European Organization for Nuclear Research
(CERN)

Summer Studentship 2016

Project Report

Product Licenses Database Application

Author:
Petar Tonkovikj
tonkovikj.petar@students.finki.ukim.mk

Supervisor:
John Evans
john.evans@cern.ch

August 19, 2016

Abstract

The goal of this project is to organize and centralize the data about software tools available to CERN employees, as well as provide a system that would simplify the license management process by providing information about the available licenses and their expiry dates. The project development process is consisted of two steps: modeling the products (software tools), product licenses, legal agreements and other data related to these entities in a relational database and developing the front-end user interface so that the user can interact with the database. The result is an ASP.NET MVC[1][2] web application with interactive views for displaying and managing the data in the underlying database.

List of Figures

List of Tables

Contents

1 Creating the entity-relationship model

The first step of the project development process was modeling the data in an entity-relationship diagram. To achieve this, we first had to define some key terms. After numerous discussions with the rest of the team involved in the project, the following definitions were agreed upon:

Definition 1. *A **product** is an entity that models a software tool. The scope of the product is defined by areas (e.g. engineering tool) and a functional element. The product may have multiple product versions associated with it.*

Definition 2. *A **product version** is an instance of a product and in most cases represents the software that can be installed and ran by the user. For each product version we need to keep track of the vendor, manufacturer and support giver, as they may change for different product versions.*

Definition 3. *A **license** is the right to use a software or certain features of a software. According to this we have defined product licenses (licenses that cover the entirety of a given product version) and feature licenses (licenses that cover some features of a given product version, i.e. one product version may have multiple features that have separate licenses).*

Definition 4. *A **maintenance agreement** provides support for a given license that has already been purchased. The expiry date of the maintenance agreement is independent from the expiry date of the license it is associated with. If the maintenance agreement expires and the license agreement is still valid, the software can still be used without support and updates.*

Definition 5. *A **legal agreement** is any other type of agreement associated with a product (e.g. a non-disclosure agreement).*

Definition 6. *An **order** is a purchase of license and/or maintenance agreements for one or more product versions. The order entity models the data of the CERN EDH[3] orders.*

Definition 7. *An **order line** is a single article from the order and represents a purchase of an arbitrary amount of license or maintenance agreements for a given license. The order line also specifies the start and expiry dates of the agreements it covers.*

The database diagram went through many iterations. The final diagram that best suits the above defined definitions is shown on Figure 1.

2 Choosing the appropriate technology for development

The next step was determining the appropriate technology for the project. The three main candidates were the following:

1. The content management system Drupal[4];
2. Oracle Application Express (APEX)[5];
3. A model-view-controller technology such as ASP.NET MVC[2].

After considering the pros and cons of each technology, briefly shown in Table 1, it was decided that ASP.NET MVC was the most suitable candidate as it offered much higher control over the database compared to Drupal and better possibilities for integration with JavaScript than Oracle APEX. Having in mind that we needed views to show data from a database, the model-view-controller approach matches the requirements, as suggested by the name itself.

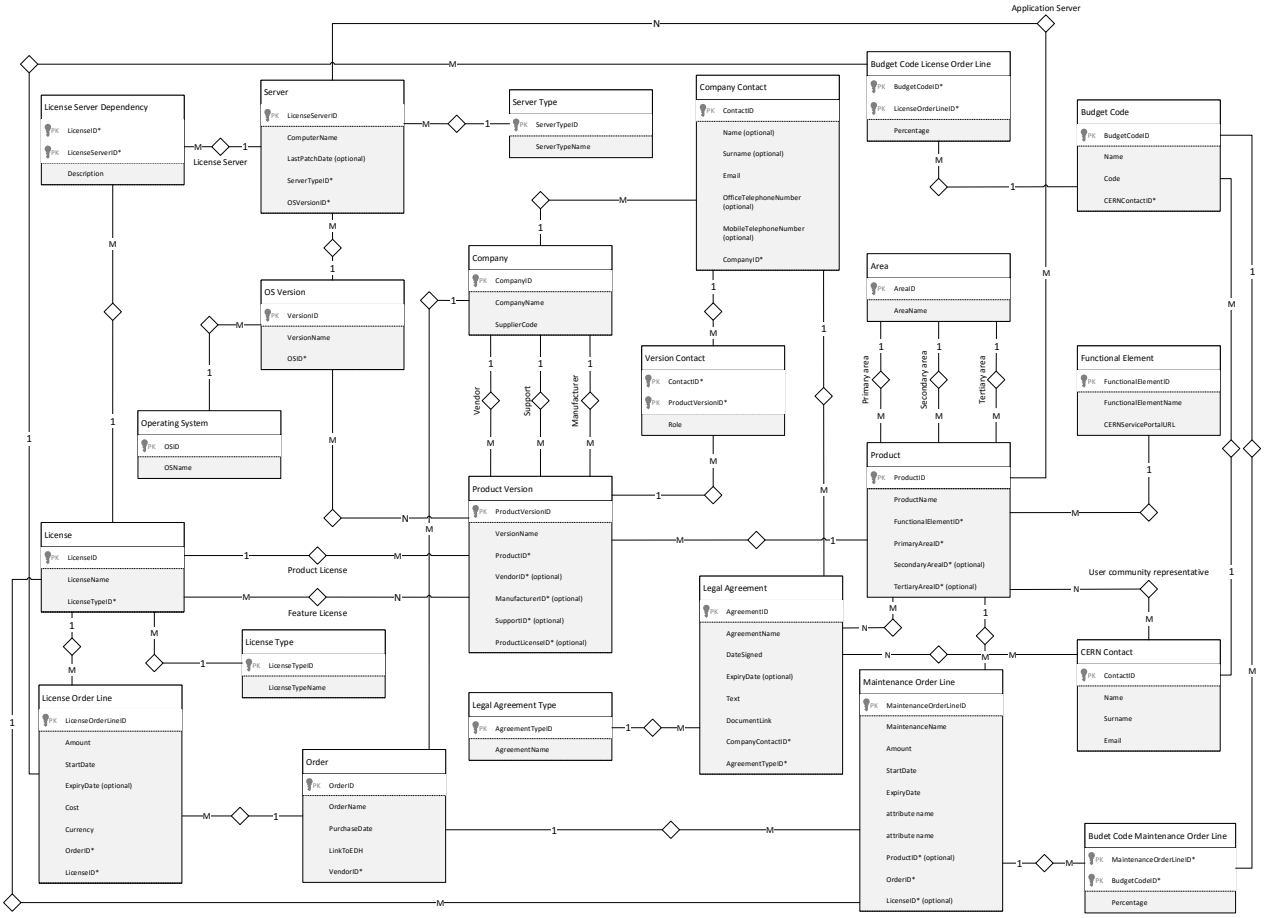

Figure 1: Final entity-relationship diagram for the proposed database.

Technology	pros	cons
Drupal	√ Good for adding static content. √ Easy management of user permissions.	× No control over the database in the background.
Oracle APEX	√ High control over database. √ Easy to implement views.	× Hard to integrate with other technologies (e.g. JavaScript).
ASP.NET MVC	√ High control over database. √ Easy to implement views. √ Works well with JavaScript.	× Questionable support for external user authentication.

Table 1: Comparison of technologies

3 Description of the ASP.NET project

This section is meant for people that are new to ASP.NET MVC and may work on maintaining and expanding the project web application. To help them get started, I will give a brief introduction to the model-view-controller concept, as well as a basic overview of the project files and folders.

3.1 ASP.NET MVC Concepts

The model-view-controller (MVC) concept of ASP.NET is a huge step forward compared to its predecessor, web forms. The inspiration for this concept came from the fact that in most cases our applications have a database that models some entities and our web application then has a task to access, display, manipulate and store the data in some shape or form. The main idea is that in this kind of scenario, it is better to have the web application centered around the entities from the database, instead of the HTML pages or web forms. This way we create the models for the entities first, and then create views to display said models.

The model for a database entity is nothing more than a class that contains attributes that match the columns of the table it represents. The view is the actual HTML structure that knows how to display the model in some way. The controller is a class that binds the model with the views that are created to display it. The binding is done by calling methods from the controller class which returns the view for the appropriate model.

For example, if we take a look at our database and if we want to display a product from the table "Product" in the database, we need to call a display method from the product controller that will extract the product from the database using a query. The controller will then pass this product (the model) to the view and return the view so that it can be shown in the browser. After this, the job of the view is to display the model that was passed to it by the controller.

For more details and further reading, please refer to [6]

3.2 Entity Framework

One of the things that makes ASP.NET MVC a convenient tool to use is Entity Framework[7]. This tool provides automatic generation of the models from given database and vice versa. It also models the relationships defined within the database by using object composition. For example, if there is a one-to-many relationship defined in the database, an object from the class on the one side will contain a list of objects from the class on the many side of the relationship. To generate all the models from a database, all that needs to be done is create a data model for the database by providing the database connection to Visual Studio.

What is more, Entity Framework also provides the option to automatically generate controllers for the newly generated models. These controllers will contain basic CRUD (Create, Read, Update, Delete) methods that are a good starting point for implementing more complex methods. This process of automatic code generation is called scaffolding[8].

For more information on Entity Framework, please refer to [9].

3.3 Project files and folders

This subsection is a brief introduction to the structure of the project. It contains several files and folders, most of which are quite self explanatory.

- The "App_Start" folder contains a couple of automatically generated C# files (with extension .cs) that deal with default authentication, method routing, rendering of style sheets and JavaScript.
- The "Content" folder contains the style sheets and is usually where other content, such as images, is stored.
- The "Controllers" folder contains the controller classes, also written in C#.
- The "fonts" folder can be used to add additional fonts, if so desired.
- The "Models" folder contains the automatically generated data model that contains the model classes.
- The "Scripts" folder contains client side scripts (JavaScript and JQuery).

- The "Views" folder contains subfolders for each controller, each of them containing the appropriate views for the controller methods. The subfolder labeled "Shared" contains the layout views that are shared among all the views in the application.
- The "Web.config" file contains the configurations for the application in XML format.

The "Models" folder also contains two files named "Metadata.cs" and "PartialClasses.cs". The first one is used to specify metadata about the model attributes which is used for validation. The second one is used for binding the metadata classes with their appropriate models. The validation metadata can also be specified within the models themselves, however, if changes are made to the database, then the data model needs to be recreated. When the data model is recreated, the model classes are regenerated and lose any additional code added to them manually, including the validations. That is why the validations should be stored in a separate file that is independent of the data model. For more information about this way of implementing validations, please refer to [10].

4 Summary

To sum up, the result of the project is a database web application developed in the ASP.NET MVC Framework. The goal of this application is to provide the users with useful information about which product licenses are currently available, as well as give out warnings about the expiry dates of various types of agreements in an attempt to make the license management process easier. The application provides interactive views for displaying, editing and deleting data from the database. Finally, the application can easily be upgraded with additional functionalities by anyone with a little bit of experience with the ASP.NET Framework.

References

[1] *Model–view–controller*. Web page retrieved 17/08/2016 from `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`.

[2] *Learn About ASP.NET MVC*. Web page retrieved 17/08/2016 from `http://www.asp.net/mvc`.

[3] *Electronic Document Handling System at CERN*. Web page retrieved 17/08/2016 from `https://edh.cern.ch/Desktop/about.jsp`.

[4] *Drupal - Open Source CMS*. Web page retrieved 18/08/2016 from `https://www.drupal.org/`.

[5] *Oracle Application Express*. Web page retrieved 18/08/2016 from `https://apex.oracle.com/en/`.

[6] *Getting Started with ASP.NET MVC 5*. Web page retrieved 18/08/2016 from `http://www.asp.net/mvc/overview/getting-started/introduction/getting-started`.

[7] *Entity Framework*. Web page retrieved 18/08/2016 from `http://www.asp.net/entity-framework`.

[8] *Scaffold (programming)*. Web page retrieved 18/08/2016 from `https://en.wikipedia.org/wiki/Scaffold_(programming)`.

[9] *Entity Framework (EF) Documentation*. Web page retrieved 18/08/2016 from `https://msdn.microsoft.com/en-us/data/ee712907`.

[10] *EF Database First with ASP.NET MVC: Enhancing Data Validation*. Web page retrieved 18/08/2016 from `http://www.asp.net/mvc/overview/getting-started/database-first-development/enhancing-data-validation`.

.