PRACTICA 07 – E-LEARNING

Servicio RESTful con ASP.Net Web API

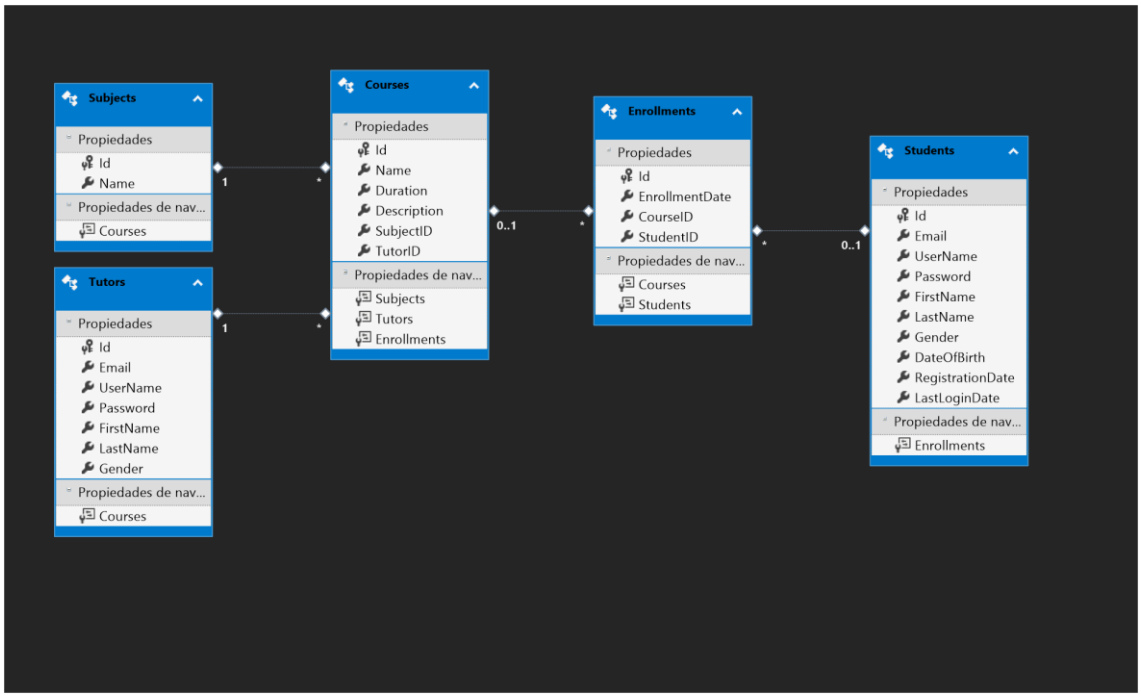
DESARROLLO RÁPIDO DE APLICACIONES

Por Juan José Camacho Hidalgo
UNIVERSIDAD DE ALMERÍA

Un servicio RESTful es aquel que implementa una arquitectura REST. La arquitectura REST se define por utilizar los métodos HTTP de manera explícita, exponer URIs con forma de directorios, transfiere XML, JSON o ambos y además no mantiene estado. Esto último quiere decir, que mueve la responsabilidad de mantener el estado al cliente de la aplicación, esto facilita el diseño, escritura y distribución del servicio a través de servidores, al contrario que ocurre en servlets/JSP o EJB de Java EE donde programadores hemos tenido que sufrir la causa de una java.io.NotSerializableException por ejemplo.

Esta solución de e-Learning está formada por un patrón Model-View-Controller y una biblioteca de clases. El modelo y el controlador ocurre en LearningMvcApp.Web y UsingDBLearningWpfApp muestra el modelo de la bd así como la vista en XAML. Al final resulta una aplicación web, si unimos la biblioteca de clases creada LearningClassLibrary.

Para la base de datos, se ha usado Entity Framework Code-First, definiendo objetos mediante POCO. El objetivo es crear una API ASP.Net. Para la construcción se ha usado el siguiente diagrama de clases, donde tenemos Subjects, Tutors, Courses, Enrollments y Students.



Dentro de nuestra librería LearningClassLibrary.Data, tenemos una carpeta llamada Mappers. En esta carpeta se muestran las clases CourseMapper, EnrollmentMapper, StudentMapper, SubjectMapper, y TutorMapper. Estas clases se encargan de aplicar reglas de asignación, definiendo el tipo de datos por columna, si es nullable un dato, así como el mapeado de las relaciones FK y PK de cada tabla, así como especificar las columnas de identidad. Todas las clases provienen de System.Data.Entity.ModelConfiguration.EntityTypeConfiguration.

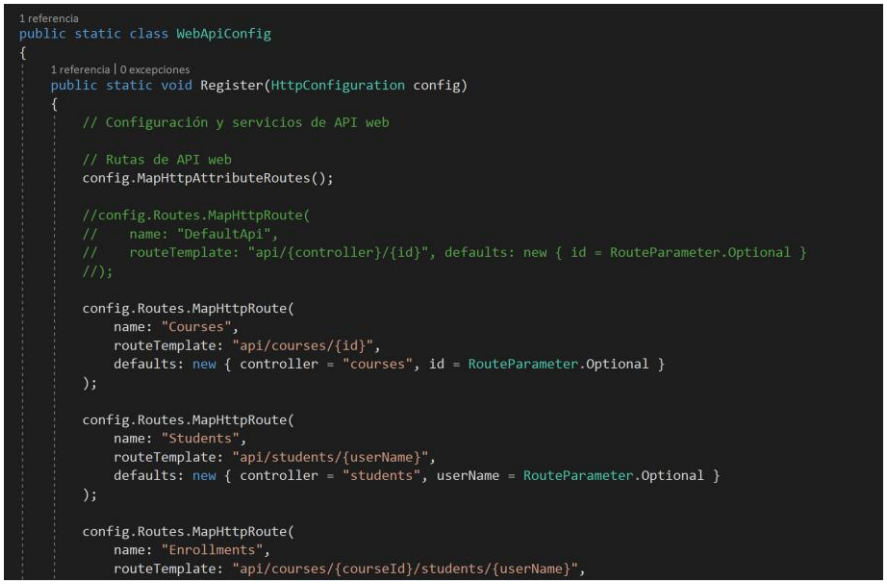
La clase LearningContext que deriva de la clase System.Data.Entity.DbContext, tiene el objetivo de llevar a cabo la persistencia de los datos mediante la definición de los DbSet de cada tabla así como su mapeo.

```
public class LearningContext : DbContext
{
    4 referencias | 0 excepciones
    public LearningContext():
        base("eLearningConnection")
    {
        Configuration.ProxyCreationEnabled = false;
        Configuration.LazyLoadingEnabled = false;

        Database.SetInitializer(new MigrateDatabaseToLatestVersion<LearningCont
    }

    14 referencias | 0 excepciones
    public DbSet<Course> Courses {get;set;}
    4 referencias | 0 excepciones
    public DbSet<Enrollment> Enrollments { get; set; }
    12 referencias | 0 excepciones
    public DbSet<Student> Students { get; set; }
    4 referencias | 0 excepciones
    public DbSet<Subject> Subjects { get; set; }
    2 referencias | 0 excepciones
    public DbSet<Tutor> Tutors { get; set; }

    0 referencias | 0 excepciones
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Configurations.Add(new StudentMapper());
        modelBuilder.Configurations.Add(new SubjectMapper());
        modelBuilder.Configurations.Add(new TutorMapper());
        modelBuilder.Configurations.Add(new CourseMapper());
        modelBuilder.Configurations.Add(new EnrollmentMapper());

        base.OnModelCreating(modelBuilder);
    }
}
}
```

Por otro lado la clase LearningContextMigrationConfiguration se hace cargo de la gestión de la migración de los datos, para evitar (o no) que los datos se pierdan, y además, se encarga de parar a la clase LearningDataSeeder, que crea la base de datos.

Además, mediante el patrón Repository, se crea la clase LearningRepository , que implementa la interfaz ILearningRepository, para implementar los métodos necesarios para el manejo de datos de la base de datos: por tanto éste actúa como una capa de acceso.

Hasta aquí la parte de datos. Ahora nos centramos en la API. Dentro de la clase WebApiConfig de App_Start del proyecto LearningMvcApp.web se define la configuración de las URI relativas, por el que se encarga del enrutamiento de la API.



```
1 referencia
public static class WebApiConfig
{
    1 referencia | 0 excepciones
    public static void Register(HttpConfiguration config)
    {
        // Configuración y servicios de API web

        // Rutas de API web
        config.MapHttpAttributeRoutes();

        //config.Routes.MapHttpRoute(
        //    name: "DefaultApi",
        //    routeTemplate: "api/{controller}/{id}", defaults: new { id = RouteParameter.Optional }
        //);

        config.Routes.MapHttpRoute(
            name: "Courses",
            routeTemplate: "api/courses/{id}",
            defaults: new { controller = "courses", id = RouteParameter.Optional }
        );

        config.Routes.MapHttpRoute(
            name: "Students",
            routeTemplate: "api/students/{userName}",
            defaults: new { controller = "students", userName = RouteParameter.Optional }
        );

        config.Routes.MapHttpRoute(
            name: "Enrollments",
            routeTemplate: "api/courses/{courseId}/students/{userName}",
```

Como vemos, así se seleccionan los directorios.

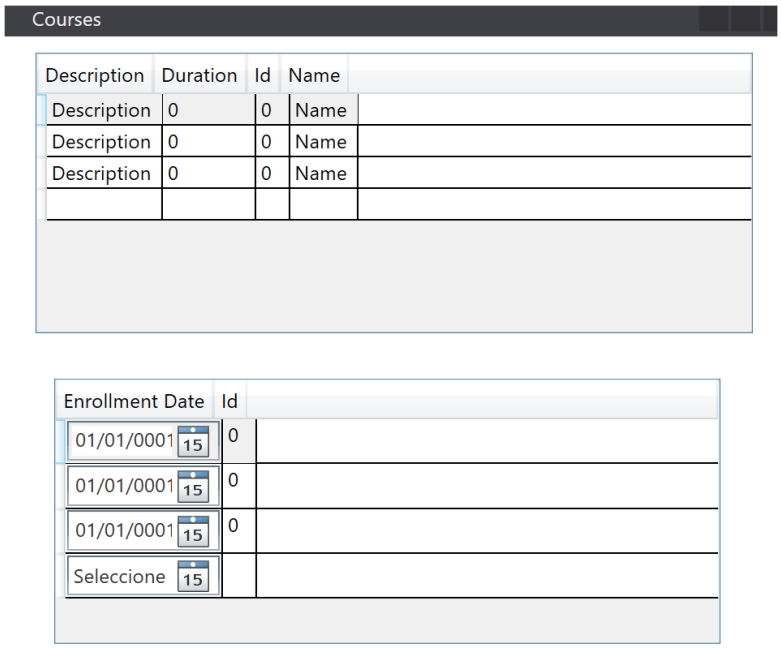
El controlador principal es BaseApiController, que se encarga de llamar al Repository (los datos), e interactuar con el modelo , mediante ModelFactory. Este patrón nos ayudará a formar y controlar la respuesta al cliente.

Para las acciones del controlador, individualmente, se han implementado métodos para las acciones HTTP: GET, POST, PUT y DELETE en la Web API, en cada una de las clases de los controladores. GET sirve para recuperar información identificada por el Request-URI. POST se usa para enviar información al servidor, como por ejemplo una actualización o información de algún dato. PUT se usa para solicitar un almacenamiento de un dato cualquiera en una ubicación dada. Y por último, DELETE elimina un recurso determinado en la ubicación dada.

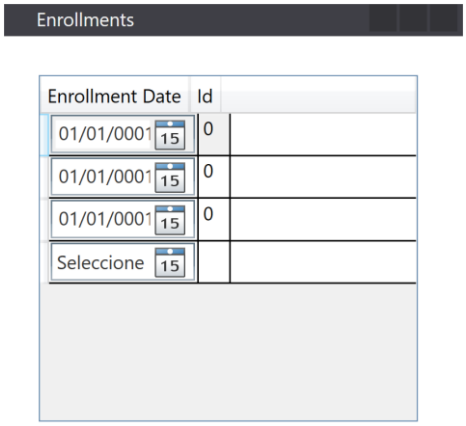
La clase NinjectWebCommon se encarga de la inyección de dependencias haciendo uso de la librería de Ninject y haciendo uso de la interfaz IKernel, la cual administrará la aplicación. Mediante esto, se pueden cargar módulos para inyectar dependencias necesarias.

En cuanto a la vista, así se define:

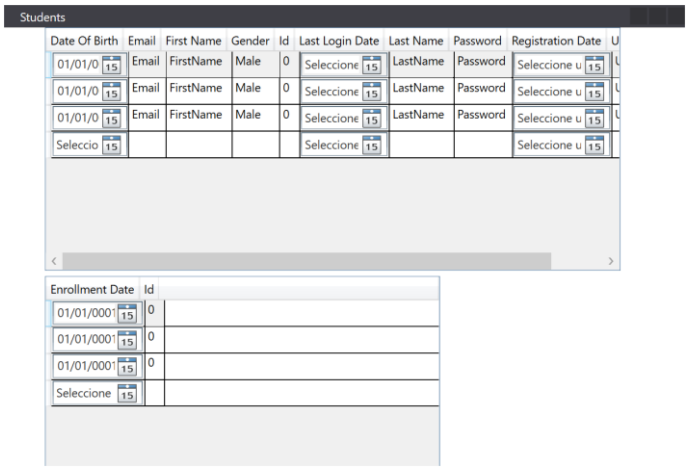
Cursos



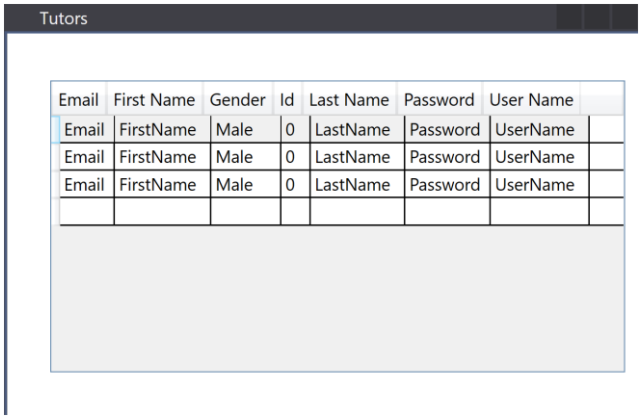
Inscripciones



Estudiantes:



Tutores:


Tutors
Email
First Name
Gender
Id
Last Name
Password
User Name
Email FirstName Male 0 LastName Password UserName
Email FirstName Male 0 LastName Password UserName
Email FirstName Male 0 LastName Password UserName