Estrategia de migración de un Sistema Legado utilizando Chicken Little

Práctica final EM - UPM

Luis Mata Aguilar - bm0613

2019

Contents

List of Figures

Resumen

1. Introducción

Este documento describe la estrategia seguida para realizar el plan de migración del sistema legado de gestión de asignaturas de la ETSISI suministrado. Este sistema se considera crítico para el correcto funcionamiento de la docencia en esta escuela, por lo que durante la migración no podrá estar caído más de 24h. El sistema lleva el registro de alumnos que hay matriculados en la escuela por curso así como su asignación en grupos de clase. Asímismo trata la asignación docente de profesores y lleva el control de las tutorías de los profesores, entre otras funcionalidades.

Debido al cariz crítico de esta aplicación para con los docentes de la escuela, el equipo encargado de la migración, tomó la decisión de realizar una migración gradual y con el menor impacto posible sobre los datos originales, reduciendo así también el tiempo de caída del servicio original.

2. Resumen de metodologías para la migración de sistemas legados

El principal problema que acontece la migración a una nueva arquitectura es la migración de datos, lo cual lleva un tiempo significativo y durante este tiempo el sistema legado es inservible. Se ha impuesto como restricción que se debe reducir al máximo en la medida de lo posbile dicho tiempo. El sitema legado podrá seguir funcionando hasta que los Stakeholders del proyecto confirmen el nuevo servicio, momento en el que la aplicación legada dejará de funcionar y su base de datos quedará inaccesible.

Tras un análisis exhaustivo de todos los métodos de migración vistos en clase, se ha llegado a la conclusión de que el mejor método para nuestro caso es “Chicken Little” por diversos motivos.

- “Chicken Little” es el mejor método para migraciones críticas en las que no se puede tener congelado el sistema legado.
- El sistema migra de forma gradual, tanto datos como funcionalidad, lo cual favorece un trabajo iterativo en sprints de funcionalidades atómicas.
- “Chicken Little” minimiza el tiempo en el que el sistema legado es inaccesible.

2.1 Otras metodologías estudiadas

- **“Cold Turkey”**: Propone una migración masiva en una única operación. Esta estrategia tiene un riesgo elevado ya que supone poner todo en riesgo a una sola acción. No contempla un trabajo incremental en funcionalidades, y el sistema legado deja de funcionar mientras se realiza la

operación masiva. Al tratarse de millones de filas de información, se descarta esta opción debido al consumo de tiempo y riesgo.

- **“Butterfly”**: Propone migrar un sistema legado a un sistema destino de forma simple, rápida y segura. En esta metodología se elimina la necesidad de acceder a ambos sistemas de forma simultánea, por lo que se ve reducida la complejidad de la migración al no tener que mantener consistencia entre ambos sistemas. Toda migración se hace en una operación masiva como en “Cold Turkey” y por ello existe un riesgo elevado de pérdida de información. Por este motivo queda descartada.
- **“Database First”**: Propone que la base de datos sea lo primero en migrar, y de forma incremental el resto del sistema. Esta metodología se apoya en un Forward Gateway que permite a la aplicación legada acceder a la nueva base de datos, de forma que no se pierda el servicio. Este Gateway traduce y redirige las llamadas de la aplicación legada a la nueva base de datos y los resultados de las consultas a la aplicación legada. El uso de este tipo de metodologías hace que el sistema legado sea más lento en el proceso de migración y convella más tiempo de desarrollo debido a la codificación del Gateway, por ello queda descartada.
- **“Database Last”**: Propone que la base de datos sea lo último en migrar, y qu de forma gradual se vaya migrando primero el resto de funcionalidades del sistema mientras la base de datos legada permanece en la plataforma original. En este caso se usaría el concepto contrario al gateway anterior, un “Reverse Gateway” cuya función es permitir a las nuevas aplicaciones acceder a la base de datos orginal, traduciendo peticiones en ambas direcciones. El ueso de este tipo de metodologías, como en el caso anterior, llevan más tiempo de desarrollo y empobrecen el desempeño del sistema global, por lo que queda descartada.

3. Propuesta para la implantación del nuevo sistema

3.1 Arquitectura objetivo

Tras realizar un estudio de los requisitos del cliente en cuanto al nuevo sistema, podemos hablar de una nueva arquitectura de comunicaciones y tecnología. Se trata de un sistema web ahora, bajo un arquitectura MVC con tecnología ASP .NET de Microsoft. El punto fuerte de este tipo de sistema es su accesibilidad universal desde dispositivos como smartphones, tables, ordenadores personales y cualquier dispositivo con acceso a internet y pantalla. Se dispone de un acceso por usuarios con login desde cualquier parte.

El nuevo sistema hospedará la nueva base de datos en los servidores de la escuela(o Azure) y podrá ser gestionado por los administradores, tal y como se comentó en la sesión de extracción de requisitos.

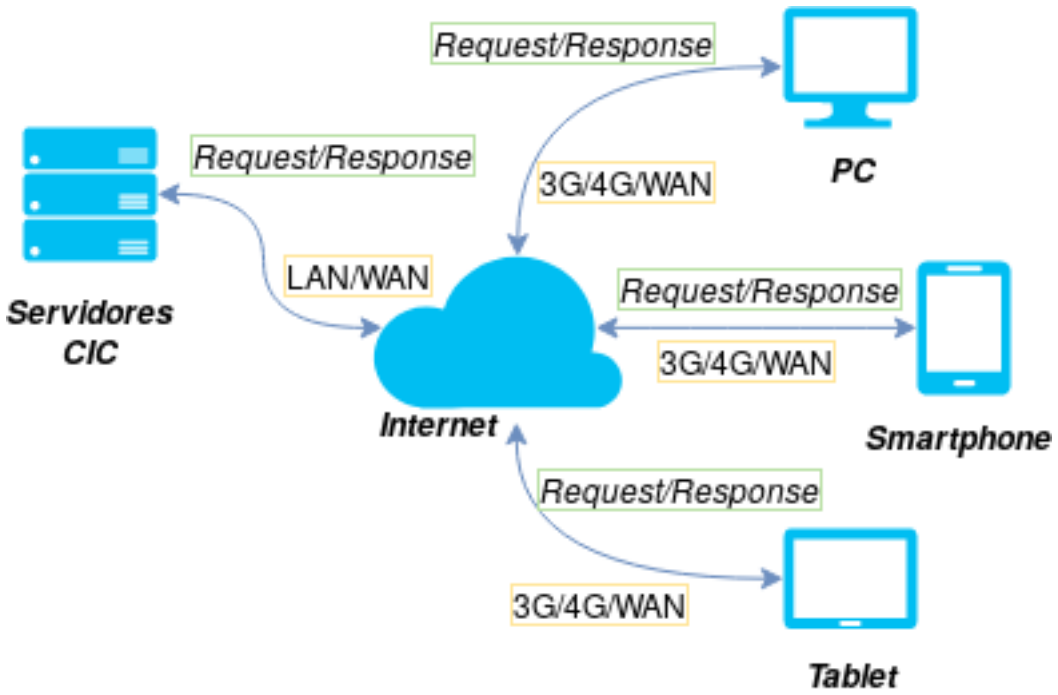

Figure 1: Diagrama de despliegue

3.2 Modelo lógico

Nuevo sistema lógico para dar soporte al servicio, tal y como se plantea en el docuemnto de requisitos.

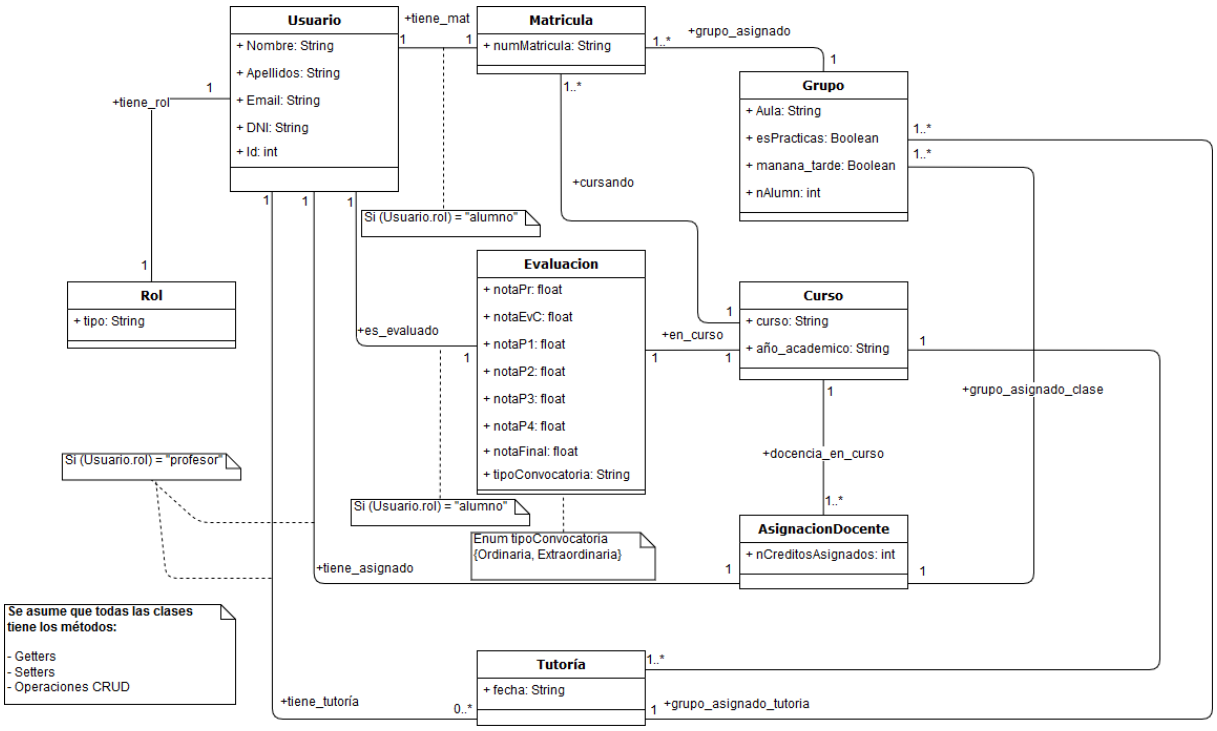

Figure 2: Diagrama de clases

4. Proceso de migración

Para llevar a cabo la migración del sistea legado se van a utilizar varias etapas incrementales en funcionalidad, siguiendo la guía MVC ASP .NET subida en la plataforma moodle de la asignatura y adaptándola a nuestras necesidades. Se trabajará con Visual Studio 19 y ASP .NET.

Tal y como se ha planteado, al usar la metodología Chicken Little, vamos a ir incluyendo funcionalidades en cada Sprint de trabajo.

4.1 Primera etapa

4.1.1 Planificación

En esta etapa se empieza el proyecto de la nueva aplicación, el cual recibe de nombre AppGestionAsignaturas. Se migran los usuarios con modelo modificado a la nueva base de datos. Se crean roles y dos clases base.

4.1.2 Resultados

1. Se ha creado el proyecto con Vistual Studio 19 usando como plantilla ASP .NET web services como plantilla y hemos seleccionado que para autenticar usuarios se use ASP.NET Identity, lo cual permite un login personal y soporta OAuth y OpenID.
2. Creamos los atributos adicionales para los usuarios ya que por defecto se tienen una serie de campos que nos serán útiles como email. Nosotros añadimos DNI y nombre y apellidos.
3. Tras compilar, y al usar EntityFramework; podemos activar las migraciones por la consola del administrador de paquetes del proyecto, usando para ello un paradigma DatabaseFirst. Se trata de migraciones incrementales pero debemos de activarlas, para ello:

```
PM> enable-migrations
```

4. Se crea un fichero de configuración donde insertamos los usuarios de prueba así como los nuevos roles de nuestro servicio, los cuales incluyen:
    - Profesores: Docentes que usarán el servicio para gestionar evaluaciones, grupos y tutorías.
    - Profesores de prácticas: Docentes encargados de grupos de prácticas.
    - Administradores: Encargados de gestionar la asignación docente de los profesores y las matrículas de los alumnos. Asimismo crearán los cursos y otras entidades básicas.
    - Alumnos: Usuarios que podrán consultar sus calificaciones en nuestro nuevo servicio.

Después de configurar las migraciones conforme a lo especificado en requisitos, creamos la primera migración para los usuarios modificados, para ello, en el administrador de paquetes:

```
PM> add-migration ModifyUsers
PM> update-database
```

5. Se crea la clase Cursos, su controlador y se añade su migración de forma similar.
    - Se incluyen atributos Curso (String) y AñoAcadémico (String).
6. Se crea la clase Grupos, su controlador y se añade su migración de forma similar.
    - Se incluyen atributos Aula (String), esPracticas(boolean), Mañana/Tarde (String) y nAlum(int).

4.2 Segunda etapa

4.2.1 Planificación

En esta etapa se continúa codificando las nuevas clases de la aplicación mientras se van migrando proceduralmente. Se pretende tener formadas las clases relación de AsignacionDocentes y Matricula, ya que son parecidas para la asignación a cursos y grupos de profesores y alumnos respectivamente.

4.2.2 Resultados

1. Se crea la clase AsginacionDocentes en Modelos con los siguientes atributos:
    - nCreditosAsignados (int): Para asignar un número de créditos entero a los docentes asignados a un grupo en un curso.
2. Se indexa por clave principal compuesta esta clase haciendo referencia a Curso con CursoId; Grupos con GrupoId y Usuario con UserId.
3. En el proceso de scaffolding (cuando creamos el controlador), detecta que no se tiene el DBSet de ApplicationUser y lo genera pero borramos esta línea de código porque ya se gestiona con la plantilla que usamos. Corregimos los errores en el controlador y rectocamos los métodos para poder interactuar con ellos desde los ActionButton de las vistas asociadas.
4. Añadimos una sentencia de autorización para que solo los administradores puedan hacer uso de esta funcionalidad.

```
[Authorize(Roles = "admin")]
```

4. Creamos la migración:

```
PM> add-migration CreateAsignacionDocentes
PM> update-database
```

Además solo se podrán asignar profesores a grupos por curso, para lo cual incluimos un filtrado por rol en el controlador, dentro del método de creación, este mecanismo se reutiliza para otras clases.

5. Repetimos el proceso para Matrícula, con sus respectivas restricciones.
    - Solo pueden asignarse a alumnos, se realiza un filtrado por roles para la creacion de matrículas.

4.3 Tercera etapa

4.3.1 Planificación

En esta etapa se contempla terminar de añadir funcionalidades y migrar los datos con EF. Se modifican las vistas del servicio web para hacerlo más amigable. Se incluyen los botones de navegación y se modifica la lógica de los controladores pertinentes para hacer lo especificado en requisitos.

4.3.2 Resultados

1. Clases creadas en este punto:

    - Tutorias (mismo procedimiento que AsignacionDocente
    - Evaluaciones
2. Controladores creados en este punto:
    - TutoriasController
    - EvaluacionesController
    - MisEvaluacionesController
3. Este último controlador permite que los alumnos puedan ver sus notas, ya que se hace un filtrado de usuarios al alumno que esté identificado en ese momento para poder mostrar sus datos. Éste no puede modificar ni eliminar nada, solo los detalles de sus notas.
4. Para las tutorías se realiza el mismo proceso que con AsignacionDocentes.
5. Se asignan los botones de navegación para cada caso de uso y una breve descripción así como un botón-link a la página de la escuela.

4.4 Cuarta etapa

En esta etapa se prueban las funcionalidades de forma aislada y se sube el código total al repositorio asignado de GitHub. Desde aquí podría ser desplegada la aplicación con las tecnologías aprendidas en la primera parte de la asignatura, de forma que el tiempo invertido en llevar el código a un servidor dedicado es mínimo.

- Repetimos etapas anteriores en caso de fallar alguna funcionalidad y corregimos.
- Estas correcciones han supuesto un tiempo considerable del desarrollo.
- Se trataba especialmente de la interactuación por clave con los botones de ActionLink en las vistas de las clases que tenían clave primaria compuesta.

5. Conclusiones

Muchos profesionales del mundo de las ciencias de la computación (ej. Guido Van Rossum) han expresado ya su precupación al ver que los alumnos de estas titulaciones recién graduados no tienen contacto ni consciencia de los llamados *“Systems in the large”*.

En nuestra opinión personal, creemos que esta práctica así como la asignatura en general ha aportado conocimiento sustancial al entorno que rodea a estos sistemas, especialmente a la forma de trabajar colaborativamente y usar métricas para ello.

Hemos aprendido a realizar ingeniería inversa a un sistema y a planificar una migración a otra tecnología. También hemos trabajado con pipelines Dev/Ops (primera parte de la asignatura). Estas tecnologías

están cerca de lo que los expertos llaman programación 2.0, alejándose de las implicaciones a bajo nivel de código para pasar a un mundo de diseño y análisis a alto nivel del que deriva todo el peso de desarrollo.

Creemos que esta metodología basada en proyectos es la mejor para aprender a desenvolverse en un mundo que utilice estas tecnologías, ya que de otra forma todo el peso de la prueba de aprendizaje quedaría reducida a un examen.

6. Referencias

- Ficheros de enunciado y ayudas para la práctica. Moodle
- Ejemplos de documentos:
  - SMART ENERGY FOR YOUR HOME REQUIREMENTS DEFINITION DOCUMENT (Marius Schlinke, Elena Kyorova, Nathan Chape)
  - SMART ENERGY FOR YOUR HOME DESIGN DESCRIPTION (Marius Schlinke, Elena Kyorova, Nathan Chape)
- Wikipedia Chicken Little
- Wikipedia Software Maintenance
- ISO/IEC 14764:2006: Software Engineering – Software Life Cycle Processes – Maintenance