Aplicación de integración de búsqueda de texto completo en bases de datos NoSQL

Application integration full text search in NoSQL databases

Marvin Ramírez Valenzo*

René Cuevas Valencia**

Fecha de recepción: 1 de mayo de 2013
Fecha de aceptación: 5 de mayo de 2013

Resumen

Este proyecto describe el desarrollo de una aplicación usando un motor de búsqueda de texto completo con sistema de índice invertido, en información contenida en una base de datos NoSQL. Con referencia un trabajo previo titulado "Integración de búsquedas de texto completo en Bases de Datos NoSQL" publicado en la Revista Vínculos, Volumen 8, Numero 1, Enero-Junio 2011, pagina 81 – 92. [1].

Palabras clave: NoSQL, Bases de Datos, MongoDB, Full Text Search, Lucene.net, búsqueda de índice invertido.

* Universidad Autónoma de Guerrero. Unidad Académica de Ingeniería. Chilpancingo Guerrero; México. Correo electrónico: valenzo@gmail.com

** Universidad Autónoma de Guerrero. Unidad Académica de Ingeniería. Chilpancingo Guerrero; México. Correo electrónico: reneecuevas@hotmail.com

Abstract

This project describes the development of an application using a search engine full-text inverted index system, on information contained in a NoSQL database. With reference previous work entitled "Integration of full text searches NoSQL Databases" in the Journal Links, Volume 8, Number 1, January-June 2011, pages 81-92. [1].

Key words: NoSQL databases, MongoDB, Full Text Search, Lucene.net, inverted index search.

1. Introducción

El trabajo actual consiste en la integración de dos tecnologías para el mejor almacenamiento y búsqueda de información, la primera es la de búsqueda de texto completo que tiene como objetivo realizar las búsquedas contra los datos de texto en índices de texto completo sobre palabras y frases basándose en las reglas de datos determinados y la otra tecnología son las bases de datos NoSQL, que son una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS), las NoSQL no usan SQL como el principal lenguaje de consultas [2].

Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, coherencia, aislamiento y durabilidad), y escalan bien horizontalmente por lo que usan menos recursos. También las bases de datos NoSQL están altamente optimizadas para las operaciones de recuperar y agregar, tienen características significativas en escalabilidad y rendimiento en los modelos de datos. [3].

Se pretende demostrar que al tener una gran cantidad de datos es necesario de un sistema de búsqueda potente con acceso en tiempo real y máximo rendimiento, por lo tanto se necesita integrar una Base de datos NoSQL con búsquedas de texto completo, aprovechando las ventajas de ambas tecnologías.

2. Herramientas a usar

La aplicación a desarrollar es un blog, que es un sitio web periódicamente actualizado que recopila cronológicamente textos de uno o varios autores, el blog es el tipo de aplicación ideal para probar grandes volúmenes de información y su búsqueda en ellos [3].

2.1. MongoDB

Se seleccionó a MongoDB porque tiene la combinación de las mejores características de las bases de datos JSON, almacenamiento clave/valor y RDBMS [4].

MongoDB es una base de datos de alto rendimiento, de código abierto y de esquema-libre orientada a documentos (schema-free document-oriented). MongoDB está escrito en C++ y ofrece las siguientes características:

- Almacenamiento orientado a documento (la simplicidad y el poder datos de esquemas de tipo JSON)
- Consultas dinámicas
- Soporte completo de índices, incluido el interior de inner objects y cadenas embebidas
- Consulta de perfiles (Query profiling)
- Replicación y soporte de fail-over
- Almacenamiento eficaz de datos binarios incluidos objetos grandes (por ejemplo, vídeos).
- Auto-sharding de escalabilidad cloud-level (en la actualidad, en etapa alfa).
- MapReduce para agregación compleja Soporte Comercial Disponible [3].

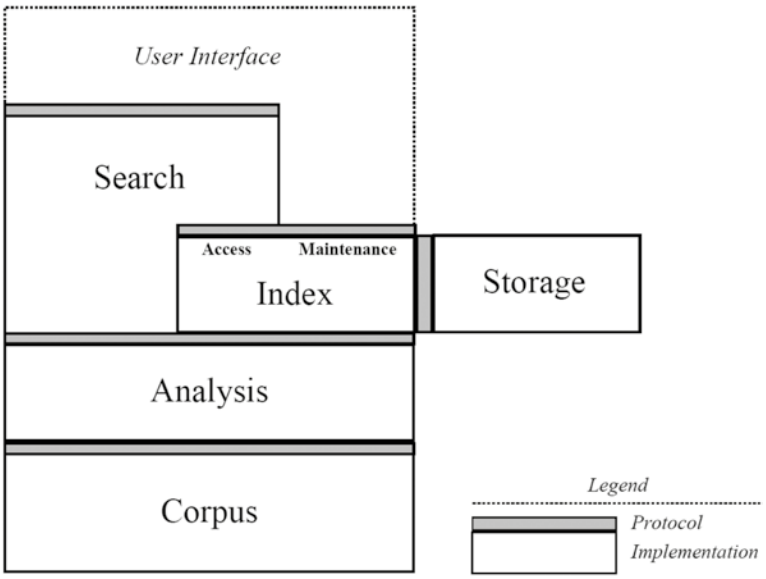
2.2. Lucene.Net

Se usa Lucene.Net como la herramienta de búsqueda de texto completo, Lucene.Net es un puerto de la biblioteca del motor de búsqueda Lucene, escrito en C # y está dirigida a los usuarios de tiempo de ejecución.

La biblioteca de búsqueda Lucene se basa en un índice invertido. Lucene.Net tiene tres objetivos principales:

- Mantener el actual puerto, línea por línea de Java a C #, la automatización completa y mercantilización del proceso de tal manera que el proyecto se puede sincronizar fácilmente con el programa Java Lucene.
- El mantenimiento de los requisitos de alto rendimiento que se espera de una primera biblioteca del motor de búsqueda de clase C #;
- Maximizar la facilidad de uso y potencia cuando se utiliza dentro del. NET. Para ello, presentará una API muy idiomático, cuidadosamente adaptados que se aprovecha de muchas de las características especiales de la. NET [5].

Figura 1. Arquitectura de Lucene.Net



Fuente: elaboración propia.

2.3. Lenguaje C#

El lenguaje para desarrollar esta aplicación es C#, que es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux. [6].

3. Diseño y estructura de la aplicación

En MongoDB se crean collections que son los equivalentes a una tabla DBMS, la colección que se crea se llama artículos que tiene la siguiente estructura en formato BSON:

Figura 2. Estructura de la aplicación en formato BSON.

```
Colletions
articulos
{
  "_id" : ObjectId(""),
  "titulo" : ,
  "texto" : ,
  "fecha" : ,
  "Tags" : [{
      "tag" :
    }],

  "Comentarios" : [{
      "_id" : ObjectId(""),
      "fecha" : ,
      "usuario" : ,
      "texto" :
    }]
}
```

Fuente: elaboración propia.

Donde un artículo tiene los campos id, titulo, texto, fecha, Tags y Comentarios. Los Tags son una colección de tag y Comentarios, es una colección que tiene id, fecha, usuario y texto.

Un registro de un artículo de la siguiente manera:

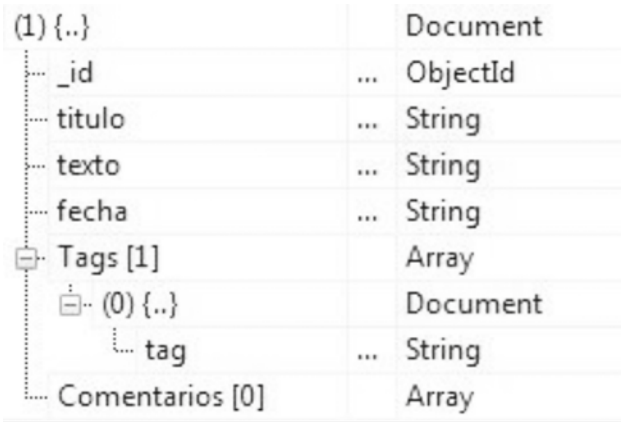
Figura 3. Ejemplo de registro en estructura BSON

```
{
  "_id" : ObjectId("51a3cc4905dd310404443af4"),
  "titulo" : "Boleto",
  "texto" : "Compra tu boleto en la Próxima hora
             con # ÚltimaLlamada y vuela a Hermosillo,
             Ciudad de México o Oaxaca. http:// vuela.am/1613zGX ",
  "fecha" : "27/05/2013 04:12:41 p.m.",
  "Tags" : [{
      "tag" : "Compra"
    }],
  "Comentarios" : [{
      "_id" : ObjectId("000000000000000000000000"),
      "fecha" : "27/05/2013 04:14:06 p.m.",
      "usuario" : "marvin ramirez valenzo",
      "texto" : "Buen trabajo"
    }]
}
```

Fuente: elaboración propia.

Esta estructura en formato BSON es capaz de soportar nuestra aplicación.

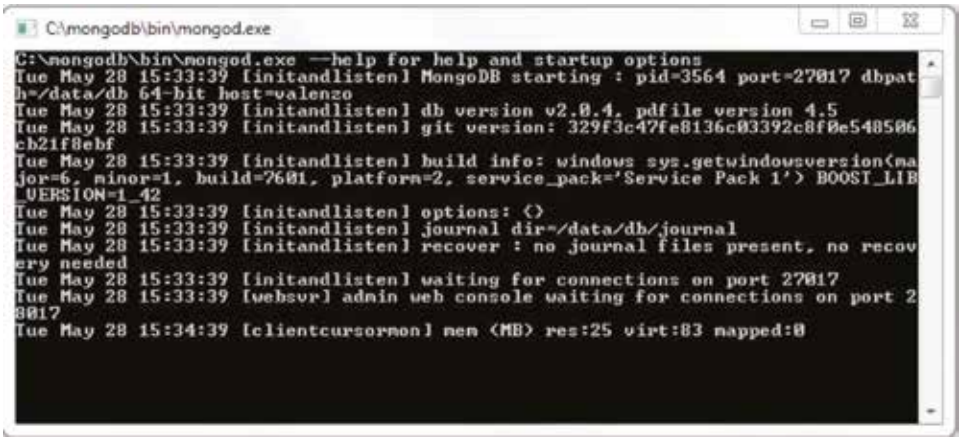
Figura 4. Arquitectura del Blog.



Fuente: elaboración propia.

Se instala mongodb en un ambiente Windows, y ejecutar el servicio, mongodb usa el puerto 27017 :

Figura 5. Servicio en ejecución de MongoDB.



Fuente: elaboración propia.

3.1. Descargar Lucene

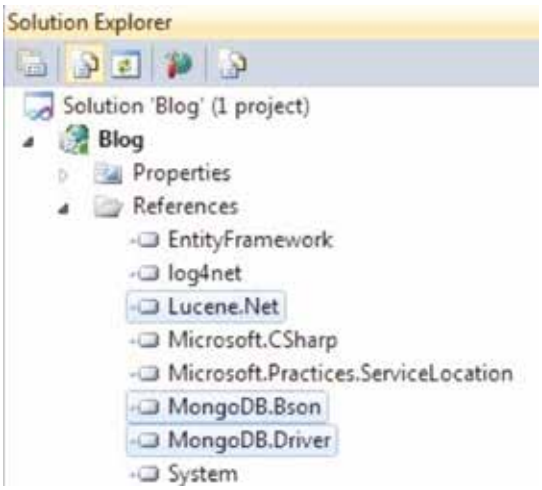
Descargar e instalar Lucene.net desde http://lucenenet.apache.org/ el motor de búsqueda Lucene.net es un DLL que se utiliza para construir y buscar en los índices. Lucene.NET puede indexar cualquier información basada en texto y luego encontrarlo más adelante basa en varios criterios de búsqueda. Lucene.NET tiene una función de búsqueda muy potente y flexible que utiliza la lógica difusa para localizar artículos indexados [5].

3.2. Integración de Lucene.Net Y MongoDB

Integrar Lucene.Net y MondoDB en nuestro proyecto de desarrollo en C# se Agrega una referencia a las DLLs del controlador C# y al Driver MongoDB

- MongoDB.Bson.dll
- MongoDB.Driver.dll
- Lucene.Net

Figura 6. Integración de MongoDB y Lucene.NET



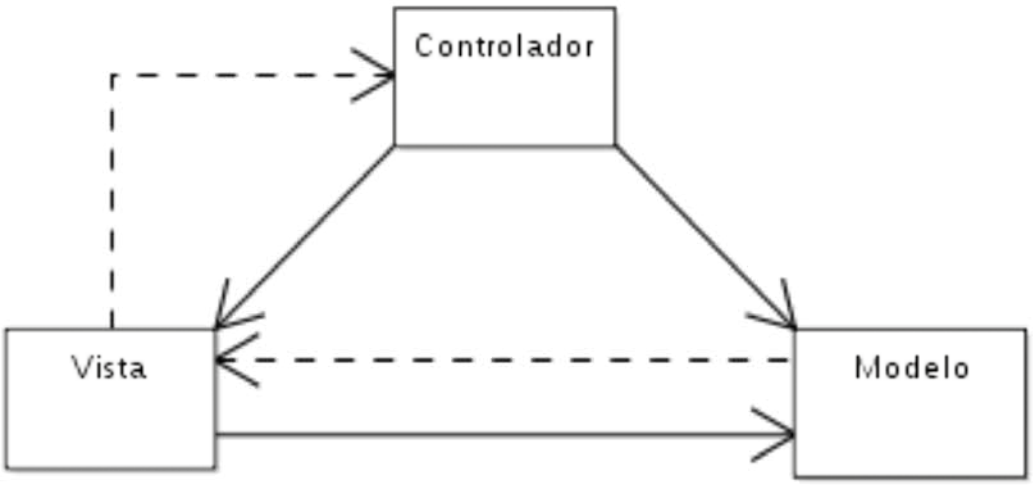
Fuente: elaboración propia.

4. Desarrollo de la aplicación

La aplicación desarrollada es de tipo Modelo-Vista-Controlador, el Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica

de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario [6].

Figura 7. Diagrama de trabajo de la arquitectura MVC



Fuente: elaboración propia.

4.1. Creación del modelo

El Modelo es la representación específica de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'. [6].

4.1.1. Creamos la conexión a MongoDB y la base de datos

Esto lo relacionamos en C# creando la clase ArticulosModel.cs donde tenemos el método Db() de tipo MongoDatabase.

```
using MongoDB.Bson.IO;
using MongoDB.Bson.Serialization;
using MongoDB.Bson.Serialization.
Attributes;
using MongoDB.Bson.Serialization.
Conventions;
using MongoDB.Bson.Serialization.
IdGenerators;
using MongoDB.Bson.Serialization.
Options;
using MongoDB.Bson.Serialization.
Serializers;
using MongoDB.Driver.Builders;
using MongoDB.Driver.GridFS;
using MongoDB.Driver.Wrappers;
using MongoDB.Driver;
using MongoDB.Bson;
namespace Blog.Models
{
   public class ArticulosModel
   {
private MongoDatabase Db()
```

```
    {
      string connectionString =
"mongodb://localhost";
      MongoServer server = Mongo-
Server.Create(connectionString);
      MongoDatabase db = server.
GetDatabase("mydb");
      return db;
    }
```

Con este método hacemos la conexión a nuestro servidor MongoDB y generamos la base de datos llamada "mydb".

4.1.2. Obtener el listado de artículos

Se crea la función que nos devuelva el listado de los artículos de nuestro blog, con el siguiente código:

```
public IEnumerable<Articulo> Ob-
tArticulos()
  {
    MongoDatabase db = Db();
    MongoCollection<Articulo> Ar-
ticulos = db.GetCollection<Articul
o>("articulos");
    return Articulos.FindAll().
ToList<Articulo>();
  }
```

4.1.3. Control de la inserción de artículos

Se realiza el control de la inserción de cada artículo con el siguiente código:

```
public void Insertar(Articulo ar-
ticulo)
  {
    MongoDatabase db = Db();
    MongoCollection<Articulo> ar-
ticulos = db.GetCollection<Articul
o>("articulos");
    articulos.Insert(articulo.
ToBsonDocument());
  }
```

4.1.4. Búsqueda y detalle de un artículo

Se realiza una búsqueda en MongoDb por artículo por su GUID que es su llave primaria, con el siguiente código:

```
  public Articulo Detalle(string
Guid)
  {
    MongoDatabase db = Db(); re-
turn db.GetCollection<Articulo>("
articulos").FindOneById(ObjectId.
Parse(Guid));
  }
```

4.1.5. Borrar un artículo

Se realiza el código para poder eliminar un artículo.

```
  public void Borrar(string Guid)
  {
    MongoDatabase db = Db();
    db.GetCollection<Articulo>("
articulos").FindAndRemove(Query.
EQ("_id", ObjectId.Parse(Guid)),
SortBy.Descending("id"));
  }
```

4.1.6. Modificar un artículo

Se realiza el código par apoder eliminar un artículo.

```
  public void Modificar(string
Guid)
  {
    MongoDatabase db = Db();
    db.GetCollection<Articulo>("
articulos").FindOneById(ObjectId.
Parse(Guid));
  }
```

4.1.7. Insertar un comentario

Se realiza el código para poder eliminar un artículo.

```
  public void
CrearComentario(string Guid, Co-
mentario comentario)
  {
    MongoDatabase db = Db();
    var articulos = db.GetCollect
ion<Articulo>("articulos");
    Articulo a = articulos.
FindOneById(ObjectId.Parse(Guid));
    a.Comentarios.
Add(comentario);
    articulos.Save<Articulo>(a);
  }
```

4.2. Crear el controlador

El Controlador responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' [6].

```
using System.Web;
using System.Web.Mvc;
using Blog.Models;
using MongoDB.Driver.Wrappers;
using MongoDB.Driver;
using MongoDB.Bson;
using Lucene.Net.Analysis.Tokenat-
tributes;
using Lucene.Net.Index;
using StandardAnalyzer = Lucene.
Net.Analysis.Standard.StandardA-
nalyzer;
using IndexWriter = Lucene.Net.
Index.IndexWriter;
using Blog.Lucene;
using System.IO;
using Blog.ViewModels;

namespace Blog.Controllers
{
  public class ArticulosController
: Controller
  {
```

4.2.1. Crear un nuevo post

Código para crear un nuevo post:

```
//Crear un nuevo post
  [HttpPost]
  public ActionResult
Create(FormCollection form)
  {
    Articulo articulo = new Arti-
culo();
    articulo.titulo =
form["titulo"];
    articulo.texto =
form["texto"];
    articulo.fecha = Convert.
ToString(DateTime.Now);
    string tags = form["tags"];
    char[] delimitadores = { ' ',
',', '.', ';' };
    string[] TagsString = tags.
Split(delimitadores);
    //articulo.guid = articulo.
id.ToString();

    foreach (String tagtexto in
TagsString)
    {
    articulo.Tags.Add(new Tag() {
tag = tagtexto });
 }

    ArticulosModel model = new
ArticulosModel();
 model.Insertar(articulo);

    //Agregamos tambien el arti-
culo al index de Lucene
```

```
     AddToIndex(articulo);

     //Mandamos a la pagina
     return
RedirectToAction("Index");
   }
```

Crear un nuevo artículo y volver a la página principal.

4.2.2. Consulta de un articulo

Código para realizar una consulta de artículo:

```
// [HttpGet]
  public ActionResult
Detalle(string Guid)
   {

     if (Guid == null)
     {
     Guid = TempData["ObjId"].ToS-
tring();
     }
     ArticulosModel model = new
ArticulosModel();
     ViewBag.Id = Guid;
     return View(model.
Detalle(Guid));
   }
```

4.2.3. Eliminar un artículo

Código para eliminar un artículo:

```
public ActionResult Borrar(string
Guid)
   {

     ArticulosModel model = new
ArticulosModel();
     model.Borrar(Guid);

     //Borramos el indice de Luce-
ne
     ClearIndexRecord(Guid);
     return
RedirectToAction("Index");
   }
```

4.2.4. Crear un comentario

Código para crear un comentario:

```
[HttpPost]
  public ActionResult
CrearComentario(FormCollection
form)
   {
     ArticulosModel model = new
ArticulosModel();
     model.
CrearComentario(form["id"].ToS-
tring(),
       new Comentario()
       {
         usuario =
form["usuario"].ToString(),
         texto =
form["comentario"].ToString(),
         fecha = Convert.
ToString(DateTime.Now)
       });
     TempData["ObjId"] =
form["id"].ToString();
     return
RedirectToAction("Detalle");
   }
```

4.3. Crear la vista

La Vista presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida [6].

4.3.1. Crear la vista del modelo

Se crea la una vista del modelo con el siguiente código:

```
using System.Collections.Generic;
using Blog.Models;
using Blog.Controllers;

namespace Blog.ViewModels
```

```
{
  public class IndexViewModel
  {
    public Articulo SampleData {
get; set; }
    public IEnumerable<Articulo>
AllSampleData { get; set; }
    public IEnumerable<Articulo>
AllSearchIndexData { get; set; }
    public IEnumerable<Articulo>
SampleSearchResults { get; set; }
    public string SearchTerm {
get; set; }
    public string SearchField {
get; set; }
  }
}
```

4.4. Uso de Lucene.Net

Las clases clave se utilizan para construir un motor de búsqueda en Lucene son:

- Documentos: el Documento de clase representa un documento Lucene.
- Campo: el campo de clase representa una sección de un documento. El campo objeto contendrá un nombre para la sección y los datos reales.
- Analizador: el analizador de clase es una clase abstracta que se utiliza para proporcionar una interfaz que se llevará a un documento y convertirlo en señales que pueden ser indexados.
- IndexWriter: el IndexWriter clase se utiliza para crear y mantener índices.
- IndexSearcher: el IndexSearcher clase se utiliza para buscar a través de un índice.
- QueryParser: el QueryParser clase se utiliza para generar un analizador que puede buscar a través de un índice.
- Consulta: la consulta de clase es una clase abstracta que contiene los criterios de búsqueda creado por el QueryParser.
- Impactos: el Impactos clase contiene los documentos objetos que se devuelven al ejecutar el Query objeto con el índice [5].

4.4.1. Iniciar el buscador de Lucene

Iniciamos el buscador de Lucene.Net en búsqueda por un campo o por más de uno agregamos al índice con el siguiente código:

```
  // main search method
  private static
IEnumerable<Articulo> _
search(string searchQuery, string
searchField = "")
  {
    // validacion
    if (string.
IsNullOrEmpty(searchQuery.Repla-
ce("*", "").Replace("?", ""))) re-
turn new List<Articulo>();

    // Iniciar buscador de lucen
    using (var searcher = new In-
dexSearcher(_directory, false))
    {
      var hits_limit = 1000;
      var analyzer = new
StandardAnalyzer(Version.LUCE-
NE_29);

      // buscar por un campo
      if (!string.
IsNullOrEmpty(searchField))
      {
        var parser = new
QueryParser(Version.LUCENE_29,
searchField, analyzer);
        var query = parser.
Parse(searchQuery.Trim());
        var hits = searcher.
Search(query, hits_limit).Score-
Docs;
        var results = _
mapLuceneToDataList(hits, sear-
cher);
        searcher.Close();
        searcher.Dispose();
        return results;
      }
      // Buscar por multiples
campos
      else
```

```
        {
          var parser = new Multi-
FieldQueryParser
            (Version.LUCE-
NE_29, new[] { "id", "titulo",
"texto","fecha"}, analyzer);
          var query = parser.
Parse(searchQuery.Trim());
          var hits = searcher.
Search(query, null, hits_limit,
Sort.INDEXORDER).ScoreDocs;
          var results = _
mapLuceneToDataList(hits, sear-
cher);
          searcher.Close();
          searcher.Dispose();
          return results;
        }
    }
}
```

4.5. Integración MongoDB y Lucene.Net

Integramos mongodb y el buscador de lucene.NET en el CONTROLADOR para que cada vez que se agreguen artículos estos se indexen al buscado Lucene.Net con el siguiente código:

```
//Agregar a Index de Luce-
ne_____________________
   [HttpPost]
   public ActionResult
AddToIndex(Articulo sampleData)
   {
   LuceneSearch.AddUpdateLuceneInde
x(sampleData);
     TempData["Result"] = "El re-
gistro ha sido agregado a los in-
dices de Lucene!";
     return
RedirectToAction("Index");
   }
   public ActionResult
Search(string searchTerm, string
searchField)
   {
     return
RedirectToAction("Index", new {
searchTerm, searchField });
   }

   public ActionResult CreateIn-
dex()
   {
     ArticulosModel model = new
ArticulosModel();
     LuceneSearch.
AddUpdateLuceneIndex(model.ObtAr-
ticulos());
     TempData["Result"] = "EL in-
dice de buscada ha sido creado!";
     return
RedirectToAction("Index");
   }
```

4.6. Resultados obtenidos de la integración

Se agregaron registros por medio de la aplicación a la base de datos y se indexaron al motor de búsqueda Lucene.NET.

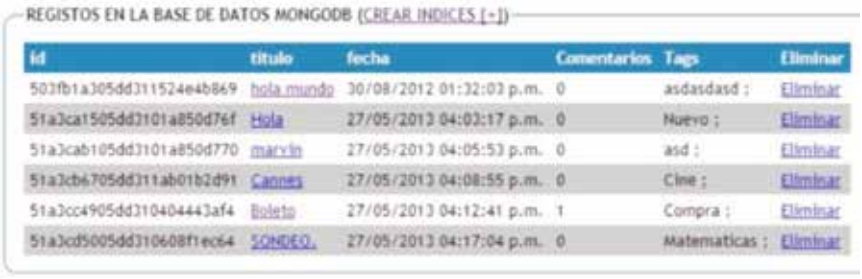
Figura 8. Pantalla de captura para nuevo artículo

Fuente: elaboración propia.

Se encontró que las búsquedas con índices es más rápida que la búsqueda directa en la base de datos. Es importante que cuando se crea un registro se crea el índice de ese registro en el motor de búsqueda.

4.6.1. Registros en la base de datos MongoDB

Figura 9. Registros en MongoDB



Fuente: elaboración propia.

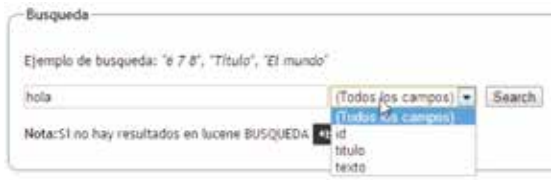
Registros en el índice de Búsqueda de Lucene.Net

Figura 10. Índices de búsqueda

Fuente: elaboración propia.

Búsqueda de información por un campo o todos los campos en el índice de búsqueda.

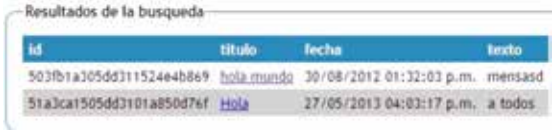
Figura 11. Índices de búsqueda



Fuente: elaboración propia.

Resultados de los índices de búsqueda

Figura 12. Resultados



Fuente: elaboración propia.

5. Conclusiones

Al realizar la implementación de esta aplicación comprobamos que al tener una gran cantidad de datos, es necesario de un sistema de búsqueda potente con acceso en tiempo real y máximo rendimiento, al usar Lucene. Net se comprobó que es un motor de búsqueda completo, fácil de implementar y proporciona un marco de adaptación para cualquier sistema. Con Lucene.Net construimos búsquedas complejas y sencillas sobre grandes volúmenes de datos. La desventaja de Lucene.NET es que solo admite texto simple y si queremos agregar archivos tendríamos que agregar otras clases a nuestro proyecto, tampoco hay documentación suficiente de Lucene.net. Las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros. La pérdida de flexibilidad en tiempo de ejecución, comparado con los sistemas SQL clásicos, se ve compensada por ganancias significativas en escalabilidad y rendimiento cuando se trata con ciertos modelos de datos. El rendimiento y sus propiedades de tiempo real son más importantes que la coherencia en una estructura de datos, en la que las bases de datos relacionales dedican una gran cantidad de tiempo de proceso. Los sistemas completos de texto son mejores para la búsqueda rápida de grandes volúmenes de texto estructurado para la presencia de la palabra o combinación de palabras. Se puede integrar satisfactoriamente ambas tecnologías impactando en la implementación de buscadores de tiempo real en grandes volúmenes de información.

6. Referencias

[1] Revista Vínculos (2012) Volumen 8 Numero 1 Enero-Junio 2011 Paginas 81 – 92. http://revistavinculos.udistrital.edu.co/files/2012/12/integracionbusqueda.pdf

[2] Chodorow Kristina & Dirolf Michael,(2010) MongoDB: The Definitive Guide. O'Reilly Media.

[3] MONGODB, (2011) http://www.mongodb.org/

[4] Chodorow Kristina (2010) Escaling MongoDB. O'Reilly Media.

[5] Lucene.NET (2011) http://lucenenet.apache.org/

[6] Wikipedia, (2013) http://es.wikipedia.org/