Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Information Technology, Double Degree

Tomas Bajar, Jan Zahradnik

Database and website implementation for controlling Juki-Lux multimedia pillars

Thesis 2012

Abstract

Tomas Bajar, Jan Zahradnik
Database and website implementation for Juki-Lux multimedia pillar, 33 pages
Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Information Technology, Double Degree

Thesis 2012
Instructors: Lecturer Pasi Juvonen, Saimaa University of Applied Sciences
Managing Director Jouni Könönen.

The bachelor thesis worked on a real task that came from the company Juki-Lux. This company has made a multimedia pillar to present advertisements. The goal was to design and make the necessary software needed to run the background of this project. ASP.NET web pages technology was used to build a central website to control all the pillars and their content. Also a custom database scheme was designed to support all the operations and connect all the software parts together into one functional unit. The ideas and knowledge about these technologies were mostly gathered from our previous experience during studies in home university and from the internet (mostly MSDN, as all the technologies are from Microsoft). In the end the thesis configured the access rights for user accounts, did testing and responded to the feedback from the customer during the development. The system works and met the expectations of the customer. Nevertheless, further development and maintenance is strongly recommended to bring all possible benefits of the proposed ideas.

Keywords: ASP.NET, LinQ, SQL, Database

Table of contents

1 About the project

The next chapters give a detailed description of the project background and where it originated. The project was offered to us by our thesis supervisor among other options. We chose it because we saw a great potential for self-realization and space to develop ourselves further in already known technologies.

1.1 Juki-Lux company overview

The Juki-Lux company is a small enterprise that resides near the city centre of Lappeenranta. The main business of this company is to offer various ways of advertising to its customers. Most of the products provided by the company consist of traditional posters and passive advertising spaces that can be illuminated by use of standard gas tubes from inside custom made frames. The company can also create big banners and eye-catchers usually hung above main entrances of various shops depending completely on the specifications of the customer.

1.2 Pylon Idea

As the product line of Juki-Lux company was certainly nothing special that would stand out in the current competing market, the idea was to design a new product that would offer something new. The idea of interactive advertisement is not new by any means and already exists in many forms throughout everyday life. The corridors of shopping centers are full of colorful ads that attack your eyes every day you walk past them, most of them glowing, or even rotating as a canvas on motorized cylinders inside plexi glass frames. Not often, however, you have the chance to interact with the advertisement yourself, or see fully customizable multimedia content running smoothly in front of your eyes. There are of course solutions that already exist offering such features, but they surely are not spread out widely yet. The second thing is that the prices of such solutions are mostly very high and not affordable for medium to small businesses. Exactly for this described hole in the market the idea about media pillar came to minds of the people from the Juki-Lux company.

The media pillar should be a device capable of showing slide-show presentations of pictures, play video files, or both in combination depending on the current need of situation, the time of a day, or simply the wish of the customer. The idea even exceeds the boundaries of passive presentation. The plan is to develop such a system that could be controlled by touching the onscreen controls on the pillar to interact with it. This would give anyone passing by an opportunity to experience something different than what one is commonly used to. In addition to such features, the pillar should be priced in a way to be available for smaller companies as well.

1.3 Technological aspects

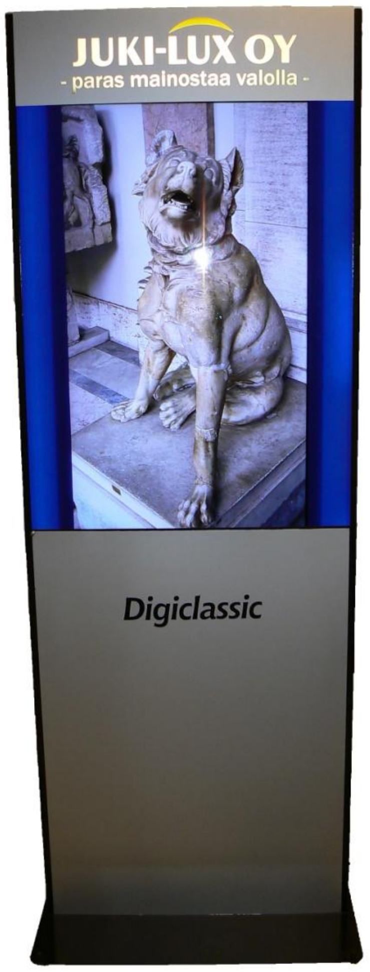


Figure 1 - Multimedia pillar protoype

The pillar itself is a custom made welded metal frame on a stand as a main skeleton. This frame supports all the components that are placed inside it in order to secure them well against outside forces that could cause damage to them, for example shaking the pillar, or careless transportation. From the outside, most of the surface consists of clear plexi glass that protects the main front display in the front side as well as other parts from the back side of the pillar. The plexi glass is inserted into the frame from the top as one big piece which prevents anyone from easily removing it deliberately or not. The only thing going outside the pillar is a main cable for connecting to a classical electric power socket. There are USB connectors on the pillar as well, for the technician (or customer) to upload the desired presentation directly. These will be of course secured behind a cover on the top, or behind a small panel with a lock so nobody unauthorized could access them.

1.3.1 Hardware

The insides of the pillar are mostly a normal personal computer. In order to minimize the needed maintenance and assembly of standard computer parts forming a computer like most of us have at home in a big case, an already assembled solution has been chosen. A small EEE computer from HP has been bought. This box of fairly small size contains everything a normal computer would, packing a satisfactory computing power with all needed ports for external devices that would be needed as well as having a WiFi module already prepared inside. This computer is the heart of the pillar that runs the applications needed to manage and present the media content on the main screen. The main screen is just an LCD television placed vertically and connected by a cable to the graphical output of the computer. Both the computer and LCD are powered from a small UPS that protects the technology from sudden power outages.

1.3.2 Connectivity

As mentioned above, the computer already has a WiFi module inside it, so the pillar is able to connect itself to any wireless networks in the area. In case the customer has his own private network with the connection to the internet, he can use it to make the pillar online accessible. Other option is provided by a 3G USB modem module that uses mobile wireless network to the internet based on what service is bought from the provider. This is a good choice for places without already existing WiFi networks and enables the pillar to be placed in almost any location, because the 3G connection is well spread across the country of Finland. Another option would be to connect the pillar to the internet by means of a standard UTP LAN cable which is also possible. If the customer wished to have his pillar only for his own personal use without connecting to the internet, he can still manage the presented content through uploading the presentation through the USB stick.

2 Project inception

The project of the media pillar was already in progress when it was encountered for the first time. The company presented to us what has been done so far, along with the presentation of the software side from the team that has been developing the applications for running and managing the content. In the premises of the company a prototype pillar has been manufactured so that the people responsible for the development of the tools could test them directly on the device. There were some minor modifications still going on but the rough and important parts of the pillar were already successfully completed.

2.1 What has been done so far

The first team consisted of two Finnish students from Saimaa University of Applied Sciences. They have been working on the needed applications for the pillar for their diploma thesis for some months already. They have done a client side application that is responsible for showing the media content on the running pillar. It takes a zipped file which includes the pictures and videos along with an XML file with configurations of times and effects and begins its presentation. It basically switches the images indefinitely in desired time intervals, or plays video instead of still picture. The second tool that they developed is a desktop based application for the purpose of creating the zipped file with the presentation. The customer can run it on his computer and add pictures or videos that he would like to run on the pillar. He can also configure all the needed properties for adjusting the times of each picture or visual effects when the pictures are changing. These two applications were already functioning when we became a part of this project.

2.2 What needs to be done

The applications that were created by the first team were crucial for the project and provided the basic functionality needed to start the project. In the big picture however this has only been a beginning. The first team prepared a whole concept of how the solution should work in details. They have made a picture describing the idea from a perspective that is shown below. If the customer wants to control what is happening on the pillar, he needs a centralized online tool to do so. That means that the internet pages need to be created and offer the customer such functionality. There is going to be more of the pillars around, so managing all of them will need to be based on some exact data about them. For this reason a database will surely be required to provide necessary data for the internet pages and the pillars to be configured properly. The applications on the side of pillar as well as the internet pages should be able to read and write to and from the database. Even the company of Juki-Lux needs some control over the customers and what they do with the pillars. This should be incorporated into the internet pages as well to reduce user confusion. If the company plans to expand their business also into other countries, more language versions will be needed online. If the pillar should be interactive and react to touch input from anyone, tools for doing so will have to be created or bought and adjusted for the use on the device.

The security side of the whole solution should be taken into account as well. In order to provide such a non-trivial functionality there will be need to merge many technologies together which always presents many security holes that could easily be exploited. The connection to the database, for example, needs to be secure. The same applies to the pillar. If the pillar is connected through the means of WiFi, the data transmission will have to be secured and encrypted. Other security problems arise on the internet pages and present a whole area that needs to be taken care of. If the pillar accepts inputs from the user from the touch screen, its scope will have to be restricted tightly to avoid potential risks such as a presentation of inappropriate content onscreen or making any unauthorized adjustments to the pillar itself.

2.3 Our part of the solution

We had to carefully decide which part of this big concept will be our part to work on as there are many difficult issues in each of them. The work was also meant to be divided between two teams, because of four persons that should work on it. Because we had some experience from our past studies with technologies concerning the development of the internet pages, we chose to make these. It was clear to us that in order to realize the desired functionality we would also need a database to store the data in, so we took that one as well. In the end this turned out to be quite enough to work on. The internet pages are the central part of the solution that connects almost all other parts together and they serve the customer as well as the company. Taking care of all the things around this area proved to be challenging, but we achieved to satisfy the demands set in the beginning. The chapters below give detailed information on how we solved each part of our work.

2.4 Team work

In the beginning we did not divide the responsibilities and work tasks between the two of us. We started our work by designing the database and from that point we were working together on things that simply needed to be done next. When somebody knew how to do it, or had some idea about the issue, he took the task and tried to solve the problem. If one did not know how, he asked the other person for help of course and we found a way how to solve the situation together. In the end the work and responsibilities have been divided between us like this:

<u>Tomas Bajar:</u>

- The first design of the database scheme + realization in MySQL and testing
- Website designing and coding
  - Custom skin file for components
  - Adding and testing CSS styles for particular pages
  - Designing and implementing the upload and management of presentations to pillars (the main page for the customer)
  - Designing and implementation of content management for administrators to take care of all the content uploaded by all users
  - Adding data handling methods to the manager class
- Testing the website and troubleshooting issues in general
- Author of the Silverlight addition idea (realized by the second team)

<u>Jan Zahradnik:</u>

- Importing the database scheme to Microsoft SQL Server + adding tables for user functionality
- Web designing and coding
  - Security providers – logging in, password recovery and changes
  - Creating, deleting and editing of users
  - Security and access rights for pages
  - Pages for assigning pillars to users and vice versa
  - Adding and testing CSS styles for particular pages
  - Adding data handling methods to the manager class
  - Master page design and menu management
- Testing the website and troubleshooting issues in general

3 The idea

In the chapters below, we will describe the aspects of our core ideas into more detail. We will argument the use of each technology and give explanation and examples of use. Basically all the technologies belong to Microsoft which made it easier for us to connect them together. Many parts of our solution were directly designed to work together and offered already prepared parts for customization. However, some of the parts needed to be done from scratch.

3.1 Centralized web-pages

As described in the chapters above, the internet pages present the centre of control over the whole system and were designed to offer functionality both to the user and to the administrators of Juki-Lux company. It is important to mention, that the site does not serve to attract customers and serve broad public. Therefore the visual design was kept simple to offer clarity and easy orientation in the system. We manually customized the appearance of each control so that operations within the site could be as intuitive as possible.

3.1.1 ASP.NET Technology

ASP.NET is part of .NET Framework and when coding ASP.NET applications we can use classes in the .NET Framework (1). There are few languages that are compatible with the common language runtime (2). Microsoft Visual Basic, C++ and C# are languages that enable us develop ASP.NET applications and take advantage of this common language runtime. This common language runtime makes execution of the code we write. ASP.NET code is a compiled into CLR code that runs on the server. We chose this technology because we are familiar with it and because we have seen many real applications and sites based on it. For example, even companies from the bank sector use this technology for their internet banking online systems, which proves both its unmatched functionality and security along with opportunity for a fully customizable visual environment.

Here are some of the main advantages:

- ASP.NET makes development simpler and easier to maintain with server-side programming model.
- drastically reduces the amount of code required to build large applications
- The source code is compiled the first time the page is requested. The server saves the compiled version of the page for use next time the page is requested.
- The web server continuously monitors the pages, components and applications running on it. In case that he notices memory leaks or other illegal software activities, it seamlessly kills those activities and restarts itself.
- The source code is executed on the server. This gives the pages a great potential for powerful functionality and flexibility.
- The HTML produced by the ASP.NET page is sent back to the browser. The application source code you write is not sent and is not easily stolen.
- There is no need to register components because the configuration information is built-in.
- The source codes and HTML codes are together (pages are easy to maintain).

3.1.2 Reasons why we use it

The main reason why we used ASP.NET was that many of the things and components were already done. The majority of us had some experience with this technology and with C# programming language as well. Using ASP.NET, C# programming language and the capabilities of Visual Studio environment made our work easier and more comfortable. Many data handling components for effortless presentation of content were already prepared and needed just a slight customization. Note for example the GridView component, which we used the most. By means of Masterpage site (3) we were able to set uniform appearance of all our pages. The configuration and various settings include connection string, so we did not have to write it repeatedly in the code. This information is centrally stored for the whole application in web.config file. Css files and skin files for setting component appearance are stored in a special folder.

3.1.3 Users and roles

Working with users in ASP.NET is quite easy and we utilized the technology that is already included in framework 2.0. These components allowed us to use the methods which are included in so-called “boxes” (Membership, Roles and Profiles). Because we are using the provider model we can use the functions belonging to Membership, Roles and Profile in our application. By using this technology we do not have to care whether the information about users and roles is stored in a text file or in a database or any other place. Any given provider (4) can take care of these things but we have to manually configure him. We can change the provider for an alternative one without changing the code of our application. On top of that if we are writing an application for the database which is done and contains information about users, we can make our own provider without significant problems. By creating such kind of provider it will be possible to verify users within that database.

Types of providers:

- **Membership provider** is a module that includes the functions for working with users - creating, deleting, changing password or verifying password for instance.
- **Roles provider** includes functions for creating, adjusting and deleting of roles as well as assigning users to these roles.
- **Profiles provider** enables remembering of additional information that is defined for example by user, such as name, surname, phone, address and so on.

Individual modules include methods by which we made operation with the tables from the database. These operations were related to user accounts. The provider object model in ASP.NET provides us complete universality and the wide options of use. For example, we can store serializable types within profiles and information about more independent applications within one structure of table. There is one fundamental disadvantage in this type of approach that is related to working with users in ASP.NET. It is quite complicated to include already defined tables to our own application because of complicated database structure. For this reason we decided to use the Altairis library (5) with simple ASP.NET SQL Providers. This set of providers was made by Michal Valášek just for this reason. By using these providers, the integration to our own application will be much easier.

The Altairis library includes 3 providers:

1. Altairis.Web.Providers.SimpleSqlMemebrshipProvider - membership provider for administration users
2. Altairis.Web.Providers.SimpleSqlProfileProvider - profile provider for storing additional information about users
3. Altairis.Web.Providers.SimpleSqlRoleProvider - role provider for administration of roles and assigning users to roles

3.1.4 User Profiles

Individual providers that we used in our application are not so universal. They do not support anonymous profiles, they include only some types of properties within profile and also they do not allow more independent applications within the database. Their main advantage is simple integration with the rest of the database structure of arbitrary application. The Tables **User**, **UserInRoles** and **Roles** have a specific structure and there is no possibility to change their content. In the table **User** we are storing basic information about new users. We have set the identification number *id* for each user of system. Each user of the system has to have an assigned role from table **Roles**. In order to determine which role belongs to a concrete user we add information to the table **UserInRoles**. The table **UserProfile** allows us to store additional information about user that is not possible to store in the table **User**. The cardinality between these two tables is 1:1 which means that each user has his own profile.

3.2 LinQ – Language integrated query

When developing the solution, we found ourselves in a situation, where we had the internet pages on one side and the database on the other. We had to connect these two together in order to bring the demanded functionality. We could have written the code manually for every access to the database and also make custom objects for mapping the data, but we chose not to. We used the LinQ (6) technology to take care of these things for us and learned how to use it to our advantage which will be described in the sub-chapters below.

3.2.1 Description

For us, LinQ is a framework for object relational mapping that allows us to easily model our relational database by using classes from standard .NET framework. We were using the LinQ to SQL (7) functions on our database which served as a standalone layer between our server application and the database. We were passing queries and parameters to this layer and it accessed the database for us and returned us classes that represented the desired database objects and their relations. The LinQ layer also transforms the input queries to the language compatible with the database, which is a great asset. This meant for us, that we were writing the queries mostly in Lambda expressions and LinQ transformed them into SQL for us.

<u>This is how the mapping was done:</u>

- Table – Object
- Column – Property

<u>Advantages:</u>

- Queries can be dynamic
- Tables are automatically created into classes
- Relationships are handled in classes as well
- Columns are automatically presented as properties
- Lambda expressions are easy to use
- Data is easy to setup and use
- It is clean and type safe

<u>Disadvantages:</u>

- Small data sets will take longer to build the query than execute
- When queries are moved from SQL to application side join commands are very slow

3.2.2 Lambda expressions

Alongside LINQ we used Lambda Expressions (8) that are commonly associated with LINQ. Lambda expressions are typical for LINQ as they enable the user to put parameters and commands into places that would normally need methods or functions with delegates. These lambda expressions return a single output even though they can have more inputs. Typical methods where we used such expressions are Where() and Select().

In the example below the method will return single output by calling Single() method:

```
public User GetUser(int UserId)
{
       using (var dt = CreateDataClasses())
       {
         return dt.Users.Single(h => h.UserId == UserId);
       }
   }
```

Figure 2 - Lambda Expressions

3.2.3 Use

Working with LinQ was very intuitive for us because of using the Visual Studio 2010 IDE which supports operations with LinQ even in the form of graphical designer. When we wanted to generate the core source file that contained all the automatically generated data classes, all that was needed to do was to drag and drop the desired tables onto the designer field and save the project. The designer also showed us the relational scheme of our database to ensure us of its correctness. In the figure below the look of this interface is shown.

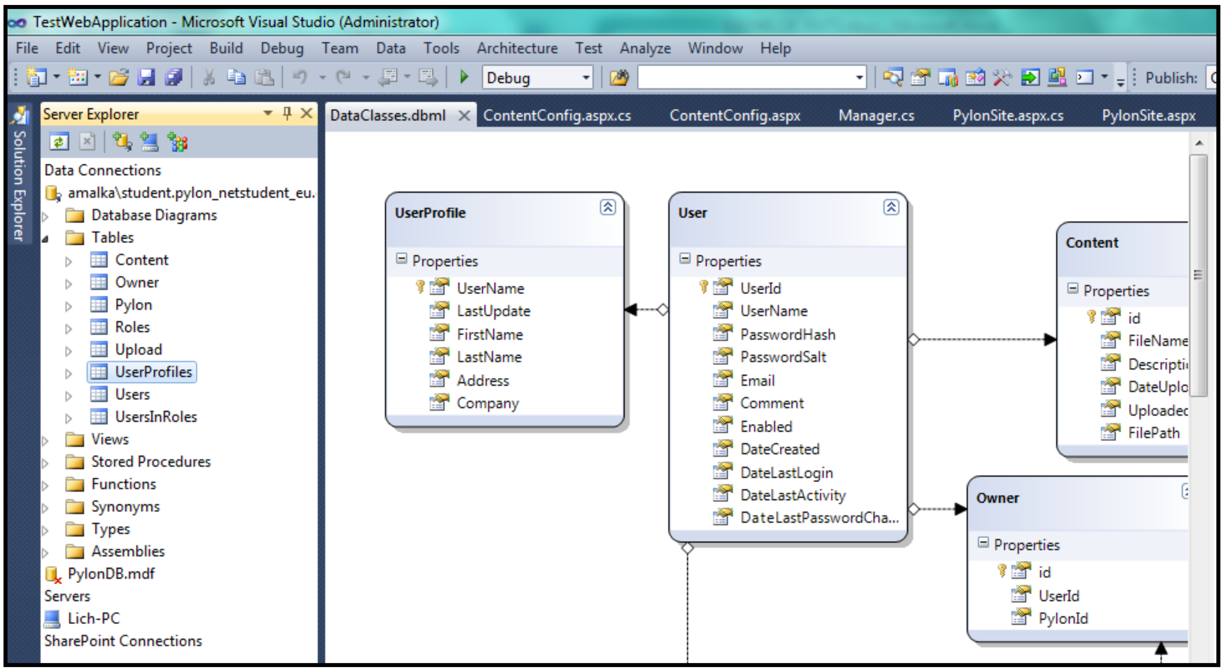


Figure 3 - LinQ graphical environment

3.3 Database

As we mentioned before, a place to safely store data on which all the pages will be based on is needed. We had to choose the way how to realize the database and tables inside it. At first, our selection was the freely available MySQL (9) technology that we downloaded and tested locally on our computers. We designed the first database schema with a few tables and tested its functionality in the MySQL Workbench tool that came as a part of the installation. Later, when we managed to find the real online hosting, we had to change the technology to Microsoft SQL database. This did not pose any problems for us because the core of this technology is still the SQL language. However, we had to download the new tools necessary to work with it and learn how to use them, which took a short time. In the end everything worked out well and our latest version successfully runs on the Microsoft SQL server located on the domain mssql.amccomp.cz\student. Still, it is important to mention, that our solution is not dependant solely on working with Microsoft SQL server because of using the LinQ technology. If the provider of the database will change and a different SQL database technology will be used, adjustments to the system will take only a short time which is an important advantage of using the LinQ.

3.3.1 Reason for DB

Why do we need the database or what benefits do we get out of using it? When we saw the scope of our proposed solution, we understood that a place to store all the data safely will be an integral part of the big system. The ASP.NET pages work with the database the most. That is because they take care of almost all the functionality presented to the users and thus need the data as input for their methods. Almost every operation done on the pages interacts with the database in some way. The client application that runs on the pillar itself needs the data as well. The listener application that was done by our second team connects to the database remotely and reads the necessary data for downloading the presentations that it should run along with data about the timing of these presentations. The benefits of using centralized data storage are many. We can separate the web server with the pages from the data server running the database which promotes security in case of an attack or failure on the pages. It is also possible to change the hosting of the pages without affecting the database. The data in the database can also be backed up to prevent their loss in case of disasters etc., which is strongly recommended when dealing with business applications.

3.3.2 Database scheme

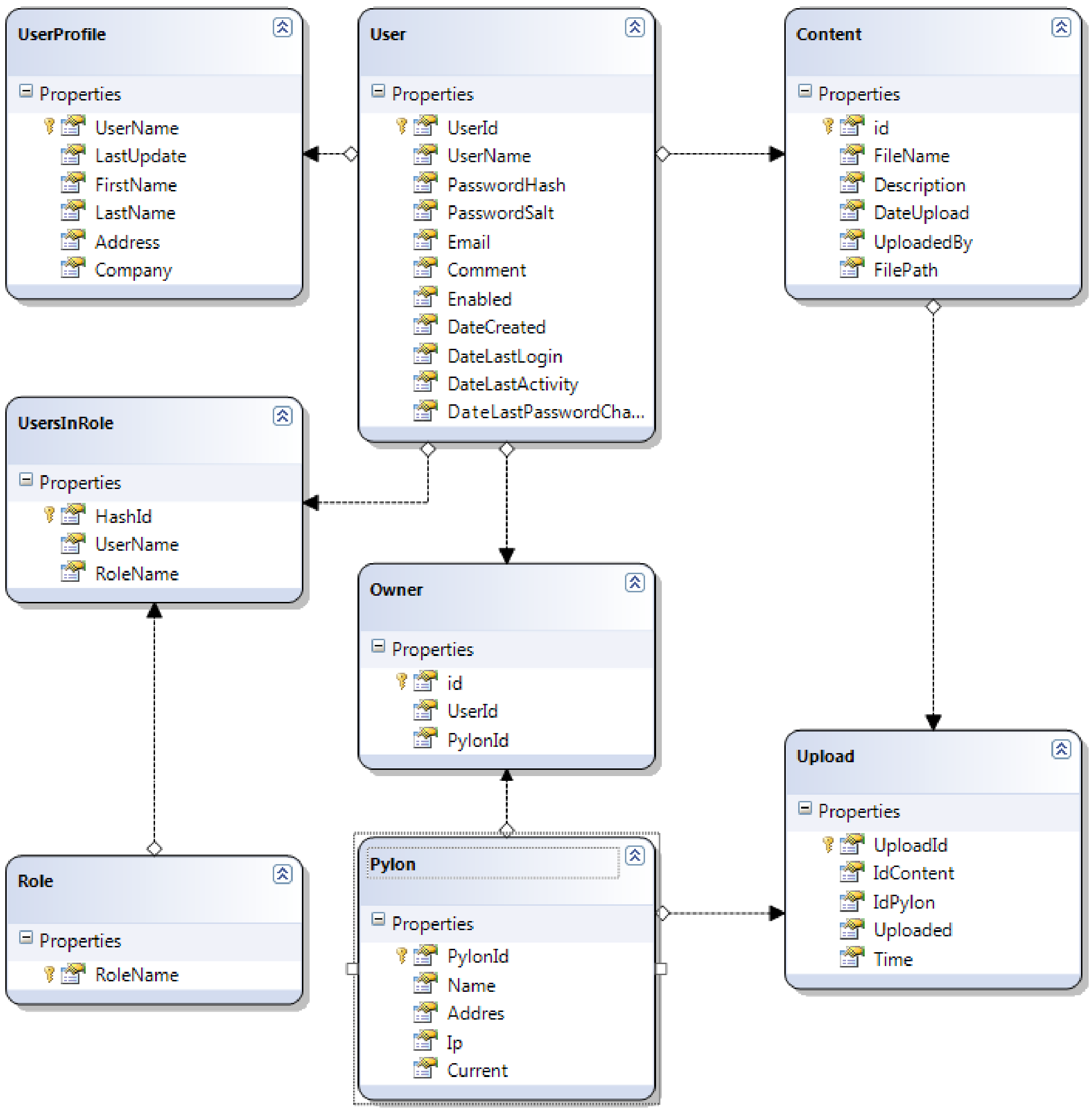


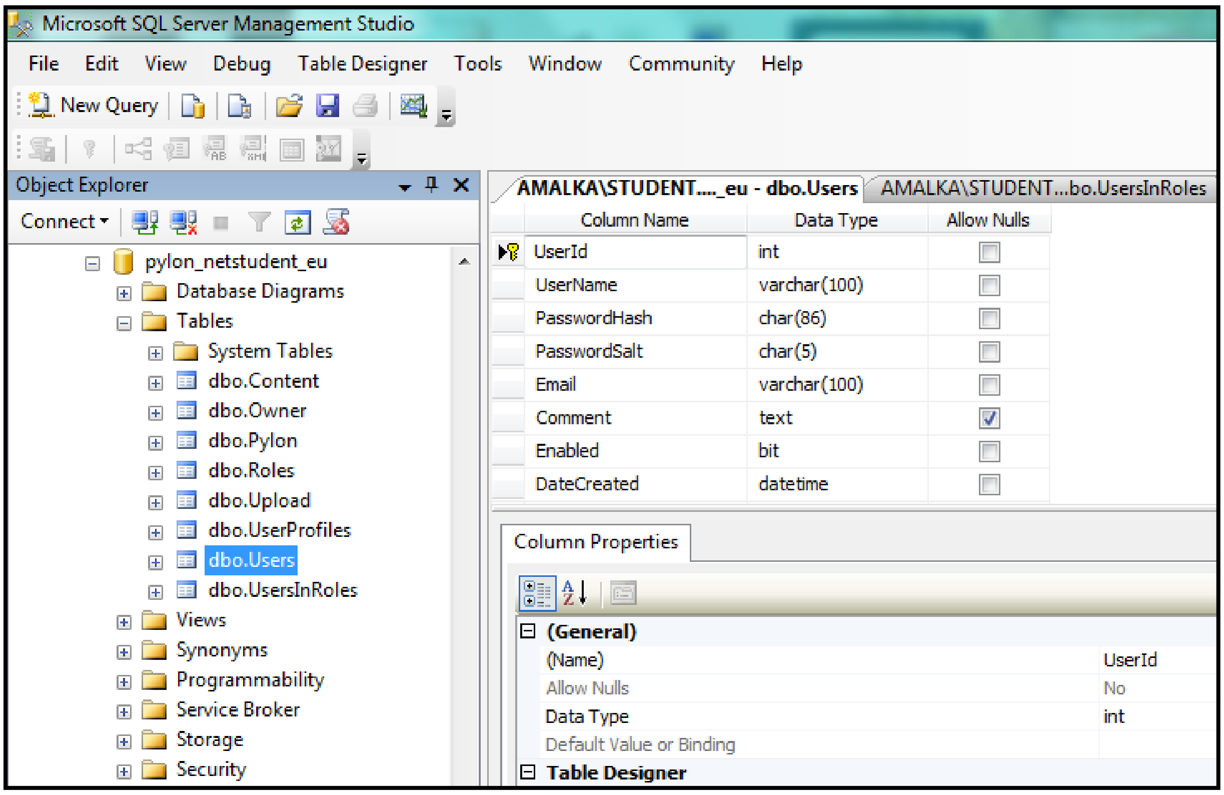
Figure 4 - Database scheme

Our suggested database scheme consists altogether of eight tables. There are four tables that serve us for working with user accounts.

We can store a lot of information in the table **User**. Most of this information is necessary to be in the database. In addition to this basic information we needed to store additional information related to the user and there is no possibility to store them in the **User** table because of a complicated database structure and the way provider components work. We created the table **UserProfile** that contains properties *Firstname*, *Lastname*, *Address* and *Company*. Having these properties we can store additional information such as address of user and company which he belongs to. Naturally, we are able to add other information in case of need. Every user has his own profile that is assigned to him by cardinality 1:1 between these tables. The tables are connected by means of the *UserName* property. This means that every user that is in the database has to be matched up with some role. There are two roles **admin** and **user** that we created in the database. The configuration permits only one role per individual user. Users that belong to the admin role can perform tasks that other users are prohibited from doing. These two roles are represented by table **Roles**. By creating a new user, a new record is stored to the table **UserInRoles**. That is the binding table where we keep the list of users and roles they belong to. There is cardinality between these tables is 1:N. The foreign keys in that table are *UserName* and *RoleName*. All of these tables have exclusive use only for operations related to users, their accounts and roles assigned to them. Other tables in the database scheme are **Owner**, **Pylon**, **Upload** and **Content**. There is cardinality M:N between tables **User** and **Pylon** which means that one user can have access to more pillars and one pillar can belong to more users. That is why we made the binding table **Owner**. The properties in this table *UserId* and *PylonId* are foreign keys and simultaneously primary keys in tables **User** and **Pylon**. The last part of the database scheme is composed by tables **Upload** and **Content**. In the table **Content** we stored name of the file in property *FileName*. In the property *UploadedBy* who was that content uploaded by. Some description related to files can be stored in the property *Description*. There is also one very important property called *FilePath*. In this property we stored the path where each file on the server is stored. Later on, the client side of application will utilize this property. Another binding table is needed between tables **Pylon** and **Content**. Table **Upload** saves information about the content that will be uploaded on the pylon.

3.3.3 Tools

Working with the database demanded a set of tools that would make all the important adjustments possible. For the first version that we tried with the MySQL, the MySQL Workbench worked fine and offered an all in one environment to take care of everything easily. When we switched to use the Microsoft SQL server, we tried to work with the database through our Visual Studio 2010 IDE. We could carry out the basic operations, but often we found that the offered functions are not enough for us. We downloaded the SQL Server Management Studio from Microsoft and started to use it. There we were finally able to perform all modifications to the database that we needed. While working on the development of ASP.NET pages in Visual Studio, we worked with the access to the database layer simply by dragging and dropping the tables as pictures in designer view which proved both easy and intuitive. In the figure below you can see how the SQL Server Management Studio looks. Being able to see clearly and easily all the needed parameters was of big help to us and greatly shortened the time needed for the maintenance of the database.


Microsoft SQL Server Management Studio
File
Edit
View
Debug
Table Designer
Tools
Window
Community
Help
New Query
Object Explorer
Connect
AMALKA\STUDENT...._eu - dbo.Users
AMALKA\STUDENT...bo.UsersInRoles
Column Properties

pylon_netstudent_eu
- Database Diagrams
- Tables
  - System Tables
  - dbo.Content
  - dbo.Owner
  - dbo.Pylon
  - dbo.Roles
  - dbo.Upload
  - dbo.UserProfiles
  - dbo.Users
  - dbo.UsersInRoles
- Views
- Synonyms
- Programmability
- Service Broker
- Storage
- Security

Column Name	Data Type	Allow Nulls
UserId	int	☐
UserName	varchar(100)	☐
PasswordHash	char(86)	☐
PasswordSalt	char(5)	☐
Email	varchar(100)	☐
Comment	text	✓
Enabled	bit	☐
DateCreated	datetime	☐

(General)	
(Name)	UserId
Allow Nulls	No
Data Type	int
Default Value or Binding	
Table Designer	

Figure 5 - SQL Server Management Studio

4 Implementation

This chapter tries to put some light on various things that we consider important to mention from the viewpoint of implementation. We will mention significant files that contain essential configuration needed to control the site, location of page files and explain design matters around skin files as well as classic CSS files. Examples of usable user functionality according to given roles will be given. We will also explain what is accessible by normal users of the system and persons with administrator rights, because they differ.

4.1 Web-pages

Significant files and folders:

- **Account** folder contains ASPX files concerning the pages that provide functionality for changing password, logging in, recovering lost passwords etc.
- **Admin** folder contains ASPX files that can be accessed by persons with administrator rights for purposes of managing all online content uploaded by users, creating and editing users, pillars etc.
- **App_themes** folder contains skin and CSS files responsible for customizing the looks and behavior of used components. It also contains pictures for upload button and file icon.
    - **Skin1.skin** is skin file that was used for setting the visual appearance of GridView and DetailsView components.
    - **Style.css** contains many styles used for formatting the positions and looks of components as well as standard HTML tags like <div>.
- **ClientBin** folder is used for the Silverlight part of our site that was developed by the second team.
- **Styles** folder includes **Site.css** file that contains style classes responsible for setting the graphical appearance of the site template.
- **Users** folder contains ASPX files that are accessible by persons with normal user rights (customers).
    - **PylonSite.aspx** file is the main page file for normal users which takes care of uploading new presentations to the pillars, editing and deleting.
- **About.aspx** file is a page visible for everyone that can serve for presenting information about the site or provider company.
- **DataClasses.dbml** file is core file for LinQ technology that contains all the necessary data classes called when dealing with the communication with the database.
- **Manager.cs** file is the most important core file that contains many methods which call LinQ functions and return data from the database. It is used throughout almost every page.

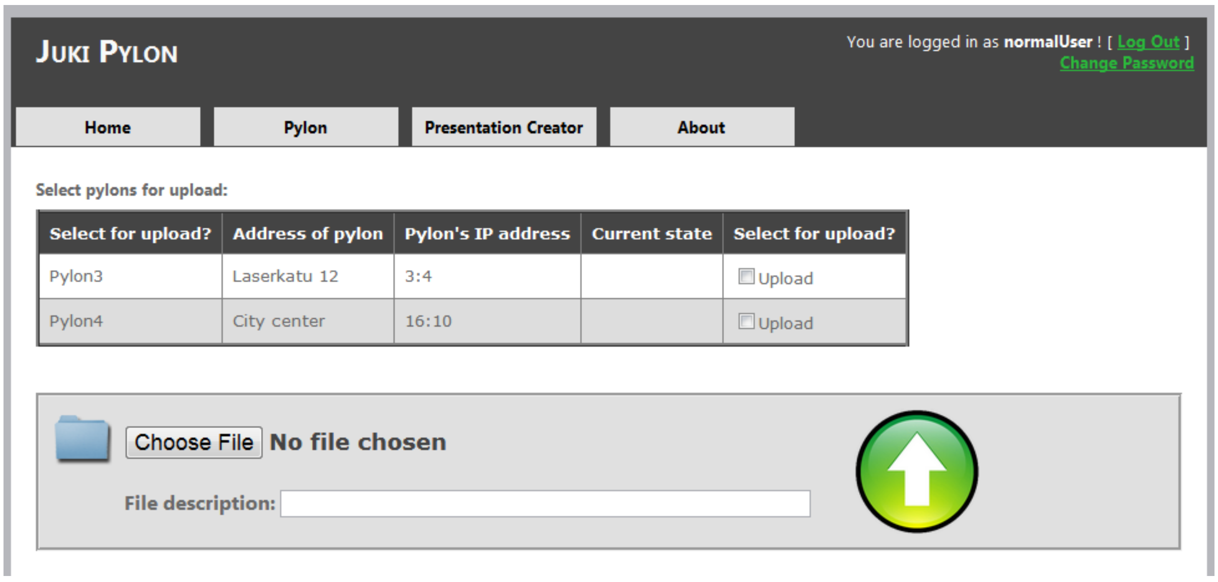
- **PresentationCreator.aspx** file is a page which houses the Silverlight sandbox for presentation creating application built by the second team.
- **Site.master** file contains the layout of our template and is responsible for the appearance of our site. The top menu buttons are defined in this file.
- **Web.config** file contains configuration of parameters concerning the whole site and its behavior. There are more files of this name that inherit properties from the file in root folder and which add or alter the properties of each nested web page.

4.1.1 Access rights for user accounts

We have set access rights in a way that users who belong to role **user** do not have access rights to the configuration section. They can use the functionality only from the section **Pylon** from the top menu. We added three security rules. The first rule allows access for all users who belong to role **user**. The second rule allows access for all users who belong to role **admin** and the last rule denies access for all other users. Users who belong to role **admin** have full access to the pages and can make changes related to various information about other user accounts and also change their roles in the system. Anonymous users do not have access rights and for access to the system logging is needed. Application accepts the first rule that complies with the given user. All rules are defined and saved in web.config section for each given folder. For example the rules for access to admin folder have their own web.config and rules for access to user folder have their own web.config. These configuration files inherit all settings from web.config of application.

4.1.2 Normal user access example

When a customer logs into the system by account with **user** role, he gains access to the **Pylon** button in the top menu. This is where he can upload new presentations to pillars that belong to him. He can also delete the presentations or modify the times of their presenting separately on each pillar. If the user does not have the zip file with an already created presentation, he can use the **Presentation Creator** button from the main menu to access the Silverlight application that enables online creation and management of presentations. These two pages offer all the demanded functionality that the user should have in order to manage all his needs around the pillars. He can also access the **Home** and **About** sections, where general information is to be found. In the figure below you can see the main interface of the site when person with role **user** is logged in.



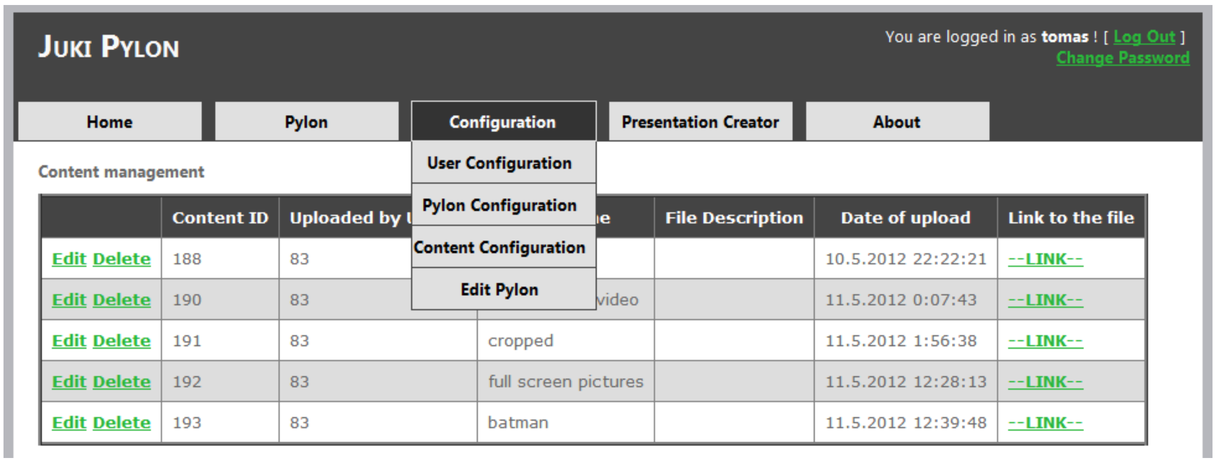
Select pylons for upload:

Select for upload?	Address of pylon	Pylon's IP address	Current state	Select for upload?
Pylon3	Laserkatu 12	3:4		☐ Upload
Pylon4	City center	16:10		☐ Upload

Figure 6 - Normal user page example

4.1.3 Administrator user access example

When a person in the **administrator** role logs in, he gains more access rights than a normal user does. He can access everything like a normal user, but he is able to use the **Configuration** button in the top menu that contains pages for management of users, pillars and content uploaded by all users. On those pages, the administrator is capable of controlling everything that is needed for smooth run of the pillar systems and he can solve problems or perform tasks for the customers. Administrator is also the only one who can add new users and assign pillars to them. In the figure below you can see the site from the view of logged in administrator.



Content management

	Content ID	Uploaded by	e	File Description	Date of upload	Link to the file
Edit Delete	188	83			10.5.2012 22:22:21	--LINK--
Edit Delete	190	83	video		11.5.2012 0:07:43	--LINK--
Edit Delete	191	83	cropped		11.5.2012 1:56:38	--LINK--
Edit Delete	192	83	full screen pictures		11.5.2012 12:28:13	--LINK--
Edit Delete	193	83	batman		11.5.2012 12:39:48	--LINK--

Figure 7 - Administrator page example

4.2 Code behind

This chapter demonstrates some examples of the C# code from the .cs files that were used to create the important parts of the system functionality. We will also describe significant classes from the solution for better understanding how our website works on the server.

4.2.1 Data operations

In the example below you can see how we approached the access for data to the database by using the LinQ classes. The example method returns iterable interface with objects that match the given query, in this case rows from the table **Content**. The **using** construction takes care of closing the data connection to the database for us safely and shortens the code. Inside the block you can see the LinQ SQL query that slightly differs from the standard SQL language. In the end you can see the use of Lambda expression to substitute the needed delegate with one input parameter and thus greatly clarifying the code. Almost all the methods from the file **Manager.cs** are written in this style.

```
public IEnumerable<Content> GetContent(string userName)
        {
            using (var dc = CreateDataClasses())
            {
                User user = (from p in dc.GetTable<User>()
                             where p.UserName == userName
                             select p).SingleOrDefault();
                int userID = user.UserId;
                return dc.Contents.Where<Content>(p => p.UploadedBy ==
userID).ToList<Content>();
            }
        }
```

Figure 8 - Using LinQ classes

4.2.2 Component data binding

In most cases, we used the prepared GridView (10) components for easily understandable interpretation of data in tables. The next example shows how to manually bind the data source to the GridView component. In this case we needed to manually make our own class to represent the table responsible for connecting tables **Content** and **Pylon** which is named **ContentAndPylon** and fill the objects by data in the code. This was because of the way how LinQ represents the nested structures. In the **foreach** section we fill the data and add them one by one to the prepared list that is bound to the GridView component as a data source in the end.

```
protected void uploadBind()
        {
            IEnumerable<Upload> uplo = manager.GetUpload(User.Identity.Name);
            List<ContentAndPylon> CaP = new List<ContentAndPylon>();
            foreach (Upload u in uplo)
            {
                ContentAndPylon newCaP = new ContentAndPylon();
                newCaP.PylonName = u.Pylon.Name;
                newCaP.FileName = u.Content.FileName;
                newCaP.Time = u.Time;
                CaP.Add(newCaP);
            }
            GridViewUpload.DataSource = CaP;
            GridViewUpload.DataBind();
        }
```

Figure 9 - Component data binding example

4.2.3 Used classes

We used some classes in the project that may not seem necessary. Here we will shortly describe their purpose and use.

- **Manager.cs** – This class is created in the beginning of many other classes that use this manager to return data from the database for further processing. This class makes the LinQ technology available to use anywhere in the code.
- **UsersAndRoles.cs** – This class serves the purpose of putting together information about user accounts which are separated between two tables with cardinality 1:1 as will be mentioned in the database chapter below. We created it because we could not modify the table **User** as we wanted.
- **ContentAndPylon.cs** – This class was created to represent easily the joining table **Upload** which realizes the M:N cardinality between tables **User** and **Content**. The reason for this class is the way LinQ layer works. It puts these joining tables as nested data objects into the returning data sets, so we could not access it directly. Instead, we made this class and filled it manually in the code.

4.2.4 Dynamic code in the ASPX files

When dealing with the ASPX files, we needed to dynamically change the properties of some controls within the page. Because the idea of ASPX files is to behave statically, we had to use special code snippets that were executed in the time of compilation of each page. That gave us the ability to dynamically change the displayed controls with every user postback. In the code example below you can see the **Bind** and **Eval** code blocks responsible for this functionality.

```
<asp:TemplateField HeaderText="Content ID">
      <EditItemTemplate>
            <asp:Label ID="ContentIDLabel" runat="server" Text='<%# Eval("id") %>'>
            </asp:Label>
      </EditItemTemplate>
      <ItemTemplate>
            <asp:Label ID="ContentIDLabel" runat="server" Text='<%# Bind("id") %>'>
            </asp:Label>
      </ItemTemplate>
```

Figure 10 - Dynamic code in ASPX file example

4.3 Testing

We were testing every version of our website that we produced. During the day we often made around five new versions of the site. We were testing them first locally through the Visual Studio environment and after successful local runs we uploaded the version to the hosting server and tested the newly added functionality online. Testing the site on a real hosting proved to be very important, because the environment and capabilities differ from those local ones and we discovered different errors and behavior in the online release. It is also important to mention that extensive testing of the whole ASP.NET site that we produced should be done once again after deploying it to different hosting server. Every hosting might have different versions of .NET framework libraries, so the functionality should be carefully checked over.

5 Development method

This chapter will describe the process in which we were constructing our proposed solution. In the beginning we had nothing done that we could start building on. We decided to start from the bottom and make the underlying database level. After having this first building block, we began our work on the internet pages to manage inputs and outputs as well as first features of functionality for the users. Our work on the pages uncovered things that needed to be changed, added, or deleted so from this point the process might resemble an iterative approach (11). In the end, real testing of functionality and presentation on the meetings with the supervisor and the customer revealed the most important issues that needed attention.

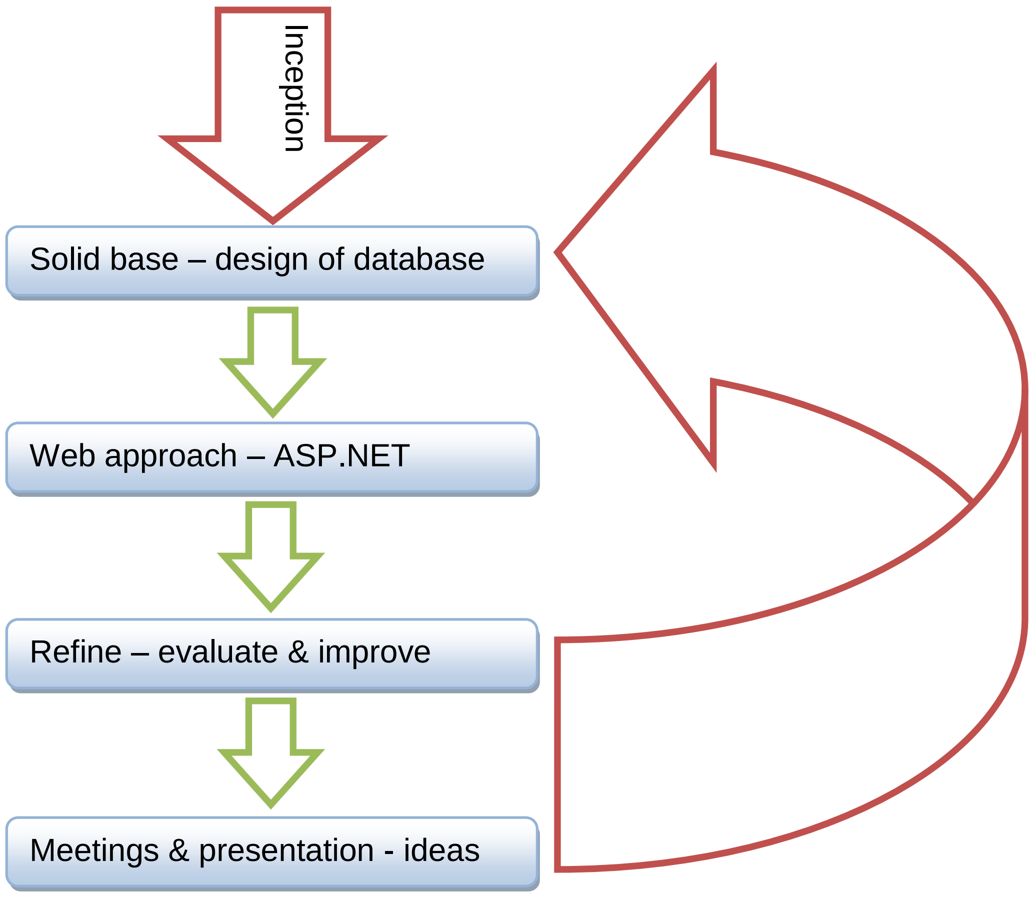

Inception
Solid base – design of database
Web approach – ASP.NET
Refine – evaluate & improve
Meetings & presentation - ideas

Figure 11 - Graphical scheme of our development method

5.1 Solid base

The database structure was the first thing that was designed. At first, only the most important tables were created and connected by relations. We made tables for users, pillars and content along with the needed tables to realize M:N relations between these. After each change made to the schema we also did some testing to ensure that everything works the way we wanted. Mostly the testing was done in the SQL Server Management Studio by creating and executing our own SQL commands to insert and delete data. By this, we tested the constraints and verified the functionality of foreign keys. We also saved the used queries into files in order to use them later which proved very useful. When we were working on the higher levels of our solution we had to return a few times to change something in the scheme of our database which needed to even drop some tables and recreate them. That is where the saved files came in handy.

5.2 Web approach

Having the database to build on, we started to add basic functionality to our pages. At first it was simply handling the inputs and outputs of data. We were designing and coding the components necessary for presenting the data in a readable form, making the methods to retrieve data from database by use of the LinQ technology, administering the user rights of access and also altering the appearance of the whole site through skin files and standard CSS files. Functionality for the users came as first, because we wanted to have something done what could be presented to the customer and the supervisor. After consulting these main features, more complex things were designed and done. We added functions for the administrators of the site to modify data about users, pillars and content so they could fix problems themselves or help the customers. In the latest stages, when the pages were nearing completion, we set up the user access rights that differentiate normal users from administrators and restrict the visible features.

5.3 Refine

While testing the pages on our computers through Visual Studio we often encountered problems that needed to go back to the database and make some changes there, so there can be seen a loop back. At this point we also uploaded the current version to the real server that provided hosting for our pages and tested the solution in real online environment. That proved to be crucial, as many aspects of functionality differed from a locally hosted run. When we found some errors, we had to understand them, replicate them locally if possible and fix them. Every day, many versions of the project were uploaded and tested online to ensure stability.

5.4 Meetings and presentation

During the development we had appointed meetings with the supervisor, the first developing team and the customer. We prepared our current version and presented it on the meetings. The feedback that we received on those meetings was the most valuable source of information. We discussed what features we could improve from their current state, or what new things could be added overall to our solution. In the meantime, we often found some issues on our own and asked how to easily fix the situation. We also cooperated with the first developing team, because we needed some technical details about the pillar, its connection, or the applications made by them. From this point, we often had to redesign parts of our already existing solution in accordance to the demands of both the supervisor and the customer.

6 Evaluation

This chapter will discuss the results of our work. We will describe how much we satisfied the demands that were set in the beginning and if the customer was content with the outcome. Also, we will describe the project from our point of view to tell our opinion regarding the run of the project in general. We will try to pinpoint weaknesses of the systems and acknowledge what could have been done better. In the final sub-chapter we will propose some features that could be added in the future to make the solution better or what could be improved above the current state.

6.1 Meeting the demands

Because of the few meetings that we had with the customer and the thesis supervisor, we had valuable feedback for our development. This feedback made it possible to create the system in a way that was expected. We were heading the right direction from the start, but many essential modifications were made during the development stages too. Both the customer and the supervisor came up with new ideas during the time of development and we realized them successfully. In the end, the customer expressed his satisfaction with the project and admitted that it even exceeded his expectations. The customer also appreciated that the site does not look complicated and does not contain any distractive elements so the work seems intuitive even without reading the user manual. At the final presentation of our work, we did not receive additional demands for adjustments or corrections.

6.2 Personal review

In the beginning of the project we were in a hurry because we got assigned to it very late due to a failed attempt to place us in different position. It helped us very much, that our supervisor was able to set up meetings in the customer's company and showed us the pillar in reality along with the description from the first team that have been working on it for two semesters. We quickly understood what our position is and started to plan what to do. We learned to concentrate on the most important things that are needed to begin working, which will surely be of use in the future.

We came with the idea that has been described in the chapters above which was mostly based on our past experience from developing smaller applications for our studies in home university. We risked a little bit by using technologies that were not so known to us, but in the end this proved to be a good choice. The technologies were not so difficult to work with and even made some things that would otherwise have to be done manually easier for us. The ability to use them together in one project in the future is surely an experience worth that little risk. We also learned that other things around the project are important too and influence the course of the whole project significantly. Without any doubt, working on real demand from a customer is completely different than what we have done so far in ordinary lessons and we gained skills that we would not gain the other way.

6.3 Known bugs or incomplete functionality

One of the biggest problems for us was to make the website look the same in various web browsers. We were mostly testing and developing the site through the Google Chrome web browser, occasionally Mozilla Firefox and the latest Internet Explorer. We tried to make the behavior and visual appearance of the site the same in every one of them, but we were unable to do so completely. For example each browser interprets the CSS styles a little bit in its own way, which can cause a slight difference in the looks of the pages. Also some components like FileUpload were represented differently. One browser showed the text field on the left side with only the file name inside, the second one showed it on the right side with a full path to the file. We do not have much experience with tuning such things, but we have done what we could. We studied this issues a bit and found out that conditional style sheets could be added to work with different web browsers in order to fix the differences (12).

There was some trouble with the security as well. When users want to reset their passwords, they have to write their username to the input field. When this system finds the matching user, it resets his password and sends a notification by email to him. The problem is that anybody can type in any username without checking, so they can possibly reset the passwords for everyone if they guessed their usernames. Fixing this problem is not easy at all, because it has to be done through the security provider components. Luckily, resetting the passwords is all that can be done and it does not give any user the possibility to gain the login information of others, because the notifications will be sent only to the email addresses that have been assigned to the users during creation of their accounts. It is not possible to change this email later.

We were unable to set the time format to any nice looking one in the GridView component fields. We found a way how to do it, but it cannot be applied with our dynamic data binding. There are other ways to do so, but it is a known bug that it actually does not work and we found it even admitted by Microsoft on the pages of MSDN. That is the reason why time is shown almost with the precision of nanoseconds.

A slight functionality gap can be seen in the management of presentations too. When the user selects the pillars he wants to put the presentation on and uploads the zip file, he cannot assign the presentation to more pillars later. He can change on which pillars the presentations will be shown but he cannot alter the number of them. For doing so, he has to upload the file again through the uploading page and select additional pillars.

6.4 Space for future development

Because of the whole idea around the multimedia pillar, it is understandable that there are many features and functions that could be added to fully seize the opportunity of such technology. We worked hard to build the core functionality from the scratch and did not have much time to develop advanced possibilities of use that this idea offers. We could however outline a few ideas that we had in mind while working on this project above fixing the bugs mentioned in the earlier section.

Better time management of the presentations could be created. Now the pillar shows the presentations the same way every day. It would be very useful to have the ability to schedule presentations differently for each day of the week for example, because the needs for advertisement might change during the weekends or other special days.

The site itself could offer more services to the customers in general. It could show new customers coming to the page the pillars, their capabilities, and make it possible for the customer to register themselves. Once registered, the customers could order new pillars by filling in some forms and even pay electronically for them.

Localization of the site to support more languages would also come in handy and offer expansion of the business for Juki-Lux company in the future.

7 Summary and discussion

The thesis dealt with designing the solution for management of multimedia pillars that were designed by the customer Juki-Lux company from Lappeenranta. From what has already been done by other teams before us, we decided what could be our part of work and designed a central we pages to control the pillars. We needed a database to store the necessary data for the operations on the web pages and designed its relational scheme. We chose the LinQ technology as a layer between the database and the web pages to work with because of the above described benefits. We were developing this solution in a small team and shared the current version of the project through our free student hosting server. For the database we found a free limited Microsoft SQL Server that we used for the development and the first deployment of the solution for the means of presentation. During the development of the system we had appointed meetings with both our supervisor and the customer. On the meetings we presented what we had done so far and collected valuable feedback that helped us correct errors in the system and also add new features.

After the last presentation that we had with the customer and the supervisor of our thesis, both expressed satisfaction with our result. The web pages were tested in the presentation to show that they are functional and that they meet the objectives which have been set in the past. The core functionality is done and we managed to add even some features that extended our idea from the beginning of the project.

We learned many things during the development of this project. We worked with technologies that were not familiar to us before and gained valuable experience from it. Combining more prepared frameworks together into one functional whole proved to be a challenging task, but we successfully managed to do so. We also learned the importance of structured and planned approach to work, without which we would not be able to put together such an extensive solution from scratch. Luckily, we had some knowledge of these issues from our past studies, so we did not have to struggle to find out. In the end we surely realized how important the feedback from the customer is for the resulting quality of the project.

References

1. Microsoft ASP.NET, ASP.NET 4 and Visual Studio 2010 Web development overview. http://www.asp.net/whitepapers/aspnet4 Accessed on 17 May 2012

2. Microsoft MSDN, CLR Hosted Environment. http://msdn.microsoft.com/en-us/library/ms131047.aspx Accessed on 17 May 2012

3. Microsoft MSDN, ASP.NET Masterpages. http://msdn.microsoft.com/en-us/library/wtxbf3hh.aspx Accessed on 17 May 2012

4. Microsoft MSDN, ASP.NET Providers 2.0: Introduction. http://msdn.microsoft.com/en-us/library/aa478948.aspx Accessed on 17 May 2012

5. CodePlex, Altairis Web Security Toolkit. http://altairiswebsecurity.codeplex.com/wikipage?title=Simple%20SQL%20Providers&ProjectName=altairiswebsecurity Accessed on 17 May 2012

6. Microsoft MSDN, LINQ (Language-Integrated Query). http://msdn.microsoft.com/en-us/library/bb397926.aspx Accessed on 17 May 2012

7. Microsoft MSDN, LinQ to SQL. http://msdn.microsoft.com/en-us/library/bb386976.aspx Accessed on 17 May 2012

8. Microsoft MSDN, Lambda Expressions (C# Programming Guide). http://msdn.microsoft.com/en-us/library/bb397687.aspx Accessed on 17 May 2012

9. MySQL, Downloads. http://dev.mysql.com/downloads/ Accessed on 17 May

10. Microsoft MSDN Magazine, Move over DataGrid, there's a new grid in town! http://msdn.microsoft.com/en-us/magazine/cc163933.aspx

11. Wikipedia, Iterative and incremental development. http://en.wikipedia.org/wiki/Iterative_and_incremental_development Accessed on 18 May 2012

12. Microsoft ASP.NET Forums, Best practices for different CSS for different browsers. http://forums.asp.net/t/1405575.aspx/1 Accessed on 18 May 2012

Figure and code example list