Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología en Internet

PROTOTIPO PARA LA GESTIÓN Y CLASIFICACIÓN DE COLECCIONES DE DOCUMENTOS DIGITALES

Trabajo Especial de Grado presentado ante la Ilustre

Universidad Central de Venezuela por el

Br. Annalicia Ostos Sánchez.

CI 18.249.031

Para optar por el título de

Licenciado en Computación

Tutor: Prof. Andrés Sanoja

Acta

Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado con el título **"Prototipo para la gestión y clasificación de Colecciones de Documentos Digitales",** el cual es presentado por la Br. Annalicia Ostos Sánchez, de Cédula de Identidad 18.249.031 a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del jurado, se fijó el día 26 de mayo de 2011, a las 3:00pm en el Centro de Computación, para que la autora lo defendiera en forma pública, mediante una presentación oral de su contenido, luego de lo cual se respondió las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con una nota de ____ puntos.

En fé de lo cual se levanta la presente Acta, en Caracas a los 26 días del mes de Mayo del año dos mil once.

_____
Prof. Andrés Sanoja
(Tutor)

_____
Prof. Robinson Rivas
(Jurado)

_____
Prof. Yusneyi Carballo Barrera
(Jurado)

Gracias a Dios, por ser mi guía y mi más fiel amigo todos estos años, por permitirme estar en donde me encuentro ahora, por darme luz en los momentos más oscuros y fuerzas cuando no perdía el ánimo.

Gracias a mi mamá, por siempre estar a mi lado apoyándome y ayudándome a salir adelante. Por enseñarme e inculcarme valores y principios. Gracias por siempre escucharme y respetar mis decisiones así muchas veces no estuvieras del todo de acuerdo.

Gracias a mi familia que por apoyarme y siempre tratar de enseñarme cosas nuevas para ser una mejor persona.

Gracias a mis Amigos, como cariñosamente los llamo los chachos, por sacarme sonrisas, aguantarme y ayudarme cuando estaba mal, por ese apoyo y compañerismo todos estos años. Gracias por siempre creer en mí y hacérmelo saber en los momentos difíciles.

Gracias a mi tutor, por aceptarme como tesista, por todos los conocimientos que me transmitió y por siempre tener palabras de alientos en los momentos difíciles y por ser más que un tutor y profesor un amigo.

Con mucho cariño,

Annalicia Ostos.

RESUMEN

El objetivo del presente Trabajo Especial de Grado consiste en el desarrollo de una aplicación Web que permita apoyar los procesos de digitalización de documentos, clasificación y visualización de documentos digitales basados en una taxonomía. Dentro de las necesidades contempladas en la aplicación se tiene lo siguiente: contar con una aplicación generalizada que le permita a los usuarios de forma rápida y eficiente realizar una estructura jerárquica de los documentos digitalizados y así mismo se pueda manejar la data de los mismos de una forma eficiente.

Para cumplir con los objetivos de la aplicación se utilizó el método ágil XP (Programación Extrema), la cual se basa fundamentalmente, en la construcción progresiva del software sin hacer énfasis en la etapa de diseño.

Palabras Clave: Digitalización, documentos digitales, clasificación, taxonomías

Tabla de contenido

INTRODUCCIÓN 10
CAPÍTULO I: Propuesta del Trabajo Especial de Grado 13
1.1. Planteamiento del Problema 13
1.2. Justificación de la Investigación 13
1.3. Solución 15
1.4. Objetivo General 16
1.5. Objetivos Específicos 16
1.6. Alcance 17
CAPÍTULO II: MARCO CONCEPTUAL 18
2.1. Proceso de Digitalización 18
2.1.1. Definición 18
2.1.2. Etapas de un Proceso de Digitalización 18
2.1.3. Razones para Digitalizar 21
2.1.4. Aspectos Técnicos de Digitalización 23
2.2. Tecnologías de Desarrollo 25
2.2.1. Ruby on Rails 25
2.2.2. MySQL 30
2.2.3. Plugins de Rails 34
2.3. Programación Extrema 35
2.3.1. Adaptación XP 36
CAPÍTULO III: MARCO APLICATIVO 42
3.1. Iteración 0 42
3.1.1. Planificación 42
3.1.2. Codificación 42
3.1.3. Pruebas 43
3.2. Iteración 1 45
3.2.1. Planificación 45
3.2.2. Diseño 45
3.2.3. Codificación 47
3.3. Iteración 2 47

3.3.1. Planificación 47
3.3.2. Diseño 47
3.3.3. Codificación 50
3.3.4. Pruebas 52
3.4. Iteración 3 53
3.4.1. Planificación 53
3.4.2. Diseño 53
3.4.3. Codificación 57
3.4.4. Pruebas 58
3.5. Iteración 4 59
3.5.1. Planificación 59
3.5.2. Diseño 60
3.5.3. Codificación 63
3.5.4. Pruebas 65
3.6. Iteración 5 66
3.6.1. Planificación 66
3.6.2. Diseño 66
3.6.3. Codificación 73
3.6.4. Pruebas 77
3.7. Iteración 6 78
3.7.1. Planificación 78
3.7.2. Diseño 78
3.7.3. Codificación 81
3.7.4. Pruebas 84
3.8. Iteración 7 85
3.8.1. Planificación 85
3.8.2. Diseño 85
3.8.3. Codificación 88
3.8.4. Pruebas 90
CONCLUSIONES 91
BIBLIOGRAFÍA 94
APÉNDICE 1 95

ÍNDICE DE FIGURAS

Figura 1.1 - Aplicación Web del Banco Central de Venezuela para Proyectos de Digitalización de Publicaciones Periódicas…… 14

Figura 1.2 – Vista para la incorporación de Volúmenes…… 15

Figura 2.1 – Etapas de un Proyecto de Digitalización…… 21

Figura 2.2 – Patrón MVC…… 28

Figura 2.3 - Métafora de la Aplicación…… 40

Figura 3.1 - Prueba de Rendimiento *Git* y *Subversion* de forma local…… 43

Figura 3.2 - Prueba de Rendimiento *Git* y *Subversion* en servidores remotos…….. 44

Figura 3.3 - Modelo de Base de Datos…… 46

Figura 3.4 - Interfaz Módulo de Gestión de Colecciones, Taxonomías y Documentos… 49

Figura 3.5 - Modelo de Base de Datos . Relación Colección – Características…… 50

Figura 3.6 – Construcción de los menú en forma de árbol…… 51

Figura 3.7 – Actualización de Colecciones y Taxonomías en la base de datos…… 52

Figura 3.8 – Diseño para la edición de Colecciones…… 54

Figura 3.9 – Diseño para la edición de Taxonomías…… 55

Figura 3.10 – Diseño para agregar nueva Taxonomía…… 56

Figura 3.11 – Modelo de Base de Datos. Tablas para el manejo de Usuarios…….. 56

Figura 3.12 – Código para la edición de Colecciones…… 58

Figura 3.13 – Diseño Lista de Funcionalidades…….. 60

Figura 3.14 – Formulario para agregar nueva funcionalidad…… 61

Figura 3.15 – Colecciones pertenecientes a una Taxonomía…… 62

Figura 3.16 – Taxonomías sin Colecciones Asociadas…… 62

Figura 3.17 – Creación de nuevo Usuario…… 63

Figura 3.18 – Código para la búsqueda de Colecciones y Taxonomías…….. 64

Figura 3.19 – Formulario por partes para incorporar Colecciones…… 66

Figura 3.20 – Asociar Estados a una Colección…….. 67

Figura 3.21 –Vista para incorporar los Documentos a la Colección……. 68

Figura 3.22 – Selección de varios Documentos para ser incorporados…….. 69

Figura 3.23 – Documentos próximos a incorporar……. 70

Figura 3.24 – Formulario por partes para agregar nuevos Documentos. …… 71

Figura 3.25 – Inicio de Sesión para ingresar a la Aplicación…… 72

Figura 3.26 – Código para guardar Colección. …… 73

Figura 3.27 – Código para guardar Documentos…… 74

Figura 3.28 – Configuración de la Gema *Paperclip* en el modelo……. 75

Figura 3.29 – Código para la autenticación…… 75

Figura 3.30 – Vista para verificar Documentos Disponibles……. 78

Figura 3.31 – Relación entre la tabla Usuarios y la tabla Documentos…… 79

Figura 3.32 – Visualización de los Documentos……. 80

Figura 3.33 –Código para verificar documentos disponibles…… 81

Figura 3.34 – Código para verificar los documentos de un usuario. …… 81

Figura 3.35 – Código para la descarga de los documentos…… 82

Figura 3.36 – Configuración *Helper* para mostrar el siguiente estado de un Documento… 82

Figura 3.37 – Código para actualizar Documento……. 83

Figura 3.38 – Vista para la descarga de Colecciones…… 86

Figura 3.39 – Vista para la descarga de Documentos. …… 86

Figura 3.40 –Lista de estados de los Documentos……. 87

Figura 3.41 – Código para la creación del .zip……. 88

Figura 3.42 – Código para la edición y eliminación de estados de Documentos…… 88

ÍNDICE DE TABLAS

Tabla 2.1 - Esquema de Planificación de cada Iteración..... .38

Tabla 2.2 - Roles existentes durante el desarrollo..... 40

Tabla 3.1 - Planificación Iteración 0 42

Tabla 3.2 - Planificación Iteración 1 45

Tabla 3.3 - Planificación Iteración 2 47

Tabla 3. 4 - Planificación Iteración 3 53

Tabla 3. 5 - Planificación Iteración 4 59

Tabla 3. 6 - Planificación Iteración 5 66

Tabla 3. 7 - Planificación Iteración 6 78

Tabla 3. 8 - Planificación Iteración 7 85

INTRODUCCIÓN

En la actualidad existen muchas empresas, instituciones y organizaciones, tanto públicas como privadas, que han almacenado a lo largo de los años distintos tipos de documentos y oficios de interés general en archivos físicos, lo cual presenta inconvenientes para las empresas ya que empiezan a necesitar de espacios más grandes para su almacenamiento y al paso de los años estos documentos se van deteriorando y el acceso a la información que estos proveen es menor ya que las personas deben dirigirse al lugar donde se encuentra almacenado este documento.

La digitalización de documentos propone una transformación en el manejo de la información. No sólo porque grandes cantidades de datos pueden ser almacenadas en dimensiones muy pequeñas, sino porque además se libera a los objetos de la dependencia de su existencia física. Es decir, un cuadro, un informe, una fotografía o algún otro tipo de documento, ya no necesitan que alguien se acerque a ellos para conocerlos, sino que por medio de una computadora están a disposición del mundo entero.

Por lo tanto, la digitalización de documentos está cambiando el funcionamiento de miles de empresas en todo el mundo, ya que por medio de ésta, se organiza mejor el trabajo de cientos de miles de empleados, debido a los beneficios que está posee.

Así mismo, en los últimos años el uso del Internet ha crecido rápida y continuamente, lo que ha ocasionado que la cantidad de personas que se conectan a este medio sea cada vez mayor y se ha masificado de tal manera que ha modificado nuestro modo de vida. A consecuencia de esto, se han incrementado también el desarrollo de aplicaciones Web, ya que estas ofrecen numerosos beneficios para los usuarios.

Es por esto que se ha incrementado la cantidad de empresas, organizaciones y/o instituciones que desean emprender un proyecto de digitalización con el apoyo de una aplicación Web, debido a que a través de la misma pueden realizar de forma más fácil y rápida

mucha de las actividades de búsqueda y visualización de los documentos que se realizan manualmente hoy en día.

Por tal motivo se desarrolló una aplicación Web que permite automatizar el proceso de digitalización de documentos, en la cual se complementen los diversos procesos que se deben llevar a cabo para la realización de un proyecto de digitalización como son el almacenamiento, clasificación y visualización de los documentos digitalizados.

En el presente trabajo de investigación se presenta un conjunto de tecnologías empleadas para desarrollar la aplicación Web para los proyectos de digitalización, así como la especificación de la elaboración de dicha aplicación, siguiendo la metodología XP.

De acuerdo a las necesidades antes planteadas, el presente documento está organizado en tres capítulos y respectivas conclusiones, los cuales están estructurados como se describe a continuación:

- **Capítulo I: Propuesta del Trabajo Especial de Grado.** En este capítulo, se expone el problema, los objetivos a desarrollar, importancia y justificación, propuesta de la solución y alcances de la investigación.

- **Capítulo II: Marco Conceptual.** Se describen los tópicos más relevantes que se encuentran estrechamente relacionados con el problema de investigación, abarcando los siguientes aspectos:
    - Proceso de Digitalización, definición, etapas, razones para digitalizar.
    - Tecnologías de desarrollo, definición, funcionamiento y descripción de cada una de las siguientes: Ruby on Rails, MySQL, Gemas[1] utilizadas.
    - Método de Desarrollo, se describen los aspectos más relevantes del proceso de desarrollo Programación Extrema XP.

[1] Gema, o *Gem* en ingles, es un empaquetado de aplicaciones Ruby o una librería de Ruby.

- **Capítulo III: Marco Aplicativo.** Se especifica cada una de las iteraciones de construcción del sistema, siguiendo el método ágil XP (Programación Extrema).

Finalmente se presentan las conclusiones y recomendaciones de la investigación. Así como la bibliografía y el apéndice, en donde se encuentran los pasos para la instalación de Ruby y Rails en Ubuntu 10.10.

CAPÍTULO I: Propuesta del Trabajo Especial de Grado

1.1. Planteamiento del Problema

En la actualidad existen muchas empresas, instituciones y organizaciones, tanto públicas como privadas, que han almacenado a lo largo de los años distintos tipos de documentos y oficios de interés general en archivos físicos. Con el paso del tiempo los mismos van perdiendo su integridad física y las distintas empresas, instituciones y organizaciones poseen gran interés en conservarlos ya que representan algún valor para ellos, ya sea histórico o monetario.

Asimismo, dichos documentos deben ser clasificados y organizados de tal manera que se tenga un rápido acceso a ellos y se puedan realizar rápidamente las búsquedas necesarias del contenido de los documentos y oficios.

La clasificación y búsqueda de la información de dichos documentos resulta tediosa ya que no se cuenta con una herramienta de software libre que sea un experto de clasificación de documentos digitales, la cual brinde un mecanismo de clasificación dependiendo de una taxonomía.

1.2. Justificación de la Investigación

Debido a la importancia que presenta el manejo, clasificación y búsqueda de información de los documentos y oficios históricos de diferentes organizaciones, surge la necesidad de contar con una herramienta generalizada que les permita de forma rápida, eficiente y fácil de usar realizar una estructura jerárquica de dichos documentos y asimismo se puedan realizar búsquedas y manejo de la data de los mismos de una forma eficiente.

La realización de esta aplicación surge para mejorar y generalizar una aplicación Web ya existente para el manejo de proyectos de digitalización, la cual pertenece al Banco Central de Venezuela. En la figura 1.1 se muestra la interfaz de dicha aplicación.

Figura 1.1 – Aplicación Web del Banco Central de Venezuela para Proyectos de Digitalización de Publicaciones Periódicas.

En la figura 1.1, se puede observarlas diferentes funcionalidades que presenta actualmente esta aplicación. Dicha aplicación fue realizada para almacenar y administrar únicamente publicaciones periódicas del Banco Central de Venezuela, por esta razón surge la necesidad de crear una nueva aplicación la cual sea generalizada, es decir, sirva para apoyar cualquier proyecto de digitalización y almacenar cualquier tipo de archivo.

Otro inconveniente que se presenta en esta aplicación, es la carga de archivos o volúmenes la cual, se realiza a través de un archivo comprimido (.zip) o directamente

ingresando al disco donde se almacenaran las publicaciones, por lo cual se debe realizar la incorporación de las revistas en la red en donde se encuentre el servidor que almacenara las mismas. Si la carga de volúmenes se realiza a través de un archivo comprimido, el tamaño máximo del mismo dependerá de la configuración del servidor. En la figura 1.2 se muestra la pantalla para la incorporación de volúmenes.

Figura 1.2 – Vista para la incorporación de Volúmenes.

1.3. Solución

Con la finalidad de apoyar los procesos de digitalización de imágenes que se deseen comenzar, tanto en empresas, instituciones y organizaciones, se propone implementar una aplicación Web óptima y eficiente la cual permita realizar el procesamiento de los documentos luego de que los mismos sean digitalizados.

La aplicación, permitirá al usuario crear colecciones para almacenar los documentos, dichas colecciones pertenecerán a una taxonomía en particular. Para luego

proceder a la carga de los documentos digitalizados para de esta forma poder visualizarlos y procesarlos.

Dicha aplicación no dependerá de una red interna para su funcionamiento.

1.4. Objetivo General

Desarrollar una aplicación Web que permita apoyar los procesos de digitalización de documentos, clasificación y visualización de documentos digitales basados en una taxonomía.

1.5. Objetivos Específicos

- Desarrollar un módulo de procesamiento y visualización de documentos.
- Diseñar las interfaces de usuario de cada uno de los módulos del sistema.
- Diseñar e implementar la base de datos donde será almacenada la información provente de los datos obtenidos del proceso de digitalización de los documentos.
- Utilizar el método de XP para el proceso de desarrollo.
- Modelar una estructura orientada a objetos de forma genérica para representar un documento.
- Permitir realizar la definición de las jerarquías necesarias para la categorización de los documentos.
- Realizar pruebas funcionales a la aplicación.
- Integrar todos los componentes de la aplicación.

- Incorporar la aplicación final a la comunidad de software libre.

1.6. Alcance

La aplicación que se desarrollo abarca las siguientes etapas de un proceso de digitalización, después que el documento ha sido digitalizado, es decir, se brindarán las siguientes funciones:

1. La carga al sistema de los documentos previamente digitalizados.
2. El manejo, clasificación y visualización de los documentos ya cargados en la aplicación.

Además se dispone de un módulo de administración, en el cual se realiza el manejo de usuarios, las funcionalidades del sistema y los roles de usuarios en la aplicación.

CAPÍTULO II: MARCO CONCEPTUAL

En este capítulo se describen los tópicos que están estrechamente relacionados con los procesos de digitalización de documentos. Asimismo se puntualizan los aspectos más relevantes del Reconocimiento Óptico de Caracteres y finalmente se describen el conjunto de tecnologías utilizadas para el desarrollo del presente Trabajo Especial de Grado.

2.1. Proceso de Digitalización

2.1.1. Definición

Digitalizar es el proceso de transformar algo analógico, lo cual es un elemento real de precisión infinita, en algo digital que es un conjunto finito de precisión determinada de unidades binarias. Por lo tanto, se trata es de tomar una imagen, ya sea papel o film y convertirlo en un formato el cual se pueda trabajar informáticamente. (Manual de Digitalización de Documentos, 2004)

Por lo tanto, la digitalización de documentos supone un cambio total en el tratamiento de la información. Permite su almacenamiento en grandes cantidades en objetos de tamaño reducido o, lo que es más revolucionario, liberarla de los propios objetos y de sus características materiales y hacerla residir en redes informáticas, accesibles desde cualquier lugar del mundo en tiempo real.

2.1.2. Etapas de un Proceso de Digitalización

Un proceso de Digitalización, pasa por las siguientes etapas: (Ministerio del Transporte y Comunicaciones, 2006)

- **RECEPCIÓN:** Recepción de documentos y transferencia al Área de Digitalización.

- **VERIFICACIÓN:** Verificación del estado y de la cantidad de documentos recibidos.

- **PREPARACIÓN:** Retiro de los elementos extraños tales como clips, grapas, residuos de goma u otros objetos con el fin de permitir una mejor digitalización.

- **CAPTURA:** Es propiamente la digitalización de los documentos físicos.

- **PROCESAMIENTO DIGITAL DE LAS IMÁGENES:** En esta etapa se corrige dinámicamente los documentos inclinados y se compensa contrastes pobres. Cuando se detecta una imagen de mala calidad, se debe utilizar un programa de procesamiento de imágenes para resolver dicho problema.

- **INDEXACIÓN**: Ingreso manual de datos correspondientes a los documentos capturados no reconocidos por el sistema OCR (Reconocedor Óptico de Caracteres). Cada página o unidad de información serán indexadas mediante un programa diseñado para este fin. La indexación es digitalizar los campos del perfil o catálogo definidos. Esta indexación comprende 2 formas:

    - ✓ **Indexado por Plantillas**: Los documentos, una vez digitalizados, se indexan basándose en los criterios de búsqueda establecidos por serie documental. De existir bases u otras fuentes de datos que contengan los índices correspondientes se pueden aprovechar para indexar los documentos de forma automática. En caso contrario, se procede a la digitación de los campos en cuestión. A cada documento se le asigna la plantilla de la serie documental correspondiente y se le asocia los valores respectivos a dicho documento para poder realizar la búsqueda.

- ✓ **Indexación OCR:** Los documentos, una vez digitalizados, son sometidos a un reconocimiento óptico de caracteres. Esto es opcional, y solo debe hacerse sobre documentos legibles, previa prueba de reconocimiento, y que hayan pasado el control de calidad.

- **CONTROL DE CALIDAD:** Revisión del 100% de imágenes y datos de los documentos digitalizados, enviando a reproceso las imágenes que no cumplan con los criterios establecidos en el Protocolo de Calidad (Fidelidad, Integridad, Legibilidad).

- **REPROCESO:** Cuando alguna imagen sea rechazada por no cumplir con los parámetros establecidos en el Protocolo, la persona encargada de la digitalización de los documentos procede a capturar nuevamente el documento.

- **COMPAGINACIÓN:** Ordenación de los documentos físicos de acuerdo a su posición original.

- **DEVOLUCIÓN:** Transferencia de documentos físicos ya digitalizados al archivo pasivo o al Área donde pertenece la documentación.

- **ENTREGA DE MEDIOS PARA SU ALMACENAMIENTO:** Los medios (CDs, DVDs, discos duros, etc.) que contienen las micro formas son entregados para su conservación en una bóveda certificada.

- **ALMACENAMIENTO EN EL SERVIDOR:** Una vez entregados los medios que contienen las micro formas para su conservación en una bóveda certificada, se almacena las imágenes en el Servidor de imágenes en forma definitiva, para ser utilizado por el Sistema de Gestión de contenidos o Gestión Documentaria.

En la figura 2.1 se puede observar la interacción entre las etapas que se desarrollan en un proceso de digitalización.

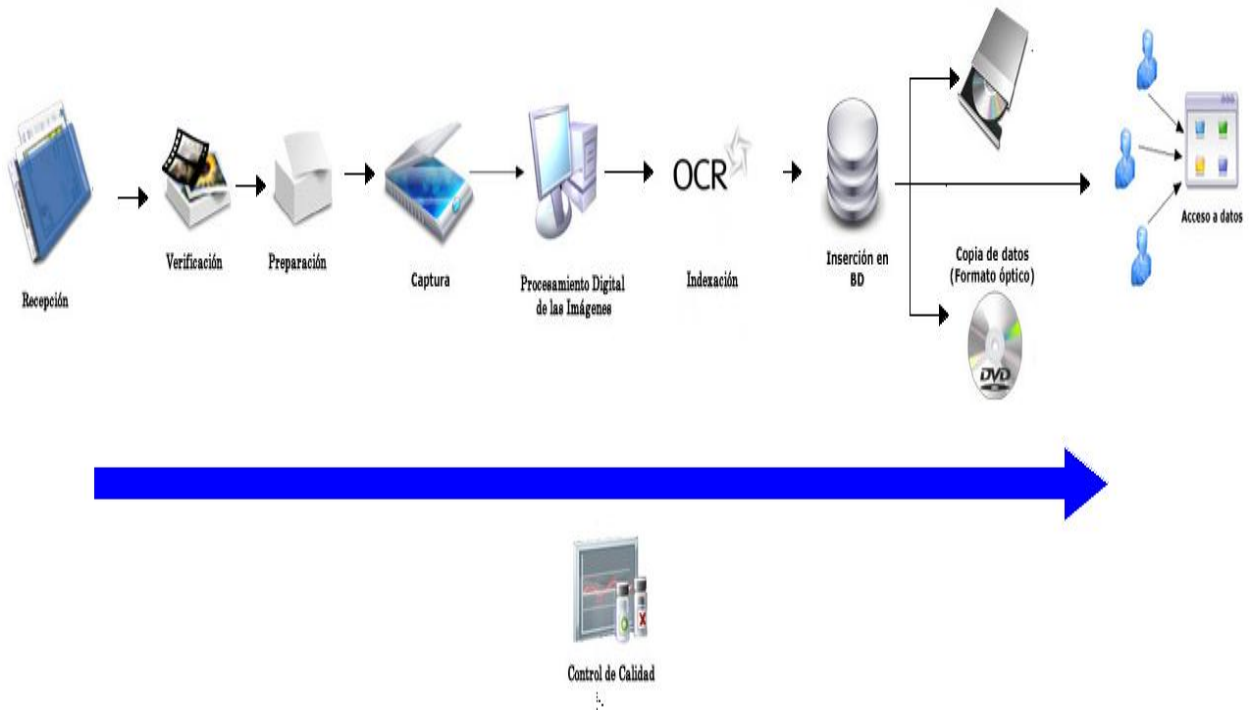

Figura 2.1 – Etapas de un Proyecto de Digitalización
Fuente eBackp (http://www.ebackup.cl/proceso-de-digitalizacon.html)
Con modificaciones realizadas por mí

2.1.3. Razones para Digitalizar

La razón de la implementación de un proyecto de digitalización, o más exactamente de la conversión digital de documentos originales no digitales son variadas y pueden solaparse. La decisión de digitalizar puede tomarse con objeto de (Unesco, 2002):

- Incrementar el acceso: Esta es razón principal y la más relevante, cuando se sabe que hay una alta demanda por parte de los usuarios y una institución desea mejorar el acceso a determinados documentos.

- Mejorar los servicios para un grupo creciente de usuarios proporcionando un acceso de mayor calidad a los recursos de la institución en relación con la educación y la formación continua.

- Reducir la manipulación y el uso de materiales originales frágiles o utilizados intensivamente y crear una "copia de seguridad" para el material deteriorado como libros o documentos quebradizos.

- Ofrecer a la institución oportunidades para el desarrollo de su infraestructura técnica y para la formación técnica de su personal.

- Impulsar el desarrollo de recursos cooperativos, compartiendo intereses comunes con otras instituciones para crear colecciones virtuales e incrementar el acceso a nivel internacional.

- Buscar intereses comunes con otras instituciones para rentabilizar las ventajas económicas de un enfoque compartido.

- Aprovechar las oportunidades financieras, como, por ejemplo, la posibilidad de asegurar una inversión para implementar un programa, o un proyecto concreto capaz de generar un beneficio significativo.

2.1.4. Aspectos Técnicos de Digitalización

2.1.4.1. Atributos de los documentos originales

En la digitalización de un documento, se deben tomar en consideración los procesos técnicos implicados en la misma, así como los atributos de los documentos originales. Estos atributos pueden ser de distintas dimensiones y nivel tonal. Además, los documentos originales también pueden caracterizarse por el modo en que se han producido, como por ejemplo, a mano, mecanografiados o impresos, o por métodos fotográficos o electrónicos.

La condición física de los documentos originales puede influir de diferentes maneras en el proceso de digitalización. Los documentos que presentan daños en su condición física, como por ejemplo, textos descoloridos, manchas de tinta, páginas quemadas, a veces destruyen el contenido informativo, pero de forma más frecuente imponen limitaciones físicas a las posibilidades para capturar información durante el proceso de escaneo. Por tanto, si el documento a digitalizar presenta un daño en su condición física, se debe identificar la posible necesidad de realizarle un tratamiento previo de los mismos antes de proceder a digitalizarlos. No tomar en cuenta la condición física del documento puede representar una amenaza para su integridad así como también limitar los beneficios y resultados de la digitalización e incrementar el costo de procesarlos. Los pasos más comunes para prevenir este problema son, por ejemplo, llevar a cabo tratamientos básicos de conservación previos y usar atriles para los volúmenes encuadernados, y rutinas para controlar la luz y otras condiciones ambientales durante la digitalización.

Cuando los riesgos de daño de los documentos originales sean altos y los documentos tengan un valor especial o estén en malas condiciones, a veces puede

ser mejor escanear a partir de ficheros intermedios de microfilm y no a partir de los documentos originales, si se dispone de dichos microfilms.

2.1.4.1.1. Calidad de la imagen

La calidad de la imagen durante la captura depende de la suma de resultados de la resolución aplicada a la digitalización, la profundidad del bit de la imagen escaneada, los procesos de mejora y el nivel de compresión aplicada, el dispositivo de digitalización utilizado o técnicas usadas, y la preparación del operador del escáner.

2.1.4.1.2. Control de Calidad

El control de calidad (QC = *quality control*) es un componente esencial de un programa de digitalización de imágenes, y tiene como fin asegurar que se han cumplido las expectativas en cuanto a calidad. El mismo abarca procedimientos y técnicas para verificar la calidad, precisión y consistencia de los productos digitales.

Por lo tanto, el control de calidad es un elemento importante en cada una de las etapas de un proyecto de digitalización. Sin este trabajo no será posible garantizar la integridad y consistencia de los archivos de imágenes. Por lo cual, deben tomarse medidas para minimizar las variaciones entre las diferentes personas encargadas del proyecto de digitalización así como entre los distintos escáneres que se utilicen. Los escáneres deben, además, revisarse regularmente para verificar su precisión y su calidad.

Aunque el control de calidad es un factor crucial para asegurar los mejores resultados no existe un modo normalizado para asegurar una

determinada calidad de la imagen durante su captura. Los diferentes documentos originales requieren diferentes procesos de digitalización, lo que debe tenerse en cuenta cuando se desarrollan programas de control de calidad.

Los objetivos del control de calidad en un proyecto de digitalización, depende de los productos requeridos y de los niveles de calidad y puntos de referencia elegidos para el mismo, como por ejemplo, si se incluirá la colección completa de imágenes o una muestra, todos los tipos de archivos producidos (archivos de conservación, archivos de acceso, archivos de miniaturas.), entre otros.

Un programa de control de calidad siempre incluye los archivos de conservación que se produce y en la mayoría de los casos también se tendrán en cuenta otros productos como los archivos de acceso, micro formas y copias en papel.

2.2. Tecnologías de Desarrollo

Las tecnologías de Desarrollo, que se explicarán en los siguientes puntos, fueron las utilizadas para la creación de la aplicación Web. Las mismas se escogieron debido a que la aplicación Web del Banco Central de Venezuela, se realizo con esta plataforma, la cual está conformada por Ruby on Rails y MySQL, por esta razón se decidió continuar con la misma plataforma para el desarrollo de la nueva aplicación.

2.2.1. Ruby on Rails

Ruby on Rails es un entorno de desarrollo Web de código abierto que está optimizado para satisfacción de los programadores y el aumento de la productividad. Te permite escribir un buen código favoreciendo la convención antes que la configuración.

En este concepto es importante definir la separación entre los dos entes importantes que conforman este entorno, Ruby y Rails. Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Según su creador, Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina.

Por otro lado, Rails, según Ruby on Rails Org, es un completo entorno para desarrollar aplicaciones Web con base de datos de acuerdo con la estructura Modelo-Vista-Controlador (*MVC*). Desde el Ajax en la vista, a la petición y respuesta en el controlador, hasta el modelo, Rails te da un entorno de desarrollo de Ruby. Para probarlo, solo se necesita una base de datos y un servidor Web.

El desarrollo sobre este entorno está basado en dos filosofías, no te repitas (del inglés *Don't repeat yourself*, *DRY*) y convención sobre configuración. *DRY* significa que las definiciones deberían hacerse una sola vez. Dado que Ruby on Rails es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Mientras que convención sobre configuración significa que el programador sólo necesita definir aquella configuración que no es convencional. Ruby on Rails se ha convertido en un entorno muy poderoso para el desarrollo de aplicaciones Web y que cada día toma más auge dentro del mundo del desarrollo Web.

2.2.1.1. Funcionamiento

Ruby on Rails funciona bajo el paradigma Modelo-Vista-Controlador (MVC), (Burbeck, 1992) el cual es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos, el cual aplicado a Web la vista es un página HTML, o html.erb en caso de Rails, el cual contiene el HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista. Específicamente en Rails el modelo MVC se especifica de la siguiente manera:

a) **Modelo**: En las aplicaciones Web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos. En Ruby on Rails, las clases del Modelo son gestionadas por *ActiveRecord.* Por lo general, la única tarea que tiene un desarrollador es hacer que los modelos hereden de la clase *ActiveRecord::Base,* y con esto Rails sabrá mediante las convenciones qué tabla usar y qué columnas tiene dicha tabla.

b) **Vista**: Es la lógica de visualización, es decir, cómo se muestran los datos provenientes del Controlador. Con frecuencia en las aplicaciones Web la vista consiste en una cantidad mínima de código de algún lenguaje incluido en HTML. El método que se emplea en Rails por defecto es usar Ruby Embebido (archivos .rhtml, desde la versión 2.x en adelante de RoR archivos .html.erb), que son básicamente fragmentos de código HTML código en Ruby. También pueden construirse vistas en HTML y XML con Builder3 o usando el sistema de plantillas *Liquid*.

c) **Controlador**: Las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones Web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador Web. La implementación del Controlador es manejada por el *ActionPack* de Rails, que contiene la clase *ApplicationController*. Un controlador en Rails debe heredar de esta clase y definir las acciones como métodos de dicha clase.

En la figura 2.2, se muestra la interrelación entre los componentes del patrón MVC.

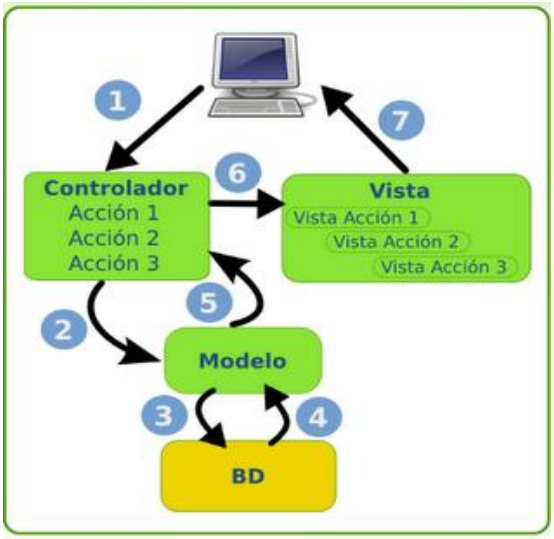

Figura 2.2 – Patrón MVC.

2.2.1.2. Framework Rails

Según Ruby on Rails Org, Rails está separado en varios paquetes, los cuales en conjunto forman el Framework. Básicamente los paquetes principales de los cuales se constituye Rails se mencionan a continuación:

a) **ActiveRecord:** Es una alternativa que facilita acceder a los datos de una base de datos. Una fila en la tabla de la base de datos (o vista) se

envuelve en una clase, de manera que se asocian filas únicas de la base de datos con objetos del lenguaje de programación usado. Cuando se crea uno de estos objetos, se añade una fila a la tabla de la base de datos. Cuando se modifican los atributos del objeto, se actualiza dicha la fila. La clase envoltorio implementa métodos de acceso para cada columna de la tabla o vista. Rails implementa este enfoque para proveer una interfaz hacia los datos que facilite su acceso y manipulación utilizando los modelos. La clase específica que implementa la interfaz se llama *ActiveRecord::Base*.

b) **ActiveResource (ARes):** Es la clase principal utilizada para mapear recursos *RESTful* con modelos en una aplicación Rails. Es decir, ARes se encarga de proveer la interfaz de una aplicación Rails con una plataforma de servicios Web, permitiendo tanto recibir como crear servicios que funcionan bajo el enfoque REST. ARes conecta objetos de negocio y servicios Web REST e implementa el mapeo objeto relacional para proveer transparencia entre un cliente y un servicio *RESTful*.

c) **Action Pack:** *Action Pack* divide la respuesta a una solicitud Web en un controlador (ejecutando la lógica) y una vista (renderizando una plantilla). Este proceso en dos partes es conocido como una acción, la cual normalmente creara, leerá, actualizará o eliminará (*create, read, update o delete*, CRUD por sus siglas en ingles) alguna parte de un modelo (comúnmente soportado por una base de datos) antes de elegir renderizar una plantilla (vista) o redirigir a alguna otra acción. *Action Pack* implementa estas acciones como métodos públicos en *Action Controllers* y *Action Views*. Los *Action Controllers* son los responsables de manejar todas las acciones relacionadas con la lógica de la aplicación. Este agrupamiento usualmente consiste de acciones de procesamiento y

CRUDs alrededor de un modelo. Las plantillas *Action View* son escritas utilizando Ruby embebido mediante etiquetas en el código HTML. Para evitar llenar las plantillas con código, un manojo clases ayudantes (llamados *Helpers* en Rails) proveen funcionalidades comunes como formularios, fechas y cadenas de caracteres

a) **ActiveSupport**: Es una colección de variedad de clases utilitarias y extensiones de librerías estándar que son de gran utilidad para Rails. Todas estas librerías fueron reunidas en este paquete para poder aprovechar todo el potencial que ofrece el lenguaje Ruby. En síntesis, son librerías de propósito general de Rails.

b) **Action Mailer**: Este paquete permite el envió de correos electrónicos desde una aplicación Rails usando modelos y vistas llamadas mailers. Los modelos *Mailer* heredan de la clase *ActionMailer::Base*, así mismo, los correos son definidos creando métodos en este modelo los cuales manipulan ciertas variables que posteriormente serán usadas por la plantilla del *Mailer*.

2.2.2. MySQL

MySQL es un sistema manejador de bases de datos relacional (SMBDR), el cual proporciona un servidor de base de datos SQL (*Structured Query Language*), multihilo, multiusuario y robusto.

MySQL es capaz de almacenar enormes cantidades de datos, de gran variedad y de distribuirlos para cubrir las necesidades de cualquier tipo de organización, desde pequeños establecimientos comerciales a grandes empresas y organismos administrativos.

Este sistema manejador de base de datos es, probablemente, el gestor más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Su gran aceptación se debe, en parte, a que existen gran cantidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.

2.2.2.1. Características Básicas

Las principales características de este sistema manejador de bases de datos son las siguientes (MySQL AB, 2006):

- **Rendimiento:** MySQL permite obtener un buen rendimiento del hardware, ya que aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo, lo cual optimiza los tiempos de respuesta. Así mismo implementa un mecanismo llamado cache de consulta, mediante el cual se almacenan las últimas consultas ejecutadas recientemente y así en caso de ser requeridas por el usuario, las mismas no deben ser resueltas nuevamente, simplemente se lista el resultado de la consulta con los datos almacenados en dicha cache.

- **ACID:** Estas son las propiedades que una base de datos debe cumplir para que el Sistema Manejador de Base de Datos (SMBD) maneje correctamente la transaccionalidad, el acrónimo ACID viene de Atomicidad, Constancia, Aislamiento, Durabilidad. Estos principios son cumplidos por MySQL.

- **Multiusuario:** MySQL es capaz de soportar que varios usuarios se conecten al mismo tiempo y que puedan manipular y administrar las

distintas bases de datos existentes y administrar al SMBDR como tal, en el mismo instante de tiempo.

- **Multiplaforma:** Disponibilidad en gran cantidad de plataformas y sistemas: AIX 4x 5x, Amiga, BSDI, Digital Unix 4x. FREEBSD 2x 3x 4x, HP-UX 10.20 11x, Linux 2x, Mac OS, NetBSD, Novell NetWare 6.0, OpenBSD 2.5, OS/2, SCO OpenServer, SCO UnixWare 7.1x, SGI Irix 6.x, Solaris 2.5, SunOS 4.x, tru64 Unix y Windows 9x, Me, NT, 2000, 2003, Vista y Seven.

- **Seguridad:** MySQL implementa un protocolo que se encarga de emplear diferentes algoritmos para cifrar los datos que viajan a través de una red pública (ejemplo: Internet), el cual tiene por nombre Capa de Seguridad de Sockets (*Secure Sockets Layer* - SSL). Este protocolo trabajaría en el cliente y el servidor MySQL. Así mismo, maneja gestión de usuarios y passwords, en el sentido de qué usuarios tienen acceso a qué tablas y con qué permisos, manteniendo de esta forma un buen nivel de seguridad de los datos.

- **Conectividad:** Los clientes pueden conectar con el servidor MySQL usando Sockets TCP/IP en cualquier plataforma. La interfaz para el conector ODBC (My-ODBC) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (*Open Database Connectivity*).

- **Replicación:** Hay grandes grupos de servidores usando replicación en producción, con buenos resultados. Sin embargo MySQL sigue trabajando para mejorar las características de replicación 5.x. Las características de MYSQL 5.x soportan replicación asíncrona unidireccional: un servidor actúan como maestro y uno o más actúan como esclavos.

- **Integridad**: Proporciona sistemas de almacenamiento transaccionales y no transaccionales (no permiten hacer "ROLLBACK"). MySQL en su versión 5.0 tiene como característica el manejo de integridad referencial y de transacciones, gracias al motor InnoDB, que permite el manejo de transacciones mediante la sentencia "BEGIN WORK" y finaliza con un "COMMIT" o "ROLLBACK", o puede terminar en modo "AUTOCOMMIT". Para trabajar con este motor es necesario especificarlo al momento de la creación de cada una de las tablas, declarando explícitamente que van a ser del tipo InnoDB.

2.2.2.2. Seguridad en MySQL

El sistema de seguridad en MySQL es referido como el Sistema de Privilegios de Acceso (IBM, 2007). Permite la autenticación de los usuarios del servidor de MySQL y la verificación de las actividades de todos los usuarios sobre el servidor y las bases de datos.

La seguridad en MySQL es aplicada en dos niveles: Nivel de servidor y Nivel de Base de Datos. Cuando un usuario trata de acceder a una base de datos, primero se verifica si el usuario tiene privilegio para acceder al servidor de base de datos, después el servidor verifica si el usuario tiene privilegios para conectarse a una base de datos. La verificación de conexión al servidor y la verificación de conexión a la base de datos son dos procesos que MySQL siempre lleva a cabo.

MySQL realiza la verificación de privilegios del servidor y la base de datos usando unas tablas del sistema llamadas tablas de concesión. Estas tablas contienen toda la información necesaria para aplicar las políticas de seguridad convenientes. Todos los *host* (otros computadores) y usuarios que se conectan al servidor MySQL deben estar representados en las tablas de concesión.

2.2.3. Plugins de Rails

2.2.3.1. SimpleCaptcha

SimpleCaptcha es un simple y robusto plugin de *captcha*. Su aplicación requiere la adición de una sola línea de código en la vista, los controladores y el modelo. *SimpleCaptcha* puede ser utilizado con Rails 3 o superior, y también proporciona la compatibilidad hacia atrás con versiones previas de Rails.

Algunas de las características que presenta son: (Captcha, 2011)

- Cero uso del sistema de archivos. El código secreto se trasladó a la base de datos de la aplicación y el almacenamiento de imágenes fue eliminado.

- Proporciona diversos estilos de imagen.

- Ofrece tres niveles de complejidad de las imágenes.

- Trabaja también sobre ambientes de desarrollo distribuidos.

- Sencilla implementación. Solo es necesario escribir en la vista en la que se quiere utilizar “<%= Show_simple_captcha %>” dentro del formulario.

2.2.3.2. Paperclip

Paperclip es un importante plugin de Rails creado por Jon Yurek en la compañía Thoughtbot[2]. Este es uno de los muchos plugins disponibles que se adaptan para la subida de archivos y *thumbnailing*[3].

Paperclip es concebido como una sencilla librería del manejo de archivos adjuntos para *ActiveRecord*. La intención detrás de ésta, es la de mantener la configuración lo más fácil posible y tratar los archivos, en la medida de lo posible, como atributos. (RubyDoc.info, 2011)

Paperclip es capaz de gestionar validaciones en función de tamaño y presencia, si es necesario. Además puede transformar su imagen asignada en miniaturas si así se desea, y los requisitos para esto es la instalación de *ImageMagick* en el sistema. Los archivos se guardarán en el sistema de archivos y se podrá acceder a ellos por el navegador a través de una referencia de fácil comprensión, que tendrá valores predeterminados.

2.3. Programación Extrema

XP (eXtreme Programming) se trata de un método ágil en contraposición a las metodologías pesadas como RUP. Se basa en la simplicidad, la comunicación y la retroalimentación o reutilización del código desarrollado. No se enfoca en la documentación sino en los requerimientos comunicados por el cliente.

El objetivo principal que se persigue es la satisfacción del cliente, por eso tiene mucha importancia la comunicación con los usuarios o clientes. Esta comunicación se va a

[2] Empresa de diseño web y desarrollo ubicada en Boston. Su dirección web es www.thoughtbot.com
[3] Término en Ingles, que se le da a las versiones reducidas de imágenes.

soportar principalmente en las historias de usuario (*User Stories*) cuando proviene desde el cliente, y de las entregas y versiones parciales del sistema cuando la comunicación es hacia el cliente. (Márquez, 2008)

A continuación se listan las características de este método de desarrollo, las cuales son puestas en práctica las actividades de XP: (Jeffries, 2011)

- Desarrollo iterativo e incremental
- Pruebas unitarias continuas
- Programación en parejas
- Comunicación constante con el cliente
- Corrección de errores
- Refactorización de código
- Simplicidad
- Propiedad de código compartida

2.3.1. Adaptación XP

A continuación se describen las tareas que involucran las actividades en la adaptación del proceso XP que se utilizó durante el desarrollo de la aplicación Web.

2.3.1.1. Iteraciones

El método XP propone dividir el trabajo en iteraciones, las cuales se enfocan en versiones parciales del sistema hasta llegar al producto final. Los nuevos requerimientos son recibidos progresivamente y son incluidos en una nueva iteración.

Las iteraciones pueden ser de dos tipos principalmente: por objetivos o por lapsos de tiempo. En el desarrollo de este Trabajo Especial de Grado, las iteraciones están basadas en intervalos de tiempo. Los intervalos de tiempo fueron acordados con el cliente para la realización de las reuniones, las cuales se estimaban que se realizaran en lapso entre dos a tres semanas. Durante el tiempo fijado para cada iteración, se realizan las implementaciones indicadas en las historias de usuario, de no completarse algún requerimiento en este lapso, el mismo es agregado a la próxima iteración.

En dicho desarrollo, una historia de usuario, puede ser un requerimiento funcional, no funcional, un evento o cualquier otra tarea que se derive del desarrollo de la aplicación.

2.3.1.1.1. Planificación

Según XP, la actividad de planificación comienza creando una serie de Historias de Usuario, en las cuales se describen en una o dos oraciones los requerimientos del sistema en terminología del cliente, proporcionando a su vez una estimación del tiempo necesario para el desarrollo. (Pressman, 2007)

En este trabajo, el formato para escribir las Historias de Usuario se presenta en la Tabla 2.1, en donde se indican la iteración que se está realizando, una descripción sobre las funcionalidades realizadas en dicha iteración, la fecha de inicio y fin de cada iteración, además, se pueden observar cuatro columnas que muestran información de cada historia de usuario. La primera corresponde al número que sirve como identificador, la segunda es la fecha de inicio, la tercera una breve descripción de los requerimientos que se trabajaran y por último el tipo de historia.

Iteración -			
Descripción	-		
Fecha Inicio / Fecha Fin	_		
Número	Fecha	Historia	Tipo

Tabla 2.1 – Esquema de Planificación de cada Iteración.

2.3.1.1.2. Diseño

El método de desarrollo Programación Extrema, propone la fase de diseño como una guía de implementación para una historia de usuario determinada.

Para esta fase, XP recomienda la creación de prototipos y/o diagramas, cuando establecer un diseño para una historia de usuario resulte complicado. Asimismo, promueve aplicar refactorización, que permita realizar mejoras al diseño del código después que se ha escrito. (Pressman, 2007)

2.3.1.1.3. Codificación

Durante la actividad de codificación, este método sugiere la programación en pareja, que consiste en dos personas en una misma estación de trabajo desarrollando el código de una historia de usuario. Esto ayuda a seguir los estándares de programación, lo cual es otro aspecto requerido por el método de Programación Extrema. Por último se deben realizar frecuentes integraciones de código entre los grupos de trabajo, permitiendo evitar problemas de

compatibilidad e interfaz y ayudando al descubrimiento de errores en el sistema. (Pressman, 2007)

Para el desarrollo de esta aplicación, no se adopta la programación en pareja. Durante el desarrollo se mantiene la consistencia y legibilidad del código para facilitar la comprensión para los involucrados en el desarrollo de la aplicación, esto pensando en que dicho prototipo será mejorado y ampliado por otros desarrolladores. También se realiza una integración constante del código por medio del sistema de control de versiones (Subversion), el cual es el repositorio utilizado a la hora de almacenar los cambios realizados.

En la presente adaptación del proceso de desarrollo se extraerán y mostrarán partes del código fuente que sean esenciales en la comprensión de la aplicación y la solución a los requerimientos de las historias de usuario.

2.3.1.1.4. Pruebas

Las pruebas serán de aceptación en donde el usuario o cliente pone a prueba el sistema y verifica que hayan quedado cubierto todos los requerimientos.

2.3.1.2. Actores y Responsabilidades

Existen diferentes roles (actores) y responsabilidades en XP para diferentes actividades y propósitos durante el proceso. Para este trabajo especial de grado los roles existentes son:

- **Desarrollador:** es el responsable de tomar las decisiones técnicas y de llevar a cabo la codificación, el diseño y realizar pruebas al software.

- **Cliente:** es parte del equipo, determina que construir y cuando, desarrolla pruebas funcionales del software para determinar cuando está completo un determinado aspecto.

En la tabla 2.2 se muestran las personas encargadas de cada rol.

Rol	Nombre
Desarrollador	Br. Annalicia Ostos Sánchez
Cliente	Prof. Andrés Sanoja

Tabla 2.2 - Roles Existentes durante el desarrollo

2.3.1.3. Metáfora del Sistema

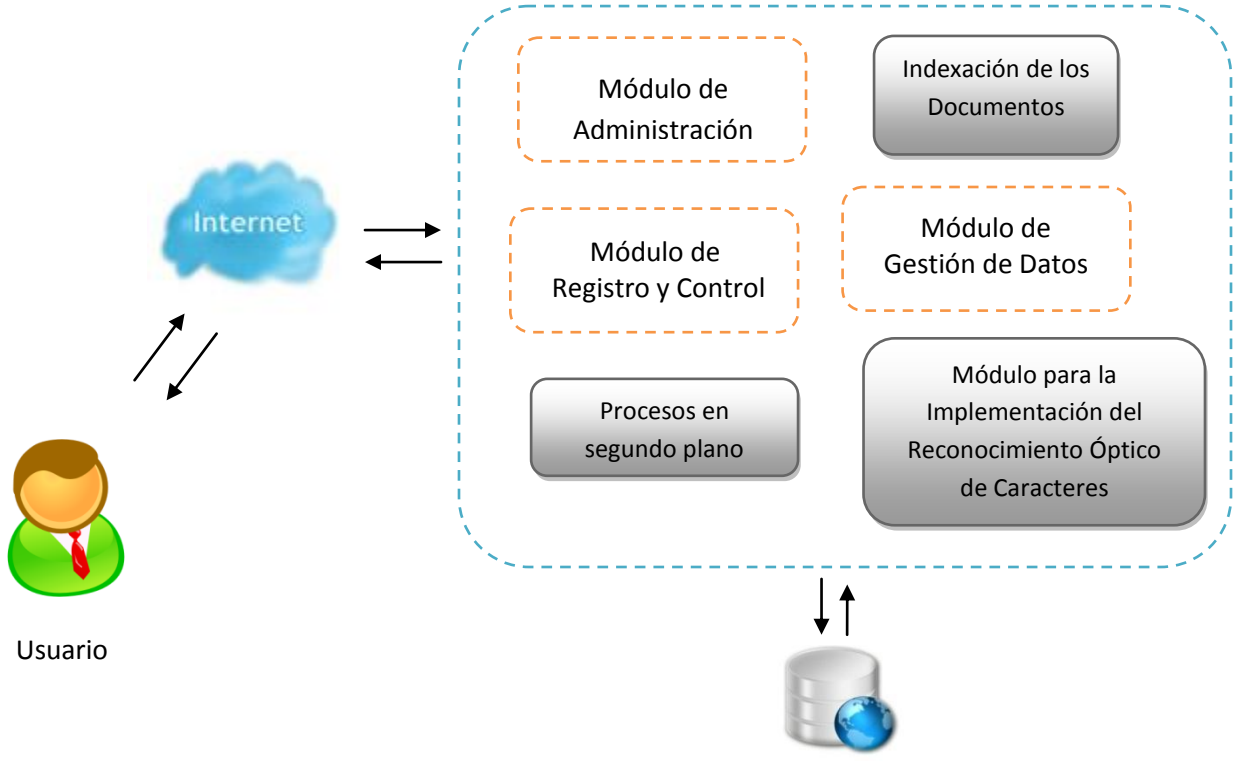

Figura 2.3 - Métafora de la Aplicación.

A continuación se describen los módulos principales que conformaran la aplicación a desarrollar:

- Módulo de “Registro y Control”: permite la incorporación de los distintos documentos al sistema, así como la creación de las colecciones. Adicionalmente, se podrán consultar todo los documentos que están en el sistema y que aun no han culminado la trayectoria de los estados que tienen asociados.

- Módulo de “Gestión de Datos”: En dicho módulo se gestionara los documentos que se tienen incorporados a la aplicación, permitiendo el traslado de un documento de una colección a otra, asociar colecciones a una taxonomía. Además se podrá editar toda la información de una colección, taxonomía y documento.

- Módulo de “Administración”: En este módulo se podrá realizar el manejo de usuarios, incorporar nuevas funcionalidades a la aplicación. Así como funcionalidades dependiendo del rol del usuario, tales como la edición y eliminación de los estados por los que ha pasado un documento, la asociación de funcionalidades con roles, entre otras.

Los módulos que aparecen en la figura 2.3 en gris, son módulos que se tomaron en cuenta al momento de empezar el desarrollo de la aplicación pero que no se llevaron a cabo en este Trabajo Especial de Grado y quedaron como mejoras para una nueva versión del prototipo desarrollado. Entre dichos módulos se encuentra la incorporación del Reconocimiento Óptico de Caracteres, la indexación a través de palabras claves de los documentos y el desarrollo de un módulo para manipular procesos en segundo plano o *background,* que permita iniciarlos y detenerlos de forma remota.

CAPÍTULO III: MARCO APLICATIVO

3.1. Iteración 0

3.1.1. Planificación

Iteración 0			
Descripción	Investigación, Instalación y Selección del Manejador de Versiones a utilizar en la aplicación para los Documentos.		
Fecha Inicio / Fecha Fin	07 – 10 – 2010 / 14 – 10 – 2010		
Número	Fecha	Historia	Tipo
1	07 – 10 – 2010	Instalación y configuración de Subversion.	Nueva
2	09 – 10 – 2010	Instalación y configuración de Git	Nueva
3	10 – 10 – 2010	Pruebas de Rendimiento de Subversion y Git.	Nueva
4	14 – 10 – 2010	Selección de Manejador de Versiones a Utilizar en la Aplicación	Nueva

Tabla 3.1 - Planificación Iteración 0

3.1.2. Codificación

En la presente iteración se realizo inicialmente una investigación de cuáles son los manejadores de contenido más utilizados y además de las ventajas y desventajas de cada uno de estos.

Al culminar dicha investigación se escogieron dos sistemas manejadores de versiones para realizar las pruebas necesarias. Estos manejadores fueron Subversion y Git.

Posteriormente se realizo la instalación de ambos manejadores de Versiones y se realizaron las pruebas de rendimiento sobre cada uno de estos.

3.1.3. Pruebas

Las pruebas realizadas en la presente iteración, consistieron en realizar una subida de archivos a un servidor de Subversion y a un servidor de Git.

Primero se realizaron las pruebas con un servidor local y luego con el servidor de Subversion del CCPD y con Github.

La prueba local se realizo en una máquina Intel Core Duo de 2.00GHz, con procesador de 64 bits. Las pruebas se realizaron bajo el sistema operativo Ubuntu 10.10.

A continuación se presentan los datos obtenidos en las pruebas de rendimiento realizadas.

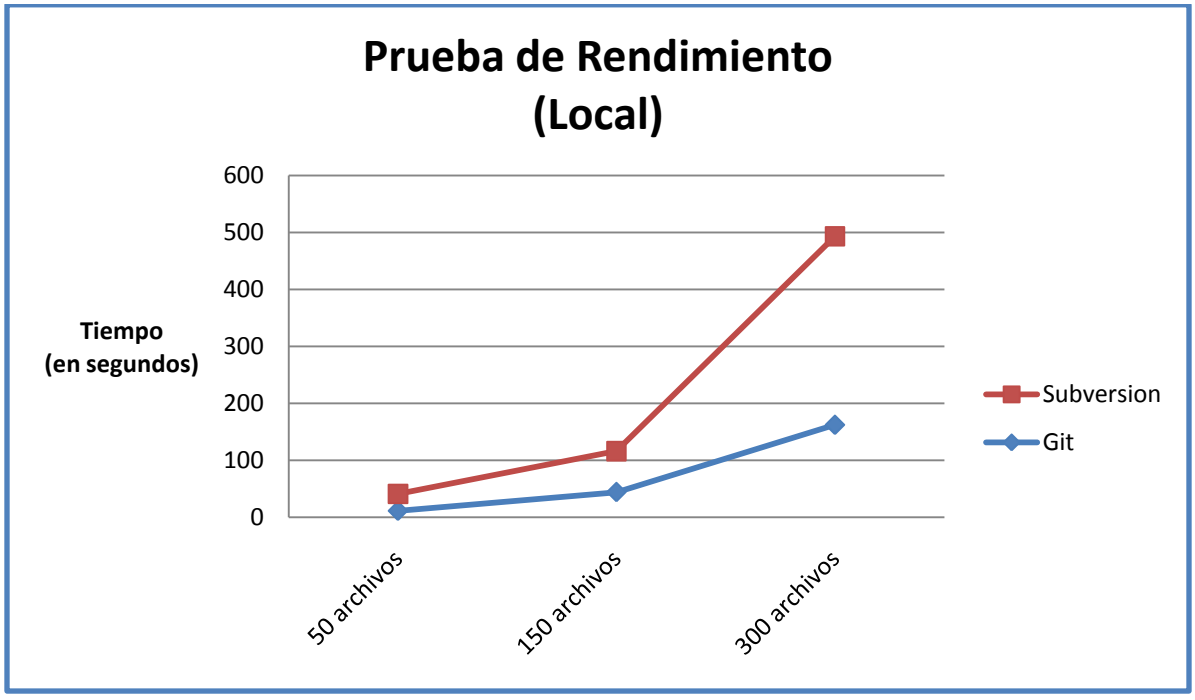

Figura 3.1 - Prueba de Rendimiento Git y Subversion de forma local

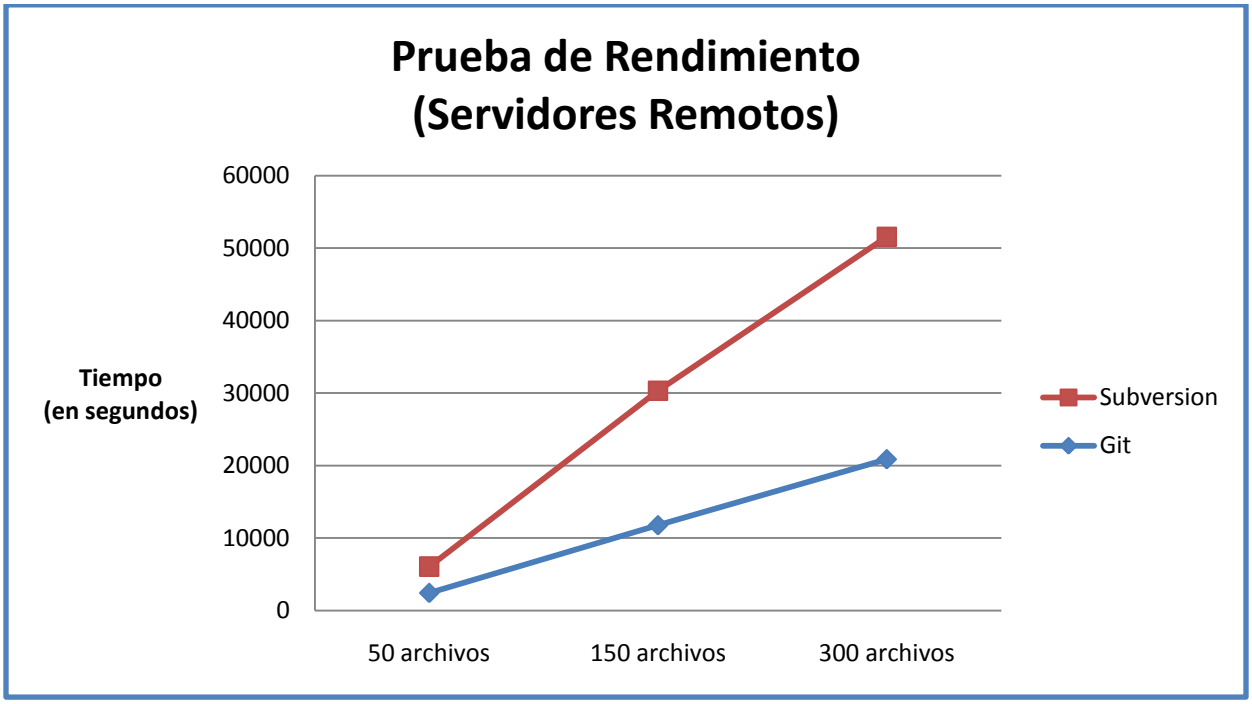

Figura 3.2 - Prueba de Rendimiento Git y Subversion en servidores remotos

En las pruebas realizadas con ambos manejadores de versiones, se pudo observar que la carga de archivos es más rápido en el manejador de versiones Git, tanto localmente como remotamente. Entre mayor era la cantidad de archivos a subir el tiempo de diferencia entre un manejador de versiones y otro era mayor.

Por esta razón se decide escoger Git como el manejador de versiones sobre el cual trabajara la aplicación, para realizar el manejo de las versiones de los documentos que se incorporen a la aplicación.

3.2. Iteración 1

3.2.1. Planificación

Iteración 1			
Descripción	Diseño del Modelo de Datos e Instalación y configuración del ambiente de desarrollo		
Fecha Inicio / Fecha Fin	15 – 10 – 2010 / 28 – 10 – 2010		
Número	Fecha	Historia	Tipo
5	15 – 10 – 2010	Diseño inicial del modelo de la base de Datos.	Nueva
6	26 – 10 - 2010	Instalar y configurar Ruby on Rails	Nueva

Tabla 3.2 - Planificación Iteración 1

3.2.2. Diseño

En la figura 3.3 se muestra el diagrama entidad-relación de las estructuras de datos de la aplicación. En ella se puede apreciar tanto la data esencial de los documentos como las estructuras para la clasificación de los mismos. El diagrama expresa de manera sencilla la naturaleza del problema y la solución.

El modelo de datos planteado está compuesto por ochos tablas, entre las que se encuentra la tabla documentos la cual se requiere para almacenar todos los documentos y la información en común que tienen, en la tabla características se tendrán todas las características que pueden tener un documento en particular las cuales se asociaran al documento a través de la tabla relación características en documentos. Los documentos se encuentran relacionados con los estados a través de la tabla “Posee”, en la que se almacenaran todos los estados por los que pasa un documento luego de ser incorporado a la aplicación.

Para clasificar los documentos se cuenta con la tabla “Colecciones”, la cual es una tabla tipo árbol en donde se encontraran todas las colecciones a las cuales puede pertenecer un documento y estas colecciones a su vez pueden pertenecer a otras colecciones. La relación entre la tabla documentos y la tabla colecciones es uno a uno.

Cada colección tiene asociada una taxonomía, dichas taxonomías también se encuentran en una tabla tipo árbol.

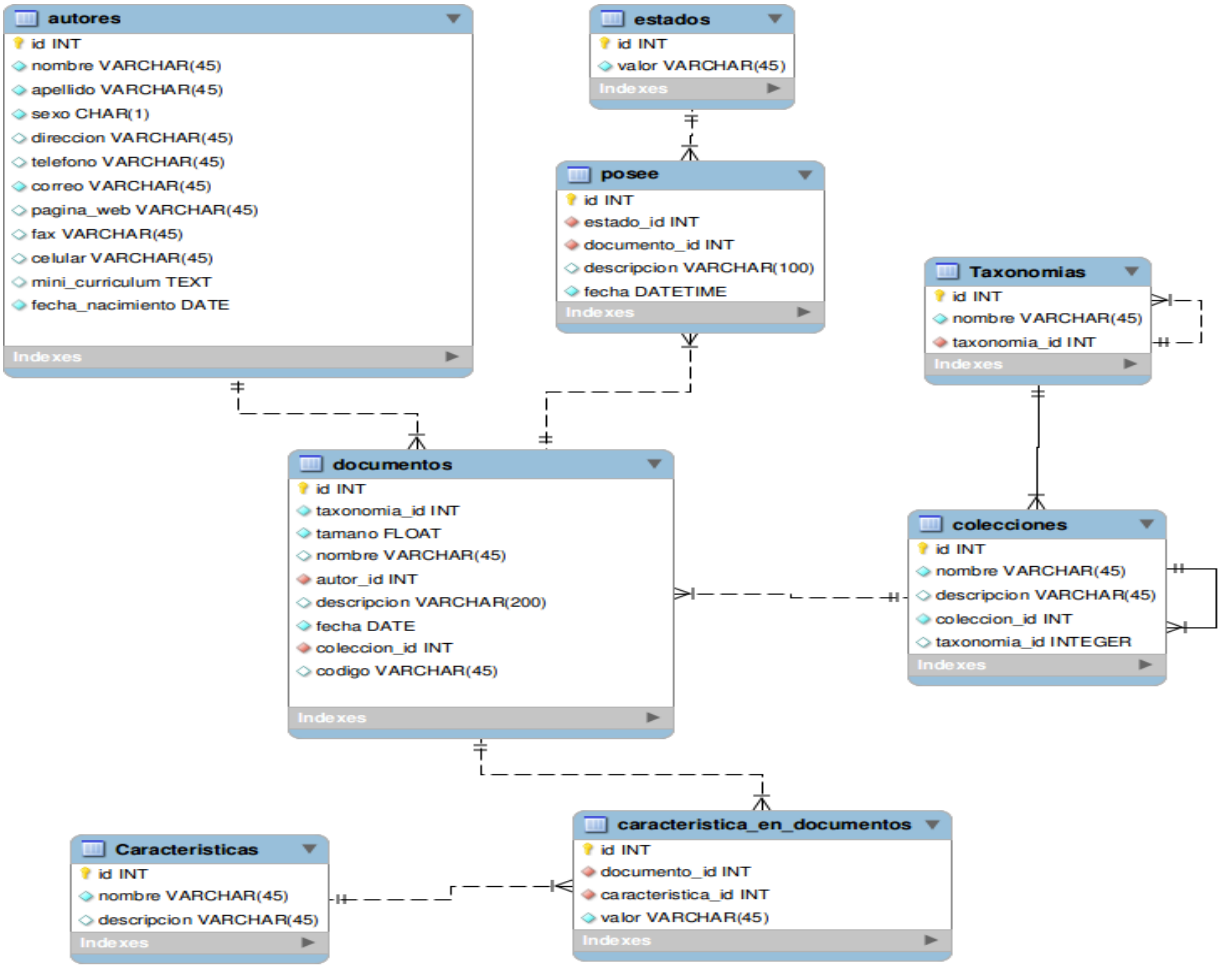

Figura 3.3 - Modelo de Base de Datos

3.2.3. Codificación

En la presente iteración se procedió a instalar todo lo necesario para el desarrollo de la aplicación, en mi máquina de trabajo personal, algunos de los programas instalados fueron: MySQL, MySQL Workbench, Ruby 1.9.2, Rails 3.0.0 en conjunto con todas las dependencias necesarias. Es importante destacar que dicha instalación fue realizada sobre el ambiente Linux, específicamente en su distribución Ubuntu 10.10, la cual se documento con el fin de aportar una guía (ver apéndice 1) que sirva de apoyo para futuras instalaciones.

3.3. Iteración 2

3.3.1. Planificación

Iteración 2			
Descripción	Creación de Vistas para el Módulo de Gestión de Datos, específicamente Colecciones y Taxonomías.		
Fecha Inicio / Fecha Fin	28 – 10 – 2010 / 25 – 11 – 2010		
Número	**Fecha**	**Historia**	**Tipo**
7	28 – 10 - 2010	Diseño de las Interfaces	Nueva
8	04 – 11 - 2010	Desarrollar un método que permita cambiar los documentos de Colecciones.	Nueva
9	18 – 11 - 2010	Desarrollar un método que permita cambiar las colecciones de taxonomías.	Nueva
10	25 – 11- 2010	Modificación del Diseño de Base de Datos	Modificación /Mejora

Tabla 3.3 - Planificación Iteración 2

3.3.2. Diseño

El desarrollo de las vistas asociadas a la gestión de colecciones y taxonomías consistió en la agregación de todas las interfaces necesarias para gestionar los

documentos pertenecientes a las colecciones y las taxonomías de cada colección. Por lo tanto el usuario, podría a través de un "Agarrar y Soltar" cambiar de colección un documento o de taxonomía una colección.

Debido a las necesidades encontradas, se planteó un diseño sencillo y fresco, en el cual en una misma pantalla se encontraran las diferentes funcionalidades, divididas por pestañas. Entre las que se encuentran:

- Colecciones, donde se consultan los documentos que se encuentran en una colección y se puede realizar el cambio de dicho documento de una colección a otra;

- Taxonomías, donde se puede consultar todas las colecciones y cambiar dicha colección de la taxonomía a la que pertenece.

- Gestión de Colecciones, Gestión de Taxonomías y Gestión de Documentos. En cada una de estas se muestra la información correspondiente a cada una y se permite editar la misma.

En la figura 3.4, se muestra la interfaz inicial de dicho módulo en la pestaña de colecciones. Del lado derecho se puede observar un árbol de todas las colecciones cargadas al sistema y del lado derecho todos los documentos cargados a la base de datos.

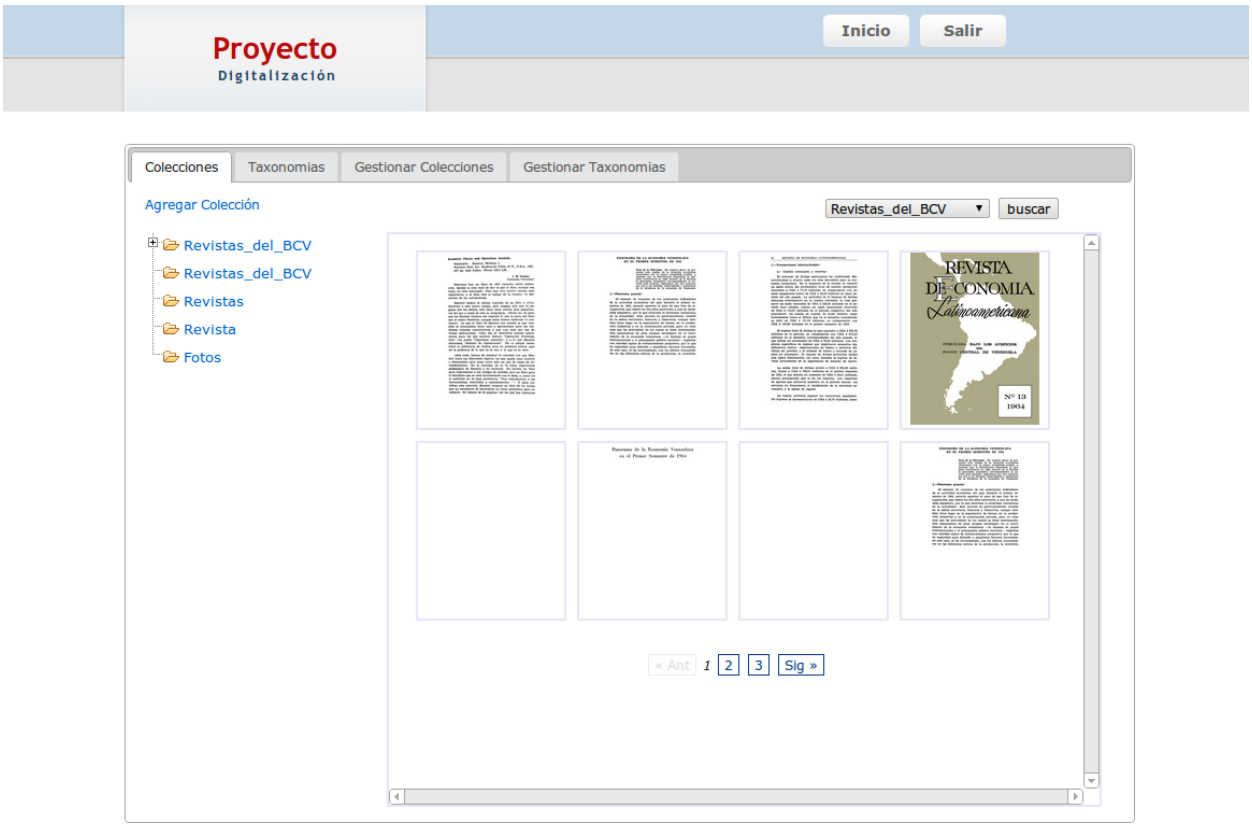

Figura 3.4 - Interfaz Módulo de Gestión de Colecciones, Taxonomías y Documentos

Al diseño inicial de la base de datos se le agrego una nueva tabla, la cual es una tabla relación entre características y colecciones. En esta tabla se almacenaran las diferentes características que pueden poseer las diferentes colecciones, las cuales no se contemplan en la tabla colección y estas características sean propias de la colección y no del documento. En la figura 3.5 se muestra dicha relación.

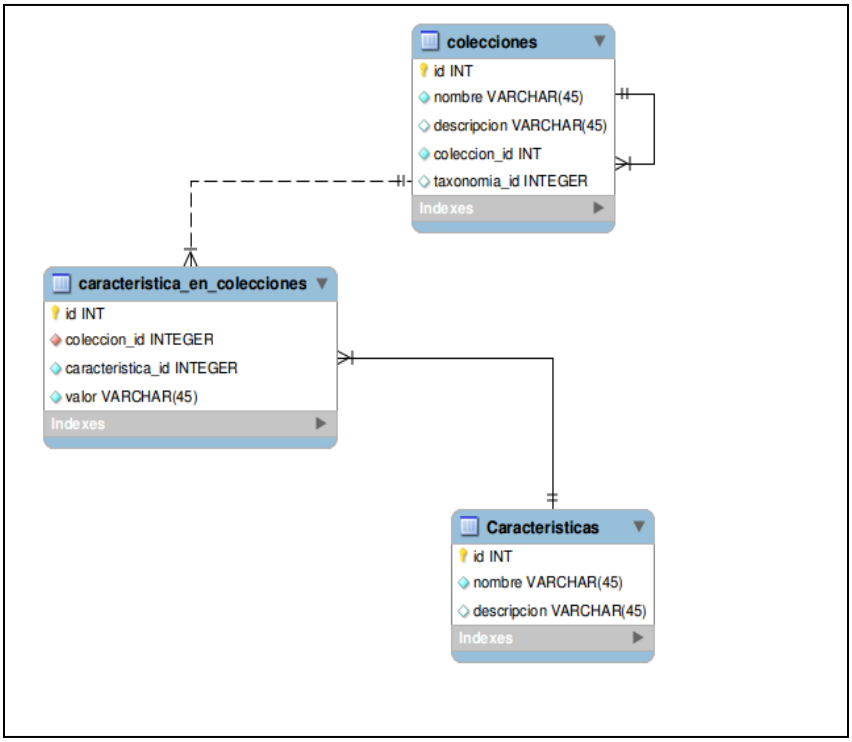

Figura 3.5 - Modelo de Base de Datos. Relación Colección - Características

3.3.3. Codificación

Con el fin de cumplir con los requerimientos planteados en la presente iteración, en primer lugar se creó un *Helper* de Rails para extraer los datos de las tablas colecciones y taxonomías y crear los árboles correspondientes a cada uno y de estar forma poder crear el menú en forma de árbol con ayuda del "Pluggin Acts_as_tree". En la figura 3.6, se puede observar parte del código correspondiente a dicho *Helper*.

```
module PrincipalHelper

def display_colecciones(colecciones)
   aux = "<ul id='browser' class='filetree' >"
     for coleccion in colecciones
       if coleccion.parent_id == nil
         aux += '<li><a href="#" class="treeItem" id="'
          aux += coleccion.id.to_s
          aux +='"><span class="folder">'
          aux += coleccion.nombre
          aux += '</span></a>'
          if coleccion.children.size > 0
              aux += find_all_subcategories(coleccion)
          end
          aux += '</li>'
       end
     end
  aux += '</ul>'
  render :inline => aux
end

def find_all_subcategories(category)
    if category.children.size > 0
      aux = '<ul>'
      category.children.each { |subcat|
        if subcat.children.size > 0
          aux += '<li><a href="#" class = "treeItem" id="'
            aux += subcat.id.to_s
            aux +='"><span class="folder">'
          aux += subcat.nombre
          aux += '</span></a>'
          aux += find_all_subcategories(subcat)
          aux += '</li>'
        else
          aux += '<li><a href="#" class = "treeItem" id="'
            aux += subcat.id.to_s
            aux +='"><span class="folder">'
          aux += subcat.nombre
          aux += '</span></a></li>'
        end
      }
      aux += '</ul>'
   end
end
```

Figura 3.6 – Construcción de los menú en forma de árbol.

Posteriormente, se procedió a realizar el método al cual se llamaría vía Ajax para guardar los cambios cuando se mueve un documento de una colección a otra o se cambia una colección de taxonomía. A continuación en la figura 3.7, se muestra el código encargado de guardar los cambios en la base de datos y a la vez mover los documentos de una carpeta a otra (si la acción es cambiar un documento de colección).

```
def guardar
 id_coleccion = params[:id_coleccion]
 id_documento = params[:id_documento]
 tabs = params[:tabs]

 if tabs == "coleccion"
  c = Coleccion.find_by_id(id_coleccion)
  if c.parent_id.nil?
        referencia_interna = "#{c.nombre}"
  else
       referencia_interna = "#{c.nombre}"
        while !c.parent_id.nil? do
         c = Coleccion.find_by_id(c.parent_id)
         referencia_interna = "#{c.nombre}/#{referencia_interna}"
        end
  end
     referencia_interna_old = Documento.find_by_id(id_documento).referencia_interna
     documento = Documento.update(id_documento, :coleccion_id => id_coleccion, :referencia_interna => referencia_interna)

     FileUtils.mv "public/resources/#{referencia_interna_old}/#{documento.documento_file_name}", "public/resources/#{referencia_interna}", :force => true
```

Figura 3.7 – Actualización de Colecciones y Taxonomías en la base de datos.

3.3.4. Pruebas

Se realizaron pruebas unitarias en cada uno de los pasos, dichas pruebas consistieron en:

- La comprobación de la creación de los menú tipo árbol de taxonomías y colecciones. En donde se probó la creación correcta de dichos menús en los diferentes casos que podrían ocurrir, tales como son que existieran solo Colecciones y Taxonomías que no contengan Colecciones o Taxonomías hijas, o que existieran Colecciones y Taxonomías con diferentes hijos y que estos contuvieran otras Colecciones y Taxonomías. Estas pruebas revelaron el correcto funcionamiento de este requerimiento.

- Comprobar que se ejecute el cambio de documentos de las colecciones y de las colecciones de taxonomías mediante las acciones de Agarrar y

Soltar. Los resultados que se obtuvieron fueron que al momento de agarrar un documento y soltarlo sobre una colección, en el menú tipo árbol se cambiaba el documento de la colección y estos cambios se guardaban en la base de datos mediante la llamada Ajax.

3.4. Iteración 3

3.4.1. Planificación

Iteración 3			
Descripción	Desarrollo de los módulos de gestión de Colecciones y Taxonomías y modificaciones a la base de datos. Definición de Roles de Usuarios.		
Fecha Inicio / Fecha Fin	03 – 12 – 2010 / 20 – 01 - 2011		
Número	**Fecha**	**Historia**	**Tipo**
11	03 – 12 - 2010	Desarrollar un método que permita la edición de las colecciones.	Nueva
12	03 – 12 - 2010	Desarrollar un método que permita la edición de las taxonomías.	Nueva
13	15 – 12- 2010	Desarrollar un método que permita la edición de los documentos.	Nueva
14	15 – 12- 2010	Crear las tablas de usuario, roles, funcionalidades, rol en funcionalidades y módulos.	Modificación /Mejora
15	15 – 12 – 2010	Definir los Roles de Usuario.	Nueva

Tabla 3. 4 - Planificación Iteración 3

3.4.2. Diseño

En la figura 3.8, se puede visualizar la interfaz para la edición de las colecciones. En dicha vista el usuario podrá seleccionar de la lista de colecciones, presentada en modo árbol que se encuentra del lado izquierdo de la página, cual es la colección a editar y se cargara vía Ajax el recuadro del lado derecho de la página con el formulario para la edición. Para comodidad del usuario la edición de las colecciones se realiza en un

formulario por partes, ya que en este caso además de la información básica de una colección también se puede editar las características de la misma.

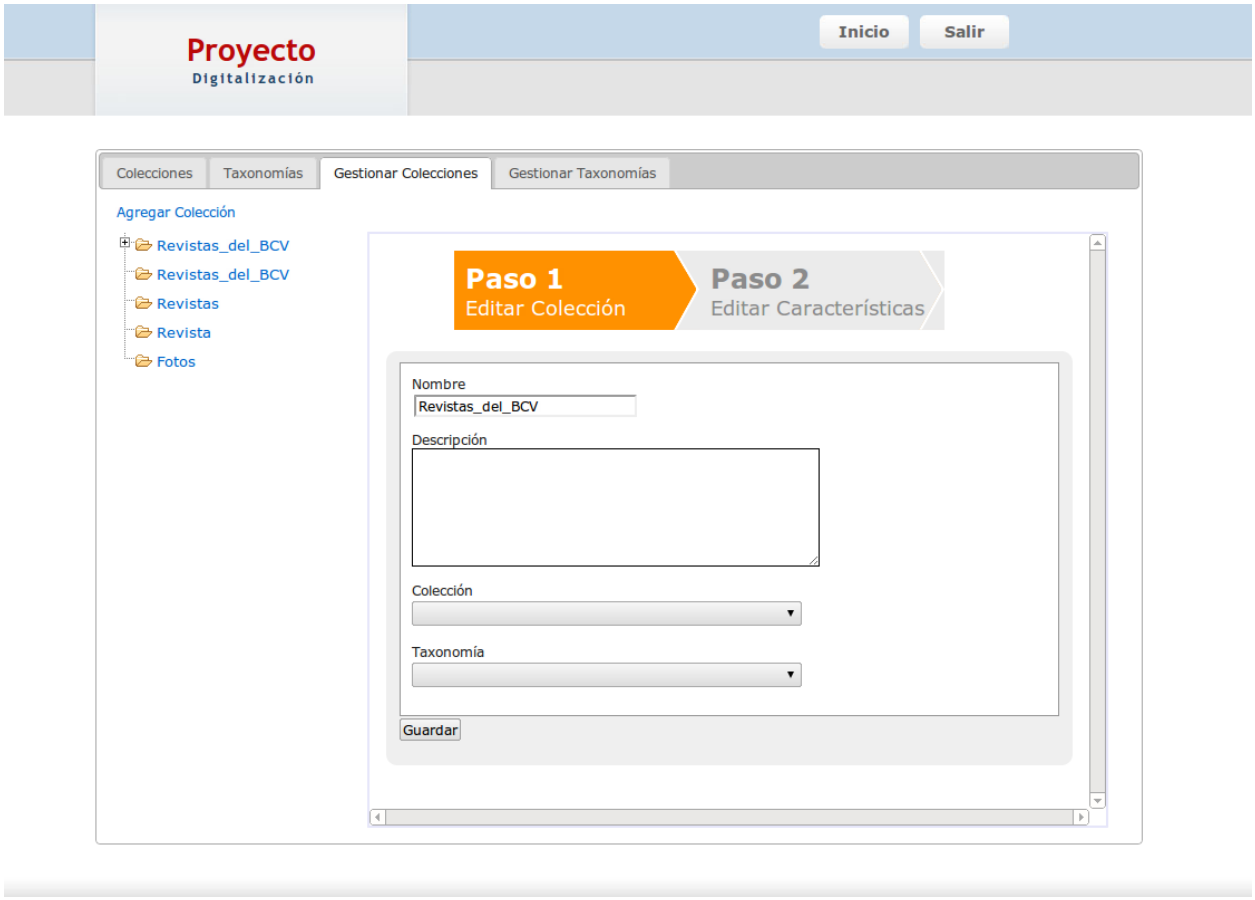

Figura 3.8 – Diseño para la edición de Colecciones

La interfaz para la edición de Taxonomías y de Documentos sigue la misma estructura con la diferencia que en la edición de Taxonomías no se cuenta con un formulario por partes. En la figura 3.9 se puede observar la interfaz para la edición de Taxonomías.

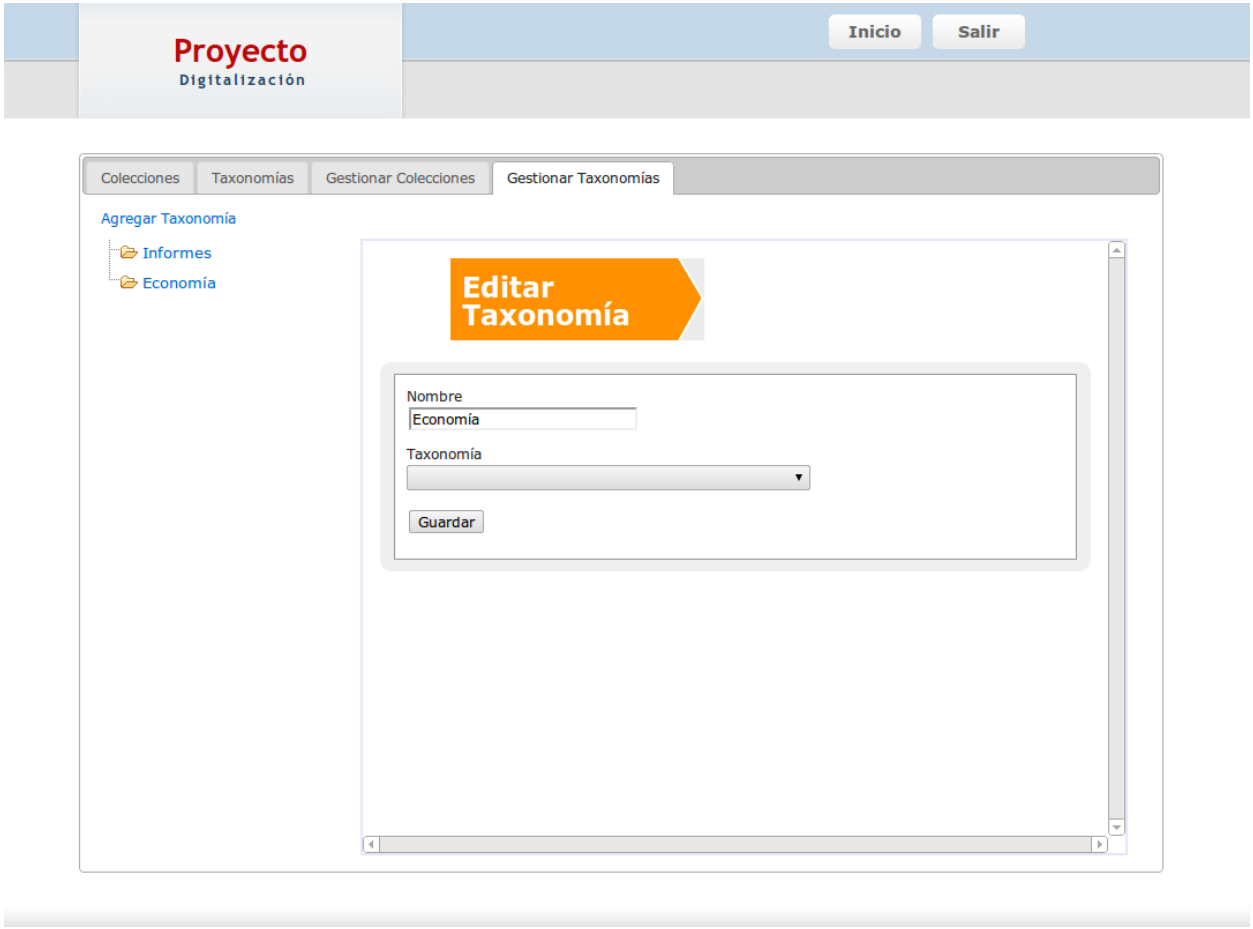

Figura 3.9 – Diseño para la edición de taxonomías

De igual forma en este mismo módulo se cuenta con un enlace para agregar una taxonomía, el cual permite desde esta misma vista, a través de un Overlay, crear una taxonomía nueva. En la figura 3.10, se puede observar cómo se presenta dicho formulario.

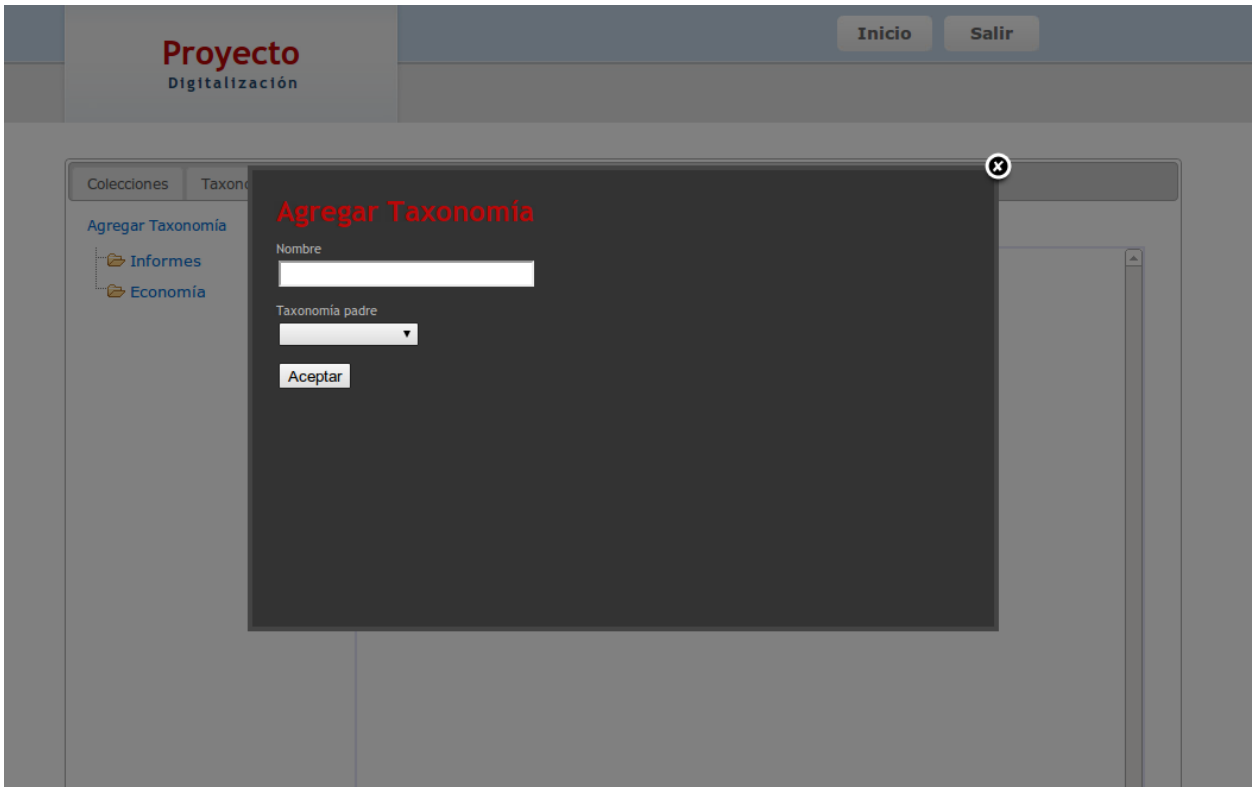

Figura 3.10 – Diseño para agregar nueva taxonomía

En esta iteración se crearon cinco tablas nuevas en la base de datos, las cuales permitirían realizar el manejo de usuarios, roles y funcionalidades y las relaciones entre los mismos. En la figura 3.11, se muestra la estructura de las cinco tablas y su relación.

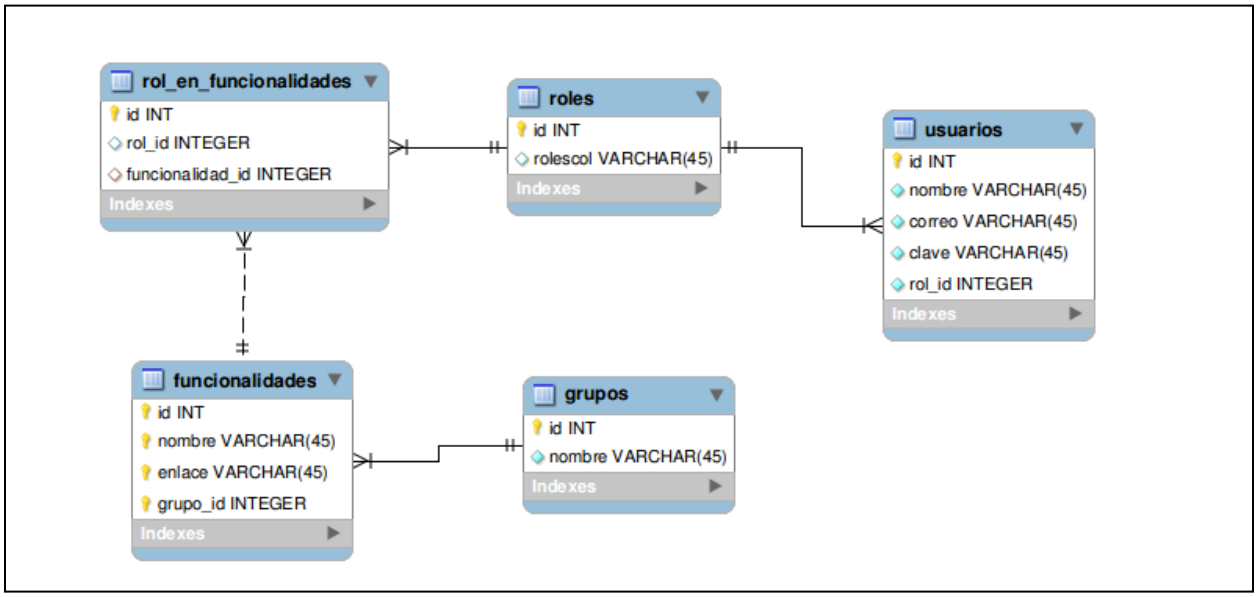

Figura 3.11 – Modelo de Base de Datos. Tablas para el manejo de Usuarios

En la tabla funcionalidades se cuenta con un atributo denominado enlace, en el cual se especificara el nombre de la acción a la que se hace referencia dentro del controlador, que se identificara con el nombre del grupo.

Este diseño de base de datos permite que la aplicación sea extensible de una forma más sencilla, ya que no es necesario modificar la aplicación completa para agregar o eliminar nuevas funcionalidades y esto hace que el crecimiento de la aplicación sea sencillo y no presente inconvenientes.

Los roles de usuarios que se manejaran en un principio en la aplicación son:

- Superadministrador: Dispondrá de todos los privilegios de la aplicación.
- Administrador: tendrá privilegios de administración pero limitados
- Cargador: Solo podrá incorporar documentos a la aplicación.
- Usuario: Podrá ejecutar cualquier funcionalidad menos las de administración.

3.4.3. Codificación

Para proceder con la edición de las colecciones se hace una llamada vía Ajax en la cual se busca en la base de datos la colección a editar y luego de completar la edición se procede a guardar en la base de datos los cambios realizados. En la figura 3.12, se muestra parte del código que se ejecuta en la llamada Ajax para obtener la colección que se va a editar.

```
def editar_coleccion
  @coleccion = Coleccion.find(params[:id])

  respond_to do |format|
      if @coleccion
        format.js { render :js => @comment, :location => @comment }

      end
end
end

def editar_taxonomia
  @taxonomia = Taxonomia.find(params[:id])
```

Figura 3.12 – Código para la edición de Colecciones.

Cuando se edita el nombre de una colección se debe renombrar la carpeta perteneciente a dicha colección y luego se debe proceder a actualizar el campo de referencia_interna de todos los documentos pertenecientes a esa colección.

3.4.4. Pruebas

Las pruebas de aceptación que se realizaron en esta iteración consistieron en:

- La Creación de Nuevas Taxonomías. Para esto se ingresaron diferentes taxonomías de prueba, para comprobar que no ocurriera ningún error en la creación de las mismas. Se obtuvo como resultados que se agregaron todas las taxonomías que tuvieran los datos mínimos exigidos por el sistema, el cual en este caso es el nombre de la taxonomía.

- La edición de Taxonomías y Colecciones, para esta edición se intentaron ingresar datos erróneos para comprobar que la aplicación mostrara los mensajes de error y no se guardarán estas modificaciones. Una de estas pruebas consistió en editar el nombre de una Colección por el nombre de otra Colección ya existente y se obtuvo como resultado que no se

guardaron los cambios y mostro el mensaje de error "Ya existe una Colección con ese nombre". Además, también, se hicieron ediciones de taxonomías y colecciones que fueran correctas para comprobar que se guardaran las actualizaciones si todo esta correcto. Se obtuvo como resultado que si se cumple con las validaciones de datos obligatorios y la validación de que el nombre de la colección es único entonces se guarda dicha actualización.

3.5. Iteración 4

3.5.1. Planificación

Iteración 4			
Descripción	Desarrollo de los módulos de Manejo de Usuarios, funcionalidades, rol en funcionalidades y módulos. Autenticación. Utilización de Simple Captcha.		
Fecha Inicio / Fecha Fin	20 – 01 – 2011 / 18 – 02 – 2011		
Número	**Fecha**	**Historia**	**Tipo**
16	20 – 01 – 2011	Desarrollar un módulo en donde se pueda agregar, eliminar y editar usuarios.	Nueva
17	20 – 01 – 2011	Desarrollar un método que encripte la clave de usuario antes de guardar.	Nueva
18	20 – 01 – 2011	Desarrollar un módulo para el manejo de funcionalidades, las cuales se puedan activar y desactivar.	Nueva
19	18 – 02 – 2011	Desarrollar un método para asociar los roles con las funcionalidades y para agregar módulos en los que se agrupen funcionalidades.	Nueva
20	18 – 02 – 2011	Desarrollador un Buscador en Colecciones y Taxonomías.	Modificación/ Mejora

Tabla 3. 5 - Planificación Iteración 4

3.5.2. Diseño

Para visualizar las listas de usuarios, módulos, funcionalidades y rol en funcionalidades se realizaron unas tablas las cuales muestran los datos de cada uno y permiten ver los detalles, editar o eliminar cualquiera de estos.

En la figura 3.13, se muestra la lista de funcionalidades. En está se puede observar que cada funcionalidad tiene un atributo que se llama “Activo”, el cual se muestra como un *checkbox*, el cual si se encuentra seleccionado significa que dicha funcionalidad se encuentra activa. Esto permite que el usuario pueda activar o desactivar una funcionalidad sin tener que cambiar de pantalla y con la simple acción de seleccionar o deseleccionar dicho *checkbox*.

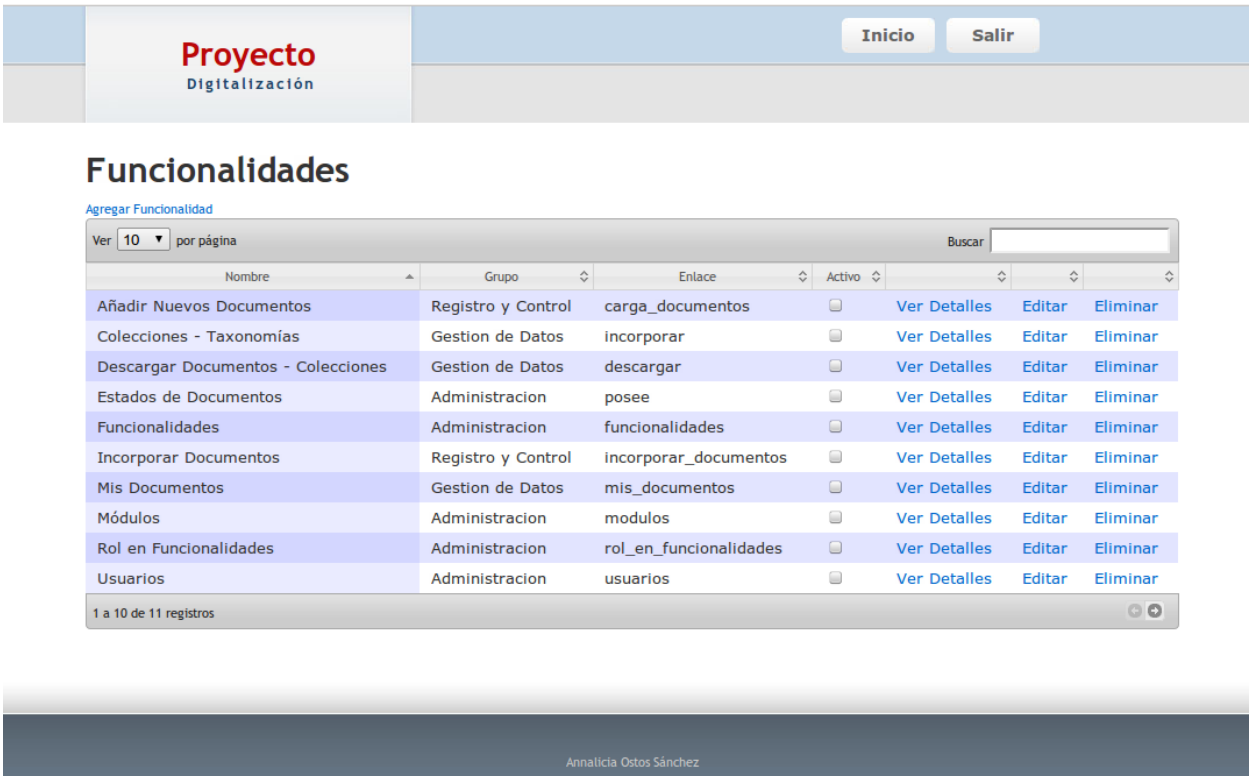

Figura 3.13 – Diseño Lista de Funcionalidades

Desde esta pantalla se tiene un enlace que permite agregar una nueva funcionalidad, la cual es posible a través del formulario que se muestra en la figura 3.14. También en esta pantalla se cuenta con un enlace para editar las funcionalidades, el cual muestra un formulario igual al de la figura 3.14 pero con los datos de la funcionalidad cargados en el formulario.

Figura 3.14 – Formulario para agregar nueva funcionalidad.

Los módulos de usuarios, rol en funcionalidades y módulos fueron diseñados igual que los mostrados en las figuras anteriores.

En esta iteración también se desarrollo un buscador en el módulo para cambiar los documentos de colección y las colecciones de taxonomía, realizado en la iteración 2. Este buscador permite consultar cuales son los documentos que se tienen en una colección en específico o cuáles son las colecciones que pertenecen a una taxonomía. En la figura 3.15, se puede observar cuales son las colecciones que pertenecen a la taxonomía con nombre “Informes”

En la figura 3.16, se puede observar el mensaje que se le muestra al usuario cuando escoge una taxonomía la cual no posee actualmente ninguna colección asociada. Este mensaje también aparece cuando se trata de una colección que no tiene ningún documento actualmente.

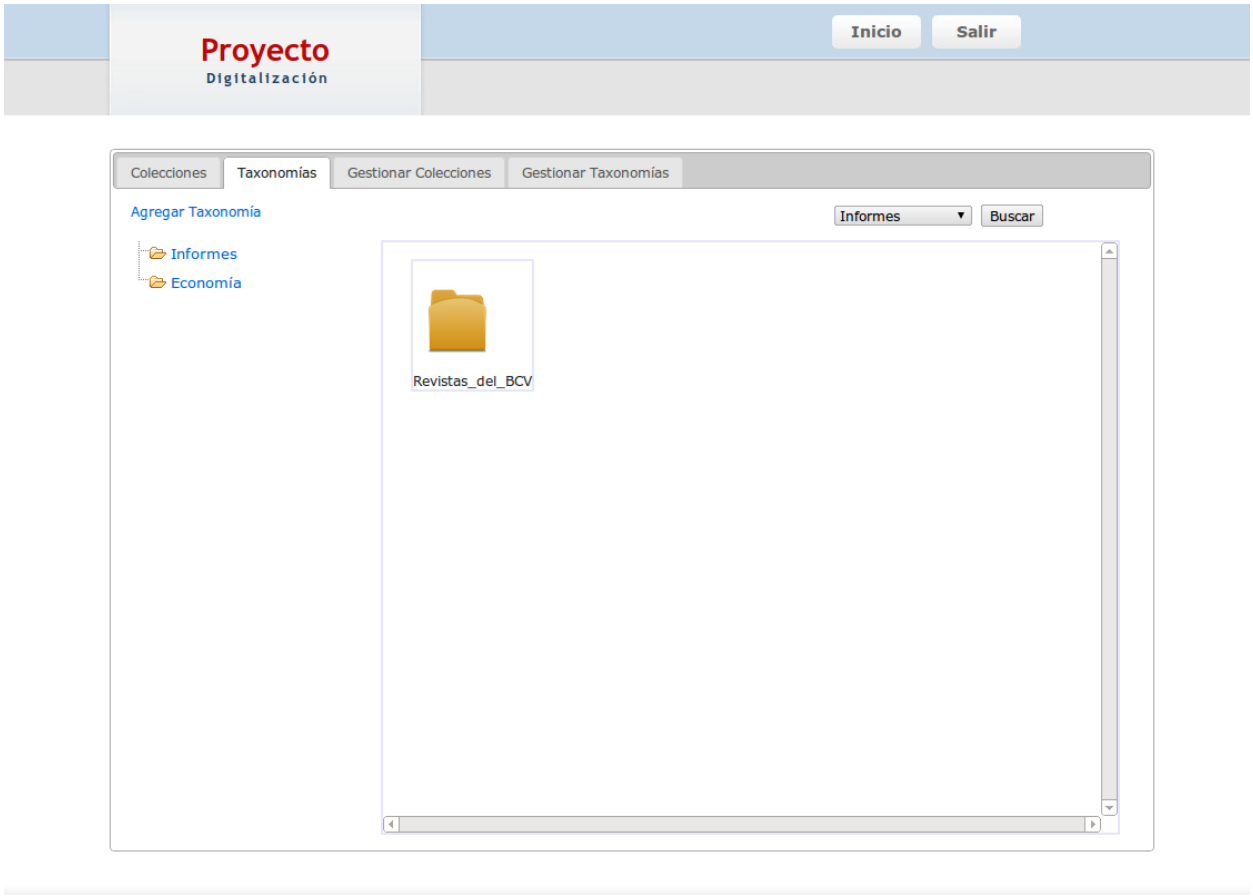

Figura 3.15 – Colecciones pertenecientes a una Taxonomía

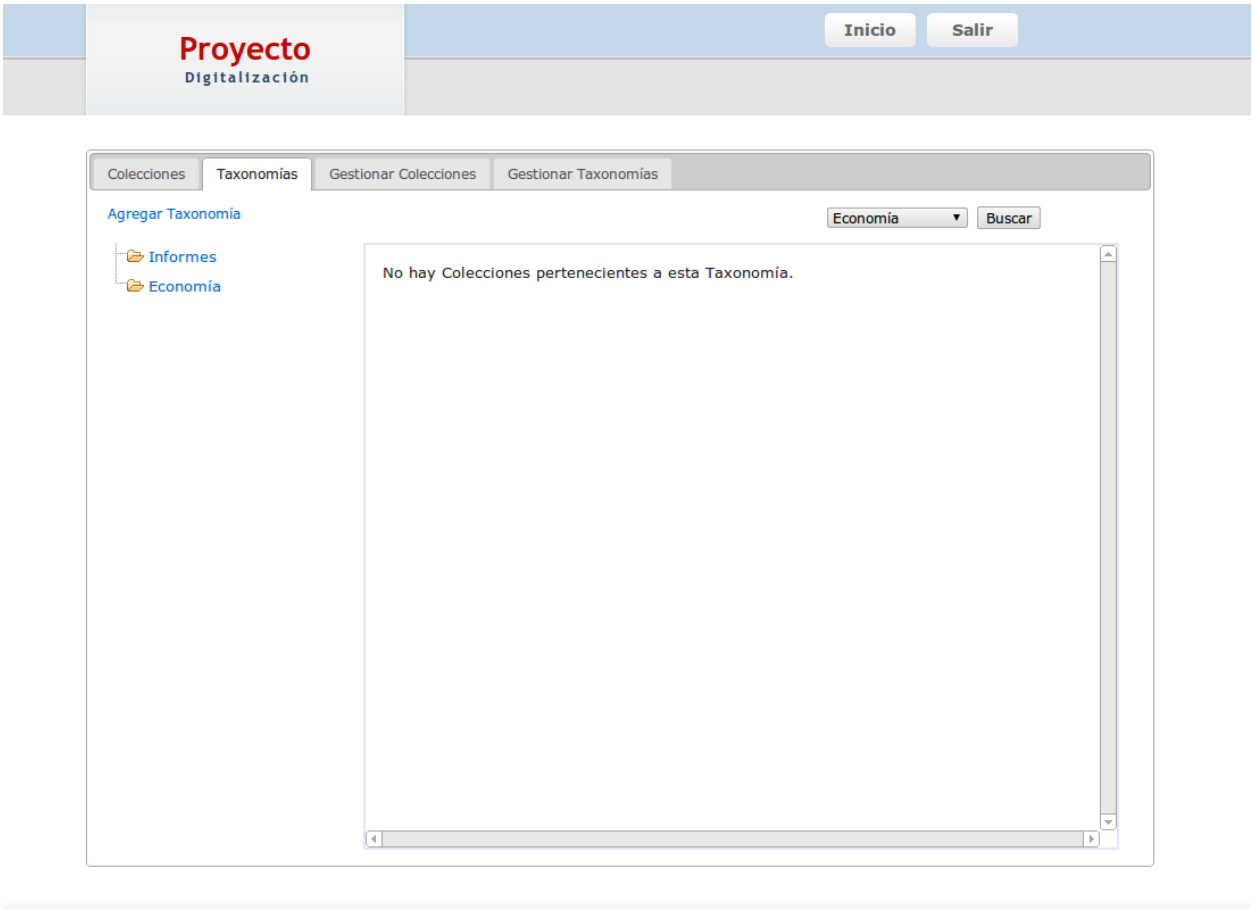

Figura 3.16 – Taxonomías sin Colecciones Asociadas.

3.5.3. Codificación

Tal cual como se especifico en la historia de usuario, se necesitaba desarrollar un módulo para gestionar los usuarios y además que se encriptara la clave de los mismos antes de ser guardada en la base de datos.

En la figura 3.17, se muestra el código que se procesa cuando en el index del módulo de usuarios se le da al enlace de "Agregar Nuevo Usuario". Este llama al método "new" en el cual, se crea la instancia del nuevo usuario a ser agregado y posteriormente se envía el formulario vía POST al método *create*. En el cual, antes de guardar el registro

en la base de datos, se encripta la clave ingresada por el usuario, tal como se aprecia en la figura 3.17.

```
class UsuariosController < ApplicationController

  def index
    @usuarios = Usuario.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @usuarios }
    end
  end

  def new
    @usuario = Usuario.new

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @usuario }
    end
  end

  def create

    clave = params[:usuario][:clave]
    params[:usuario][:clave] = Digest::SHA1.hexdigest("1#{clave}0").strip

    @usuario = Usuario.new(params[:usuario])

    respond_to do |format|
      if @usuario.save
        format.html { redirect_to(@usuario, :notice => 'Usuario was successfully created.') }
        format.xml  { render :xml => @usuario, :status => :created, :location => @usuario }
      else
        format.html { render :action => "new" }
        format.xml  { render :xml => @usuario.errors, :status => :unprocessable_entity }
      end
```

Figura 3.17 – Creación de nuevo Usuario.

Para la búsqueda de los documento pertenecientes a una colección o las colecciones pertenecientes a una taxonomía, se desarrollo un método de búsqueda el cual es llamado vía Ajax, en donde se le pasa por POST el id de la colección o taxonomía a ser consultada. En la siguiente imagen se muestra el código que se procesa para dicha búsqueda. Tal como se muestra en la figura 3.18

```
def buscar_colecciones

  buscar = params[:coleccion][:coleccion_id]
  @documentos = Documento.where("coleccion_id=#{buscar}")

end

def buscar_taxonomias

  buscar = params[:taxonomia][:taxonomia_id]
  @colecciones = Coleccion.where("taxonomia_id=#{buscar}")

end
```

Figura 3.18 – Código para la búsqueda de colecciones y taxonomías.

3.5.4. Pruebas

Para probar que el funcionamiento de los módulos desarrollados en dicha interacción funcionará correctamente, se agregaron diferentes funcionalidades, usuarios, módulos y los roles con sus funcionalidades asociadas.

Al realizar estas pruebas se obtuvo que dependiendo del rol del usuario que ingresa a la aplicación, se muestran las funcionalidades asociadas al rol del mismo y si se asocia o se elimina una nueva funcionalidad a dicho rol esta se muestra o desaparece del panel principal mostrado en la pantalla de inicio.

Otra de las pruebas realizadas consistió en comprobar que la dirección a la que redirige el enlace de la funcionalidad este formada de la siguiente forma, el nombre del controlador sea el nombre del grupo al que pertenece esa funcionalidad y el nombre de la acción a la que se llama dentro del controlador sea el nombre que se asigno en el atributo “enlace” de dicha funcionalidad.

Igualmente se realizaron pruebas sobre las búsquedas posibles en los módulos de colecciones y taxonomías. Para percatar de que no diera ningún error al momento de las llamadas Ajax y de los resultados que arrojaban dichas búsquedas.

3.6. Iteración 5

3.6.1. Planificación

Iteración 5			
Descripción	Desarrollar los módulos para Incorporar una Colección y agregar documentos nuevos a una colección ya existente.		
Fecha Inicio / Fecha Fin	18 – 02 – 2011 / 10 – 03 – 2011		
Número	**Fecha**	**Historia**	**Tipo**
21	18 – 02 – 2011	Desarrollar un método para crear una colección y posteriormente incorporar los documentos de la misma.	Nueva
22	18 – 02 – 2011	Desarrollar un método que permita la incorporación de los documentos de forma múltiple.	Nueva
23	18 – 02 – 2011	Desarrollar un método que permita agregar nuevos documentos a colecciones ya existentes.	Nueva
24	03 – 03 - 2011	Colocar El Captcha después del tercer intento erróneo.	Modificación / Mejora
25	03 – 03 – 2011	Se descarta a utilización del manejador de Versiones Git para la carga de archivos y seguimiento de las versiones de los mismos.	Modificación

Tabla 3. 6 - Planificación Iteración 5

3.6.2. Diseño

El ciclo de carga documentos comienza en el módulo de registro y control en la funcionalidad de incorporar documentos. Donde inicialmente se muestra un formulario por partes, el cual está compuesto por 3 pasos.

En el primer paso se le pide al usuario los datos de la colección a agregar y las características asociadas a dicha colección.

Para agregar las características el usuario puede escoger una de las almacenadas en el sistema o agregar una característica nueva. En la figura 3.19, se puede apreciar la primera parte de dicho formulario.

Figura 3.19 – Formulario por partes para incorporar Colecciones

Luego se le pedirá al usuario que escoja los estados por los cuales pasaran los documentos pertenecientes a dicha colección. Además de elegir cuales son los estados que poseerán los documentos el usuario elegirá el orden que estos deben seguir. En la figura 3.20 que se encuentra a continuación se muestra dicho paso del formulario.

Al culminar estos dos pasos la colección se almacenara en el sistema antes de proseguir con la carga de los documentos. Por esta razón se puede observar que el botón dice “Guardar y Continuar”

Figura 3.20 – Asociar Estados a una Colección

Por último, en el paso tres, tal como se aprecia en la figura 3.21, se cuenta con un botón que permite agregar múltiples documentos para luego proceder a la carga de los mismos.

Figura 3.21 –Vista para incorporar los Documentos a la Colección

En la figura 3.22, se puede apreciar que es posible realizar la carga de múltiples documentos, seleccionando los mismos con CTRL. Esto facilita y hace cómodo para el usuario la incorporación de los documentos a la aplicación. Debido a que no es necesario comprimir los documentos para incorporarlos ni esta obligado a incorporar todos los documentos al sistema de una sola vez.

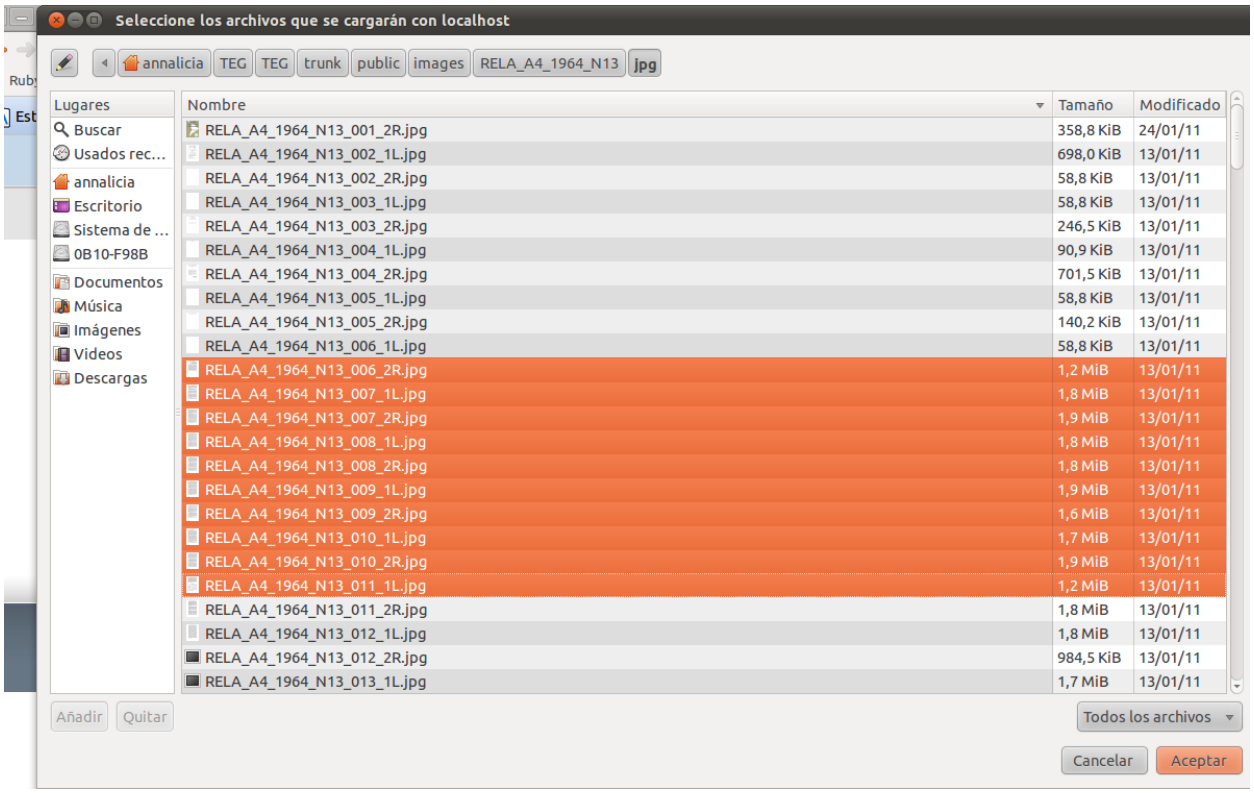

Figura 3.22 – Selección de varios Documentos para ser incorporados

Después de seleccionado los documentos a ser incorporados en la colección, se les mostrara al usuario una lista de los documentos que escogió permitiéndole que pueda revisarla y eliminar alguno de los documentos seleccionados o agregar otros documentos mas antes de comenzar la carga de los mismos. Tal cual como se muestra en la figura 3.23.

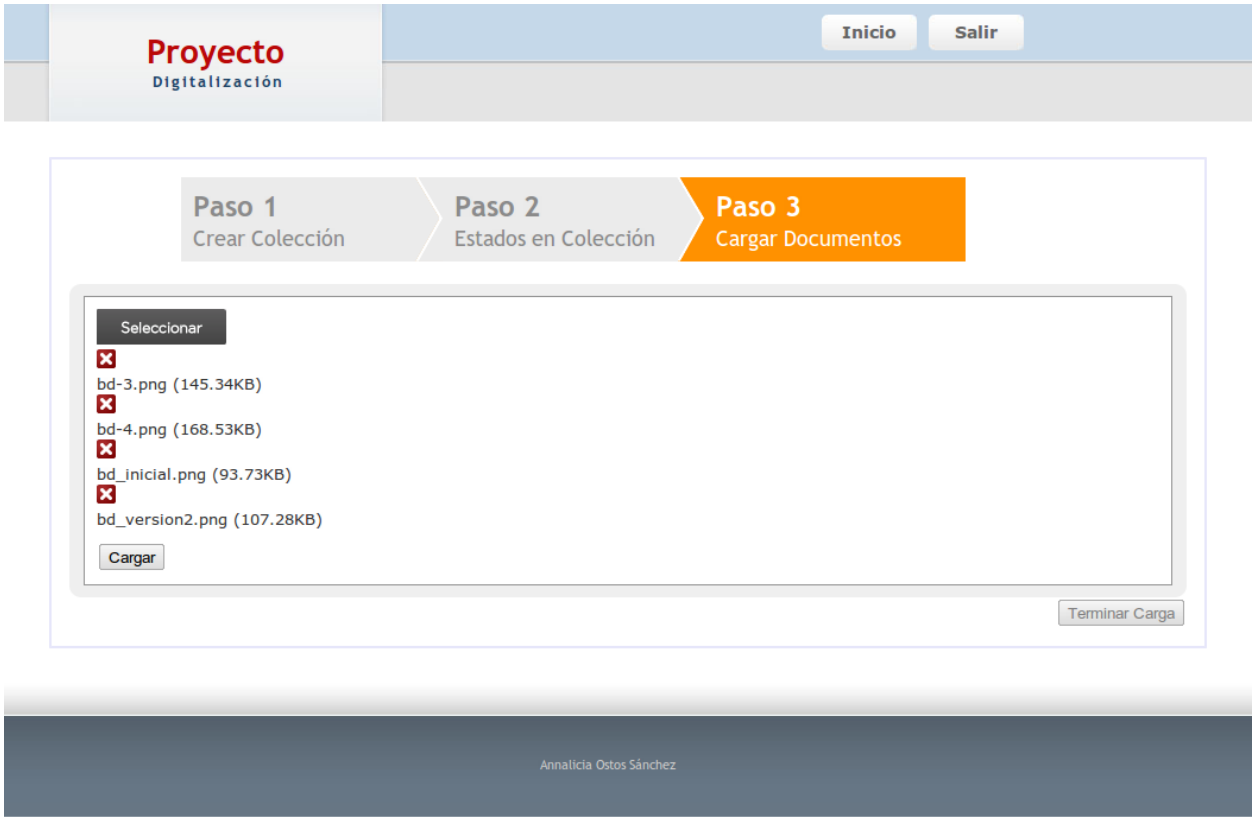

Figura 3.23 – Documentos próximos a incorporar.

Adicionalmente se desarrollo una funcionalidad para añadir o reemplazar documentos en colecciones ya creadas en el sistema. Lo que permitirá al usuario reemplazar documentos luego de realizarles algún cambio al mismo o incorporar nuevos documentos que no se tenían o no se pudieron incorporar al momento de la creación de la colección.

Para esto se diseño un formulario parecido al anterior, pero el cual solo consta de dos pasos, el primer paso se basa en escoger la colección a la cual pertenecerá el documento a incorporar y el segundo paso es igual al descrito anteriormente, para agregar los documentos que se van a incorporar a la aplicación.

En la figura 3.24, que se muestra a continuación se puede ver el primer paso de dicho formulario.

Figura 3.24 – Formulario por partes para agregar nuevos Documentos.

En esta iteración también se procedió a realizar la autenticación para el ingreso a la aplicación. En la figura 3.25, se pueden observar los datos que se piden para el ingreso a la aplicación.

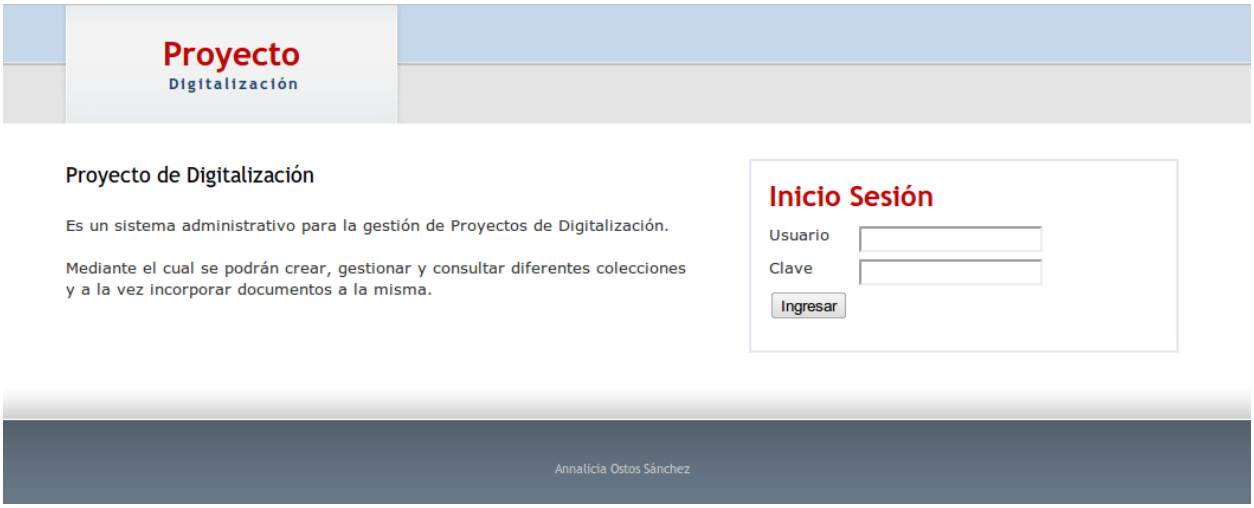

Figura 3.25 – Inicio de Sesión para ingresar a la Aplicación

Se decidió desechar la idea de la utilización de Git para el manejo y carga de los documentos, debido a que sería necesario desarrollar una aplicación de escritorio la cual el usuario debía descargar en su máquina para poder trabajar con GIt, lo cual hacía que el proceso de la carga de archivos fuera un poco engorroso.

Gracias a las investigaciones se encontró el plugin para la carga de archivos *Paperclip* el cual al realizar las pruebas se evidencio que cubría todos los requerimientos planteados inicialmente para la carga de archivos.

3.6.3. Codificación

Tal como se explico en la parte de Diseño, para la incorporación de documentos se consta con un formulario por partes, en donde primero se guarda la colección y luego se procede a la carga de los documentos.

En la figura 3.26, se muestra parte de la implementación para realizar guardar la colección. En este caso, se reciben todos los datos de la colección por POST. Primero se procede a guardar la colección llamando al método agregar definido en el modelo

Colección, si dicha colección es almacenada entonces se procede a almacenar las características de la misma en la tabla Característica en Colección mediante el método agregar definido en el modelo de dicha tabla. Para finalmente proceder a almacenar los estados que poseerán los documentos pertenecientes a la colección.

```
def guardar_coleccion

  n_estados = session[:estados]
  n_caracteristicas = session[:caracteristicas]
  session[:estados] = nil
  session[:caracteristicas] = nil

  coleccion = Coleccion.agregar(params[:incorporar][:nombre],params[:incorporar][:descripcion],params[:incorporar][:parent_id], params[:incorporar]
[:taxonomia_id])

  if coleccion

    coleccion_id = Coleccion.find_by_nombre(params[:incorporar][:nombre]).id

    session[:coleccion] = coleccion_id

    1.upto(n_caracteristicas){|i|

      if params[:incorporar]["nombre_#{i}"].empty?

      CaracteristicaEnColeccion.agregar(params[:incorporar]["caracteristica_id_#{i}"],coleccion_id,params[:incorporar]["valor_#{i}"])

    else

      caracteristica = Caracteristica.create(:nombre => params[:incorporar]["nombre_#{i}"])

      CaracteristicaEnColeccion.agregar(caracteristica.id,coleccion_id,params[:incorporar]["valor_#{i}"])
    end
    }

    params[:incorporar]["estado_id_0"] = Estado::INCORPORAR

    0.upto(n_estados){|i|

      EstadoEnColeccion.agregar(params[:incorporar]["estado_id_#{i}"],coleccion_id,i)
```

Figura 3.26 – Código para guardar Colección.

Para agregar los documentos a la aplicación, se utilizo la Gema *Paperclip* y el plugin de *Jquery Uploadify*, con los cuales se logro la carga múltiple de archivos y el almacenamiento de los archivos en la aplicación. En la figura 3.27, se muestra el código para guardar los archivos en la aplicación y almacenar la información de los mismos en la base de datos al momento de cargarlos.

```
def guardar_documentos

  if params[:documento]

    params[:documento][:documento].content_type = MIME::Types.type_for(params[:documento][:documento].original_filename).to_s

    params[:documento][:fecha] = Date.today

   params[:documento][:coleccion_id] = session[:coleccion] unless session[:coleccion].nil?

    c = Coleccion.find_by_id(params[:documento][:coleccion_id])

     session[:coleccion] = nil
    if c.parent_id.nil?

      params[:documento][:referencia_interna] = "#{c.nombre}"

    else

     referencia = "#{c.nombre}"

      while !c.parent_id.nil? do
       c = Coleccion.find_by_id(c.parent_id)
       referencia = "#{c.nombre}/#{referencia}"
      end

      params[:documento][:referencia_interna] = referencia

    end

    @documento = Documento.create(params[:documento])

    posee = Posee.create(:documento_id => @documento.id, :estado_id => Estado::INCORPORAR, :fecha => Time.now, :estado_actual => 1)
```

Figura 3.27 – Código para guardar Documentos.

Al momento de incorporar un documento en la aplicación, se guarda en la tabla “Posee” el primer estado por defecto para toda colección, el cual se denomina “Incorporado al Sistema”. De esta forma se lleva un control de cuando se incorporó el documento a la aplicación y el inicio de la secuencia de los estados por los cuales debe pasar.

Gracias a la Gema *Paperclip*, el documento se copia en el directorio especificado en la configuración de la Gema. En la figura 3.28, se muestra las especificaciones que debe tener el modelo sobre el que se trabajara dicha Gema. En las cuales se encuentra la URL para acceder al documento desde la aplicación y el PATH, la cual es la ruta de la imagen dentro del servidor.

```ruby
class Documento < ActiveRecord::Base

belongs_to :coleccion
belongs_to :autor

has_many :caracteristica_en_documentos
has_many :posee

has_attached_file :documento,

                  :path => ":rails_root/public/resources/:referencia_interna/:filename",
                  :url =>  "../resources/:referencia_interna/:filename"
```

Figura 3.28 – Configuración de la Gema *Paperclip* en el modelo.

A continuación, en la figura 3.29, se muestra parte de la implementación para la autenticación para poder ingresar a la aplicación. En esta parte se trabajo con la Gema de SimpleCaptcha. Este captcha se le muestra al usuario después del tercer intento fallido de ingresar al sistema.

```ruby
def autenticar

 login = params[:inicio]["usuario"]
 clave = params[:inicio]["clave"]

 unless (login.empty? || clave.empty?)

   clave = Digest::SHA1.hexdigest("1#{clave}0").strip

   usuario = Usuario.find(:first,:conditions => ["clave=? AND correo=?", clave, login]);

   if session[:autenticar] >= 3

     if usuario and simple_captcha_valid?

       session[:id] = usuario.id
       session[:correo] = usuario.correo
       session[:rol] = usuario.rol_id

       redirect_to :controller => 'principal', :action => 'index'
       session[:autenticar] = nil
       return
     else
       flash[:error] = "Usuario Invalido" unless usuario
       flash[:error] = "El codigo invalido" unless simple_captcha_valid?
       redirect_to :controller => 'inicio', :action => 'index'
       return
     end
```

Figura 3.29 – Código para la autenticación.

3.6.4. Pruebas

En esta iteración las pruebas se centraron en la creación de nuevas colecciones y la incorporación de documentos a las mismas, para de esta forma corroborar el buen funcionamiento de la Gema. Además de comprobar la eficiencia en la incorporación de documentos de gran tamaño.

Para estas pruebas se incorporaron al sistema diferentes documentos, de diferentes tamaños, para corroborar que el servidor no diera error cuando se intentaba subir documentos demasiado pesados. El tamaño de los documentos con los que se realizaron las pruebas estaba comprendido entre 10Kb hasta 100Mbits. Los resultados obtenidos fueron satisfactorios ya que aunque entre mayor era el tamaño del archivo mas tardaba la incorporación, esta no dio error de respuesta del servidor con ninguna carga de los archivos, uno de los principales problemas que se presentaba en la incorporación de volúmenes de gran tamaño en la aplicación del Banco Central de Venezuela.

3.7. Iteración 6

3.7.1. Planificación

Iteración 6			
Descripción	Desarrollar los módulos para verificar los documentos disponibles en el sistema y el módulo para consultar los documentos que tiene un usuario asignado en un momento dado.		
Fecha Inicio / Fecha Fin	10 – 03 – 2011 / 7 – 04 – 2011		
Número	**Fecha**	**Historia**	**Tipo**
26	10 – 03 – 2011	Desarrollar un método para verificar cuales son los documentos que no están siendo trabajados por un usuario.	Nueva
27	10 – 03 – 2011	Agregar una relación entre la tabla documentos y la tabla usuarios	Modificación/ Mejora
28	10 – 03 – 2011	Desarrollar un método que permita ver todos los documentos que tiene el usuario que está conectado en ese momento.	Nueva
29	10 – 03 – 2011	Desarrollar un método para poder descargar los documentos que se van a trabajar.	Nueva
30	24 – 03 - 2011	Permitir la visualización de los documentos y pasar los mismos.	Nueva
31	24 – 03 – 2011	Desarrollar un método para el cambio de estado de un documento.	Nueva

Tabla 3. 7 - Planificación Iteración 6

3.7.2. Diseño

El diseño de los módulos de verificar los documentos disponibles que no están siendo trabajados por otro usuario y el módulo de consultar los documentos que pertenecen al usuario que está conectado en el sistema actualmente, son iguales. Este se basa en la presentación de una imagen a escala de los documentos en donde se le presentan dos opciones al usuario: descargar el documento, o pasarlo al siguiente estado según lo define la colección.

En la figura 3.30, se puede observar lo descrito anteriormente.

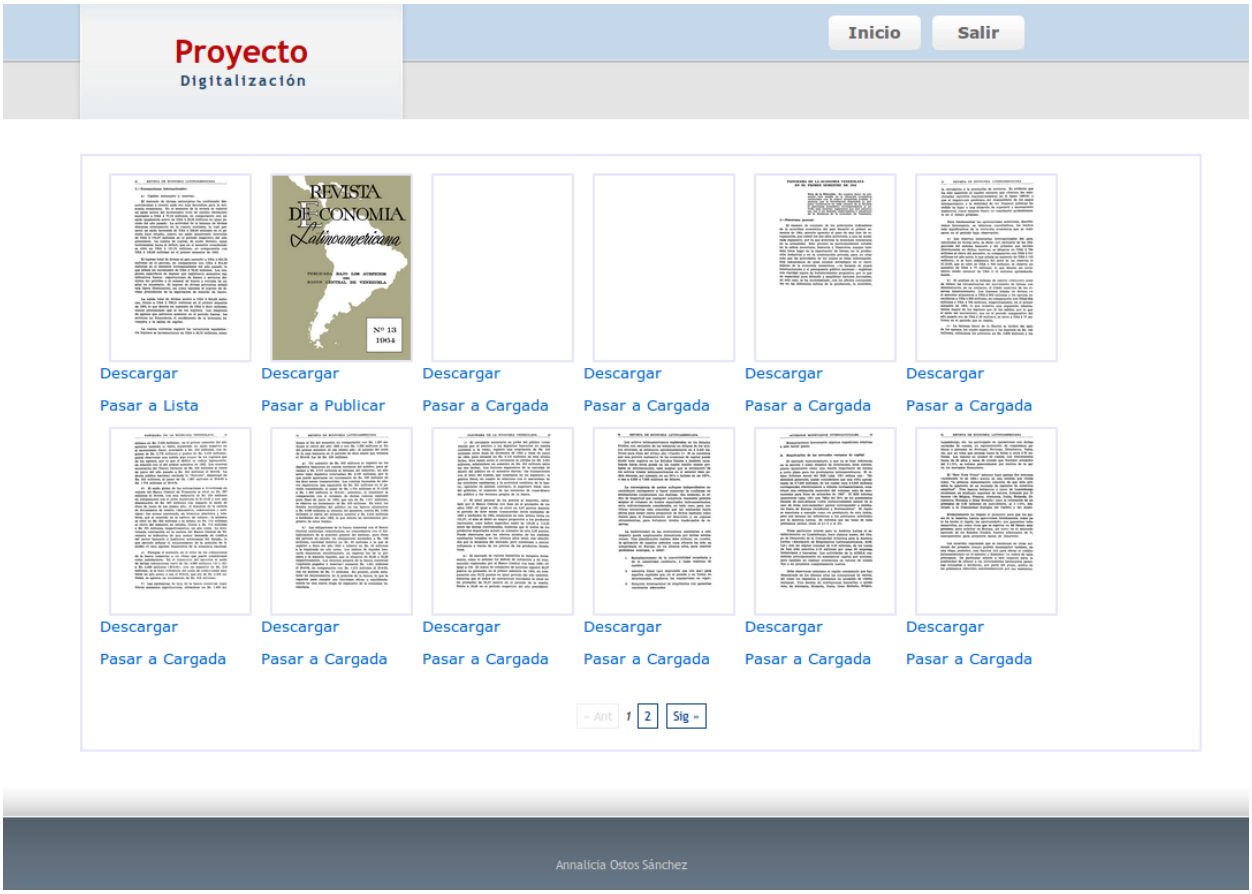

Figura 3.30 – Vista para verificar Documentos disponibles.

Al momento de descargar un documento en el módulo de verificación, el documento pasa a pertenecer al usuario que lo descargo hasta el momento en que este le cambia el estado. Por tal motivo al descargar el documento este deja de visualizarse en el módulo de verificación para visualizarse en el módulo de los documentos del usuario.

Para poder conocer cuáles son los documentos que están siendo trabajados en un momento determinado por un usuario y cuál es el usuario, se agrego en la tabla

Documentos una relación con la tabla Usuarios. Tal cual como se puede observar en la figura 3.31.

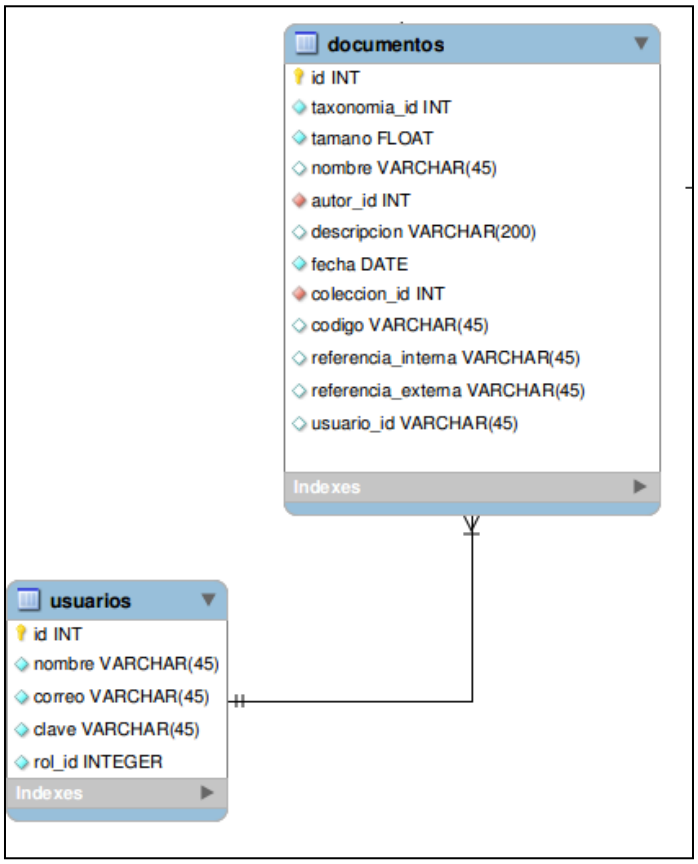

Figura 3.31 – Relación entre la tabla Usuarios y la tabla Documentos.

En ambos módulos se trabajo con el plugin de Jquery, YoxView, para la visualización de los documentos, con este plugin los documentos se ven en primer plano y queda desactivada las demás opciones de la aplicación y permite que se puedan recorrer los documentos que se encuentran en cada módulo. En la figura 3.32, se tiene como se pueden visualizar los documentos gracias a este plugin en los módulos de verificar documentos y en los documentos del usuario conectado.

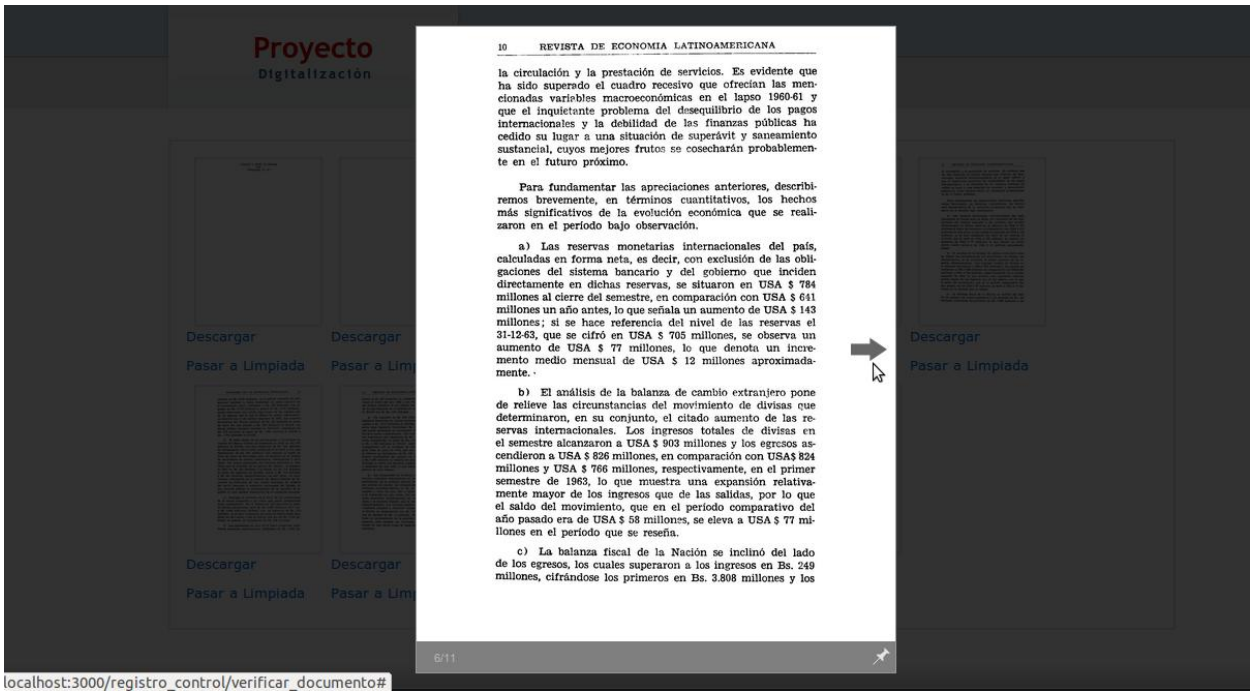

Figura 3.32 – Visualización de los Documentos.

3.7.3. Codificación

En la figura 3.33, se muestra el código correspondiente para obtener los documentos que pueden ser editados o trabajados por el rol del usuario actual y el cual no está siendo trabajado por otro usuario. Como se puede observar se obtienen los valores de las variables de sesión donde se tiene almacenado el rol y el id del usuario para luego proceder con las consultas.

```
class RegistroControlController < ApplicationController

  def verificar_documento

  user = session[:rol]

  @user_id = session[:id]

  estados = EstadoRol.where(["rol_id=?", user]).collect {|p| p.estado_id }

  documentos = Documento.where(["usuario_id IS NULL"]).collect {|p| p.id }

  @disponibles = Posee.where(["estado_actual=1 AND estado_id IN (?) AND documento_id IN (?)", estados, documentos])

  @colecciones = Coleccion.all

  end
```

Figura 3.33 –Código para verificar documentos disponibles.

En primer lugar se obtienen los estados que pueden ser trabajados por el rol del usuario, luego los documentos que no están siendo trabajados por ningún otro usuario y finalmente se realiza una búsqueda sobre la tabla “Posee” para obtener todos los documentos disponibles que cumplan con las 2 condiciones anteriores.

En la figura 3.34, se muestra otro fragmento de código el cual se utilizo para la obtención de los documentos que están siendo trabajados por un usuario. Este proceso es mucho más sencillo ya que solo se tiene que buscar los documentos en donde el atributo usuario_id sea igual al id del usuario que se guardo en la variable de sesión.

```
def mis_documentos

  user = session[:id]

  @documentos = Documento.where(["usuario_id=?", user])

  end
```

Figura 3.34 – Código para verificar los documentos de un usuario.

En la figura 3.35, se muestra el código necesario para poder realizar la descarga de los documentos. Este método se encuentra en el application_controller para poder ser invocado desde los diferentes controladores. El atributo *disposition* que en este caso

tiene el valor *attachment*, es lo que indica que el documento se descarga y no se visualizará en el navegador.

```
def download

  documento = Documento.find_by_id params[:id]

  send_file "public/resources/#{documento.referencia_interna}/#{documento.documento_file_name}", :disposition=> 'attachment'

end
```

Figura 3.35 – Código para la descarga de los documentos.

Para desarrollar el método que permitiera realizar el cambio de estado de un documento se realizo un *Helper* en el cual se obtendría cual es el estado actual del documento y de esta forma es que se puede mostrar en e diseño el nombre del siguiente estado que poseerá el documento.

En la figura 3.36, se muestra el código de dicho *Helper*. Este recibe el id del estado actual y el id de la colección a la que pertenece el documento, para de esta forma proceder a buscar en la tabla Estado en Colección cual es el nivel del estado actual del documento y de esta forma realizar otra búsqueda sobre la misma tabla pero para buscar el estado del siguiente nivel de esta colección. El llamado a este *Helper* se realiza por cada documento que se encuentre en el módulo verificar documentos y en el módulo mis documentos.

```
def estado_siguiente(estado_id, coleccion_id)

  nivel = EstadoEnColeccion.where("coleccion_id = ? and estado_id = ?", coleccion_id, estado_id).collect{|p| p.nivel + 1}

  estado_siguiente = EstadoEnColeccion.find(:first, :conditions => ["coleccion_id = ? and nivel = ?", coleccion_id, nivel])

  return estado_siguiente.estado.valor if estado_siguiente
end
```

Figura 3.36 – Configuración Helper para mostrar el siguiente estado de un Documento.

En la figura 3.37, se tiene un fragmento del código perteneciente al método que se realizo para guardar el nuevo estado del documento. En este método se aplica la misma lógica que se aplica en el *Helper* para obtener el estado siguiente del documento y luego de esto se procede a guardar el nuevo estado que tiene ese documento en la tabla "Posee".

```
def actualizar_estado

   nivel = EstadoEnColeccion.where("coleccion_id = ? and estado_id = ?", params[:documento][:coleccion_id], params[:documento][:estado_id]).collect{|p|
p.nivel + 1}

   estado_siguiente = EstadoEnColeccion.find(:first, :conditions => ["coleccion_id = ? and nivel = ?", params[:documento][:coleccion_id], nivel])

   Posee.agregar(params[:documento][:documento_id], params[:documento][:estado_id],estado_siguiente.estado_id)

end
```

Figura 3.37 – Código para actualizar Documento.

3.7.4. Pruebas

Las pruebas consistieron en la verificación del funcionamiento correcto de cada uno de los módulos desarrollados. Entre estas se tiene la obtención de los documentos correctos en cada módulo, la obtención correcta de los siguientes estados de cada documento y el almacenamiento de los mismos.

En el módulo para verificar los documentos que no están siendo trabajados por ningún usuario, se realizó inicialmente la prueba para comprobar que todos los documentos que se mostraran, no estaban siendo trabajados por ningún usuario en particular, además de comprobar que se visualizarán todos los documentos al hacerles click sobre la imagen de los mismos. Al realizar esta prueba se comprobó que la visualización solo es posible, actualmente, para imágenes y si se incorpora un documento que no sea una imagen a la aplicación no se podrá visualizar directamente en la aplicación.

Lo mismo ocurre en el módulo de "mis documentos". En este módulo se comprobó que al momento de ejecutar la descarga de un documento en el módulo de "verificar documentos", este se asigna a un usuario y deja de presentarse en dicho módulo y se presenta en el módulo de "mis documentos".

3.8. Iteración 7

3.8.1. Planificación

Iteración 7			
Descripción	Desarrollar los módulos para descargar documentos y colecciones. Además de permitir la eliminación y modificación de los estados por los que ha pasado un documento.		
Fecha Inicio / Fecha Fin	07 – 04 – 2011 / 28 – 04 – 2011		
Número	**Fecha**	**Historia**	**Tipo**
32	07 – 04 – 2011	Desarrollar un método para descargar los documentos y colecciones en formato .zip	Nueva
33	07 – 04 – 2011	Desarrollar un método que le permita al superadministrador eliminar el estado en el que estaba un documento o modificarlo.	Nueva
34	07 – 04 – 2011	Validar que los nombre de las colecciones y documentos solo posean letras y números y sean únicos.	Modificación/ Mejora

Tabla 3. 8 - Planificación Iteración 7

3.8.2. Diseño

En esta iteración se desarrollo un módulo, el cual permite al usuario descargar en un archivo .zip todos los documentos pertenecientes a una colección sin que la

misma sea asociada al usuario que las descargue, como ocurre en el caso de verificar documentos. Asimismo también se permite que se descargue un solo documento.

Esto permitirá al usuario poder descargar documentos o colecciones completas para diferentes tareas, como realizar una copia de la misma, consultarla, entre otras.

En las figuras 3.38 y 3.39, se muestra el diseño de dicho módulo. Este es presentado al usuario mediante pestañas, en donde en la primera pestaña se muestran todos los documentos que se encuentran en la aplicación y en la segunda pestaña se muestran todas las colecciones. En ambos casos, debajo de cada objeto, se presenta un enlace que permite descargar los mismos. Como se menciono anteriormente, en el caso de que se proceda con la descarga de una colección esta se realizará en un archivo comprimido[4].

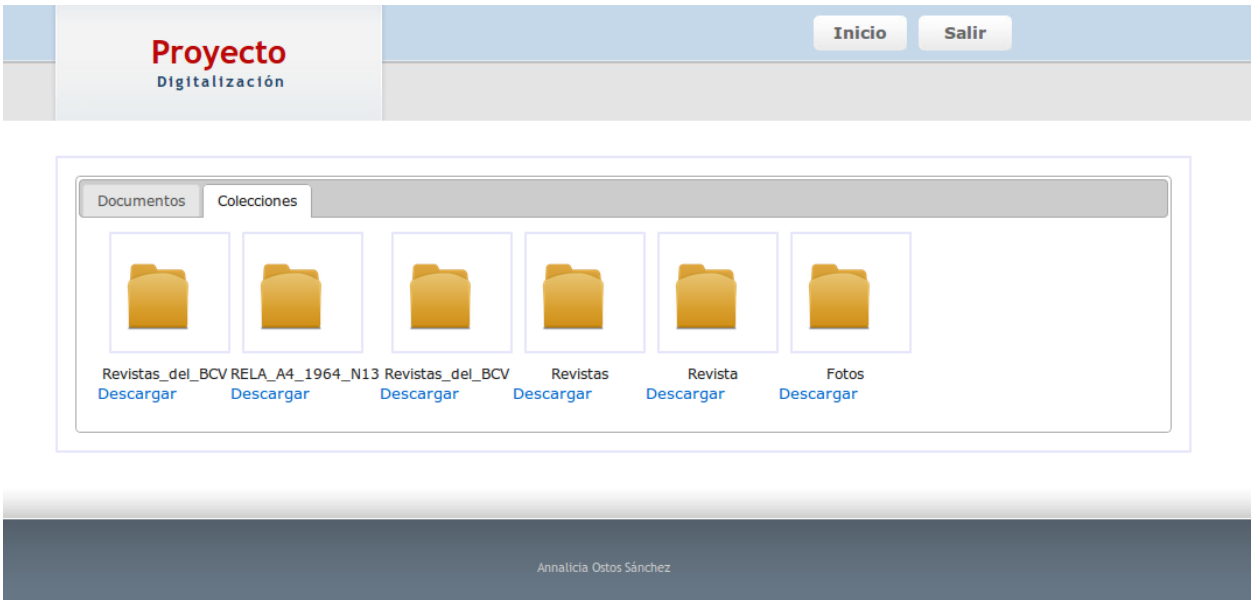

Figura 3.38 – Vista para la descarga de Colecciones

[4] Actualmente, solo se puede comprimir en el formato .zip.

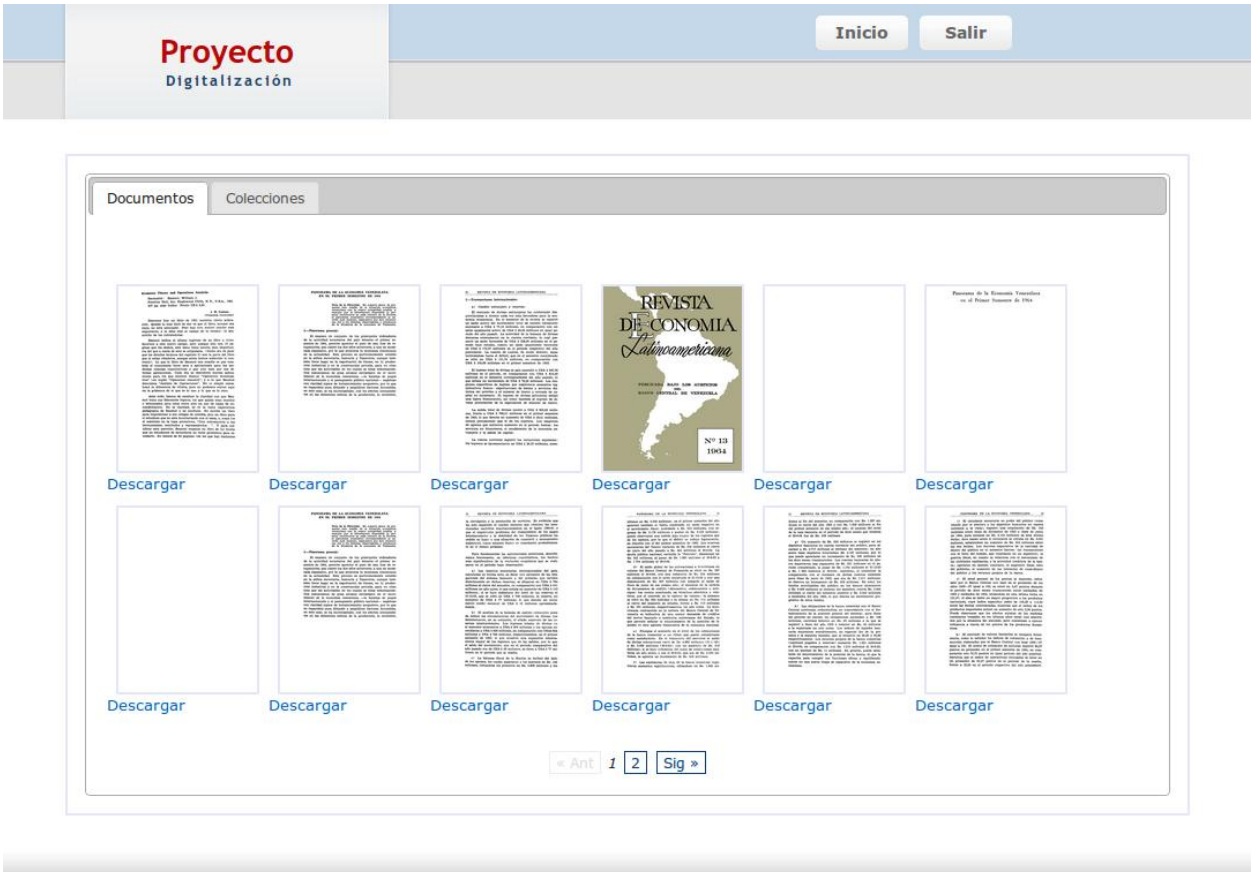

Figura 3.39 – Vista para la Descarga de Documentos

Finalmente, en la figura 3.40, se muestra el diseño que tiene el módulo para la edición o eliminación de los estados por los que ha pasado un documento. Estos estados se presentan en una tabla, en la cual se utiliza paginación. Este módulo sólo lo visualizarán los usuarios con el rol superadministrador

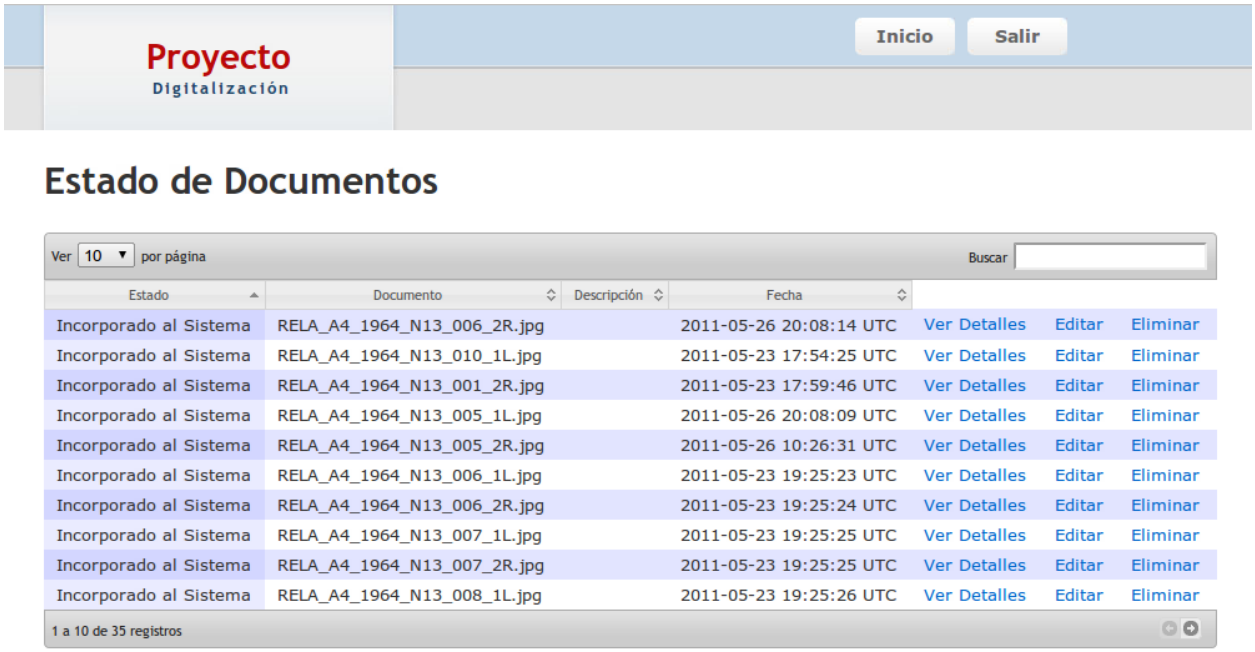

Figura 3.40 –Lista de Estados de los Documentos

3.8.3. Codificación

En la figura 3.41, se muestra parte del código necesario para la creación y descarga de las colecciones en un archivo .zip. Para crear dicho archivo primero se crea la dirección en donde se encuentra dicha colección en el sistema y luego a través del método bundle se genera el archivo .zip con el contenido de toda la colección. Y posteriormente se procede a descargar dicho archivo a través del método send_file.

```
def crear_zip

    c = Coleccion.find_by_id(params[:id])
    coleccion = c
    if c.parent_id.nil?

        referencia = "#{c.nombre}"

      else

       referencia = "#{c.nombre}"

        while !c.parent_id.nil? do
         c = Coleccion.find_by_id(c.parent_id)
         referencia = "#{c.nombre}/#{referencia}"
        end
    end

    coleccion.bundle(referencia)
      send_file "public/resources/#{referencia}.zip", :type => 'application/zip', :disposition => 'attachment'
  end
```

Figura 3.41 – Código para la creación del .zip

En la figura 3.42, se muestran los métodos realizados para actualizar y eliminar los estados que posee un documento. En ambos casos, los métodos reciben el id de la relación y se realiza la búsqueda de ese registro en la tabla “Posee” y luego de tenerlo se procede a actualizar o eliminar según sea el caso.

```
  def update
    @posee = Posee.find(params[:id])

    respond_to do |format|
      if @posee.update_attributes(params[:posee])
        format.html { redirect_to(@posee, :notice => 'Posee was successfully updated.') }
        format.xml  { head :ok }
      else
        format.html { render :action => "edit" }
        format.xml  { render :xml => @posee.errors, :status => :unprocessable_entity }
      end
    end
  end

  # DELETE /posee/1
  # DELETE /posee/1.xml
  def destroy
    @posee = Posee.find(params[:id])
    @posee.destroy

    respond_to do |format|
      format.html { redirect_to(posee_index_url) }
      format.xml  { head :ok }
    end
  end
```

Figura 3.42 – Código para la edición y eliminación de estados de documentos

3.8.4. Pruebas

Las pruebas de esta iteración se basaron en la descarga de diferentes colecciones, para comprobar el funcionamiento de la creación del archivo .zip y la descarga de los documentos individualmente.

En estas pruebas se obtuvo como resultado que al momento de descargar una colección se crea el archivo comprimido, actualmente con formato .zip, el cual contiene todos los documentos pertenecientes a la colección, además de las colecciones que pertenecen a dicha colección de ser el caso.

En el caso de la descarga de Documentos, esta se realiza en el archivo original del archivo.

En ambas pruebas no se obtuvieron resultados negativos, ya que la descarga de tanto de Colecciones como de Documentos se realizaba correctamente.

CONCLUSIONES

La investigación sobre las tecnologías actuales relacionadas con la digitalización de documentos, permitió, el desarrollo de este Trabajo Especial de Grado, el cual se basó en el desarrollo de una aplicación Web que permita apoyar los procesos de digitalización de Documentos, clasificación y visualización de Documentos Digitales basados en una taxonomía.

El marco conceptual ayudó al logro de los objetivos propuestos, ya que gracias a la investigación que se realizó antes de comenzar el desarrollo de la aplicación se obtuvieron las bases sobre los conocimientos que necesitaba para conocer lo relacionado con los proyectos de digitalización, las tecnologías que se iban a utilizar y el método ágil de programación extrema (XP).

La adaptación del método ágil "Programación Extrema" (XP) facilitó trabajar de forma organizada al permitirme agrupar los requerimientos en un conjunto de iteraciones que se fueron desarrollando progresivamente, la comunicación constante con el cliente fue otro aspecto que me brindó resultados positivos, ya que permitió realizar constante revisiones a los avances del sistema. Con todo esto se obtuvo la construcción de una aplicación acoplada a las necesidades del usuario, además de dejarme conocimientos y experiencia para seguir asumiendo proyectos innovadores.

Se debe mencionar que una de las características en las que se basa la programación Extrema que no se llevó a cabo en dicho trabajo, es la programación en pareja.

El uso de una modelación práctica en el diseño permitió una clara comprensión de requerimientos para la implementación. A la vez, se aplicaron estándares de programación y se implementó el patrón MVC, lo que ayudó a la obtención de un código legible y ordenado, que facilitará a futuros programadores continuar con el mantenimiento y evolución de la aplicación.

En el desarrollo de la aplicación se lograron la mayoría de los objetivos específicos planteados en un inicio. Entre los que se encuentra el desarrollo de los módulos para el procesamiento y visualización de los documentos, la integración de todos los módulos de la aplicación, el diseño de las interfaces de usuario de cada uno de los módulos de la aplicación, el diseño de la base de datos donde se almacena toda la información relevante de los diferentes proyectos de digitalización.

Es importante destacar que la aplicación Web desarrollada es un prototipo para la gestión y clasificación de colecciones de documentos digitales. En la cual se busco a lo largo de su desarrollo de obtener una aplicación lo más genérica posible y que a la vez no estuviera atada a ninguna red interna sobre la cual funcionar. Aunque la aplicación acepta cualquier tipo de documento las pruebas realizadas solo fueron con imágenes de diferentes formatos y videos.

Debido a que la aplicación desarrollada es un prototipo, no se pudo lograr el objetivo planteado en un inicio, de incorporar dicha aplicación a la comunidad de software libre.

Para el diseño y construcción de esta aplicación se tomaron en cuenta aspectos de usabilidad, eficiencia, robustez y escalabilidad. En el desarrollo se utilizo el lenguaje de Programación Ruby sobre el *framework* Rails (en su versión 3.0.3), el manejador de Base de datos MySQL, la librería de javascript Jquery, y un conjunto de librerías (Gems) adicionales tales como *Paperclip*, *SimpleCaptcha*, *RubyZip*, con las cuales se logro la carga de archivo, la mejora de la autenticación y la creación de archivos .zip para la descarga de los documentos.

En el transcurso del desarrollo de la aplicación me pude encontrar con algunos inconvenientes, los cuales surgieron por el poco conocimiento sobre los cambios realizados en el framework de desarrollo Rails, aunque ya había trabajo anteriormente con Rails lo había hecho en su versión anterior la 2.3.8 y no en la que realice la aplicación que es la versión 3.0.3. Pero a medida que fueron pasando las semanas pude mejorar y superar estos inconvenientes.

Para finalizar, el desarrollo de esta aplicación Web es el inicio para próximas mejoras de las diferentes entidades de la misma, entre las cuales se puede destacar la incorporación del proceso de OCR, la búsqueda de contenido de los documentos, la manipulación de procesos en segundo plano a través de la aplicación, entre muchas otras funcionalidades.

BIBLIOGRAFÍA

Bangor University. (Abril de 2008). *SALT cymru.* Recuperado el 21 de Agosto de 2010, de http://www.saltcymru.org/english/saltcymru_document5.pdf

Breuel, T. (Abril de 2007). *Google Code Blog .* Recuperado el 21 de Agosto de 2010, de http://googlecode.blogspot.com/2007/04/announcing-ocropus-open-source-ocr.html

Burbeck, S. (1992). *Model-View-Controller Architecture*. Obtenido de http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

Captcha, G. H.-S. (2011). Obtenido de http://github.com/galetahub/simple-captcha

IBM. (2007). *Guía del Estudiante Base de Datos I.*

ImageMagick. (2010). Recuperado el 28 de Agosto de 2010, de http://www.imagemagick.org/script/index.php

Jeffries, L. R. (2011). *Xtreme Programming an Agile Software Development Resource*. Recuperado el 15 de Abril de 2011, de http://xprogramming.com/

Manual de Digitalización de Documentos. (2004). Obtenido de http://www.imaginar.org/dgd/manuales/manual_digitalizacion.pdf

Márquez, M. &. (2008). *Tópicos para el desarrollo de un módulo de generación de reportes del sistema Conest.* Universidad Central de Venezuela.

Ministerio del Transporte y Comunicaciones. (2006). Obtenido de http://www.mtc.gob.pe/pdd/PDF/ANEXOS/07%20AREA%20DE%20DIGITALIZACION.pdf

Pressman, R. (2007). *Ingeniería del Software. Unenfoque práctico. Sexta Edición.* Mc Graw Hill.

RMagick . (2008). Recuperado el 28 de Agosto de 2010, de http://rmagick.rubyforge.org/

RubyDoc.info. (2011). Obtenido de http://rdoc.info/github/thoughtbot/paperclip/master/file/README.md

Unesco. (2002). *Directrices para Proyectos de Digitalización.* Secretaría General Técnica .

APÉNDICE 1

INSTALACIÓN RUBY 1.9.2 Y RAILS 3.0.3

En el siguiente apéndice se mostraran pasos para la instalación de Ruby 1.9.2 y Rails 3.0.3 sobre Ubuntu 10.10 con 64 bits.

- **Paso 1**

Instalar los siguientes paquetes antes de proceder a la instalación de Ruby, es posible que ya se tengan dichos paquetes instalados en el sistema, de ser así no se ejecutan las siguientes instrucciones.

```
$ sudo apt-get install gcc g++ build-essential libssl-dev libreadline5-dev zlib1g-dev linux-
headers-generic libsqlite3-dev
```

- **Paso 2**

Ahora se procede a descargar las fuentes de Ruby 1.9.2, desempaquetarlas e instalaras:

```
$ wget ftp://ftp.Ruby-lang.org//pub/Ruby/1.9/Ruby-1.9.2-p0.tar.gz
$ tar -xvzf Ruby-1.9.2-p0.tar.gz
$ cd Ruby-1.9.2-p0/ $ ./configure --prefix=/usr/local/Ruby
$ make && sudo make install
```

- **Paso 3**

Agregar ruta de acceso a archivos binarios de Ruby.

```
$ sudo gedit /etc/environment
```

Es necesario añadir en la variable PATH esto /usr/local/Ruby/bin, luego debería obtenerse:

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/Ruby/bin"

- **Paso 4**

A continuación, se ejecuta el comando “source” sobre el archivo /etc/environment para aplicar los cambios

```
$ source /etc/environment
```

Ahora se comprueba si Ruby está instalado correctamente

```
$ Ruby -v
```

Se debe obtener: 1.9.2p0 rubí (08/18/2010 revisión 29.036) [x86_64-linux]

- **Paso 5**

Ahora se procede a instalar las Gemas necesarias, incluyendo Rails 3

```
$ sudo gem install tzinfo builder memcache-client rack rack-test erubis mail text-format
bundler thor i18n
$ sudo gem install rack-mount
$ sudo gem install Rails --version 3.0.3
```

Se compruebe la versión de Rails:

```
$ Rails -v
```