UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Uso de tecnologías de la información aplicadas al seguimiento, desarrollo y crecimiento profesional en diversas áreas de conocimiento

TESIS

Que para obtener el título de

Ingeniera en Computación

P R E S E N T A

Abigail Ortega Valdivia

DIRECTOR DE TESIS

Ing. Jorge Alberto Rodríguez Campos

Ciudad Universitaria, Cd. Mx. 2018

DEDICATORIAS.

A DIOS

Por permitirme realizar uno de mis más grandes sueños y

por la paz y salud que me ha regalado todo este tiempo.

A MI MADRE

Por el apoyo incondicional que siempre me ha brindado,

la compresión y su invaluable compañía.

AGRADECIMIENTOS.

Al Ing. Jorge Alberto Rodríguez Campos, por el apoyo que siempre me mostro, el tiempo, los consejos y la pasión por el desarrollo de software que me contagió en cada enseñanza.

A la UNAM, por regalarme la maravillosa experiencia de ser universitaria y por ser mi segunda casa desde los 15 años.

A mis profesores, por compartirme un pedazo de su ser, por las enseñanzas y el aprendizaje, tesoros invaluables.

A Sebastian Cuatepotzo, por el apoyo que siempre me ha brindado, la compañía, honestidad y amor.

A mis amigos y familiares, gracias por el apoyo.

ÍNDICE

1. INTRODUCCIÓN.

A lo largo del presente trabajo se muestra la creación de una herramienta como solución ante la problemática que viven los profesionales del área de Tecnologías de la Información (TI) al no saber hacia dónde enfocar el crecimiento profesional. Dicha herramienta está orientada a la automatización de los procesos de negocio de las empresas enfocadas en ofrecer servicios relacionados al crecimiento profesional en el área de TI.

A través de los capítulos que componen este trabajo de tesis se muestra la solución para la creación de la herramienta propuesta, desde la etapa de análisis hasta el ciclo de pruebas efectuadas.

En el capítulo 3 se presenta una amplia descripción de las principales metodologías existentes para el desarrollo de software, con especial énfasis en metodologías ágiles, también se muestra una comparación entre ellas con base en sus características más importantes y representativas. Finalmente se justifica la elección de la metodología elegida tomando en cuenta aspectos de interés.

En el capítulo 4 se muestran los requerimientos para el desarrollo de la herramienta, funcionales y no funcionales, también se presentan algunos artefactos útiles para el análisis de los mismos, como lo son los diagramas de lenguaje de modelado unificado (UML por sus siglas en inglés) y las historias de usuario.

Este capítulo es de gran interés puesto que muestra la justificación de la elección de la tecnología a utilizar con base en los requerimientos previamente establecidos por el usuario.

El capítulo 5 contiene una amplia explicación de los más populares tipos de diseño y arquitectura para el desarrollo de software, además de la justificación de la elección de los mismos.

El capítulo 6 muestra el desarrollo de la herramienta propuesta. El capítulo se encuentra dividido por iteraciones, por cada iteración se presentan una serie de diagramas, imágenes ilustrativas y fragmentos de código de interés. Se presenta la implementación de lo visto en los capítulos anteriores, desde el empleo de las metodologías ágiles hasta la utilización de la arquitectura y diseño elegidos. Por tratarse de un sistema muy amplio el desarrollo se ha limitado a mostrar solo un conjunto de iteraciones con casos representativos.

El capítulo 7 representa la etapa de pruebas, en este capítulo se ilustran, diseñan y ejecutan algunos casos de prueba con el objetivo de asegurar que el funcionamiento de la herramienta corresponda al esperado.

Finalmente, el capítulo 8 corresponde a las conclusiones mediante las cuales se analizará el resultado obtenido y se verificará el cumplimiento del objetivo primordial.

2. MARCO TEÓRICO.

2.1. ANTECEDENTES.

Hoy en día, vivimos en un mundo cambiante, un mundo que cada segundo evoluciona, se actualiza y mejora, por ende, para los profesionistas es de vital importancia mantenerse en sincronía con éste, evolucionar junto con él. Cuando hablamos de profesionistas del área de TI debemos poner especial atención, pues el mundo tecnológico tiene un crecimiento exponencial. Vivimos una revolución tecnológica, en el mercado existen gran cantidad de soluciones, alternativas y opciones para las diferentes áreas, lo que provoca incertidumbre para decidir hacía donde enfocar el crecimiento profesional.

El desarrollo del crecimiento profesional no solo tiene como objetivo el ser competitivo en el mundo actual, pues va de la mano con algunos otros conceptos de gran interés, como lo son la autorrealización, satisfacción y superación.

En la actualidad se cuenta con variadas maneras de mitigar esta problemática, pues existen empresas enfocadas en ayudar a los profesionales de diversas áreas en este proceso, con la creación de material escrito, cursos presenciales y en línea, etc., pero en algunos casos existen deficiencias en sus procesos, ya que no cuentan con una herramienta óptima para evaluar el conocimiento, dar una retroalimentación eficaz y puntual, o dotar de un seguimiento adecuado.

Esta propuesta presenta una solución a través de la creación de una herramienta integral que esté a la altura de sistemas empresariales modernos, permitiendo automatizar los procesos de negocio de las empresas encargadas de ofrecer servicios orientadas a contribuir con el crecimiento, seguimiento y especialización profesional en el área de TI.

Cuando se habla de sistema de software empresarial surgen diversos aspectos que llegan a la mente. Primeramente, las metodologías de desarrollo de software. En la actualidad se tiene una amplia gama de metodologías a utilizar, desde metodologías tradicionales hasta metodologías ágiles. Su adopción depende de las exigencias de cada proyecto a realizar.

Existen ya, algunas investigaciones realizadas sobre metodologías las cuales han aportado información que compete en este trabajo por lo cual se mencionan en la Tabla 2.1.

INVESTIGACIÓN	COMENTARIOS
Bautista Zaragoza, M.G. & Garibay Tierra dentro M. (2004). Metodologías de desarrollo como herramienta para asegurar la calidad en los proyectos informáticos (tesis de licenciatura). Universidad Nacional Autónoma de México. Ciudad de México.	Este trabajo de investigación busca establecer las bases para clasificar la metodología ideal a utilizar en un proyecto en específico de software. Tanto metodologías ágiles, estructuradas, orientadas a objetos o de prototipo.
Figueroa, R.G., Solis, C.J & Cabrera, A.A. (2008). Metodologías tradicionales vs. metodologías ágiles. abril 2, 2017, de Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación Sitio web: http://www.academia.edu/download/41231515/articulo-metodologia-de-sw-formato.doc	Este artículo contiene una investigación y comparación de algunas metodologías tradicionales de desarrollo de software, tales como "Rational Unified Process (RUP)", "Microsoft Solution Framework (MSF)", y las metodologías ágiles, por ejemplo, el "Extreme Programming (XP)", "Agil Unified Process (AUP)", "Scrum" e "Iconix".
Carvajal Riola, J.C.. (2008). Metodologías ágiles: Herramientas y modelo de desarrollo para aplicaciones JAVA EE como metodología empresarial (Tesis de maestría). Universitat Politécnica de Catalunya BarcelonaTech. Barcelona.	En este trabajo de investigación, se hizo un análisis profundo de las metodologías tradicionales y ágiles de desarrollo de software, así como las ventajas del uso de Java EE y la aplicación de dos metodologías ágiles ("Scrum" y "XP") en un pequeño caso de estudio.
Orjuela Duarte, A. & Rojas C., M. (mayo 24, 2008). Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo. Avances en Sistemas e Informática, 5 No.2, pp. 159-171.	Este artículo explica las metodologías ágiles de desarrollo de software, como "XP", "Crystal" y "Scrum", con el objetivo de adaptarlas a la ingeniería del software educativo.
Arauz Ortiz, G. (2013). Integración de métodos ágiles a procesos de desarrollo de software en una empresa de nivel 5 CMMI-DEV: Caso de estudio (tesis de maestría). Universidad Nacional Autónoma de México. Ciudad de México.	Esta investigación detalla un caso de estudio del desarrollo de software en una empresa, además de explicar metodologías ágiles como "Scrum", "XP" y "Kanban".

Tabla 2. 1. Investigaciones relevantes sobre metodologías para el desarrollo de software.

Otro punto a considerar al momento de pensar en el desarrollo de software empresarial es la tecnología a utilizar. En las siguientes secciones se presenta una breve descripción y

análisis de las tecnologías de desarrollo de software empresarial que permitirá realizar una correcta toma de decisiones con base a los requerimientos de esta solución, en especial, requerimientos funcionales.

2.2. PLATAFORMA JAVA.

Según Stefan Jablonski una plataforma se puede definir como un conjunto de tecnologías o módulos de software sobre los cuales otras tecnologías y programas se ejecutan.[1]

Usualmente las plataformas actuales están integradas por hardware y software, sin embargo, la plataforma Java rompe este esquema, pues solo se compone de software el cual puede ser ejecutado (sin sufrir cambios) en diferentes arquitecturas y dispositivos computacionales. La plataforma Java se compone de los siguientes elementos:

- Máquina Virtual de Java (JVM por sus siglas en inglés):

Es considerada la base la plataforma, pues es la encargada de garantizar la portabilidad de las aplicaciones Java y ejecutar las mismas.

- API de Java:

Corresponde a una colección de componentes de software prefabricados los cuales proporcionan diversas funcionalidades.

- El lenguaje Java:

Caracterizado por ser un lenguaje sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, independiente de las arquitecturas, portable, eficaz, multihilo y dinámico.

Es de gran importancia entender que Java no corresponde solo a un lenguaje de programación sino a una plataforma, ya que proporciona un medio de ejecución y una API.

2.2.1. BREVE HISTORIA.

El nacimiento de la plataforma Java data del año 1991, cuando la compañía Sun Microsystems tenía a su cargo uno de los proyectos más importantes. Se trataba de realizar el control de dispositivos hogareños, por lo que el equipo de desarrollo, en el cual se encontraba James Gosling (ahora considerado padre de Java), decidió crear un lenguaje de programación al que llamarón Oak. Sin embargo, el proyecto fracasó, y no fue hasta

[1] Stefan Jablonski & Ilia Petrov & Christian Meiler & Udo Mayer. (2004). Guide to Web Application and Platforms Arquitectures. Alemania: Springer.

1995 que se decidió cambiar el nombre de dicho lenguaje y lanzarlo al mundo, con el nombre de Java.

Java presentaba muchas ventajas sobre los lenguajes de programación de su época, pero al mismo tiempo semejanzas que lo volvieron poderoso y popular entre la comunidad informática. Java nace con grandes similitudes a C++, pues los creadores consideraron importante y necesario, mostrar un ambiente conocido. “Escribe una vez, ejecuta en cualquier parte” fue el slogan con el que Java fue lanzado.

Una de sus características más populares consistía en escribir programas que pudiesen ser ejecutados en cualquier sistema operativo, sin importar las características de hardware, esto fue posible gracias a la implementación de la JVM.

La capacidad que este lenguaje posee para realizar el manejo de memoria fue un boom, pues anteriormente el programador era el encargado de administrar la memoria y optimizar ésta, sin embargo, ahora con Java el recolector de basura se encarga de liberar la memoria que no es necesaria para la aplicación.

2.3. JAVA EE.

2.3.1. SISTEMA EMPRESARIAL.

Hoy día, en un mundo totalmente cambiante e impredecible, para las empresas es de vital importancia contar con un sistema que permita mantener los servicios que ofrecen al alcance de sus clientes, estar presente en cada momento en el que estos son requeridos es la clave del éxito. La respuesta ante la problemática anterior es la que la plataforma Java nos brinda mediante su versión empresarial.

Según Bernhard Hitpass un sistema empresarial se define como una representación de la lógica para los procesos de negocios mediante la infraestructura de TI, especificando los requerimientos de integración y estandarización de los procesos.[2]

Entre las características de un sistema empresarial se encuentran las siguientes:

- Busca la optimización y mejora continua de los procesos de negocio.
- Provee herramientas para la toma de decisiones sobre procesos, tecnología y estructura organizacional.
- Permite identificar áreas de interés, oportunidad y preocupación de la empresa.
- Asegura calidad, eficiencia y fiabilidad.

[2] Bernhard Hitpass. (2017). BPM Bussines Process Manag. Santiago de Chile: BPM Center.

El objetivo principal es lograr una producción de recursos mucho más efectiva a partir del manejo adecuado de la información, del cruce correcto de la misma y de su uso rápido, eficaz y seguro.

2.3.2. DEFINICIÓN.

Java EE es una especificación para el desarrollo de aplicaciones Java portables, robustas, escalables y seguras del lado del servidor, proporciona una arquitectura multicapa y cuenta con APIs que facilitan el desarrollo.

Entre las principales características destacan las siguientes:

- Reducción del tiempo de desarrollo.
- Reducción de la complejidad.
- Mejora en el rendimiento.
- Modelo de programación simplificado.

Las características mencionadas anteriormente han permitido que Java encabece las listas de popularidad hoy día.

2.3.2.1 MODELO MULTICAPA.

Java EE implementa un modelo multicapa, el cual consiste en dividir complicados sistemas en capas, esto se realiza con el objetivo de poder simplificar las tareas, generar sistemas mantenibles, extensibles, flexibles y escalables. Un modelo multicapa presenta muchas ventajas desde el desarrollo hasta el mantenimiento del mismo. Entre las principales ventajas que presenta este tipo de arquitectura destacan las siguientes:

- Permite la “encapsulación” de cada capa, evitando que lo que se realice en ésta afecte a las otras.
- Se puede tener un amplio conocimiento de cierta capa del sistema sin tener conocimiento alguno del resto del sistema.
- Permite la estandarización del modo en que se realiza el diseño.
- Los servicios de cada capa permiten la interacción con las capas adyacentes.

Existen muchas formas de dividir un sistema mediante capas, pero Java EE usualmente lo hace mediante tres capas: la primera corresponde a la capa de presentación, la segunda lleva el nombre de capa de negocio y finalmente la tercera es la capa de datos. A

continuación, se presenta una descripción breve de cada una de estas capas haciendo especial énfasis en los componentes básicos que cada una define.

- Capa de Presentación: Es la vista al usuario final, tiene la característica de ser amigable para éste, lleva los objetos de la lógica de negocio al alcance del usuario.

  Java EE ofrece para esta capa las siguientes tecnologías:

  - JavaServer Faces (JSF): Es un framework Modelo Vista Controlador (MVC), el cual proporciona un conjunto de componentes en forma de etiquetas, definidas en páginas XHTML. Es útil para el desarrollo de interfaces de usuario avanzadas en aplicaciones web.
  - Java Servlets: Es una API de la cual surgen los llamados Servlets, los cuales corresponden a un pequeño código Java que el servidor web utiliza para manejar peticiones del cliente, ampliando las capacidades de éste.
  - JavaScript Object Notation (JSON): Es un formato para el intercambio de datos, utiliza una sintaxis específica para identificar y gestionar datos. Nació como una alternativa a XML. Utiliza la API for JSON Binding.
  - WebSockets: Hace uso de Java API for WebSocket, se refiere a una conexión bidireccional punto a punto entre navegador y servidor, sobre un único socket TCP.
  - JavaServer Pages (JSP): Permite crear fácilmente contenido web que tiene componentes estáticos y dinámicos.

- Capa de Negocio: Es en esta capa donde se llevan a cabo los procesos de la lógica de negocio, esta capa es la encargada de interactuar con la capa de presentación y la de datos.

  Java EE ofrece para esta capa las siguientes tecnologías:

  - Enterprise JavaBeans (EJB): Se trata de un componente de negocio, pues implementan la lógica de éste, son reusables y adaptables.
  - Java Message Service (JMS): Se encarga de la mensajería empresarial, el objetivo es transferir información entre sistemas heterogéneos.
  - JAX-RS: Hace uso de REST para el intercambio de mensajes, utiliza HTTP.
  - Context Dependency Injection (CDI): define el mecanismo para resolver dependencias entre servicios.
- Capa de Acceso a Datos: También llamada capa de persistencia de datos, es la encargada de realizar el acceso a los datos y proporcionar los mismos a las capas

superiores, está integrada por los servidores de bases de datos, sistemas ERP y otras fuentes de datos.

Java EE ofrece para esta capa las siguientes tecnologías entre otras:

- Java Database Connectivity API (JDBC): Se refiere a una interface de acceso a bases de datos estándar SQL, una API para la ejecución de sentencias SQL.
- Java Persistence API (JPA): Es una API de persistencia, permite establecer una correlación entre una base de datos y un sistema orientado a objetos, mediante el uso de archivos de mapeo o anotaciones.
- Java Transaction API (JTA): Es una especificación, se encarga de definir la forma en que los servidores JEE implementan la administración de transacciones tanto locales como distribuidas, abstrayendo casi en su totalidad al programador de dicha tarea.

2.3.3. BREVE HISTORIA.

La primera versión de Java empresarial fue: J2EE lanzada en diciembre de 1999, contenía 10 especificaciones, entre ellas las más representativas son: Servlets, Java Server Pages, Enterprise Java Beans y Java Messages Service. Para el 2001 la versión 1.3 ya se encontraba disponible, esta vez con 13 especificaciones e incrementos muy representativos como la mejora en la búsqueda de nombres mediante JNDI y la ejecución de Java Enterprise Beans. En noviembre del 2003 llega la última versión de J2EE, la versión 1.4 la cual cuenta con la incorporación de API's y 20 especificaciones.

Hasta este punto solo se ha hablado de las bondades que Java ofrecía, pero es importante hacer mención de las áreas de mejoría que existían. Primeramente, las líneas de código XML necesarias para definir algunos recursos como Enterprise Java Beans (EJB) o Servlets eran excesivas, el manejo de dependencias era totalmente un caos, pues cada que un componente dependía de otro, tenía que hacer una búsqueda exhaustiva de todas las dependencias, además, los EJB eran pesados, inflexibles y era muy complejo trabajar con ellos. Ante esta problemática los desarrolladores decidieron buscar una solución, la cual fue posible en versiones posteriores.

En el 2003 la comunidad de programadores Java esperaba ansiosamente la versión 1.5, sin embargo, Sun Microsystems lanzó una versión nueva, esta vez con un nombre diferente, Java EE 5.

Java EE 5 mostraba claramente la simplificación del modelo de persistencia con la introducción de JPA, una convención sobre configuración, inyección de dependencias, la actualización de la API XML, una drástica simplificación del modelo de componentes EJB y el apoyo de anotaciones. El lanzamiento de esta versión ayudó a impulsar el futuro de Java EE mediante la utilización de Plain Old Java Objecs (POJOS que corresponden a clases simples, que no extienden ni implementan a otras), además de la reducción del número de archivos XML, e interfaces.

Java EE 6 se hizo presente en 2009 con características tremendamente atractivas como la gestión y validación de EJB, inyección de dependencias, mejoras en JSP y la Servlet API. Este año fue muy importante para la comunidad Java, pues justo en 2009 Oracle compra Sun Microsystems con lo cual aceptó la responsabilidad de la evolución de la plataforma. Una de las versiones más poderosas tuvo lugar en 2013, ya en manos de Oracle, Java EE 7, es aquí donde los Java Server Faces son liberados, existen nuevas características de desarrollo web como son los WebSockets API, JAX-RS y JSON-P. Cada versión integró nuevas características que se alinean con las necesidades de la industria, permitiendo incrementar y mejorar la productividad del desarrollador y la portabilidad de las aplicaciones.

Finalmente, la versión más reciente es Java EE8, entre las mejoras que ésta presentan se encuentran:

- The Java API for JSON Binding
- Java Message Service 2.1
- Java Servlet 4.0
- Java API for RESTful Web Services 2.1
- Model-View-Controller 1.0
- Java Server Faces 2.3
- Java EE Management API 1.0
- Java API for JSON Processing 1.1
- Java EE Security API 1.0

Finalmente, en 2017 Oracle decidió dejar de controlar el desarrollo de Java EE y otorgar esta responsabilidad a alguna otra compañía open source, esto con el objetivo de mejorar el desarrollo de la especificación. La compañía elegida para pasar la batuta ha sido Eclipse Foundation, sin embargo, la transición lleva consigo el cambio de nombre, por lo que pasa de llamarse Java EE a Jakarta EE. Dado a que es un proceso que aún se encuentra en transición se ha acordado conservar el nombre de Java EE para este trabajo.

2.4. SPRING IO.

2.4.1. SPRING.

Desde sus inicios la especificación Java EE tuvo bastantes detalles que dificultaban el desarrollo de aplicaciones, como ya se mencionó anteriormente, el manejo de EJB era complejo e inflexible, además la cantidad de código era excesiva, los desarrolladores dedicaban gran parte de tiempo y esfuerzo en códigos de utilería dejando de lado lo realmente importante, la lógica de negocio.

En este contexto nace Spring, un framework para el desarrollo de aplicaciones Java, de código abierto. Fue creado por Rod Johnson y presentado en su libro "Expert One-on-One: J2EE Design and Development" con un propósito en específico: ser una alternativa a los componentes EJB, y con ello hacer sencillo el desarrollo de una aplicación Java EE. Su estructura busca cumplir con los objetivos de un sistema empresarial como son la persistencia, productividad, manipulabilidad, etc. Este framework ha tenido excelente aceptación en la industria de TI y es hoy en día muy valorado por aportar simplicidad, fácil entendimiento y gran facilidad de testeo.

Spring implementa cuatro estrategias clave:

- Desarrollo ligero y mínimamente invasivo con POJOs.

Como se explicó anteriormente la implementación de POJOs es una gran ventaja cuando se habla de aplicaciones empresariales, Spring fue el pionero en su utilización. Los POJOs se caracterizan por ser clases simples, que no implementan interfaces o extienden clases particulares a una tecnología, API, o modelo de programación, para ser ejecutados no requieren un servidor de aplicaciones y son independientes. De ahí la idea de ser objetos simples, no contiene código especifico, únicamente lógica de negocio.

- Acoplamiento ligero a través del uso de inyección de dependencias.

Inyección de dependencias puede sonar como un concepto bastante complejo, pero realmente no lo es. Para explicar de forma concreta su significado debemos recordar cómo es que antiguamente cada objeto era responsable de obtener sus propias referencias o dependencias hacia otros objetos, esto hacía que el código fuese difícil de testear, reusar y entender. Cuando utilizamos inyección de dependencias los objetos

reciben sus dependencias en tiempo de creación por parte de un tercero que coordina cada objeto en el sistema, no se espera que los objetos creen sus dependencias.

- Programación orientada a aspectos.

Este tipo de programación permite la separación de actividades a lo largo de la aplicación en componentes reutilizables. Cada componente en una aplicación tiene encargada una tarea o funcionalidad principal, pero a menudo estos componentes también realizan actividades adicionales más allá de su funcionalidad central. Dichas actividades suelen ser repetidas en múltiples componentes y están relacionadas con temas de transaccionalidad, seguridad, administración, etc.

Lo anterior introduce mucha complejidad al código pero, a manera de solución Spring utiliza la programación orientada a aspectos, la cual consiste a grandes rasgos en anotar con `@Pointcut` los métodos que anteriormente poseían código repetitivo e implementar dicho código mediante un Bean con la anotación `@Aspect` a través de métodos los cuales deben poseer alguna de la siguientes anotaciones `@Before`, `@AfterReturning`, `@AfterThrowing`, `@After` o `@Around`, la elección de la anotación tiene que ver con el momento en que se desea su ejecución con respecto al método anotado con `@Pointcut`. Es importante mencionar que Spring permite implementar lo anterior mediante archivos XML.

- Eliminación de código repetitivo con la implementación de patrones de diseño.

Spring permite el uso de plantillas para la realización de procesos repetitivos. Para explicar lo anterior, se propone considerar el procesamiento de una consulta en base de datos. Cuando se requiere realizar una consulta existe código que siempre debe estar presente: primeramente, la conexión a base de datos, luego la declaración de la misma, la ejecución y finalmente el cierre de la conexión, como consecuencia, el proceso anterior se debe codificar cada vez que una consulta sea solicitada por lo que genera código repetitivo.

Spring permite el uso de plantillas que son estructuras de código que ocultan procesos repetitivos, por ejemplo: `JdbcTemplate` con la estructura siguiente:

```
jdbcTemplate.queryForObject(querryaEjecutar, new
RowMapper(tipoDeObjetoaRecibir))
public tipoDeobjeto mapRow(ResultSet rs, int rowNum){
```

```
//seteo de caracteristicas del objeto, provenientes de la consulta
return tipoDeObjeto}), parámetrosDeConsulta);
```

Con lo anterior se puede ver la creación de una consulta se reduce a unas cuantas líneas con el uso de plantillas.

2.4.1.1 CONTENEDOR DE BEANS.

Según la documentación oficial de Spring el contenedor de Beans es definido por la interfaz `org.springframework.context.ApplicationContext` y es el encargado de crear instancias, configurar, ensamblar y manejar el ciclo de vida de los Beans. Obtiene las configuraciones sobre que objetos instanciar, configurar y ensamblar leyendo los metadatos de configuración proporcionados mediante archivos XML, anotaciones o código Java. Administra los objetos de una aplicación y sus interdependencias entre ellos.[3] Es considerado el núcleo de Spring framework.

2.4.1.2 MÓDULOS.

Spring divide su funcionalidad en módulos, ésta es una gran ventaja ya que permite que aplicaciones hagan uso solo de los módulos que necesiten. A continuación, y de manera breve se explicarán algunos de los principales módulos (mostrados en la Figura 2.1) que conforman a este popular framework.

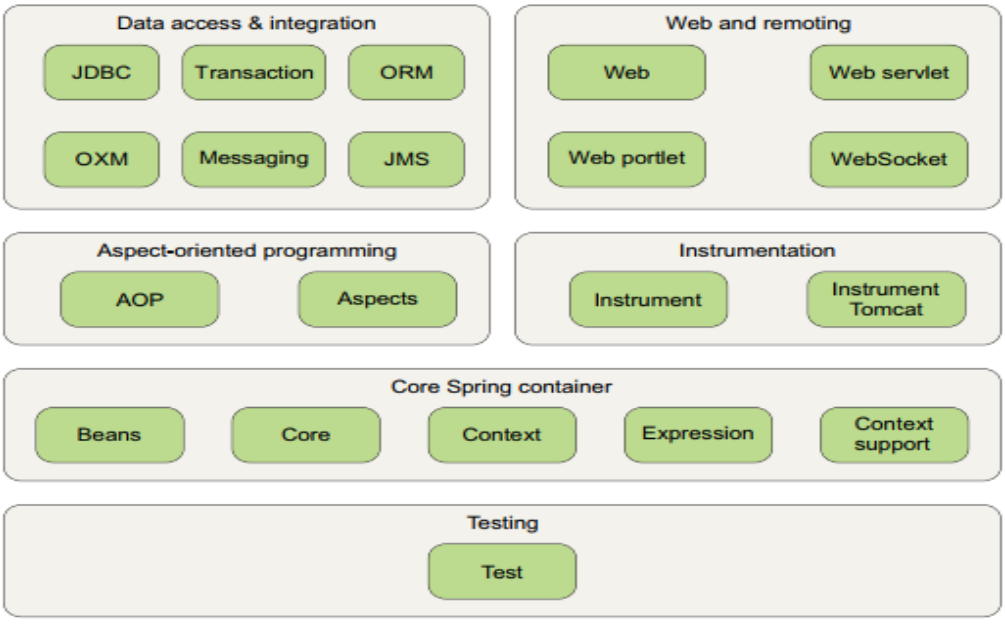

Figura 2. 1. Principales módulos de Spring Framework.

[3] https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html

- Spring Core

Spring Core provee la funcionalidad esencial al framework y está compuesto por las interfaces: `Bean Factory y Aplication Context`, es aquí donde se define el contendor de Beans. Adicionalmente este módulo ofrece servicios como email, Java Naming and Directory interface (JNDI), integración de EJB,etc.

- Data Accesss and Integration

Este módulo se encarga del acceso a datos de forma simple, homogénea, y, sobre todo, evitando código repetitivo: abrir y cerrar conexiones a bases de datos, manejo de errores, etc. Esto permite al desarrollador enfocarse únicamente en la lógica de negocio dejando al framework de Spring la administración y manejo de recursos y dependencias. Una característica importante es la implementación de patrones de diseño como Data Access Object (DAO) el cual permite ocultar la tecnología de acceso a datos al utilizar: JDBC, ORM (Object Relational Mapping), etc. Lo anterior trae como consecuencia un bajo acoplamiento entre la capa de servicios y la capa de acceso a datos.
Entre las tecnologías de acceso a datos más populares se encuentran JDBC y el uso de ORMs cuyas principales implementaciones son Hibernate y JPA (Java Persistence API). Para ellas, Spring ofrece excelentes soluciones de integración.

- Web

El paradigma Modelo Vista Controlador (MVC) es un enfoque comúnmente aceptado para la construcción de aplicaciones web tales que la interface de usuario es independiente de la aplicación lógica. Spring es capaz de integrarse con varios frameworks MVC, pero también posee su propia implementación. Además de las aplicaciones web orientadas al usuario, este módulo también proporciona varias opciones remotas para crear aplicaciones que interactúan con otras aplicaciones. Las capacidades remotas de Spring incluyen la Invocación Remota de Método (RMI), Hessian, Burlap, JAX-WS, y el propio invocador HTTP de Spring. En su última versión Spring incluye un nuevo módulo, Spring-webflux el cual proporciona soporte para programación reactiva.

El módulo Spring-webflux está basado en programación funcional, se crea por la necesidad de una pila web no bloqueable para manejar la concurrencia y escalar con menos recursos de hardware, es ampliamente utilizado para la creación de aplicaciones asíncronas y sin bloqueos. Ofrece dos modelos de programación: controladores anotados y puntos finales funcionales, además este módulo contiene soporte para clientes reactivos HTTP y WebSocket, así como para aplicaciones web de servidores reactivos que incluyen REST, navegador HTML e interacciones de estilo WebSocket.

3. METODOLOGIAS PARA EL DESARROLLO DE SOFTWARE.

3.1. CARACTERÍSTICAS.

Había una vez un equipo de desarrollo, el cual se encontraba elaborando un producto de software. Conforme iba avanzando en dicha tarea, se presentaron una serie de problemas; primeramente, los tiempos, el trabajo no pudo concretarse para la fecha que se esperaba; pues en varias etapas del desarrollo existieron retrasos que provocaron un desfase de tiempos en etapas posteriores, que, sin darse cuenta, dependían de las primeras. Posteriormente existió una desorganización muy grande, ya que, a la hora de delegar actividades entre los integrantes del equipo, cada uno tenía total desconocimiento sobre lo que el otro realizaba, no existía documentación del trabajo y esto solo por mencionar algunos contratiempos.

Finalmente, el equipo empezó a perder motivación, pues, aunque contaban con todos los conocimientos técnicos necesarios para efectuar el trabajo, indudablemente algo faltaba, una metodología para el desarrollo de software.

Podría definirse metodología para el desarrollo de software como un proceso, un conjunto de actividadesa no específicas para el desarrollo de un producto de software (comunicación, planeación, modelado, construcción y despliegue). La metodología comprende todos los aspectos de producción, desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de éste después que se utiliza, entre las características que deben cumplir se tienen las siguientes:

- Especificación: Hace referencia a la recolección de requerimientos necesarios, con la finalidad de que el sistema solicitado funcione de manera correcta y satisfactoria para el cliente. La metodología debe contemplar una solución para realizar la especificación del sistema.

- Diseño e implementación: Manos a la obra. La metodología elegida debe ser capaz de gestionar de manera exitosa la implementación. Es aquí donde se debe crear el sistema, basado totalmente en la especificación acordada.

- Validación y evaluación: Se debe verificar la funcionalidad del sistema, cada requerimiento debe ser comprobado en esta etapa. La metodología empleada debe hacer hincapié en la validación.

En la actualidad existen diversos paradigmas o enfoques para el desarrollo de software en los cuales están basados la mayoría de las metodologías; cada uno promete ser mejor que

el anterior, sin embargo, como requisito, todos llevan en su interior las características antes mencionadas. Entre los principales enfoques destacan:

- Enfoque en cascada.
- Desarrollo iterativo.
- Ingeniería de software basada en componentes (CBSE).

El empleo de cada uno tiene que ver mucho con el tipo de software que se va a desarrollar, pues si bien es cierto, cada sistema tiene un comportamiento diferente y por ende debe ser tratado de diferente manera. Una buena metodología de desarrollo es la clave para obtener un software de alta calidad.

3.2. METODOLOGIAS ÁGILES VS METODOLOGIAS TRADICIONALES.

Hasta ahora se ha hablado solamente de metodologías tradicionales, pero es momento de introducir un nuevo conocimiento: Metodologías ágiles para el desarrollo de software. Definir lo anterior no es tarea fácil, sin embargo, empecemos por definir agilidad; la agilidad es la capacidad de cambiar el estado de un "cuerpo o situación" de manera eficaz y rápida. Con lo aprendido hasta este punto podríamos decir que una metodología ágil se refiere a: un proceso para el desarrollo de software cambiante, el cual se debe realizar de forma rápida y eficaz.

Teniendo conocimiento del mundo en el que nos encontramos, un mundo totalmente dinámico y competitivo, es entendible que las metodologías ágiles sean una necesidad hoy día. Fue en el año 2001 cuando un grupo de notables desarrolladores de software, escritores y consultores, bajo el nombre de "Alianza Ágil" firmaron el "Manifiesto por el software ágil". En él se exponía haber encontrado formas mejores para el desarrollo de software y entre las ideas que establecían se encuentran las siguientes:

- Los individuos y sus interacciones, sobre los procesos y las herramientas.
- El software que funciona, más que la documentación exhaustiva.
- La colaboración con el cliente, y no tanto la negociación del contrato.
- Responder al cambio, mejor que apegarse a un plan.

El principal objetivo del manifiesto fue proporcionar métodos para superar las debilidades que presentaba el desarrollo de software convencional, sin embargo, las metodologías ágiles no vienen a sustituir a las convencionales más bien, son una herramienta más, pues su utilización está recomendada para los casos en los que realmente sea necesaria.

Es normal preguntarse, ¿cuándo es recomendable utilizar una metodología ágil? A decir verdad, la utilización de una metodología ágil es recomendable cuando no sea posible definir los requerimientos por completo antes de que el proyecto comience o sea imposible predecir la forma en que el proyecto evolucionará. Las metodologías tradicionales se rigen bajo una serie de actividades consecutivas y fases, un plan de trabajo inflexible, donde es impensable aceptar un cambio en los requerimientos; pues de ser así, el trabajo debería volverse a realizar, pero esta vez con la integración del nuevo requerimiento, provocando un costo muy alto.

La principal característica y la más atractiva de una metodología ágil, se refiere a la capacidad de adaptarse al cambio, si bien es cierto las metodologías ágiles también hacen uso de un proceso, pero éste está caracterizado por la capacidad de adaptación que posee. En la Tabla 3.1 se muestra un comparativo entre las metodologías tradicionales y metrologías ágiles.

METODOLOGIAS TRADICIONALES	METODOLOGIAS ÁGILES
Rigidez ante los cambios, seguimiento estricto de un plan de desarrollo.	**Flexibilidad ante los cambios.**
Los clientes solo interactúan en las reuniones programadas al inicio del proyecto.	**Los clientes son parte del equipo de desarrollo.**
Existe una dependencia alta de la arquitectura de software mediante modelos.	**Menor dependencia de la arquitectura del software.**
La retroalimentación no se hace de manera continua lo que extiende los procesos.	**Retroalimentación continua.**
Basadas en normas de estándares de desarrollo.	**Basadas en heurísticas a partir de prácticas de producción de código.**
Los procesos son muy controlados por políticas y normas.	**Procesos menos controlados, pocas políticas y normas.**
La especificación del proyecto se realiza al inicio.	**Los procesos de especificación, diseño e implementación son concurrentes, no existe una especificación del sistema detallada.**
El sistema se desarrolla como un todo.	**El sistema se desarrolla en una serie de incrementos.**

Tabla 3. 1. Comparativa entre metodologías tradicionales y metodologías ágiles.

Como se puede ver las ventajas de una metodología ágil son muchas, pero es claro que las desventajas también existen, primeramente, el involucrar al cliente, hacerlo parte del equipo de desarrollo puede no resultar tan fácil, pues en muchas ocasiones el cliente no está dispuesto a jugar este rol; por otro lado, la priorización de los requerimientos puede ser difícil cuando hay varios interesados (*stakeholders)* con expectativas diferentes. Redactar un contrato cuando no se tienen claros los requerimientos es un tanto imposible, en estos casos el cliente paga por el tiempo invertido más no por los requerimientos directamente.

A continuación, se presentan las metodologías ágiles más populares hoy en día.

3.3. PRINCIPALES METODOLOGIAS ÁGILES.

3.3.1. PROGRAMACIÓN EXTREMA XP.

La programación extrema consiste primeramente en ver a los requerimientos como historias de usuarios. Posteriormente, éstas serán transformadas en tareas. Las pruebas se diseñan y se desarrollan antes de iniciar la etapa de desarrollo. Después de lo anterior es momento de comenzar la codificación. Una vez que esta etapa ha sido terminada se deberán ejecutar las pruebas y si estas son exitosas se podrá realizar la entrega o incremento del software. El detalle de estas 4 etapas se muestra a continuación.

- Planeación: Consiste en escuchar al cliente, recabar todos los requerimientos posibles. Con base en esto se crean las historias de usuario, las cuales deben tener un grado de importancia asociado, el cual será tomando en cuenta para elegir que debe realizarse primero. Estas historias de usuario son utilizadas para crear las tareas, para la realización de cada tarea, el tiempo estimado debe ser menor a tres semanas; de no ser así la tarea deberá ser dividida en tareas más pequeñas.
- Diseño: El diseño debe ser lo más sencillo posible, por lo que se hace uso de tarjetas CRC. El diseño es visto como un artefacto en transición que puede y debe ser modificado conforme avanza la construcción.
- Codificación: Se deben crear las pruebas unitarias y posteriormente empezar a codificar lo necesario para que la prueba sea considerada exitosa. Después de realizar la prueba el desarrollador está completamente capacitado para codificar exitosamente, pues realmente tiene conocimiento del problema que debe resolver. A esta técnica se le conoce como diseño dirigido a pruebas (TDD).
- Pruebas: Las pruebas unitarias deben ser fáciles de implementar. Es necesario tomar en cuenta que estas deben ser ejecutadas periódicamente, pues de no ser así, se da la oportunidad de generar problemas mayores y difíciles de resolver.

Es de gran importancia aclarar que este proceso se realiza conforme los requerimientos van cambiando o se van agregando más, permitiendo así tener planeación y al mismo tiempo flexibilidad.

3.3.2. KANBAN.

3.3.2.1. MÉTODO LEAN.

El futuro es impredecible, observamos continuamente que los mercados tienen alzas y bajas en todo momento, y que la producción resulta afectada, debido a que las empresas tienen excedentes de inventarios, generando desperdicios y grandes pérdidas de dinero. Además, éstas se encuentran con otro gran conflicto: satisfacer al cliente.

Cuando hablamos del cliente debemos estar conscientes que está a la espera de resultados, pero no se conforma con cualquier tipo de resultados, él desea obtener resultados de calidad en el menor tiempo posible. Todo ello conllevó a la Empresa japonesa Toyota, a idear un nuevo método de producción, descrito en su libro "The Toyota way, 14 management principles from the world greatest manufacturing", el cual fue nombrado "Lean Manufacturing". Dicho método es muy famoso en la actualidad y pretende resolver el problema, ya que tiene por objetivo utilizar un tipo de producción llamada "just-in-time", la cual propone que sólo se produzca lo necesario, en el momento que se necesite.

Es por la razón anteriormente descrita que se debe llevar un buen flujo de desarrollo, asignando pequeñas tareas a equipos de trabajo, evitando los desperdicios de producción y dando entregables demasiado rápido, no sin antes garantizar la calidad. Para cumplir esto, se utiliza la retroalimentación continua con el cliente, para conocer si lo que se está realizando, es realmente lo que se espera y lo correcto. Este método busca retrasar la toma de decisiones lo más que se pueda, con el fin de tener la mayor información posible y con ello seleccionar la mejor de éstas.

- Para otorgar un conocimiento más amplio se enlistan los principios de "Lean Manufacturing": Las decisiones se deben basar en una filosofía de largo plazo.
- Los productos se mueven continuamente al ritmo en que lo demande el cliente.
- Utilizar el sistema "Pull", el cual propone ordenar las tareas en diferentes áreas, y asignar o quitar tareas cuando ya se han completado haciendo "push y pull".
- Nivelar la carga de trabajo "HEIJUNKA".
- Construir de acuerdo a la calidad.
- Obligaciones estandarizadas.

- Administración visual. Es más sencillo encontrar problemas en las etapas de cada tarea, si son colocadas en un tablero.
- Usar sólo tecnología rentable y comprobada.
- Líderes. Se debe construir un equipo que sea capaz de entender lo que se tiene que hacer, y el encargado de saber cómo hacerlo y guiarlos.
- Se debe lograr que el personal siga la filosofía de la empresa.
- Las relaciones con los socios deben ser fuertes.
- Supervisar el trabajo realizado de manera personal.
- Se deben Resolver los problemas investigando la causa central de éstos.
- Aprender mediante reflexión y una mejora continua.

3.3.2.2. MÉTODO LEAN STARTUP.

Fue Eric Ries, un reconocido emprendedor, quien retomó el concepto anterior y creo uno nuevo: Método Lean Startup, del cual habla en su libro que recibe el mismo nombre.[4] Éste nuevo método lleva en su interior los principios fundamentales de Lean Manufacturing, pero se encuentra enfocado a empresas que buscan innovar en el mercado. Resulta normal preguntarse la razón por la cual es mencionado; la respuesta es sencilla, el autor propone un ciclo de retroalimentación, y se considera que es de mucha utilidad en cualquier sistema de producción, como lo es en nuestro caso el desarrollo de software. Se promueve un ciclo de vida para la retroalimentación en tres etapas: Crear-Medir-Aprender.

Cada servicio o requerimiento del cliente debe ser aprobado tras haber sido validado por el ciclo Crear-Medir-Aprender. En el primer paso, llamado “Crear”, se desarrollan soluciones rápidas de lo que se ha solicitado, mientras que en la etapa de “Medir”, se utilizan diversas técnicas, en esencia pruebas, para saber si es que se cubre lo solicitado. En dicha etapa es donde se debe involucrar al cliente para que éste compruebe si se cumple lo que se necesita. En la última etapa, que lleva por nombre “Aprender”, es en la cual se toma la decisión de desechar o preservar, si se decide no preservar el requerimiento dado a que no cumple con lo establecido vuelve a entrar en el ciclo.

El método establece la importancia de medir la productividad de nuestro equipo de trabajo de forma eficiente, ya que, por ejemplo, la forma tradicional de medir la productividad de un programador, era contar el número de horas que había estado programando sin parar. Sin embargo, la forma que el método propone es un llamado “Aprendizaje validado”, que es lo que se obtiene al recorrer un circuito de

[4] Eric Ries. (2011). El método Lean Startup.: DEUSTO.

retroalimentación continua el cual consiste en obtener retroalimentación por iteración realizada permitiéndonos conocer cuánto se ha perdido y ganado , además de la posibilidad de corregir las cosas que se están haciendo mal desde ese momento y desarrollar el requerimiento de una manera correcta.

3.3.2.3. DESARROLLO DE SOFTWARE CON KANBAN.

Esta metodología de desarrollo de software viene a ser un derivado del Método Lean, explicado en el punto anterior, ya que surgen en el mismo sistema de producción de la compañía japonesa Toyota. Básicamente su origen surge de aplicar esos procesos de producción "just-in-time". Por ello esta metodología es noble y permite que los requerimientos puedan ser modificados en cualquier momento.

La característica primordial de esta metodología es la simplicidad que posee al ser demasiado sencilla de implementar, esto por tener que utilizar una herramienta gráfica indispensable, como lo es una pizarra. El tablero designado se divide en los procesos de desarrollo que sean convenientes para el equipo de producción. Puede ser desde fragmentarlo entre: tareas por hacer, realizando o ya finalizadas, o simplemente asignar cada requerimiento en un pequeño enfoque en cascada, como se mencionó en los capítulos anteriores en cuestión de metodologías tradicionales.

Al momento de tener el tablero, se crean pequeñas tarjetas cuyo propósito es ser asignadas a la sección correspondiente. Estas tarjetas contienen el nombre de cada pequeño grupo de trabajo que tendrá asignado el completar la tarea en la cual sea colocada, así como un tiempo estimado. Kanban no resulta ser un sistema de producción, es de cierto modo, una metodología de desarrollo ágil que busca apoyar en todo momento al equipo de producción.

A causa del sistema "just-in-time" que tiene como base esta metodología, las tareas deben ser rápidas de desarrollarse y con la mejor calidad posible, la cual depende de la buena retroalimentación que se tiene con el cliente, pero no existe algún principio en estas metodologías que guie al equipo a llevar en buenos términos este proceso. Es por esta razón que se ha explicado en el subtema anterior el ciclo "Crear-Medir-Aprender" del método "Lean Startup", el cual es de una gran utilidad para la metodología aquí descrita.

Cuando se escucha que Kanban permite modificar los requerimientos en el instante que se desee, se podría pensar que se tienen que seguir demasiados pasos o principios para lograrlo, pero no es así. Esta metodología cuenta con únicamente tres principios, aunque ellos engloban varias características importantes. Éstos son:

- Empezar con lo que se necesita hacer en ese momento. Existen las llamadas historias de usuario, la cuales se traducen en tareas y estas a su vez en tarjetas (mencionadas anteriormente). Las tarjetas se deben colocar de manera jerárquica, en un orden de importancia.

- Todo el equipo debe aceptar el seguir un cambio incremental y evolutivo, lo que viene siendo la adaptación y aplicación general de un flujo de trabajo. Este es el objetivo central de utilizar un tablero, observar que dicho flujo de trabajo dará una clara idea a todos los integrantes del avance de cada actividad y la facultad de observar los problemas que se presenten, como puede ser: un cuello de botella en una sección, solo por mencionar algo, cualquier evento que implique un descuido durante las etapas de desarrollo.

- Respetar las tareas asignadas a cada integrante. Se debe resaltar que esta metodología deja de lado las iteraciones y las reuniones para planificar. El objetivo consiste en realizar la asignación de tareas, otorgar una fecha de entrega que queda registrada en el tablero y que deberá ser cumplida. No existe un tiempo fijo, cada tarea tiene un tiempo diferente.

Es importante hacer notar que en todo momento se respeta la filosofía del método Lean Manufacturing.

3.3.3. SCRUM.

En la actualidad millones de empresas han optado por desarrollar sus productos mediante Scrum como metodología, entre las más importantes por su impacto mundial se encuentran: Spotify, Adobe, Google y Apple. Estas empresas han destacado por ofrecer a sus clientes un producto completamente apegado a sus necesidades, ser innovadoras y siempre estar flexibles al cambio y a la adaptación. Lo cual se ha logrado de manera exitosa, en gran parte, gracias a la metodología que se utiliza para cada una de las soluciones que ofrecen.

Scrum remonta su aparición a los años 90, nace justo en el boom de la tecnología, cuando mentes creativas preocupadas por resolver los problemas de las metodologías tradicionales y de las ágiles ya existentes, toman cartas en el asunto y se embarcan en la aventura de crear una nueva metodología, basada por su puesto en el manifiesto ágil. Jeff Sutherland y su equipo, fueron los creadores de Scrum, una metodología que destaca por su desarrollo ágil, alta calidad del software, equipos integrados, colaboradores satisfechos

y la fácil escalabilidad. Scrum maneja una filosofía empírica en la que propone aprender de las experiencias que se han tenido, emplea un enfoque incremental e iterativo para optimizar la predictibilidad y el control de riesgo.

Las tres características principales de Scrum son: transparencia, inspección y adaptación. La transparencia hace referencia a que el proceso debe ser visible para cada individuo involucrado en el producto, estos deben compartir un entendimiento común de lo que se está realizando, tener una idea clara de cuál es el objetivo, a dónde se quiere llegar. La inspección, se deben examinar regularmente los productos, pues cualquier variación con lo acordado se debe corregir. Finalmente, la adaptación, si un requerimiento se ha desviado de los límites aceptables o se ha modificado, éste debe tener la capacidad de tomar el camino hacia el objetivo actual.

En el ambiente de Scrum existen algunos roles que hacen posible la metodología:

- Product Owner: También conocido como cliente, representa a todos los individuos interesados en el proyecto, posee gran autoridad para tomar decisiones. Tiene especial intervención en el Product Backlog, del cual se hablará más adelante.
- Development Team: Es el conjunto de individuos autoorganizables y responsables de generar un incremento de producto “terminado”. SCRUM no reconoce sub-roles en el equipo de desarrollo, para SCRUM todos son desarrolladores y la responsabilidad de generar un producto recae sobre todo el equipo. Los equipos generalmente no son mayores a 9 personas, pues SCRUM considera que un equipo pequeño tiene mejores resultados en cuanto a agilidad.
- Scrum Master: Es el líder y responsable de que la teoría y reglas de SCRUM se lleven a cabo y se cumplan por los involucrados. Es el encargado de ser una guía para el Development Team y para el Product Owner, pues trabaja estrechamente con estos. Ayuda a eliminar problemas que pudiesen obstaculizar el logro de los objetivos y proporciona una visualización de la planificación del proyecto en un entorno ágil y empírico.
- Scrum Team: Corresponde a todos los anteriores.

Los anteriores corresponden a los roles que intervienen en la metodología, pero además de roles existen diversos eventos que toman lugar y parte en Scrum. Todos estos eventos son bloques de tiempo, los cuales se enlistan a continuación:

  - Sprint: Se considera el corazón de Scrum, es un bloque de tiempo constante que no va más allá de un mes, entre sus principales características destacan las siguientes:

- Cada que termina un Sprint inmediatamente comienza otro.
- Durante el tiempo que dura cada Sprint se crea un incremento del producto terminado y completamente funcional.
- Un Sprint consta de un diseño y un plan bastante flexible que guiará la construcción del mismo.
- Cada sprint es considerado como un pequeño proyecto al cual no se le realizan cambios que afecten el objetivo.
- Un Sprint puede ser cancelado solo cuando éste ha perdido sentido.
- El Product Owner es el único individuo que tiene autoridad para cancelar un Sprint.
- Cada Sprint consta de Sprint Planning Meeting, Daily Scrums, Sprint Review y Sprint Retrospective, cada evento anterior será explicado detalladamente más adelante.

- Sprint Planning Meeting: Es una reunión de planificación a la que asiste todo el Development Team, la cual está enfocada en resolver las siguientes preguntas:
  - ¿Qué puede entregarse en el incremento resultante del Sprint que comienza?
  - ¿Cómo se conseguirá hacer el trabajo necesario para entregar el incremento?

  Usualmente la reunión tiene una duración de 8 horas como máximo y es aquí donde se elabora el Sprint Goal, el objetivo que tendrá el siguiente Sprint.

- Daily Scrum: Corresponde a una reunión diaria, en la cual el Development Team debe estar presente, la duración es de aproximadamente 15 minutos. En ella se planean las actividades de las próximas 24 horas. Consiste en visualizar el trabajo realizado después del último Daily Scrum y el que se podría concluir antes del siguiente. Está enfocada a responder las siguientes preguntas:

  - ¿Qué hice ayer que ayudó al Equipo de Desarrollo a lograr el Objetivo del Sprint?
  - ¿Qué haré hoy para ayudar al Equipo de Desarrollo a lograr el Objetivo del Sprint?
  - ¿Veo algún impedimento que evite que el Equipo de Desarrollo o yo logremos el Objetivo del Sprint?

- Sprint Review: Es una reunión que se realiza al finalizar el Sprint con una duración de 4 horas como máximo, en ella debe asistir el Scrum Team. En esta reunión cada rol expuesto anteriormente tiene una intervención de gran importancia. El Development Team habla de los retos que se encontró al realizar el producto y responde preguntas de este mientras que el Product Owner habla acerca de los elementos terminados del Product Backlog y de su estado actual.

- Sprint Retrospective: Se trata de una reunión a la que debe asistir el Scrum Team y que tiene lugar después del Sprint Review y antes del próximo Sprint. En dicha reunión el equipo se inspecciona a sí mismo para encontrar sus debilidades y fortalezas, y de este modo poder crecer.

Después de conocer los roles y eventos que tienen lugar en Scrum es momento de presentar los artefactos, estos fueron pensando con el objetivo de crear un mayor nivel de transparencia en el proceso.

- Product Backlog: Corresponde a la lista de requerimientos que el Product Owner solicitó, él es el responsable de actualizarla, la lista nunca estará completa, pues irá evolucionando conforme los requerimientos se modifiquen. Esta lista contiene todas las características y requerimientos del sistema, cada requerimiento posee una descripción, una ordenación y un valor. Los elementos se orden basándose en el grado de prioridad que estos tienen para el sistema.
- Sprint Backlog: corresponde a la lista de tareas que es elaborada en el Sprint Plannig Meeting, con el objetivo de generar exitosamente una iteración. Para cada uno de los objetivos se muestran las tareas que se deben realizar para alcanzarlo.

En próximos capítulos se presenta una lista de pros y contras entre las tecnologías presentadas.

3.4. INTEGRACIÓN CONTINUA.

Como define Martin Fowler, uno de los autores más reconocidos en el ámbito de prácticas ágiles, la integración continua es una práctica en la que miembros de un equipo integran su trabajo frecuentemente, típicamente cada persona íntegra al menos una vez al día, generando varias versiones por día. Cada versión ejecutable es verificada por un sistema automático de integración y pruebas para detectar errores de integración lo más rápido

posible.[5] Es considerado uno de los pilares de las metodologías ágiles debido al gran número de ventajas que aporta. Las principales ventajas se enlistan a continuación:

- Disminución de la cantidad de cambios a integrar y por ende del número de errores.
- Acelera la detección de fallas.
- Disminuye el tiempo dedicado a depurar errores.
- Automatiza el proceso de integración y ejecución de pruebas.
- Monitoreo de métricas de calidad para el proyecto.
- Automatiza el despliegue.

Con la implementación de la integración continua se puede tener un proceso automatizado para ensamblar y probar versiones ejecutables del producto de software que se está generando.

3.4.1. ¿CÓMO FUNCIONA?

Para poder comprender el funcionamiento de la integración continua debemos definir y tener claros algunos conceptos:

- Repositorio: Software mediante el cual se almacena el código fuente de un proyecto, así como algunos otros recursos. Muchas veces funciona como versionador del mismo. En cuanto a integración continua, solo debe existir un único repositorio (llamado repositorio maestro) al que cualquier integrante del equipo tenga acceso.
- Máquina de integración: Es aquella donde se realizan las construcciones a partir del repositorio. De esta forma se reducen ciertas suposiciones sobre el entorno y la configuración de éste. Aquí se tiene la posibilidad de hacer uso de un servidor de integración continua.
- Automatización de Integración: Consiste en realizar una serie de tareas con el objetivo de generar un ejecutable de nuestra aplicación, que haya sido probado, compilado y medido a través de métricas de calidad (según las reglas de negocio).

Una vez que hemos entendido lo anterior no queda más que explicar el funcionamiento. Primeramente, cada desarrollador debe tener de manera local una copia del repositorio, una vez que ha realizado una actualización al código debe enviar sus cambios al repositorio maestro. El servidor de integración continua, monitorea los cambios al repositorio maestro y al detectar una actualización, realiza la integración y corre las pruebas. Finalmente publica los ejecutables e informa al equipo el resultado de la integración.

[5] Carlos Blé y colaboradores. (2010). Diseño Ágil con TDD.: Creative Commons

3.5. COMPARACIÓN ENTRE METODOLOGÍAS ÁGILES.

Hasta ahora pudiera parecer complicado elegir la metodología adecuada, teniendo ante nuestros ojos tantas características ventajosas por cada una, sin embargo, es importante preguntarse, antes que nada: ¿Qué es lo que realmente se necesita en el proyecto que se realizará?, pues las metodologías fungen como una herramienta para el desarrollo de un proyecto y es importante conocer y estar conscientes que ninguna herramienta es completa y perfecta. En el presente capítulo se hará una comparación entre metodologías, con el objetivo de realizar una elección adecuada.

Una herramienta se considera útil porque limita las opciones que se tienen, no nos permite realizar todo, pero lo que nos permite es justo lo que se necesita. Lo anterior es llamado "prescripción". Prescripción se puede definir como un conjunto amplio de reglas a seguir.
La Figura 3.1 muestra el grado de prescripción que posee cada metodología expuesta.

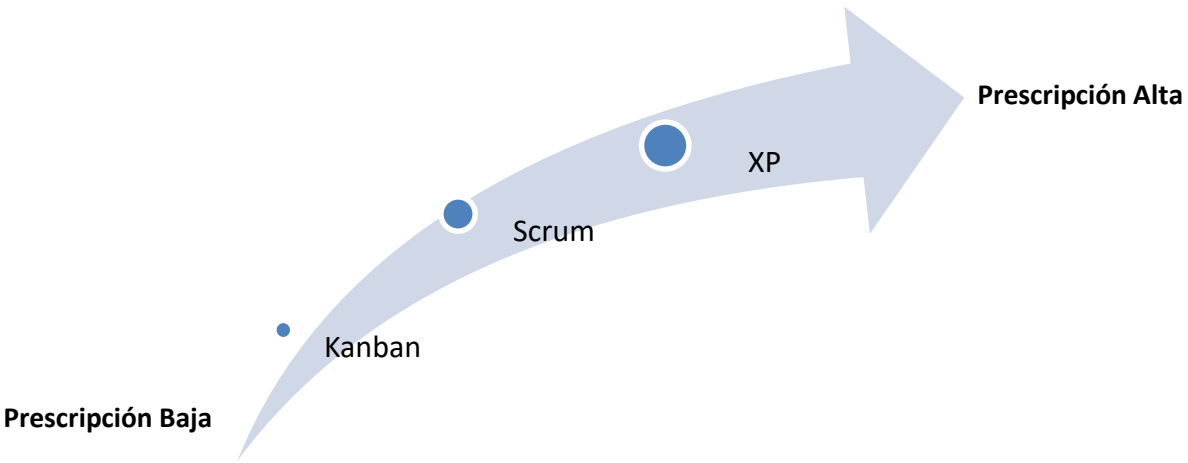

Figura 3. 1. Grado de prescripción de las principales metodologías ágiles.

Como la Figura 3.1 muestra, Kanban es una metodología bastante adaptativa, primeramente, esto se debe a que no prescribe roles, sin embargo, como se vió

anteriormente Scrum y XP si lo hacen. La cantidad de roles que maneja Scrum es menor en consideración a la que maneja XP. En algunos casos el empleo de ciertos roles no aporta absolutamente ningún valor, pero provoca ciertas perdidas, sobre todo en proyectos pequeños.

El manejo del tiempo en las iteraciones es otra característica muy debatible. Scrum y XP manejan iteraciones de tiempo fijo, en las cuales se entregan productos terminados en un determinado plazo. Para Kanban es diferente, las iteraciones de tiempo fijo desaparecen y en su lugar se manejan iteraciones bajo demanda o de manera regular. Además, Scrum y Kanban integran un pizarrón de tareas, éste maneja para Kanban un número limitado de actividades en curso, mientras que para Scrum no existe tal limitación; sin embargo, no es recomendable mantener gran cantidad de tareas en curso. Es importante mencionar que la única limitación que presenta Scrum ante el pizarrón, es por sprint, una vez que el sprint ha terminado las tareas deben ser sacadas de la pizarra.

En Scrum, no se permiten cambios cuando una iteración o Sprint ya ha iniciado, pero Kanban es bastante flexible en este aspecto, aceptando cambios sin importar el estado de cada iteración. En cuanto al equipo de desarrollo, Kanban maneja regularmente más de un equipo de desarrollo para cada iteración, mientras que Scrum solo un equipo, capacitado en su totalidad para completar una iteración; se manejan equipos multifuncionales. Sin embargo, los equipos deben estimar el tamaño relativo de cada elemento al que se comprometen, para poder asegurar su realización exitosa y planeación realista, esto generalmente se refiere a la velocidad, la cual se calcula de la siguiente forma:

$$v = capacidad - cantidad\ de\ elementos\ ofrecidos\ por\ sprint$$

Kanban no realiza ningún tipo de estimación.

Scrum realiza el seguimiento de las actividades apegándose lo mayormente posible al Product Backlog, para XP es tremendamente estricto seguir el orden de prioridad de las tareas.

	SCRUM	KANBAN	PROGRAMACIÓN EXTREMA
Grado de prescripción	Medio	Bajo	Alto
Manejo de iteraciones de tiempo fijo	Aplica	No aplica	Aplica
Manejo de roles	Aplica	No aplica	Aplica
Pizarrón de tareas	Aplica	Aplica	No aplica
Flexibilidad ante los cambios	Media	Alta	Media
Grupos de trabajo	Uno	Más de uno	Uno
Prioridad de actividades	Media	Baja	Alta
Cálculo de velocidad de tarea	Aplica	No aplica	No aplica

Tabla 3. 2. Comparativa entre principales metodologías ágiles.

La Tabla 3.2 muestra un comparativo entre metodologías basándose en sus principales características.
En el próximo subcapítulo se muestra la elección de la metodología adecuada para el desarrollo de la herramienta propuesta.

3.6. APLICACIÓN.

El primer paso en la elección de una metodología consiste en realizar un análisis entre el comportamiento que tiene el negocio, el cliente y el equipo de trabajo. Dicho análisis se muestra a continuación.

- Negocio
  - Los requerimientos de la herramienta a desarrollar ya están definidos casi en su totalidad, pero se esperan pocos cambios cuyo impacto se desconoce.

- Cliente
  - Solo se cuenta con un cliente o interesado.
  - Disponibilidad para ser involucrado en el desarrollo media/baja.

- Equipo de desarrollo

- El equipo de desarrollo es muy pequeño.
- Solo se involucrará a un equipo de desarrollo.

Con base en lo anterior se ha realizado un análisis detallado de la mejor metodología a elegir, sin embargo, se llegó a la conclusión de que ninguna satisface por completo las necesidades de la herramienta a desarrollar. Sin embargo, lo anterior no es sinónimo de preocupación; es común que esto ocurra en la práctica, pues como se dijo anteriormente, las metodologías no son perfectas, a pesar de esto, se tiene la libertad de realizar una "mezcla", puede sonar como un proceso bastante arriesgado, en el que algo pudiese salir mal, un descontrol total, pero la realidad es que no es así. Es habitual encontrarse con equipos de desarrollo que han optado por mezclar dos metodologías (incluso más), pues ambas tienen características necesarias y útiles. Para este trabajo de tesis se utilizará una mezcla entre Kanban y Scrum. Por considerar que el complemento de ambas satisface por completo las necesidades del proyecto.

Para un amplio entendimiento la Tabla 3.3 muestra la justificación a la decisión tomada.

	SCRUM	**KANBAN**	**PROGRAMACIÓN EXTREMA**
Roles: El manejo de roles en el equipo no es necesario, pues el equipo es pequeño.	No satisface	Satisface	No satisface
Equipos de trabajo relacionados: Solo se trabajará con un único equipo.	Satisface	No satisface[6]	Satisface
Flexibilidad ante los cambios: La herramienta sufrirá pocos cambios. Se requiere una flexibilidad ante los cambios media.	Satisface	No satisface	Satisface
Reuniones de gestión: Se pretenden evitar reuniones por la complejidad de tiempos con los involucrados.	No Satisface	Satisface	No Satisface
Iteraciones: se requiere el manejo de iteraciones de tiempo fijo.	Satisface	No satisface	No satisface[7]

[6] Kanban está orientado para trabajar con más de un equipo.

[7] 2Las iteraciones suele tener un periodo de tiempo muy corto

Tabla 3. 3. Comparativa entre principales metodologías ágiles basada en características relacionadas con el equipo de trabajo y naturaleza de la herramienta a crear.

Se considera importante especificar qué elementos o características serán implementados por metodología. Primeramente, por tratarse de un equipo pequeño, se ha optado por no trabajar con roles, y evitar las reuniones diarias, estas características, corresponden a Kanban. Posteriormente se utilizarán solo dos eventos que maneja Scrum, pues se cree, proporcionan una retroalimentación y fomentan la comunicación del equipo de una forma muy exitosa, estos son el Sprint Planning y Sprint Retrospective. Finalmente se ha optado por hacer uso de los artefactos de Scrum (Sprint Backlog y Product Backlog), esto, por considerar la facilidad de implementación y el gran impacto que poseen.

Una vez elegida la metodología el siguiente paso es comenzar a hacer uso de ésta y explotarla lo mejor posible.

4. REQUERIMIENTOS DE SOFTWARE.

Hasta ahora se ha hablado de requerimientos de manera implícita, pero ha llegado el momento de profundizar en ello.

Los requerimientos de software de forma general pueden definirse como las características y/o requisitos con los que debe cumplir un sistema de software para que éste pueda ser aceptado y operar de manera exitosa. De la buena definición de tales depende la calidad del producto final.

4.1. INGENIERÍA DE REQUERIMIENTOS.

La ingeniería de requerimientos tiene por objetivo mejorar la forma en que comprendemos y definimos un sistema de software en particular, consiste en transformar las necesidades del cliente en requerimientos mediante un proceso. A través de tal proceso se recopilan, analizan y verifican dichas necesidades teniendo como resultado una especificación de requerimientos, clara, concisa y completa, evitando ambigüedades. Los individuos involucrados en este proceso son principalmente el cliente, usuarios y diseñadores del sistema.

4.2. LENGUAJE UNIFICADO DE MODELADO (UML).

El Lenguaje Unificado de Modelado (UML) es un estándar para la creación de representaciones gráficas de sistemas de software basado en los requerimientos de usuario. Permite una comunicación y entendimiento clara entre el equipo de desarrollo y el cliente, ya que organiza el proceso de diseño de tal forma que los involucrados comprendan y convengan con él.

Existen diferentes tipos de diagramas UML que permiten analizar al sistema desde diferentes puntos de vista. A continuación, se presenta una breve clasificación y descripción de cada uno.

- Enfocados en el análisis y requerimientos:
  - Diagrama de casos de uso: Establece una idea dinámica del sistema. Es una descripción de las acciones de un sistema desde el punto de vista del usuario, modelan la funcionalidad del sistema usando actores y casos de uso (servicios o funcionalidades).

  - Diagrama de estados: Muestra la secuencia de estados por los que pasa un caso de uso o el sistema en general a lo largo de su vida.
  - Diagrama de secuencias: Muestra la mecánica de las interacciones con base a tiempos.
  - Diagrama de actividades: Demuestra la serie de actividades que deben ser realizadas en un caso de uso, así como las distintas rutas que pueden surgir. Una actividad suele representar una operación en alguna clase del sistema.
- Enfocados en el diseño:
  - Diagrama de clases: Describen la estructura estática de un sistema. Consiste en ver al problema con un conjunto de clases las cuales poseen atributos y métodos, así como relaciones entre ellas.
  - Diagrama de objetos: Describe la estructura estática de un sistema en un momento en particular. Los diagramas de objetos están vinculados a los diagramas de clases, son vistos como una instancia de estos.
  - Diagrama de componentes: Describe la interacción entre componentes, siendo un componente un módulo de clases que representa a un sistema independiente.

Para el caso de estudio que se está trabajando se considera adecuado el empleo de los siguientes tipos de diagramas:

- Diagrama de casos de uso: Se ha elegido debido a la facilidad con que realiza la representación de las interacciones entre el usuario y el sistema, además de modelar el flujo básico de los eventos en el sistema.
- Diagrama de actividades: Se ha elegido debido a su capacidad de describir los pasos realizados en un caso de uso, el proceso de negocio y el flujo de trabajo.
- Diagrama de clases: Se ha elegido debido a la abstracción del problema que permite realizar mediante el uso de clases.

Gracias a la implementación de los diagramas anteriores los tiempos de desarrollo disminuyen, existe una alta reutilización y minimización de costos.

4.3. TIPOS DE REQUERIMIENTOS.

Los requerimientos usualmente se dividen en diversos tipos basados en sus características primordiales. Generalmente éstos van surgiendo mientras avanza la entrevista con el cliente y se tiene mayor conocimiento del sistema a desarrollar.

4.3.1. REQUERIMIENTOS FUNCIONALES.

Estos se basan en la definición de las reglas de negocio funcionales, generalmente no están presentes aspectos técnicos o conceptos muy detallados sobre lo que se desea, van completamente orientados a los servicios y restricciones que se esperan. Usualmente estos requerimientos se apoyan en diagramas UML.

Para el caso de estudio que se ha propuesto en este trabajo los principales requerimientos funcionales se encuentran a continuación.

"Se desea la creación de un sistema enfocado a contribuir y dar seguimiento al crecimiento profesional de un individuo mediante la ejecución de una estructura jerárquica de aprendizaje basada en diversos cursos de capacitación enfocados en una o más áreas de interés. Dicha Jerarquía será propuesta por el propio sistema, o puede ser personalizada por el individuo".

Dicho sistema debe comprender las siguientes funcionalidades:

- Gestión de un catálogo de cursos dinámico que abarca desde su creación, seguimiento, inscripciones por parte de los interesados, administración de costos, promociones y proceso de cobranza.
- Construcción de Jerarquías de especialización basada en diversos perfiles. Por ejemplo: Especialista en desarrollo Front End, especialista en análisis de datos, etc.
- Publicación dinámica de apertura de cursos sin la necesidad de realizar programación alguna por parte de la organización o empresa que ofrece este tipo de servicios en su sitio o portal en Internet.
- Gestión de promociones y descuentos que pueden ofrecerse a los interesados.
- Registro de interesados, así como de la información necesaria para iniciar con un proceso de seguimiento y construcción personalizada de su propia jerarquía de
- especialización.
- Gestión de las inscripciones de cada curso las cuales comprenden procesos administrativos y financieros, como son: gastos de operación, cobranza de cursos empleando diversos mecanismos y facilidades de pago.
- En una segunda etapa se propone realizar una gestión automática de la información con la finalidad de construir campañas de marketing digitales dirigidas principalmente a las redes sociales

Como se puede observar con la información arriba proporcionada, la descripción del sistema resulta un tanto ambigua y poco clara, sin embargo, representa un primer acercamiento a los requerimientos reales. Los interesados en este tipo de requerimientos

usualmente son administradores, gerentes y usuarios finales, para los cuales la implementación del sistema es de gran interés.

Se ha decidido realizar una representación gráfica de los requerimientos antes mencionados mediante el empleo de un diagrama de casos de uso el cual corresponde a la Figura 4.1.

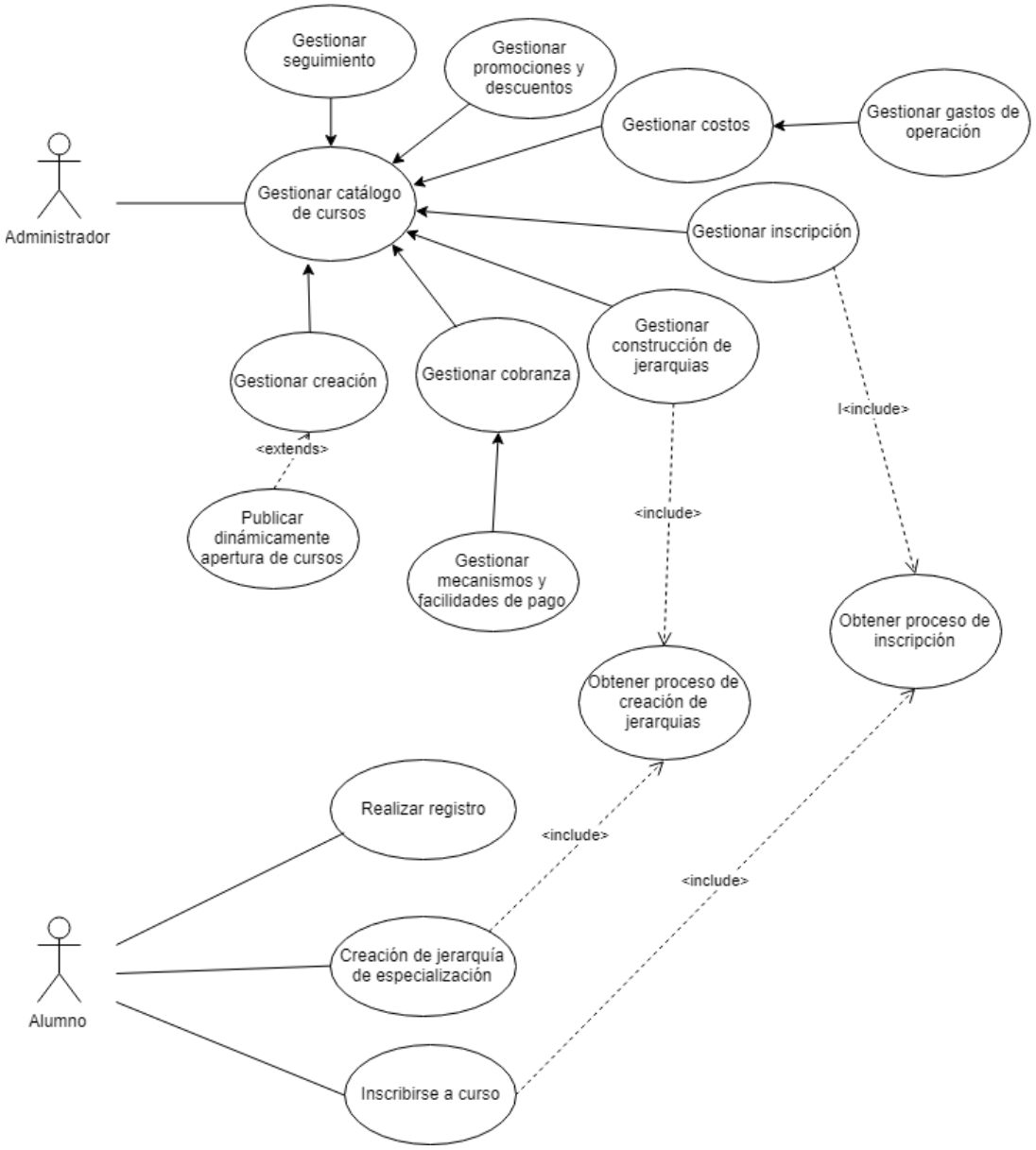

FIgura 4. 1. Diagrama de caso de uso.

4.3.1.1. HISTORIAS DE USUARIO.

Como lo definen Sondra Ashmore & Ph.D.Kristin Runyan, una historia de usuario es la descripción de una característica solicitada, pequeña y simple, incorpora la perspectiva del usuario, no especifica un marco de tiempo, pero hay un concepto de tiempo basado en la priorización. Generalmente se utilizan tarjetas donde estas características son plasmadas.[8] El artefacto de Scrum llamado Product Backlog podría verse como una lista de historias de usuario donde el orden en el que aparecen determina la prioridad. Cada historia concluida corresponde a un incremento del producto final.

Para que una historia de usuario resulte exitosa debe ser:

- Independiente: Debe ser capaz de implementarse y probarse como un único elemento. No debe depender de otras actividades.
- Negociable: Debe propiciar la discusión sobre la mejor forma de resolver el problema planteado.
- Valuable: Debe aportar valor al negocio, de no ser así no tiene sentido trabajar en ella.
- Estimable: Se debe poder estimar la complejidad y el tiempo de entrega.
- Pequeña: Debe ser completada en una iteración.

Las historias de usuario deben responder las siguientes preguntas:

- ¿Quién?
- ¿Qué?
- ¿Por qué?

Los detalles determinados a la historia se vuelven pruebas. Las pruebas se pueden anotar en la parte posterior de la tarjeta de historia.

A continuación, se presenta la implementación de una historia de usuario la cual corresponde al caso de uso gestión de creación de curso (Figura 4.2).

[8] Sondra Ashmore & Ph.D. Kristin Runyan. (2014). Introduction to Agile Methods. United States of America: Pearson.

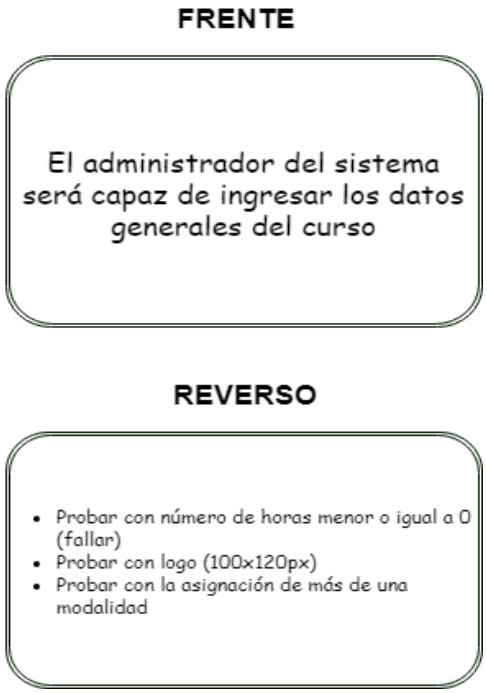

Figura 4. 2. Tarjeta de historia de usuario.

La Implementación de la anterior nos permite entender de una forma clara y completa el requerimiento solicitado, al mismo tiempo que facilita la búsqueda de soluciones.

4.3.2. REQUERIMIENTOS NO FUNCIONALES.

Representan características que deben ser implementadas por el sistema para responder ante la ocurrencia de ciertos eventos independientes a las reglas funcionales o de negocio. Estas características permiten una correcta operación y son referidas como atributos. Ejemplos de estos atributos: eficiencia, seguridad, usabilidad, fiabilidad, tiempo de respuesta, rendimiento, etc.

Los requerimientos no funcionales surgen a través de necesidades de usuario, debido a restricciones presupuestales, políticas, de interoperabilidad y factores externos como regulaciones de seguridad o legislaciones de privacidad.

Este tipo de requerimientos visualizan al sistema como un todo, pues el fracaso al cubrir un requerimiento no funcional provocaría deficiencias importantes. Es por ello que, los requerimientos no funcionales afectan en buena medida a la arquitectura global que a los componentes individuales.

Los requerimientos no funcionales se pueden clasificar en tres grandes grupos:

- Requerimientos del producto: representan una especificación o restricción ante el comportamiento del software, pues dictan que tan fiable, seguro y eficiente debe ser.
- Requerimientos de la organización: son derivados de políticas y procedimientos del cliente y desarrollador. Corresponden a condiciones de usabilidad, desarrollo y entorno.
- Requerimientos externos: corresponde a requerimientos externos al sistema y su proceso de desarrollo. Generalmente se refiere a regulaciones externas con las que el sistema debe cumplir.

En cuanto a este caso de estudio los requerimientos no funcionales corresponden a los siguientes:

- Se requiere de una aplicación Web que cuente con una disponibilidad de al menos el 98%.
- La experiencia del usuario debe ser similar a la experiencia adquirida en un sistema de escritorio: responsiva, sin bloqueos por tiempos de procesamiento.
- El registro de los datos del estudiante debe ser resguardados con los mecanismos de seguridad necesarios para evitar problemas como robo de identidad, robo de información.
- El sistema deberá ser capaz de comunicarse con los servicios necesarios para realizar la cobranza segura de cursos empleando los siguientes tipos de pago: Tarjeta de crédito y transferencias bancarias.
- En una segunda etapa, se pretende que el sistema sea capaz de comunicarse de manera segura con sistemas de redes sociales como Facebook, Twitter con la finalidad de publicar contenido en cuanto ofertas, promociones, y en general campañas digitales de marketing a través de Servicios Web tipo REST principalmente.
- La interface de administración deberá ser lo suficientemente amigable, y eficiente para realizar una adecuada administración de los catálogos y procesos de negocio de la empresa.
- El sistema deberá estar compuesto por módulos funcionales de tal forma que, si se requiere realizar cambios a uno de ellos, el otro modulo deberá seguir en línea sin que este se vea afectado.
- El esquema de seguridad estará basado en la definición de un conjunto de roles funcionales permitiendo la conexión segura sobre Internet.
- El sistema requiere el envío masivo de correos electrónicos para realizar la promoción de nuevos cursos, ofertas y promociones.

- El sistema deberá ser diseñado y construido de tal forma que permita una integración relativamente simple para que pueda ser instalado en cualquier momento en una solución basada en la nube. El sistema deberá ser diseñado de tal forma que maximice las características de este tipo de soluciones: Sistemas distribuidos.

Como se puede observar con los requerimientos obtenidos, la especificación del sistema ya ha tomado claridad y consistencia lo cual es de vital importancia en el desarrollo del mismo.

4.4. TECNOLOGÍAS A UTILIZAR

Una vez que se han analizado y comprendido los requerimientos del cliente (funcionales y no funcionales) ha llegado el momento de seleccionar las tecnologías que se utilizarán, para ello se presenta la Tabla 4.1 donde se muestra una comparación entre las tecnologías más populares del momento.

REQUERIMIENTO A SATISFACER	**Java EE**	**Spring Framework**	**.NET**
Clusterización	No ofrece soporte directamente.	No ofrece soporte directamente.	No ofrece soporte directamente.
Seguridad	Si cuenta con soporte.	Si cuenta con soporte.	SI cuenta con soporte.
Soporte para implementación de modelos de computo distribuido	No cuenta con soporte directamente.	Si cuenta con soporte directamente.	Si cuenta con soporte.
Microservicios	No cuenta con servicio directamente.	Si cuenta con soporte directamente.	No cuenta con soporte directamente.
Envío masivo de email	SI cuenta con soporte directamente.	Si cuenta con soporte directamente.	Si cuenta con soporte.
Servicios web Rest	Si cuenta con soporte directamente.	Si cuenta con soporte directamente.	Si cuenta con soporte directamente.
Código abierto	SI aplica.	Si aplica.	No aplica.
MULTIPLATAFORMA	SI aplica.	Si aplica.	No aplica.

Tabla 4. 1. Comparativa entre tecnologías con base en requerimientos previamente establecidos.

Como se puede observar la tecnología que cumple en mayor medida con las características requeridas es Spring, por lo cual se ha elegido para realizar el desarrollo del caso de estudio planteado.

5. DISEÑO Y ARQUITECTURA DE SOFTWARE.

Una vez que se ha realizado la selección de las tecnologías a utilizar lo siguiente es definir el diseño y la arquitectura del sistema.

Se considera importante y de gran valor presentar las definiciones de los conceptos diseño y arquitectura de software.

- Diseño de Software: Según Ian Sommerville corresponde a una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados, las interfaces entre componentes del sistema y, en ocasiones, los algoritmos usados.[9]
- Arquitectura de Software: Según Isidro Ramos Salavert & Maria Dolores Lozano Pérez se puede definir como una representación de alto nivel de la estructura de un sistema o aplicación, que describe las partes que lo integran y las interacciones entre ellas, estableciendo un marco de referencia para guiar la construcción del software.[10]

Establecer el diseño y arquitectura de software de manera adecuada no es tarea fácil, pero es de vital importancia para la construcción de software de calidad. Contar con una selección apropiada de los anteriores nos traería un sinfín de beneficios entre los que destacan:

- Desarrollo de software escalable, flexible y mantenible.
- Reducción de costos.
- Aumento en el rendimiento.
- Incremento en calidad.
- Aumento en la velocidad de desarrollo.
- Simplicidad.

A continuación, se presentan algunas estrategias de diseño y arquitecturas de software comúnmente empleados.

5.1. DISEÑO ORIENTADO A ENTIDADES (Domain Driven Desing).

Este diseño de desarrollo de software tiene sus bases en el conocimiento a fondo del dominio del software que se pretende desarrollar, es decir, entender cuál es su finalidad y los problemas que se pueden presentar en su ciclo de vida, esto con el objetivo de

[9] Ian Somerville. (2011). Ingeniería de software. Argentina: Pearson
[10] Isidro Ramos Salavert & Maria Dolores Lozano Pérez. (2000). Ingeniería de software y bases de datos, tendencias actuales.: Ediciones de la Universidad de Castilla-La Mancha.

mantener una relación de armonía entre el software y el dominio, al mismo tiempo que se construye software robusto y adaptable al cambio. Sin embargo, conocer el dominio no es tarea fácil, pues las personas que tienen total conocimiento del mismo son los usuarios finales y en algunos casos el cliente, finalmente los expertos. La tarea del equipo de desarrollo es realizar una abstracción de la información recolectada a través de las personas involucradas y construir un modelo. El modelo puede ser representado de muchas maneras las más comunes son: diagramas, casos de usos, dibujos, de manera escrita, etc., el objetivo es plasmar y comunicar.

Para poder tener un entendimiento claro del dominio, es importante que expertos y equipo de desarrollo entiendan un mismo lenguaje, es decir que ninguno hable su propia jerga impidiendo que los otros entiendan, para esto se debe hacer uso del modelo. Una vez que se ha construido un lenguaje entendible para todos, el siguiente paso es transferirlo a código, el diseño orientado a entidades consta de cuatro capas:

- Capa de interface de usuario: Es la responsable de presentar la información al usuario e interpretar los comandos de usuario.
- Capa de aplicación: Es la encargada de coordinar la actividad de la aplicación, no contiene lógica de negocio ni posee la capacidad de guardar el estado de los objetos. Delega el trabajo a los objetos de dominio de la capa siguiente.
- Capa de Infraestructura: Funciona como una librería para las otras capas, provee comunicación entre capas y maneja la persistencia de los objetos de negocio.
- Capa de dominio: Es considerada el corazón del software, pues contiene información del dominio y las reglas de negocio, el estado de los objetos de negocio se encuentra aquí. Resuelve los problemas para los que el software es creado.

El diseño orientado a entidades posee algunos elementos y/o características de gran importancia para poder funcionar de manera óptima, los cuales serán explicados ampliamente a continuación:

- Entidades: Este tipo de objetos poseen una identidad, son procesados en memoria y poseen un tipo de identificador único entre todos los objetos de un mismo tipo. Se tiene interés en conocer el ciclo de vida de cada objeto considerado una entidad.
- Objetos de Valor: Los objetos de valor son aquellos que no tienen identidad propia y es recomendable que sean inmutables, no es de interés distinguir uno de otro y su estado no afecta el estado de la aplicación, es posible que éstos contengan otros objetos de valor, son totalmente dependientes de las entidades ya que están inmersos en ellas.

- Servicios: Se podrían definir como aquellas actividades que no son propias de ninguna entidad, es decir ninguna entidad es responsable de su realización, simplemente corresponden a funcionalidades utilizadas por éstas. Generalmente los servicios de negocio son modelados y diseñados a través del uso de interfaces las cuales permite ser implementadas de diferentes formas y con diferentes tecnologías.
- Módulos: Cuando un modelo comienza a crecer es mejor dividirlo en pequeños módulos, esto con el objetivo de reducir el acoplamiento y aumentar la cohesión.
  El acoplamiento corresponde al grado de dependencia que tienen las unidades de software entre ellas, cuando se tiene un bajo acoplamiento las consecuencias resultan favorables, pues la detección y corrección de errores se vuelve sencilla, así como el mantenimiento, además el bajo acoplamiento da lugar a la reutilización de código. Por otro lado, la cohesión tiene que ver con la adecuada definición del propósito de cada unidad de software, es decir, se debe buscar que cada unidad de software tenga un único propósito, esto conlleva unidades de software más sencillas de programar, diseñar, probar y mantener.
  Usualmente los módulos tienen definidas interfaces las cuales son accedidas por otros módulos.
- Agregados: Cuando se tienen varias entidades relacionadas entre sí y dependientes entre ellas se suelen agrupar en agregados. En un agregado se define quien será la entidad raíz y mediante esta se accede a las demás, la entidad raíz posee acceso público. Si los objetos de un agregado están en la base de datos solo la raíz puede obtenerlos a través de querréis, los objetos en un agregado mantienen referencia a la raíz.
- Fábricas: Las fábricas son utilizadas para encapsular el conocimiento necesario para la creación de los objetos y son empleadas especialmente para crear agregados, esto por la complejidad que implica la creación de un agregado.
- Repositorios: Los repositorios son utilizados para proporcionar un puntero a cada objeto creado y cuando éste no esté creado debe tener la capacidad de indicarle a la fábrica la creación del mismo, se comporta como un intermediario entre la persistencia y la capa de negocio. Son utilizados para acceder únicamente a las entidades raíces.

Es importante entender que el objetivo de este diseño es dividir nuestro dominio en capas y ser capaces de reconocer los elementos antes mencionados a través de dichas capas de manera exitosa.

5.2. DISEÑO ORIENTADO A INTERFACES.

El objetivo del diseño orientado a interfaces es desarrollar una solución de software después de un análisis exhaustivo de la problemática a resolver a través de la abstracción y mediante el empleo de interfaces, según Ken Pugh[11] este tipo de diseño se rige por tres principales reglas:

1. La implementación de una interface debe hacer lo que sus métodos dicen que hace, los nombres de éstos deben corresponder a la acción que ejecutan.
2. Una interfaz no debe inferir con otros módulos de un programa o con otros programas, es decir no debe utilizar recursos de otros módulos o afectar su funcionamiento.
3. Si una implementación no es capaz de realizar su responsabilidad debe notificar a quien la llamó la existencia de algún problema empleando un adecuado mecanismo de manejo de excepciones o errores.

Además de las tres reglas básicas explicadas anteriormente, cuando se trabaja con interfaces un concepto nuevo aparece, los llamados contratos. Como su nombre lo dice, establecen cláusulas que deben cumplirse para un trabajo exitoso. El que "llama" tanto como el que "implementa" deben estar conscientes de las precondiciones y postcondiciones que la interface posee, además de los aspectos invariantes que cada clase debe satisfacer. Para asegurar que el contrato se cumpla se debe afirmar que las condiciones han ocurrido, esto se logra a través de código (se puede hacer uso de la programación orientada a aspectos). Es importante entender que una interface debe conocer su contrato.

5.2.1. TIPOS DE INTERFACES.

Existen varios tipos de interfaces basados en características sustanciales, las cuales se desglosan a continuación:

- Interface de datos: esta interface se caracteriza por contener solamente atributos y por ende un estado.
- Interface de servicio: solo contiene métodos y opera con parámetros que han sido pasados a sus métodos.
- Interface con estado: La interface actúa de diferente manera según el estado en el que se encuentra, el orden de los métodos tiene gran importancia y generalmente existe una lista corta de parámetros.

[11] Ken Pugh. (2006). Interface Oriented Design. Texas: The pragmatic bookshelf.

- Interface sin estado: El comportamiento de la interface siempre es el mismo. El orden de los métodos no es de importancia y la lista de parámetros es larga.
- Interface cohesiva: Se considera que una interface es cohesiva si ésta provee servicios que giran alrededor de un mismo concepto. Los métodos relacionados con el mismo conocimiento deben ir juntos. Es completamente deseable que las interfaces a desarrollar sean de este tipo.
- Interface desacoplada: Una interface es desacoplada cuando sus métodos no dependen de la implementación de ninguna clase en particular completamente deseable que las interfaces a desarrollar sean de este tipo.
- Interfaces de almacenamiento: Son utilizadas para la persistencia de datos.
- Interfaces de entidades: Reflejan el modelo y reglas de negocio.

Una interface puede ser clasificada en más de un tipo sin que esto implique algún riesgo, es decir existe la posibilidad de tener una interface cohesiva y que ésta a su vez también pueda ser clasificada como una interface desacoplada, solo por poner un ejemplo.

5.2.2. MEDIDA DE INTERFACES.

Quizá la idea de medida de interfaces suene un poco confusa, pero se refiere a la forma en la que las interfaces se clasifican de acuerdo a sus características medibles.

- Interface mínima: Se refiere a una interface fácil de implementar y probar con pocos métodos. El usuario posee una cantidad limitada de métodos, pero suficiente.
- Interface completa: El usuario posee todos los métodos necesarios.
- Interface simple: Es fácil de implementar, pues sus métodos son sencillos, las variaciones a los métodos se deben implementar como nuevos métodos.
- Interface compleja: Los usuarios tiene la flexibilidad de hacer las cosas a su manera, pero con el paso del tiempo se vuelve difícil de entender para los usuarios.

5.3. ARQUITECTURA ORIENTADA A MICROSERVICIOS.

Los microservicios son una forma de arquitectura orientada a servicios, surgen como una alternativa a la arquitectura monolítica y son parte de la computación distribuida. Definen una configuración en la que el funcionamiento del sistema o aplicación se basa en la división de la misma en módulos o servicios independientes, cada servicio corresponde a una pequeña aplicación que se desarrollara por separado. Esto servicios se comunican entres si a través de una red y un protocolo de comunicación.

Entre las principales características se encuentran las siguientes:

- Generación de aplicaciones modulares y descentralizadas.
- Ideal para volúmenes altos en cuanto a transacciones o tráfico de datos.
- Generación de aplicaciones desacopladas y altamente cohesivas.
- Generación de aplicaciones independientemente escalables.

Este tipo de arquitectura se ha vuelto muy popular en los últimos años y compañías como Netflix, Ebay y Amazon, que se han caracterizado por manejar niveles de concurrencia y transaccionalidad altos han optado por su uso.

5.3.1. MODULOS O SERVICIOS.

La división de la aplicación en módulos se realiza teniendo en cuenta las funcionalidades del sistema a desarrollar, así como las capacidades del negocio, primeramente, se debe entender que cada módulo corresponde a una sola funcionalidad y que la ejecución de los mismos es totalmente independiente. Existe gran flexibilidad con respecto a la elección de las tecnologías a utilizar para su desarrollo y despliegue, pues éstas pueden variar de un módulo a otro.

5.3.2. COMUNICACIÓN.

La comunicación entre servicios se hace a través de APIs utilizando protocolos estándar. Es importante diferenciar entre la comunicación de servicios internos y externos, pues existe la posibilidad de utilizar diferentes tipos de protocolos, esto con el objetivo de mejorar el rendimiento. Un ejemplo podría ser hacer uso de HTTP para externos y TCP/UDP para internos.

5.3.3. MANEJO DE MENSAJERÍA.

El manejo de mensajes es un tema de gran importancia, pues existen muchos retos que se deben superar, primeramente, la utilización asíncrona y síncrona de mensajes es un tema muy amplio, ya que por una parte existen ventajas al trabajar con la forma asíncrona como la velocidad principalmente, pero también están presentes algunas desventajas como la incapacidad para controlar el orden de mensajes. Entre otros retos se encuentran el manejo de mensajes repetidos o corruptos y esto solo por mencionar algunos, es claro que se deben tomar en cuenta medidas para su utilización.

5.3.4. GENERALIDADES.

Cuando se trabaja con microservicios existen varias consideraciones que se deben tomar en cuenta:

- Diseñar para fallos: El diseño debe estar orientado a recuperarse de los estados de error que se presenten, por lo que poseer una muestra del estado correcto es de gran importancia para detectar errores mediante el monitoreo.
- Diseñar para Idenpotencia: El sistema debe ser capaz de realizar una determinada acción un número indefinido de veces y conseguir el mismo resultado como si se tratara de la primera vez, en los procesos para los cuales sea necesario.
- ID de solicitud: A cada solicitud se le debe asociar un id con el que ésta será identificada a lo largo de su proceso de vida, pues en muchas ocasiones una solicitud pasa a través de varios servicios por lo que se vuelve de interés identificarla.
- Asegurar compatibilidad: Los servicios deben ser compatibles con otros servicios.
- Estados de aplicación: Existen servicios en los cuales el estado es de gran importancia por lo que debe ser almacenado de alguna manera, la mejor práctica para este caso es colocar los datos hacia servicios de alta disponibilidad.

Como se puede notar la implementación de este tipo de arquitectura es todo un reto, principalmente por todos los factores que esto conlleva.

5.4. ARQUITECTURA ORIENTADA A SERVICIOS (SOA).

SOA es un enfoque nacido en los 90's para la construcción de sistemas de tecnologías de la información, el cual proporciona un amplio soporte para los requerimientos de negocio. Surge como solución a las deficiencias de los sistemas monolíticos, aportando gestión a grandes volúmenes de datos y mostrando gran flexibilidad al cambio, provocando manejabilidad en el negocio. La principal promesa de SOA es liberar al negocio de las restricciones de tecnología, es decir lograr que la tecnología se adapte al negocio y no el negocio a ésta. SOA no se define como metodología de tecnología o metodología de negocio, se considera una mezcla de ambas.

5.4.1. CARACTERISTICAS.

Entre las principales características que presentan los servicios y por las cuales han obtenido gran popularidad destacan las siguientes:

- Evita redundancia, uno de sus principales ideales, es generar código limpio y reutilizar el existente sin importar la forma en cómo se construyó.
- Está diseñado para la construcción de aplicaciones de negocio.
- Los componentes se encuentran desacoplados.
- Permite una orquestación de componentes para que estos a través del proceso de negocio entreguen un buen trabajo.
- Utiliza servicios como base de su arquitectura, los servicios pueden estar representados por web services, servicios REST e incluso otros componentes.

5.4.2. COMPONENTES.

La arquitectura orientada a servicios está integrada por una serie de componentes que trabajan en conjunto para un resultado en común, dichos componentes son descritos a continuación.

- Supervisor SOA: Es el encargado de asegurar que la plataforma trabaje bajo un ambiente SOA de una manera constante y predecible, el principal objetivo es crear un entorno donde todos los componentes trabajen en conjunto para asegurar un proceso de negocio exitoso.
- Enterprise Service Bus: La comunicación entre las piezas de software es muy importante, y es a través del service bus donde ésta ocurre. El Service Bus está formado por una colección de componentes de software que dan lugar a dicha comunicación, en la actualidad existe un gran número de compañías que ofrecen este servicio.
- SOA Registry: Se refiere a una especie de catálogo electrónico donde se almacena información acerca de los componentes, éste es útil para conocer: disponibilidad, reglas y descripción de los mismos. Actúa como un punto central de referencia, pues contiene los metadatos sobre los componentes, por lo que define el dominio de la arquitectura.
- Workflow engine: Es un componente de software diseñado para conectar el proceso de negocio de principio a fin, permite generar un set de instrucciones para asegurar el éxito.
- Service broker: Realiza las conexiones entre los componentes de trabajo, usa la información que encuentra en el Service registry.

5.4.3. CLASIFICACION DE SERVICIOS.

Según algunas de sus características principales los servicios pueden ser clasificados de la siguiente manera:

- Servicios de utilidad: Poseen funcionalidad multiproceso, no cubren solo una necesidad en específico, sino que participan en diversas, es por ello que poseen un alto potencial de reusabilidad.
- Servicios de entidad: Están centrados en el contexto de las entidades de negocio, entre las operaciones que poseen se encuentran las famosas CRUD (Create, Read, Update y Delete), son altamente reusables.
- Servicios de tarea: Engloban un proceso de negocio, consisten en una serie de pasos para completar una tarea específica, por lo cual no tienen un grado de reutilización alto. La anterior clasificación nos permite tener un mayor conocimiento de los servicios que se poseen en cuanto a su lógica y grade reutilización.

5.5. APLICACIÓN.

En este capítulo se pretende presentar la manera en que se estructurará el sistema con base en los diseños y arquitecturas mostrados anteriormente.

En cuanto al diseño, se propone trabajar con un diseño hibrido, al integrar el diseño orientado a entidades y el diseño orientado a interfaces, la decisión se toma por considerar que ambos diseños aportan gran valor, por un lado, el primero propone la identificación de elementos de gran importancia y un amplio estudio del dominio, mientras que el segundo agrega la implementación de interfaces permitiendo encapsular funcionalidades.

A continuación, se presenta la implementación referente a cada diseño y su integración conjunta.

Primeramente, el diseño orientado a entidades propone entender, asimilar y analizar de una manera exhaustiva el dominio del sistema. Atendiendo a ello se ha desarrollado un diagrama como el que se muestra en la Figura 5.1, pues además de corresponder al diagrama resultante del estudio del domino, este diagrama también representa al modelo de datos relacional que será empleado para implementar la base de datos del sistema.

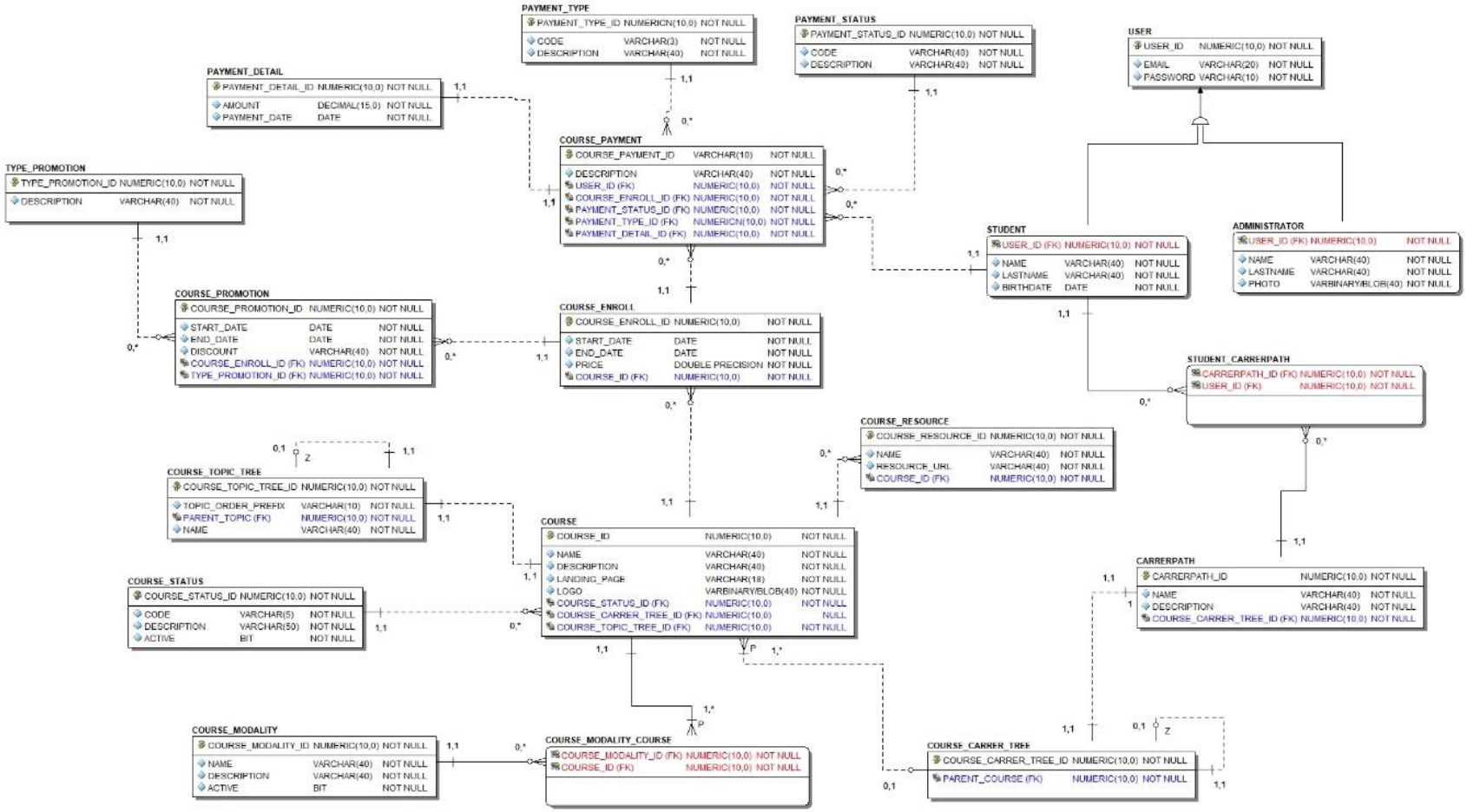

Figura 5. 1. Diagrama resultante del estudio del dominio.

Con base en el diagrama anterior se pretende comenzar a identificar ciertos elementos planteados por el diseño y así generar una propuesta del mismo, sin embargo, por tratarse de un sistema cuya complejidad es elevada resulta imposible mostrar la estructura completa del diseño por lo que se ha elegido mostrar el desarrollo alrededor de la entidad `course`, identificada a partir del diagrama mismo y como problemática se plantea la creación de un curso.

En la Figura 5.2 se puede apreciar la implementación del diseño orientado a entidades de manera general.

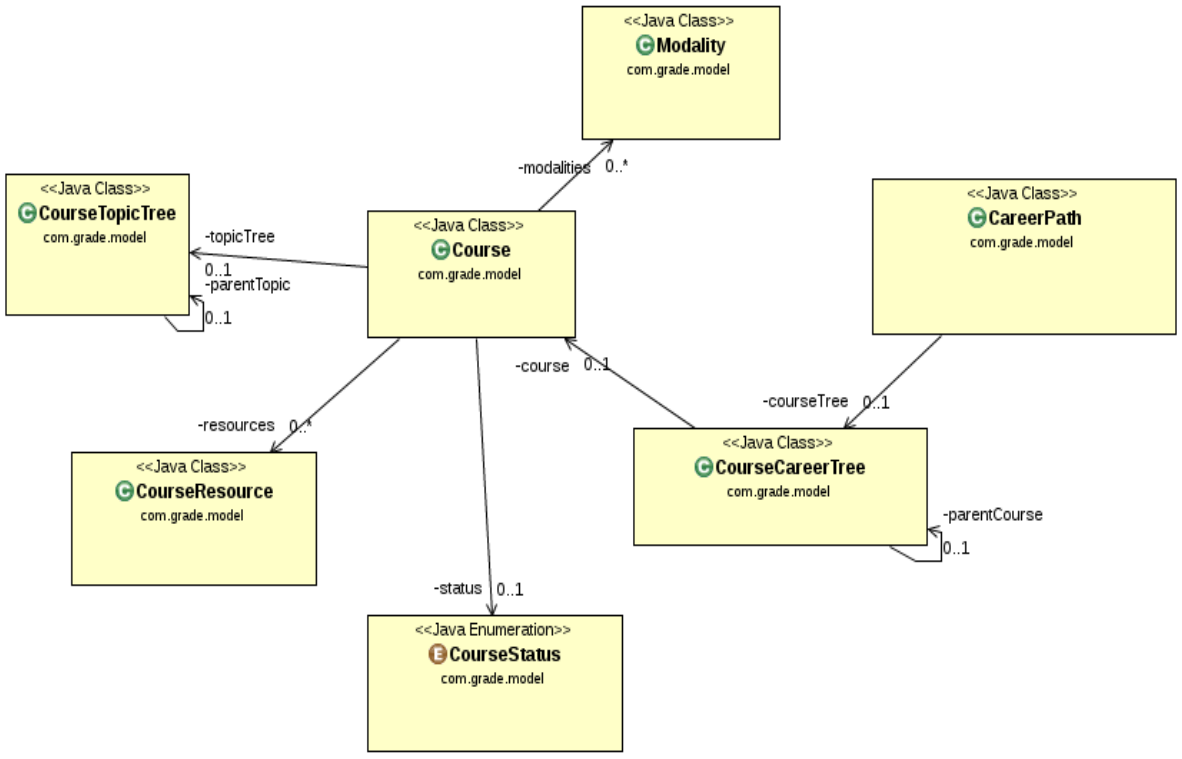

Figura 5. 2. Implementación del diseño orientado a entidades.

Como se comentó anteriormente se hará uso de un diseño hibrido, es por ello que se ha decidido mostrar la implementación del diseño orientado a interfaces con base en el diseño orientado a entidades (anteriormente planteado en la Figura 5.2).

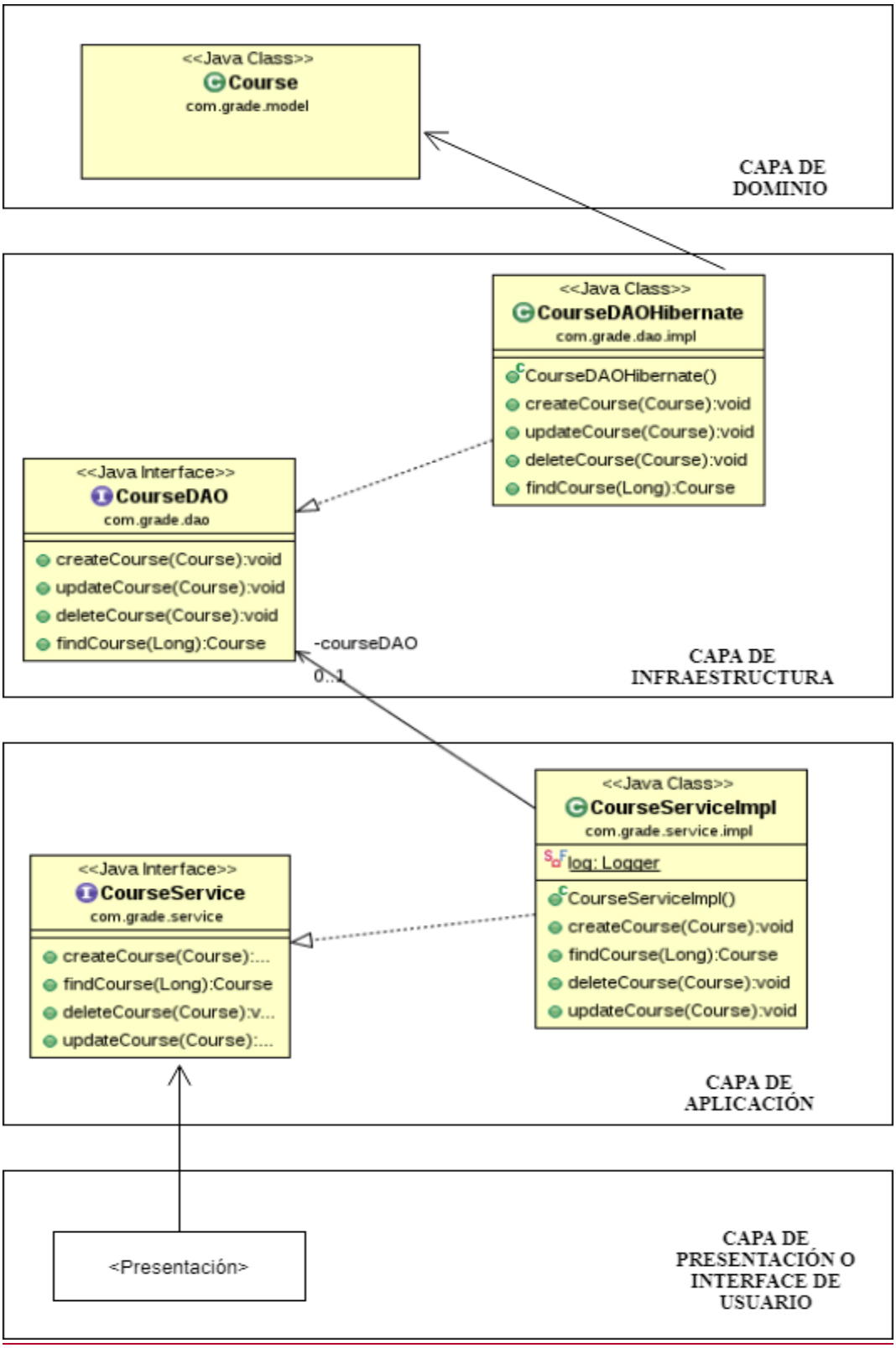

Figura 5. 3. Implementación de diseño hibrido (orientado a entidades y orientado a interfaces).

Como se puede observar en la Figura 5.3 se ha realizado la implementación de dos tipos de diseño de software, por un lado, se tienen las características de un diseño orientado a entidades mientras que se adquieren nuevas provenientes del diseño orientado a interfaces. Como resultado se obtiene un diseño completamente simple y altamente mantenible.

Finalmente se ha decidido fragmentar al sistema en módulos según la principal característica de la arquitectura orientada a microservicios, si bien es cierto ambas arquitecturas mostradas ofrecen fragmentar al sistema según sus funcionalidades en pequeños módulos llamados servicios, sin embargo, el enfoque resulta distinto, en la Tabla 5.1 se muestra un comparativo entre ambas arquitecturas.

DISEÑO ORIENTADO A MICROSERVICIOS	DISEÑO ORIENTADO A SERVICIOS
El uso compartido de componentes se da a través de un contexto limitado.	Trata de sacarle partido a todos los componentes que posee sin limitar el uso compartido de los mismos**.**
Los servicios que genera son de propósito único, provocando que sean muy especializados.	Los servicios que genera suelen ser versátiles en su mayoría dejando de lado la especialización.
La coordinación entre servicios es mínima.	La coordinación entre servicios es vital, pues cada servicio debe coordinar con muchos grupos.

Tabla 5. 1. Comparativa entre diseño orientado a microservicios y diseño orientado a servicios.

Como se puede observar la arquitectura orientada a servicios está pensada para sistemas muy complejos y con una transaccionalidad muy alta, por no ser el caso del sistema a desarrollar se ha elegido la utilización de los ya mencionados microservicios.

En la Figura 5.4 se muestra la división del dominio a través de pequeños módulos los cuales representan a los microservicios. La división se realizó a partir de la Figura 5.1 (anteriormente mostrada) con la identificación de subsistemas.

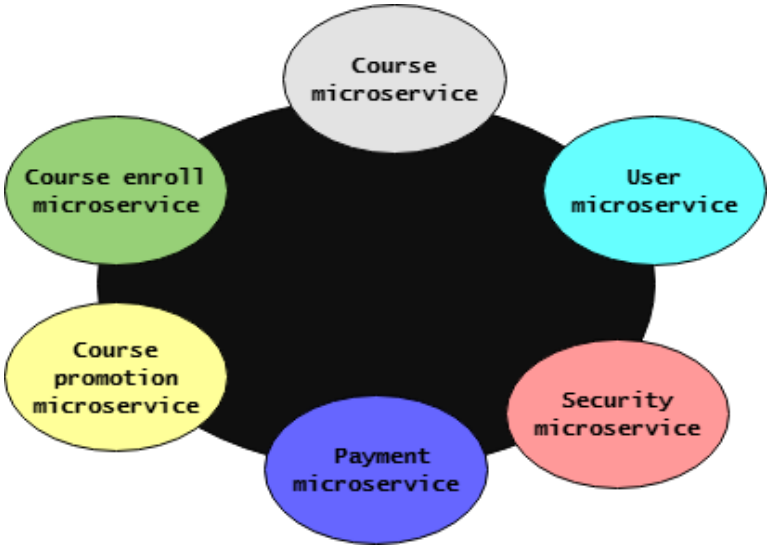

Figura 5. 4. División del dominio en microservicios.

A continuación, se presenta una breve descripción de las funcionalidades de cada microservicio mostrado en la Figura 5.4.

- `Course` lleva a cabo todas las operaciones relacionadas a un curso, desde su creación y actualización hasta su eliminación, entre otras funciones.
- `Course Enrroll` tiene por objetivo el dar de alta cursos asignando algunas características como un horario y lugar, además es el encargado de manejar la disponibilidad de los mismos.
- `Course Promotion` es el encargado de manejar la promoción de los cursos, está completamente enfocado en la publicidad de estos.
- `Payment` maneja la cobranza.
- `Security` es el encargado de manejar la seguridad de la aplicación en general.
- `User` maneja a los usuarios, la creación de los mismos y sus interacciones entre componentes de la aplicación.

6. DESARROLLO.

En este capítulo finalmente se presenta el desarrollo del sistema propuesto como solución a la problemática descrita, con base en el exhaustivo análisis previo y mediante las tecnologías anteriormente seleccionadas.

6.1. ESTRUCTURA GENERAL DEL SISTEMA.

Como se definió en capítulos anteriores para el desarrollo de este sistema se hará uso de la idea principal de la arquitectura orientada a microservicios (la fragmentación del sistema en pequeños módulos), además de la implementación de un diseño hibrido (orientado a entidades e interfaces) en conjunto con el modelo multicapa. En cuanto a las tecnologías a utilizar la elegida corresponde a Spring Framework sobre la plataforma Java.

Por tratarse de un sistema cuya complejidad es alta el desarrollo solo se centrará en el microservicio denominado Course microservice, a continuación, se presentan algunos Sprints referentes a dicho módulo.

6.1.1. DETALLE DEL DISEÑO.

El diseño orientado a entidades, con el cual se trabajará establece la implementación de cuatro capas (capa de dominio, capa de infraestructura, capa de aplicación y capa de interface de usuario o presentación), sin embargo, también se pretende implementar el modelo multicapa propuesto por Java EE por considerar que cuenta con una gran simplicidad y objetividad, dicho modelo propone tres capas (capa de acceso a datos, capa de negocio y capa de presentación). La Figura 6.1 ilustra tal situación.

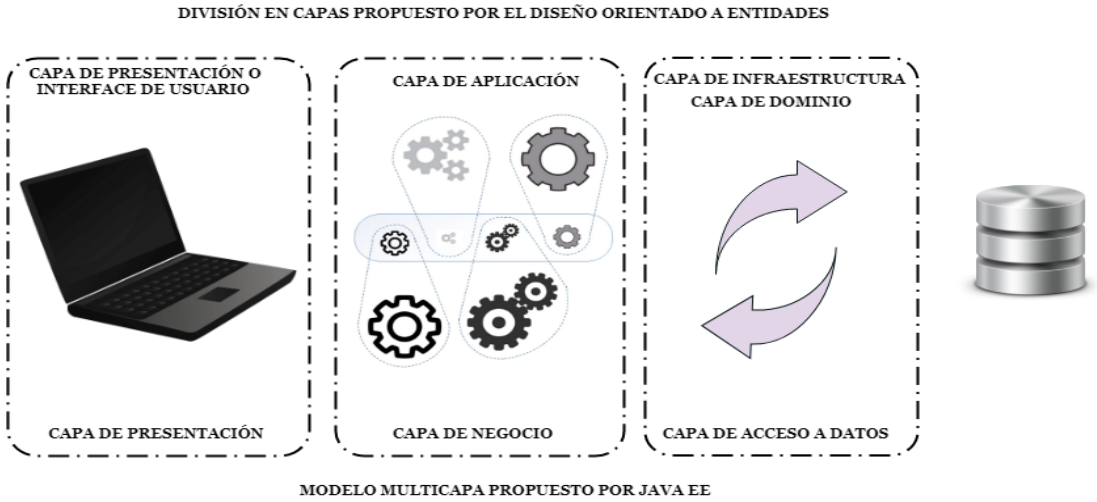

Figura 6. 1. Comparativa entre modelo multicapa propuesto por Java EE y modelo multicapa propuesto por el diseño orientado a entidades.

En la parte superior de la Figura 6.1 se aprecia la división en capas propuesta por el diseño orientado a entidades, mientras que en la parte inferior la propuesta del modelo multicapa de Java EE, se puede observar una división por recuadros donde la capa inferior realiza las mismas funciones que la capa o capas superiores. Como se aprecia ambas propuestas presentan grandes similitudes, por lo que trabajar con cualquiera de las dos no implicaría un cambio significativo, sin embargo, para este trabajo se hará referencia a la propuesta de modelo multicapa de java EE.

6.1.2. CAPA DE ACCESO A DATOS.

Como ya se habló anteriormente Spring posee soporte para patrones de diseño como lo son los objetos de acceso a datos (DAO), de los cuales se hará uso en este trabajo. Estos permiten ocultar la tecnología de acceso a datos al utilizar un objeto de mapeo relacional (ORM). El ORM seleccionado es Hibernate.

Entre las principales características que posee Hibernate destacan:

- Ofrece un mecanismo transparente para la persistencia.
- Es software libre, bajo los términos de las licencias GNU.
- Hace uso del llamado Hibernate Query Language (HQL).
- Busca solucionar el problema de la diferencia entre modelos de base de datos coexistentes en una aplicación.
- Posee un API para la construcción de consultas programáticamente llamada Criteria.
- Presenta flexibilidad en cuanto al esquema de tablas utilizado.
- Ofrece un paradigma 100% orientado a objetos.
- Es una implementación de Java Persistence API (JPA).
- Facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación.

En la Figura 6.2 se muestra la arquitectura de Hibernate, Hibernate hace uso de JDBC quien proporciona una capa de abstracción a la base de datos, también hace uso de Java Transaction API (JTA) Y Java Naming and Directory Interface (JNDI).

La arquitectura de Hibernate consiste principalmente en tres interfaces `org.hibernate.Session`, `org.hibernate.Transaction` y `org.hibernate.Query`. La primera es la encargada de proveer los métodos para la comunicación con la base de datos, la segunda establece métodos para ayudar a determinar las unidades atómicas de trabajo, finalmente la tercera proporciona métodos para la realización de consultas ya sea en HQL o SQL.

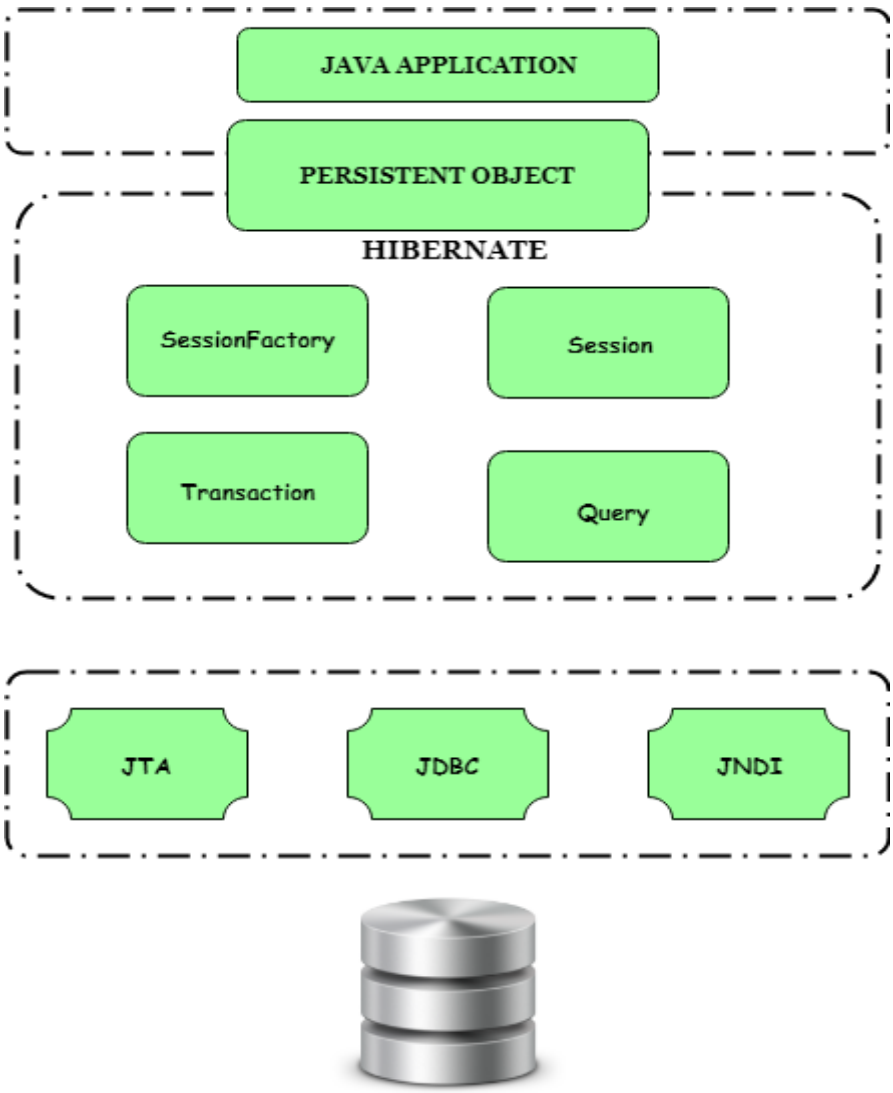

Figura 6. 2. Arquitectura de Hibernate.

6.1.3. CAPA DE NEGOCIO.

El detalle de la capa de negocio se centra en el negocio mismo, y dado a que no se hará uso de alguna tecnología no mencionada o recurso externo se deja la explicación de la misma para capítulos posteriores.

6.1.4. CAPA DE PRESENTACIÓN.

Para la implementación de esta capa se hará uso de Sprng mvc. Spring mvc corresponde a una alternativa del framework basada en el patrón de diseño front controller. Se ha elegido debido a la facilidad con que nos permite separar los componentes de la capa de presentación dependiendo la responsabilidad que estos poseen. Este patrón divide la capa

de presentación en tres capas o componentes (modelo, vista y controlador), y define como se comunican entre ellas.

El modelo representa los datos (una colección de objetos de dominio) que viajan a través de toda la aplicación o sistema, la vista hace referencia al despliegue de los mismos mientras que el controlador es el encargado de manejar las peticiones de usuario, es un intermediario entre el modelo y la vista.

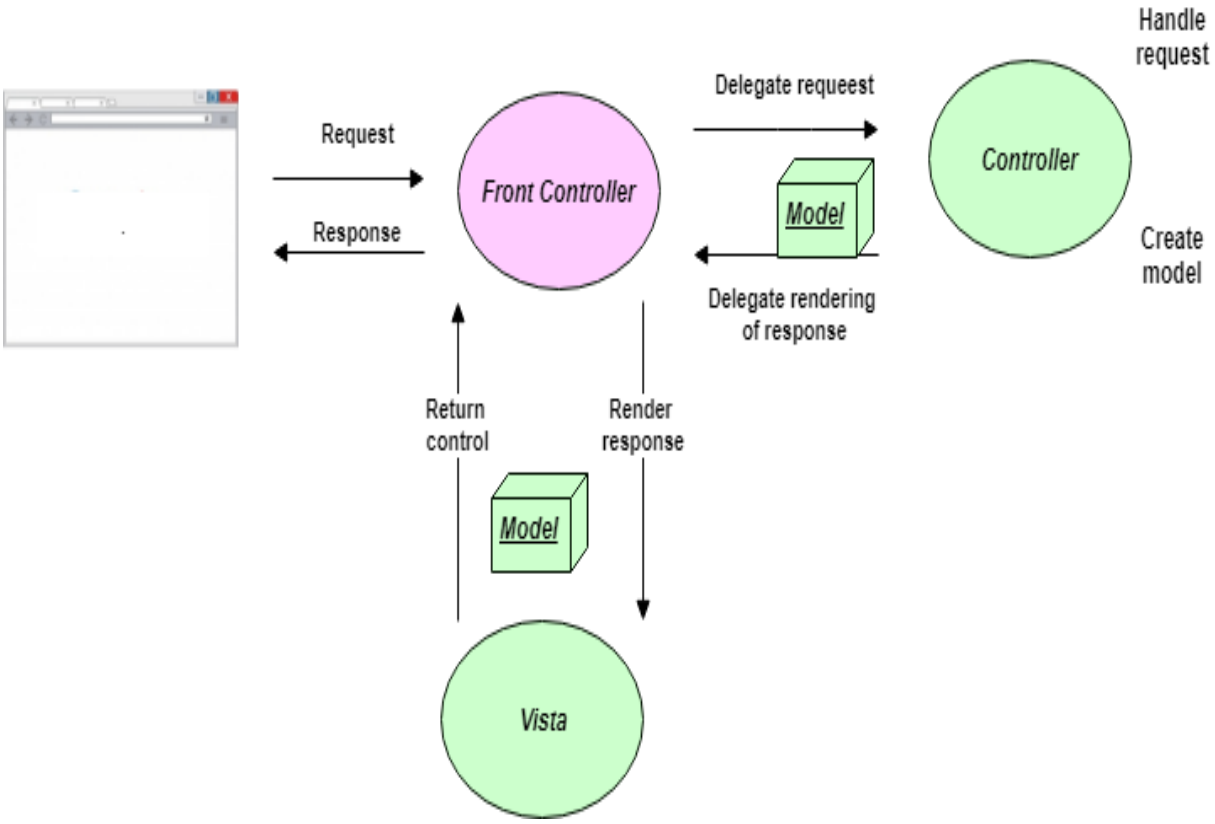

Figura 6. 3. Diagrama de procesamiento de petición http mediante Spring mvc.

En la Figura 6.3 se puede apreciar el procesamiento de una petición http mediante Spring mvc. Las peticiones http provenientes del navegador son canalizadas por el Front Controller, el cual representa un Servlet (en Spring mvc está implementado con la clase `DispatcherServlet`), su principal función es averiguar a que Controller hay que llamar (a partir de la URL ingresada) para responder a la petición, para lo anterior hace uso de la interface `HandlerMapping`.

Una vez que se ha identificado el Controller correspondiente éste realiza la lógica de negocio y devuelve los resultados encapsulados como el modelo además del nombre lógico de la vista a mostrar.

Para averiguar el nombre físico de la vista a mostrar se hace uso de la interface `ViewResolver`. Finalmente, `el DispatcherServlet` redirige la petición hacia la vista para mostrar los resultados solicitados.

6.2. PRODUCT BACKLOG

Como se vio en capítulos anteriores el Product Backlog es la lista de requerimientos o historias de usuario, corresponde a una de las primeras actividades a realizar, sin embargo, esta lista se puede actualizar a lo largo del desarrollo. Para su implementación se propone el uso de la plantilla mostrada en la Figura 6.4. A continuación se explicará el detalle de cada columna:

- Identificador (ID) de historia: Corresponde al código que identifica el requerimiento, es único, y una vez asignado no puede ser utilizado por ninguno otro, ni siquiera cuando el requerimiento sea descartado. El código identifica la historia no solo en el Product Backlog sino en cualquier otro documento.
- Enunciado de la historia: Corresponde a una descripción concisa del requerimiento.
- Estado: Identifica los posibles estados que cada historia pudiese presentar:
    - Vacío: Ocurre cuando la historia ha sido identificada pero aún no ha sido asignada a una iteración.
    - Planificada: Ocurre cuando la historia ha sido identificada y asignada a una iteración, pero aún no ha comenzado su ejecución.
    - En proceso: La historia se encuentra en ejecución.
    - Terminada: La historia fue desarrollada exitosamente.
    - Descartada: Ocurre cuando se determina que la historia ya no es relevante, dado a que su contenido se incluyó en otro grupo de historias o fue cancelada.
- Sprint (Iteración): Sprint o Iteración al que se asigna la historia.
- Dimensión o esfuerzo: Corresponde a la medición del esfuerzo que implica desarrollar la historia, existen muchas técnicas para realizar la medición del esfuerzo, pero en este trabajo se hará uso de los puntos de historia (Story Points). Los puntos de historia proponen realizar las mediciones con base en la historia más simple, dicha historia tendrá asignado 1 punto de historia y será la referencia, pues

cada vez que se pretenda estimar el esfuerzo de cualquier otra deberá ser comparada con ésta, por ejemplo, si cree que la historia a estimar nos llevara el doble de trabajo que la historia de referencia entonces se le deberán asignar 2 puntos de historia.

- Valor de negocio: Es un valor numérico (Bussines Value) asignado a cada historia donde a más alto valor, mayor valor para el cliente.

Prioridad: Según el marco de trabajo de Scrum, cada historia debe tener asignada cierta prioridad. Existen muchas y muy variadas técnicas para el cálculo de la prioridad, pero la que se utilizará para este trabajo es: Bussines Value & Story Points, consiste en obtener el cociente entre el valor de negocio y el esfuerzo (Bussines Value/ Story Points). Mientras mayor sea el resultado del cociente mayor grado de prioridad tendrá la historia.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad

Figura 6. 4. Plantilla para Product Backlog.

La implementación del Product Backlog del sistema a desarrollar se muestra en la Figura 6.5, el cual ha sido basado en los requerimientos funcionales y no funcionales mostrados en capítulos anteriores. Como se puede apreciar la columna que lleva por nombre Sprint(iteración) se encuentra vacía esto debido a que las historias han sido identificadas, pero no han sido asignadas a ninguna iteración, la asignación se llevará a cabo a través de los Sprint Plannings por lo que el Product Backlog se irá actualizando.

Como se comentó anteriormente para el cálculo del esfuerzo por historia debe existir una historia de referencia, en este caso la historia referencia corresponde a la identificada por el ID 00-00-00-02.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.			5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.			1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.			7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe tener adjuntar un archivo de descripción por curso.			2	4	2

Figura 6. 5. Product Backlog antes del primer Sprint.

6.3. SPRINT 1.

6.3.1. SPRINT PLANNING.

El Sprint que comienza tendrá una duración de cinco semanas y por tratarse de un equipo de desarrollo pequeño asignará una historia de usuario por Sprint. Para este Sprint se ha decidido asignar la historia de usuario identificada por el ID 00-00-00-01, la cual corresponde a la historia con mayor prioridad. También se decidió postergar la historia con ID 00-00-00-02 para el Sprint2. La tarjeta de historia de usuario para este Sprint se puede apreciar en el Figura 6.6.

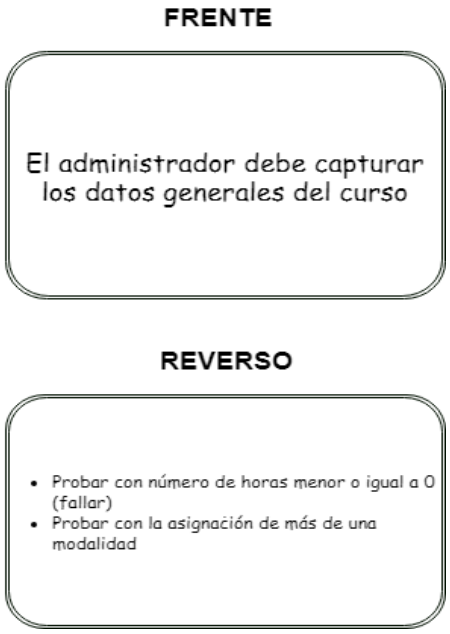

Figura 6. 6. Tarjeta de historia de usuario para Sprint 1.

Como conclusión del Sprint Planning se ha obtenido lo siguiente:

- El entregable del Sprint que comienza corresponde a la implementación de una pantalla, mediante la cual el administrador del sistema pueda realizar la captura de los datos generales del curso, la pantalla será amigable para el usuario y muy intuitiva. Los datos a capturar serán: nombre, descripción, número de horas, modalidades disponibles, estado, aprendizaje adquirido y sitio web.
- La forma en que se conseguirá un Sprint exitoso será invirtiendo 3 horas diarias de trabajo.

La Figura 6.7 muestra la actualización del Product Backlog después del Sprint Planning.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	En proceso	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Planificada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.			7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe tener adjuntar un archivo de descripción por curso.			2	4	2

Figura 6. 7. Product Backlog en Sprint 1 después de la reunión de Sprint Planning.

6.3.2. CONSTRUCCIÓN.

La demostración del desarrollo se centrará nuevamente en la división por capas ya antes planteada y se limitará a mostrar aspectos relevantes.

6.3.2.1. ACCESO A DATOS.

Para la capa de acceso a datos primeramente se realizó la abstracción del dominio mediante un modelo orientado a objetos, dicha abstracción es representada en la Figura 6.8 por un diagrama de clases. Se puede observar la identificación de tres entidades: `Course, CourseStatus y Modality.`

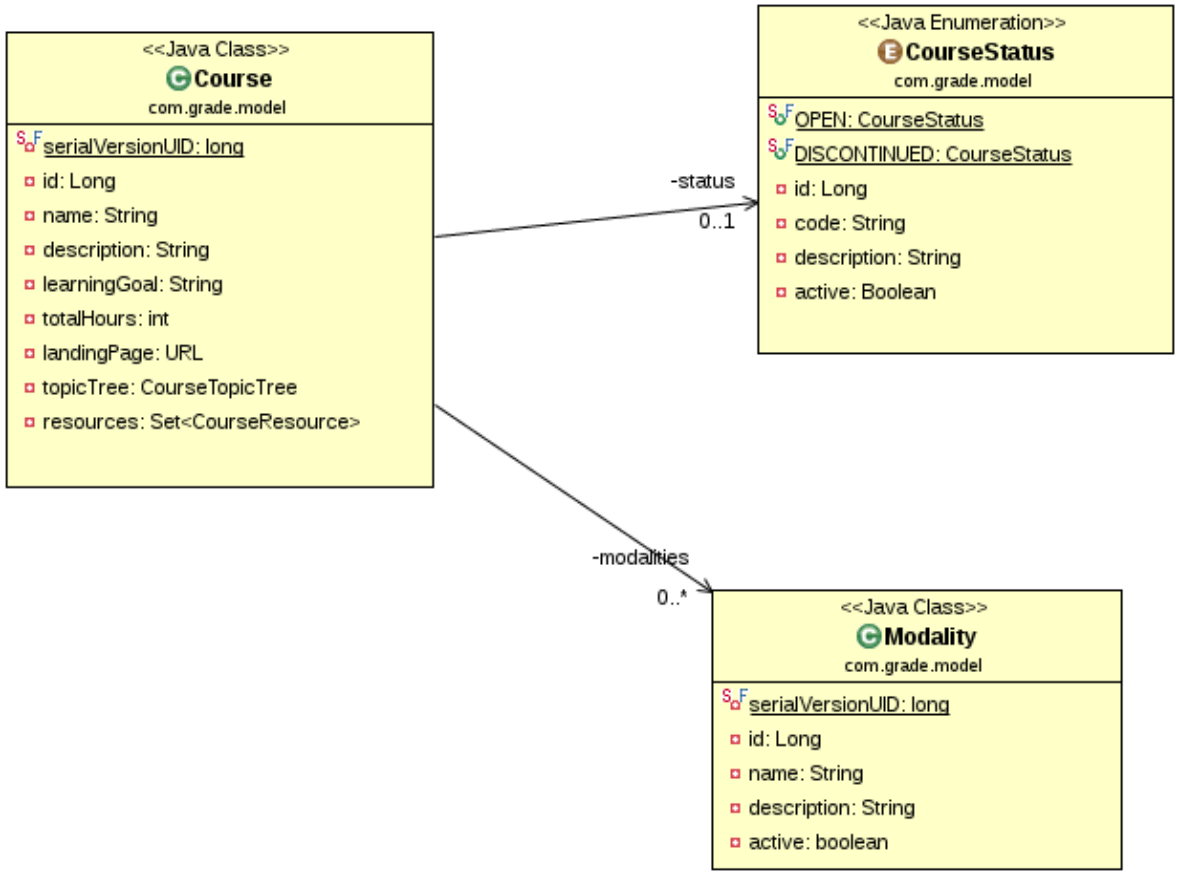

Figura 6. 8. Diagrama de Clases para Sprint 1 (identificación de tres entidades Course, CourseStatus y Modality).

Se considera de interés mostrar el diagrama entidad relación, el cual representa la estructura de tablas en la base de datos, dicho diagrama se ilustra en la Figura 6.9.

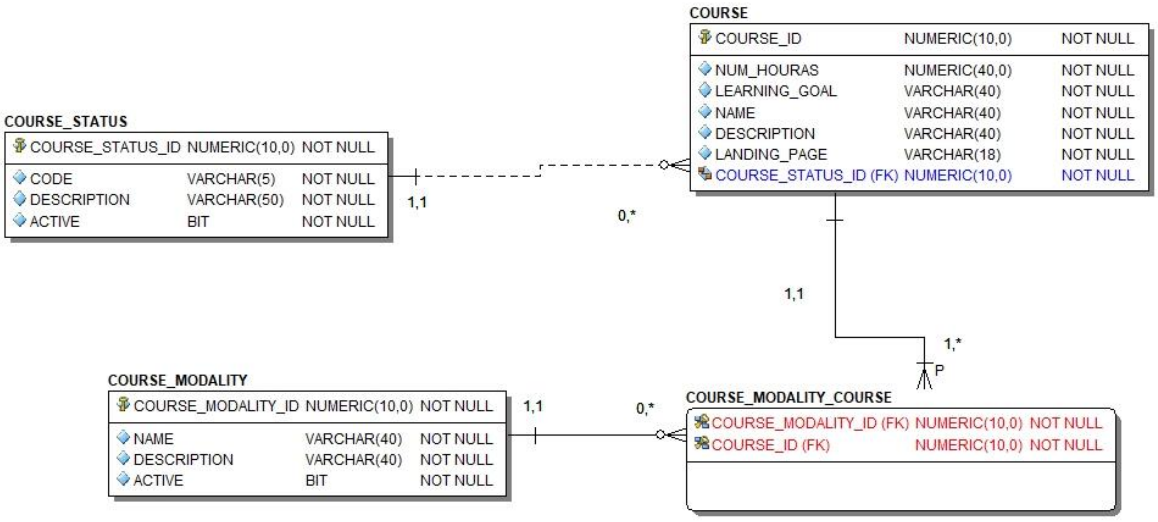

Figura 6. 9. Diagrama entidad relación correspondiente a Sprint 1.

Como ya se ha mencionado el objetivo de Hibernate es realizar un mapeo entre el modelo de objetos y el modelo relacional, Hibernate propone realizar dicho mapeo mediante anotaciones o archivos XML, para este trabajo se ha optado por realizar los mapeos mediante archivos XML. En la Figura 6.10 se puede observar un extracto del archivo de mapeo correspondiente a la entidad `Course`.

```
<hibernate-mapping package="com.grade.model">
    <class name="Course" table="course">
        <id name="id" column="course_id">
            <generator class="identity" />
        </id>
        <property name="name"  />
        <property name="description"/>
        <property name="landingPage" column="landing_page" />
        <property name="learningGoal" column="learning_goal" />
        <property name="totalHours" column="total_hours" />
        <property name="status" column="course_status_id">
            <type name="com.grade.dao.impl.CourseStatusType">
            </type>
        </property>
       <set name="modalities" table="COURSE_MODALITY" cascade="save-update" >
            <key column="course_id"/>
            <many-to-many column="modality_id" class="Modality" />
        </set>
    </class>
</hibernate-mapping>
```

Figura 6. 10. Archivo de mapeo para la entidad Course.

Hibernate cuenta con soporte para varios tipos de relaciones existentes entre entidades, por lo que se puede observar en las líneas 17-20 la implementación de un mapeo de tipo many to many entre la entidad `Modality` y la entidad `Course`.

Para el acceso a datos se implementó un DAO, en la Figura 6.11 se muestra su interfaz correspondiente.

```
5  /**
6   * <b>Usuario Email FechaDescripción</b>
7   * Abigail Ortega Valdivia abigailortega31@gmail.com
8   * Marzo 5, 2018 Creación
9   */
10 public interface CourseDAO {
11
12     /**
13      * @param course
14      *        used to create a new course
15      */
16     void createCourse(Course course);
17
18 }
19
```

Figura 6. 11. Interfaz Course DAO.

En la interfaz del DAO se deben declarar de los métodos necesarios para trabajar las acciones básicas con la base de datos (actualización, creación, borrado y búsqueda), conforme avancen los Sprints y sea necesario, se agregará la declaración de los métodos faltantes en la Figura 6.11, por ahora es suficiente.

6.3.2.2. NEGOCIO.

Se hace uso de una clase anotada con `@Service` (esto indica que se trata de un servicio) y su respectiva interface mediante la cual se llevará a cabo la lógica de negocio solicitada.

```
20 @Service
21 @Slf4j
22 public class CourseServiceImpl implements CourseService {
23
24     @Resource
25     private CourseDAO courseDAO;
26
27     /*
28      * (non-Javadoc)
29      * @see com.grade.service.CourseService#createCourse(com.grade.model.Course)
30      */
31     @Override
32     @Transactional
33     public void createCourse(Course course) {
34         log.debug("Creating new course: {}", course);
35         courseDAO.createCourse(course);
36
37     }
38
39 }
```

Figura 6. 12. Clase CourseServiceImpl.

La Figura 6.12 muestra la clase anotada con `@Service`, en las líneas número 24 y 25 se puede apreciar la ya muy mencionada inyección de dependencias, Spring simplifica dicha característica al hacer uso de la anotación `@Resource` sobre los objetos a "inyectar". Es notable la facilidad con la que se hace uso del objeto `courseDAO` tan solo por haber sido "inyectado" .

En cuanto al negocio, se puede apreciar en las líneas 32-39 la implementación del método `createCourse`, este método es responsable de implementación de la lógica de negocio, por tratarse de un módulo simple dicho método se limita a hacer uso del objeto `courseDAO`.

6.3.2.3. PRESENTACIÓN.

Con respecto a la capa de presentación se hizo uso de un controlador para el manejo de las peticiones http, el cual es mostrado en la Figura 6.13.

```
43  */
44 @Controller
45 @RequestMapping("/course")
46 @Slf4j
47 @SessionAttributes(names = { "course", "nodeTree" })
48 public class CourseGeneralDataController {
49
```

Figura 6. 13. Fragmento de controlador CourseGeneralDataController.

En la Figura 6.13 se puede apreciar el uso de dos anotaciones `@Controller` y `@RequestMapping` ambas provistas por Spring mvc, la primera es útil para establecer el tipo de componente al que corresponde la clase, mientras que la segunda es empleada para identificar a que controlador se tiene que direccionar cada llamada del cliente.

En la Figura 6.14 se puede observar un fragmento de la clase `CourseGeneralDataController`, dicha clase contiene el método `captureGeneralData` el cual es encargado de preparar la pantalla que el usuario ha solicitado. En la línea número 54 se puede observar la declaración de un elemento de tipo final correspondiente al nombre lógico de la pantalla a mostrar. Mientras que en la línea número 92 se puede notar como ese valor es retornado como respuesta a la petición.

```
48  public class CourseGeneralDataController {
49
50      @Resource
51      private CourseModalityService courseModalityService;
52      @Resource
53      private CourseValidator courseValidator;
54      private static final String viewCaptureGeneralData = "course/captureGeneralData";
56       * @param binder
58      @InitBinder(value = { "course" })
59      public void initBinderLibro(WebDataBinder binder) {
60          binder.setValidator(courseValidator);
61      }
63       * This method is invoked by Spring to initialize the {@link Course} object to be used in
68      @ModelAttribute("course")
69      public Course setupForm() {
70          Course course = new Course();
71          return course;
72      }
74       * Starting point of the Course creation.
80      @RequestMapping("captureGeneralData")
81      public String captureGeneralData(@ModelAttribute Course course, ModelMap model) {
82          List<Modality> modalityList;
83
84          modalityList = setupModalities(model);
85          // adding modalities to the course object with id = null to avoid initial selection
86          if (course.getModalities() == null) {
87              course.setModalities(new HashSet<>(modalityList.size()));
88              modalityList.forEach(modality -> course.getModalities()
89                      .add(new Modality(null, modality.getName(), modality.isActive())));
90          }
91
92          return viewCaptureGeneralData;
93      }
```

Figura 6. 14. Controlador CourseGeneralDataController.

La construcción de la pantalla solicitada se desarrolló mediante la implementación de Sitemesh, JQuery y JSP. Dicha pantalla se puede apreciar en la Figura 6.15.

La decisión de elegir las herramientas antes mencionadas tiene que ver con las grandes ventajas que se obtienen con su implementación, primeramente, Sitemesh permite de manera sencilla y nada intrusiva separar de forma simple y efectiva el contenido html, mediante la utilización del patrón de diseño decorator. El empleo de JQuery proporciona efectos modernos, animaciones de calidad y soporte para la mayoría de los navegadores, finalmente la utilización de JSP provee un sin fin de beneficios entre los que destacan el soporte que posee Spring mvc mediante su librería de etiquetas, su capacidad multiplataforma, robustez y reusabilidad.

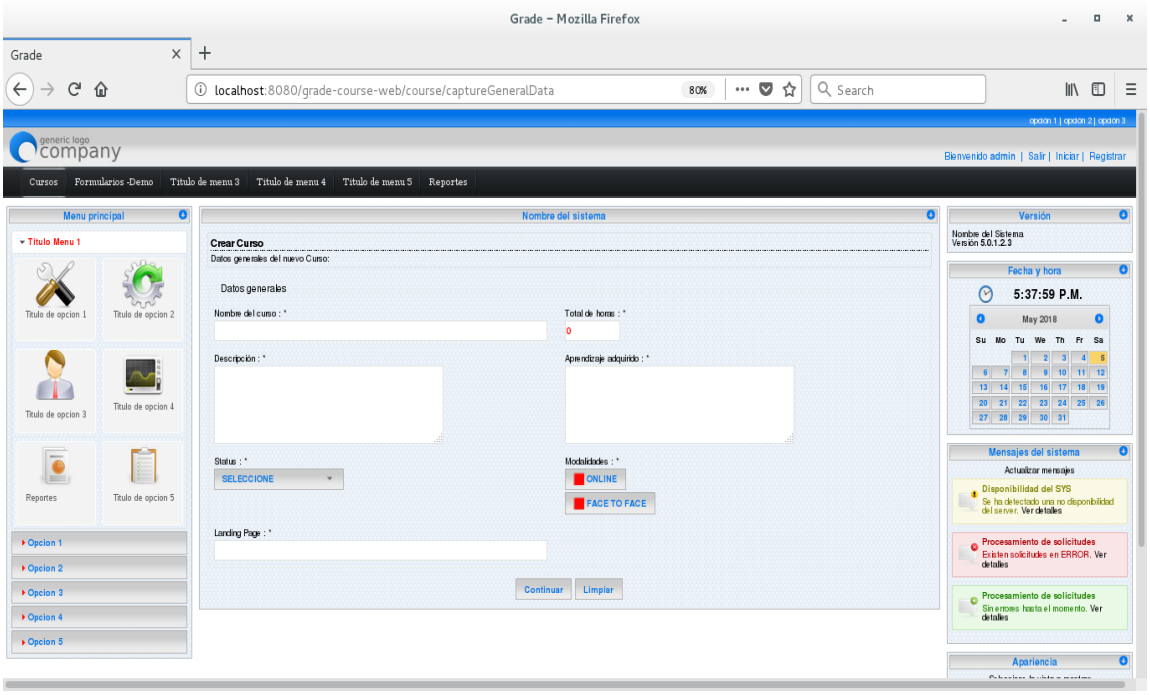

Figura 6. 15. Entregable del Sprint 1.

La pantalla mostrada en la Figura 6.15 (correspondiente al entregable de este Sprint) presenta un funcionamiento bastante simple. El administrador deberá llenar los campos mostrados y pulsar el botón continuar, dicho botón generará una petición Http hacia otro método del controlador, el cual se encargará mediante el uso del servicio (componente previamente inyectado y mostrado en la Figura 6.16) de llevar a cabo la lógica de negocio ya anteriormente explicada.

```
@Resource
private CourseService courseService;
```

Figura 6. 16. Inyección de componente courseService.

Las pruebas pertinentes se presentarán en el capítulo número 7. Hasta este punto se ha concluido la realización e implementación de la historia de usuario con ID 00-00-00-01.

6.3.3. SPRINT RETROSPECTIVE.

En el Sprint Retrospective de este Sprint se habló de los aspectos que contribuyeron al éxito de la iteración, entre los que destacaron se encuentran: la constancia que el equipo presento para trabajar, el aprendizaje continuo y el manejo de objetivos diarios. Se debe procurar que estos aspectos estén presentes a lo largo de los Sprints, ya que trajeron importantes beneficios al Sprint actual.

El equipo también presento algunos aspectos a mejorar, sin embargo, el que se considera más importante conlleva cuidar el orden de desarrollo. El desorden debe ser evitado y erradicado en los siguientes Sprints.

En la Figura 6.17 se muestra la actualización del Product Backlog.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Planificada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.			7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe tener adjuntar un archivo de descripción por curso.			2	4	2

Figura 6. 17. Product Backlog después del Sprint 1.

6.4. SPRINT 2.

6.4.1. SPRINT PLANNING.

El Sprint que comienza tendrá una duración de una semana y se ha decidido asignar la historia de usuario identificada por el ID 00-00-00-02. La tarjeta de historia de usuario para este Sprint se puede apreciar en el Figura 6.18.

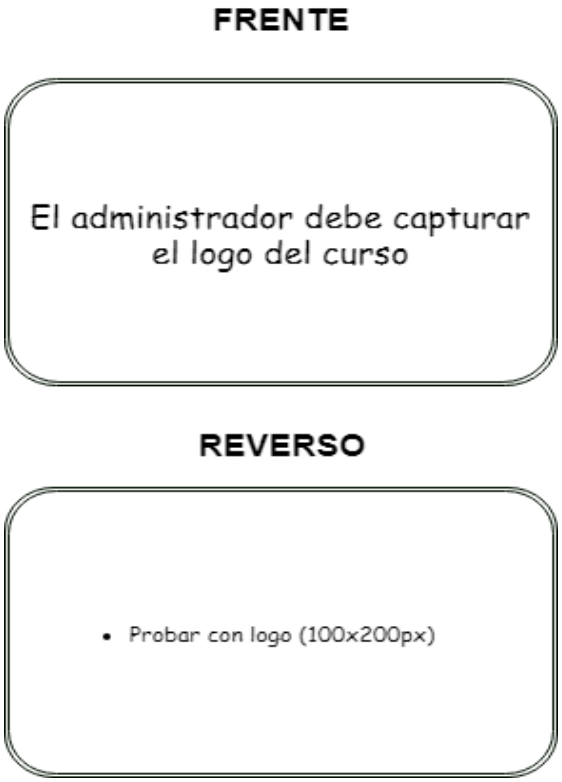

Figura 6. 18. Tarjeta de historia de usuario para Sprint 2.

Como conclusión del Sprint Planning se ha obtenido lo siguiente:

- El entregable del Sprint que comienza, corresponde al incremento de la pantalla mostrada en la Figura 6.15, mediante la cual el administrador del sistema podrá realizar la carga del logo del curso. En esta reunión también se tomó la decisión de cancelar la actividad correspondiente al ID 00-00-00-07 por considerar que ha perdido sentido e importancia para el cliente.
- La forma en que se conseguirá un Sprint exitoso será invirtiendo 3 horas diarias de trabajo.

La Figura 6.19 muestra la actualización del Product Backlog después del Sprint Planning.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	En proceso	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.			7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 19. Product Backlog después del Sprint Planning del Sprint 2.

6.4.2. CONSTRUCCIÓN.

La demostración del desarrollo se centrará nuevamente en la división por capas ya antes planteada, con la excepción de la capa de negocio, por la simplicidad de la historia, no existe ningún tipo de modificación a dicha capa, como consecuencia será omitida.

6.4.2.1. ACCESO A DATOS.

La abstracción del dominio básicamente corresponde a la abstracción del sprint pasado, pues el modelo sufrió un muy ligero cambio, se agregó un atributo a la entidad Course para alojar al logo.

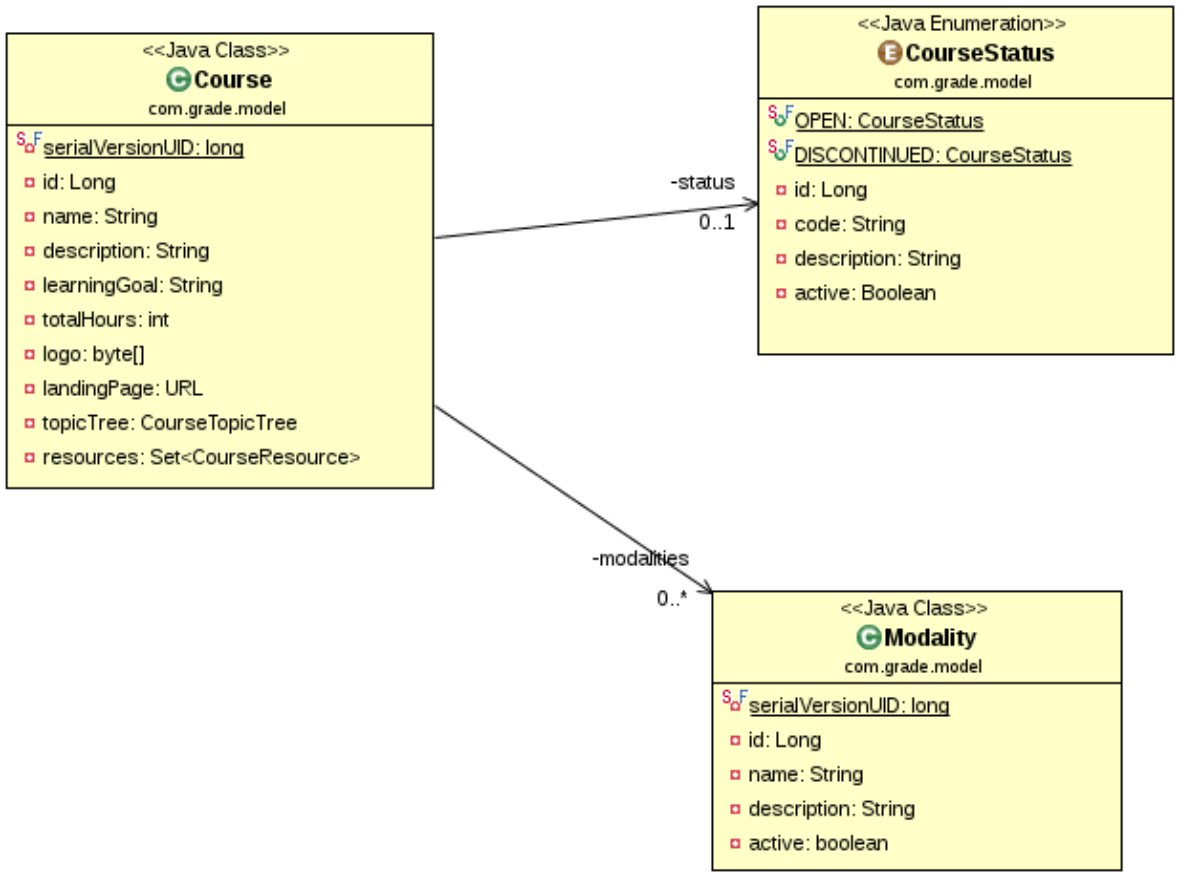

Figura 6. 20. Diagrama de clases para entidades Course, Modality y CourseStatus.

En la Figura 6.20 y 6.21 se puede observar el diagrama de clases y el diagrama entidad-relación correspondientes.

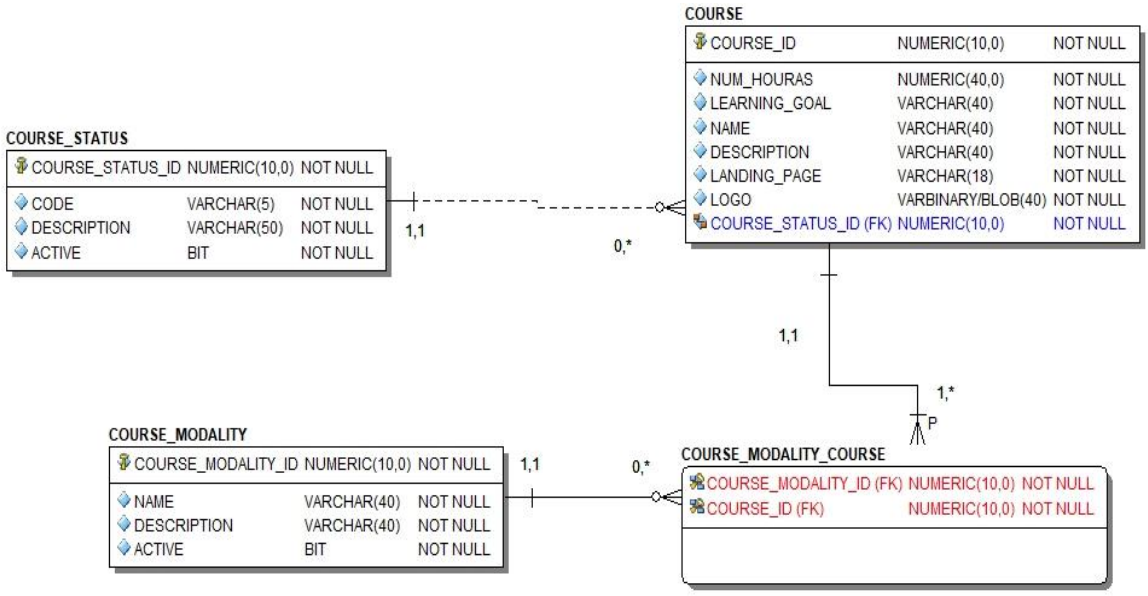

Figura 6. 21. Diagrama entidad relación para Sprint 2.

6.4.2.2. PRESENTACIÓN.

En la capa de presentación se realizaron un par de cambios para poder satisfacer la historia de usuario, a continuación, se describen.

```
58      <ul class="two">
59         <li>
60            <label>Landing Page : *</label>
61            <form:errors cssClass="validation-error" path="landingPage"/>
62            <form:input path="landingPage" class="large" />
63         </li>
64         <li>
65         <label>Logo (100*200 px) : *</label>
66               <form:errors cssClass="validation-error" name="file"/>
67               <input type="file" name="file" class="large">
68         </li>
69      </ul>
70   </fieldset>
71   <br>
72   <div align="center">
73      <input type="submit" value="Continuar">
74      <input type="reset" value="Limpiar">
75   </div>
76 </form:form>
```

Figura 6. 22. Fragmento de archivo JSP.

En la Figura 6.22 líneas 64-68 se puede observar el código necesario para la carga del logo en la pantalla. Es importante mencionar que el logo será cargado como un objeto de tipo `MultipartFile` y el tamaño máximo permitido se puede establecer en el archivo de configuración llamado `webAppContext.xml`

```
50    <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
51        <property name="maxUploadSize" value="100000" />
52    </bean>
```

Figura 6. 23. Fragmento de archivo webAppContext.xml.

En la Figura 6.23 se puede observar el valor máximo establecido para el logo, dicho valor se encuentra en bytes. De esta manera se satisface la condición establecida en la historia de usuario.

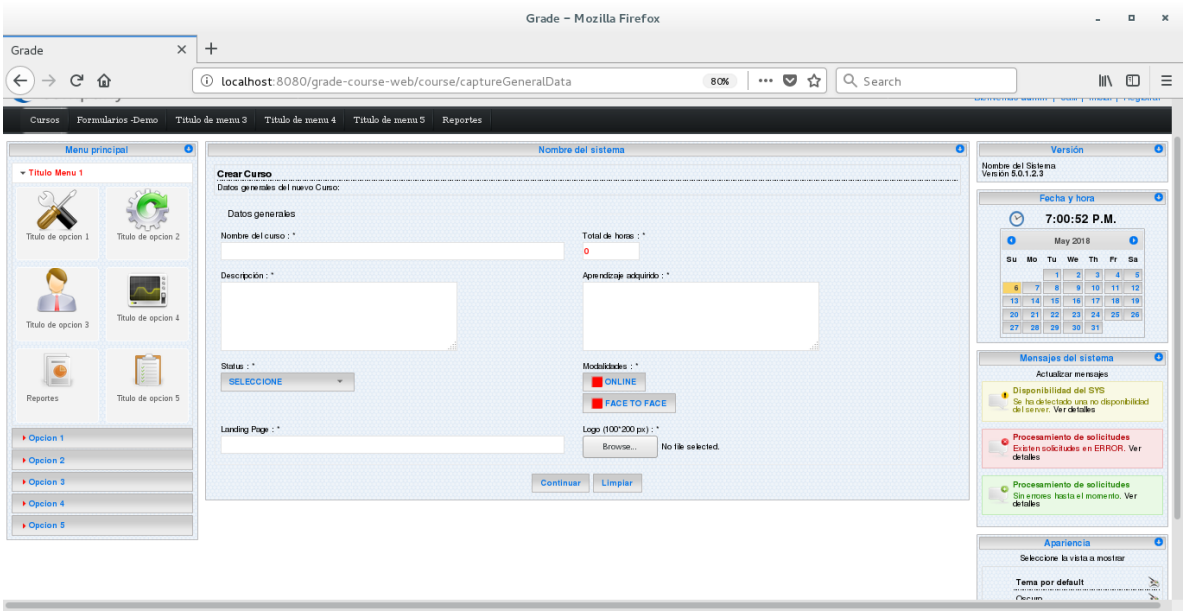

Figura 6. 24. Entregable del Sprint 2.

La Figura 6.24 corresponde al entregable de este Sprint, la pantalla mostrada en la Figura 6.15 con el incremento solicitado. Las pruebas pertinentes serán mostradas en el capítulo 7.

6.4.3. SPRINT RETROSPECTIVE.

En el Sprint Retrospective de esta iteración se habló del aspecto a mejorar relacionado con la falta de orden, encontrando que se consiguió erradicarlo por completo. Sin embargo, existieron algunos otros aspectos a destacar que permitieron la realización exitosa del Sprint como lo fueron: la proactividad del equipo además de la alta responsabilidad que ha mostrado.

En cuanto a aspectos negativos no existe nada por mencionar.

En la Figura 6.25 se muestra la actualización de la plantilla de Product Backlog.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.			7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 25. Product Backlog después del Sprint 2.

6.5. SPRINT 3.

6.5.1. SPRINT PLANNING.

Este Sprint tendrá una duración de 7 semanas y la historia de usuario asignada corresponde al ID 00-00-00-03. En la Figura 6.26 se muestra la tarjeta de historia de usuario correspondiente.

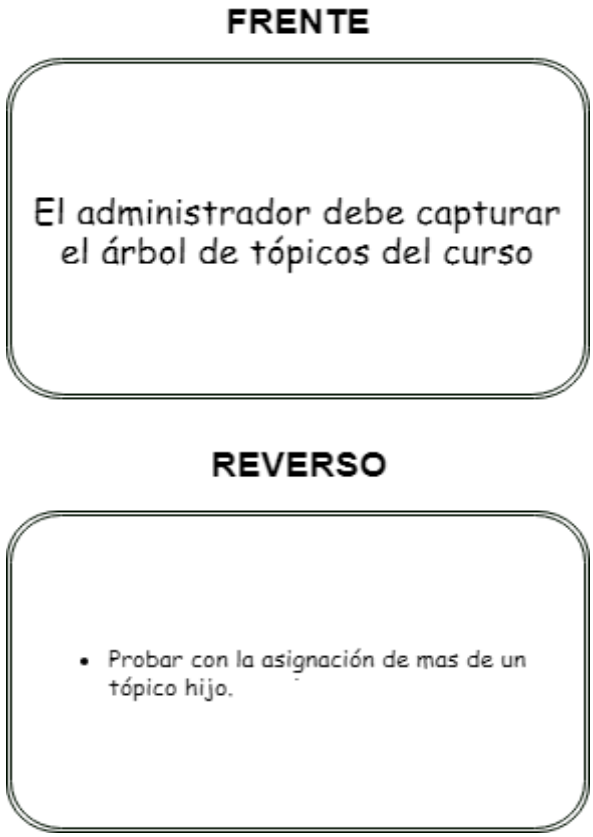

Figura 6. 26. Tarjeta de historia de usuario para Sprint 3.

Como conclusión del Sprint Planning se ha obtenido lo siguiente:

- El entregable del Sprint que comienza corresponde a la implementación de una pantalla, mediante la cual el administrador del sistema pueda realizar la captura del árbol de tópicos del curso, la pantalla será amigable para el usuario y muy intuitiva. Los datos a capturar serán: nombre, orden de tópico y tópico padre.
- La forma en que se conseguirá un Sprint exitoso será invirtiendo 3 horas diarias de trabajo.

En la Figura 6.27 se muestra la actualización del Product Backlog.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	En Proceso	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 27. Product Backlog después de Sprint Planning para Sprint 3.

6.5.2. CONSTRUCCIÓN.

6.5.2.1. ACCESO A DATOS.

Con base en el estudio del dominio se ha conseguido identificar una nueva entidad (`CourseTopicTree`), lo anterior se ilustra en la Figura 6.28 mediante un diagrama de clases.

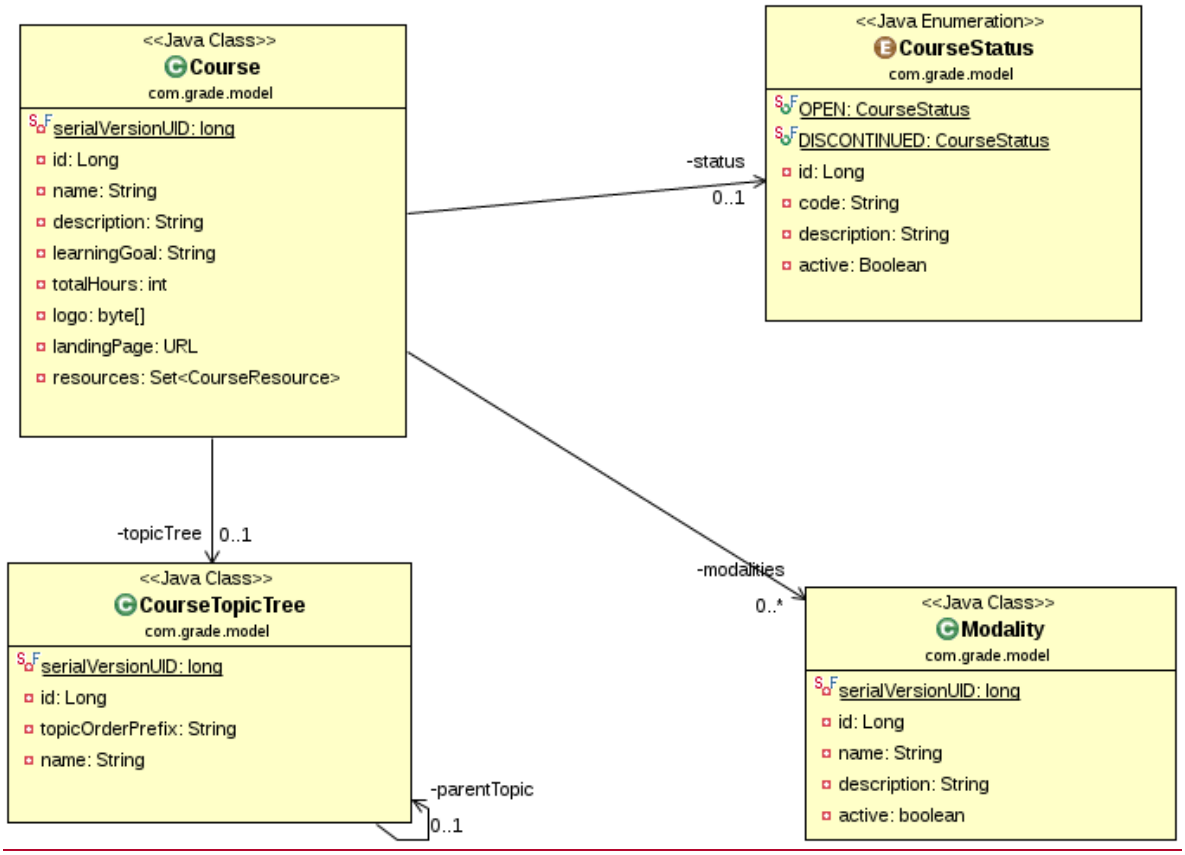

Figura 6. 28. Diagrama de clases con la identificación de la nueva entidad CourseTopicTree.

Se considera poco relevante ilustrar el diagrama entidad relación por lo que ha sido omitido.

La Figura 6.28 muestra la entidad `CourseTopicTree` la cual presenta dos relaciones de tipo composición, una recursiva y otro directa con la entidad `Course`, este tipo de relaciones ocurren cuando el tiempo de vida de un objeto está condicionado al tiempo de vida del que lo incluye, por ejemplo, un objeto de tipo `CourseTopicTree` pierde sentido al no estar relacionado con un objeto de tipo Course.

6.5.2.2. NEGOCIO.

En la Figura 6.29 se puede apreciar la clase `CourseTopicTreeServiceImpl` anotada con `@Service` esta clase es la encargada de llevar a cabo la lógica de negocio a través del método `createCourseTopicTree`. Dicho método hace uso de la clase `NodeTree` mostrada en la Figura 6.30.

```
24 @Service
25 @Slf4j
26 public class CourseTopicTreeServiceImpl implements CourseTopicTreeService {
27
28     private List<CourseTopicTree> nodos = new ArrayList<CourseTopicTree>();
29     @Resource
30     private CourseTopicTreeDAO courseTopicTreeDAO;
31     /*
32      * (non-Javadoc)
33      * @see com.grade.service.CourseTopicTreeService#createTopicAndChildren(com.grade
34      * .model.CourseTopicTree)
35      */
36     @Override
37     @Transactional
38     public void createCourseTopicTree(NodeTree<CourseTopicTree> rootNode) {
39
40         CourseTopicTree rootTopic;
41         List<NodeTree<CourseTopicTree>> children;
42
43         rootTopic = rootNode.getData();
44
45         log.debug("Creating topic: {}", rootNode.getData());
46         courseTopicTreeDAO.createTopic(rootTopic);
47         children = rootNode.getChildren();
48         children.forEach(child -> {
49             child.getData().setParentTopic(rootTopic);
50             if (child.getChildren().size() > 0) {
51                 createCourseTopicTree(child);
52             } else {
53                 log.debug("Creating topic: {}", child.getData());
54                 courseTopicTreeDAO.createTopic(child.getData());
55             }
56         });
```

Figura 6. 29. Servicio CourseTopicTreeServiceImp.

```
18 public class NodeTree<T> {
19
20     /**
21      * @param node
22      */
23     public NodeTree(T node) {
24         this.data = node;
25     }
26
27     @Getter
28     private T data;
29
30     @Getter
31     private List<NodeTree<T>> children = new ArrayList<>(2);
32
33     @Getter
34     private NodeTree<T> parent;
35
36     /**
37      * @param child
38      * @return
39      */
40     public NodeTree<T> addChild(T child) {
41         NodeTree<T> node;
42         node = new NodeTree<T>(child);
43         node.parent = this;
44         children.add(node);
45         return node;
46     }
47
```

Figura 6. 30. Clase NodeTree<T>.

En la Figura 6.30 se puede observar la clase `NodeTree` la cual corresponde a una clase genérica, cuenta con tres atributos:

- Data: Almacena un objeto.
- Children: Almacena una lista de `NodeTree` del mismo tipo de objeto que el atributo data.
- Parent: Almacena un objeto de tipo `NodeTree` del mismo tipo de objeto que el atributo data.

En la Figura 6.30 líneas 48-56 se puede observar la manera de recorrer la estructura antes explicada, con el objetivo de realizar la persistencia de los tópicos pertenecientes al árbol a través del DAO (`CourseTopicTreeDAO`).

6.5.2.3. PRESENTACION.

En la Figura 6.31 se puede observar el controlador encargado de responder a la petición del usuario.

```
37 public class CourseTopicTreeController {
38
39     @Resource
40     private TopicValidator topicValidator;
41
43      * @param binder
46     public void initBinderLibro(WebDataBinder binder) {
50      * @return
53     public CourseTopicTree setupForm() {
57      * @param courseTopicTree
63
64     @RequestMapping("addTopic")
65     public String addTopic(@Valid CourseTopicTree courseTopicTree, BindingResult result,
66             @SessionAttribute NodeTree<CourseTopicTree> nodeTree, ModelMap model) {
67
68         List<CourseTopicTree> topicList;
69
70         if (result.hasErrors()) {
71             topicList = new ArrayList<>(5);
72             buildTopicsAsList(nodeTree, topicList);
73             model.addAttribute(topicList);
74             return viewCaptureTopics;
75         }
76         addNodeTree(nodeTree, courseTopicTree);
77
78         // update the list of topics
79         topicList = new ArrayList<>(5);
80         buildTopicsAsList(nodeTree, topicList);
81         model.addAttribute(topicList);
82
83         return viewCaptureTopics;
84     }
85
```

Figura 6. 31. Controlador CourseTopicTreeController.

La estructura que almacenará los tópicos del curso se puede apreciar en la línea número 66, dicha estructura lleva por nombre `nodeTree` y como lo indica la anotación `@SessionAttribute` corresponde a un objeto de sesión.

En la línea número 65 se puede observar el uso de la anotación `@Valid`, Spring implementa el uso de validadores, esto permite validar los datos introducidos en el JSP antes de llegar al controlador.

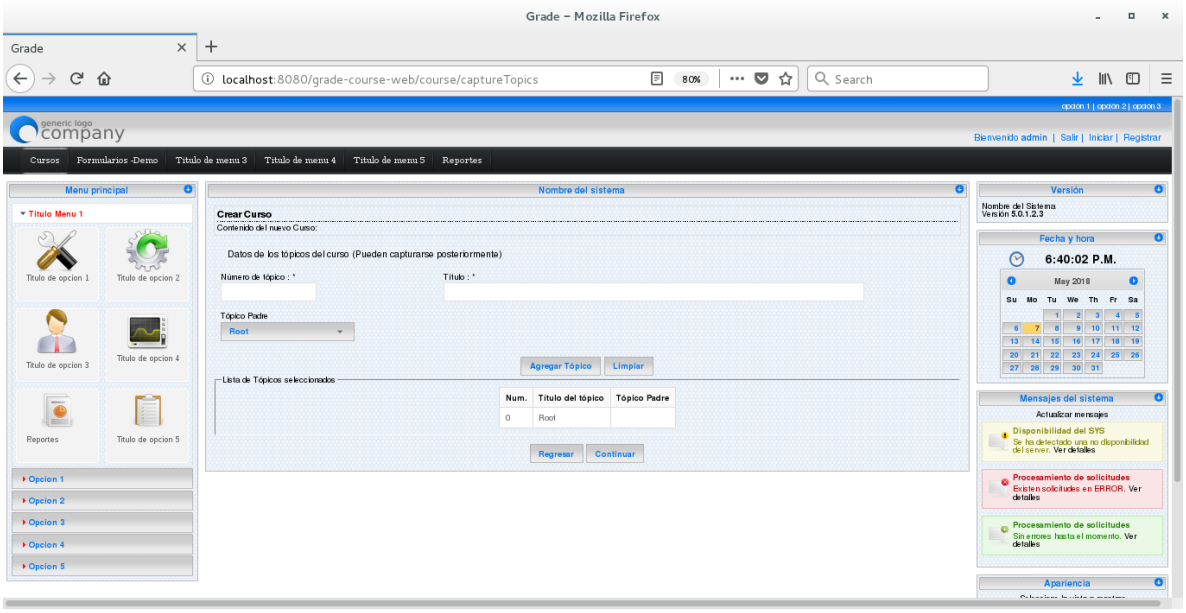

Figura 6. 32. Entregable deL Sprint 3.

La Figura 6.32 muestra la pantalla que corresponde al entregable de este Sprint.

6.5.2.4. SPRINT RETROSPECTIVE.

En el Sprint Retrospective de esta iteración se habló de una problemática detectada en el equipo y que se considera un foco rojo. El equipo está perdiendo enfoque (lo cual se traduce en permitir posibles distracciones), lo anterior es importante dado a que esto podría generar retrasos en la ejecución del Sprint y comprometer la calidad. Sin embargo, como parte del Sprint Restrospective se presentaron posibles soluciones y una serie de consejos sobre cómo mitigar tal situación.

En cuanto a los aspectos positivos que contribuyeron a la realización exitosa del Sprint destaca la actitud positiva que manejo el equipo ante los retos que se presentaron.

En la Figura 6.33 se muestra la actualización de la plantilla de Product Backlog

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	Terminada	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.			4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.			4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 33. Product Backlog después del Sprint 3.

6.6. SPRINT 4.

6.6.1. SPRINT PLANNING.

La duración de este Sprint fue establecida a 4 semanas y la historia de usuario asignada corresponde a la de ID 00-00-00-05. Es interesante notar que en este momento existen 2 historias de usuario con la misma prioridad (ID 00-00-00-05 y ID 00-00-00-04) por lo que la elección de cualquiera de ellas no implicaría un cambio relevante en el negocio. Se ha decidido planificar la historia de usuario con ID 00-00-00-04 para el Sprint 5. En la Figura 6.34 se muestra la tarjeta de historia de usuario correspondiente.

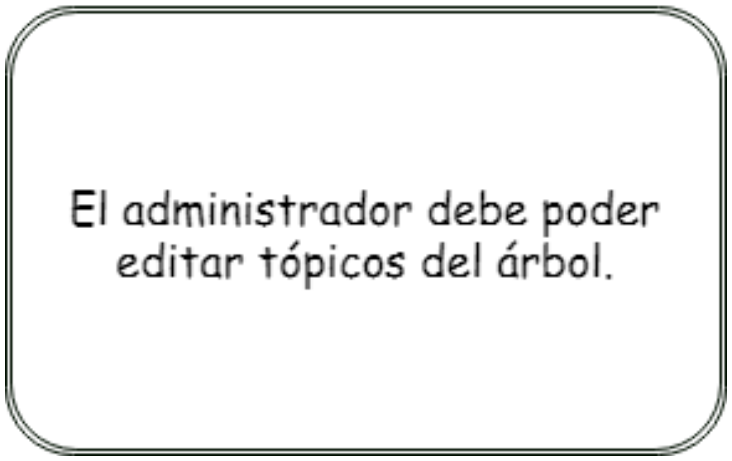

Figura 6. 34. Tarjeta de historia de usuario para Sprint 4.

Por tratarse de una historia de usuario sumamente sencilla se ha omitido el reverso de la tarjeta, ya que este no cuenta con contenido.

Como conclusión del Sprint Planning se ha obtenido lo siguiente:

- El entregable del Sprint que comienza corresponde a la implementación de la funcionalidad de edición de tópicos, esta funcionalidad será agregada a la pantalla mostrada en la Figura 6.32.
- La forma en que se conseguirá un Sprint exitoso será invirtiendo 3 horas diarias de trabajo.

En la Figura 6.35 se muestra la actualización del Product Backlog.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	Terminada	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.	Planificada	5	4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.	En proceso	4	4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 35. Product Backlog después del Sprint Planning del Sprint 4.

6.6.2. CONSTRUCCIÓN.

Por tratarse de una historia de usuario muy simple no se mostrará el desarrollo para la capa de acceso a datos y negocio.

6.6.2.1. PRESENTACIÓN.

En la Figura 6.36 se puede observar la implementación de la historia de usuario, ya que se muestra una columna extra en la tabla de tópicos, dicha columna contiene una liga con la leyenda “editar”, al hacer click sobre la liga se genera una petición hacia el método `modifyNodeTree` del controlador mostrado en la Figura 6.37.

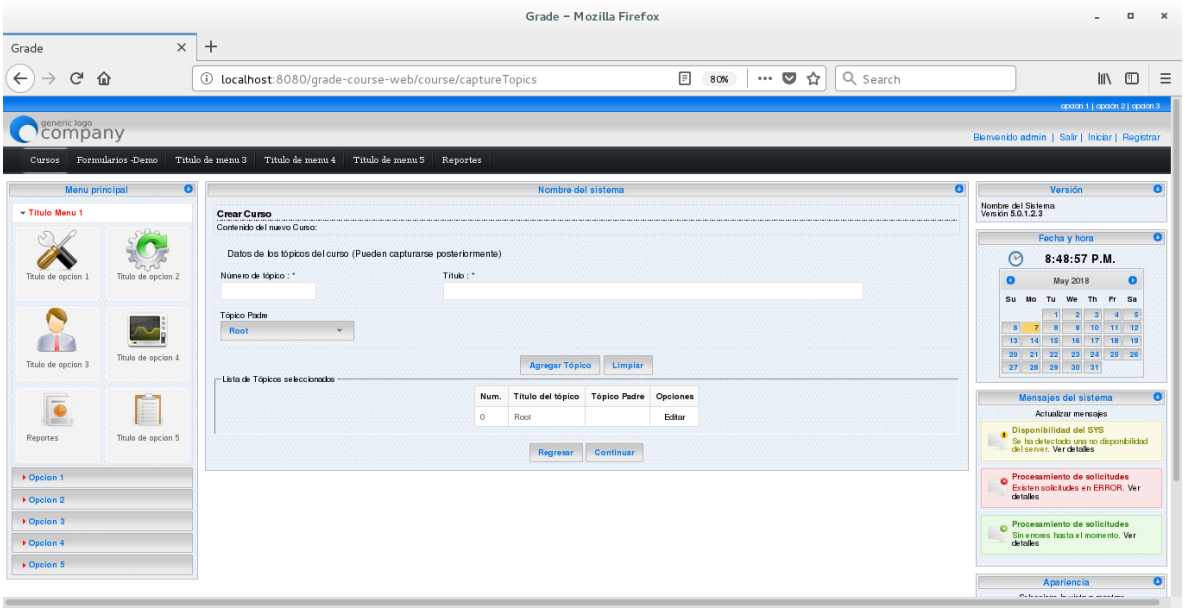

Figura 6. 36. Entregable del Sprint 4.

```
207    @RequestMapping("modifyTopic")
208    private String modifyNodeTree(@RequestParam("topicOrder") String topicOrderPrefix,
209            @SessionAttribute NodeTree<CourseTopicTree> nodeTree, ModelMap model) {
210
211        List<CourseTopicTree> topicList;
212        topicList = new ArrayList<>(5);
213        buildTopicsAsList(nodeTree, topicList);
214        CourseTopicTree courseTopicTree = new CourseTopicTree();
215        courseTopicTree.setTopicOrderPrefix(topicOrderPrefix);
216        model.addAttribute(
217                topicList.stream().filter(p -> p.equals(courseTopicTree)).findFirst().get());
218        model.addAttribute(topicList);
219
220        return viewCaptureTopics;
221    }
222
223 }
```

Figura 6. 37. Método modifyNodeTree.

Finalmente, el contenido de la Figura 6.36 ilustra el entregable final de este Sprint.

6.6.3. SPRINT RETROSPECTIVE.

En el Sprint Retrospective de esta iteración se habló de la problemática detectada en el Sprint 3, encontrando que dicho problema está a punto de ser erradicado en su totalidad, sin embargo, se exhorto a continuar trabajando por ello. Entre otros puntos positivos se habló del compromiso y profesionalismo que el equipo ha manejado a lo largo de los Sprints y cómo es que dichas virtudes han ayudado en gran manera a la realización de Sprints exitosos.

En cuanto a los aspectos negativos no existe nada por destacar.

En la Figura 6.38 se muestra la actualización de la plantilla de Product Backlog.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	Terminada	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.	Planificada	5	4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.	Terminada	4	4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.			6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 38. Product Backlog después del Sprint 4.

6.7. SPRINT 5.

La historia de usuario con ID 00-00-00-04 se considera que presenta una implementación similar a la historia de usuario con ID 00-00-00-05, por tal razón se ha omitido la explicación y desarrollo correspondiente al Sprint 5, limitándose a mostrar el entregable ilustrado en la Figura 6.39. A partir de este momento la actividad aparecerá como terminada.

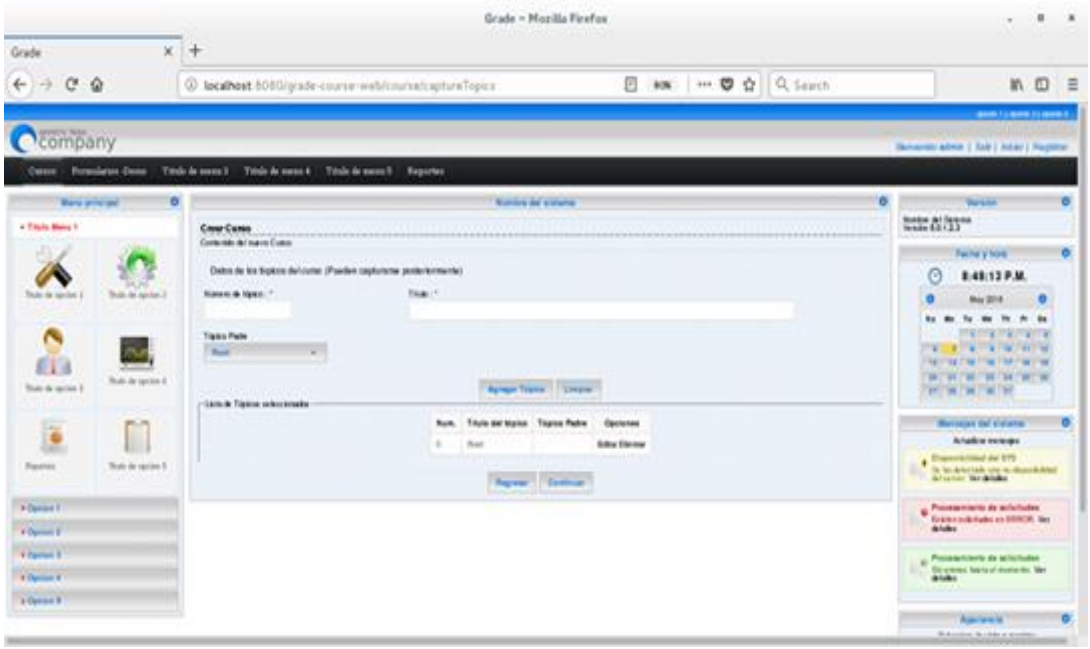

Figura 6. 39. Entregable del Sprint 5.

6.8. SPRINT 6.

6.8.1. SPRINT PLANNING.

La duración de este Sprint fue establecida a 6 semanas y la historia de usuario asignada corresponde al ID 00-00-00-06. En la figura 6.40 se puede observar la tarjeta de historia de usuario correspondiente.

Como conclusión del Sprint Planning se ha obtenido lo siguiente:

- El entregable del Sprint que comienza corresponde al desarrollo de una pantalla mediante la cual el administrador pueda capturar los recursos correspondientes a cada curso con la posibilidad de eliminar y editar los mismos.
- La forma en que se conseguirá un Sprint exitoso será invirtiendo 3 horas diarias de trabajo.

En la Figura 6.41 se muestra la actualización del Product Backlog.

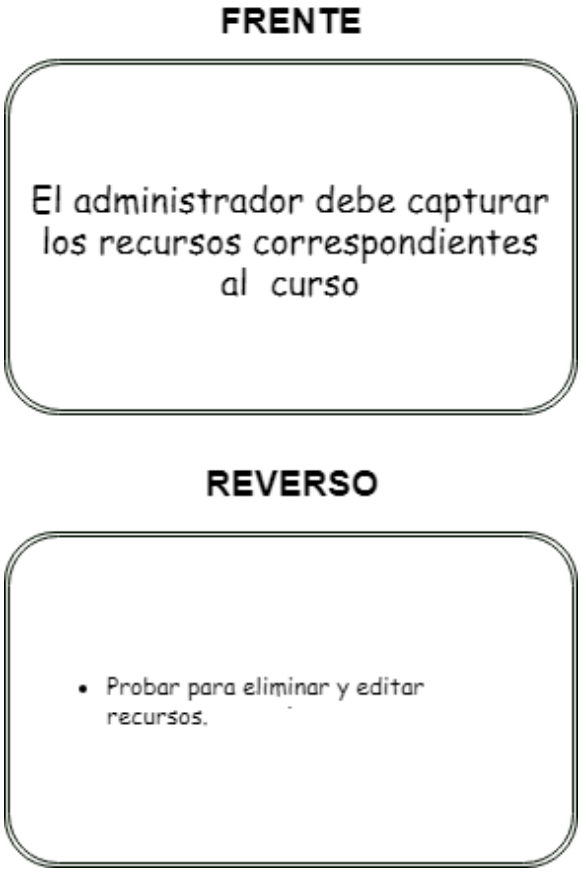

Figura 6. 40. Tarjeta de usuario para Sprint 5.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	Terminada	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.	Terminada	5	4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.	Terminada	4	4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.	En proceso	6	6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 41. Product Backlog después del Sprint Planning para Sprint 6.

6.8.2. CONSTRUCCIÓN.

Para la demostración del desarrollo correspondiente a este Sprint se ha omitido la capa de negocio debido a que los aspectos más relevantes no se centran en dicha capa.

6.8.2.1. ACCESO A DATOS.

Con base en el estudio del dominio se ha conseguido identificar una nueva entidad (CourseResource), lo anterior se ilustra en la Figura 6.42 mediante un diagrama de clases.

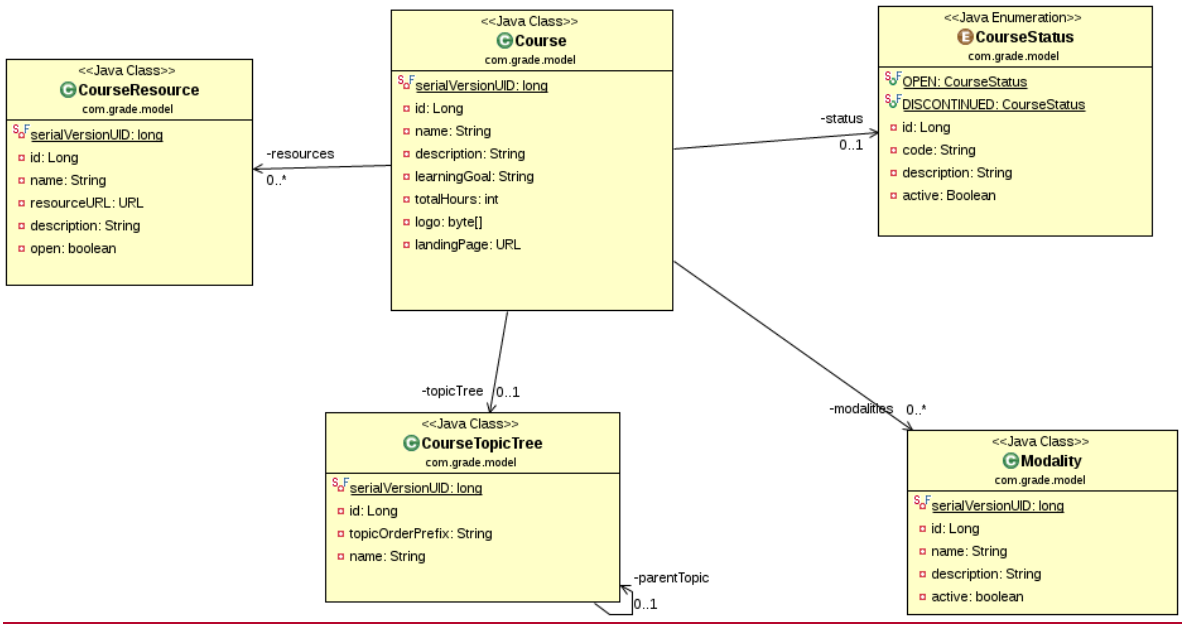

Figura 6. 42. Diagrama de clases con la identificación de una nueva entidad, CourseResource.

La Figura 6.42 muestra otra relación de composición la cual fue identificada con base en el estudio y entendimiento del dominio, pues un recurso no tiene razón de ser sin un curso que lo incluya. Se puede observar como el diagrama de clase ha ido creciendo notablemente a lo largo de los Sprints.

6.8.2.2. PRESENTACIÓN.

En la Figura 6.43 se observa el entregable que corresponde a este Sprint el cual cumple con todos los requisitos necesarios. Mediante la pantalla mostrada el administrador tendrá la posibilidad de agregar recursos al curso, así como eliminarlos y editarlos según desee.

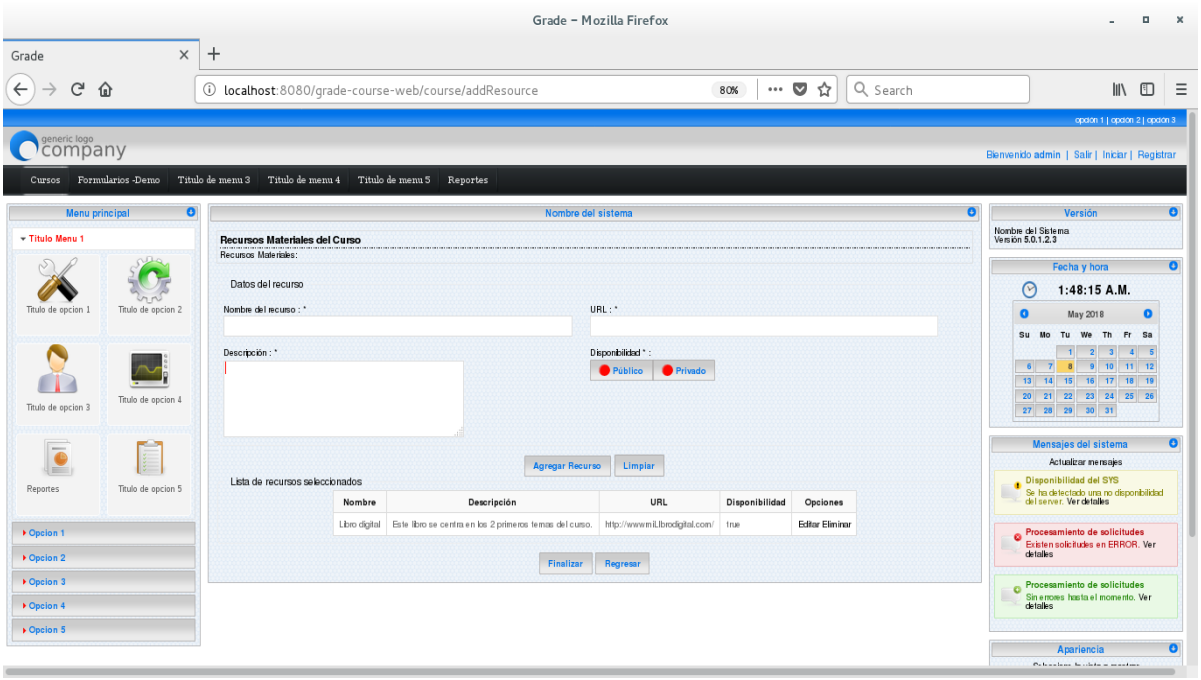

Figura 6. 43. Entregable del Sprint 6.

6.8.3. SPRINT RETROSPECTIVE.

En el Sprint Retrospective de esta iteración se habló de los puntos positivos que se tuvieron a lo largo del Sprint, pero sobre todo se hizo hincapié en los aspectos negativos que se lograron corregir.

En la Figura 6.44 se puede observar el Product Backlog completado, cuando no existen más historias de usuario por hacer entonces se puede decir que el trabajo ha sido completado de manera exitosa.

Identificador (ID)	Enunciado de la historia	Estado	Sprint (Iteración)	Dimensión / Esfuerzo	Valor de negocio	Prioridad
00-00-00-01	El administrador debe capturar los datos generales del curso.	Terminada	1	5	18	3.6
00-00-00-02	El administrador debe poder adjuntar un logo para cada curso.	Terminada	2	1	3	3
00-00-00-03	El administrador debe capturar los datos generales del árbol de tópicos para cada curso.	Terminada	3	7	18	2.5
00-00-00-04	El administrador debe tener la posibilidad de poder eliminar tópicos del árbol.	Terminada	5	4	12	2.4
00-00-00-05	El administrador debe tener la posibilidad de poder editar tópicos del árbol.	Terminada	4	4	12	2.4
00-00-00-06	El administrador debe tener la posibilidad de capturar datos generales de los recursos para cada curso.	En proceso	6	6	14	2.3
00-00-00-07	El administrador debe adjuntar un archivo de descripción por curso.	Cancelada		2	4	2

Figura 6. 44. Product Backlog después del Sprint 6.

6.9. ADICIONAL.

A continuación, se mostrarán algunos entregables referentes a otros microservicios por considerarlos de gran interés.

La Figura 6.45 muestra la pantalla mediante la cual el alumno puede realizar su registro al sistema, esta pantalla corresponde al microservicio denominado `user microservice`. Como se puede observar se pide capturar los datos básicos y de contacto del alumno, los datos de contacto serán de gran importancia para la etapa de promoción de cursos.

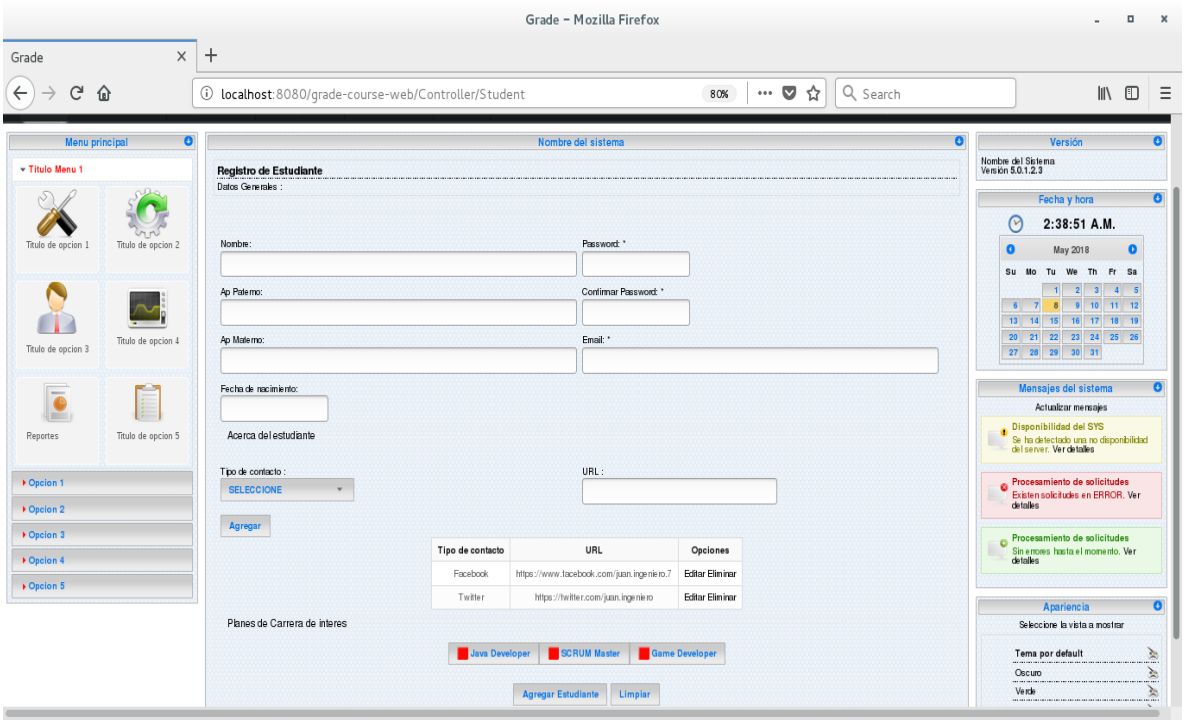

Figura 6. 45. Pantalla para registro de alumnos.

La Figura 6.46 corresponde a la pantalla mediante la cual el alumno puede registrarse a un curso y registrar el detalle de pago como lo es el número de pagos y la forma de pago, dicha pantalla corresponde al microservicio denominado `payment microservice`.

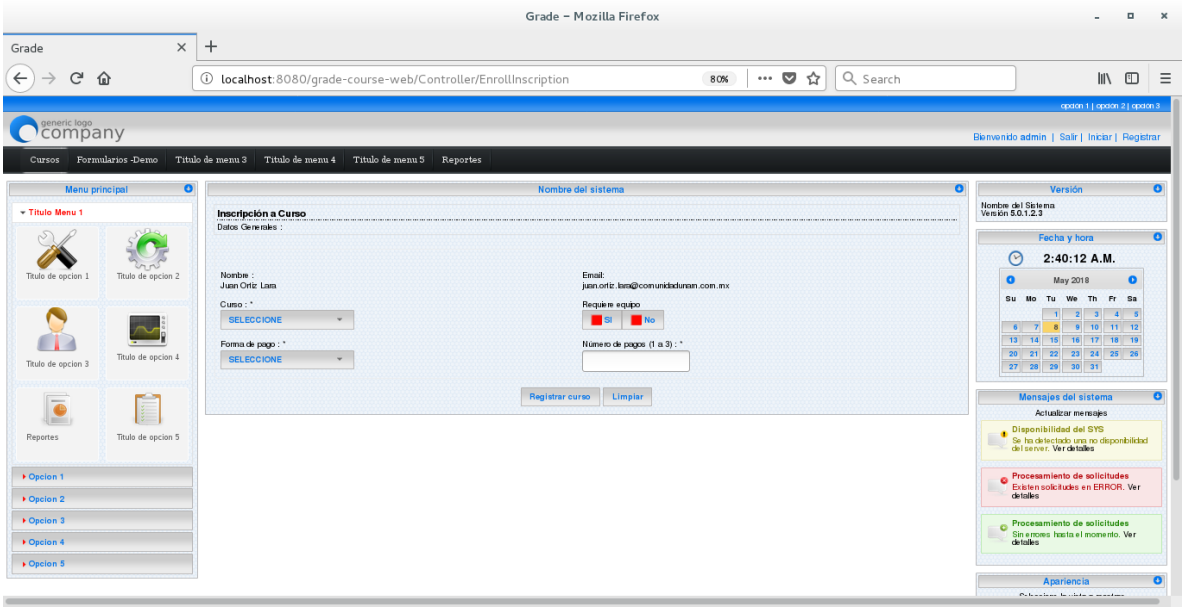

Figura 6. 46. Pantalla para registro de alumnos a cursos.

La Figura 6.47 corresponde a la pantalla necesaria para habilitar cursos a inscripción de alumnos. Esta pantalla pertenece al microservicio denominado course enroll microservice.

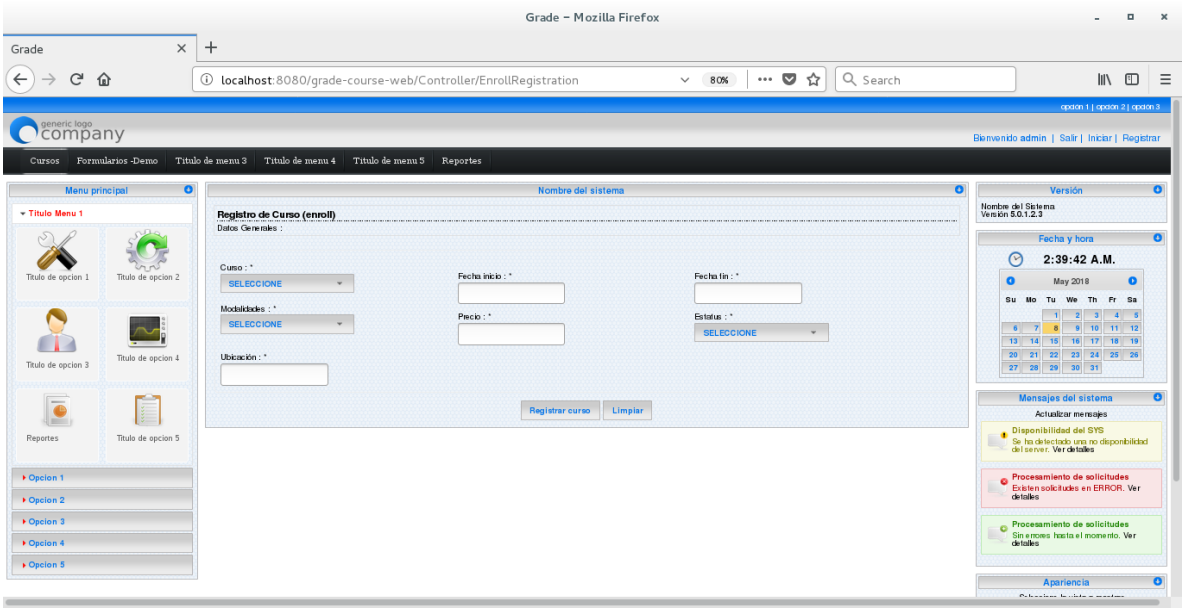

Figura 6. 47. Pantalla para habilitar cursos.

7. PRUEBAS Y ANÁLISIS DE RESULTADOS.

Según Glendford Myers el proceso de pruebas se puede definir como la ejecución del software con la intención de encontrar errores.[12] Dicho proceso lleva por objetivo asegurar que el software satisface los requerimientos acordados, sin embargo, su utilización es vital para encontrar situaciones de comportamiento indeseado.

La generación de una prueba de calidad consiste primeramente en seleccionar la característica del software a probar, posteriormente se debe elegir el tipo de prueba (más adelante se hablará de los tipos de pruebas más representativos), una vez que se ha realizado lo anterior es momento de generar los casos de prueba, estos consisten en un conjunto de entradas, condiciones de ejecución y resultados esperados para un objetivo en particular. Finalmente, el último paso es ejecutar el caso de prueba.

A continuación, se presenta un compendio de pruebas efectuadas al sistema desarrollado.

7.1. DISEÑO DE PRUEBAS UNITARIAS.

Las pruebas unitarias llevan por objetivo probar módulos pequeños de código de manera independiente, estas no acceden a recursos externos como archivos, bases de datos o webservices.

Para la realización de las pruebas unitarias se hará uso del soporte que ofrece Spring Framework, Junit, Spring mvc y Mockito, un Framework de pruebas.

Las herramientas anteriormente mencionadas han sido elegidas debido a la facilidad de utilización, alta confiabilidad, simplicidad y soporte que presentan, además de versatilidad.

Seguidamente, se presentan algunas pruebas unitarias efectuadas al sistema.

ID	01U
Funcionalidad	Probar la creación de objetos de tipo `Course`.
Descripción	Se desea probar que cada objeto de tipo `Course` contenga los atributos necesarios.
Consideraciones	La prueba fallara si alguno de los atributos presenta un valor nulo.

Figura 7. 1. Prueba unitaria con ID 01U.

[12] Glenford J. Myers. (2004). The Art of Software Testing. Canada: John Wiley & Sons, Inc.

Antes de insertar un objeto de tipo `Course` en la base de datos se debe asegurar que éste contenga los atributos asignados a él. Esta prueba lleva por objetivo certificar lo anterior, pues al encontrar atributos nulos o no validos se conseguirá un fallo. A continuación, se presenta el código empleado.

```
@ContextConfiguration(locations={"/courseServiceAppContext.xml","/hibrnate5AppContext.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class CourseValidationTest {
   Course course = new Course();

   /**
   * create data to test
   * @throws MalformedURLException
   */
   @Before
   public void createData() throws MalformedURLException {
     course.setModalities(generateModalities());
     course.setResources(generateResources());
     course.setTopicTree(generateCourseTopicTree().getData());
     course.setName("Java");
     course.setDescription("Basic");
     course.setLandingPage(new URL("https://javaBasics.com/"));
     course.setLearningGoal("Learn basic concepts");
     course.setTotalHours(4);
     course.setLogo(new byte[100]);
     course.setStatus(CourseStatus.OPEN);
   }

   /**
   * test data
   */
   @Test
   public void validateCourse() {
     assertTrue(course.getName() != null);
     assertTrue(course.getDescription() != null);
     assertTrue(course.getLandingPage() != null);
     assertTrue(course.getLearningGoal() != null);
     assertTrue(course.getTotalHours() >= 0);
     assertTrue(course.getLogo() != null);
     assertTrue(course.getStatus() != null);
     assertTrue(course.getModalities() != null);
     assertTrue(course.getResources() != null);
     assertTrue(course.getTopicTree() != null);
   }

   Set<Modality> generateModalities() {
     Set<Modality> modalitySet = new HashSet<Modality>();
     Modality modality1 = new Modality("on line", "via internet", true);
     Modality modality11 = new Modality("face to face ", "at School", true);
     modalitySet.add(modality1);
     modalitySet.add(modality11);
     return modalitySet;
   }
```

```
    Set<CourseResource> generateResources() throws MalformedURLException {
      Set<CourseResource> courseResourceList = new HashSet<CourseResource>();
      CourseResource courseResource1 = new CourseResource("Apuntes",
      new URL("https://notes.com/apuntes"), "Notes about course", true);
      CourseResource courseResource11 = new CourseResource("Book",
      new URL("https://book.com/book"), "Book for topics", true);
      CourseResource courseResource111 = new CourseResource("Video",
      new URL("https://video.com/video"), " Explicative video", false);

      courseResourceList.add(courseResource1);
      courseResourceList.add(courseResource11);
      courseResourceList.add(courseResource111);
      return courseResourceList;
    }

    NodeTree<CourseTopicTree> generateCourseTopicTree() {
      NodeTree<CourseTopicTree> rootNode;
      NodeTree<CourseTopicTree> node1, node11, node111, node2;
      rootNode = new NodeTree<>(new CourseTopicTree("", "Root Topic"));
      // child 1
      node1 = rootNode.addChild(new CourseTopicTree("1", "Level 1"));
      node11 = node1.addChild(new CourseTopicTree("1.1", "Level 1.1"));
      node111 = node11.addChild(new CourseTopicTree("1.1.1", "Level 1.1.1"));
      node111.addChild(new CourseTopicTree("1.1.1.1", "Level 1.1.1.1"));
      // child 2
      node2 = rootNode.addChild(new CourseTopicTree("2", "Level 2"));
      node2.addChild(new CourseTopicTree("2.1", "Level 2.1"));
      node2.addChild(new CourseTopicTree("2.1", "Level 2.2"));
      return rootNode;
    }
}
```

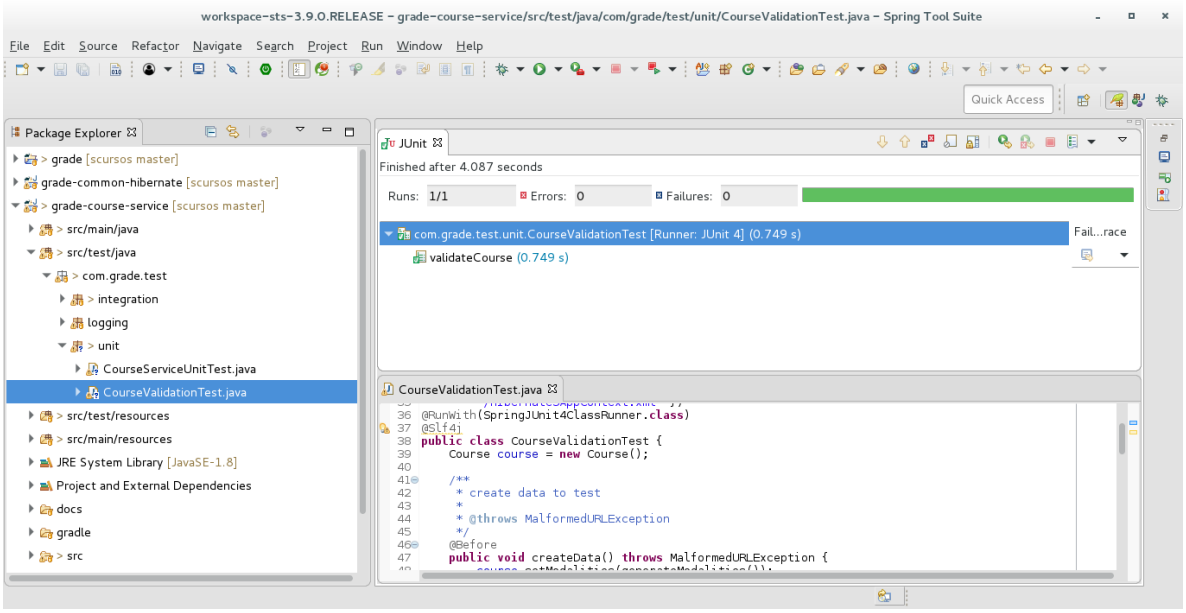

Figura 7. 2. Prueba unitaria con id 01U exitosa.

Como se puede observar en la Figura 7.2 la prueba ha sido pasada con éxito, lo cual asegura que el componente (entidad de tipo `Course`) funciona correctamente.

ID	02U
Funcionalidad	Probar el funcionamiento del servicio (`CourseService`)
Descripción	Se desea probar el método `createCourse`.
Consideraciones	La prueba fallará si existe algún problema con la funcionalidad del método.

Figura 7. 3. Prueba unitaria con id 02U.

La prueba con ID 02U lleva por objetivo verificar el funcionamiento del método `createCourse` del servicio `CourseService`, para realizar esta prueba se ha hecho uso de los mocks de mockito, siendo estos objetos de prueba que simulan objetos reales. A continuación, se presenta el código empleado.

```
@ContextConfiguration(locations = {
"/courseServiceAppContext.xml","/hibernate5AppContext.xml" })
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class CourseServiceUnitTest {
  @Mock
  private CourseDAO courseDAO;
  @Mock
  private Course course;

  @InjectMocks
  private CourseService coureService = new CourseServiceImpl();

  @Before
  public void setUP() {
    MockitoAnnotations.initMocks(this);
  }

  @Test
  public void testCreate() {
    doNothing().when(courseDAO).createCourse(course);
    coureService.createCourse(course);
    verify(courseDAO).createCourse(course);
  }
}
```

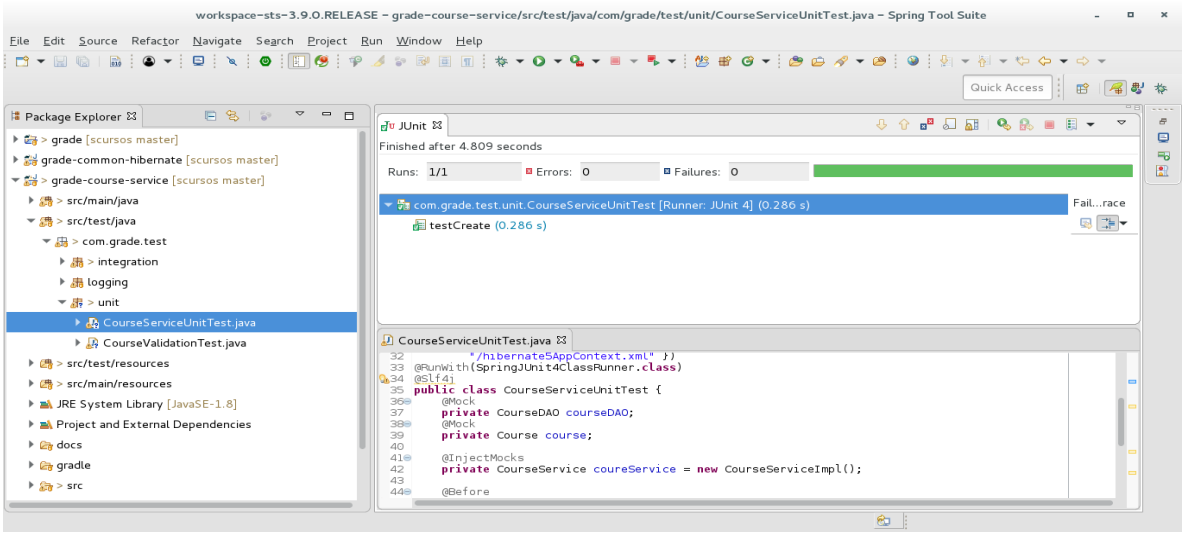

Figura 7. 4. Prueba unitaria con id 02U exitosa.

Como se puede observar en la Figura 7.4 la prueba ha sido aprobada lo cual garantiza el correcto funcionamiento del método `createCourse` del componente `CourseService`.

ID	03U
Funcionalidad	Probar el funcionamiento del controlador (`CourseGeneralDataController`).
Descripción	Se desea probar que el método `captureGeneralData` funcione optimamente.
Consideraciones	La prueba fallará si la respuesta de dicho método no es la esperada.

Figura 7. 5. Prueba unitaria con id 03U.

La prueba con ID 03U mostrada en la Figura 7.5 lleva por objetivo probar el método `captureGeneralData` del controlador `CourseGeneralDataController`, para la realización de esta prueba se ha utilizado el soporte que brinda Spring mvc para pruebas. La principal funcionalidad del método es devolver como respuesta la página de captura de datos generales para un curso. El código implementado se muestra a continuación.

```
@ContextConfiguration(locations = { "/courseServiceAppContext.xml"
"/hibrnate5AppContext.xml" })
@WebAppConfiguration
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class GeneralDataControllerTest {
  @Resource
  private WebApplicationContext wac;

  private MockMvc mockMvc;

  @Before
  public void init() {
    mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
  }

  @Test
  public void testCaptureGeneralData() throws Exception {
    mockMvc.perform(get("/course/captureGeneralData")).andExpect(status().isOk())
    .andExpect(view().name("course/captureGeneralData"));
  }
}
```

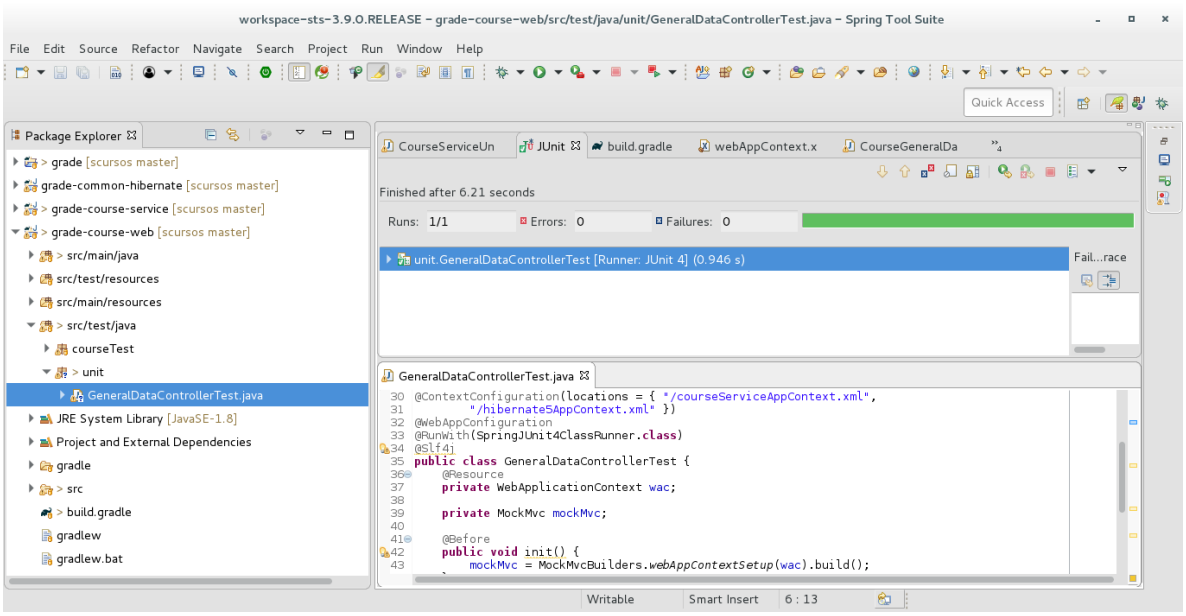

Figura 7. 6. Prueba unitaria con id 03U exitosa.

La Figura 7.6 muestra la prueba aprobada, esto garantiza el correcto funcionamiento del método.

7.2. DISEÑO DE PRUEBAS DE INTEGRACIÓN.

Las pruebas de integración verifican que los componentes de la aplicación funcionan correctamente actuando en conjunto y se realizan para dos o más unidades de software.

Para la realización de las pruebas de integración se hará uso del soporte que ofrece Spring Framework.

ID	01I
Funcionalidad	Probar la funcionalidad de creación de un recurso desde la capa de negocio.
Descripción	Se desea probar el funcionamiento de los componentes `CourseResourceService` y `CourseResourceDAO` trabajando en conjunto con la base de datos.
Consideraciones	La prueba fallará si el objeto no ha sido insertado en la base de datos.

Figura 7. 7. Prueba de integración con id 01I.

La prueba con ID 01I mostrada en la Figura 7.7 lleva por objeto probar la funcionalidad de los componentes encargados de insertar un recurso en la base de datos. A continuación, se presenta el código empleado.

```
@ContextConfiguration(locations = {
"/courseServiceAppContext.xml","/hibernate5AppContext.xml" })
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class CourseResourceTest extends AbstractTransactionalJUnit4SpringContextTests {
  @Resource
  private CourseResourceService courseResourceService;
  @Resource
  private CourseResourceDAO courseResourceDAO;

  @Test
  public void createResource()  throws MalformedURLException {
    CourseResource courseResource = createTestData();
    log.debug("Starting test - Creating Resource");
    courseResourceService.createResource(courseResource);
    ((GenericHibernateDAOImpl) courseResourceDAO).flush();
    validate(courseResource);
  }
  CourseResource createTestData() throws MalformedURLException {
    CourseResource courseResource = new CourseResource("Libros complementarios",new
       URL("http:\\www.librosComp.com"),
                    "Libros de apoyo para los temas 1 y 2 del curso", true);
    return courseResource;
  }
  private void validate(CourseResource courseResource) {
    log.debug("checking resource insertion: {}", courseResource);
    assertNotNull("topic id was null", courseResource.getId());
    assertNotNull("topic url was  null", courseResource.getResourceURL());
    assertNotNull("topic description was  null",courseResource.getDescription());
    assertNotNull("topic open was  null", courseResource.isOpen());
       }
}
```

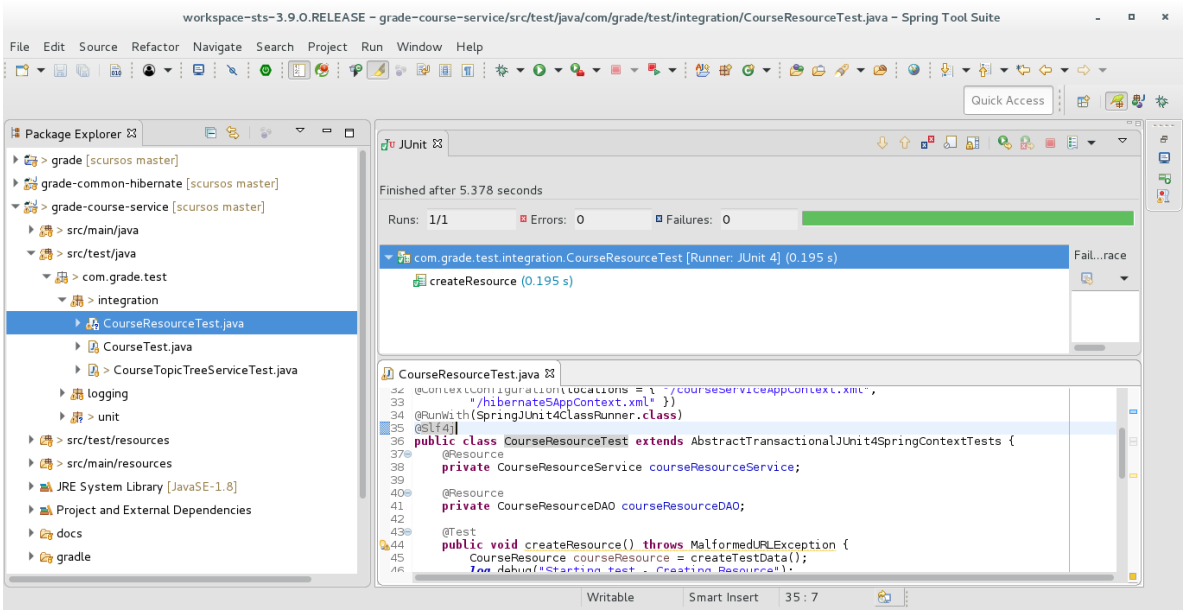

Figura 7. 8. Prueba de integración con id 01I exitosa.

La Figura 7.8 muestra la prueba aprobada lo cual implica que el objeto ha sido insertado de manera correcta a la base de datos y los componentes funcionan de manera correcta trabajando en conjunto.

ID	02I
Funcionalidad	Probar la funcionalidad de creación de un árbol de tópicos desde la capa de negocio.
Descripción	Se desea probar el funcionamiento de los componentes CourseTopicTreeService y CourseTopicTreeDAO trabajando en conjunto con la base de datos.
Consideraciones	La prueba fallará si el objeto no ha sido insertado en la base de datos.

Figura 7. 9. Prueba de integración con id 02I.

La prueba con ID 02I mostrada en la Figura 7.9 lleva por objeto probar la funcionalidad de los componentes encargados de insertar un árbol de tópicos en la base de datos. A continuación, se presenta el código empleado.

```
@ContextConfiguration(locations = {
"/courseServiceAppContext.xml","/hibernate5AppContext.xml" })
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class CourseTopicTreeServiceTest extends
AbstractTransactionalJUnit4SpringContextTests {

  @Resource
  private CourseTopicTreeService courseTopicService;

  @Resource
  private CourseTopicTreeDAO courseTopicTreeDAO;

      /**
       * This test creates a Course Topic Tree. All data is rolled back. This Behavior is
       * possible because we are extending of
       * {@link AbstractTransactionalJUnit4SpringContextTests}
       */
  @Test
  public void createTopicTree() {
    NodeTree<CourseTopicTree> topicTree;

    log.debug("Starting test - Creating Course Topic Tree");
    topicTree = createTestData();
    courseTopicService.createCourseTopicTree(topicTree);
    log.debug("Validating data ");
    // Invoke flush to force db sync and avoid false positives.
    ((GenericHibernateDAOImpl) courseTopicTreeDAO).flush();
     validate(topicTree);
  }
  private void validate(NodeTree<CourseTopicTree> topicTree) {
    List<NodeTree<CourseTopicTree>> children;
    log.debug("checking node insertion: {}", topicTree.getData());
    assertNotNull("topic id was null", topicTree.getData().getId());
    assertNotNull("topic name was  null", topicTree.getData().getName());
    assertNotNull("topic orderPrefix was  null",topicTree.getData().getTopicOrderPrefix());
    children = topicTree.getChildren();
    children.forEach(child -> {
    validate(child);
     });
   }
```

```
 NodeTree<CourseTopicTree> createTestData() {
   NodeTree<CourseTopicTree> rootNode;
   NodeTree<CourseTopicTree> node1, node11, node111, node2;
   rootNode = new NodeTree<>(new CourseTopicTree("", "Root Topic"));
   // child 1
   node1 = rootNode.addChild(new CourseTopicTree("1", "Level 1"));
   node11 = node1.addChild(new CourseTopicTree("1.1", "Level 1.1"));
   node111 = node11.addChild(new CourseTopicTree("1.1.1", "Level 1.1.1"));
   node111.addChild(new CourseTopicTree("1.1.1.1", "Level 1.1.1.1"));
    // child 2
   node2 = rootNode.addChild(new CourseTopicTree("2", "Level 2"));
   node2.addChild(new CourseTopicTree("2.1", "Level 2.1"));
   node2.addChild(new CourseTopicTree("2.1", "Level 2.2"));
    // child 3 rootNode.addChild(new CourseTopicTree("3", "Level 3"));
   return rootNode;
    }
}
```

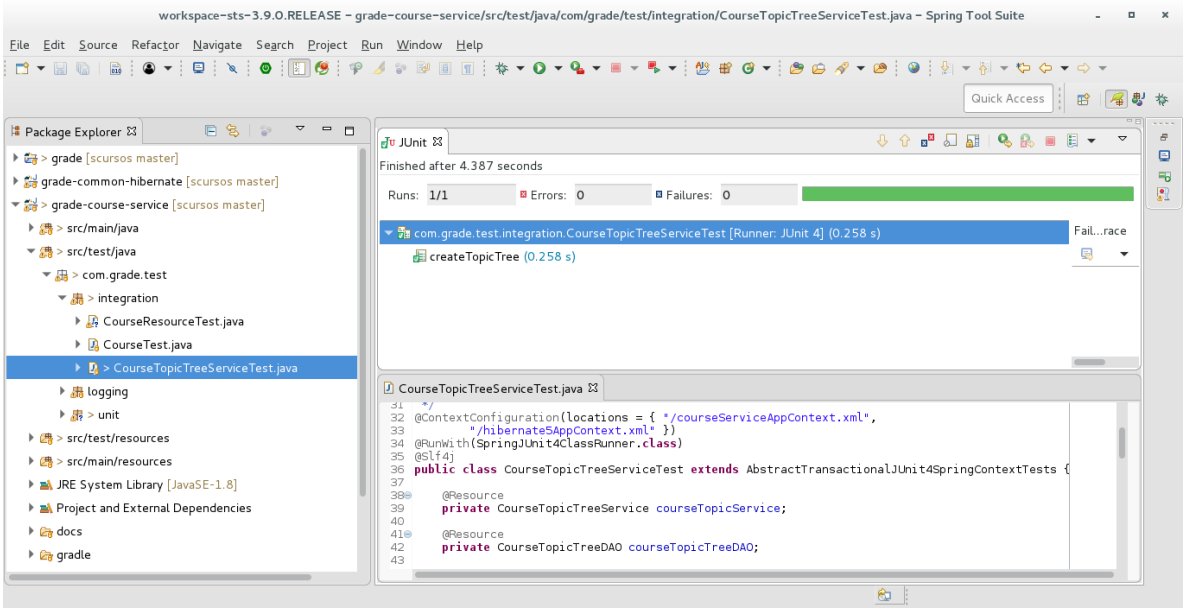

Figura 7. 10. Prueba de integración con id 02I exitosa.

La Figura 7.10 muestra la prueba aprobada lo cual implica que los objetos han sido insertados de manera correcta a la base de datos y los componentes funcionan de manera correcta trabajando en conjunto.

7.3. DISEÑO DE PRUEBAS FUNCIONALES.

Las pruebas funcionales se enfocan en garantizar que se cubran los requerimientos funcionales de la aplicación. Este tipo de pruebas se escriben desde la perspectiva del usuario, con la intensión de confirmar que el sistema trabaja como el usuario espera.

Para la realización de las pruebas de funcionales se hará uso del soporte que ofrece Spring Framework, Junit y Selenium.

Se ha decidido utilizar Selenium una herramienta de software libre, debido a la capacidad que presenta para: grabar, editar y depurar casos de prueba, que podrán ser ejecutados de forma automática e iterativa posteriormente. Es útil para pruebas de compatibilidad entre navegadores.

ID	01F
Funcionalidad	Probar la funcionalidad de creación de un curso incluyendo, creación de recursos y el árbol de tópicos.
Descripción	Se desea probar el funcionamiento de la creación de un curso desde la perspectiva del usuario.
Consideraciones	La prueba fallará si la creación del curso no se culmina.

Figura 7. 11. Prueba funcional con id 01F.

La prueba con ID 01F mostrada en la Figura 7.11 tiene por objetivo probar la creación de un curso a través del navegador, como lo haría el usuario final. A continuación, se presenta el código empleado.

```
@ContextConfiguration(locations = {
"/courseServiceAppContext.xml","/hibernate5AppContext.xml" })
@RunWith(SpringJUnit4ClassRunner.class)
@Slf4j
public class CourseTestWeb {

  private static final String geckoDriver = "/home/ABIGAIL/Downloads/geckodriver";
  private static final String URL =
              "http://localhost:8080/grade-course-web/course/captureGeneralData";
  private static final String URLImage = "/home/ABIGAIL/Pictures/logo.jpg";
  private static WebDriver driver;
  private static WebElement element;

  /**
  * This method open the browser
  */
  @BeforeClass
  public static void openBrowser() {
    System.setProperty("webdriver.gecko.driver", geckoDriver);
    driver = new FirefoxDriver();
    driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
  }

  /**
  * This method testing course creation
  */
  @Test
  public void valid_createCourseTest() {
    driver.get(URL);
    generalDataCourseTest();
    topicTreeCourseTest();
    resourceCourseTest();
    driver.findElement(By.xpath(".//*[@value='Finalizar']")).click();
    element = driver.findElement(By.xpath(".//*[@id='command']"));
    Assert.assertNotNull(element);
  }
```

```
/**
* This method close the browser
*/
@AfterClass
public static void closeBrowser() {
  driver.quit();
}

/**
* This method testing general data capture
*/
public void generalDataCourseTest() {
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("JAVA");
  driver.findElement(By.xpath(".//*[@name='totalHours']")).sendKeys("5");
  driver.findElement(By.xpath(".//*[@name='description']")).sendKeys("Curso avanzado");
  driver.findElement(By.xpath(".//*[@name='learningGoal']"))
                .sendKeys("Te convertiras en java developer");
  driver.findElement(By.xpath(".//*[@id='landingPage']"))
                .sendKeys("http://cleventy.com/tutorial-selenium-primeros-pasos/");
  driver.findElement(By.xpath(".//*[@id='status-button']")).sendKeys("OPEN");
  driver.findElement(By.xpath(".//*[@for='modalities0.id1']")).click();
  driver.findElement(By.xpath(".//*[@for='modalities1.id1']")).click();
  driver.findElement(By.xpath(".//*[@name='file']")).sendKeys(URLImage);
  driver.findElement(By.xpath(".//*[@value='Continuar']")).click();
}

/**
* This method testing topicTree capture
*/
public void topicTreeCourseTest() {
  driver.findElement(By.xpath(".//*[@name='topicOrderPrefix']")).sendKeys("1");
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Level 1");
  driver.findElement(By.xpath(".//*[@id='parentTopic-button']")).sendKeys("Root");
  driver.findElement(By.xpath(".//*[@value='Agregar Tópico']")).click();

          cleanObjectTopic();

  driver.findElement(By.xpath(".//*[@name='topicOrderPrefix']")).sendKeys("2");
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Level 2");
  driver.findElement(By.xpath(".//*[@id='parentTopic-button']")).sendKeys("Root");
  driver.findElement(By.xpath(".//*[@value='Agregar Tópico']")).click();

          cleanObjectTopic();

  driver.findElement(By.xpath(".//*[@name='topicOrderPrefix']")).sendKeys("1.1");
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Level 1.1");
  driver.findElement(By.xpath(".//*[@id='parentTopic-button']")).sendKeys("Level 1");
  driver.findElement(By.xpath(".//*[@value='Agregar Tópico']")).click();

          cleanObjectTopic();

  driver.findElement(By.xpath(".//*[@name='topicOrderPrefix']")).sendKeys("1.1.1");
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Level 1.1.1");
  driver.findElement(By.xpath(".//*[@id='parentTopic-button']")).sendKeys("Level 1");
  driver.findElement(By.xpath(".//*[@value='Agregar Tópico']")).click();

          cleanObjectTopic();

  driver.findElement(By.xpath(".//*[@href='removeTopic?topicOrder=1']")).click();
  driver.findElement(By.xpath(".//*[@href='modifyTopic?topicOrder=2']")).click();
  driver.findElement(By.xpath(".//*[@name='name']")).clear();
  driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Level dos");
  driver.findElement(By.xpath(".//*[@value='Agregar Tópico']")).click();
}
```

```
  /**
  * This method testing resourceCourse capture
  */

  public void resourceCourseTest() {
    driver.findElement(By.xpath(".//*[@href='/grade-course-web/course/captureResources']"))
                    .click();
    driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Libros Electrónicos");
    driver.findElement(By.xpath(".//*[@name='resourceURL']"))
                    .sendKeys("http://libros.com");
    driver.findElement(By.xpath(".//*[@name='description']"))
                    .sendKeys("De gran ayuda para temas 1 y 2");
    driver.findElement(By.xpath(".//*[@for='publico']")).click();
            driver.findElement(By.xpath(".//*[@value='Agregar Recurso']")).click();
            cleanObjectResource();
    driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Videoconferencia");
    driver.findElement(By.xpath(".//*[@name='resourceURL']"))
                    .sendKeys("http://videoConferencia.com");
    driver.findElement(By.xpath(".//*[@name='description']"))
                    .sendKeys("De gran ayuda para temas 3 y 4");
    driver.findElement(By.xpath(".//*[@for='privado']")).click();
    driver.findElement(By.xpath(".//*[@value='Agregar Recurso']")).click();
            cleanObjectResource();

    driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Apuntes");
    driver.findElement(By.xpath(".//*[@name='resourceURL']"))
                    .sendKeys("http://apuntes.com");
    driver.findElement(By.xpath(".//*[@name='description']"))
                    .sendKeys("De gran ayuda para temas 5 y 6");
    driver.findElement(By.xpath(".//*[@for='publico']")).click();
    driver.findElement(By.xpath(".//*[@value='Agregar Recurso']")).click();
            cleanObjectResource();
    driver.findElement(
    By.xpath(".//*[@href='editResource?resourceURL=http://libros.com']")).click();
    driver.findElement(By.xpath(".//*[@name='name']")).clear();
    driver.findElement(By.xpath(".//*[@name='name']")).sendKeys("Libros Electronicos");
    driver.findElement(By.xpath(".//*[@value='Agregar Recurso']")).click();
    driver.findElement(By.xpath(".//*[@href='deleteResource?resourceURL=http://videoConferenc
    ia.com']")).click();
  }

  /**
  * This method clean ObjectTopic
  */
  public void cleanObjectTopic()
  driver.findElement(By.xpath(".//*[@name='topicOrderPrefix']")).clear();
   driver.findElement(By.xpath(".//*[@name='name']")).clear();
  }

  /**
  * This method clean ObjectResource
  */
  public void cleanObjectResource() {
    driver.findElement(By.xpath(".//*[@name='name']")).clear();
    driver.findElement(By.xpath(".//*[@name='resourceURL']")).clear();
    driver.findElement(By.xpath(".//*[@name='description']")).clear();
  }
}
```

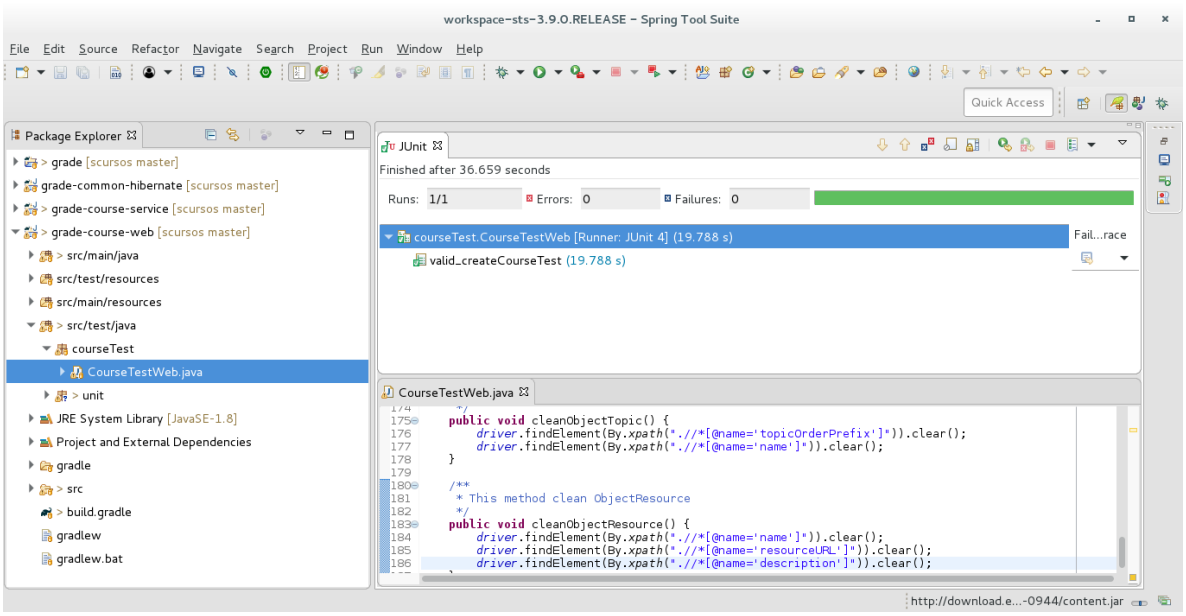

Figura 7. 12. Prueba de funcional con id 01F exitosa.

La Figura 7.12 muestra la prueba exitosa por lo que se garantiza que el software realiza las funciones esperadas por el usuario, esto con base en los requerimientos antes analizados.

Finalmente, la etapa de pruebas se concluye satisfactoriamente, pues se ha garantizado que los requerimientos establecidos anteriormente han sido cubiertos. Esto se ha logrado concluir con base en el análisis de resultados mostrado por cada prueba ejecutada.

8. CONCLUSIONES.

El objetivo de la realización de la presente tesis fue otorgar una solución ante la problemática que sufren los profesionistas del área de TI al no saber hacia dónde enfocar el crecimiento profesional. La solución consistió en la creación de una herramienta encargada de automatizar los procesos de negocio de las empresas enfocadas en ofrecer este tipo de servicios.

Este trabajo presentó el desarrollo para la construcción de dicha herramienta, desde la etapa de análisis hasta las pruebas efectuadas, todo esto a través del uso de tecnologías de vanguardia. El desarrollo se realizó a través del lenguaje Java y Spring, uno de los frameworks más populares hoy día, en conjunto con algunas tecnologías complementarias como lo fueron Hibernate, Selenium y Java Script entre otras. El diseño y arquitectura empleados para su construcción la posicionan a la altura de sistemas empresariales modernos.

El desarrollo de la herramienta descrita demandó el conocimiento de diversos temas referentes a la carrera de Ingeniería en computación como lo fueron lenguajes de programación, ingeniería de software y bases de datos, entre otros. Además de exigir la investigación y de muchos más.

El futuro que se vislumbra para la herramienta creada consiste en someterla a una segunda etapa de desarrollo la cual permitirá adicionarle funcionalidades y posteriormente comercializarla, primeramente, mediante una versión gratuita, de prueba, para las empresas que ya se dedican a prestar servicios enfocados al crecimiento del desarrollo profesional en cuanto a profesionista de TI.

Como objetivo a largo plazo se espera que la herramienta evolucione y no solo se presente como solución ante empresas encargadas de ofrecer servicios orientados al crecimiento profesional en el área de TI, sino sea capaz de brindar ayuda en los procesos de negocio para empresas encargadas de ofrecer servicios orientados al crecimiento profesional en cualquier área de estudio.

Finalmente, la herramienta desarrollada representa una muestra clara del uso de la tecnología actual para la resolución de un problema social como el presentado en este trabajo de tesis, que de no resolverse podría desembocar en la generación de otro tipo de problemas.

9. BIBLIOGRAFÍA Y MESOGRAFÍA.

- Eric Ries. (2011). El método Lean Startup.: DEUSTO.
- Stefan Jablonski & Ilia Petrov & Christian Meiler & Udo Mayer. (2004). Guide to Web Application and Platforms Arquitectures. Alemania: Springer.
- Bernhard Hitpass. (2017). BPM Bussines Process Manag. Santiago de Chile: BPM Center.
- Carlos Blé y colaboradores. (2010). Diseño Ágil con TDD.: Creative Commons
- Ian Somerville. (2011). Ingeniería de software. Argentina: Pearson
- Isidro Ramos Salavert & Maria Dolores Lozano Pérez. (2000). Ingeniería de software y bases de datos, tendencias actuales.: Ediciones de la Universidad de Castilla-La Mancha.
- Ken Pugh. (2006). Interface Oriented Design. Texas: The pragmatic bookshelf.
- Glenford J. Myers. (2004). The Art of Software Testing. Canada: John Wiley & Sons, Inc
- Craig Walls & Ryan Breidenbach. (2008). Spring in action. United States of America: Manning.
- Joseph Schmuller. (2000). Aprendiendo UML en 24 horas. México: Prentice Hall.
- Ken Schwaber & Jeff Sutherland. (2013). La guía definitiva de Scrum: Las Reglas del Juego. 2017, de scrumguides Sitio web: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf
- Spring. (2017). Spring Framework Documentation. 2017, de Spring.io Sitio web: https://docs.spring.io/spring/docs/5.1.0.BUILD-SNAPSHOT/spring-framework-reference/
- Oracle. (2017). PREVNEXTFRAMESNO FRAMES Java(TM) EE 8 Specification APIs. 2017, de Oracle Sitio web: https://javaee.github.io/javaee-spec/javadocs/
- SeleniumHQ. (2012). Selenium Documentation. 2017, de SeleniumHQ Sitio web: https://www.seleniumhq.org/docs/
- Mockito. (2017). Mockito Documentation. 2018, de mockito Sitio web: https://static.javadoc.io/org.mockito/mockito-core/2.18.3/org/mockito/Mockito.html