Spring 2015
.NET Web Application Development
Homework 3

- Due March 31, 2015 at 3:00 PM Eastern Daylight Time.
- Submit via NYU Classes.
- Late homework will not be accepted after the deadline. To avoid problems, submit well in advance.
- **No collaboration permitted at all**. Not even with members of your assigned group. All submissions must be original. Originality of all submissions will be confirmed.
- **Do not wait until the last minute to complete this assignment**.

For this homework assignment, you will create a data model for a hypothetical web site using Entity Framework.

1 Introduction

Businesses often need to keep track of multiple projects and the people working on those projects. The questions that most businesses need to answer are: how much time is each person spending on a particular project and what specific tasks are those people working on?

There is often a monetary aspect too. People working on the project may bill their time on an hourly basis, in which case the project cost is a function of each person's time spent. Conversely, the business may be charging their client an hourly rate for any time spent on the project, in which case all time spent on the project (among the people working on it) will generate revenue for the company.

For this assignment we will create the lowest layer of the hypothetical web site—the data layer. You should use Entity Framework to do this. You may choose between a "model first" or "code first" approach. The "database first" approach is also available, but not recommended unless you have database expertise.

In the real world, descriptions of software requirements are usually incomplete even when considerable effort has been made to make them as complete as possible. In a real world scenario, you would discuss any missing pieces with your customer to ascertain any missing information. For this assignment, you are permitted to make reasonable assumptions whenever you encounter a situation in which you find that incomplete information exists. In this case, you should **state your assumptions explicitly**. The best place to do this is either in the source code comments or separately in the inline portion of your submission. If you are confused about the meaning of a requirement, believe your approach may be considered unreasonable, or believe there is a conflict between different parts of the requirements, then by all means please seek clarification before submitting your assignment.

Almost all questions should be posted to the Forum on NYU Classes for the benefit of other students. The only exception is when the nature of your question requires you to reveal parts of your solution. When possible, try to pose your question in a way that does not require you to share your solution so that other students may benefit from the answers. I reserve the right to repost questions that are sent by email, but should have been submitted by NYU Classes. Likewise, any clarifications I provide on the NYU Classes Forum are considered official for grading purposes. Therefore, ensure that you read the Forum posts (and answers) regularly. Students are also welcome (and highly encouraged) to assist each other on the forum, rather than waiting for me or graders to answer.

2 The data model

Here we will describe the requirements that you must meet in creating the data model. Note that this description is intentionally given in “plain English” and not in technical terms because this is the most likely form it will take in the real world. As such, terms like “record” and “container” may or may not correspond directly to any particular programming language construct, database concept, or anything else. The exact design of the data model is ultimately your choice.

All of the model classes you submit must be your own. Do not use any Identity classes or other classes originating from the .NET or other framework.

- A *user* is any person who has a registered account on the web site. A *visitor* is a person who uses the web site, but is not registered. For each user, the website should maintain a record consisting of the person’s first name, middle initial, and last name, as well as an indication of whether they are an employee or subcontractor.
- A *project* consists of a container of “user entries,” a container of “time entries” and a container having a subset of “activities,” all of which are described below. A project and its constituent parts mentioned above cannot be deleted permanently, but it can be *archived*, meaning that the project and its parts are hidden from view. This might happen if the project has ended and the user wishes to keep the project data around, but not actively work with the project. Once archived, a project (and its parts) may be unarchived, in which case the project is visible again as if it were never archived. One may archive and un-archive projects at will, with no limitations.
- An *activity* is a short description of the work being performed. Example activities might include consulting, software development, accounting, etc. They represent the different tasks that a person might carry out when working on a project. Each user’s account should maintain a list of activities that can be used by any projects belonging to the user.
- A *user entry* is a record consisting of a user (who is working on the project), the aforementioned user’s project-specific hourly cost, and the aforementioned user’s project-specific hourly billing rate. The user entry should also store a flag for each activity indicating whether the activity is billable to the customer for the project in question (the same activity may be billable to the client in one project, but not another.) An activity being *billable* means that the user will bill her client for any work performed on that activity, resulting in project revenue. By way of example, if a user’s cost is $50 and billing rate is $75 and the activity is billable, then that means that the user will spend $50 on the user, but collect $75 from the customer for a net profit of $25.
- A *time entry* is a record consisting of a project, activity, user, a time value, and a description of the work performed (as text). Intuitively, this

represents an amount of time that a user spends on a particular activity for a particular project. One constraint is that the activity must be a member of the user entry's subset of activities.

The model must perform validation. It must assume that the user will always enter the wrong value, in the wrong format, at an incorrect length, etc. Value ranges must be enforced. Strings must have a specified maximum length. The model must specify which fields are required. The model must supply default values where it makes sense to do so. The model must perform validation on any input requiring a specific format, such as phone numbers, email addresses, credit card numbers, dates, etc. All of this can be accomplished easily using data annotations. See this web page for helpful information on data annotations that are specific to validation.

If you are using "code first," then just write the attributes right into your code. If you are using "model first," then your validation attributes will be overwritten each time you generate the code/database from the model. To avoid your attributes being overwritten, the proper way to handle this situation is using this approach.

3 Business logic

Although this homework assignment will not involve building any business logic or user interface features, here we provide information for your benefit in understanding the context in which the data model will be used. You may find that you will need to add certain model classes in order to perform the business logic described below. You may also find that some of the descriptions below will dictate how you implement the model classes you already created above.

A person may become a user by registering for the web site. Their email address will uniquely identify them among other users. As part of the registration process, the site should solicit the information described above under the "user" item.

Once registered, the user may "delete" their account, which will have the appearance that the item has been deleted, but without removing any data from the database. Visitors to the site may only access static informational pages (About, FAQ, etc.)

Upon logging in, users should be directed to a *dashboard* upon login. The dashboard should display statistics about all active projects that were created by the user. The statistics should include at least the number of non-archived projects and a listing of each project, with each item in the listing to include: the project's name, its total costs to date and its total revenues to date. If both pieces of information are available, then it should also display the profit to date (computed as revenues minus cost to date).

The dashboard should also allow the user to select from the following options, where each option will direct the user to a page specifically dedicated to the selected option:

- Track time: this will allow the user to submit time entries for a particular day. The user should be able to add, edit, or delete time entries. To add a new time entry, the user must select the project and activity to enter the time for, and then enter the description and total time spent on that activity. There could be multiple records for the same activity if the user so desired. Time entries should be entered in hour:minute format.
- Manage projects: this allows the user to add, edit, or archive projects. It also allows the user to maintain a list of activities which are global to all projects. By default only active (unarchived) projects should be shown. There must be some way to make archived projects visible and also a means of unarchiving a project. When adding or editing projects, the user can add another user to the project by entering the email address and the required information for that user as explained above. The user should also have a means of selecting existing activities or adding new activities to the project. For simplicity, you may assume that activities may not be deleted once added.
- Reporting: this screen allows the user to see information relating to projects, including the ability to view project activity over selected periods of time. The reporting should allow viewing of statistics over a particular date window, relating to either:
  - all projects
  - a single user, across all projects
  - a single project
  - a single user, across one project

  Any of the selections above should result in cost, revenue and profit information (if cost and revenue are both provided) being shown for the selected level of granularity above.

4 Grading and Deliverables

The homework will be graded primarily based on adherence to the requirements and good coding and design practices. It is not the objective of this assignment to grade the assumptions that are made, if any. However, if an assumption is stated and the model you submit does not properly reflect that assumption or is a poor implementation of the assumption, then a point deduction may occur. A point deduction may occur if the assumption is (quite) unreasonable.

You must submit a fully functioning Visual Studio project consisting of a data layer, implemented using Entity Framework, per the requirements above. Please **clean** the solution from the build menu before submitting so that derivative files such as executables and object files are not included.

In addition to the data layer itself, you should write a console application (as a project within the solution) which:

1. Creates your data layer objects and populates them with data from an XML file. Standalone classes must have at least one object. Containers of objects should contain at least three objects.

2. In one instance of your context, persist the data in the entities to the database using `SaveChanges`. At least one object of each class type should be included.

3. In a different instance of the same context, query information from the entities and print out the object's properties to the console. (You can use `System.Console.WriteLine` for this purpose. I recommend using `System.Console.ReadKey` to prevent the console window from closing when executing.

Do not wait until the last minute to complete this assignment.