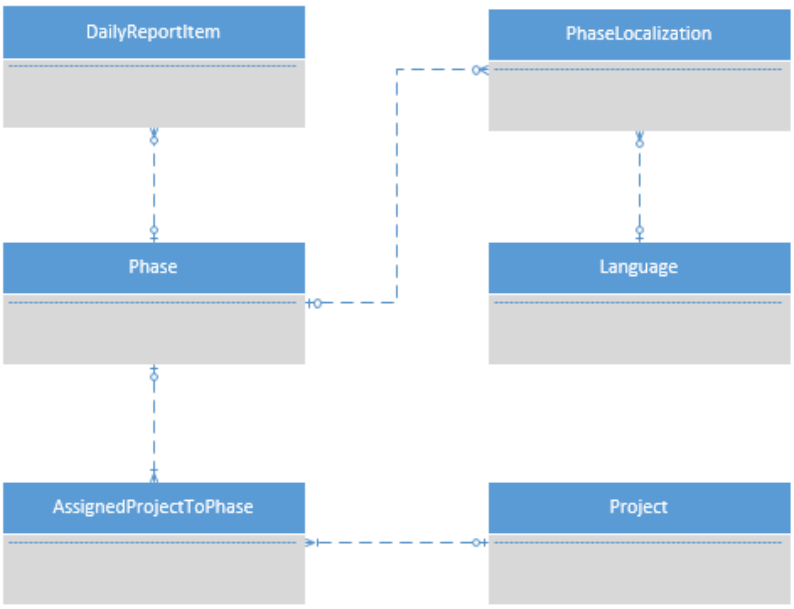


Figure 4: Conceptual model of Phases module

The next step in data modeling is usually creation of Logical model which helps to define the detailed structure of entities and their relationships. This step was not applied since the conceptual model is quite simple. Toad Data Modeler has been used for creating a relational model, that uses so called Information engineering style notation. The figure bellow depicts such a phases module model.

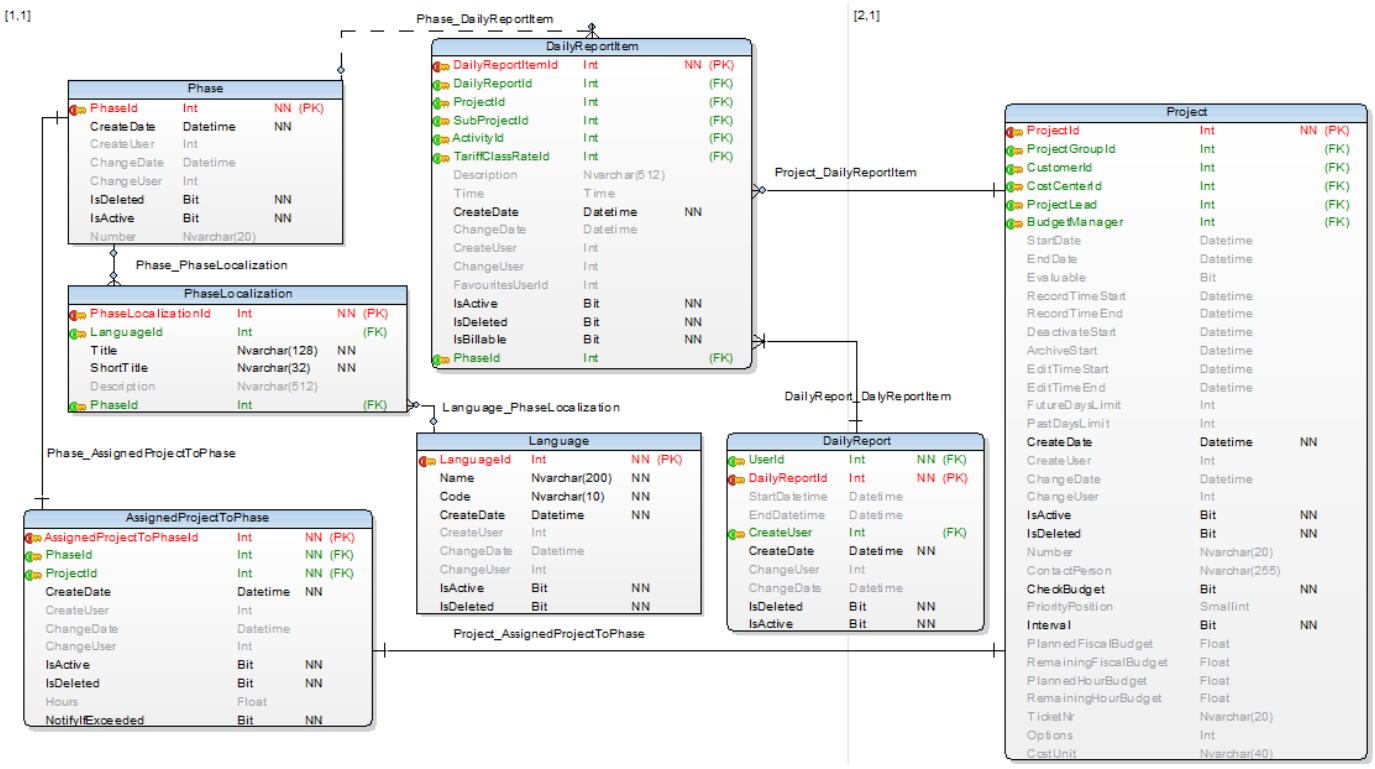


Figure 5: Phases model

The next chapters deal with architectural design of TP divided into concepts- frontend for client side and backend for server side.

4.2 Server-side architecture

4.2.1 Architecture introduction

According to W3Techs.com from 7th January 2017, PHP[20] is the most widespread programming language for server-side development immediately followed by ASP.NET. It can be assumed that Microsoft's ASP.NET framework, which TP is built on, is very unlikely to extinct. Since the TP has been marked as an enterprise project an appropriate software architecture needs to be chosen. Nothing like solid right or wrong answer exists. It can basically be said that if specifications and requirements for project are fulfilled than it can be considered as the right approach. According to (Fowler, et al., 2002) design patterns and principles make project more robust, reliable and maintainable. In addition, it makes the code highly cohesive and loosely coupled. For enterprise level application, it is recommended to follow SoC[21] which divides an application into several layers where each layer has its own responsibility => N-tier architecture. Layering of an enterprise ASP.NET application and SoC of presentation, business logic, and data access is very individual. Fortunately, domain logic patterns encompass three methods for organizing business logic.

1) Transaction script (Anemic Domain Model) – this method concentrates mainly on organization of business logic in a procedural fashion rather than object-oriented approach

2) Active record – this pattern is very useful when database model matches business model => one-to-one mapping, unfortunately it is primarily specification for simple applications and to build application with data first approach

3) Domain model (Rich Domain Model) – an abstraction of real domain objects where both data and behavior is modeled

[20] Hypertext Preprocessor- platform independent server-side scripting language
[21] Separation of Concerns

Domain model typically uses repository pattern to solve persistence between business object and domain model. In other words, when employing the Domain model pattern, it is the responsibility of the Repository object and data mapper to map business entity (POCO[22]) to data model (Millett, 2010). After figuring out that the Domain model is correct business organizational pattern for TP purposes, it is time to distinguish between different types such as anemic domain model (hereafter ADM) and rich domain model (hereafter RDM). Next paragraphs focus on description and comparison of these patterns and conclusion which one fits better for TP needs.

4.2.2 Domain models

Very similar to the classic domain model is ADM due to the fact that domain objects represent the business domain as well. Major difference between these models is that any behavior (domain logic) is not contained within the domain objects therefore they become simple data transfer classes (getters and setters). For the domain logic than serves set of service objects which are on the top of domain model. Unfortunately, this pattern by its behavior interferes with the basic idea of OOP[23] => combine data and process together. RDM, on the other hand, encapsulates all business logic and data so there is no violation of OOP which is the major advantage against ADM approach. Both approaches use a collection of principles and patterns called DDD[24]. Following data shows advantages and disadvantages when using these approaches which has been collected from many sources for instance: (Evans, 2003), (Fowler, et al., 2002), (Kumar, 2015), (Samolysov, 2016).

[22] Plain Old CLR Object
[23] Object-oriented programming
[24] Domain-Driven Design

CONS

ADM	RDM
Does not follow OOP principles – that is why it is called “anti-pattern”	Cannot be generated by code generators without resorting to complex inheritance models
	More complicated approach
	Less reusable than ADM – business logic is hard coded in the domain object classes

PROS

ADM	RDM
Easier to implement than RDM	Follows OOP principles
	Can ensure that its state is always correct
	Better for more complex business logic

Among the biggest doubters of ADM approach belongs (Fowler, et al., 2002) who said “*The fundamental horror of this anti-pattern is that it's so contrary to the basic idea of object-oriented design; which is to combine data and process together. The anemic domain model is really just a procedural style design, exactly the kind of thing that object bigots like me and* (Evans, 2003) *have been fighting since our early days in Smalltalk. What's worse, many people think that anemic objects are real objects, and thus completely miss the point of what object-oriented design is all about*.”. However, having said that, he also says this: “*It's also worth emphasizing that putting behavior into the domain objects should not contradict the solid approach of using layering to separate domain logic from such things*

as persistence and presentation responsibilities. The logic that should be in a domain object is domain logic – validations, calculations, business rules – whatever you like to call it.". This can be argued by View Models, they contain all required data to execute a view and any logic required to render the view. Considering using ORM framework and IoC[25] which saves a great amount of effort, the ADM than becomes the best solution.

The whole backend concept of TP was inspired by the best practices gathered from Microsoft's official websites msdn.microsoft.com and from MVP's (Microsoft's most valuable professional)

[25] Inversion of Control

4.2.3 Anemic domain model

The main intension is to keep architecture clean and create a highly loosely coupled solution. Several patterns (listed below) had to be followed to define the right manner of Data access, Business and Presentation layers.

- ORM – EF Code First development
- Generic repository pattern
- Dependency injection using Unity
- And many others described in more detail in Implementation chapter

Picture below shows chosen architecture diagram of TP based on ADM.

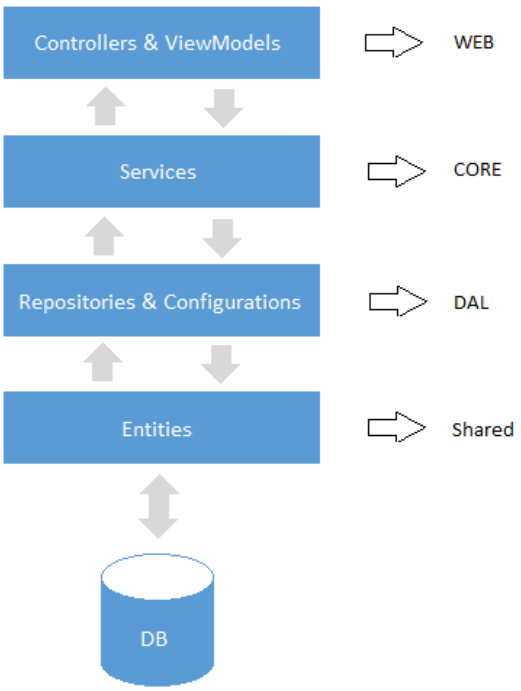


Figure 6: ADM layers

Database – used database is described in more detail in identically named chapter.

Shared – in the chapter Theory different approaches were outlined in its EF section. Since DDD patterns are followed, the Code First approach is the best choice. This approach requires creation of domain objects (Entities) in order to build the database. For configuration of these Entities, two approaches exist: DataAnnotation attributes and Fluent API. Considering configuration options and flexibility, the Fluent API is a better choice.

DAL[26] – is the **only** responsible layer for direct communication with the database. This is the place where configuration files are stored. These files use DbContext which is responsible for access to the database. DAL also includes repository layer which is the only layer available for "higher" layers (in this case Service layer) which eventually wants to access the database.

Core – a layer where business logic is implemented is called Service layer. It is responsible for validations and exposing operations to API controllers.

WEB – API controllers for handling incoming requests (sent for further processing to service layer) from client and viewmodels which represent used data. More about ASP.NET WEB API in identically named chapter.

[26] Data Access Layer

4.3 Client-side architecture

4.3.1 Java script frameworks

In today's world of endless possibilities, it is almost impossible to choose the right frontend JavaScript framework. To help determine which one is the most widespread, the google trends graph (collected data from github, stackoverflow etc.) and indeed.com were used as a source. After a small research of available JavaScript frameworks in 2016, the research ends up with ten possible candidates. They were added one by one into google trends graph to determine popularity. To sum the research up, there are three best solutions (Figure 7).

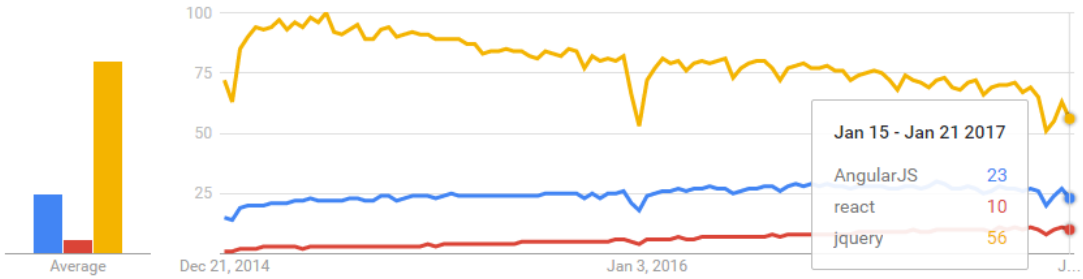


Figure 7: jQuery vs AngularJS vs React

The graph shows the popularity of jQuery, AngularJS and React worldwide from December 21, 2014 to January 15, 2017. It can be implied that jQuery is slowly decreasing whereas AngularJS and React are slowly increasing. Portal indeed.com finds approximately 13,5K jobs positions including jQuery keyword, followed by 6,5K for Angular and 4,5K for React. Because this research was done in early 2017, including angular 2 into comparison would not be relevant.

Considering the recent drop of jQuery, AngularJS is the best option. However, as it was mentioned previously, TP is built on modern technologies, that is why Angular 2 was chosen.

4.3.2 Angular 2 architecture

The purpose of this chapter is to describe the Angular 2 architecture in more detail. Angular 2 uses eight main building blocks, namely: modules, components, templates, metadata, data binding, directives, services, dependency injection (Figure 8)

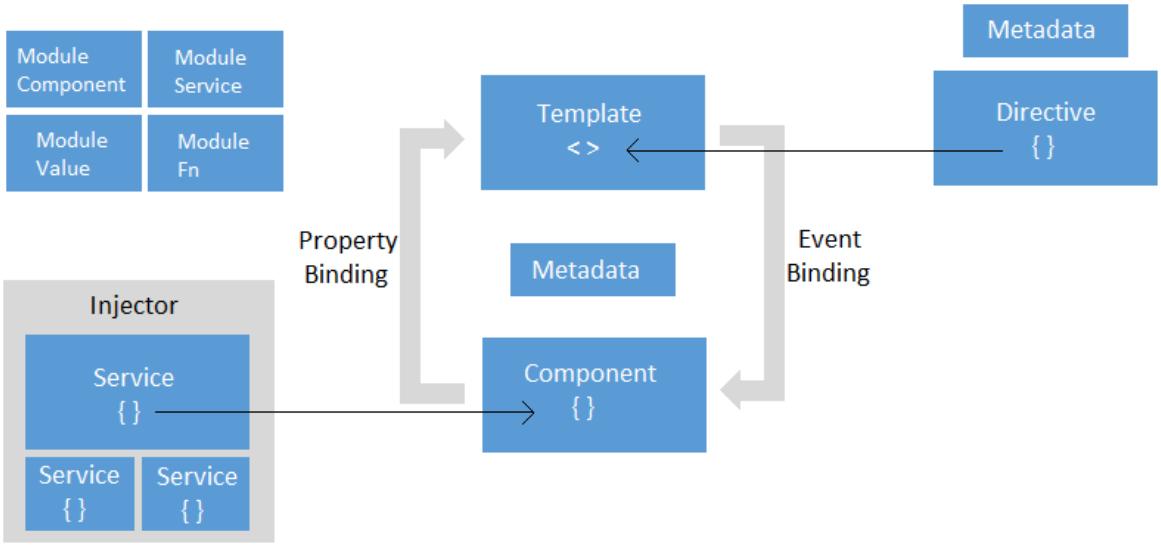


Figure 8: Angular 2- architecture (Google, 2016)

Modules

Angular modules exist to organize an application (components, directives and pipes consolidation) into cohesive blocks of functionality, each focused on a feature area, application business domain, workflow, or common collection of utilities.

Components

Angular components control a patch of screen called views. Application logic to support the view is placed right there inside a class which interacts with the view through an API of properties and methods. As well as in MVVM[27], the component represents the ViewModel part and the template represents the view.

[27] Model-View-ViewModel

Templates

Templates are a form of HTML that tells Angular how to render the component. The difference between regular HTML template and Angular template is in template syntax. Among the basic elements belong build-in directives, expressions, statements and many others.

Metadata

When for instance a component is created it is nothing but a class. To tell Angular how to process a class, metadata must be attached => usage of TypeScript decorators.

Data binding

To be able to push and pull data between DOM[28] and component, data binding mechanism offers four ways of doing that – interpolation, property binding, event binding and two-way data binding

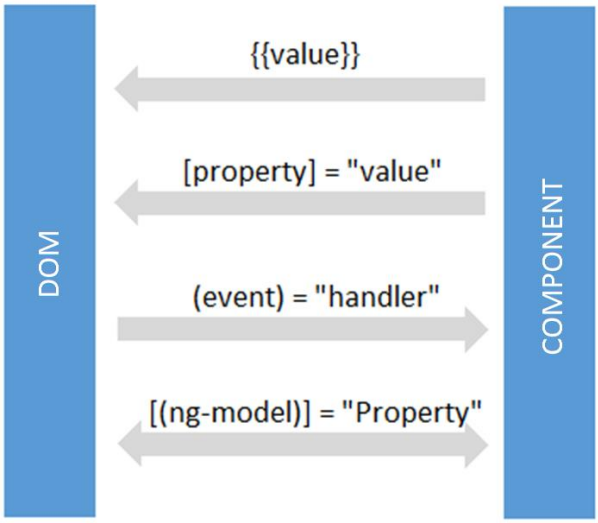


Figure 9: Angular 2- data binding

[28] Document Object Model

Directives

Because Angular templates are dynamic, directives provide instruction for transformation of the DOM structure when they are rendered. Directives of this type which alter layout by adding, removing, and replacing elements in DOM are called structural (for instance **ngFor* and **ngIf*). Attribute directives, on the other hand, alter the appearance or behavior of an existing element (for instance *[(ng-model)]*).

Services

A service is designed for specific and well-defined purpose, typically for handling the server communication. In contrast with components which should be lean – properties and methods for data binding, services should handle everything else.

Dependency injection

DI serves as a provider of services needed by components. This is maintained by using a container of service instances called Injector.

Now, when the use of Angular 2 framework is established, one of the last things that must be resolved is to choose the right UI[29] library to fulfill requirements such as Filter and Sort phase grid and Export to excel. Since Kendo UI supports this behavior from default, it is a perfect solution for this situation.

[29] User Interface

4.4 Application design

As well as TP's former version, the new version follows the same color scheme (yellow, black, grey and white) by default. After many graphical proposals, the final design was approved (Figure 10).

Figure 10: TP final design

4.5 Subcomponents design

4.5.1 Phases design

Several mockups are listed in implementation chapter for sententiousness of the specific situation. Therefore, the following text shows function description together with a few mockups. Phases module is designed in a way to accomplish all requirements mentioned in Analysis chapter. These requirements are mapped on following sections:

1. *Phase list grid (in Implementation chapter in more detail)*
2. *Detail view of Phase Basic Data*
3. *Assigned Project to Phase detail view*

This connection also stores:

- Planned hours of the phase within the project
- When planned hours are filled, checkbox NotifyIfExceed is enabled

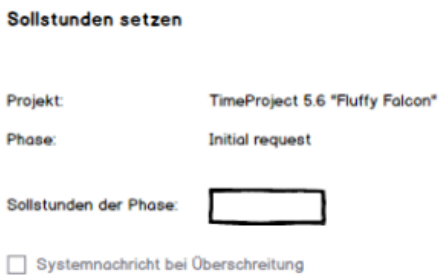


Figure 11: Assigned Project to Phase detail

4. *Project in Phase list grid*

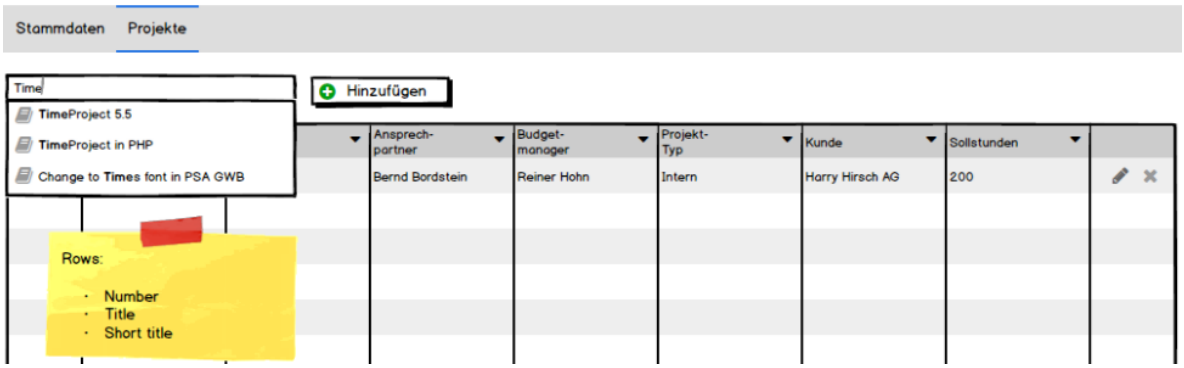


Figure 12: Project in Phase list grid

5. *Phase tab in Project module*
    - Assigned Phases to Project grid
    - Assigned Project to Phase detail data
6. *Autocomplete field for phases in the daily record item form*

There are three possible scenarios:

1. Sub project entered, phase not entered
2. Sub project and phase entered
3. Sub project not entered, phase entered

First scenario is possible when sub projects are part of the project and the user has selected one of them. It can be that there were no phases assigned to the project, then the field for phases is not visible. Or the entering of phases is set as optional in the system settings, then the autocomplete field can be left empty.

Second scenario is possible when subprojects and phases are assigned to the project and the user has selected both in the appropriate autocomplete fields.

Third scenario is possible when no subproject is assigned to the project or the selection of subprojects is optional. The user then has not selected one. The phase of Implementation was assigned to the project and selected by the user for this daily report item.

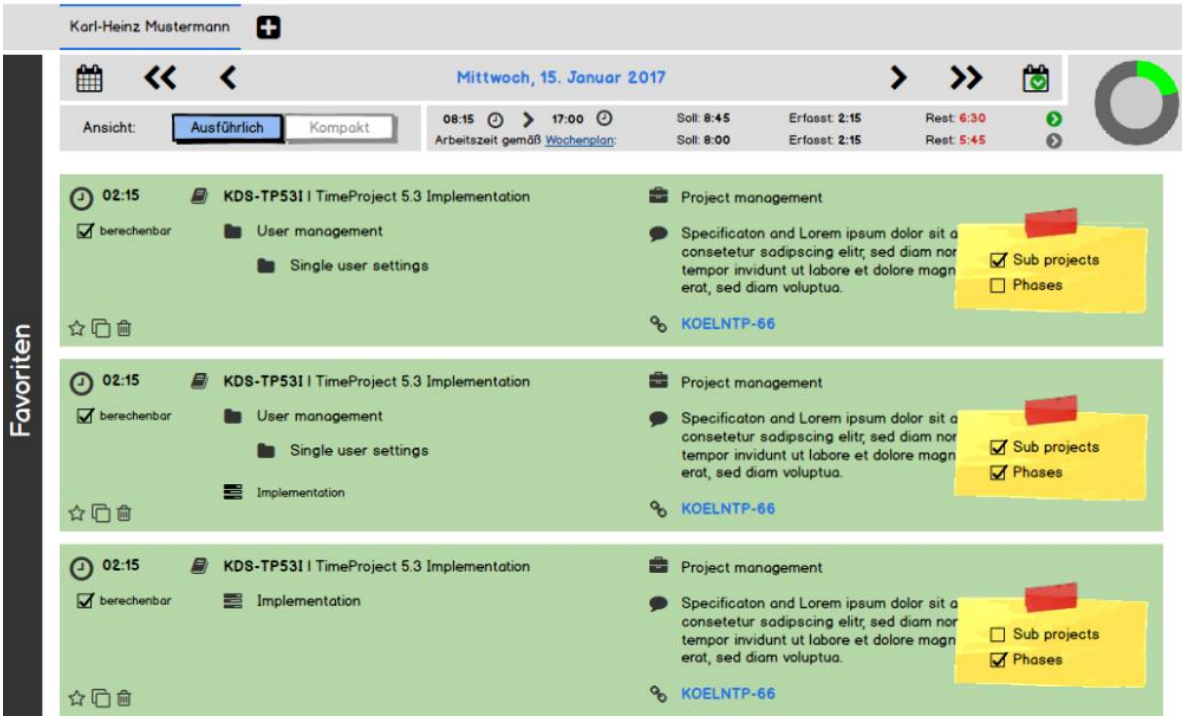


Figure 13: Autocomplete field for phases in the daily record item

Paragraphs above indicate the last important functionality which is the possibility to set Phases for daily reports as optional in system settings. Additionally, to either set the whole module Phases as optional.

4.5.2 Notification design

Significant part of requirements defined in Analysis chapter is about being able to notify the user about changes made in TP application. Therefore, this chapter reveals several following steps which need to be introduced and briefly described.

1. *Indicator of unread messages*

User can see an indicator at the bell icon that shows the number of unread messages. When a user clicks on the bell icon, a short menu appears that shows two links for important and regular notification list grid

2. *Notification list grid*

Displayed mockup has unfortunately been changed in terms of column “text” addition and “eye” icon was replaced for typical “detail” icon. Additionally, all notifications are by

default “MarkAsRead = false”. This state can be changed by deleting or opening a detail view of notification.

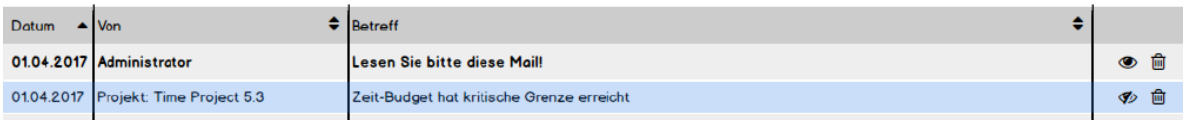


Figure 14: Notification list grid

3. *Detail view of notifications*
4. *Notification templates list grid*

As an Administrator or Administrator tenant a grid in the system settings can be opened to see all notification templates and select one of them for a detail view.

5. *Detail view of notification templates*

As an Administrator or Administrator tenant a detail view of a notification template can be opened and the content of the “title” and the “texts” for all languages changed. Additionally, the detail view stores noneditable “original text” value (if an error in text value editing occurs) and “status” radio buttons (editable severity of notification)

4.5.3 TP Localization

As phase requirement “UI localization” might suggest, the whole module must be localizable. At the beginning of design phase, the thesis proposed an extensible solution which allowed to translate not only phases module, but also all others. It is important to realize that this solution consists of two approaches.

1. *Kendo UI component localization*

Localization of Kendo UI components is handled by minified javascript localization files (such as kendo.messages.cs-CZ.min.js) offered directly from Telerik.

2. *General TP localization – Translation module*

TP localization was originally designed to use JSON localization files for each language that consisted of KEY(unique) and VALUE combination. This solution was later redesigned to a form of using database table instead. The final solution for general text

localization within TP application is described in activity diagram below (for bigger resolution, please visit Appendix).

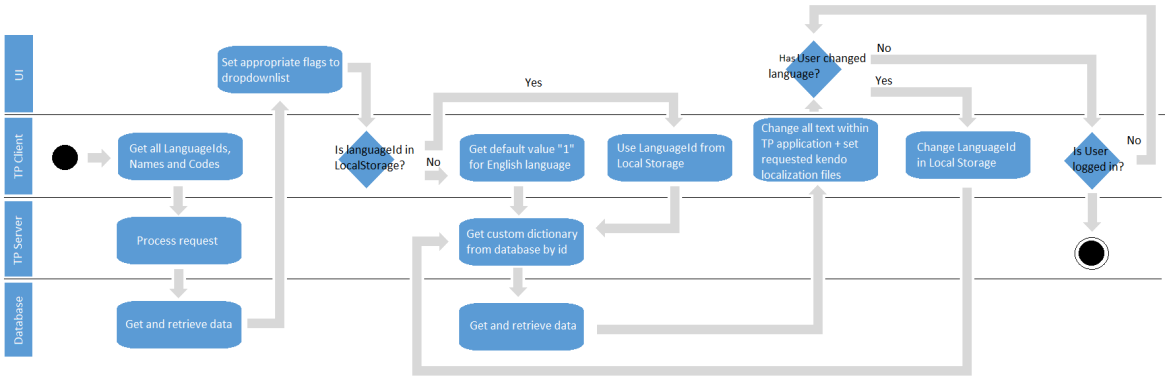


Figure 15: TP localization activity diagram

4.5.4 External interface – Jira concept

As it can be implied from requirements (in Analysis chapter), the connections between Jira server to TP database and TP server to Jira server must be established. For the first mentioned connection, it is necessary to follow Kepler add-ons documentation (the whole procedure is described in detail in Implementation chapter). The second connection is a little bit more complex. The trickiest part was to implement authentication. Three types of authentication are offered by Jira developer documentation – basic, cookie-based and OAuth authentication. Since Jira connection functionality is available only for Baader computer intranet, the basic authentication can be used. However, TP must later be ready for OAuth usage.

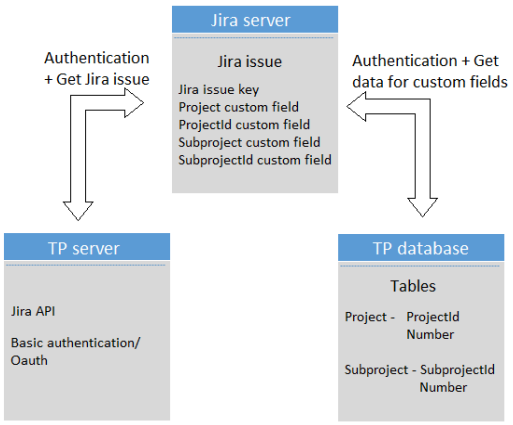


Figure 16: TP to Jira connection

Basic authentication

Because of the fact, that both user name and password are transferred as unencrypted base64 encoded text, the HTTPS[30] protocol must be used. Otherwise these credentials can be easily captured and reused (Franks, et al., 1999). For example, encoded text "michalkortan:secret" decoded to base64 looks like this: "bWljaGFsa29ydGFuOnNlY3JldA==".

OAuth

This chapter serves for brief mechanism description of OAuth 1.0 that Jira supports. The OAuth protocol enables Customers to access Protected Resources from a Service Provider via an API, without requiring users to acknowledge their Service Provider credentials to Customers. So, it is obvious that OAuth does not require user interface or interaction pattern (Atwood, et al., 2007). A couple of definitions need to be clarified to understand the OAuth mechanism:

- Customer – application or website; to access the Service Provider
- Service Provider – Web application
- User – An individual
- Protected Resources – Service Provider controlled data

[30] HTTP over SSL/TLS and HTTP Secure- protocol for secure communication

- Consumer Key – Consumers value to identify itself
- Consumer Secret – To establish ownership of the Consumer Key
- Request Token – Consumers value for obtaining authorization from User and exchange for an Access Token
- Access Token – Consumers value to get access to the protected resources
- Token Secret – Customers secret for establishing ownership of a given Token

The authentication mechanism than consists of three steps:

1. The Consumer obtains an unauthorized Request token
2. The Request Token is authorized by the user
3. Request Token is exchanged for an Access Token by Consumer

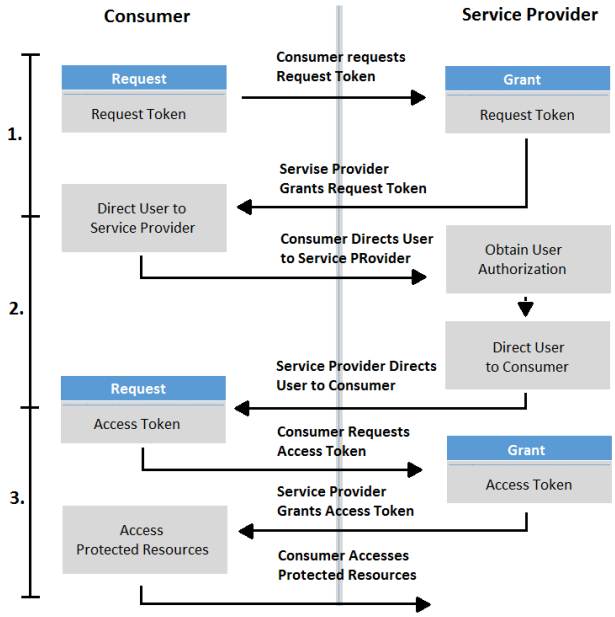


Figure 17: OAuth mechanism (Atwood, et al., 2007)

Following chapters detail the implementation of all data models followed by server and client implementation demonstrated on Phases module. Other modules similarly follow this implementation process, hence the thesis in further sections describes only selected parts. Further they describe used patterns and methods. In addition to that, they follow best coding practices such as naming conventions, commenting and so on.

Chapter 5

Implementation

This is the longest phase of software development cycle where the process of writing a source code begins. The objective is to transform the gathered information from analysis and design into code (Burback, 1998).

5.1 Data Model

A model using Toad data modeler was created in design phase. This application is also capable of scripts generation or even connecting to database and executing them. Following section shows and briefly describes generated script of phase module.

5.1.1 Phases script

It is important to realize that SQL scripts are not the only way to create relational database tables and relations. Microsoft SQL server management studio GUI[31], can be used instead. It consists of pretty straightforward process of creation (basically just click on create table and include relations in database diagram).

As a result of using Toad modeler with its script generation capability, the approach of SQL script tables and relations was chosen. First, what needs to be specified are tables creation which includes column name, data type(size) and if necessary constraint syntax. Secondly, appropriate primary and foreign keys. The example bellow shows only Phase table together with appropriate keys. The rest of the script for phase module can be found in Appendix section. All other scripts are stored on attached CD.

[31] Graphical User Interface

```
CREATE TABLE [dbo].[Phase]
(
 [PhaseId] Int IDENTITY(1,1) NOT NULL,
 [CreateDate] Datetime NOT NULL,
 [CreateUser] Int NULL,
 [ChangeDate] Datetime NULL,
 [ChangeUser] Int NULL,
 [IsDeleted] Bit DEFAULT 0 NOT NULL,
 [IsActive] Bit DEFAULT 1 NOT NULL,
 [Number] Nvarchar(20) NULL
)
GO
ALTER TABLE [dbo].[Phase] ADD CONSTRAINT [PK_ProjectPhaseId] PRIMARY KEY
([PhaseId])
GO
ALTER TABLE [dbo].[SideCost] ADD CONSTRAINT [Phase_SideCost] FOREIGN KEY
([PhaseId]) REFERENCES [dbo].[Phase] ([PhaseId])
GO
```

5.1.2 Phases model

When this script is executed on TP database using MS SQL Management studio, the following database structure is created. Figure is complemented by brief description that is mapped to requirements.

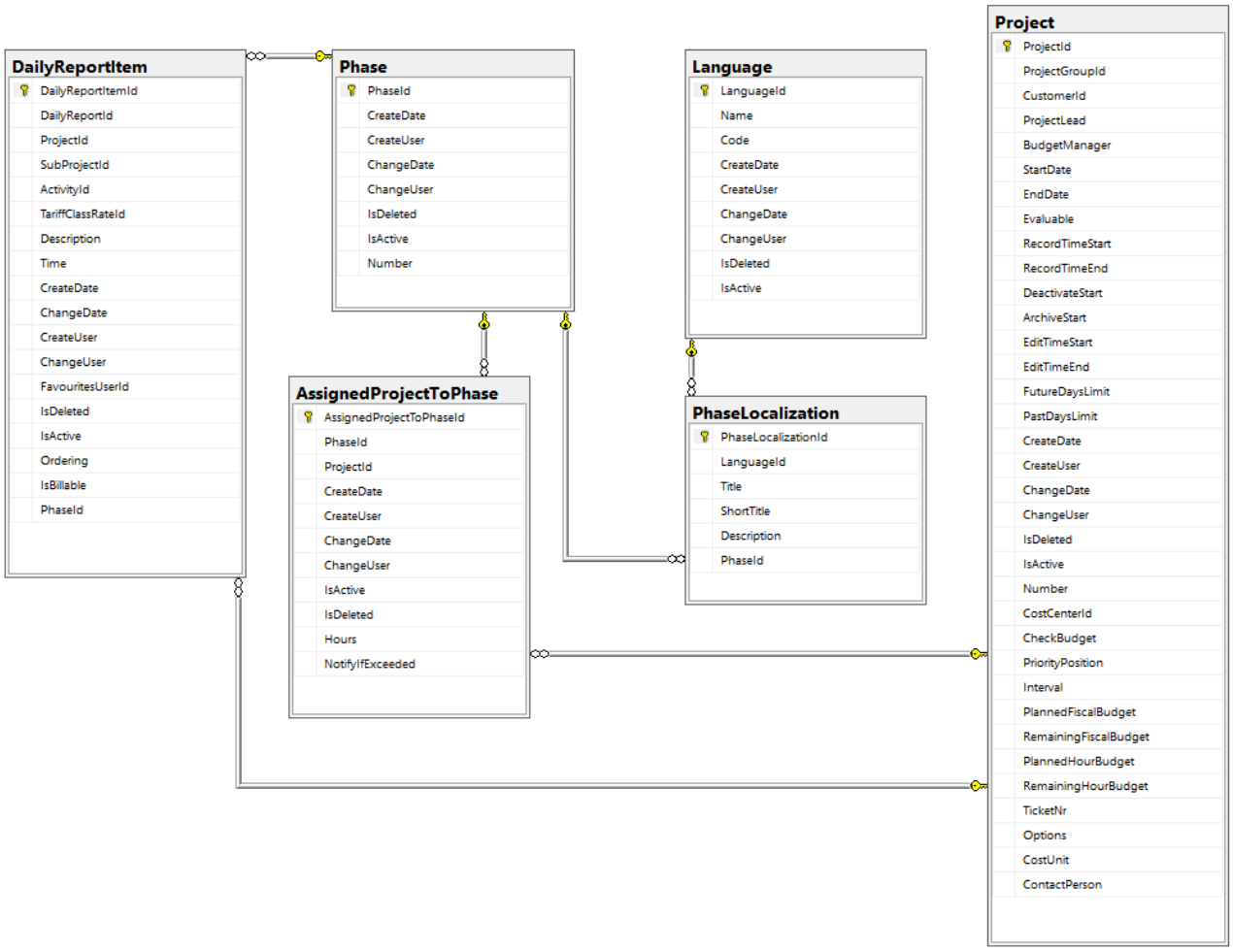


Figure 18: Phases physical model

There are several tables designed in a way to accomplish phases requirements. To be able to:

1) Localize phases into three different languages – PhaseLocalization
2) Assigned phase to project – M:N entity AssignedProjectToPhase

Phase, PhaseLocalization and AssignedProjectToPhase columns has been specifically designed to fulfill these requirements:

1) CreateDate, CreateUser, ChangeDate, ChangeUser – Logging/ Last changes logging
2) Number – Phase must include a string identifier for users
3) Title, ShortTitle, Desciption – Data localization into EN, DE, CZ
4) Hours – Set the duration of assigned project in phase
5) NotifyIfExceeded – Notify the user when set project hours in phase exceeded

In addition, IsDeleted and IsActive signifies whether specific row is deleted or active. The only requirement which is, so far, unfulfilled is the ability to notify the user that phase has been created, edited, deleted or exceeded.

5.1.3 Notification model

In order to fulfill the requirement for notification when new phase is created, edited, deleted and exceeded, the notification model has to be designed. Since scalability is one of the major focuses of TP 5 development, the model is designed to be applicable not only for phases module, but also for others. The final version of DB model can be found bellow (Figure 19).

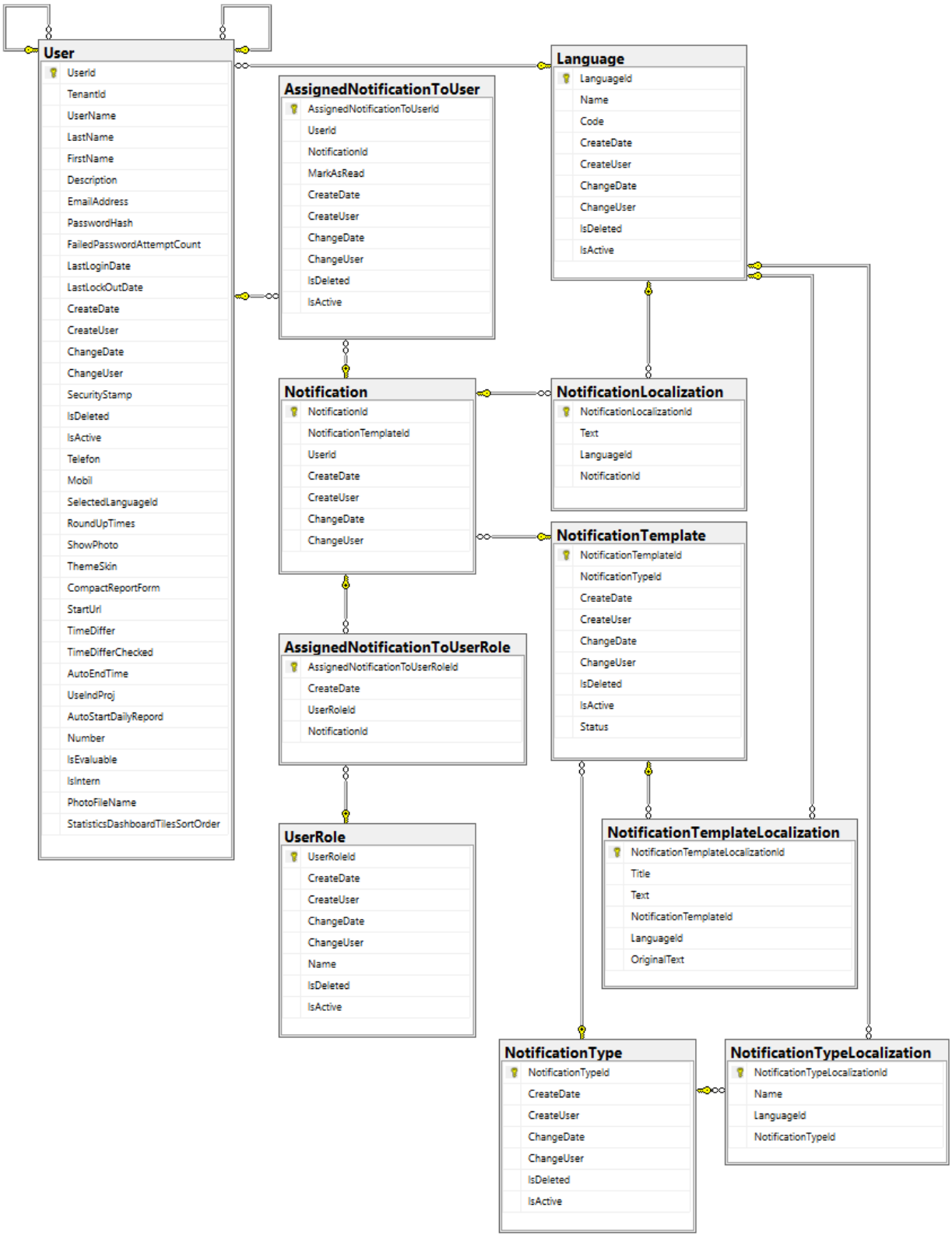


Figure 19: Notification physical model

Since many users can have many notifications as well as user roles can have many notification, two M:N tables – AssignedNotificationToUser and AssignedNotificationToUserRole exist. These tables include standard attributes mentioned in chapter above, however, there is one very important attribute called MarkAsRead, which is a self-explanatory bool value. The idea behind design of the other tables comes from several requirements:

1) Notification and NotificationLocalization – Notification must be stored in all languages
    a. Text – localized text of notification
2) NotificationTemplate and NotificationTemplateLocalization – Edit notification template in all languages
    a. Status – bool; important and regular notifications
    b. Title – read only notification subject
    c. Text – editable template text
    d. Original text – read only original template text
3) NotificationType and NotificationTypeLocalization – notification generation source
    a. Name – notification generation source name

5.1.4 External interface model

All attributes covered in ExternalInterface table are designed in such way to store all necessary data for establishing a connection among TP and other systems.

1) Name – user defined connection name
2) Host – server URL or IP address
3) Port and Protocol
4) UserName and PasswordHash as credentials

Since all this data is obviously not localizable, table ExternalInterfaceLocalization is there just for storing so called hints (whisperers) for user interface. ExternalInterfaceType than defines connection name for instance “Jira connection”.

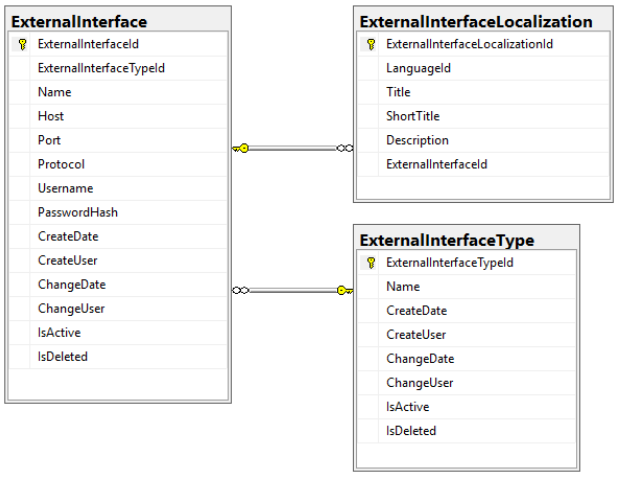


Figure 20: External Interface physical model

5.1.5 Translations model

The last substantial model to achieve requirements accomplishment is to create translations model. The ApplicationTranslation table has only one (relevant) relationship with ubiquitous Language table via LanguageId foreign key. This table further provides following attributes:

1) [Key] attribute stores unique string value
2) CurrentText attribute is a language specific string value
3) OriginalText exists due to translation administration (possibility to change CurrentText)

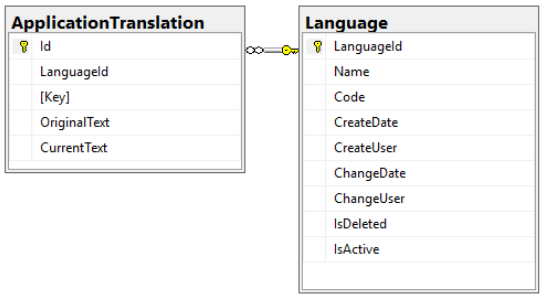


Figure 21: ApplicationTranslation physical model

5.2 Server-side implementation

It is important to mention that all code examples listed here are omitted for brevity in terms of not showing references, namespaces etc. Complete codes can be found on attached CD.

5.2.1 Generic Repository pattern

Necessity for creating repository pattern comes from the need for abstraction layer which separates data access layer and business logic. This pattern can be implemented with or without Unit of Work class which is used as a wrapper around repository and DbContext. Its responsibility is to ensure that all repositories use the same DbContext. This very handy class is very good practice, however, since a predominant part of requests has exactly one transaction, this class does not need to be implemented in this case. In addition, in cases where multiple transactions are necessary, transaction scope can be used.

Repository classes are typically implemented in ratio 1:1 for each entity type (such as Phase and Notification). Next chapter shows parts of EF implementations through repositories and configurations to services.

5.2.2 Entity framework

First, a context class (code below) needs to be implemented, which can be easily recognized by the fact that it derives from DbContext (in this case it derives from EfContext which derives from DbContext). This class establishes a “bridge” between database and domain/entity classes.

```
public partial class TimeProjectContext : EfContext
{
public TimeProjectContext() : base("DefaultConnection")
{
      this.Configuration.LazyLoadingEnabled = false;
      InitializePartial();
}
partial void InitializePartial();
partial void OnModelCreatingPartial(DbModelBuilder modelBuilder);
```

```
//Entity set
public DbSet<AssignedNotificationToUser> AssignedNotificationToUsers { get; set;
}
/*The rest DbSets for AssignedNotificationToUserRole, AssignedProjectToPhase,
Notification, NotificationLocalization, NotificationTemplate,
NotificationTemplateLocalization, NotificationType,
NotificationTypeLocalization,
Phase, PhaseLocalization, ExtenalInterface, ExtenalInterfaceType,
ExtenalInterfaceLocalization*/

//Fluent API
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
       //Configuration of domain classes
       base.OnModelCreating(modelBuilder);
       modelBuilder.Properties<string>().Configure(c => c.IsMaxLength());
       modelBuilder.Configurations.Add(new
       AssignedNotificationToUserConfiguration());
       //The rest of configurations for all entities
       OnModelCreatingPartial(modelBuilder);
}
```

Parameter passed in base constructor signalizes Code-First API that connection string exists (in web.config) thus it checks if database exists and then uses existing one or creates a new one.

Moving on to tables creation. The usual way to create tables in Code-first approach is to write domain classes and let the EF Code-first API do the rest. In addition, when using code-first conventions, it can eventually estimate which property is PK, FK and even data type. It is certainly one of the best ways for creating conceptual model. For TP, however, a reverse approach has been chosen since substantial part of the database has already been created. This approach employs EF reverse POCO generator, which reverse engineers an existing database and generates POCO classes, configuration mappings, even DbContext. A following class PhaseLocalizationConfiguration has been chosen to represent configuration file.

```
public partial class PhaseLocalizationConfiguration :
System.Data.Entity.ModelConfiguration.EntityTypeConfiguration<PhaseLocalization>
{
public PhaseLocalizationConfiguration() : this("dbo")
{
}

public PhaseLocalizationConfiguration(string schema)
{
ToTable(schema + ".PhaseLocalization");
HasKey(x => x.PhaseLocalizationId);

Property(x => x.PhaseLocalizationId).HasColumnName(@"PhaseLocalizationId")
       .IsRequired().HasColumnType("int")
       .HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema
       .DatabaseGeneratedOption.Identity);
Property(x => x.LanguageId).HasColumnName(@"LanguageId")
       .IsOptional().HasColumnType("int");
Property(x => x.Title).HasColumnName(@"Title")
       .IsRequired().HasColumnType("nvarchar").HasMaxLength(128);
Property(x => x.ShortTitle).HasColumnName(@"ShortTitle")
       .IsRequired().HasColumnType("nvarchar").HasMaxLength(32);
Property(x => x.Description).HasColumnName(@"Description")
       .IsOptional().HasColumnType("nvarchar").HasMaxLength(512);
Property(x => x.PhaseId).HasColumnName(@"PhaseId")
       .IsOptional().HasColumnType("int");

// Foreign keys
HasOptional(a => a.Language).WithMany(b => b.PhaseLocalizations)
       .HasForeignKey(c => c.LanguageId);
HasOptional(a => a.Phase).WithMany(b => b.Localizations)
       .HasForeignKey(c => c.PhaseId);
InitializePartial();
}
partial void InitializePartial();
}
```

EntityTypeConfiguration is the class which allows to configure entities and its properties. To do that, it applies several essential methods, among major belong:

- ToTable: table name configuration
- HasKey<TKey>: primary key configuration
- Property<T>: struct property configuration
- IsRequired, IsOptional: required and optional relationship configuration
- And so on

5.2.3 Repository layer

As it has already been said, this is the only layer responsible for preserving CRUD operations above entities. To handle such behavior, EF manages so called entity lifecycle. Its purpose is to incorporate entity state based on operations executed in it via DbContext. Among the DbContext functionalities belongs a feature known as Change Tracking, that can store all entity states: Added, Modified, Detached, Unchanged and Deleted (Figure 22)

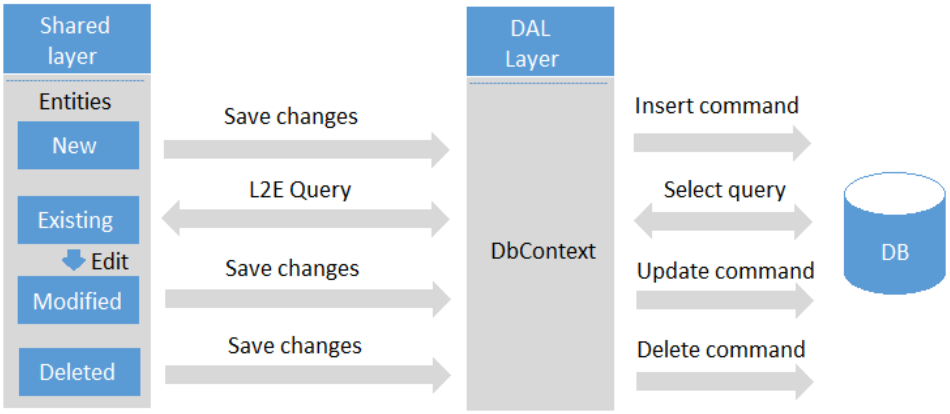


Figure 22: EF Entity Lifecycle (EntityFrameworkTutorial, 2016)

The only change handled by DbContext itself is from unchanged to modified state. The rest of the figure speaks for itself, new entity is in Added state, until insert command is executed and so on.

EDM[32] handles several query types when querying database. On the first hand, there is strongly typed code with query comprehension syntax – LINQ[33] to Entities. On the other hand, traditional string based query – Entity SQL. In today development, there is no reason for string being used as queries because of the fact it increases the chance of making a typo. LINQ to Entities offers two ways how to access data: LINQ Method syntax and LINQ Query syntax. Each of these are better in different scenarios. However, the thesis applies mostly LINQ Method syntax. Important is to realize that each repository such as

[32] Entity Data Model
[33] Language Integrated Query

PhaseRepository or NotificationRepository, should handle just Get requests and operations, above entities are then handled be EfRepository. The following code displays method for Get operation for Phases. Complete source codes can be found on attached CD. In case of interest, in CD (section LINQ method syntax vs LINQ Query syntax) the same query for retrieve notification templates written in both syntaxes can be found.

```
public DataSourceResult GetPhasesForGrid(DataSourceRequest request, int
languageId)
        {
            var query = Set.Where(w => w.IsActive).Select(p => new
            {
               Id = p.PhaseId,
               Title = p.Localizations
             .Where(l => l.LanguageId == languageId).Select(l =>
                   l.Title).FirstOrDefault(),
               ShortTitle = p.Localizations
             .Where(l => l.LanguageId == languageId).Select(l =>
                         l.ShortTitle).FirstOrDefault(),
               Description = p.Localizations
             .Where(l => l.LanguageId == languageId).Select(l =>
                         l.Description).FirstOrDefault(),
               CreateDate = DbFunctions.TruncateTime(p.CreateDate),
               ChangeDate = p.ChangeDate != null ?
                         DbFunctions.TruncateTime((DateTime)p.ChangeDate) :
                         null,
               CreateUser = p.CreateUser != null ? p.CreateUser.FirstName + " "
                         + p.CreateUser.LastName : "",
               ChangeUser = p.ChangeUser != null ? p.ChangeUser.FirstName + " "
                         + p.ChangeUser.LastName : "",
               IsDeleted = p.IsDeleted,
               IsActive = p.IsActive,
               Number = p.Number
            });
            return query.ToDataSourceResult(request);
        }
```

LINQ method syntax structure consists of extension method (standard query operator for instance Where or Select) and lambda expression. Lambda expression is used quiet a lot during TP development, thus it is important to mention that it is just a shorter way of representing anonymous method instead of using delegate repeatedly. What is worth to mention is FirstOrDefault method which returns an element from the zeroth index from the collection.

.

5.2.4 Service layer

Major responsibilities of this business logic layer are validations and operation exposure to API controllers. Especially all Get requests should just pass the data to the next layer as displayed in this method:

```
public DataSourceResult GetPhasesQueryable(DataSourceRequest request, int
languageId)
        {
            return phaseRepository.GetPhasesForGrid(request, languageId);
        }
```

For the rest Create, Update and Delete operations it is necessary to implement business logic. Necessary data is provided by higher layer (Web layer) in form of Model.

```
public int CreatePhaseBasicData(PostPhaseBasicModel toModel)
        {
            var phase = new Phase
            {
                Number = toModel.Number,
                Localizations = toModel.Localizations
            };
            phaseRepository.Add(phase);
            phaseRepository.SaveChanges();
            return phase.PhaseId;
        }

public void UpdatePhaseBasicData(PostPhaseBasicModel toModel, int id)
        {
            var oldPhase= phaseRepository.FindById(activity1 =>
                          activity1.PhaseId== id,
                          activity1 => activity1.Localizations);

            if (oldPhase == null) throw new
                          NotFoundException("Phase doesn't exist");
            oldPhase.Number = toModel.Number;

            // Remove localizations
            foreach (var localization in oldPhase.Localizations.ToList())
            {
                if (toModel.Localizations.All(c => c.LanguageId !=
                          localization.LanguageId))
```

```
                oldPhase.Localizations.Remove(localization);
            }

            //  Add or update localizations
            foreach (var localization in toModel.Localizations)
            {
                var existingChild = oldPhase.Localizations
                    .SingleOrDefault(
                        c => c.LanguageId == localization.LanguageId);

                if (existingChild != null)
                {
                    existingChild.Description = localization.Description;
                    existingChild.ShortTitle = localization.ShortTitle;
                    existingChild.Title = localization.Title;
                    existingChild.LanguageId = localization.LanguageId;
                }
                else
                {
                    oldPhase.Localizations.Add(localization);
                }
            }
            phaseRepository.SaveChanges();
        }

public void DeletePhase(int id)
        {
            var old = phaseRepository.FindById(p => p.PhaseId== id);
            if (old == null) throw new NotFoundException("Phase doesn't exist");
            phaseRepository.DeactivateById(id);
            phaseRepository.SaveChanges();
        }
```

In these examples, it can be seen how EfRepository handles entity lifecycle. Add accepts TEntity and lets DbSet handle the rest. Delete methods default behavior has been modified – it serves as so called “fake delete” which sets IsDeleted property to “true” value. Update method accepts TEntity as well, however, this method is handled directly by EF change tracking functionality. Next there are a couple of very specific methods such as DeactivateById which is designed to set property IsActive to false. This method is used everywhere where it is not desirable to remove entity. Get request on repository layer must then by customized in terms of filtering data where IsActive property is set to True. Finally, EF wrapper method for CUD operations SaveChanges is executed.

5.2.5 ASP.NET WEB API

The basic way of communication between controller and service (even between service and repository) would include controller class which creates the instance of service class. Nonetheless, the idea of hardcoded dependency is not optimal for several reasons:

- Hard to unit test
- In case of replacing service implementation, controller modification is necessary
- And others

To eliminate these tight coupled classes, the Dependency injection pattern is used. More specifically constructor injection realized by IoC Unity container which performs managing dependencies. The code below demonstrates DI in PhasesController.

```
private readonly PhaseService _phaseService;
public PhasesController(PhaseService phaseService)
        {
            this._phaseService = phaseService;
        }
```

Now, when the loosely coupled concept is described, let us have a look at implementation part of ASP.NET WEB API. It is built at the top of server side implementation for exposing HTTP services and data. It accepts XML and JSON formats by default and paste/retrieve serialized data into the body of HTTP response/request message. Before client performs any HTTP request the URL of requested API and HTTP method must be specified.

- WEB API 2 provides attribute routing to define routes – PhasesController class uses [RoutePrefix("api/phases")] because all methods inside this class start with it and then [Route("queryable")] which concatenates with RoutePrefix.

- HTTP methods – also attribute based [HttpGet], [HttpPost], [HttpPut], [HttpDelete] and others
    - GET – Read operation
    - POST – Create operation
    - PUT – Update operation
    - DELETE – Delete operation

```
[Route("queryable")]
[Authorize]
public DataSourceResult GetPhasesQueryable(
     [ModelBinder(typeof(WebApiDataSourceRequestModelBinder))]
      DataSourceRequest request)
        {
            var languageID = GetCurrentLanguageId();
            var p = _phaseService.GetPhasesQueryable(request, languageID);
            return p;
        }
```

Interesting fact about this method is the absence of [HttpGet] attribute. When ASP.NET WEB API convention is followed – HTTP method begins with Get, Post…, the attribute is not essential (Class diagram for a Web layer can be found in Appendix). Then there are Kendo.MVC.UI specific classes DataSourceResult (as a return type) and WebApiDataSourceRequestModelBinder especially designed for Kendo grid GET requests. Notice Authorize attribute which allows only signed user to request this method. In the code below, what can be seen, is that the specification of rights for user roles is also possible.

```
[Route("")]
[HttpPost]
[ValidateModel]
[Authorize(Roles = "Admin, Admintenant, Support, Supervisor")]
public HttpResponseMessage CreatePhase(PostPhaseBasicViewModel
basicPhaseViewModel)
        {
            int id =
             _phaseService.CreatePhaseBasicData(basicPhaseViewModel.ToModel());
            var uri = Url.Link("GetPhaseById", new { id });
            var response = new HttpResponseMessage(HttpStatusCode.Created);
            response.Headers.Location = new Uri(uri);
            return response;
        }
```

The code above represents Post method marked with ValidateModel attribute. It is a more convenient way of writing “if (ModelState.IsValid){}” inside all methods require validation. Received model is validated according to ViewModel. Following code represents PostPhaseBasicViewModel which has attributes such as Required and MaxLenght(20) according to which validation occurs.

```
public class PostPhaseBasicViewModel
{
        [Required]
        public List<PhaseLocalizationViewModel> Localizations { get; set; }

        [Required, MaxLength(20)]
        public string Number { get; set; }

}
```

Finally, when server-side implementation is complete, it is time to test its functionality. To do that, Postman which allows to make all necessary requests, has been used. Firstly, make a POST request with filled credentials to gain access token. Secondly, add this token inside the header of required request. Lastly, fill the HTTP method and URL. For instance, in case of using GET method with URL: http://localhost:65181/api/phases/queryable, the response in JSON format would look like this:

```
{
  "data": [
    {
      "id": 1,
      "title": "Phase 1",
      "shortTitle": "P1",
      "description": "Phase 1 description",
      "createDate": "1994-01-01T00:00:00",
      "changeDate": "2017-03-07T00:00:00",
      "createUser": "Testovaci Uživatel",
      "changeUser": "Testovaci Uživatel",
      "isDeleted": false,
      "isActive": true,
      "number": "PH-01"
    },
    // omitted for brevity
  ],
  "total": 8,
  "aggregateResults": null,
  "errors": null
}
```

5.3 Client-side implementation

At the time of Phases implementation, necessary environment has already been created. Therefore, the configuration process of development environment is not the subject of this Thesis. However, a few things need to be clarified before specific implementation arises.

The reason behind determining why to use TypeScript instead of ES5(regular Javascript) is collaboration of two greatest companies Microsoft and Google. As a result of choosing TypeScript as a primary language for developing TP 5, the Node.js with npm[34] tool and Visual Studio Code as an editor need to be installed. In addition to use utility to automate tasks like creating projects and so on, Angular-CLI[35] must be installed too. TypeScript is a superset of ES6[36] as well as ES6 is superset of ES5. However very few browsers support ES6 so far, for that reason so called transpilers (transcompilers) exist which take TypeScript code as an input and outputs ES3 or ES6 that is supported nearly by all browsers. There are another several essential tools to build required modules in TP 5. Among them belong:

- Webpack – module bundler; decrease the number of requests and overall size; managing tree-shaking = remove unused part of JS code etc.
- RxJS – Observables, BehaviorSubjects, Subject; especially used for UI based applications- functional-reactive programming
- Ngrx – build on Redux and RxJs powered state management which internally uses BehaviorSubjects
- Gulp – task runner which includes gulp-sass plugin as a connector on node-sass
- And others…

[34] Node Package Manager for JavaScript
[35] Angular Command-Line Interface
[36] ECMAScript 6

The last issue that needs to be mentioned is which UI library to use. Since the selection was predefined by Baader computer because of purchased license together with great experience, the Kendo UI was the best choice. Following chapters describe Phase implementation of each building blocks.

5.3.1 Template

Templates operate with Angular specific template syntax (almost all HTML syntax is supported) to display data and handle user events using data binding. Since all developed modules (Phases, Notifications and so on) must follow the same design as in the rest of the application, the next section focuses only on Angular template syntaxes complemented with code examples and mockup with brief structure description.

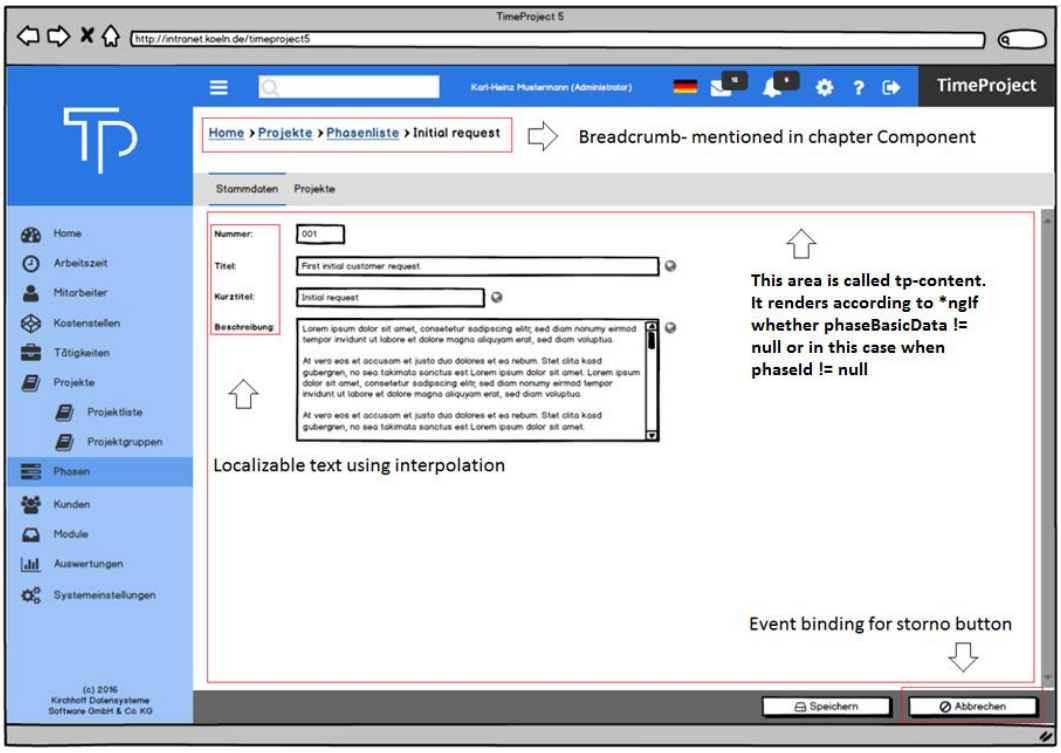


Figure 23: Phase detail view

Build-in directives

- Structural directives
  - *ngIf, *ngFor, *ngSwitch
  - Starts with asterisk prefix (*)
  - Responsible for shape or reshape DOM (Document Object Model) structure
  - Example:
    - ```
      <div *ngIf="projectPhaseDetail == null"
      class="tp-form"> 
      ```
  - Description: when the condition is satisfied the <div> element is rendered
- Attribute directives
  - ngModel, ngStyle, ngClass
  - Applied to elements as HTML attributes
  - Listening and modification of HTML elements, components etc.
  - Example:
    - ```
      <input type="text" readonly="true"
      [(ngModel)]="projectPhaseDetail.projectTitle"
      formControlName="projectTitle" />
      ```
  - Description: two-way data binding to form elements

Binding syntax

- One-way data binding – data source to view
  - Interpolation
    - {{expression}}
  - Property binding
    - [target]= ”expression”
  - Attribute, style and class binding

- One-way data binding – view to data source
    - Event binding
        - (event)= ”statement”
        - Example:

```
<toolbutton alignRight="true"
(clickButton)="storno()"
iconClass="glyphicon glyphicon-ban-circle">
{{ 'STORNO_BUTTON' | translate }}
</toolbutton>
```

- Two-way data binding
    - [(target)] = “expression”
    - Display and update property when changes are made

Binding syntaxes example:

```
<select [(ngModel)]="projectPhaseDetail.hours" formControlName="hours">
      <option *ngFor="let item of timeOptions" [value]="item.value">
      {{item.text}}</option>
</select>
```

Finally, template tells Angular how to render the component. To do that, specific tag placed inside a template must be implemented and defined inside selector of components metadata. In other words, when bootstrapping, a class Angular looks for a specified selector for instance *<tp-phases-list> (*see section Component), finds it, instantiates an instance of the class and renders it inside the *<tp-phases-list>* tag.

5.3.2 Component

All applications written in Angular 2 are nothing but a tree of components which have a top-level component called AppComponent (application itself) and the rest of them are children. The application is a component that recursively renders other/child components which are composable (possibility of building large components from smaller).

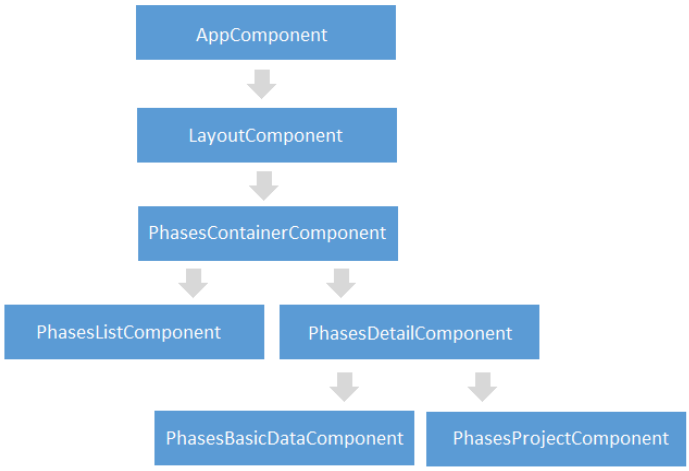


Figure 24: Phases tree diagram

Component class

As it has already been mentioned, component composes of the combination of HTML template and component class. The minimal component would look like this:

```
import { Component } from "@angular/core";
@Component({
    selector: "tp-phases-list",
    templateUrl: './phases-list.component.html'
})

export class PhasesListComponent {
    constructor() {}
}
```

The import rules of ES6 follow the convention: import {Something} from ‘./some/path’; Then there is a typescript decorator function @Component which employs metadata objects:

- selector – display component inside “tp-phases-list”
- templateUrl – separate HTML template definition (see section above).

Now, let us have a look at body implementation of this class. To achieve user interface navigation in application, breadcrumb component has been developed.

```
breadcrumb: IBreadcrumbSettings = {
        items: [
            {
                display: "MENU_HOME", localized: true,  link: ["/"]
            },
            {
                display: "MENU_PHASES", localized: true,  link:
                    ["/phases"]
            },
            {
                display: "MENU_PHASES_LIST", localized: true, link:
                    ["/phases-list"]
            }]
    };
```

This component implements interface with couple of properties. In this case, it uses items property which is a type of IBreadcrumbItem interface. Display property is a string in breadcrumb chain where if Localized property is true then Display string is used as a Key for translation pipe (more about translation pipes and others in section Translations). And finally link property is used to navigate user to desired destination.

The next very important Angular 2 building block is Dependency Injection (hereafter DI). When module X needs to communicate with other module Y it means that Y is a dependency of X. This is solved by simple import of a file like this: *import {Y} form 'Y';* and then use it in code like this: *Y.someMethod()*. This approach is undoubtedly sufficient, however, it leads to testing difficulties and impossibility of implementing some patterns. DI on the other hand works on IoC (Inversion of Control) principle also known as "do not call us, we will call you". It is obvious from this that DI is a step in achieving loosely coupled architecture (Google, 2016). The following figure represents the way DI is handled and code below shows the actual implementation.

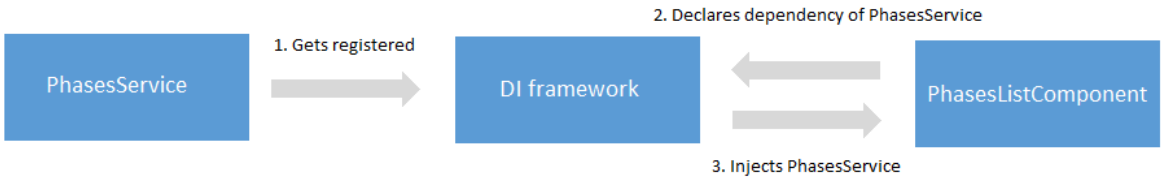


Figure 25: DI framework

```
import { PhasesService } from '../../../shared/services/phases.service';
//omitted for brevity
constructor(private phasesService: PhasesService) {}
```

Since the required behavior is to render grid with data in time whenever the component is instantiated, the implementation must be placed inside the body of constructor. First two JavaScript functions are bound to click commands (detail and delete obtained in Kendo UI grid) and then referenced. Then there is phasesListOptions which is a type of Kendo.ui.GridOptions interface. Kendo UI provides TypeScript definition file which enables strongly-typed access to all widgets and their configuration. Among the TypeScript typing configuration belong:

- Filename specification in case of grid data to excel export
- DataSource specification
    - url – corresponds to actual WEB API route (see ASP.NET WEB API chapter)
    - setRequestedHeader – token and languageID
- Columns specification
    - field – name of Key from received request
    - title – displayed name (more about localizations in TP localization chapter)
    - command – click event handled by mentioned functions

To better imagine and understand the specification for Kendo UI grid, mockup together with code is presented.

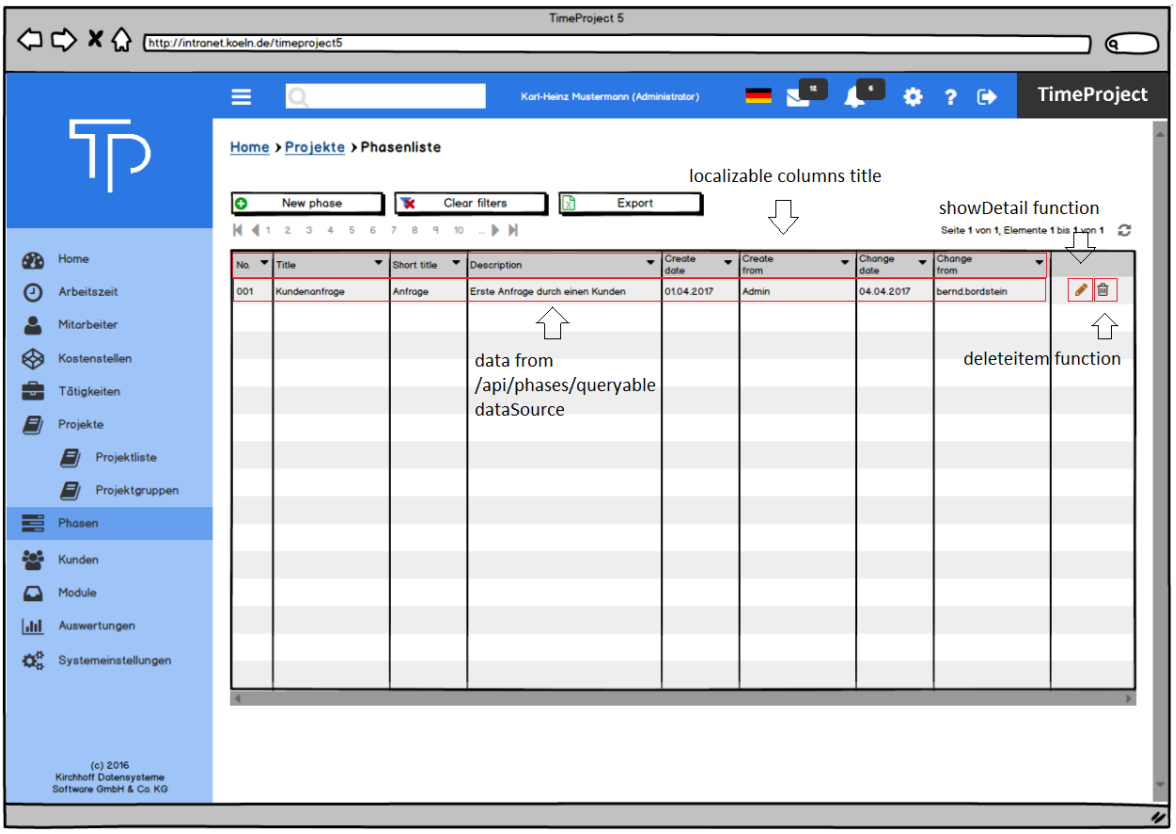


Figure 26: Phase grid

```
let self = this;
        let showDetail = function (event: any) {
            self.showDetailsInternal(this, event);
        }
        let deleteItem = function (event: any) {
            self.deleteItemInternal(this, event);
        }
         this.phasesListOptions = {
            excel: {
                fileName: "phase-list.xlsx"
            },
            dataSource: {
                type: "aspnetmvc-ajax",
                transport: {
                    read: {
                        type: 'GET',
                        url: URL.SERVER_URL + "/api/phases/queryable",
                        beforeSend: function (req) {
                            req.setRequestHeader("Authorization",
                                "Bearer " + localStorage.getItem("token"));
                            req.setRequestHeader("languageID",
                                localStorage.getItem("TPLang"));
```

```
                    }
                }
            },
            //omitted for brevity
        columns: [
            {
                field: "number",
                title: this.translate.instant(
                                "PHASES_LIST_GRID_NR_COLUMN"),
                filterable: {
                    cell: {
                        suggestionOperator: "contains",
                        minLength: 2
                    }
                }
            }
                //omitted for brevity -> Other columns: shortTitle...
            {
                //omitted for brevity
        command: [
        { name: "Detail", text: "", click: showDetail, className:
                "glyphicon glyphicon-pencil transparent-button" },
        { name: "Delete", text: "", click: deleteItem,className:
                "glyphicon glyphicon-trash transparent-button" }],
                width: "90px"
            }
        ],
        //omitted for brevity
};
```

As mentioned above, click methods are handled by JavaScript function which calls appropriate functions. One of this functions is showDetailsInternal that redirects user to *'phases/phases-detail?id=*' + phase identifier. This identifier is later used to fetch phase basic data using service layer.

5.3.2 Services

Service is basically nothing but a class with few differences. First of all, it must be provided with @Injectable decorator. And second of all, it must be registered with an Angular injector. One of the major tasks of this layer is to fetch data from server for instance to perform typical CRUD operations. So, it is obvious that this layer should contain all application logic and make services accessible for components through dependency injection. In addition, component does not talk directly to the Angular Http client in order to perform requests – also a responsibility of Service layer.

The data flow between component and service is pretty straightforward for GET and DELETE methods. They just accept ID (except queryable GET requests) as a parameter in appropriate request and put it in URL. Code below represents a fetchPhaseBasicData which is triggered by click event on grid detail.

```
public fetchPhaseBasicData(phaseId: number) {
        this.http.get<PhasesBasicDataModel>(
                    URL.SERVER_URL + '/api/phases/' + phaseId + '/basic',
             (basicDataHttp) => this.mapBasicData(basicDataHttp),
                    this._busy)
            .subscribe((basicData) => {
             this.store.dispatch(new actions.ShowPhaseBasicData(
                    basicData));
            });
    }
```

Get is a generic method, which returns an Observable of HTTP responses (Observable<T>). In all other HTTP responses return Observable<Response> from the RxJS library[37]. Observable is basically a stream of events published by a source. In order to listen to this stream, subscribe is used. Subscribe specifies the action to take when request delivers success or fail event (data in payload or error). To be able to specify the action to take, ngrx/store is used (described in chapter below).

[37] Reactive Extensions library for composing asynchronous and event-based programs

The last thing to mention about code above is *mapBasicData* method. Its responsibility is to map data from incoming *GetPhaseBasicHttpModel* to *PhasesBasicDataModel* to be able to use them for PUT and POST requests.

```
export class PhasesBasicDataModel {
    phaseId: number;
    number: string;
    title: string;
    shortTitle: string;
    description: string;
}
```

All incoming responses including attached ViewModel from WEB API must have its own HttpModel. To make creating HttpModels easier, the Typewriter tool for Visual Studio has been used. Once the server side for, let us say, updating a phase is implemented, all that needs to be done is to build solution and Typewriter handles HttpModels generation.

Ngrx/store

Is a RxJS powered state management for Angular applications inspired by Redux. It is composed of three main blocks: store, actions and reducers.

- Store – client side "single source of truth" which represents relevant application state
    - Can be thought of as a client database
- Reducers – pure function that takes the previous state and an action, and returns the next state
    - Can be thought of either as a database table or event store
- Actions – state update: Dispatch -- Reducers -- New state -- Store

5.3.3 Modules

Every application has at least one Angular module – root module that is bootstrapped to launch the application. All modules must include @ngModule decorator with metadata objects:

- Import – array consists of NgModule classes; consolidating features into discrete unit
- Declaration – array of declarables; components, directives and pipes
- Bootstrap – array of components supplied for bootstrapping process (inserting to DOM)

TP is designed to use app.module as a root module that imports several other modules (such as shared.module or portal.module.). These modules import others etc. etc. => tree structure. A short version of phases.module with a structure can be seen described above.

```
import { SharedModule } from '../shared/shared.module';
import { PhasesContainerComponent  } from './components/phases-container/phases-
container.component';
import { PhasesBasicDataComponent  } from '../phases/components/phases-
detail/phases-basic-data.component';
@NgModule({
  imports:  [
      SharedModule
    ],
  declarations: [
      PhasesContainerComponent,
      PhasesBasicDataComponent
      ],
  bootstrap:    [  ],
  exports: [ PhasesContainerComponent ]
})
export class PhasesModule {}
```

5.4 Notification module

As it was emphasized at the beginning of Implementation chapter, the thesis introduces only selected implemented parts from now on. Thus, it is important to realize that whenever any functionality needs data from the database, it is necessary to implement:

- Entities, Configuration files, Repositories, Services and API Controllers on Server side
- Templates, Components and Services on Client side

In case of Phase module, the notification generation must be called whenever a CreatePhaseBasicData, UpdatePhaseBasicData, DeletePhase and DeleteProjectInPhase methods are executed. However, notification generation is scalable enough to be applicable to all methods across TP server implementation. The implementation itself uses TransactionScope which major responsibility is to support transactions from a code block. This code block usually contains two transactions. TransactionScope assures that whenever one of the transactions fails, no other transaction is executed. The thesis already presented the code below, however this code is now no longer omitted for brevity.

```
public int CreatePhaseBasicData(PostPhaseBasicModel toModel, string userName,
int userId)
        {
            var phase = new Phase
            {
                Number = toModel.Number,
                Localizations = toModel.Localizations
            };

            using (TransactionScope scope = new TransactionScope())
            {
                _phaseRepository.Add(phase);
                _phaseRepository.SaveChanges();
                _notificationRepository.GenerateNotification(
                   NotificationTemplateEnum.createPhaseTemplate,
                   new { userName, phaseTitle = toModel.Localizations.Select(
                   s => s.Title).FirstOrDefault() }, userRoles, userId);
                _notificationRepository.SaveChanges();
                scope.Complete();
            }
```

```
            return phase.PhaseId;
        }
```

The GenerateNotification method which takes four parameters is applied.

1. Enumerator which stores the right identifier for NotificationTemplate – more programmer friendly approach than passing the identifier right away
2. Object which stores replacement parameters – NotificationTemplate text contains "tags" that are in generation notification process replaced by these parameters
3. List of user roles for which the notification is assigned
4. Identifier of user who created notification (by method execution)

```
public void GenerateNotification(NotificationTemplateEnum templateId, object
replacementParams, List<UserRoleEnum> userRoles = null, int userId = 0)
        {
// Assigned replacementParams to Dictionary – Later used for Replacer method
            Dictionary<String, String> replacements =
replacementParams.GetType().GetProperties().ToDictionary(x => x.Name,
x => (string)x.GetValue(replacementParams));
                // new Notification Entity creation
                var notification = new Shared.Entities.Notification
                { /* .... */ };

                if (userRoles != null)
                {
                    // method for get all user Ids in specified role
                    foreach (var item in GetUserIdsByRole(userRoles).Distinct())
                    {
                        // new AssignedNotificationToUser entity creation
                        var assignmentToUser = new AssignedNotificationToUser
                        { /* .... */ };
                    }
                }
                if (userRoles != null)
                {
                    // new AssignedNotificationToUserRole entity creation
                    (entity just for logging purposes)
                 foreach (var item in userRoles)
                    {
                   var assignmentToUserRole = new AssignedNotificationToUserRole
                        { /* .... */ };
                    // three records in NotificationLocalization table are
                    created for each Notification
                    // new NotificationLocalization entity creation for English
                    language
           var notificationLocalizationEN = new NotificationLocalization
           {
```

```
            Text = Replacer(_context.NotificationTemplateLocalizations
             .Where(l => l.LanguageId == (int)LanguageEnum.English &&
             l.NotificationTemplateId == (int)templateId).Select(s =>
             s.Text).FirstOrDefault(), replacements),
                    LanguageId = (int)LanguageEnum.English,
                    NotificationId = notificationId,
                };
                    // new NotificationLocalization entity creation for German
                    language
                var notificationLocalizationDE = new NotificationLocalization
                { /* .... */ };
                    // new NotificationLocalization entity creation for Czech
                    language
                var notificationLocalizationCS = new NotificationLocalization
                { /* .... */ };
                SaveChanges();
        }
```

As always, the code was as omitted for brevity as possible. In the code above, the noteworthy method can be seen – Replacer. It accepts template text (text with “tag” or “tags” – for instance “Phase <phaseTitle> has been created by <userName>”) extracted from NotificationTemplateLocalizations table and previously mentioned replacement parameters. Replacer than uses Regex.Replace() to do the rest.

```
public string Replacer(string templateText, Dictionary<String, String>
replacements)
        {
            string pattern = @"(<)(.*?)(>)";
            var replaced = Regex.Replace(templateText, pattern, (match) =>
replacements[match.Groups[2].Value]);
            return replaced;
        }
```

5.5 Translation module

First what needs to be implemented is dropdown list located on a login page with corresponding flags.

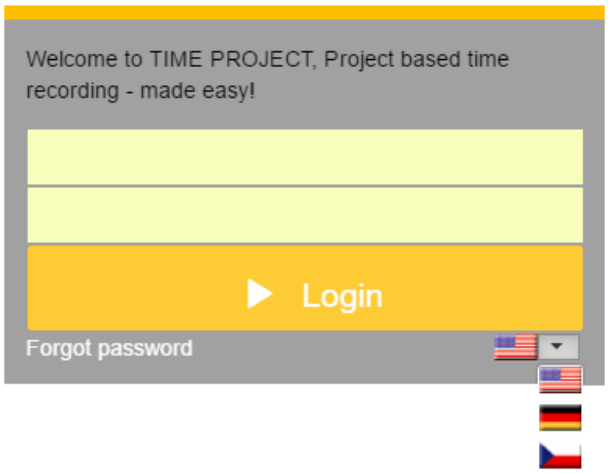


Figure 27: TP Login page language drop-down list

For better understanding of following text, it is recommended to list Appendix chapter (Translation module – Activity diagram). The change of the language is handled by a login component which requests language service to contact API (translation controller). According to languageId, the repository layer than query the database in order to retrieve requested custom dictionary. Among the major attributes in this dictionary belongs a LanguageId, Key (unique value) and Current text. LanguageId determines which Key and Current text are included in dictionary. The data contained in Key attribute is characterized by upper case notation for instance “STORNO_BUTTON”. CurrentText attribute stores following values for this Key:

- For [Key] = “STORNO_BUTTON” is English CurrentText = “Cancel”
- For [Key] = “STORNO_BUTTON” is German CurrentText = “Abbrechen”
- For [Key] = “STORNO_BUTTON” is Czech CurrentText = “Zrušit”

The actual translation is than handled by interpolation or using instant method. Interpolation has already been introduced in Angulars binding syntax section as a one-way

data binding. Additionally, it uses translate pipes: {{ 'STORNO_BUTTON' | translate }}. The value inside is nothing else but a Key, which is replaced by CurrentText value in the moment of making language choice.

Finally, the minified files (kendo.messages.cs-CZ.min.js and so on) directly from Telerik must have been imported to be able to localize all Kendo UI components.

5.6 External interface – Jira concept implementation

5.6.1 Jira server to TP database

First, what needs to be implemented are necessary custom fields provided by Kepler add-ons documentation. These fields allow to import a field value from external database. The value is selected by the user from the set of results (as a select list or autocomplete) retrieved be executing a query on the database. To be able to modify Jira server in such way, the administrator rights must be gained. It would be very unwise to test the connection in full version, that is why the local version of Jira server has been used. After successful connection and testing it was implemented in full version under a control of an administrator.

The whole process consists of:

- Databasecf add-on installation
- Configuration:
    - Database information and Database information custom fields
    - JNDI[38] datasource configuration

Database information and Database information custom fields

After a successful databasecf installation must be checked whether database custom field are enabled. Then it is necessary to create requested custom fields: Database

[38] Java Naming and Directory Interface

information for Project and Subproject number; Database child information for ProjectId and SubprojectId. The following figure shows how these custom fields are configured.

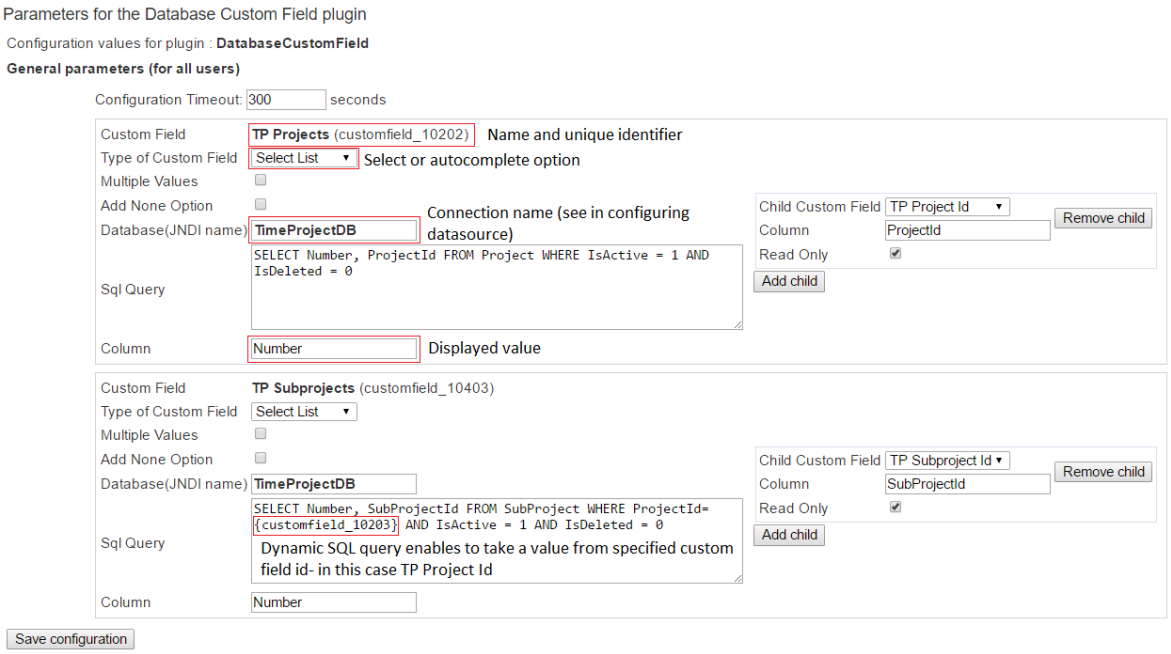


Figure 28: Database custom fields configuration

JNDI datasource configuration

This configuration requires to have a proper driver in JIRA_HOME/lib directory and editing a context.xml file in JIRA_HOME/conf directory. The resulting part of context.xml file looks like this:

```
<Resource name="TimeProjectDB"
      auth="Container" type="javax.sql.DataSource"
      username="xxx"
      password="xxx"
      driverClassName="net.sourceforge.jtds.jdbc.Driver"
      url="jdbc:jtds:sqlserver://IP address:Port/databasename"
 />
```

5.6.2 TP server to Jira server

Basic authentication

As it was requested, the first version of connection to Jira has been made by using a basic authentication. After a research of existing solutions which can be used for interacting with Atlassian Jira in .NET, the Atlassian.NET SDK[39] has been found. This SDK can be easily used for creating a REST client and then requesting required data. The code below represents a method used to fulfill "Jira autocomplete inside daily report item" requirement from Analysis chapter.

```
public List<GetJiraProjectViewModel> GetJiraProjectByKey(string name)
        {
            // Get credentials from ExternalInterface table
            var credentials = _externalInterfaceService.GetCredentials();
            var jira = Atlassian.Jira.Jira.CreateRestClient(credentials.Host,
             credentials.UserName, credentials.PasswordHash);
            jira.MaxIssuesPerRequest = 500;
            //Get request for all Issues
            var issues = jira.Issues.Queryable.Where(x => x.Project ==
             "KOELNTP").OrderByDescending(o => o.Created);
            List<Issue> list = new List<Issue>();
            foreach (var item in issues)
            {
                if (item.Key.Value.Contains(name))
                {
                    list.Add(item);
                }
            }
            return list.Select(GetJiraProjectViewModel.ToModel).ToList();
            }
```

In accordance with the law of cyber security, the password is encrypted in an appropriate way. The thesis, however, do not describe this part from obvious reasons.

[39] Software Development Kit

OAuth

In this section the thesis introduces the very first version of second connection to Jira. This type of connection requires more preparation on both Jira server and TP server side before the actual implementation happens. The preparation consists of following steps:

1. RSA[40] keypair generation
2. Jira application link creation

A Unix-like command-line interface called Cygwin was used to generate 2048-bit RSA key pair using this command:

```
openssl genrsa -out PrivateKey.key 2048 && openssl rsa -pubout -in PrivateKey.key -out PublicKey.pub
```

It is important to realize that Jira server has first been installed locally for testing purposes. Therefore, all the filled in data used in following configuration process corresponds to it. To be able to create Jira application link it is necessary to have administration rights and after that visit Administration/Add-ons/Application Links. First the URL of application to create a new link must be entered. The rest of the configuration can be seen in figure below.

[40] Rivest-Shamir-Adleman cryptosystem

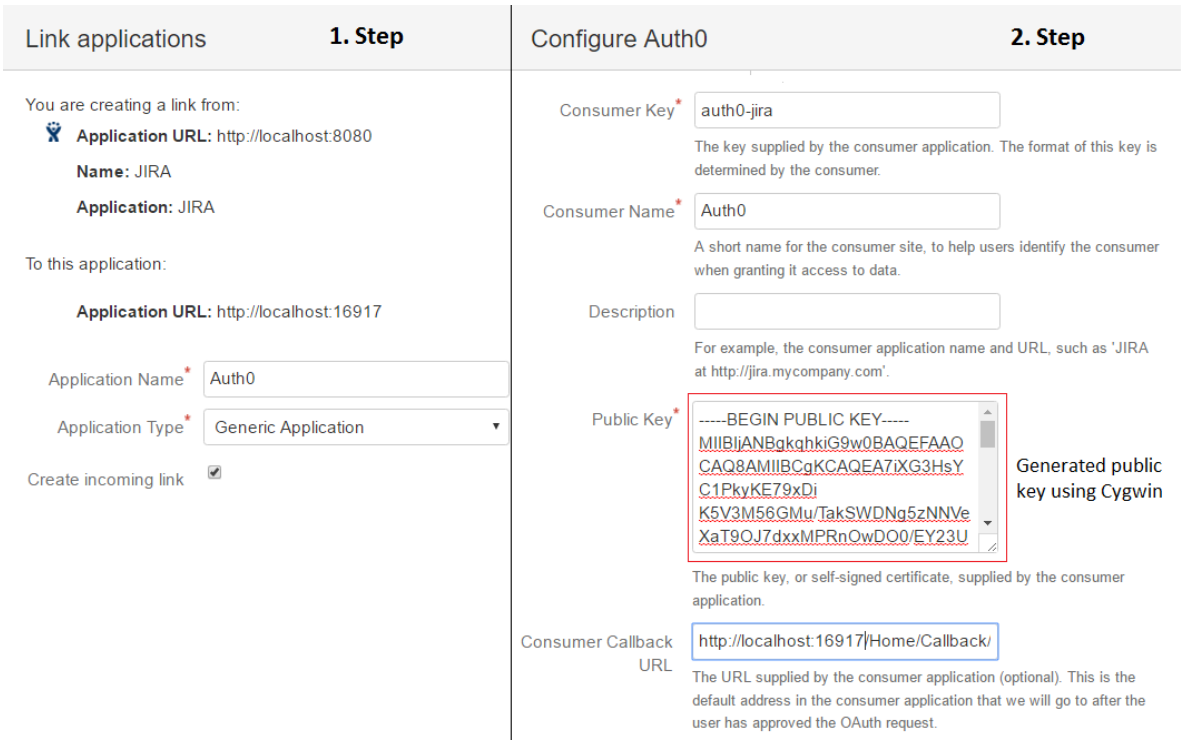


Figure 29: Jira – Application link

At this moment, everything from Jira server side is configured and ready to use. The next step is TP server implementation part. This part of the thesis depicts only the key parts of code together with brief explanation (the rest of the code can be found on attached CD).

```
JiraApplicationCredentials = new ApplicationCredentials
{
ConsumerKey = "auth0-jira",
ConsumerSecret = @"<RSAKeyPair>
                        <Modulus>…</Modulus>
                        <Exponent>…</Exponent>
                        <P>…</P> <Q>…</Q> <DP>… </DP> <DQ>…</DQ>
                        <InverseQ>…</InverseQ> <D>…</D>
                    </RSAKeyPair> ",
};
JiraOAuth1AProvider = new JIRAOAuth1aProvider("http://localhost:8080");
OAuth10AStateManager = new OAuth10aStateManager((k, v) =>

                                Session[k] = v, k => (string)Session[k]);
```

- ConsumerKey – the same value as in Step 2 in Jira configuration
- ConsumerSecret – to get this W3C [XKMS] 2.0 format[41], the private key (PrivateKey.key) needs to be first converted into a valid JSON string and then convert it from pem to W3C [XKMS] 2.0 format
- JiraOAuthAProvider – application URL from Step 1
- OAuth10aStateManager – temporary save and load Token Secret

These values together with callback URL definition are necessary to gain authorization URI[42]:

```
Var authorizationUri = OAuth1aProcess.GetAuthorizationUri(
JiraOAuth1AProvider, JiraApplicationCredentials, callback,
OAuth10AStateManager);
```

The last step is to request Jira API for specific data. For instance, Get issue with issueId “DFO-28”.

```
public string GetIssue()
{
       // Credentials
       var provider = JiraOAuth1AProvider;
       var jiraCredentials = JiraApplicationCredentials;
       var accessToken = Session["access_token"] as string;
       var accessTokenSecret = Session["accessTokenSecret"] as string;
       // IssueId definition, JIRA API URL for Get Issue
       const string issueId = "DFO-28";
       var http = new Http { Url = new
       Uri("http://localhost:8080/rest/api/2/issue/" + issueId) };
            http.ApplyAccessTokenToHeader(provider, jiraCredentials,
       accessToken, accessTokenSecret, "GET");
       var response = http.Get();
       // Parsing returned content- customfields ids can be seen on figure
       Database custom fields configuration
       JObject jObject = JObject.Parse(response.Content);
       string project = jObject["fields"]["customfield_10202"].ToString();
       string subproject = jObject["fields"]["customfield_10403"]
       .ToString();
       return project;
}
```

[41] XML Key Management Specification
[42] Uniform Resource Identifier

Chapter 6

Testing, Deployment and Proposals

6.1 Testing and Deployment

Initial testing of each functionality and behavior within implemented subcomponent has been executed by developer himself. Further sequential testing has been accomplished by TP project manager together with testers from parental branch situated in Prague. The test results have been than reflected into previously mentioned system called Jira and assigned to the responsible developer.

Deployment of TP 5 itself is estimated to be approximately the middle of the year 2017. However, since TP 5 follows agile software development, the system is continuously deployed to an internal test environment every time a cycle ends. To be able to do that the powerful continuous integration tool called TeamCity has been used.

6.2 Proposals for further extensibility

This paragraph briefly introduces some proposals for additional functionalities regarding implemented subcomponents. Firstly, since the OAuth implementation has introduced the first version of this authentication system, it is obvious that it is necessary to reimplement existing solution in following terms:

- All information related to authorization such as Consumer Key, Consumer Secret and so on should be stored in TP database
- Code refactoring in general
- GetIssue method should provide string parameter (the same one as in Basic Authentication – GetJiraProjectByKey method) for autocomplete field in daily reports

Secondly, as far as notification improvements are concerned, the following issues are proposed:

- Instead of requesting server every 30 seconds to get new notification count, use signalR
- Possibility to switch on and off each notification template directly in the system settings grid
- Multi select function – the possibility to delete several notifications at once
- Identify unread notifications by bold font style and read by normal font style instead of using detail view
- Change pencil icon to eye icon for mark messages as read or unread
- Possibility to share a notification by email (envelope icon)

Lastly, in the external connectivity – add other options of connections:

- Web service
- File (containing value and text for combo box/autocomplete)
- DB table from different Databases
- And many others…

Chapter 7

Conclusion

The thesis objectives were to analyze, design and implement subcomponents of TP 5. The reason behind subcomponents design was to extend currently developed solution to cover even more potential customers from different industrial fields.

Within the Theory chapter, a brief description of used technologies such as Entity Framework, ASP.NET WEB API 2 and so forth were introduced. The Analysis chapter shed light on TP core features together with initial description of mentioned subcomponents. Additionally, this chapter presents gathered requirements based on not only customer's expectations but also the needs for an application whether from customer or Baader computer point of view. The following Design chapter reflects these requirements into cohesive blocks consisting of data model, server and client side application architecture, subcomponent design and finally the application design. The last thesis objective was fulfilled via Implementation chapter which essentially follows the same structure as previous chapter. The last chapter shortly depicts how the testing and deployment of TP was made.

By the time the thesis is written, the first version of TP was successfully deployed. According to customer feedback, this solution met with success in all aspects. Thus, it can be presumed that implemented solution suits all defined requirements. However, despite of the solution success, the thesis introduces the very last chapter called Proposals for further extensibility, which according to thesis Author and key members of development team, should be implemented in the next TP version.

On the basis of previous information it can be claimed that all thesis objectives are considered as successfully accomplished.

Bibliography

EntityFrameworkTutorial, 2016. *Entity Framework Tutorial.* [Online]
Available at: http://www.entityframeworktutorial.net/EntityFramework5/entity-framework5-introduction.aspx
[Accessed 22 12 2016].

Anderson, R., Pasic, A. & Dykstra , T., 2012. *ASP.NET Web API.* [Online]
Available at: https://docs.microsoft.com/en-us/aspnet/web-api/index
[Accessed 2 1 2017].

Atwood, M. et al., 2007. *OAuth Core 1.0.* [Online]
Available at: https://oauth.net/core/1.0/
[Accessed 24 3 2017].

Burback, R. L., 1998. *The Design Phase.* [Online]
Available at: http://infolab.stanford.edu/~burback/watersluice/node11.html
[Accessed 20 1 2017].

Daux, P., Gallus, J. & Speelpenning, J., 2002. *Data Modeling and Relational,* s.l.: Christine Markusic.

Dee, J., 2014. *WCF or ASP.NET Web API: Which one to choose?.* [Online]
Available at: http://blog.andolasoft.com/2014/10/wcf-or-asp-net-web-api-which-one-to-choose.html
[Accessed 19 1 2017].

Evans, E., 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* 1st ed. s.l.:Addison Wesley.

Fowler, M. et al., 2002. *Patterns of Enterprise Application Architecture.* s.l.:Addison Wesley.

Franks, J. et al., 1999. *HTTP Authentication: Basic and Digest Access Authentication.* [Online]
Available at: https://tools.ietf.org/html/rfc2617
[Accessed 24 3 2017].

Google, 2016. *Angular- Architecture overview.* [Online]
Available at: https://angular.io/docs/ts/latest/guide/architecture.html
[Accessed 26 1 2017].

Google, 2016. *ANGULAR DOCS.* [Online]
Available at: https://angular.io/docs/ts/latest/
[Accessed 5 12 2016].

Jamsheer, K., 2017. *12 Best Software Development Methodologies with Pros and Cons.* [Online]
Available at: http://acodez.in/12-best-software-development-methodologies-pros-cons/
[Accessed 4 3 2017].

KG, K. D. S. G. &. C., 2008. *Time project.* [Online]
Available at: http://www.time-project.de/funktionen.html
[Accessed 14 12 2016].

Kirchhoff Datensysteme Software , 2014. *Kirchhoff Datensysteme Software.* [Online]
Available at: http://software.kds-kg.de/daten-und-fakten.html
[Accessed 14 12 2016].

Kumar, S., 2015. *The Anaemic Domain Model is no Anti-Pattern, it's a SOLID design.* [Online]
Available at: https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/
[Accessed 8 1 2017].

Martin, R. C., 2000. *ObjectMentor.* [Online]
Available at: http://mil-oss.org/resources/objectmentor_design-principles-and-design-patterns.pdf
[Accessed 8 1 2017].

Martin, R. C. & Martin, M., 2006. *Agile principles, Patterns, and Practices in C#.* s.l.:Prentice Hall.

McLaughlin, M., 2013. *What Is Agile Methodology?.* [Online]
Available at: https://www.versionone.com/agile-101/agile-methodologies/
[Accessed 4 3 2017].

Microsoft, 2016. *Introduction to Entity Framework.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx
[Accessed 22 12 2016].

Millett, S., 2010. *Professional ASP.NET Design Patterns.* 1st ed. Indianapolis, IN 46256: Wiley Publishing, Inc..

Samolysov, P., 2016. *Why the Anemic Domain Model pattern?.* [Online]
Available at: http://psamolysov.blogspot.cz/2016/02/why-anemic-domain-model-pattern.html
[Accessed 8 1 2017].

List of figures

List of abbreviations

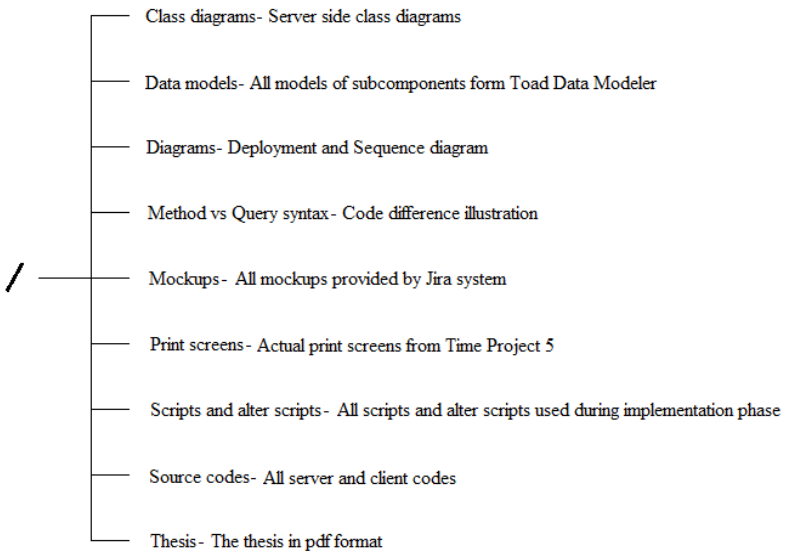
TP	*Time Project*
SAP	*Systems, Applications & Products in Data Processing*
ZEP	*Zeiterfassung für Projekte- web solution for project oriented work*
RDBMS	*Relational Database Management System*
API	*Application Programming Interface*
SDLC	*Software/System Development Life Cycle*
DBMS	*Database Management System*
RDM	*Relational Data Model*
ERM	*Entity-Relationship Model*
SQL	*Structured Query Language*
ORM	*Object-Relational Mapping*
ADO.NET	*ActiveX Data Objects .NET*
LINQ	*Language Integrated Query*
CRUD	*Create, Read, Update, Delete*
WCF	*Windows Communication Foundation*
REST	*Representational State Transfer*
HTTP	*Hypertext Transfer Protocol*
URL	*Uniform Resource Locator*
JSON	*JavaScript Object Notation*
XML	*Extensible Markup Language*
SOAP	*Simple Object Access Protocol*
PHP	*Hypertext Preprocessor*
SoC	*Separation of Concerns*
POCO	*Plain Old CLR Object*
OOP	*Object-oriented programming*
EDM	*Entity Data Model*

DAL	*Data Access Layer*
DDD	*Domain-Driven Design*
IoC	*Inversion of Control*
MVVM	*Model-View-ViewModel*
DOM	*Document Object Model*
UI	*User Interface*
GUI	*Graphical User Interface*
HTTPS	*HTTP over SSL/TLS and HTTP Secure*
EF	*Entity Framework*
CD	*Compact Disk*
MVC	*Model View Controller*
NPM	*Node Package Manager*
CLI	*Command-Line Interface*
ES	*ECMAScript*
DI	*Dependency Injection*
RxJS	*Reactive Extensions library for Java Script*
JNDI	*Java Naming and Directory Interface*
SDK	*Software Development Kit*
RSA	*Rivest-Shamir-Adleman cryptosystem*
XKMS	*XML Key Management Specification*
URI	*Uniform Resource Identifier*

Chapter 8

Appendix

8.1 Content structure of attached CD



8.2 Translation module – Activity diagram

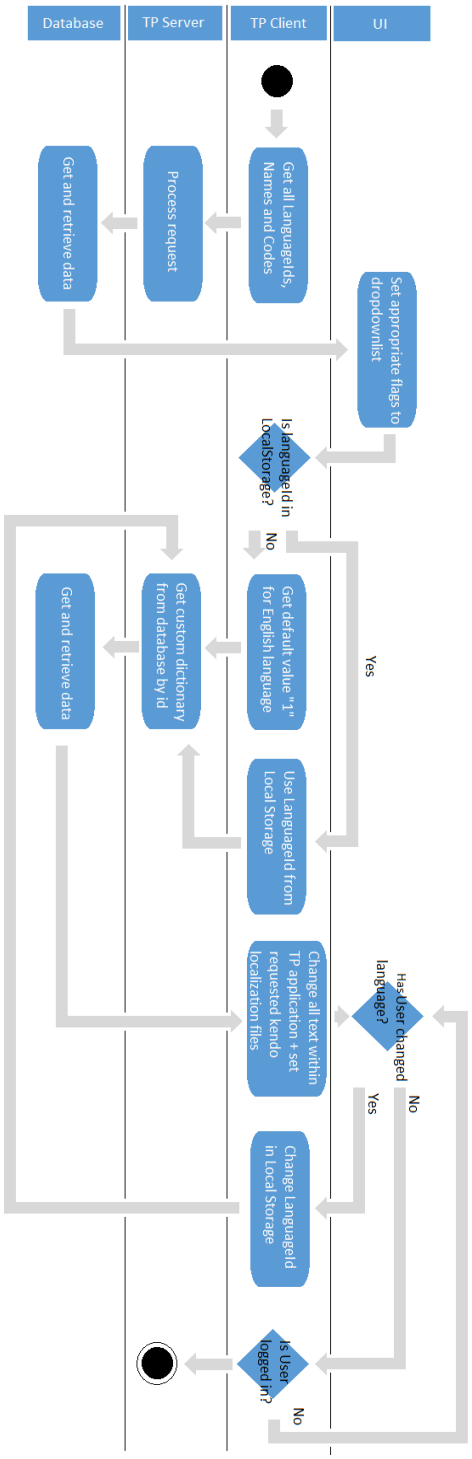


Figure 30: TP localization activity diagram 2

8.2 Web layer class diagram

Figure 31: Web layer class diagram