PA181-3-DotNetGen

DotNetGenerator: Generador de Código para Arquitectura Microsoft .NET a partir de modelos ISML

Julián Andrés Sánchez Lozada

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2018

PA181-3-DotNetGen

DotNetGenerator: Generador de Código para Arquitectura Microsoft .NET a partir de modelos ISML

Autor:

Julián Andrés Sánchez Lozada

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TÍTULO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Director

Ing. Jaime Andrés Pavlich Mariscal Ph.D.

Comité de Evaluación del Trabajo de Grado

Carlos Andrés Parra Acevedo

Luis Guillermo Torres

Página web del Trabajo de Grado

http://pegasus.javeriana.edu.co/~PA181-3-DotNetGen

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Mayo, 2018

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico

Jorge Humberto Peláez, S.J.

Decano Facultad de Ingeniería

Ingeniero Jorge Luis Sánchez Téllez

Director Maestría en Ingeniería de Sistemas y Computación

Ingeniera Angela Carrillo Ramos

Director Departamento de Ingeniería de Sistemas

Ingeniero Efraín Ortíz Pabón

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

Deseo brindar mis más sinceros agradecimientos a mis padres quienes siempre estuvieron a mi lado dándome el apoyo necesario. A mi familia quienes me respaldaron durante el desarrollo de este proyecto, y en especial a Dios, el cual me ha dado fortaleza para no decaer ante las adversidades encontradas. A los docentes del programa por compartir sus conocimientos. A Jaime Andrés Pavlich Mariscal por su respaldo en el desarrollo de este proyecto y a todas aquellas personas que de una manera u otra participaron en la elaboración de este trabajo.

Julián Andrés Sánchez Lozada

Contenido

INTRODUCCIÓN **13**

1. DESCRIPCIÓN GENERAL **13**

1.1 PROBLEMÁTICA 13

1.2 OPORTUNIDAD 14

2. DESCRIPCIÓN DEL PROYECTO **14**

2.1 OBJETIVO GENERAL 14

2.2 OBJETIVOS ESPECÍFICOS 14

2.3 METODOLOGÍA 14

3. MARCO TEÓRICO / ESTADO DEL ARTE **15**

3.1 MDE (MODEL DRIVEN ENGINEERING) 16

3.1.1 MDA (Model Driven Architecture) *16*

3.1.2 ISML (Information Systems Modeling Language) *17*

3.1.3 AndroMDA *18*

3.1.4 OpenArchitectureWare (OAW) *19*

3.1.5 Acceleo *20*

3.1.6 Generación de código utilizando T4 (Text Template Transformation Toolkit) *21*

3.2 ASP.NET MVC 21

3.2.1 Entity Framework *22*

3.2.2 Fluent API *23*

3.2.3 SQL Server 2016 Express LocalDB *23*

3.2.4 Patrón de diseño Repository en ASP.NET MVC *23*

4. TRABAJOS RELACIONADOS **24**

4.1 GENERADORES DE CÓDIGO PARA VISUAL STUDIO 24

4.1.1 Xomega *24*

4.1.2 CodeTrigger *25*

5. DESARROLLO DEL PROYECTO **26**

5.1 DESCRIPCIÓN GENERAL DEL SISTEMA 26

5.1.1 Limitaciones del Sistema *27*

5.2 REQUERIMIENTOS DEL SISTEMA 27

5.2.1 Requerimientos Funcionales *27*

5.2.2 Requerimientos No Funcionales *29*

5.3 DISEÑO DEL GENERADOR DOTNETGENERATOR 30
5.3.1 Vista de Procesos del Sistema 30
5.3.2 Arquitectura del Sistema 31
5.3.3 Vista Lógica del Sistema 34
5.3.4 Vista Física y Despliegue del Sistema 35
5.3.5 Estructura del Sistema 37

6. PRUEBA DE CONCEPTO 44

6.1 DESCRIPCIÓN DE LA APLICACIÓN DE PRUEBA 44
6.1.1 Módulos: Aplicación para Gestionar una Tienda de Mascotas 45

6.2 DESCRIPCIÓN DEL MODELO ISML 46
6.2.1 Definición de Controladores, Páginas y Servicios 48

6.3 APLICACIÓN WEB ASP.NET MVC GENERADA 49
6.3.1 Arquitectura de la Aplicación Generada 49
6.3.2 Estructura Gráfica de la Aplicación 51
6.3.3 Interfaz Gráfica: Proceso para crear una nueva Venta 51
6.3.4 Interfaz Gráfica: Proceso para agregar productos a la Venta 53
6.3.5 Interfaz Gráfica: Proceso para registrar el Pago 54

6.4 ANÁLISIS DE LOS RESULTADOS 55
6.4.1 Reducción de esfuerzo en términos de líneas de código 55

7. CONCLUSIONES 57

REFERENCIAS 58

Listado de Tablas

Tabla 1: Especificación de las Fases, Etapas y Entregables del proyecto..... 15

Tabla 2: Requerimientos Funcionales del sistema..... 29

Tabla 3: Requerimientos No Funcionales de Mantenibilidad..... 29

Tabla 4: Requerimientos No Funcionales de Portabilidad..... 30

Tabla 5: Características técnicas de la aplicación Web generada..... 33

Tabla 6: Listado de componentes externos del sistema..... 35

Tabla 7: Listado de los métodos principales de un generador. 39

Tabla 8: Listado de las clases definidas para cada generador..... 41

Tabla 9: Listado de las clases definidas para cada plantilla..... 43

Tabla 10: Listado de los métodos definidos en la clase UtilGenerator..... 44

Tabla 11: Módulos de la aplicación Web para gestionar una tienda de mascotas 46

Tabla 12: Listado de Controladores, Páginas y Servicios especificados en ISML 49

Tabla 13: Interfaces de usuario del proceso para crear una nueva Venta 52

Tabla 14: Interfaces de usuario del proceso para agregar productos a la Venta 54

Tabla 15: Interfaces de usuario del proceso para registrar el pago..... 55

Tabla 16: Comparativa entre líneas de código ISML y líneas generadas en ASP.NET MVC 56

Listado de Figuras

Ilustración 1: Adaptación de la Metodología DAD al Ciclo de vida del proyecto 15

Ilustración 2: Ciclo de vida de un proceso de desarrollo utilizando MDA [15] 17

Ilustración 3: Elementos de un modelo expresado en ISML [2]..... 18

Ilustración 4: Patrón Repository en ASP.NET MVC con Entity Framework..... 24

Ilustración 5: Vista de procesos del sistema 31

Ilustración 6: Modelo detallado para la generación de código con DotNetGenerator 32

Ilustración 7: Vista lógica del sistema 34

Ilustración 8: Vista física y despliegue de sistema 36

Ilustración 9: Diagrama con la estructura general del sistema..... 38

Ilustración 10: Paquete co.edu.javeriana.dotnetgenerator.generators..... 39

Ilustración 11: Paquete co.edu.javeriana.dotnetgenerator.templates 42

Ilustración 12: Entidades del Modelo ISML..... 46

Ilustración 13: Entidad “Producto” modelada en ISML 47

Ilustración 14: Controlador “ProductoController” modelado en ISML..... 47

Ilustración 15: Servicio “ServicioPagos” modelado en ISML..... 47

Ilustración 16: Página “CrearProducto” modelada en ISML..... 47

Ilustración 17: Arquitectura de la aplicación Web Generada 50

Ilustración 18: Distribución gráfica de la aplicación Web generada 51

Ilustración 19: Comparativa entre líneas de código ISML y líneas en ASP.NET MVC 56

ABSTRACT

Building Web applications has become a complex issue for organizations, as more effort is being invested in implementation details and less in other important aspects such as business logic and quality. There are also constant changes in the technological platforms and programming languages, which make it difficult for them to learn in organizations. To address this problem the DotNetGenerator code generator has been developed, which applies the approach of Model Driven Engineering (MDE), for the generation of source code ASP.NET MVC Web applications, from Models defined in ISML.

RESUMEN

Construir aplicaciones Web se ha convertido en un tema complejo para las organizaciones, ya que se está invirtiendo más esfuerzo en los detalles de implementación y menos en otros aspectos importantes como la lógica de negocio y la calidad. También existen cambios constantes en las plataformas tecnológicas y lenguajes de programación, que dificultan su aprendizaje en las organizaciones. Para enfrentar esta problemática se ha desarrollado el generador de código DotNetGenerator, el cual aplica el enfoque de la Ingeniería Dirigida por Modelos (MDE), para la generación de código fuente de aplicaciones Web ASP.NET MVC a partir de Modelos definidos en ISML.

RESUMEN EJECUTIVO

Microsoft cuenta con una plataforma robusta para el desarrollo Web denominada ASP.NET MVC [1], la cual está conformada por un Framework, lenguajes de programación, y ambientes de desarrollo integrado IDE que facilitan el desarrollo. Pero como en cualquier plataforma de desarrollo existen un conjunto de tareas de desarrollo que se repiten una y otra vez en cada proyecto. Un ejemplo de esto se puede evidenciar en el proceso de construcción de los componentes de la Arquitectura MVC, la cual tiene como elementos base: Modelos, Páginas, Controladores, y en algunos casos Servicios de respaldo. Se requiere de un mecanismo que permita automatizar la generación de estos componentes comunes, que ayude a reducir el esfuerzo en costo y tiempo invertido en la generación de estos componentes.

La Ingeniería Dirigida por Modelos MDE (Model Driven Engineering) [2], permite definir modelos, métodos y herramientas aptas para la representación de sistemas de software. Permite abstraer cualquier tipo de aplicaciones a modelos, y reducir significativamente los niveles de complejidad relacionados con la especificación de una aplicación [2]. MDE Facilita la posibilidad de automatizar el proceso de transformación de un Modelo en una Aplicación, mediante la construcción de generadores de código para una plataforma específica.

El ambiente de ISML (Information Systems Modeling Language) [2], construido por Investigadores de la Pontificia Universidad Javeriana, e ingenieros de la empresa Heinsohn Business Technology (HBT), permite aplicar el enfoque MDE. Este ambiente permite definir el modelo de una aplicación MVC, mediante un lenguaje de dominio específico DSL [3] denominado ISML. Este lenguaje textual es mucho más sencillo que Java y C#, y provee una infraestructura tecnológica que facilita la construcción de generadores de código para una plataforma específica.

Producto de Software Generado

El objetivo alcanzado por este proyecto fue la construcción del generador de código denominado DotNetGenerator. Este generador le permite a Ingenieros de Desarrollo y Arquitectos de Software; generar el código fuente de una Aplicación Web ASP.NET MVC [1] a partir de un modelo previamente definido en el lenguaje ISML (Information Systems Modeling Language). Este código fuente generado puede ser abierto utilizando la herramienta de desarrollo Visual Studio Community 2017 [4], desde donde podrá ser compilado y ejecutado. También contiene los componentes más comunes de una Aplicación Web ASP.NET MVC [1], como, por ejemplo: Modelos, Vistas, Controladores e Interfaces de Servicio. Esto le permite al ingeniero de desarrollo abstraerse de los detalles de implementación de estos aspectos de una aplicación, logrando una reducción del esfuerzo que se hubiera tenido que invertir en la construcción de dichos componentes. DotNetGenerator forma parte de los Plugins de Eclipse [5] utilizados para la generación de código fuente, del ambiente de modelado MDE, denominado ISML (Information Systems Modeling Language) [2].

Prueba de concepto

Para realizar las pruebas sobre este generador, se construyó una prueba de concepto, la cual consistió en modelar una aplicación en ISML para gestionar una tienda de mascotas. Se modelaron los siguientes procesos: Gestión de productos, Gestión de Clientes, Gestión de Venta de Productos. Este modelo fue procesado por el generador de código DotNetGenerator. Producto de este proceso se generó la estructura de archivos y directorios de una aplicación Web ASP.NET MVC [1], que utiliza Entity Framework [6] con el enfoque Code First [7]. Este código contiene todos los componentes modelados en el ISML (Entidades, Controladores, Páginas y Servicios).

Análisis de resultados

Para analizar los resultados se realizó la siguiente medición: se contabilizaron las líneas de código correspondientes a cada uno de los componentes de ISML modelados (Entidades, Páginas, Controladores y Servicios). También se contabilizaron las líneas de código generadas en ASP.NET MVC [1]. El objetivo de este análisis fue identificar la reducción del esfuerzo asociada al uso del generador de código, teniendo como criterio las líneas de código tanto del modelo como del resultado final de la generación. Como resultado del proceso de medición, se pudo concluir que a partir de las 1994 líneas modeladas en ISML, el generador escribió un total de 3497 líneas de código en la plataforma destino. Por lo cual se determinó la siguiente proporción de equivalencia: cada línea en escrita en ISML equivale a 1,8 líneas de código en ASP.NET MVC [1]. Este resultado muestra una reducción del esfuerzo en términos de líneas de código, gracias al uso del generador de código.

El componente que generó la mayor proporción de líneas de código en la plataforma destino fue el de tipo Entidad, ya que según los resultados cada línea en ISML de entidad equivale a 3,1 líneas de código en C#. Esto se debe a que en ASP.NET MVC, las entidades no solo se mapean como archivos de clase en C# por cada entidad. Estas también son la base para construir otros archivos como por ejemplo el "ApplicationDbContext.cs", en donde cada entidad no solo es declarada, allí también se establecen las relaciones entre las entidades mediante el Fluent API [8] del Entity Framework [6]. El segundo componente que generó la mayor proporción de líneas de código en la plataforma destino fue el de tipo Servicio, ya que según los resultados cada línea en ISML de servicio equivale a 2,3 líneas de código en C#. Este resultado se obtuvo porque cada servicio genera dos archivos de código independientes en C#, uno contiene la interfaz del servicio y otro su implementación.

El tercer componente en orden de proporción de líneas de código en la plataforma destino fue el correspondiente a las páginas, ya que según los resultados cada línea en ISML de página equivale a 1,8 líneas de código en archivos de sintaxis Razor [9]. El cuarto componente en orden de proporción de líneas de código en la plataforma destino fue el de tipo Controlador, ya que según los resultados cada línea en ISML de controlador equivale a 1,3 líneas de código en C#. Este componente no tuvo una proporción significativa, esto se debe a la gran similitud que existe entre la estructura y la sintaxis del lenguaje ISML con la de una clase de tipo controlador en C#.

INTRODUCCIÓN

Las organizaciones de la industria del software enfrentan varios desafíos: construir software en el menor tiempo posible, reducir los costos en el desarrollo, garantizar la calidad en cada uno de los productos, y una reducción de los equipos de desarrollo de software.

Para ayudar a resolver esta problemática se realizó a través de este trabajo la construcción de un generador de código fuente denominado "DotNetGenerator". Este generador le permite a los equipos de desarrollo de software reducir el tiempo y el esfuerzo invertido en la construcción de aplicaciones Web ASP.NET MVC [1], aplicando el enfoque de la Ingeniería Dirigida por Modelos MDE [2], y utilizando como base tecnológica el ambiente de ISML (Information Systems Modeling Language). Para lograr la automatización de todas aquellas tareas comunes y repetitivas asociadas a la creación de los componentes básicos de una Arquitectura MVC [10].

Este documento cuenta con una estructura compuesta por 7 secciones en las cuales se especifica en detalle todo el proceso realizado para la construcción del generador de código fuente.

En la primera sección del documento se encuentra la descripción general, en esta sección se describe la oportunidad y problemática que se está intentando resolver. En la segunda sección se encuentran los objetivos, el alcance del proyecto, y la metodología utilizada en el proceso. En la tercera sección del documento se encuentra el marco teórico utilizado como base para la construcción del proyecto. En la cuarta sección se describen y analizan otros trabajos relacionados con el proyecto. En la quinta sección se encuentra todo el proceso para el desarrollo del proyecto: Descripción general del sistema, requerimientos funcionales y no funcionales, Diseño del sistema (Vista de procesos, Arquitectura del sistema, Vistas física, lógica y despliegue) y la estructura del sistema.

En la sexta sección se encuentra la descripción de la prueba de concepto realizada para probar el generador. Se describe el Modelo ISML construido para ser procesado. También se describe la aplicación Web generada por medio del generador DotNetGenerator. Finalmente un análisis de los resultados obtenidos, a través de un criterio de medición de líneas de código, que pretende medir la reducción del esfuerzo en términos de líneas de código. En la séptima sección del documento se encuentran las conclusiones del trabajo.

1. DESCRIPCIÓN GENERAL

1.1 Problemática

Se requiere de un generador de código fuente para ASP.NET MVC [1], que permita automatizar en gran medida la construcción de todos los componentes comunes de una aplicación Web. Que ayude a reducir el esfuerzo en costo y tiempo invertido en la generación de estos componentes. También que facilite el desarrollo de un mínimo

producto viable en corto plazo. Este generador de código debe poder ajustarse a los estándares que requiera la compañía, y lo más importante que permita una abstracción de los detalles técnicos de la implementación.

1.2 Oportunidad

La Ingeniería Dirigida por Modelos MDE (Model Driven Engineering) [2], ofrece una oportunidad para hacer frente a esta problemática. Plantea la posibilidad de automatizar el proceso de transformación de un Modelo en una Aplicación, mediante la construcción de generadores de código para una plataforma específica.

El ambiente de ISML (Information Systems Modeling Language) [2], cuenta con un lenguaje textual, el cual se utiliza para modelar aplicaciones de software bajo el patrón de diseño MVC, y posee una sintaxis mucho más sencilla de escribir que C#. También proporciona una API robusta, y un ambiente de ejecución adecuado para la construcción de generadores de código fuente. Este ambiente proporciona toda la base tecnológica necesaria para la construcción de un generador para ASP.NET MVC [1].

2. DESCRIPCIÓN DEL PROYECTO

2.1 Objetivo general

Diseñar y construir un generador de código fuente que permita convertir un modelo ISML (Information Systems Modeling Language) a código en Arquitectura Microsoft .NET.

2.2 Objetivos específicos

1. Definir y Especificar los requerimientos necesarios para el generador de código.
2. Diseñar el generador de código teniendo en cuenta los requerimientos previamente especificados.
3. Implementar los artefactos de software del generador de código diseñado.
4. Probar y validar la implementación del generador de código, utilizando como fuente un Modelo ISML previamente definido de una aplicación Web MVC.

2.3 Metodología

Para el desarrollo de este proyecto, se escogió como metodología de trabajo una adaptación de la metodología DAD (Disciplined Agile Delivery) [11]. Esta metodología ofrece las siguientes ventajas: es ágil y flexible y puede adaptarse a equipos pequeños de trabajo como es el caso de este proyecto. Es compatible con el modelo de Backlogs de SCRUM [12], y facilita la generación de un producto mínimo viable en cortos periodos de tiempo, en cada iteración de su fase de construcción.

La siguiente ilustración especifica la adaptación de la metodología DAD [11] realizada para el ciclo de vida del proyecto. En esta adaptación se realizó un reemplazo de la fase de Transición de la metodología original y fue reemplazada por una fase de documentación. Se incluyó un ciclo de iteración para validar el diseño del generador, y otro para la implementación de cada uno de los requerimientos de diseño.

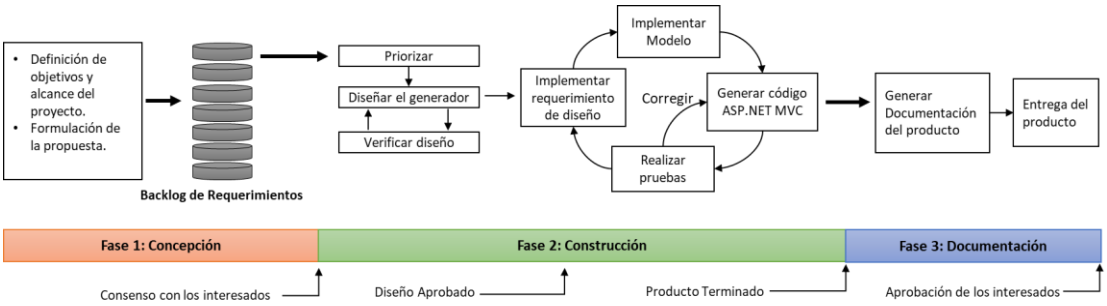


Ilustración 1: Adaptación de la Metodología DAD al Ciclo de vida del proyecto

En la Tabla 1 se especifican las 3 fases del proyecto correspondientes a la adaptación de la metodología DAD y sus respectivas etapas, también incluye el listado de resultados o artefactos generados como resultado de la ejecución de cada etapa.

Fase	**Etapa**	**Resultados esperados**
Concepción	1. Estudio de la Arquitectura de aplicaciones ASP.NET MVC	✓ Código fuente de una aplicación ASP.NET MVC, que pueda ser utilizada como referencia para el diseño del generador.
	2. Definición y Especificación de requerimientos.	✓ Documento de especificación de requerimientos del generador de código.
Construcción	3. Diseño del generador de código.	✓ Documento de Diseño del generador de código.
	4. Implementación del generador de código.	✓ Código fuente del generador de código.
	5. Pruebas y refinamiento.	✓ Documento con el reporte de pruebas y la validación de la implementación.
Documentación	6. Construcción de Manuales y Documentación.	✓ Manual de instalación y de usuario. ✓ Documento de memoria del Trabajo de Grado. ✓ Sitio web en Pegasus.

Tabla 1: Especificación de las Fases, Etapas y Entregables del proyecto.

3. MARCO TEÓRICO / ESTADO DEL ARTE

En esta sección se especifican las referencias teóricas y conceptos que se tuvieron en cuenta para el desarrollo de este proyecto. Se definen los principales conceptos de MDE (Model Driven Engineering), ISML (Information Systems Modeling Language) y ASP.NET MVC.

3.1 MDE (Model Driven Engineering)

La Ingeniería Dirigida por Modelos MDE (Model Driven Engineering) [13] es un enfoque en el cual el Modelo es el artefacto central del proceso de construcción de software, durante el ciclo de vida del proyecto. MDE Surge como una alternativa para hacer frente tanto a los problemas de complejidad de los lenguajes y plataformas de programación, como a la situación de obsolescencia en la cual pueden caer, debido a la constante evolución de las tecnologías. MDE es una alternativa para ayudarle a las empresas a proteger sus inversiones a nivel de software de estas amenazas, por medio del aumento de la abstracción y la reutilización.

MDE [14] permite definir modelos, métodos y las herramientas adecuadas para la representación de sistemas de software. Permite abstraer cualquier tipo de aplicaciones a modelos y reducir significativamente los niveles de complejidad relacionados con la especificación de una aplicación. MDE [14] Facilita la posibilidad de automatizar el proceso de transformación de un Modelo en una Aplicación.

3.1.1 MDA (Model Driven Architecture)

El concepto de MDA (Model Driven Architecture) [15] es una iniciativa que aplica el enfoque de MDE y fue propuesta por el Object Management Group (OMG) [16]. MDA propone que en la construcción de un software se debe partir de la formulación de un Modelo, y no de la escritura manual del código fuente. Ya que con este enfoque se podría aumentar el nivel de abstracción y de reutilización del software. Estos modelos deben ser: formales, precisos y con semántica bien definida. También proporciona un enfoque para especificar sistemas independientes de la plataforma, y la transformación de esta especificación en una plataforma específica. MDA cuenta con los siguientes modelos, expresados en diferentes lenguajes y niveles de abstracción:

- **CIM (Computation-Independent Model)** [15]: Es un modelo con un alto nivel de abstracción e independiente de cualquier metodología computacional, se utiliza para especificar los requerimientos.
- **PIM (Platform Independent Model)** [15]: Es un modelo independiente de cualquier plataforma tecnológica, describe el comportamiento y la estructura de la aplicación modelada, sin importar la plataforma de implementación.
- **PSM (Platform Specific Model)** [15]: Es un modelo que contiene toda la información específica de la plataforma destino.
- **IM (Implementation Model)** [15]: Este modelo representa el código fuente generado para la plataforma destino.

El ciclo de vida propuesto por MDA para el proceso de desarrollo tiene fases muy similares al ciclo de vida tradicional, pero con la ventaja de una reducción significativa del esfuerzo, en las etapas de diseño y codificación. La etapa de codificación es totalmente transparente para

el desarrollador, ya que esta se genera automáticamente. La siguiente ilustración especifica el ciclo de vida del proceso de desarrollo Utilizando MDA, describe las etapas y los modelos que intervienen en cada una de ellas.

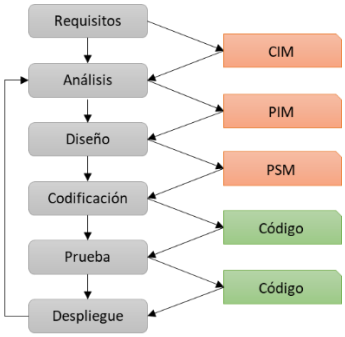


Ilustración 2: Ciclo de vida de un proceso de desarrollo utilizando MDA [15]

3.1.2 ISML (Information Systems Modeling Language)

Investigadores de la Pontificia Universidad Javeriana e ingenieros de la empresa Heinsohn Business Technology (HBT) [17], desarrollaron los proyectos Lion y Lion2 [2]. Su objetivo era lograr una alta automatización y reutilización de componentes, en el desarrollo de los proyectos de software de la empresa. Como resultado de estos proyectos se construyó el ambiente de modelado textual ISML (Information Systems Modeling Language) [2]. Este ambiente permitió la automatización del desarrollo de software aplicando ingeniería dirigida por modelos (MDE). Y tiene las siguientes características:

- Cuenta con un Lenguaje de Dominio Específico DSL [3] denominado ISML, el cual permite modelar gran parte de las funcionalidades de una aplicación empresarial, bajo el patrón Modelo-Vista-Controlador (MVC) [10].
- El ambiente cuenta con Plugins de Eclipse [5], Soporta Verificación de sintaxis y la navegación entre los elementos de un modelo.
- El ambiente permite la construcción de generadores de código fuente, ya que proporciona una API que facilita su construcción.
- Los generadores construidos sobre el ambiente ISML [2], toman como entrada un modelo expresado en lenguaje ISML, y generan como resultado el código fuente de una plataforma destino.
- ISML Permite la reutilización de componentes existentes de la empresa.

Un modelo expresado en lenguaje ISML contiene los siguientes elementos especificados en la siguiente Ilustración:

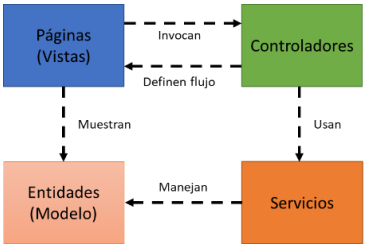


Ilustración 3: Elementos de un modelo expresado en ISML [2]

El proceso de generación de código fuente en el ambiente de ISML, inicia con la construcción de un modelo ISML. Este modelo es procesado por cualquiera de los generadores de código existentes, y el producto generado después de este proceso, es el código fuente de una plataforma destino. Actualmente ISML cuenta con generadores de código para las plataformas JEE, PHP, IONIC, y Angular.

3.1.3 AndroMDA

AndroMDA [18] es un Framework de código abierto, que sigue el paradigma de la ingeniería dirigida por modelos MDA. Permite transformar modelos, en archivos de código fuente y componentes de cualquier plataforma destino, como por ejemplo: Java, .NET, PHP, entre otras. Estos modelos pueden ser expresados en UML a partir de Herramientas Case. Incluye un conjunto de plugins (Cartuchos y librerías de traducción) listos para usar tales como: Spring, EJB, Webservices, Hibernate, Struts, JSF, Java y XSD. También incluye un Kit para que la comunidad pueda desarrollar sus propios cartuchos personalizados, y facilitar la construcción de generadores de código. Su sitio Web oficial no se encuentra actualizado, para obtener más información consulte las publicaciones de otros autores sobre este proyecto, a través de las referencias biblio-gráficas incluidas en este trabajo.

AndroMDA cuenta con las siguientes características:

- Diseño modular: Todos los bloques de construcción principales de AndroMDA [18], son conectables y pueden intercambiarse para satisfacer las necesidades que se requieran.
- Soporta las principales herramientas UML como: MagicDraw [19], Poseidon [20], Enterprise Architect [21] entre otras.
- Es compatible con el metamodelo UML 1.4 y UML 2.0. También se puede usar un metamodelo propio en MOF XMI [22] y generar código a partir de modelos basados en él.
- Valida los modelos de entrada utilizando restricciones OCL [23], que están relacionadas con las clases del metamodelo. Viene con restricciones preconfiguradas que brindan protección contra los errores de modelado más comunes. También permite agregar restricciones específicas.
- Las transformaciones Model-to-Model ayudan a elevar el nivel de abstracción. Con AndroMDA es posible escribir transformaciones específicas, ya sea en Java o en

cualquier lenguaje de transformación, como por ejemplo ATL(Atlas Transformation Language) [24].
- Permite generar cualquier tipo de salida de texto, usando plantillas para: código fuente, páginas web, scripts de base de datos, entre otras.
- Las plantillas se basan en reconocidos motores de plantillas. Actualmente soporta Velocity [25] y FreeMarker [26].

3.1.4 OpenArchitectureWare (OAW)

OpenArchitectureWare (OAW) [27] es una plataforma de código abierto flexible, para el desarrollo de software dirigido por modelos. Puede definirse como una herramienta para la creación de herramientas MDSD/MDA [28]. Cuenta con un conjunto de herramientas de apoyo como editores y navegadores de modelos, construidos sobre Eclipse. Su sitio Web oficial no se encuentra actualizado, para obtener más información consulte las publicaciones de otros autores sobre este proyecto, a través de las referencias bibliográficas incluidas en este trabajo.

Workflow Engine: En su núcleo OAW [27] cuenta con un motor de Workflow, el cual permite la definición de los flujos de trabajo de transformación, así como también un conjunto de componentes de tipo workflow preconstruidos, que pueden ser utilizados para: leer y crear instancias de modelos, verificar violaciones y restricciones, luego transformarlos en otros modelos y finalmente generar código fuente. Este motor controla con precisión el flujo de trabajo del generador, de acuerdo con lo especificado en el XML de la definición del Workflow.

Modelos: Con un instanciador adecuado, OAW puede leer cualquier modelo. Cuenta con soporte inmediato para: EMF [29], y varias herramientas UML (MagicDraw [19], Poseidon [20], Enterprise Architect [21], entre otras).

Generación de código: OAW [27] puede generar cualquier tipo de salida, utiliza un poderoso lenguaje de plantilla denominado **Xpand** [30] el cual soporta polimorfismo, programación orientada a aspectos, modelo de transformación, y otras características avanzadas, necesarias para construir generadores de código no triviales. Por ejemplo: el tipado opcional de elementos del modelo y clasificación y filtrado de conjuntos de elementos.

OAW puede leer varios modelos como parte de la ejecución de un generador. Estos modelos pueden usar diferentes sintaxis concretas (uno podría estar en UML, otro en XML y un tercero en Visio). Luego, el generador "entreteje" estos modelos para formar un sólo modelo integral que lo abarque todo. Las restricciones se pueden verificar sobre los límites del modelo; y se pueden establecer referencias entre los modelos.

Transformaciones Model-to-Model: Las transformaciones de modelo a modelo se pueden implementar utilizando el lenguaje Wombat [27], el cual es un lenguaje de transformación textual y funcional. Este lenguaje tiene una sintaxis muy concisa y poderosa y tiene la capacidad de transformar un modelo definido con el metamodelo clásico de OAW [27], en un

modelo EMF [29]. Alternativamente, puede integrar lenguajes de transformación de terceros como ATL (Atlas Transformation Language) [24].

3.1.5 Acceleo

Acceleo [31] es un generador código (Open Source), de la fundación Eclipse que permite aplicar el enfoque dirigido por modelos, para la generación automatizada de aplicaciones. Es una implementación de estándar "MOFM2T" [32] del Object Management Group (OMG) [16], para realizar transformaciones de un modelo a texto. Cuenta con un lenguaje de generación de código que utiliza un enfoque basado en plantillas. Una plantilla es un texto que contiene secciones que serán procesadas, a partir de un conjunto de elementos proporcionados por un modelo. También permite la generación de código para cualquier lenguaje textual, como por ejemplo C#, Java, Python, etc. Acceleo [31] Proporciona herramientas para la generación de código a partir de modelos basados en EMF [29].

Despliegue: Acceleo esta implementado como un plugin para el IDE de Eclipse. Aunque también es posible utilizar en modo "stand-alone" el analizador sintáctico, el motor de generación, y otros componentes críticos de Acceleo, sin tener que implementarse en Eclipse. Acceleo actualmente funciona en plataformas Linux, Mac OS y Windows.

Generación incremental: Acceleo permite la generación incremental de código, lo cual permite que se pueda generar una pieza de código, realizar modificaciones sobre este código, y finalmente regenerar el código una vez más sin perder las modificaciones anteriores. Esta característica hace que sea flexible frente a los posibles cambios que se puedan realizar a un modelo original.

Metamodelos: Acceleo también permite la generación de código, a partir de cualquier tipo de metamodelo compatible con EMF [29], como UML 1, UML 2 e incluso metamodelos personalizados DSL [3].

Herramientas: Acceleo cuenta con un editor de módulos de generación de código con resaltado de sintaxis, detección de errores en tiempo real, y opciones de refactoring. También proporciona un depurador que brinda la posibilidad de pausar una generación mediante puntos de interrupción, facilitando la verificación del estado de las variables, y la ejecución paso a paso. Esto facilita la identificación de problemas y da mayor control del proceso. El motor de Acceleo puede calcular la información de trazabilidad de todos los elementos involucrados en la generación de un archivo. Esto permite identificar de forma precisa, los elementos de los modelos de entrada que se han procesado, y el componente específico del generador involucrado en el proceso.

3.1.6 Generación de código utilizando T4 (Text Template Transformation Toolkit)

El Kit de herramientas de transformación de plantillas de texto denominado T4 [33], es una característica de Microsoft Visual Studio, que le permite a los desarrolladores automatizar la generación del código fuente de una aplicación, a través de un conjunto de herramientas que procesan plantillas de texto. T4 se utiliza en ASP.NET MVC [1] para automatizar la creación de vistas y controladores, y a nivel de Entity Framework [6] para generación de entidades.

Una plantilla de texto T4 es una mezcla de bloques de texto y lógica de control que puede generar un archivo de texto. La lógica de control se escribe en forma de fragmentos de código de programa en Visual C# o Visual Basic. El archivo generado puede ser texto de cualquier tipo, como una página web, un archivo de recursos o el código fuente de un programa en cualquier lenguaje. Existen dos tipos de plantillas de texto T4: Plantillas de texto T4 de tiempo de ejecución o "pre-procesadas" y Las plantillas de texto T4 en tiempo de diseño.

Plantillas de texto T4 en tiempo de diseño [33]: Permiten generar el código fuente de un programa y sus archivos de texto complementarios. Estas plantillas son escritas con una sintaxis especial, para de forma dinámica varíen su resultado final de acuerdo con la información consignada en un **Modelo**. Un modelo puede ser un archivo o una fuente de datos que contiene la información clave, acerca de los requisitos de la aplicación.

Una plantilla de texto contiene una combinación que integra: el texto que se desea generar y un código de programación que genera partes parciales variables dentro del texto. El código de programa y el texto permiten repetir u omitir condicionalmente partes del texto generado. Las plantillas T4 cuentan con las siguientes características:

- Permiten generar aspectos variables de una aplicación y codificarlos en uno o más modelos origen.
- Permiten utilizar más de un modelo en un proyecto. Por ejemplo, se puede definir un modelo para la navegación entre páginas web y otro modelo independiente para el diseño de las páginas.
- Cuenta con herramientas para depurar y probar el código generado.
- Permiten distribuir bloques grandes de código en ensamblados independientes.
- Permiten colocar métodos en una clase abstracta que puede heredada por la plantilla.

3.2 ASP.NET MVC

ASP.NET MVC [1] es un Framework para la construcción de aplicaciones Web, que aplica el modelo arquitectónico MVC (Modelo Vista Controlador) [10] al Framework de ASP.NET [34]. MVC separa una aplicación en los siguientes tres componentes principales:

- **Modelos**: Son los componentes que implementan la lógica del dominio de datos en una aplicación. Generalmente los objetos del modelo recuperan y almacenan el esta-

do del modelo en una base de datos. Por ejemplo, un objeto "Cliente" podría recuperar información de una base de datos, trabajar con ella y, a continuación, escribir la información actualizada en una tabla "Clientes" de una base de datos de SQL Server [35]. Los modelos son expresados en archivos de clase C#, y generalmente cuentan con una capa de acceso a datos que utiliza una herramienta como Entity Framework [6] o NHibernate [36].

- **Vistas**: Son los componentes representan la interfaz de usuario en una aplicación. una interfaz de usuario se crea a partir de los datos de un modelo. Un ejemplo de este componente sería una vista para la creación de un registro para la de una tabla "Clientes", la cual se muestra a través de: cuadros para el ingreso de texto, listas desplegables, casillas de verificación, y otros controles de ASP.NET [34] necesarios para cargar los datos de un objeto de tipo "Cliente". Las vistas son expresadas en formularios escritos en código HTML y sintaxis Razor [9].
- **Controladores**: Son los componentes que controlan la interacción del usuario. Los controladores trabajan con el modelo y seleccionan la vista para representar la interfaz de usuario. En ASP.NET MVC [1] El controlador administra y responde a los datos proporcionados por el usuario a través de su interacción. Los controladores son expresados en archivos de clase en C#.

ASP.NET MVC facilita la creación de aplicaciones, mediante la separación de elementos tales como: la lógica de entrada, la lógica de negocios y la lógica de interfaz de usuario, proporcionando un acoplamiento moderado. Esta separación ayuda a administrar y reducir la complejidad de una aplicación. Al no existir una dependencia tan fuerte entre estos componentes, se pueden dividir responsabilidades entre el equipo de desarrollo. Mientras un ingeniero de desarrollo trabaja en la vista, otro puede ocuparse de la lógica del controlador, y otro puede estar trabajando en la lógica de negocios del modelo, de forma paralela.

3.2.1 Entity Framework

Entity Framework [6] es un conjunto de tecnologías de ADO.NET [37] que permiten el desarrollo de aplicaciones de software orientadas a datos. Permite a los desarrolladores trabajar con datos en forma de propiedades y objetos de dominio específico, como por ejemplo objetos de tipo "Cliente" y "Venta", sin tener que preocuparse por las tablas y columnas adyacentes de la base de datos, en donde se encuentran almacenados estos datos. Entity Framework [6] les permite a los desarrolladores trabajar con mayor nivel de abstracción, todo el manejo de datos. También permite crear aplicaciones con acceso a datos con un número reducido de líneas de código comparado con aplicaciones tradicionales.

Entity Framework [6] es compatible con el modelo tradicional de creación aplicaciones, en el cual se divide la aplicación en tres partes: un modelo de dominio, un modelo lógico y un modelo físico. El modelo de dominio define las entidades y relaciones del sistema que se está modelando. El modelo lógico de una base de datos relacional normaliza las entidades y relaciones en tablas con restricciones de claves externas. El modelo físico abarca las capacidades de un motor de datos determinado, especificando los detalles del almacenamiento en forma de particiones e índices.

Entity Framework permite a los programadores consultar las entidades y relaciones especificadas en el modelo de dominio, también denominado modelo conceptual. También traduce las operaciones de consulta a comandos específicos del origen de datos. Esto libera a las aplicaciones de las dependencias codificadas de forma rígida, en un origen de datos determinado. También provee el enfoque Code First [7], este enfoque permite que a partir del código especificado en un modelo conceptual, se pueda crear una base de datos con todas sus tablas y relaciones, con la ayuda de una convención, los DataAnnotations y el Fluent API [8].

3.2.2 Fluent API

Entity Framework Fluent API [8] provee un mecanismo para especificar las configuraciones del modelo, que no alcanzan a ser soportadas por los DataAnnotations [38]. Fluent API Permite configurar los siguientes aspectos de un modelo:

- **Configuración a nivel del modelo**: configura el esquema predeterminado, las entidades incluidas y también las que serán excluidas en el mapeo.
- **Configuración de a nivel entidad**: configura el mapeo de entidades a tablas y relaciones. También especifica: Llaves primarias, Índices, nombre de la tabla. Relaciones uno-a-uno, uno-a-muchos, muchos-a-muchos, etc.
- **Configuración de propiedades**: Permite configurar propiedades a las columnas mapeadas, como por ejemplo: nombre de la columna, llave foránea, tipo de dato, definir si el campo es NULL, entre otros.

3.2.3 SQL Server 2016 Express LocalDB

Microsoft SQL Server 2016 Express LocalDB [35] es una característica de SQL Server Express [39] dirigida a desarrolladores de software. LocalDB contiene un conjunto mínimo de archivos necesarios para iniciar el Motor de base de datos de SQL Server. permite iniciar una conexión usando una cadena de conexión especial. Al conectarse, la infraestructura necesaria de SQL Server se crea e inicia automáticamente, permitiendo que la aplicación use la base de datos sin tareas complejas de configuración. Permite escribir y probar el código de Transact-SQL sin tener que administrar una instancia de servidor completo de SQL Server. Es sobre esta base de datos pequeña que el enfoque Code First de Entity Framework, crea las entidades y relaciones del modelo, con las configuraciones especificadas mediante el Fluent API del Entity Framework.

3.2.4 Patrón de diseño Repository en ASP.NET MVC

El Patrón Repository [40] facilita una división entre la interfaz de usuario, la lógica de negocio y los componentes de acceso a datos de la aplicación, en diferentes capas fáciles de mantener y probar. Le brinda a la aplicación un punto centralizado, para las operaciones de acceso a datos, y es a través de este repositorio que los componentes de negocio pueden realizar las operaciones de persistencia sobre los datos, que pueden ser utilizadas por cualquier entidad. Este repositorio puede contener un conjunto de operaciones CRUD genéricas, o contener operaciones más complejas. La siguiente ilustración especifica una implementación sencilla del Patrón Repository en una aplicación ASP.NET MVC [1].

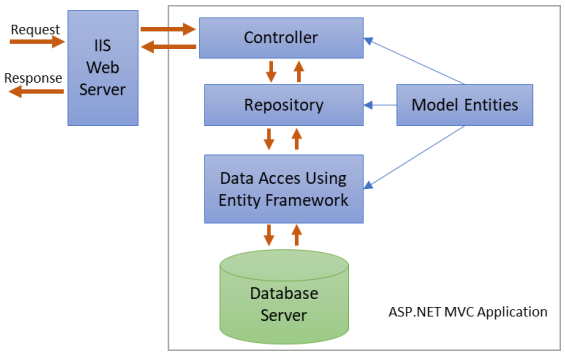


Ilustración 4: Patrón Repository en ASP.NET MVC con Entity Framework

El controlador se encarga de canalizar y recibir las solicitudes del servidor Web IIS [41], para realizar el llamado al repositorio utilizando el modelo. El repositorio interactúa el componente de acceso a datos que proporciona el Entity Framework [6]. Este componente interactúa directamente con la base de datos. El modelo de entidades es utilizado tanto por el controlador, el repositorio y el componente de acceso a datos de Entity Framework.

4. TRABAJOS RELACIONADOS

En esta sección se analizan un conjunto de generadores de código fuente para ASP.NET MVC, y se realiza una comparativa entre sus características y ventajas frente al generador de código DotNetGenerator realizado en este trabajo.

4.1 Generadores de Código para Visual Studio

Después de realizar una investigación sobre otros generadores de código comerciales y de código abierto disponibles, se evidenció que la mayoría tienen los siguientes factores en común. Su integración con el IDE de Visual Studio y el uso de un modelo de entidades y relaciones, que generalmente representan una base de datos. A partir de estos modelos estos generadores construyen el código fuente de la aplicación. Otro factor común, es que estos generadores se basan en un modelo diseñado solo para el generador, teniendo como única plataforma destino la arquitectura .NET. En este sentido los modelos en ISML, son más prácticos ya que pueden ser utilizados por cualquiera de los generadores disponibles, abriendo la posibilidad de generar código para múltiples plataformas. A continuación se describen dos generadores de código para la plataforma Microsoft .NET:

4.1.1 Xomega

Xomega [42] es una solución que facilita el desarrollo Dirigido por modelos, y la generación de código fuente para aplicaciones .NET N-Tier [43]. La plataforma Xomega consta de los siguientes componentes:

- **Tecnología de modelado de objetos XML**: Esta tecnología permite la generación extensible de artefactos como código, scripts de bases de datos, documentación, etc. utilizando la tecnología XSLT estándar.
- **Plugin Xomega.Net para Visual Studio** [4]: Este Plugin proporciona un entorno integrado para explorar y editar modelos de objetos Xomega, y también para configurar y ejecutar transformaciones de modelos directamente desde Visual Studio, o como una tarea de MSBuild [44].
- **Frameworks de código abierto Xomega y XomegaJS**: estos Frameworks permiten construir robustas aplicaciones .NET utilizando las últimas tecnologías de Microsoft, como por ejemplo: ASP.NET [1], HTML5, TypeScript, WPF [45], Web API [46], WCF [47] y Entity Framework.
- **Conjunto de generadores extensibles**: estos generadores permiten generar diversos artefactos a partir de un modelo de objetos propio de Xomega [42], para todas las capas de una aplicación, incluyendo la base de datos, los objetos de negocio, la capa de servicios y la capa de presentación.

Una de las principales fortalezas de esta plataforma para la generación de código fuente es su integración natural con Visual Studio, y su facilidad gráfica para la generación de un modelo de entidades. El modelado en esta plataforma se hace principalmente en XML.

Si se realiza una comparación entre Xomega, y DotNetGenerator, se debe tener en cuenta que DotNetGenerator tiene como insumo un Modelo expresado en ISML [2]. Este lenguaje mucho más sencillo que C#, y permite modelar en unas pocas líneas de código componentes como Entidades, Páginas, Controladores y Servicios. En Xomega, se deben modelar estos componentes en lenguaje XML. Además los modelos expresados en ISML pueden ser convertidos en código fuente de diferentes plataformas destino, ya que cuenta con generadores de código para Java, PHP, Ionic. Mientras que el modelo expresado en Xomega a través de Visual Studio, está especializado en Aplicaciones .NET.

4.1.2 CodeTrigger

CodeTrigger [48], es un generador de código fuente comercial, que está disponible como un Complemento para el IDE de Visual Studio [4]. Está diseñado para trabajar con los motores de bases de datos: SQL Server, Oracle y MySQL. Cuenta con un sistema que permite importar esquemas de base de datos predefinidos y a partir de estos esquemas, modela las relaciones en una aplicación para la arquitectura .NET. la aplicación puede ser modelada para ASP.NET MVC [1], WCF [47], WPF [45] y WinForms [49].

CodeTrigger [48] tiene como objetivo simplificar y agilizar el proceso de desarrollo de software, mediante el establecimiento de una plantilla no intrusiva para su equipo de desarrollo. A diferencia de otros generadores de código, CodeTrigger es en gran medida independiente del diseño. El equipo de desarrollo no está obligado a seguir ninguna filosofía particular de diseño, ni está vinculado a capas complejas de código. Este generador permite generar aplicaciones con una capa de negocio, una capa de datos, una capa de presentación, y una capa de servicios.

Una de las principales fortalezas de este generador es que aparte de tener una integración con Visual Studio, también provee un plugin para Enterprise Architect [21]. El objetivo de este Plugin es poder generar el código fuente de una aplicación para .NET, a partir de un modelo entidad Relación cargado en Enterprise Architect.

CodeTrigger se diferencia del generador DotNetGenerator, en el enfoque que aplica para el Modelo, ya que este utiliza como Modelo un esquema de una base de datos, el cual está integrado principalmente por Entidades y relaciones. DotNetGenerator utiliza como insumo un modelo mucho más completo construido en ISML. Este modelo no solo está construido a partir de un modelo de entidades y relaciones, este también incluye otros componentes de la Arquitectura MVC [10], la cual incluye Controladores, Páginas y Servicios. CodeTrigger puede ser muy útil cuando se requiere generar aplicaciones con sus operaciones CRUD muy rápidamente, y agregar personalizaciones a las capas de negocio través de herramientas gráficas.

5. DESARROLLO DEL PROYECTO

En esta sección se encuentra la descripción general del sistema, los requerimientos funcionales y no funcionales, y el diseño detallado del sistema.

5.1 Descripción general del Sistema

El Software generador de código DotNetGenerator es el resultado final de un proceso de ingeniería de software aplicado a un proyecto de trabajo de grado, sobre Ingeniería Dirigida por Modelos MDE (Model Driven Engineering) [2]. El cual tiene como objetivo la generación de código fuente a partir de un modelo ISML.

DotNetGenerator es un componente de software que le permitirá a Ingenieros de Desarrollo y Arquitectos de Software, generar el código fuente de una Aplicación Web ASP.NET MVC [1] a partir de un modelo previamente definido en un lenguaje de dominio específico (DSL) [3] denominado ISML (Information Systems Modeling Language). Este código fuente generado podrá ser abierto utilizando la herramienta de desarrollo Visual Studio Community 2017 [4], desde donde podrá ser compilado y ejecutado.

El código fuente generado contendrá los componentes más comunes de una Aplicación Web ASP.NET MVC [1], como por ejemplo: Modelos, Vistas, Controladores y Servicios. Esto le permitirá al ingeniero de desarrollo abstraerse de los detalles de implementación de estos aspectos de una aplicación. DotNetGenerator forma parte de los Plugins de Eclipse utilizados para la generación de código fuente, del ambiente de modelado MDE, denominado ISML (Information Systems Modeling Language) [2]. El código fuente está disponible en el repositorio de GitHub creado para el proyecto DotNetGenerator [50].

5.1.1 Limitaciones del Sistema

El usuario puede agregar cambios al Modelo ISML original, después de haber realizado una generación de código anterior. Esto provocará la reescritura total del código generado. Se debe tener en cuenta que si el código generado fue manipulado y sufrió cambios después de la última generación, estos cambios se perderán. El sistema aún no cuenta con un mecanismo de generación incremental [51], que permita mezclar de forma adecuada los nuevos cambios modelados, con el código fuente ya modificado. Es por esta razón que no se recomienda ejecutar nuevamente la generación de código, si el usuario ha modificado el código.

5.2 Requerimientos del Sistema

En esta sección se especifican los requerimientos funcionales y no funcionales, para el sistema DotNetGenerator. En la especificación de los requerimientos no funcionales se tuvieron en cuenta requerimientos de Mantenibilidad y Portabilidad.

5.2.1 Requerimientos Funcionales

En esta sección se especifican los requerimientos funcionales del sistema en la Tabla 2:

ID	Autor	Descripción
RF-01	Julián Sánchez	**Estructura de Directorios**: El Sistema debe generar una estructura de directorios, y archivos correspondientes a la de una aplicación ASP.NET MVC 5 [51] con soporte para Entity Framework 6 [6], y con facilidad para aplicar el enfoque Code First [7].
RF-02	Julián Sánchez	La solución generada debe contar con los siguientes 4 proyectos: • **AplicacionWeb**: Proyecto que contiene los controladores, vistas y demás componentes de una de la aplicación Web. • **AplicacionWeb.DAL**: Proyecto que contiene todos las clases y componentes de acceso a datos de la aplicación. • **AplicacionWeb.Models**: Proyecto que contiene las entidades del modelo de la aplicación. • **AplicacionWeb.Services**: Proyecto que contiene las interfaces y las implementaciones de los servicios de la aplicación.
RF-03	Julián Sánchez	**Modelos**: El Sistema debe generar dinámicamente un archivo de clase en C# con extensión "*.cs", por cada Modelo definido en el ISML. Los modelos deben ser generados en la raíz del proyecto **AplicacionWeb.Models.**
RF-04	Julián Sánchez	**Vistas**: El Sistema debe generar dinámicamente un archivo de extensión "*.cshtml" y con sintaxis Razor [9], por cada page definido en el ISML. Los directorios que contienen los page en ISML deben ser generados en la carpeta **Views** del proyecto **AplicacionWeb.**

RF-05	Julián Sánchez	**Controladores**: El Sistema debe generar dinámicamente un archivo de clase en C# con extensión "*.cs", por cada controlador definido en el ISML y deben cumplir con las siguientes especificaciones: • Los controladores podrán utilizar inyección de dependencias, en el caso en que sea requerido. • Los controladores deben ser generados en el directorio **Controllers** del proyecto **AplicacionWeb.**
RF-06	Julián Sánchez	**Servicios**: El Sistema debe generar dinámicamente dos archivos de clase en C# con extensión "*.cs", por cada servicio definido en el ISML. Uno corresponderá a la interfaz y el otro será la implementación del servicio. Los servicios deben cumplir las siguientes especificaciones: Archivo de Interfaz de Servicio: • Cada interfaz de servicio será nombrado anteponiendo la letra "I" de la siguiente manera: I{nombre del servicio}. • El archivo de interfaz debe ser generado en el directorio **Interfaces** del proyecto **AplicacionWeb.Services** Archivo de implementación del servicio: • Cada implementación del servicio tendrá el mismo nombre del servicio. • El archivo de implementación debe ser generado en el directorio: **Implementation** del proyecto **AplicacionWeb.Services**
RF-07	Julián Sánchez	**Archivo de proyecto**: El sistema debe generar dinámicamente un archivo de proyecto con extensión ".csproj", para cada uno de los proyectos de la solución.
RF-08	Julián Sánchez	**Inyección de dependencias**: El sistema debe generar dinámicamente un archivo de clase de C# denominado "SimpleInjectorInitializer.cs" [52], el cual registre la inyección de dependencias que será utilizada en los constructores de cada controlador, y debe ser generado en el directorio **App_Start** del proyecto **AplicacionWeb.**
RF-09	Julián Sánchez	**Entity Framework**: El sistema debe generar una aplicación que permita aplicar el enfoque Code First [7] de ASP.NET MVC. Para lograr esto el sistema debe generar dinámicamente un archivo de clase de C# denominado "ApplicationDbContext.cs", el cual deriva de la clase DbContext [53] con las siguientes especificaciones: • Incluir la declaración de objetos de tipo DbSet [54], por cada entidad especificada en el modelo. La clase DbSet representa un conjunto de entidades que se puede utilizar para las operaciones de creación, lectura, actualización y eliminación. • Incluir la Configuración y asignación de relaciones entre las entidades del modelo, utilizando el Entity Framework Fluent API [8], Específicamente en el método "OnModelCreating" [55].

		• El archivo debe ser generado la raíz del proyecto **Aplicacion-Web.DAL**
RF-10	Julián Sánchez	**Persistencia**: El sistema debe contar con un mecanismo de persistencia, el cual permita abstraer las operaciones sobre la base de datos soportadas por ISML. Estas operaciones pueden ser invocadas desde los controladores. Para esto el sistema incorporará a la aplicación generada el patrón de diseño Repository, mediante la creación de una interfaz con las operaciones soportadas y su respectiva implementación. La implementación debe hacer uso del objeto ApplicationDbContext, descrito en el requerimiento RF-09. Las operaciones implementadas deben ser las siguientes: create, save, remove, find, findAll, isPersistent, count.
RF-11	Julián Sánchez	El sistema debe generar dinámicamente un archivo con sintaxis Razor [9] denominado "_Layout.cshtml", con las siguientes especificaciones: • Debe incluir la Definición del formulario maestro de la aplicación. • Debe incluir una Barra de menú ubicada en el encabezado de la página web, y cuyas opciones corresponden a las acciones especificadas como default en cada controlador ISML.

Tabla 2: Requerimientos Funcionales del sistema

5.2.2 Requerimientos No Funcionales

En esta sección se especifican los requerimientos no funcionales en la Tabla 3 y Tabla 4.

5.2.2.1 Mantenibilidad

ID	Tipo	Autor	Descripción
RNF-12	Mantenibilidad	Julián Sánchez	El sistema debe estar dividido en paquetes, que faciliten su mantenimiento.
RNF-13	Mantenibilidad	Julián Sánchez	El sistema debe contar plantillas definidas en clases separadas.
RNF-14	Mantenibilidad	Julián Sánchez	El sistema debe contar con generadores definidos en clases separadas.
RNF-15	Mantenibilidad	Julián Sánchez	El sistema debe agrupar las funcionalidades más comunes en una o más clases utilitarias, de tal forma que el sistema pueda reutilizar funcionalidades.
RNF-16	Mantenibilidad	Julián Sánchez	El sistema debe contar con una clase maestra que cuente con referencias a cada uno de sus generadores.

Tabla 3: Requerimientos No Funcionales de Mantenibilidad

5.2.2.2 Portabilidad

ID	Tipo	Autor	Descripción
RNF-17	Portabilidad	Julián Sánchez	El sistema debe ser compatible y poder ser ejecutado en los sistemas operativos: Linux, Windows y MacOS.
RNF-18	Portabilidad	Julián Sánchez	El sistema debe generar como resultado un código fuente con la estructura de un proyecto con la arquitectura de una aplicación ASP.NET MVC 5 [51] compatible con Entity Framework 6 [6].
RNF-19	Portabilidad	Julián Sánchez	El sistema debe generar como resultado un proyecto que pueda ser abierto, compilado y ejecutado por el ambiente de desarrollo: Visual Studio Community 2017 y ediciones comerciales de Visual Studio.
RNF-20	Portabilidad	Julián Sánchez	El sistema debe generar como resultado un proyecto ASP.NET MVC 5, que pueda desplegado en los siguientes navegadores: Microsoft Edge, Google Chrome y Mozilla Firefox.

Tabla 4: Requerimientos No Funcionales de Portabilidad

5.3 Diseño del Generador DotNetGenerator

Esta sección contiene una descripción general de la arquitectura del sistema y especifica la vista de procesos, la vista física, la vista lógica, y la vista de despliegue de una aplicación Web ASP.NET MVC generada por el sistema. También contiene un diseño detallado de cada uno de los paquetes y componentes que integran el sistema, sus relaciones y dependencias.

5.3.1 Vista de Procesos del Sistema

En esta sección se especifica la vista de procesos del sistema. El siguiente diagrama de procesos BPMN describe el proceso del sistema y cada una de las actividades que lo conforman, así como también las interacciones del usuario con el sistema:

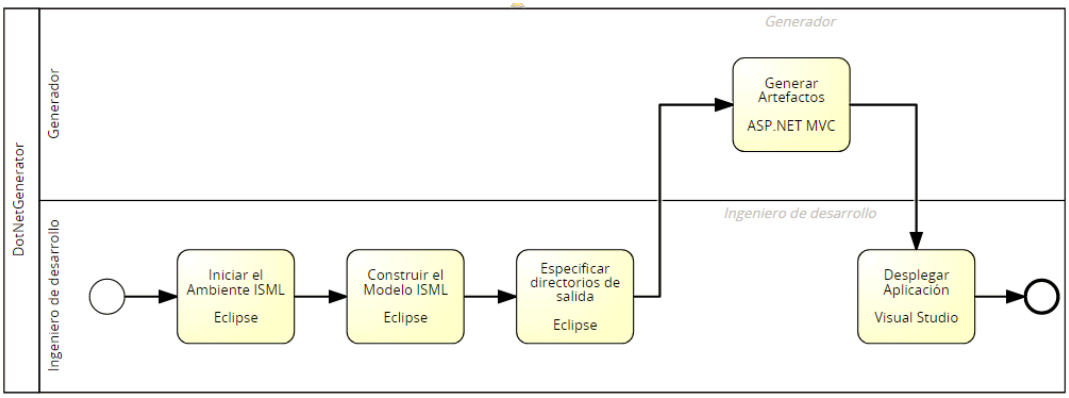


Ilustración 5: Vista de procesos del sistema

A continuación, se especifican cada una de las actividades descritas en el diagrama BPMN anterior:

Inicializar el ambiente ISML en Eclipse: En esta actividad el Ingeniero de Desarrollo podrá inicializar el ambiente de trabajo MDE denominado ISML, utilizando el plugin de Eclipse proporcionado por el sistema.

Construir el Modelo ISML: En esta actividad el Ingeniero de Desarrollo podrá construir el Modelo ISML de la aplicación en un proyecto de eclipse, utilizando el editor textual proporcionado por el sistema. El usuario también puede utilizar un modelo existente, los modelos quedarán guardados como archivos de texto con extensión *.isml. En este Modelo el usuario puede expresar los siguientes componentes de una aplicación: Modelos (Entity), Vistas (Pages), Controladores (Controllers), Servicios -> (Services).

Especificar directorios de salida: En esta actividad el Ingeniero de Desarrollo podrá configurar los directorios destino en donde serán generados los archivos de código fuente producto de la generación. Para esto el ingeniero modifica el archivo de texto "**generation.conf.json**", estableciendo las rutas de salida para: Entidades, Controladores, Páginas, Servicios, y los demás componentes de salida.

Generar Artefactos ASP.NET MVC: En esta actividad el sistema genera de forma automática en los directorios de salida, la estructura de directorios y archivos de código fuente de una Aplicación ASP.NET MVC 5 con Entity Framework 6 [6] aplicando el enfoque Code First [7], lista para ser utilizada. Esta aplicación generada contiene la lógica especificada en las Entidades, Controladores, Páginas y Servicios, definidos en Modelo ISML especificado.

Desplegar Aplicación en Visual Studio: En esta actividad el Ingeniero de Desarrollo carga el proyecto generado, abriendo el archivo de solución que tiene la extensión "*. sln", lo cual cargará el proyecto en el ambiente de desarrollo de Visual Studio Community 2017 [4], en donde el usuario podrá compilar la aplicación y ejecutarla en un Navegador Web. El Ingeniero de Desarrollo podrá extender o modificar las funcionalidades de la aplicación generada.

5.3.2 Arquitectura del Sistema

La arquitectura del sistema está compuesta por los siguientes tres componentes principales tal como se especifican en la ilustración 1. El primero es un modelo ISML, este modelo es construido por el Ingeniero de Desarrollo en un proyecto de Eclipse, en archivos de texto de extensión *.isml, en este modelo se especifican: Entidades, Controladores, Páginas y Servicios, así como también las relaciones y el flujo de información entre cada uno de ellos.

El Segundo componente es el generador de Código DotNetGenerator el cual es instalado como un plugin de Eclipse, este generador procesa el Modelo ISML y a través de sus generadores internos, y como resultado de este proceso se genera el código fuente de un proyecto de

una aplicación Web ASP.NET MVC. Este proyecto contiene toda la lógica especificada en el Modelo ISML, pero expresada en un conjunto de directorios y archivos propias de un proyecto ASP.NET MVC. Este proyecto puede ser abierto desde el IDE Visual Studio, desde donde podrá compilar y ejecutar la aplicación para ser visualizada en un navegador web.

La siguiente Ilustración 6, especifica de manera detallada cada uno de los componentes internos de la arquitectura de la solución.

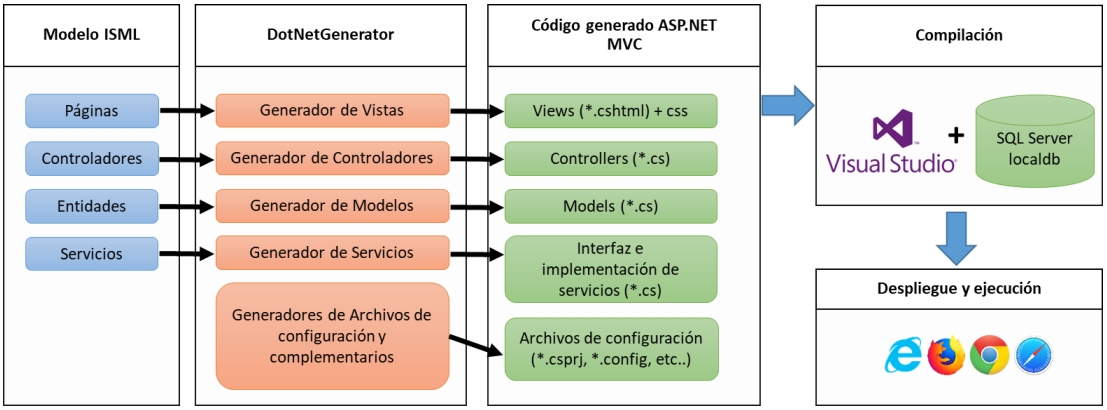


Ilustración 6: Modelo detallado para la generación de código con DotNetGenerator

5.3.2.1 Modelo ISML (Information Systems Modeling Language)

Este modelo es construido en un proyecto de Eclipse, en el modelo se especifican los principales componentes de una Aplicación que implementa el patrón de diseño Modelo Vista controlador MVC [10]. Los modelos se especifican en un lenguaje textual denominado ISML (Information Systems Modeling Language), el cual es un lenguaje mucho más sencillo que Java o C#, y se almacenan en archivos de extensión *.isml dentro de un proyecto de eclipse.

El proyecto de eclipse en donde se estructura el modelo contiene los siguientes elementos:

- Una estructura de directorios para contener los elementos de tipo: Entity, Controller, Page y Services, en archivos de extensión *.isml.
- Un archivo denominado **generation.conf.json**, el cual contiene las rutas de los directorios en donde se van a generar los archivos producto de la generación. Estas rutas apuntan a directorios del Arquetipo definido, el cual contiene el esqueleto básico de la aplicación, para posteriormente ser llenado con los demás elementos generados.

5.3.2.2 Generador DotNetGenerator

DotNetGenerator es el componente fundamental del sistema, este está conformado por un conjunto de generadores de código fuente, para realizar la generación de cada uno de los archivos y directorios de un proyecto de una Aplicación Web ASP.NET MVC. Contiene 12 generadores de propósito específico, quienes actúan como unidades de trabajo para cada pro-

cesan las entidades, controladores, páginas y servicios, modeladas en el ISML, y convertirlas en su equivalente en la plataforma destino. Para ver el detalle de cada generador consulte la sección 5.3.5.2.

5.3.2.3 Código fuente Generado en ASP.NET MVC

Este componente es el resultado final de la generación de código producto de los generadores del DotNetGenerator. Este código fuente contiene la estructura de directorios y archivos de una aplicación con las características técnicas definidas en la Tabla 5:

Tipo de Aplicación	Aplicación Web ASP.NET MVC 5 [51]
Lenguaje de Programación	• Lenguaje C#. • Sintaxis Razor [9] en el código HTML de las vistas.
Persistencia	• Incorporación del Patrón de Diseño Repository [40] para la capa de acceso a datos. • Consumo de datos mediante Entity Framework 6 [6] con el enfoque Code First [7].
Base de Datos	SQL Server 2016 Express LocalDB [35].
IDE Compatible	Visual Studio 2017 Community, o cualquier edición comercial de Visual Studio Comercial superior o igual a 2017.

Tabla 5: Características técnicas de la aplicación Web generada

5.3.2.4 Compilación del Código Fuente Generado

El usuario puede abrir la aplicación generada, a partir del archivo de solución de extensión *.sln, ubicado en el directorio raíz del código fuente generado. El proyecto de la aplicación Web se cargará en el IDE Visual Studio 2017 Community [4], en donde el usuario podrá compilar este código fuente, así como también agregar o modificar las clases C# o demás archivos de la aplicación. Adicionalmente el visual Studio permitirá generar la base de datos SQL Server LocalDB [35], a partir del código en C# del modelo especificado. Visual Studio fabricará las entidades y relaciones de la base de datos, a partir de las especificaciones detectadas tanto en el modelo Como en el archivo "ApplicationDbContext.cs".

5.3.2.5 Despliegue y Ejecución del Código Fuente Generado

El despliegue y ejecución de la aplicación se podrá hacer en cualquiera de los navegadores web como, por ejemplo: Google Chrome, Mozilla Firefox, Microsoft Edge, y será el IDE de Visual Studio el que invocará el navegador web seleccionado por el usuario, después de compilar y ejecutar la aplicación, en Modo depuración o cualquier otro modo configurado por el usuario. Visual Studio utiliza la herramienta IIS Express [41], para hacer la publicación del

directorio virtual de la aplicación de forma automática, para que esta pueda ser consultada por un navegador Web a través de la dirección http://localhost:57832/, la cual redirigirá a la página Home especificada en la aplicación. El puerto por defecto puede ser modificado en el archivo de proyecto de la aplicación de extensión *.csproj.

5.3.3 Vista Lógica del Sistema

El generador de código DotNetGenerator está integrado por un componente interno, el cual tiene interfaces con cinco componentes externos, estos componentes están especificados en la Ilustración 7:

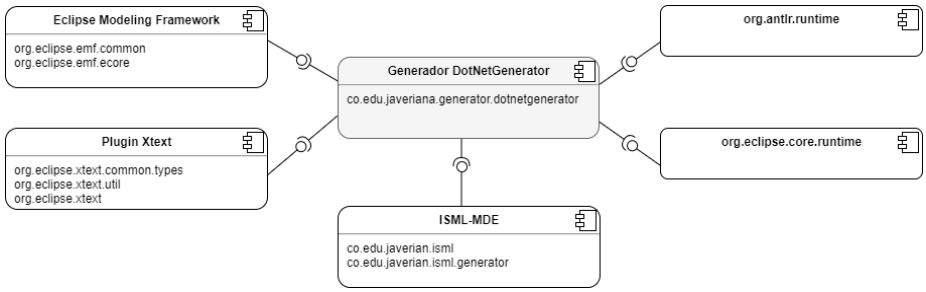


Ilustración 7: Vista lógica del sistema

5.3.3.1 Componentes Internos

En esta sección se especifica el componente interno del sistema:

co.edu.javeriana.dotnetgenerator: es el componente principal del sistema. Tiene como objetivo procesar un modelo especificado en ISML, y generar como resultado la estructura de directorios y archivos de un proyecto de una aplicación Web ASP.NET, que utiliza Entity Framework. Contiene un conjunto de generadores de código que se encargan de procesar el modelo con el apoyo de unas plantillas específicas.

5.3.3.2 Componente Externos

En esta sección se especifican los componentes externos con los cuales tiene interfaces el sistema. La Tabla 6 contiene el listado de componentes externos del sistema

Nombre	Descripción
Eclipse Modeling Framework (EMF) [29]	Es un Framework que facilita el trabajo con Modelos y Metamodelos en Eclipse. Proporciona herramientas y soporte de tiempo de ejecución para producir un conjunto de clases de Java para el modelo. Es sobre este componente que se encuentra construido el ISML (Information Systems Modeling Language) [2]. Sus principales paquetes son org.eclipse.emf.common, el cual contiene funciones comunes para el

	manejo de modelos y metamodelos y org.eclipse.emf.ecore es cuál es la API para el metamodelo denominado ECORE.
Plugin Xtext [56]	Es un Frameworks especializado para el desarrollo de lenguajes de programación y lenguajes de dominio específico DSL [3], como es el caso de ISML (Information Systems Modeling Language. Es sobre Xtext que se encuentra soportado el lenguaje ISML. los principales paquetes utilizados por ISML son: El paquete org.eclipse.xtext.util para el procesamiento de texto, El paquete org.eclipse.xtext.common.types para el tipado sobre Xtext, y el paquete org.eclipse.xtext como componente raíz para el procesamiento del lenguaje textual.
Plugin ISML-MDE [2].	Es el Plugin, necesario para poder hacer uso del ISML (Information Systems Modeling Language) [2]. Permite realizar operaciones con un modelo escrito en ISML para poder realizar su respectivo mapeo sobre una plataforma específica, en este caso ASP.NET. Tiene como principales paquetes: co.edu.javeriana.isml.generator el cual contiene una API de ISML, que brinda soporte para la generación de código en otros lenguajes en este caso C#, y el paquete raíz de ISML co.edu.javeriana.isml el cual contiene los constructos del lenguaje.
org.antlr.runtime	Es un componente de software que permite realizar procesos de análisis léxico y sintáctico de un proyecto ISML. Utiliza ANTLR [57] el cual es herramienta para el reconocimiento de lenguajes, es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto estructurado.
org.eclipse.core.runtime	Es un componente de software que permite la ejecución de otra instancia del IDE de Eclipse sobre la actual en tiempo de ejecución. Para que sobre esta nueva instancia de Eclipse que se puedan crear los proyectos ISML que serán procesados por el generador de código DotNetGenerator.

Tabla 6: Listado de componentes externos del sistema

5.3.4 Vista Física y Despliegue del Sistema

El sistema puede ser instalado y ejecutado en un mismo nodo, en el cual se podrá construir el modelo ISML de una aplicación, generar la aplicación Web utilizando el generador de código DotNetGenerator a partir del modelo, y Compilar y ejecutar la aplicación. No se requieren de nodos adicionales para satisfacer las necesidades del sistema. Este nodo debe contar con las siguientes aplicaciones de Software: el IDE Eclipse [5], el IDE de Visual Studio [4], la aplicación de IIS Express [41] y un Navegador Web. La ilustración 8 contiene la Vista Física y despliegue del sistema.

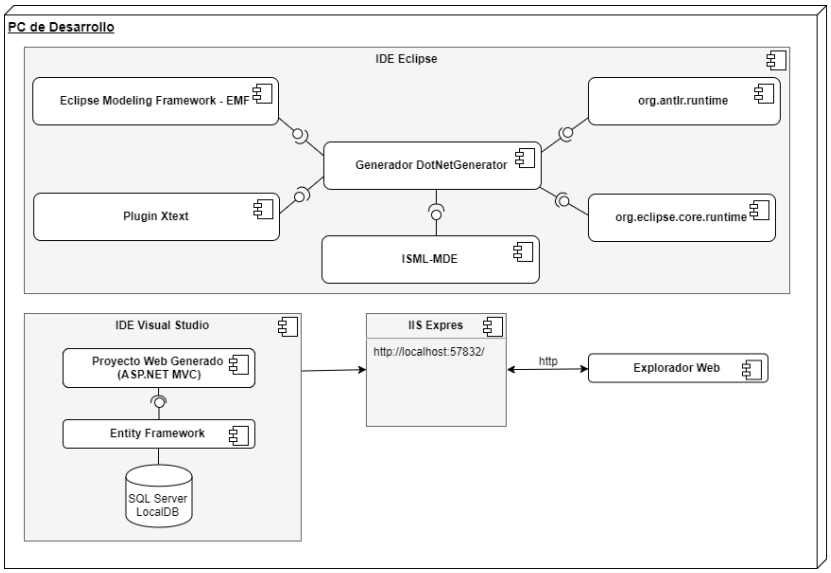


Ilustración 8: Vista física y despliegue de sistema

IDE Eclipse: El IDE Eclipse es en donde se realizará el proceso de Creación del Modelo ISML, y también el proceso de la generación del código fuente de una aplicación Web a partir de este modelo, mediante el generador de código denominado "DotNetGenerator". El IDE de Eclipse debe tener instalados los siguientes componentes de Software: ISML-MDE Plugin, Plugin XText [56], Eclipse Modeling Framework (EMF) [29], org.antlr.runtime, org.eclipse.core.runtime, por supuesto el DotNetGenerator.

IDE Visual Studio: El IDE Visual Studio 2017 Community [4], será utilizado para abrir el código fuente generado, el usuario podrá compilar y ejecutar la aplicación, así como también hacer modificaciones al código generado y ajustarlo a sus necesidades. La aplicación Web generada utiliza el Entity Framework como mecanismo de acceso a datos y con el enfoque Code First. Este enfoque consiste en la generación de una base de datos a partir de un Modelo especificado el código.

El usuario podrá crear la base de datos de la siguiente manera: Una vez abierto el proyecto, en la consola de administración de paquetes, se ejecuta el comando "**update-database -force**". Este comando construirá automáticamente una base de datos SQL Server LocalDB [35], con las entidades y relaciones especificadas en el Modelo y en la clase ApplicationDbContext.cs. Esta base de datos quedará ubicada de manera local en el proyecto y el usuario podrá actualizar su estructura utilizando el mismo comando, cada vez que realice cambios en el modelo. Adicionalmente también se podrá utilizar el IDE de Visual Studio para visualizar los objetos de la base de datos y realizar consultas.

IIS Express [41]: es una herramienta que viene por defecto instalada con el IDE de Visual Studio, y tiene como objetivo publicar el directorio virtual de la aplicación Web, de tal forma que a través de la siguiente dirección http://localhost:57832/, un navegador web podrá acceder a la aplicación Web, en modo de depuración o en cualquier otro modo configurado por el

usuario. Una vez compilada y ejecutada la aplicación el IDE de Visual Studio Carga de forma automática y de forma transparente el IIS Express, publicando el directorio virtual de la aplicación.

Explorador Web: Una vez publicada la aplicación web mediante el IIS Express, el IDE de Visual Studio ejecuta el navegador Web seleccionado por el usuario, este navegador web puede ser: Google Chrome, Mozilla Firefox, Microsoft Edge, o inclusive un navegador de dispositivo móvil compatible con la aplicación.

5.3.5 Estructura del Sistema

En esta sección se especifican los paquetes que hacen parte del sistema, las clases que lo conforman, y una descripción de sus funcionalidades. DotNetGenerator está diseñado en el IDE (Entorno de Desarrollo Integrado) de Eclipse, y para su construcción se utilizó un lenguaje de programación denominado Xtend [58], el cual es un dialecto de Java que facilita el trabajo con cadenas de caracteres complejas. Cada una de las clases del sistema se guardará en un archivo con extensión *.xtend.

El sistema está conformado por el paquete principal co.edu.javeriana.dotnetgenerator, este paquete a su vez está integrado por los paquetes, **co.edu.javeriana.dotnetgenerator, co.edu.javeriana.dotnetgenerator.generators**, **co.edu.javeriana.dotnetgenerator.templates**, y **co.edu.javeriana.dotnetgenerator.utils**. Estos Paquetes contienen las clases de los generadores necesarios para identificar la estructura de un Modelo ISML, y clases con las plantillas necesarias para convertir este modelo en un código fuente con la estructura de directorios y archivos de una Aplicación Web ASP.NET. La siguiente Ilustración 9, especifica la estructura de paquetes que integran el sistema:

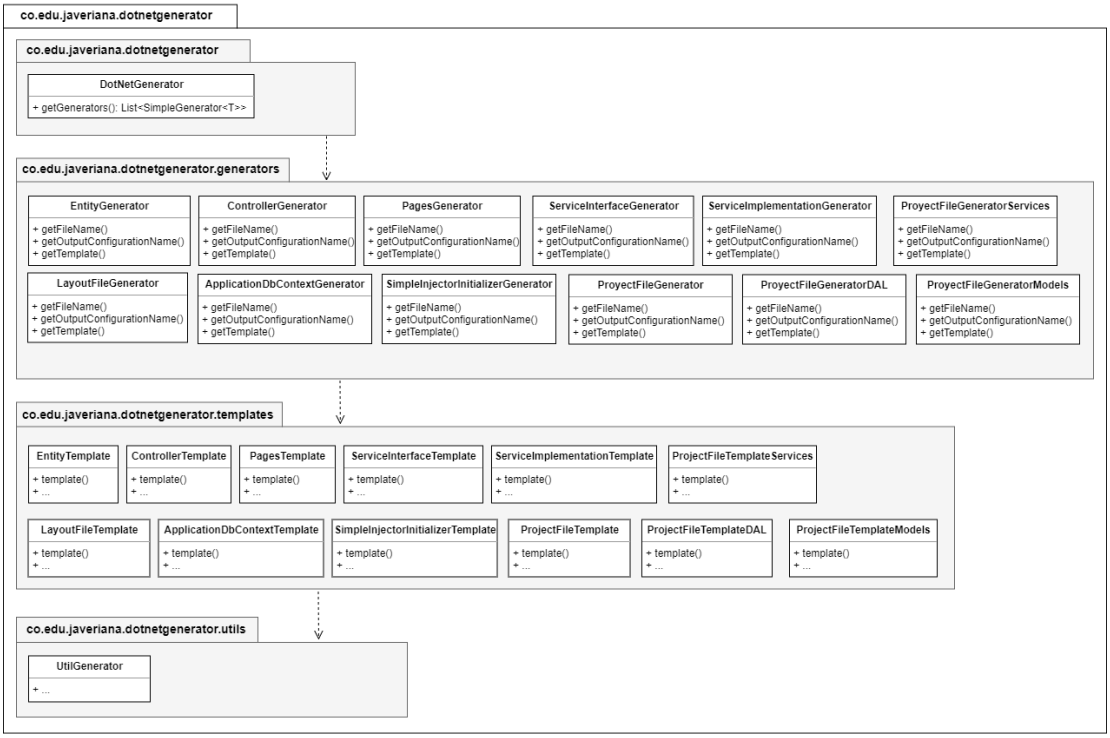


Ilustración 9: Diagrama con la estructura general del sistema

5.3.5.1 Paquete co.edu.javeriana.dotnetgenerator

El paquete co.edu.javeriana.dotnetgenerator contiene la clase dotnetgenerator, esta es la clase principal y puede ser considerada como punto de arranque del sistema y método principal es el getGenerators(). Esta clase se encarga de ejecutar cada uno de los generadores que integran el sistema. Estos generadores se encuentran como clases independientes dentro del paquete co.edu.javeriana.dotnetgenerator.generators.

5.3.5.2 Paquete co.edu.javeriana.dotnetgenerator.generators

El paquete co.edu.javeriana.dotnetgenerator.generators los 12 generadores de propósito específico que conforman el sistema. Cada generador está definido como una clase independiente que hereda de la clase SimpleGenerator. La Ilustración 11 contiene la representación gráfica del paquete con sus clases y métodos:

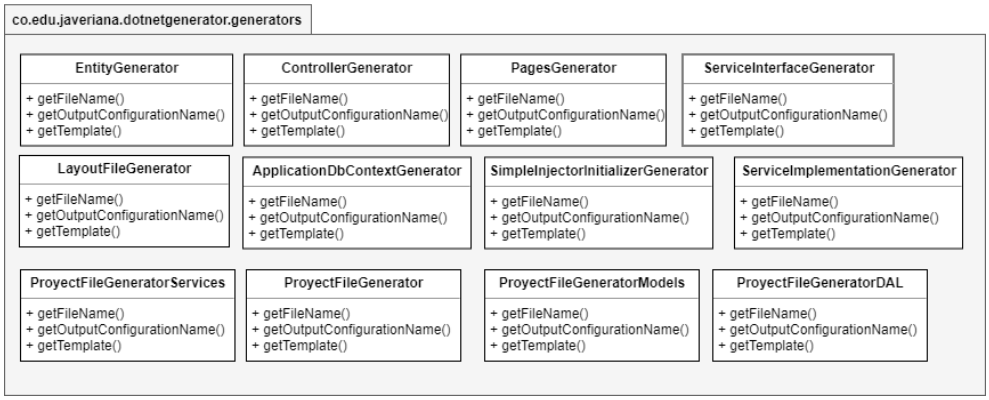


Ilustración 10: Paquete co.edu.javeriana.dotnetgenerator.generators

La siguiente Tabla 7 contiene los métodos principales de un generador son :

Nombre del método	Descripción
getFileName()	Este método permite obtener el nombre del archivo físico resultado de la generación.
getOutputConfigurationName()	Este método permite obtener el parámetro de configuración especificado para el generador.
getTemplate()	Método permite obtener la plantilla definida para el generador.

Tabla 7: Listado de los métodos principales de un generador.

La Tabla 8 contiene la especificación de las clases correspondientes a cada uno de los generadores definidos en la Ilustración 11:

Nombre de Clase	Descripción
EntityGenerator	**Generador de Modelos**: Este generador se encarga de procesar los elementos de tipo Entity definidos en el ISML, y generar como resultado archivos de clase de C# con extensión *.cs, con las clases y atributos definido en cada entidad. Este generador procesa objetos de tipo Entity especificados en el ISML y utiliza como plantilla la clase EntityTemplate.
PagesGenerator	**Generador de Vistas**: Este generador se encarga de procesar los elementos de tipo Page definidos en el ISML, y generar archivos de sintaxis Razor [9] con extensión *.cshtml. Este generador procesa objetos de tipo Page especificados en el ISML y utiliza como plantilla la clase PagesTemplate.
ControllerGenerator	**Generador de Controladores**: Este generador se encarga de procesar los elementos de tipo Controller definidos en el ISML, y generar como resultado archivos de clase de C# con extensión *.cs, con la lógica implementada en

	cada controlador. Este generador procesa objetos de tipo Controller especificados en el ISML y utiliza como plantilla la clase ControllerTemplate.
ServiceInterface-Generator	**Generador de Interfaz de Servicios**: Este generador se encarga de procesar los elementos de tipo Service definidos en el ISML, y generar como resultado archivos de clase de C# con extensión *.cs, con la interfaz para cada servicio. Este generador procesa objetos de tipo Service especificados en el ISML y como plantilla la clase ServiceInterfaceTemplate.
ServiceImplemen-tationGenerator	**Generador de Implementación de Servicios**: Este generador se encarga de procesar los elementos de tipo Service definidos en el ISML, y generar como resultado archivos de clase de C# con extensión *.cs, con la implementación interfaz para cada servicio. Se debe tener en cuenta que las implementaciones contendrán métodos no implementados. Este generador procesa objetos de tipo Service especificados en el ISML y utiliza como plantilla la clase ServiceInterfaceTemplate.
ProyectFileGenera-tor	**Archivo de proyecto "AplicacionWeb"**: Este generador tiene como objetivo crear el Archivo de proyecto de la aplicación de extensión *.csprj, este archivo XML contiene el listado de todos los archivos y directorios del proyecto "AplicacionWeb". El generador procesa objetos de tipo CompositeTypeSpecification, el cual proporciona acceso a los objetos de tipo (Entity, Page, Controller y Service), y utiliza como plantilla la clase ProjectFileTemplate.
ProyectFileGenera-torDAL	**Archivo de proyecto "AplicacionWeb.DAL"**: Este generador tiene como objetivo crear el archivo *.csprj del proyecto "AplicacionWeb.DAL". El generador procesa objetos de tipo CompositeTypeSpecification y utiliza como plantilla la clase ProjectFileTemplateDAL.
ProyectFileGenera-torModels	**Archivo de proyecto "AplicacionWeb.Models"**: Este generador tiene como objetivo crear el archivo *.csprj del proyecto "AplicacionWeb.Models". El generador procesa objetos de tipo CompositeTypeSpecification y utiliza como plantilla la clase ProjectFileTemplateModels.
ProyectFileGenera-torServices	**Archivo de proyecto "AplicacionWeb.Services"**: Este generador tiene como objetivo crear el archivo *.csprj del proyecto "AplicacionWeb.Services". El generador procesa objetos de tipo CompositeTypeSpecification y utiliza como plantilla la clase ProjectFileTemplateServices.
ApplicationDbCon-textGenerator	**Archivo DbContext**: Este generador tiene como objetivo generar dinámica-mente un archivo de clase de C# denominado "ApplicationDbContext.cs" el cual deriva de la clase DbContext [53] con las siguientes especificaciones:

	• Incluye la declaración de objetos de tipo DbSet, por cada entidad especificada en el modelo. La clase DbSet [54] representa un conjunto de entidades que se puede utilizar para las operaciones de creación, lectura, actualización y eliminación. • Incluye la Configuración y asignación de Relaciones entre las entidades del modelo, utilizando el Entity Framework Fluent API [8], Específicamente en el método "OnModelCreating" [55]. El generador procesa objetos de tipo Entity especificados en el ISML, y utiliza como plantilla la clase ApplicationDbContextTemplate.
SimpleInjectorInitializerGenerator	**Archivo para establecer la Inyección de dependencias**: Este generador tiene como objetivo generar dinámicamente un archivo de clase de C# denominado "SimpleInjectorInitializer.cs" [52], el cual registra la inyección de dependencias que será utilizada en los constructores de cada controlador. Este generador procesa objetos de tipo CompositeTypeSpecification y utiliza como plantilla la clase SimpleInjectorInitializerTemplate.
LayoutFileGenerator	**Archivo "_Layout.cshtml"**: Este generador tiene como objetivo crear el archivo "_Layout.cshtml", bajo sintaxis Razor [9], con las siguientes especificaciones: • Contiene la Definición del formulario maestro de la aplicación Web. • Contiene una Barra de menú ubicada en el encabezado de la página web, y cuyas opciones corresponden a las acciones especificadas como default en cada controlador ISML. Este generador procesa objetos de tipo CompositeTypeSpecification, y utiliza la plantilla LayoutFileTemplate.

Tabla 8: Listado de las clases definidas para cada generador

El paquete de generadores se apoya en el paquete co.edu.javeriana.dotnetgenerator.templates, el cual contiene cada una de las plantillas definidas para cada generador.

5.3.5.3 Paquete co.edu.javeriana.dotnetgenerator.templates

El paquete co.edu.javeriana.dotnetgenerator.templates contiene 12 plantillas de propósito específico, estas plantillas se encargan de generar el texto de cada uno de los archivos solicitados por cada generador.

Cada plantilla está definida como una clase independiente que hereda de la clase SimpleTemplate. Todas las plantillas contienen el método template() el cual es el encargado de la construcción de la plantilla del documento solicitado. Este método puede apoyarse en otros métodos con la definición de plantillas para ciertas secciones de cada documento. La Ilustración 12 contiene la representación gráfica del paquete con sus clases:

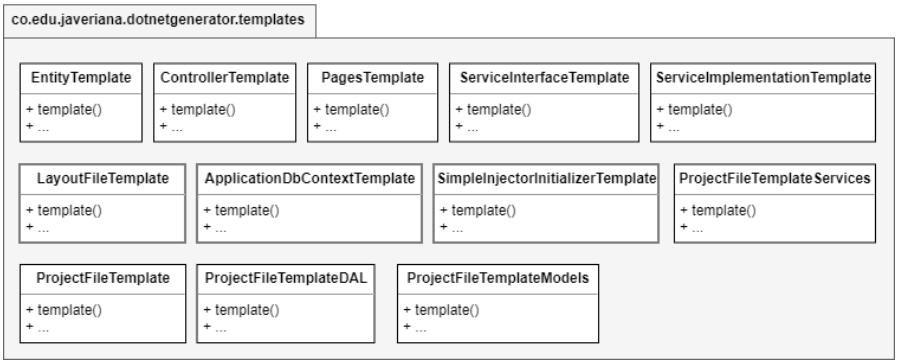


Ilustración 11: Paquete co.edu.javeriana.dotnetgenerator.templates

La Tabla 9 especifica las clases correspondientes, a cada una de las plantillas disponibles en el sistema, definidas en la Ilustración 12:

Nombre de Clase	Descripción
EntityTemplate	Plantilla definida para la generación de los archivos de clase en C# que representarán el Modelo de la aplicación. Procesa un objeto de tipo de tipo Entity. Contiene los métodos con la definición de las plantillas para: Tipos de datos, Atributos, DataAnnotations y otros elementos propios de un modelo en C#.
PagesTemplate	Plantilla definida para la generación de los archivos de las vistas en sintaxis Razor. Procesa un objeto de tipo de tipo Page. Contiene los métodos con las plantillas de controles de usuario denominadas widget. los widgets soportados por esta versión del generador son: Panel, Form, Text, Image, Password, Link, RadioChooser, Calendar, Media, Label, OutputText, CheckBox, Button, PanelButton, y DataTable.
ControllerTemplate	Plantilla definida para la generación de los archivos de clase en C#, que representarán los controladores de la aplicación y procesa un objeto de tipo de tipo Controller. Contiene los métodos con la definición de plantillas para: Métodos con retorno de tipo ActionResult. Declaración de variables, Invocación de métodos, Sentencias IF, Foreach, Entre otros.
ServiceInterface-Template	Plantilla definida para la generación de los archivos de clase en C# que representarán las interfaces de los servicios de la aplicación. Procesa un objeto de tipo de tipo Service. Contiene métodos con las plantillas para definir las firmas de los métodos de una interfaz y otros elementos de una interfaz en C#.
ServiceImplementationTemplate	Plantilla definida para la generación de los archivos de clase en C# que representarán la implementación de las interfaces de los servicios de la aplicación. Procesa un objeto de tipo de tipo Service. Contiene métodos con plantillas para definir: la implementación de métodos de cada interfaz en C# y otros elementos de una clase en C#.

ProjectFileTemplate	Plantilla definida para la generación del *.csprj del proyecto "AplicacionWeb". Procesa un objeto de tipo de tipo CompositeTypeSpecification. Contiene un método con la plantilla de un documento XML de un proyecto de Visual Studio. Este método permite iterar y obtener los elementos de tipo (Entity, Page, Controller y Service), para agregarlos a las secciones correspondientes de este documento.
ProjectFileTemplateDAL	Plantilla definida para la generación del *.csprj del proyecto "AplicacionWeb.DAL". Procesa un objeto de tipo de tipo CompositeTypeSpecification. Implementa los mismos métodos de la plantilla "ProjectFileTemplate", pero con un XML diferente.
ProjectFileTemplateModels	Plantilla definida para la generación del *.csprj del proyecto "AplicacionWeb.Models". Procesa un objeto de tipo de tipo CompositeTypeSpecification. Implementa los mismos métodos de la plantilla "ProjectFileTemplate", pero con un XML diferente.
ProjectFileTemplateServices	Plantilla definida para la generación del *.csprj del proyecto "AplicacionWeb.Services". Procesa un objeto de tipo de tipo CompositeTypeSpecification. Implementa los mismos métodos de la plantilla "ProjectFileTemplate", pero con un XML diferente.
ApplicationDbContextTemplate	Plantilla definida para la generación del archivo ApplicationDbContext.cs. Procesa un objeto de tipo de tipo Entity. Contiene un método con la plantilla de un documento ApplicationDbContext, Este método permite iterar y obtener los elementos de tipo Entity, para agregarlos a las secciones correspondientes de este documento.
SimpleInjectorInitializerTemplate	Plantilla definida para la generación del archivo SimpleInjectorInitializer.cs. Procesa un objeto de tipo de tipo CompositeTypeSpecification. Contiene un método con la plantilla de un documento SimpleInjectorInitializer. Este método permite iterar y obtener los elementos de tipo (Entity, Page, Controller y Service), para agregarlos a las secciones correspondientes de este documento.
LayoutFileTemplate	Plantilla definida para la generación del archivo "_Layout.cshtml". Procesa un objeto de tipo de tipo CompositeTypeSpecification. Contiene un método con la plantilla de un documento _Layout. Este método permite iterar y obtener los elementos de tipo (Entity, Page, Controller y Service), para agregarlos a las secciones correspondientes de este documento con sintaxis Razor.

Tabla 9: Listado de las clases definidas para cada plantilla

5.3.5.4 Paquete co.edu.javeriana.dotnetgenerator.utils

El paquete co.edu.javeriana.dotnetgenerator.utils, contiene la clase UtilGenerator. Esta clase proporciona un conjunto de métodos, que permiten realizar operaciones en común para todas las clases de las plantillas, facilitando la reutilización de código.

La siguiente Tabla 10 relaciona los métodos disponibles en la clase UtilGenerator:

Nombre de método	Descripción
CharSequence getDataType(Type)	Método encargado de obtener un tipo de dato en C#, a partir de un objeto Type, enviado como parámetro.
CharSequence generateArray(ViewInstance)	Método que recibe una instancia de vista de ISML y retorna un arreglo construido con la sintaxis C#. Esto con el objetivo de ser utilizado en el método valueTemplate en el caso en que la expresión ISML sea un ViewInstance.
CharSequence generateParametersActionCall(ActionCall)	Método encargado de generar los parámetros de un llamado a una acción enviada como parámetro.
CharSequence generateReferencedElement(Reference)	Método encargado de generar los elementos referenciados de la referencia enviada como parámetro.
CharSequence generateTailedElement(VariableReference)	Método encargado de generar la referencia enviada por parámetro con su respectiva cola).
CharSequence generateTailedElementOmitFirst(VariableReference)	Método encargado de realizar la misma operación del método generateTailedElement, pero omitiendo el primer elemento de la cola.
CharSequence generateLastTailedElement(VariableReference)	Método encargado de obtener la cadena de último elemento de la cola.
CharSequence getParameters(ParameterizedReference<?>)	Método encargado de generar los parámetros, específicamente referencias parametrizadas que pueden ser de cualquier clase.
boolean hasTail(Reference)	Método encargado de verificar si una referencia enviada como parámetro, tiene cola.
CharSequence valueTemplate(Expression)	Método encargado de convertir una expresión enviada como parámetro, en una cadena específica dependiendo del tipo de expresión.
CharSequence valueTemplateForEntities(Expression)	Método con la misma funcionalidad de valueTemplate, pero con un retorno específico diferente, si la expresión es una entidad de un Elemento referenciado.

Tabla 10: Listado de los métodos definidos en la clase UtilGenerator

6. PRUEBA DE CONCEPTO

6.1 Descripción de la aplicación de prueba

Para validar el funcionamiento del sistema se realizó una prueba de concepto, la cual consistió en la fabricación de un Modelo ISML de una aplicación para gestionar una tienda de mascotas, el nombre del proyecto ISML se denominó **GestionPetStoreISML.** Este modelo con-

tiene las especificaciones de una aplicación Web que permitirá realizar las siguientes operaciones, descritas en los siguientes módulos:

6.1.1 Módulos: Aplicación para Gestionar una Tienda de Mascotas

La siguiente Tabla 11, describe cada uno de los módulos de la aplicación:

Módulo	Descripción
Configuración	Este módulo permite realizar la gestión de los datos de las tablas base del sistema, estas tablas son: País, Departamento, Ciudad, Género, y está compuesto por los siguientes Submódulos: • **Administrar Datos País**: Permite realizar operaciones de consulta, creación, actualización y eliminación sobre la tabla País. • **Administrar Datos Departamento**: Permite realizar operaciones de consulta, creación, actualización y eliminación sobre la tabla Departamento. • **Administrar Datos Ciudad**: Permite realizar operaciones de consulta, creación, actualización y eliminación sobre la tabla Ciudad. • **Administrar Datos Género**: Permite realizar operaciones de consulta, creación, actualización y eliminación sobre la tabla Género.
Tipo de producto	Este módulo permite realizar la gestión de los tipos de productos que serán ofrecidos por la tienda. Permite realizar operaciones de consulta, creación, actualización y eliminación, sobre la tabla Tipo_Producto.
Producto	Este módulo permite realizar la gestión de los productos que se venden en la tienda, permite realizar operaciones de consulta, creación, actualización y eliminación, sobre la tabla Producto.
Cliente	Este módulo permite realizar la gestión de los clientes de la tienda, permite realizar operaciones de consulta, creación, actualización y eliminación sobre la tabla Cliente.
Venta	Este módulo permite realizar la venta de uno o varios productos de la tienda y generar el registro del pago realizado sobre esta venta, utilizando un datáfono. **Crear nueva venta** 1. El vendedor selecciona la opción “Crear nueva Venta” 2. El sistema muestra un formulario solicitando los datos del cliente. 3. El vendedor selecciona el cliente y presiona el botón “Crear”. 4. El sistema muestra un formulario para agregar productos a la venta, o para registrar el pago. **Agregar productos a la venta** 5. El vendedor selecciona la opción “Agregar producto (+)”. 6. El sistema muestra un formulario con el listado de productos disponibles. 7. El vendedor selecciona la opción “Seleccionar”, de un producto específico. 8. El sistema muestra un formulario con los datos del producto y una caja de texto para ingresar la cantidad.

	9. El vendedor ingresa la cantidad y selecciona la opción “Agregar” 10. El sistema regresa al formulario con el listado de los productos seleccionados, mostrando sus totales y el valor a pagar. 11. El vendedor repite los pasos del 5 al 10, hasta registrar todos los productos de la venta. **Registrar el pago** 12. El vendedor selecciona la opción “Registrar Pago” 13. El sistema muestra un formulario mostrando un resumen de la venta, y un botón “Capturar Datos Datáfono”. 14. El vendedor presiona el botón “Capturar Datos Datáfono”. 15. El sistema invoca a un servicio “ServicioPagos” y le solicita ejecutar la operación “ObtenerDatosDatafono”, enviándole un objeto “RegistroPago”, con los datos del pago. 16. El sistema asume que este servicio le retornará un código de aprobación, y genera un registro en la tabla de “RegistroPago” mostrándole al usuario un formulario con los datos del pago y el código de aprobación.
Pagos	En este módulo un vendedor podrá ver el listado con la información de todos los pagos que se han registrado.

Tabla 11: Módulos de la aplicación Web para gestionar una tienda de mascotas

6.2 Descripción del Modelo ISML

El modelo ISML de la aplicación Web para gestionar una tienda de mascotas, contiene los siguientes componentes modelados de tipo: Entidad, Página, Controlador, y Servicio. La Ilustración 12 contiene los componentes de tipo entidad modelados en ISML:

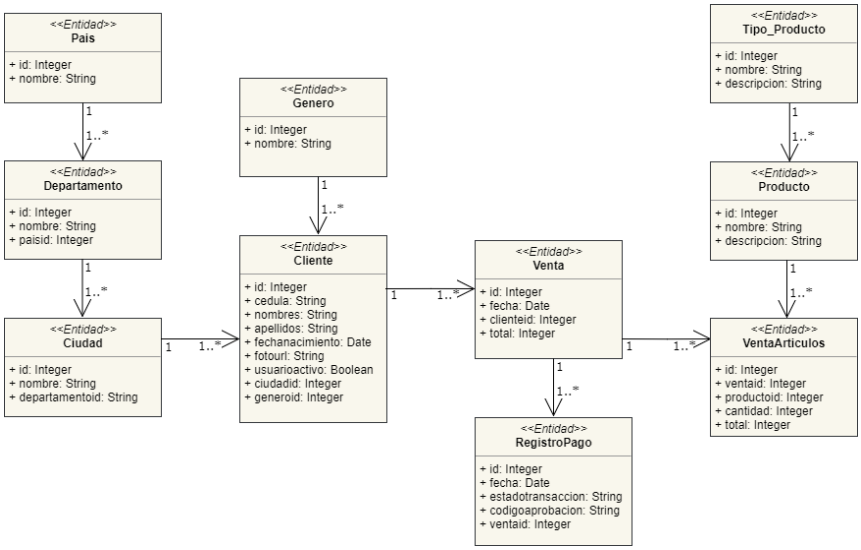


Ilustración 12: Entidades del Modelo ISML

Las ilustraciones 13, 14, 15 y 16 son un ejemplo de cada componente modelado en ISML:

Ilustración 13: Entidad "Producto" modelada en ISML

Producto.isml

```
package AplicacionWeb.Models;

entity Producto {
    Integer id;
    String nombre must be NotNull;
    String descripcion;
    String sitioweb;
    String fotourl;
    String videourl;
    Integer tipo_productoid;
    Tipo_Producto tipo_producto;
    Integer precio;
    List<VentaArticulos> ventaarticulos opposite producto;
}
```

Ilustración 14: Controlador "ProductoController" modelado en ISML

ProductoController.isml

```
package Controllers;

import AplicacionWeb.Models.*;
import Producto.*;

controller ProductoController {
    has Persistence<Producto> productopersistence;
    has Persistence<Tipo_Producto> tipoproductopersistence;

    default consultarProducto() {
        show ConsultarProducto(productopersistence.findAll());
    }

    crearProducto() {
        Producto producto = new Producto;
        Array<Tipo_Producto> listatipoproducto = tipoproductopersistence.findAll();
        show CrearProducto(producto, listatipoproducto);
    }

    crearProducto(Producto producto) {
        productopersistence.create(producto);
        -> consultarProducto();
    }

    eliminarProducto(Integer id) {
        show EliminarProducto(productopersistence.find(id));
    }

    eliminarProducto(Producto producto) {
        productopersistence.remove(producto);
        -> consultarProducto();
    }

    editarProducto(Integer id) {
        Array<Tipo_Producto> listatipoproducto = tipoproductopersistence.findAll();
        show EditarProducto(productopersistence.find(id),listatipoproducto);
    }

    editarProducto(Producto producto) {
        if(productopersistence.isPersistent(producto))
        {
            productopersistence.save(producto);
        }
        else
        {
            productopersistence.create(producto);
        }
        -> consultarProducto();
    }
}
```

Ilustración 15: Servicio "ServicioPagos" modelado en ISML

ServicioPagos.isml

```
package Services;

service ServicioPagos <RegistroPago>{
    native String obtenerDatosDatafono(RegistroPago obj);
    native Boolean anularTransaccion(RegistroPago obj);
    native RegistroPago consultarTransaccion(String id);
    native Long consultarSaldoTarjeta(RegistroPago obj);
    native List <RegistroPago> consultarReporteTransacciones();
}
```

Ilustración 16: Página "CrearProducto" modelada en ISML

CrearCiudad.isml

```
package Ciudad;

import Models.*;
import Controllers.*;

page CrearCiudad(Ciudad ciudad, Array<Departamento> listadepartamento)
controlledBy CiudadController {

    Panel("Crear Ciudad") {
        Form {

            Text("Nombre:", ciudad.nombre, 15, 3);
            ComboChooser("nombre", listadepartamento,
                ciudad.departamentoid,"--Seleccione el departamento--"
            );

            PanelButton("Botones") {
                Button("Crear") -> crearCiudad(ciudad);
            }
        }

        PanelButton("Botones") {
            Button("Volver al listado") -> consultarCiudades();
        }
    }
}
```

6.2.1 Definición de Controladores, Páginas y Servicios

El modelo de la aplicación contiene un conjunto de Controladores, Páginas y Servicios. La Tabla 12 contiene el listado de los controladores modelados, las páginas que cada uno de estos controla y el listado de servicios utilizados por cada controlador:

Controlador	Páginas	Servicios	Descripción
GeneroController	ConsultarGenero CrearGenero EditarGenero EliminarGenero	Persistence	Controlador encargado de coordinar las operaciones sobre la tabla Género.
PaisController	ConsultarPais CrearPais EliminarPais	Persistence	Controlador encargado de coordinar las operaciones sobre la tabla País.
Departamento-Controller	ConsultarDepartamentos CrearDepartamento EditarDepartamento EliminarDepartamento	Persistence	Controlador encargado de coordinar las operaciones sobre la tabla Departamento.
CiudadController	ConsultarCiudades CrearCiudad EditarCiudad EliminarCiudad	Persistence	Controlador encargado de coordinar las operaciones sobre la tabla Ciudad.
ClienteController	ConsultarClientes CrearCliente EditarCliente EliminarCliente	Persistence	Controlador encargado de coordinar las operaciones sobre los datos de los clientes de la tienda.
Tipo_ProductoController	ConsultarTipo_Producto CrearTipo_Producto EditarTipo_Producto EliminarTipo_Producto	Persistence	Controlador encargado de coordinar las operaciones sobre los tipos de producto ofrecidos en la tienda.
ProductoController	ConsultarProducto CrearProducto EditarProducto EliminarProducto	Persistence	Controlador encargado de coordinar las operaciones sobre los datos de los productos de la tienda
VentaController	ConsultarVenta CrearVenta CrearVentaArticulo EditarVenta EliminarVenta EliminarVentaArticulo SeleccionarVentaArticulo	Persistence	Controlador encargado de coordinar todas las operaciones asociadas a una Venta.
PagosController	ConsultarRegistroPago EliminarRegistroPago RegistrarPago ResultadoTransaccion	ServicioPagos Persistence	Controlador encargado de coordinar las operaciones relacionadas con los pagos.

HomeController	Index	N/A	Controlador encargado de coordinar el formulario de bienvenida.
Configuracion-Controller	ConsultarConfiguracion	N/A	Controlador encargado de coordinar el formulario con las opciones de configuración del sistema.

Tabla 12: Listado de Controladores, Páginas y Servicios especificados en ISML

El controlador utiliza el servicio de Persistencia propio de ISML, para poder realizar operaciones sobre los datos de las entidades modeladas. Y utiliza un servicio denominado "ServicioPagos", el cual contiene operaciones relacionadas con el pago. Este servicio se modeló asumiendo que la organización ya tiene un componente de software que tiene las operaciones especificadas en este servicio. En el código generado en C# para este servicio se genera una interfaz y una implementación con métodos vacíos, para que el ingeniero de desarrollo se encargue de su implementación. La ilustración 15 contiene un ejemplo del servicio "ServicioPagos" modelado en ISML.

6.3 Aplicación Web ASP.NET MVC Generada

En esta sección se describe la arquitectura de la aplicación generada, la estructura gráfica de la aplicación, y las interfaces gráficas de los procesos que se pueden realizar en la aplicación.

6.3.1 Arquitectura de la Aplicación Generada

La aplicación generada está conformada por 4 proyectos y una base de datos, la Ilustración 17 describe estos 4 proyectos y sus relaciones entre sí.

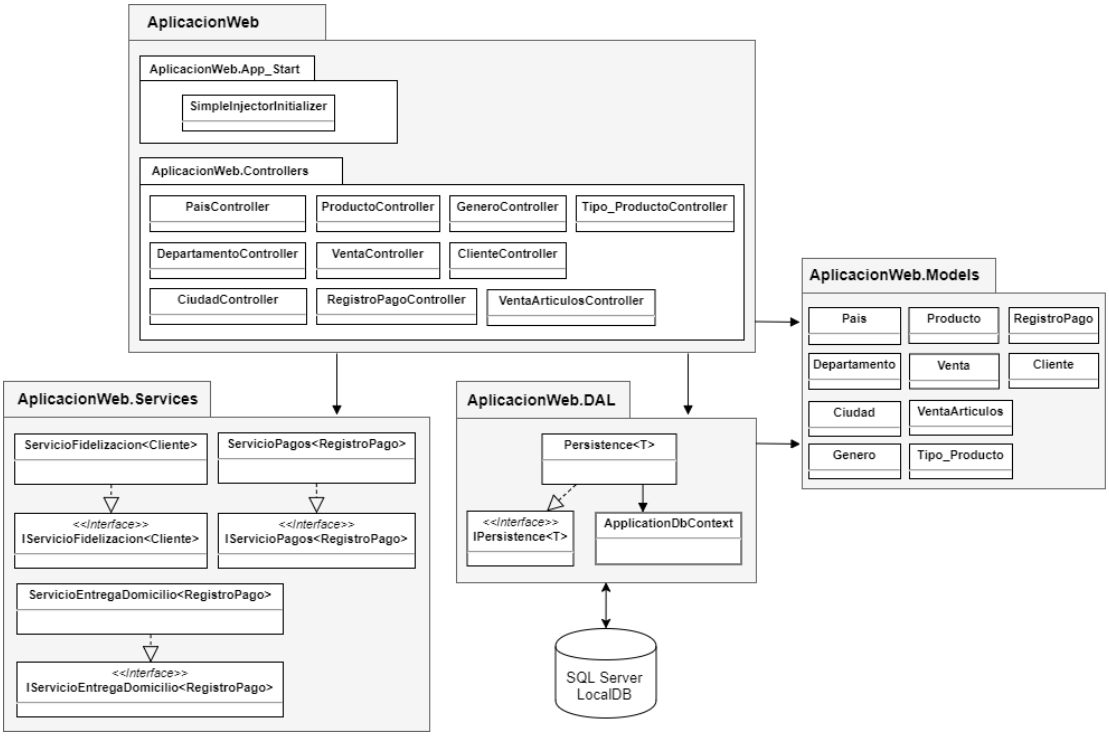


Ilustración 17: Arquitectura de la aplicación Web Generada

A continuación se describen los proyectos especificados en la ilustración 17:

AplicacionWeb: Este proyecto representa la capa de presentación de la aplicación. Contiene Vistas (formularios cshtml), Controladores, Scripts de JQuery y Bootstrap, y todos los demás componentes de presentación de la aplicación. También contiene la clase "SimpleInjectorInitializer" para la inyección de dependencias de los controladores.

AplicacionWeb.Models: Este proyecto contiene los modelos transversales a toda la aplicación. Estos modelos son utilizados por la capa de presentación "AplicacionWeb" y por la capa de acceso a datos "AplicacionWeb.DAL".

AplicacionWeb.DAL: Este proyecto representa la capa de acceso a datos de la aplicación. Utiliza el patrón Repository, mediante la definición de la Interfaz "IPersistence<T>" y su implementación a través de la clase "Persistence<T>". Esta clase implementa los métodos de persistencia (create, save, remove, find, findAll, entre otros), y utiliza el objeto "ApplicationDbContext" para persistir los datos en la base de datos a través del EntityFramework.

AplicacionWeb.Services: Este proyecto representa la capa de servicios de la aplicación. Contiene las interfaces de los servicios y sus respectivas implementaciones. Estas implementaciones generan la siguiente excepción por defecto "NotImplementedException" por defecto.

Base de datos SQL Server LocalDB: Esta base de datos es generada a partir del modelo especificado en proyecto "AplicacionWeb.Models", y las relaciones de entidad especificadas en la clase "ApplicationDbContext" de la capa de acceso a Datos.

6.3.2 Estructura Gráfica de la Aplicación

La aplicación cuenta con una barra de menú que tiene como ítems, el listado de todos aquellos controladores que tienen una acción establecida por defecto. Si el usuario selecciona uno de estos ítems, el sistema carga una página específica. Este menú se ajusta al tamaño del formulario, cuentan con despliegue vertical u horizontal. Bajo la barra de menú se encuentran las páginas. La ilustración 18, contiene la distribución gráfica de la aplicación.

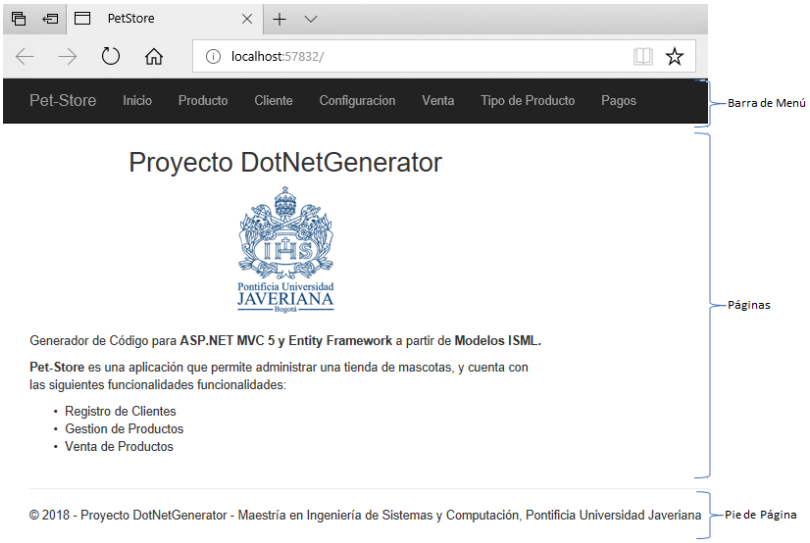
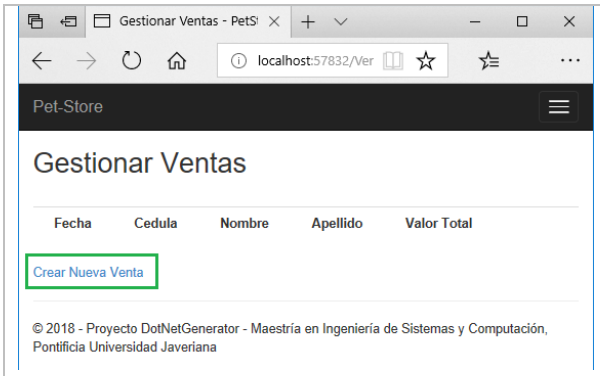


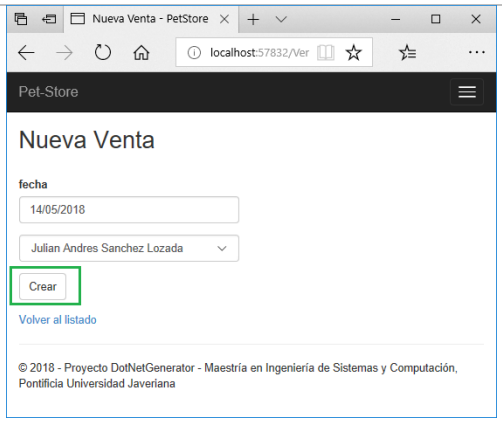
Ilustración 18: Distribución gráfica de la aplicación Web generada

6.3.3 Interfaz Gráfica: Proceso para crear una nueva Venta

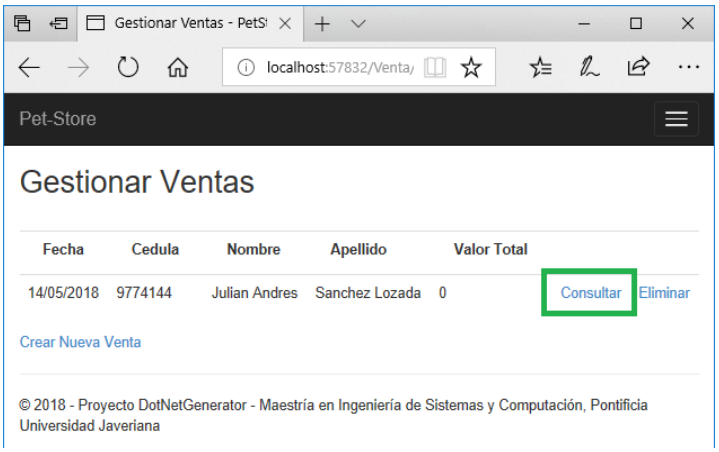
En esta sección se describen las Interfaces de usuario correspondientes al proceso para crear una nueva Venta, estas interfaces se encuentran descritas en la tabla 13.

1. El usuario ingresa al módulo de Ventas y Selecciona la opción "Crear nueva venta".	2. El usuario selecciona el Cliente y presiona el botón "Crear".





3. El usuario selecciona la opción “Consultar” ubicado en la misma fila de la venta.



4. El sistema muestra un formulario con los datos de la venta creada, y está disponible para agregar los productos a la venta.

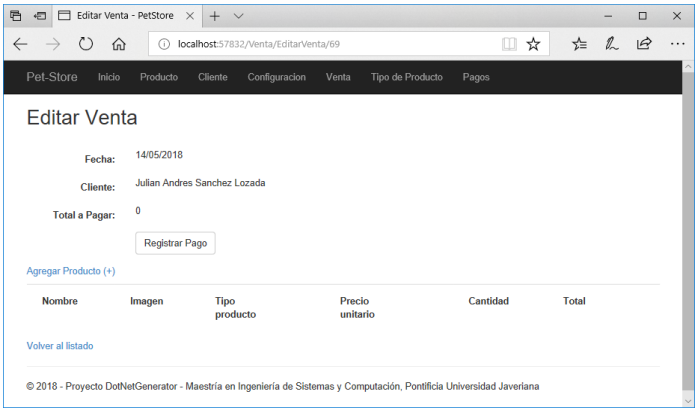
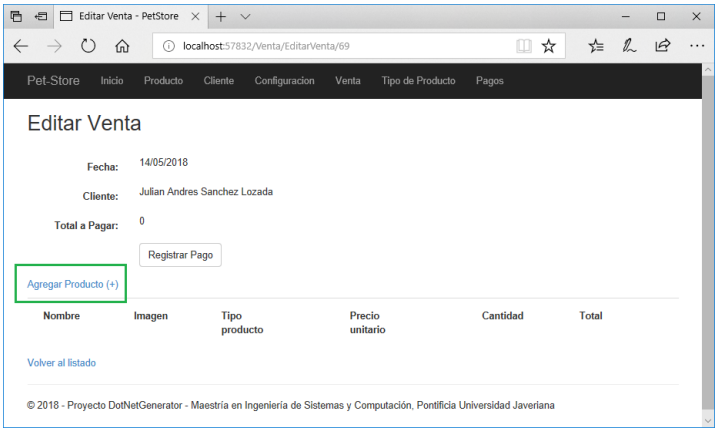


Tabla 13: Interfaces de usuario del proceso para crear una nueva Venta

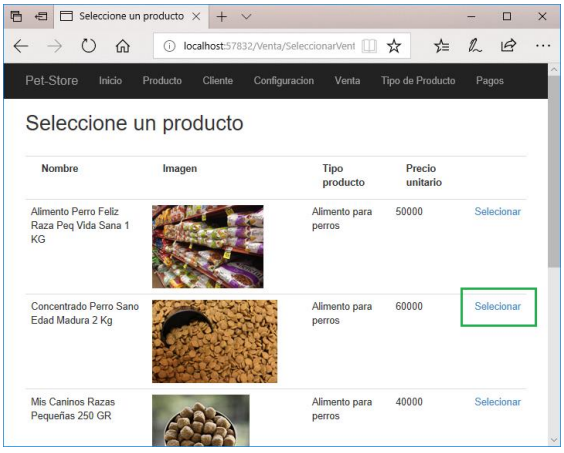
6.3.4 Interfaz Gráfica: Proceso para agregar productos a la Venta

En esta sección se describen las Interfaces de usuario correspondientes al proceso para agregar productos a la Venta, estas interfaces se encuentran descritas en la tabla 14.

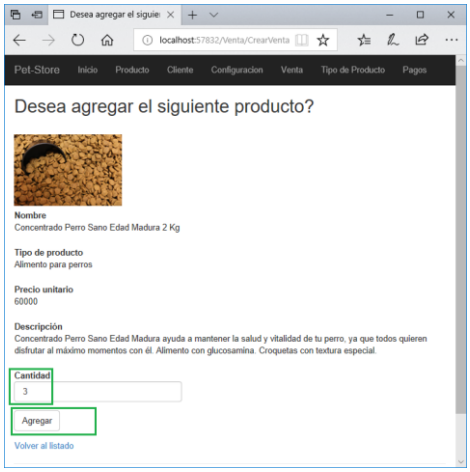
5. El usuario selecciona la opción “Agregar Productos”



6. El sistema muestra un formulario con el listado de productos disponibles. El usuario selecciona la opción “Seleccionar”.



7. El Sistema muestra un formulario con los detalles del producto. El usuario ingresa la cantidad y presiona el botón “Agregar”.



8. El sistema muestra el listado de productos seleccionados con sus respectivas cantidades y el valor a pagar. El usuario puede agregar más productos utilizando la opción “Agregar Producto (+)” .

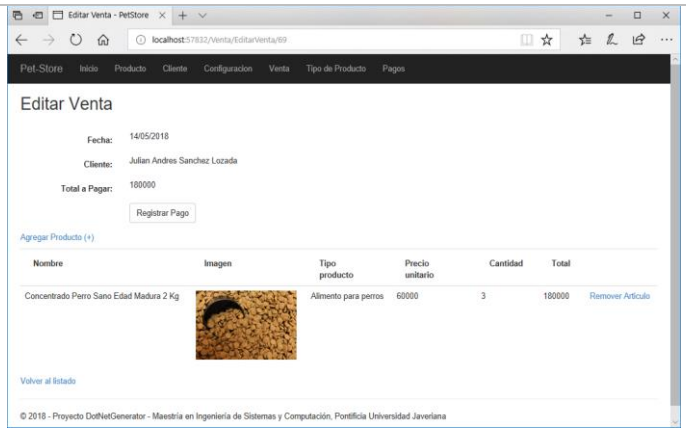
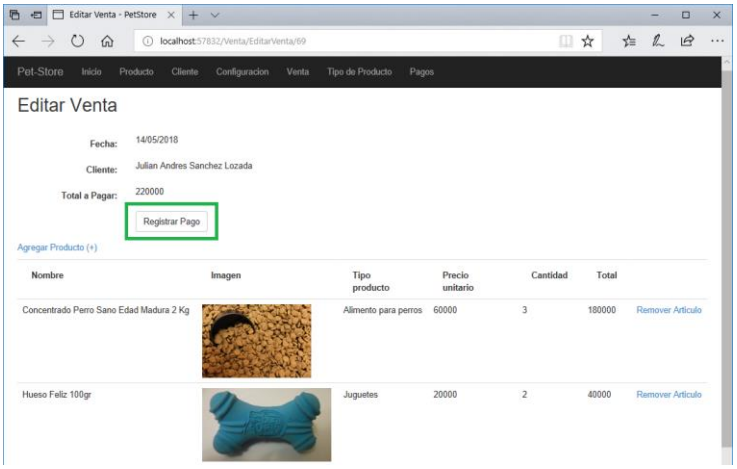


Tabla 14: Interfaces de usuario del proceso para agregar productos a la Venta

6.3.5 Interfaz Gráfica: Proceso para registrar el Pago

En esta sección se describen las Interfaces de usuario correspondientes al proceso para registrar el Pago, estas interfaces se encuentran descritas en la tabla 15.

9. El usuario selecciona el botón “Registrar Pago”. 	
10. El sistema muestra un formulario con los detalles del pago. El usuario presiona el botón “Capturar Datos Datáfono.	11. El sistema genera el pago utilizando el datáfono y muestra el número de aprobación, proporcionado por el datáfono.

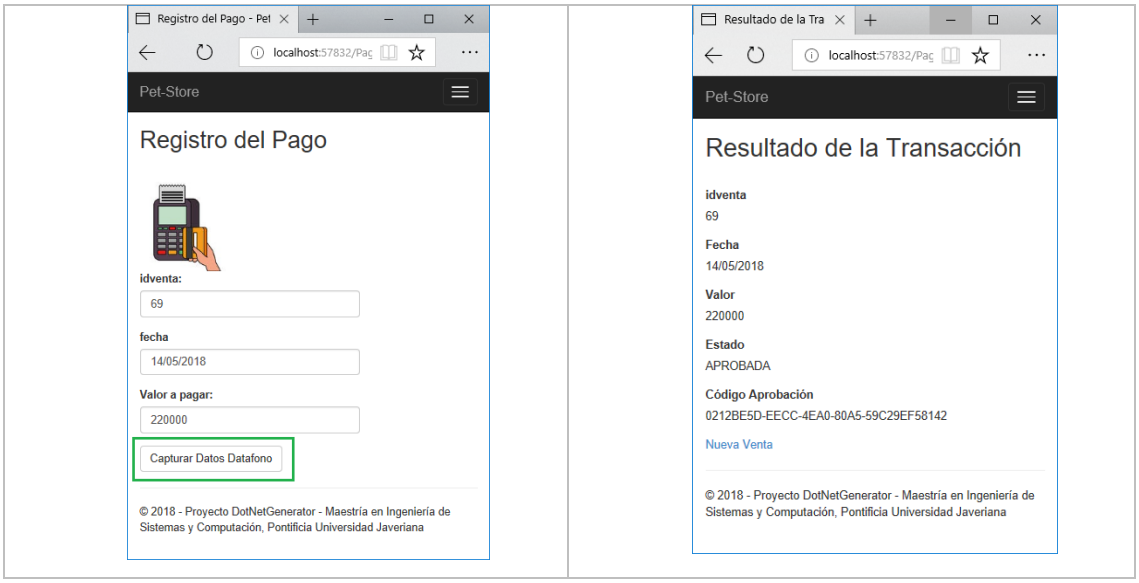


Tabla 15: Interfaces de usuario del proceso para registrar el pago

6.4 Análisis de los resultados

Una vez realizado el proceso de generación de código, se realizó la siguiente medición: Se contabilizaron las líneas de código correspondientes a cada uno de los componentes de ISML modelados (Entidades, Páginas, Controladores y Servicios). También se contabilizaron las líneas de código generadas en ASP.NET MVC. El objetivo de este análisis es identificar la reducción del esfuerzo asociada al uso del generador de código DotNetGenerator. Teniendo como criterio las líneas de código tanto del modelo como del resultado final de la generación.

6.4.1 Reducción de esfuerzo en términos de líneas de código

A partir de la medición realizada se obtuvieron los siguientes resultados. La Tabla 16 contiene las siguientes métricas tomadas a cada componente ISML:

- Componente ISML: Es el nombre del componente modelado en ISML
- Número de archivos ISML: Es el número de archivos de extensión *.isml modelados.
- Número de líneas de código ISML: Total de líneas en escritas en ISML por componente.
- Número de líneas generadas en ASP.NET MVC: número de líneas generadas en la plataforma destino por componente.
- Proporción de líneas de código ISML en ASP.NET MVC: Define la equivalencia de una línea de ISML en líneas de código generado en C# o CSHTML de ASP.NET MVC.

Componente ISML	Total Archivos ISML	Número de líneas en ISML	Líneas generadas en ASP.NET MVC	Proporción de líneas de código ISML en ASP.NET MVC

Entidad	10	123	383	1 línea en ISML equivale a 3,1 en C#
Controlador	11	629	831	1 línea en ISML equivale a 1,3 en C#
Página	40	1114	1988	1 línea en ISML equivale a 1,8 en CSHTML
Servicio	4	128	295	1 línea en ISML equivale a 2,3 en C#
	Total	1994	3497	1 línea en ISML equivale 1,8 en ASP.NET MVC

Tabla 16: Comparativa entre líneas de código ISML y líneas generadas en ASP.NET MVC

Como resultado del proceso de medición anterior, se pudo concluir que a partir de las 1994 líneas modeladas en ISML, el generador escribió un total de 3497 líneas de código en la plataforma destino. Por lo cual se determinó la siguiente proporción de equivalencia: cada línea en escrita en ISML equivale a 1,8 líneas de código en ASP.NET MVC. Este resultado muestra una reducción del esfuerzo en términos de líneas de código, gracias al uso del generador de código.

La Ilustración 19 especifica gráficamente en detalle el número de líneas de código escritas en ISML, para cada tipo de componente modelado. También contiene el número de líneas de código generadas después de realizar el proceso de generación por componente.

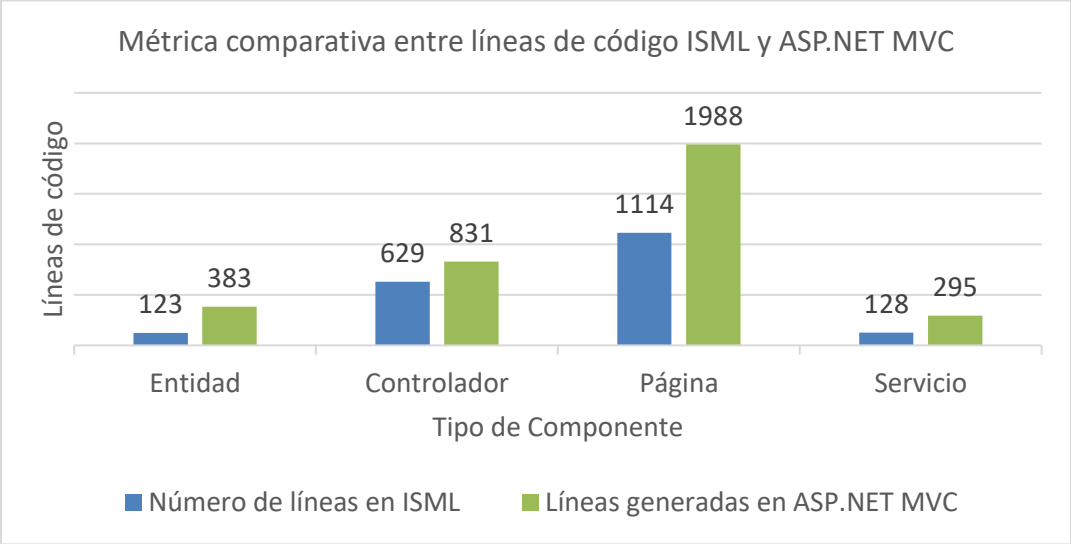


Ilustración 19: Comparativa entre líneas de código ISML y líneas en ASP.NET MVC

El componente que generó la mayor proporción de líneas de código en la plataforma destino fue el de tipo Entidad, ya que según los resultados cada línea en ISML de entidad equivale a 3,1 líneas de código en C#. En este componente se evidenció la mayor reducción de esfuerzo, ya que en ASP.NET MVC las entidades no solo se mapean como archivos de clase en C# por cada entidad. Estas también son la base para construir otros archivos como por ejemplo el

"ApplicationDbContext.cs", en donde cada entidad no solo es declarada, allí también se establecen las relaciones entre las entidades mediante el Fluent API [8] del Entity Framework, lo cual incrementa las líneas de código.

El segundo componente que generó la mayor proporción de líneas de código en la plataforma destino fue el de tipo Servicio, ya que según los resultados cada línea en ISML de servicio equivale a 2,3 líneas de código en C#. Este resultado se obtuvo porque cada servicio genera dos archivos de código independientes en C#, uno contiene la interfaz del servicio y otro su implementación.

El tercer componente en orden de proporción de líneas de código en la plataforma destino fue el correspondiente a las páginas, ya que según los resultados cada línea en ISML de página equivale a 1,8 líneas de código en archivos de sintaxis Razor [9]. Este componente tuvo como resultado una reducción moderada del esfuerzo, ya que la sintaxis Razor en términos de líneas de código es casi tan sencilla de implementar, como las modeladas en ISML. Este resultado hubiera sido superior, si en la implementación se hubieran incrustado controles de usuario más complejos, que requieren de un mayor detalle en la codificación.

El cuarto componente en orden de proporción de líneas de código en la plataforma destino fue el de tipo Controlador, ya que según los resultados cada línea en ISML de controlador equivale a 1,3 líneas de código en C#. Este componente no tuvo una proporción significativa, como los resultados logrados en las entidades y lo servicios. Esto se debe a la gran similitud que existe entre la estructura y la sintaxis del lenguaje ISML, con la de una clase de tipo controlador en C#, lo cual hace que en este tipo de componente no se hubiera podido ganar una reducción de esfuerzo más alta.

7. CONCLUSIONES

El enfoque de la Ingeniería Dirigida por Modelos MDE (Model Driven Engineering) [2] y su particular aplicación a través del ISML (Information Systems Modeling Language) [2], brinda a las organizaciones de la industria de la construcción de software, una ventaja competitiva que les permite: producir el código fuente de una plataforma específica en el menor tiempo posible. Esto se da gracias a la construcción de generadores de código, que pueden ser personalizados y ajustados con los estándares de la organización. Estos generadores encapsulan todos los detalles de implementación, que son comunes en una arquitectura MVC [10].

El proyecto DotNetGenerator tuvo como resultado un generador de código especializado en la generación de código fuente de aplicaciones ASP.NET MVC [1]. Este generador utiliza toda la infraestructura tecnológica que brinda ISML. El generador de código no pretende generar como resultado, una aplicación Web totalmente terminada y lista para ser desplegada en producción. El objetivo del generador es automatizar la mayor parte de componentes comunes, y reducir el esfuerzo que involucra escribir las líneas de código correspondientes a estos componentes.

DotNetGenerator facilita la generación de aplicaciones Web a partir de un modelo expresado en el lenguaje textual ISML. Permite la construcción de aplicaciones en menos tiempo y reduce los costos de producción del software. Facilita el desarrollo de un mínimo producto viable en corto plazo. Puede ajustarse a los estándares que requiera la empresa. Permite una abstracción de los detalles técnicos de la implementación de Entidades, Páginas y Controladores.

DotNetGenerator al igual que todos los generadores de código construidos bajo ISML, con el tiempo puede caer en una situación de obsolescencia sobre algunos de sus componentes. Esto se debe al constante cambio tecnológico de la industria. Es por esta razón que se recomienda a los usuarios de este sistema, mantener actualizadas las plantillas de sus generadores, con las últimas actualizaciones presentes en ASP.NET MVC. El generador contiene la estructura de un proyecto Web estándar, pero el sistema está diseñado para ajustarse a los cambios y actualizaciones ya sea a nivel de plantillas o generadores.

Existen varios trabajos futuros que pueden ser implementados para el generador de código. Uno de ellos es extender su funcionalidad para que también pueda generar código para la plataforma abierta ASP.NET Core [59]. Esta plataforma contiene gran parte de las funcionalidades presentes en ASP.NET MVC 5 [51], y otras están en proceso de construcción. Una ventaja de esta nueva plataforma es que su código fuente puede ser compilado y ejecutado en plataformas Linux, Mac y Windows. Otro trabajo Futuro para este proyecto es la construcción de un Wizard (Asistente Gráfico), que ayude a personalizar características del código fuente generado, y que facilite su integración con otros componentes de software existentes en la compañía, de una manera gráfica y más interactiva.

REFERENCIAS

[1] Microsoft, «Learn About ASP.NET MVC,» Microsoft, 2018. [En línea]. Available: https://www.asp.net/mvc. [Último acceso: 2 Mayo 2018].

[2] M. C. Franky, J. A. Pavlich Mariscal, J. C. Olarte, M. C. Acero, A. Zambrano, J. L. Camargo y J. N. Pinzón, «ISML: A language and MDE environment to model and generate web applications with integration of existing components,» de *Computing Colombian Conference (10CCC)*, 2015.

[3] S. Kelly y J. P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, Primera ed., Wiley-IEEE Computer Society Pr, 2008.

[4] Microsoft, «Visual Studio Community,» Mayo 2018. [En línea]. Available: https://www.visualstudio.com/vs/community/. [Último acceso: 01 Mayo 2018].

[5] Eclipse Foundation, «Eclipse Oxygen,» 2018. [En línea]. Available: https://www.eclipse.org/oxygen/. [Último acceso: 2 Mayo 2018].

[6] Microsoft, «Entity Framework 6,» 27 Octubre 2016. [En línea]. Available: https://docs.microsoft.com/en-us/ef/ef6/. [Último acceso: 1 Mayo 2018].

[7] Microsoft, «Getting Started with Entity Framework 6 Code First using MVC 5,» 10 Octubre 2015. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application. [Último acceso: 1 Mayo 2018].

[8] Microsoft, «Entity Framework Fluent API - Configuring and Mapping Properties and Types,» 23 Octubre 2016. [En línea]. Available: https://msdn.microsoft.com/en-us/library/jj591617(v=vs.113).aspx. [Último acceso: 1 Mayo 2018].

[9] W3Schools, «ASP.NET Razor - Markup,» 2018. [En línea]. Available: https://www.w3schools.com/asp/razor_intro.asp. [Último acceso: 1 Mayo 2018].

[10] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[11] S. W. Ambler y M. Lines, «Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise,» *IBM Press*, 2012.

[12] IBM, «Rational Team Concert for Scrum Projects: Scrum as a methodology,» 22 Noviembre 2010. [En línea]. Available: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Rational+Team+Concert+for+Scrum+Projects/page/SCRUM+como+metodolog%C3%ADa. [Último acceso: 3 Mayo 2018].

[13] D. C. Schmidt, «Model - Driven Engineering,» *IEEE Computer Society*, pp. 25-31, 2006.

[14] M. Brambilla, J. Cabot y M. Wimmer, "Model-Driven Software Engineering in Practice", Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2012.

[15] A. G. Kleppe, J. B. Warmer y W. Bast, MDA Explained. the Model Driven Architecture: Practice and Promise, New York: Addison-Wesley, 2003.

[16] Object Management Group OMG, «About Object Management Group OMG,» 2018. [En línea]. Available: http://www.omg.org/about/index.htm. [Último acceso: 1 Mayo 2018].

[17] Heinsohn Business Technology HBT, «Heinsohn Business Technology Web Site,» 2018. [En línea]. Available: https://www.heinsohn.com.co/. [Último acceso: 2 Mayo 2018].

[18] A. Cuesta M, M. Lopez T y L. Joyanes A, Comparativo de herramientas MDA (AndroMDA, ArcStyler, OptimalJ)., Universidad de Caldas, 2009.

[19] MagicDraw, «MagicDraw Intro,» 2018. [En línea]. Available: https://www.nomagic.com/products/magicdraw. [Último acceso: 1 Mayo 2018].

[20] Gentleware, «Poseidon for UML,» 2010. [En línea]. Available: http://www.gentleware.com/products.html. [Último acceso: 1 Mayo 2018].

[21] Sparx Systems Pty Ltd., «Introduction to Enterprise Architect,» [En línea]. Available: http://www.sparxsystems.com.au/enterprise_architect_user_guide/9.3/index.html. [Último acceso: 2 Mayo 2018].

[22] Object Management Group, «MetaObject Facility Specification,» 2018. [En línea]. Available: https://www.omg.org/mof/. [Último acceso: 1 Mayo 2018].

[23] Technische Universität Dresden , «OCL Portal,» 2018. [En línea]. Available: http://www-st.inf.tu-dresden.de/ocl/. [Último acceso: 1 Mayo 2018].

[24] Eclipse Foundation, «ATL - a model transformation technology,» 2018. [En línea]. Available: http://www.eclipse.org/atl/. [Último acceso: 1 Mayo 2018].

[25] The Apache Software Foundation, «The Apache Velocity Project,» 2016. [En línea]. Available: http://velocity.apache.org/. [Último acceso: 3 Mayo 2018].

[26] The Apache Software Foundation, «FreeMarker Java Template Engine,» 2018. [En línea]. Available: https://freemarker.apache.org/. [Último acceso: 1 Mayo 2018].

[27] M. Völter, «openArchitectureWare - a flexible Open Source platform for model-driven software development,» 2006. [En línea]. Available: http://www.voelter.de/data/workshops/EtxMarkusVoelter.pdf. [Último acceso: 1 Mayo 2018].

[28] T. Stahl y M. Völter, Model-Driven Software Development, John Wiley & Sons Ltd, 2006.

[29] Eclipse Foundation, «Eclipse Modeling Framework (EMF),» Eclipse Modeling Project, 2018. [En línea]. Available: https://www.eclipse.org/modeling/emf/. [Último acceso: 3 Mayo 2018].

[30] Eclipse Foundation, «Eclipse Xpand,» 22 Junio 2016. [En línea]. Available: https://projects.eclipse.org/projects/modeling.m2t.xpand. [Último acceso: 1 Mayo 2018].

[31] Eclipse Foundation, «Acceleo Portal,» 22 Febrero 2018. [En línea]. Available: http://wiki.eclipse.org/Acceleo. [Último acceso: 1 Mayo 2018].

[32] Object Management Group, «About the MOF Model to Text Transformation Language Specification Version 1.0,» 2008. [En línea]. Available: https://www.omg.org/spec/MOFM2T/1.0/. [Último acceso: 1 Mayo 2018].

[33] Microsoft, «Code Generation and T4 Text Templates,» 2015. [En línea]. Available: https://msdn.microsoft.com/en-us/library/bb126445.aspx. [Último acceso: 2 Mayo 2018].

[34] Microsoft, «Información general sobre ASP.NET,» 1 noviembre 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/4w3ex9c2(v=vs.100).aspx. [Último acceso: 1 Mayo 2018].

[35] Microsoft, «SQL Server 2016 Express LocalDB,» Microsoft Docs, 10 Agosto 2018. [En línea]. Available: https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/sql-server-2016-express-localdb?view=sql-server-2017. [Último acceso: 2 Mayo 2018].

[36] Microsoft, «NHibernate architecture,» [En línea]. Available: http://nhibernate.info/doc/nhibernate-reference/architecture.html. [Último acceso: 1 Mayo 2018].

[37] Microsoft, «ADO.NET,» 30 Marzo 2017. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/. [Último acceso: 1 Mayo 2018].

[38] Microsoft, «Entity Framework Code First Data Annotations,» 23 Octubre 2016. [En línea]. Available: https://msdn.microsoft.com/en-us/library/jj591583(v=vs.113).aspx. [Último acceso: 1 Mayo 2018].

[39] Microsoft, «Microsoft SQL Server 2017 Express,» 2018. [En línea]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=55994. [Último acceso: 1 Mayo

2018].

[40] Microsoft, «Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10),» Microsoft Docs, 30 Julio 2013. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application. [Último acceso: 1 Mayo 2018].

[41] Microsoft, «IIS Express Overview,» Microsoft Docs, 6 Julio 2010. [En línea]. Available: https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview. [Último acceso: 6 Mayo 2018].

[42] Xomega.Net, «What is Xomega,» 2018. [En línea]. Available: http://www.xomega.net/Default.aspx. [Último acceso: 1 Mayo 2018].

[43] Microsoft, «N-Tier Data Applications Overview,» [En línea]. Available: https://msdn.microsoft.com/en-us/library/bb384398.aspx. [Último acceso: 1 Mayo 2018].

[44] Microsoft, «MSBuild,» 2015. [En línea]. Available: https://msdn.microsoft.com/en-us/library/dd393574.aspx. [Último acceso: 1 Mayo 2018].

[45] Microsoft, «Introducción a WPF,» 2018. [En línea]. Available: https://msdn.microsoft.com/es-es/library/aa970268(v=vs.100).aspx. [Último acceso: 2 Mayo 2018].

[46] Microsoft, «ASP.NET Web API,» 2018. [En línea]. Available: https://msdn.microsoft.com/es-es/library/hh833994(v=vs.108).aspx. [Último acceso: 3 Mayo 2018].

[47] Microsoft, «What Is Windows Communication Foundation,» 30 Marzo 2017. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf. [Último acceso: 2 Mayo 2018].

[48] Exotechnic Corporation (UK) Ltd, «CodeTrigger - Code Generation For C#, WPF, WCF, SQL SERVER/ORACLE/MYSQL and Visual Studio 2013-2017,» 20 Noviembre 2017. [En línea]. Available: https://www.codetrigger.com/Default.aspx?vwsess=68491. [Último acceso: 1 Mayo 2018].

[49] Microsoft, «Getting Started with Windows Forms,» 30 Marzo 2017. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/framework/winforms/getting-started-with-windows-forms. [Último acceso: 4 Mayo 2018].

[50] DotNetGenerator, «Repositorio DotNetGenerator,» 1 Mayo 2018. [En línea]. Available: https://github.com/sanchezjl/DotNetGenerator. [Último acceso: 3 Mayo 2018].

[51] Microsoft, «What's New in ASP.NET MVC 5,» Microsoft Docs, 20 Enero 2015. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/mvc5. [Último acceso: 1 Mayo 2018].

[52] NuGet, «Simple Injector MVC Integration Quick Start,» 2017. [En línea]. Available: https://www.nuget.org/packages/SimpleInjector.MVC3/. [Último acceso: 1 Mayo 2018].

[53] Microsoft, «DbContext Class,» 2018. [En línea]. Available: https://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext(v=vs.113).aspx. [Último acceso: 1 Mayo 2018].

[54] Microsoft, «DbSet Class,» Microsoft Developer Network Documentation, 2018. [En línea]. Available: https://msdn.microsoft.com/en-us/library/system.data.entity.dbset(v=vs.113).aspx.

[Último acceso: 1 Mayo 2018].

[55] Microsoft, «DbContext.OnModelCreating Method (DbModelBuilder),» Microsoft Developer Network, 2018. [En línea]. Available: https://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext.onmodelcreating(v=vs.113).aspx. [Último acceso: 1 Mayo 2018].

[56] Eclipse Foundation, «Xtext Language Engineering For Everyone!,» 2018. [En línea]. Available: https://www.eclipse.org/Xtext/. [Último acceso: 2 Mayo 2018].

[57] Terence Parr, «What is ANTLR?,» 2014. [En línea]. Available: http://www.antlr.org/. [Último acceso: 6 Mayo 2018].

[58] Eclipse Foundation, «Xtend Documentation,» 2018. [En línea]. Available: http://www.eclipse.org/xtend/documentation/index.html. [Último acceso: 1 Mayo 2018].

[59] Microsoft, «Introduction to ASP.NET Core,» Microsoft Docs, 28 Febrero 2018. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.0. [Último acceso: 1 Mayo 2018].