UNIVERSIDAD TECNICA PARTICULAR DE LOJA

TESIS DE INGENERIA

GitEDU:
Un Sistema de Integracion de LTI y Git

Autor:
Nicholas SPALDING
EARLEY-DOLENC

Supervisor:
Ing. Jorge LOPEZ

Un tesis entregado para cumplir con los requisitos
del titulo de Ingeneria en Sistemas Informaticos y Computacion

en el

Laboratorio de Tecnologias Web
Escuela de Ciencias de la Computacion

2 de agosto de 2017

Declaracion of Autoria

Yo, Nicholas SPALDING EARLEY-DOLENC, declaro que esta tesis titulado, «GitEDU: Un Sistema de Integracion de LTI y Git» y que el trabajo presentado aqui es mi propio. Yo afirmo que:

- Este trabajo fue realizado principalmente mientras que fui candidato de un titulo de investigacion en la Universidad.
- Partes de esta tesis que se han sido utilizados por algun otro titulo o cualificacion, se lo ha señalado de una forma clara e transparente.
- Partes de esta tesis donde he consultado el trabajo publicado de terceros, se lo ha citado de la forma correcta.
- Donde he citado el trabajo de tercersos, la fuente siempre esta dada. Con la excepcion de aquellos citas, esta thesis es totalmente mi propio trabajo.
- He documentado mis principales fuentes de ayuda.
- En lugares donde esta tesis esta basada en trabajo realizado por terceros, he documentado claramente cuales partes realizaron ellos y cuales partes yo realicé.

Firmado:

Fecha:

«Elegimos… hacer las… cosas, no porque sea fácil, sino porque es difícil… Porque esta meta, servirá para organizar y probar lo mejor de nuestras energías y habilidades.»

John F. Kennedy

Universidad Tecnica Particular de Loja

Resumen

Departamento de Ciencias de la Computacion y Electronica
Escuela de Ciencias de la Computacion

Ingeneria en Sistemas Informaticos y Computacion

GitEDU:
Un Sistema de Integracion de LTI y Git

by Nicholas SPALDING EARLEY-DOLENC

Para mejorar el entorno educativo para los estudiantes de carreras que involucran programacion, es necesario el desarrollo de un sistema que integra los LMS de ahora atravez del estandar LTI con GitLab CE, un editor de codigo en linea y un entorno para la ejeccucion del mismo conlleva varias fases. En un fase inicial, es importante analizar la problematica, el entorno existente y caracteristicas de sistemas similares para poder formular una solucion facible y novedoso desde sus requisitos funcionales y diseño. Con un marco de diseño, se puede proceder a definir una arquiectura que asegurará que se cumple con los requisitos no funcionales. Con el entorno bien estudiado y un diseño robusto, se puede inicializar con varias fases agiles e incrementales de desarrollo, pruebas y desplegue hasta llegar a un producto final.

Índice general

Índice de figuras

Índice de cuadros

Lista de Abbreveaturas

CSS	**C**ascading **S**yle **S**heets (Hojas de Estilo en Cascada)
FOSS	**F**ree and **O**pen **S**ource **S**oftware (Software Abierto/Gratuito/Libre)[1]
FLOSS	**F**ree/**L**ibre and **O**pen **S**ource **S**oftware (Software Abierto y Libre)
GNU	**G**NU is **N**ot **U**NIX (GNU no es UNIX)
GPL	**G**NU **P**ublic **L**icence (Licencia Publica de GNU)
GPLv3	**GPL v**ersion **3** (Version 3 de la GPL)
HTML	**H**yper**T**ext **M**arkup **L**anguage (Lenguaje de Marcas de HiperTexto)
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol (Protocolo de Transferencia de HiperTexto)
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure (Protocolo Seguro de Transferencia de HiperTexto)
IaaS	**I**nfrastructure **a**s **a** **S**ervice (Infrastructura como un Servicio)
IAENG	**I**nternational **A**ssociation of **Eng**ineers
IDE	**I**ntegrated **D**evelopment **E**nvironment (Entorno de Desarrollo Integrado)
IMAP	**I**nternet **M**essage **A**ccess **P**rotocol (Protocolo de Accesso de Mensajes de Internet)
IMAPS	**IMAP** over **S**SL (IMAP atravez de SSL)
IPSec	**I**nternet **P**rotocol **Sec**urity (Seguridad de Protocolos de Internet)
KLOC	**K**ilo-**LOC** (mil lineas de codigo)
L2F	**L**ayer **2** **F**orwarding (Renvio de Capa 2)
L2TP	**L**ayer **2** **T**unneling **P**rotocol (Protocolo de Tunel de Capa 2)
LA	**L**earning **A**sset (Activo de Aprendizaje)
LAN	**L**ocal **A**rea **N**etwork (Red de Area Local)
LaTeX	**La**mport **TeX** (sistema de tipografia TEXpor Lamport)
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol (Protocolo Ligero de Acceso a Directorios)
LMS	**L**earning **M**anagement **S**ystem (Sistema de Gestion de Aprendizaje)
LO	**L**earning **O**bject (Objeto de Aprendizaje)
LOC	**L**ine **o**f **C**ode [measurement] (medida, linea de codigo)
LOR	**L**earning **O**bjects **R**epository (Repositorio de Objetos de Aprendizaje)
LTI	**L**earning **T**ools **I**nteroperability (Interoperabilidad entre Harramientas de Aprendizaje)
MOOC	**M**assive **O**nline **O**pen **C**ourse (Curso Online Masivo Abierto)
OAS	**O**pen **A**doption **S**oftware (Software de Adoptacion Abierta)
PPTP	**P**oint to **P**oint **T**unneling **P**rotocol (Protocolo de Tunel de Punto a Punto)
RHEL	**R**ed**H**at **E**nterprise **L**inux
SSH	**S**ecure **SH**ell (Inteprete Seguro de Commandos)
SSL	**S**ecure **S**ocket **L**ayer (Capa de Puertos Seguros)
TeX	Sistema de **T**ipografia por Donald Knuth
TLS	**T**ransport **L**evel **S**ecurity (Seguridad en la Capa de Transporte)
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol (Protocolo para Transferencia Simple de Correo)
SMTPS	**SMTP** over **S**SL (SMTP atravez de SSL)
UTPL	**U**niversidad **T**ecnica **P**articular de **L**oja
VCS	**V**ersion **C**ontrol **S**ystem (Sistema de Control de Versiones)
VPN	**V**irtual **P**rivate **N**etwork (Red Privada Virtual)
WYSIWYG	**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (Lo Que Ves Es Lo Que Obtienes)

[1] Significado abierto a debate

WWW **W**orld **W**ide **W**eb (Red Informatica Mundial)

Dedicado a las tres mujeres más importantes en mi vida, mi hija Valerie, mi esposa Paola y mi madre Diana…

Capítulo 1

Introducción

Hoy en día se encuentra una necesidad creciente para programadores, informáticos y personas que pueden leer y entender codigo debido a una tendencia a tratar de siempre automatizar a un mayor grado las tareas humanas con la tecnología. Actualmente la Universidad Técnica Particular de Loja cuenta con métodos clásicos y manuales para enseñar y calificar codigo que estudiantes programan lo cual no es beneficioso para los docentes, quienes pierden mucho tiempo realizando tareas que se podría realizar de forma automática, ni estudiantes quienes tengan menos atención de sus docentes debido a que tienen mucho codigo que calificar.

Dentro del mercado actual, existen varias sistemas que permiten escribir, ejecutar y probar código en línea. Repl.it se integra con sistemas que disponen el protocolo de integración entre sistemas de aprendizaje LTI (Repl.it y Neoreason, Inc., 2017) y puede calificar codigo de estudiantes en base a pruebas unitarias pero el mismo tiene un alto coste que no se escala bien y no está orientando tanto al aprendizaje o colaboración si no a evaluación (López, 2017). Io.livecode.ch vuelta es un sistema libre que utiliza contenedores de Docker para permitir la ejecución de código en línea, pero el problema de este sistema es que solo está orientado a enseñanza de programación, no a realizar pruebas en línea y además no implementa nada de seguridad para proteger el servidor contra usuarios maliciosos (Amin, 2017). Cloud9 IDE es otra herramienta popular para programar en linea ya que el mismo integra máquinas virtuales de Ubuntu con el mismo para permitir hacer pruebas en tiempo real (*Your development environment, in the cloud*), pero el mismo tampoco es factible por su falta de interoperabilidad con LTI y falta de enfoque ni en enseñanza ni en evaluación y calificación.

Como cualquier otro tipo de institución con fines comerciales, las universidades y otras tipos de instituciones educativas siempre deben estar buscando la manera en que pueden mantener su competitividad para dar la mejor educación posible al menor costo. La tecnología de hoy en día ha alcanzado una madurez en donde puede apoyar en temas de automatizar algunos procesos dentro de la enseñanza, evaluación y calificación de estudiantes y de la misma forma liberar el tiempo de professores para que puedan ayudar a aquellos estudiantes realmente necesitan un poco de ayuda adicional.

Frente este problema, se propone un sistema para editar código en línea que a su vez integra LMS externos (para autenticación y notas), un servidor de control de versiones externo (para la persistencia de código), y un servicio web de ejecución de código en línea de una forma segura, eficaz y eficiente (para dar un ambiente de ejecución y pruebas tanto para los usuarios del sistema como para calificar de una forma automática). A continuación se presenta la problemática para definir el contexto del tema frente el cual se espera resolver dentro de este trabajo de titulación.

1.1. Problematica

Dentro de la universidad, existe la necesidad de contar con una mejor forma de enseñar y evaluar conocimientos de programación en los estudiantes de las titulaciones de Sistemas Informáticos y Computación y Electrónica, especialmente quienes estudian a distancia. Para solucionar este problema, existen sistemas alternativos que se analizan en el capítulo 2, pero estos resultan demasiado costosos para su implementación completa con los recursos actualmente disponibles.

Se considera que cualquier sistema implementada como solución debe ser capaz de captar información acerca de su propio uso con un fin de ayudar con la administración del mismo, debe guardar de alguna manera segura y confiable codigo escrito dentro del mismo y debe proporcionar de las herramientas que requieren estudiantes y sus profesores para desarrollar, probar y calificar de manera eficaz, segura y eficiente codigo escrito en linea. Además como plataforma en línea se considera que se podría aprovechar para temas de colaboración como programación en pares y otras técnicas que apliquen los equipos de desarrollo para cumplir con sus responsabilidades de una forma paralela.

1.2. Metodología

La fase de investigación se consiste en realizar un breve recopilación de que hay dentro del entorno de despliegue para realizar un estudio del mismo. Además se realiza un estudio de aquellas tecnologías se utiliza dentro de la solución planteada. Para la investigación de todos los anteriores, se basa solo en fuentes confiables de los cuales muchas veces son fuentes académicos o comerciales con un fin de explicar alguna tecnología para vender la misma.

Después de la fase de investigación, donde se formula buenas bases y entendimiento del contexto del ambiente en que se encuentra se empieza una fase de análisis el cual se basa en una documento de visión que desglose las necesidades existentes para el sistema seguido por un ERS o especificación de requerimientos que demuestra de forma más exacta que debe tener capacidad de hacer el sistema.

Una vez que se tiene entendido qué capacidades debe tener el sistema, se procede a una fase de diseño en donde se realiza diagramas para definir la arquitectura, despliegue e interacción que el sistema tendrá con sus usuarios.

Con la definición de los requerimientos y el diseño del sistema como tal, lo que sigue es una fase de desarrolla el cual se plantea llevar bajo una metodología de desarrollo iterativo incremental marcado por prototipos que se revisan de forma contínua con los stakeholders principales del proyecto. De esta forma se puede mantener un poco de agilidad en el desarrollo y de esta manera, desarrollar un producto que cumple con las necesidades de los usuarios mientras que se mantiene dentro de los límites de tiempo y recursos preestablecidos.

Además se propone tener solo iteraciones (basados en entregables) que duren máximo un mes, de lo cual solo se lo propone para los dos entregables más grandes; el subsistema de ejecución de código por su complejidad y el capítulo seis que se trata del despliegue de la aplicación final. Todos los demás iteraciones son de dos semanas para tratar de mantener el proyecto en un estado fluido de actividad y entrega de entregables de forma contínua.

Las pruebas se llevaran igual en fases donde primero en una prueba alfa se integra con sistemas reales para probar temas de integración. En base a los resultados de estas pruebas se realiza recomendaciones y mejoras previo a la próxima fase de

pruebas que son constituidos por unas pruebas beta donde se prueba la aplicación en un entorno real con usuarios finales, tanto estudiantes como profesores para que puedan dar retroalimentación previo al despliegue de una versión final.

Finalmente se trabaja juntamente con los que serán encargados de mantener la aplicación para el despliegue del mismo. Esto se lleva en fases en donde se levantan primero las dependencias necesarias como bases de datos y servicios externos entre otros previo al levantamiento y configuración del sistema. La última fase del despliegue consiste en levantar los proxies y servicios auxiliares que ayuden mejorar temas de seguridad y rendimiento a la aplicación desplegado.

1.3. Objetivos

El objetivo principal es mejorar la enseñanza y evaluación de estudiantes de programación de la Universidad Técnica Particular de Loja. Para cumplir este objetivo se propone un sistema de editar y probar código en línea que ayuda a docentes calificar código que escriben sus estudiantes en base a pruebas unitarias e integra funcionalidades de sistemas LMS institucionales en adición a servidores de control de versiones institucionales. Para alcanzar este fin, también se plantea cumplir con algunos objetivos específicos en vía al principal.

- Autenticar contra LTI de los sistemas LMS institucionales. Esto permite que los estudiantes pueden autenticarse dentro del sistema sin crear nuevos cuentas de usuario con los nuevos nombres de usuario y contraseñas que el mismo involucra.
- Proveer la capacidad de poder ejecutar código en línea. Con esta funcionalidad, el sistema se acerca más a un entorno completo de desarrollo donde estudiantes requieren un mínimo de programas instalados en sus equipos personales.
- Autocalificar código en base a pruebas unitarias. El fin de esta característica es que el sistema sea un apoyo a los docentes en ayudarles automatizar y por lo tanto optimizar su flujo de calificar para que tengan más tiempo para estar enseñando y ayudando sus alumnos. Las mismas pruebas unitarias, por la forma en que automatizan la forma en que se pueden usar para calificar código en base a las funcionalidades que tiene y cómo reacciona frente ciertas entradas, también hace el proceso completamente objetivo debido a que elimina el factor humano del proceso.
- Sincronizar las notas con sistemas LMS institucionales a través de LTI. Eso dejará que estudiantes pueden revisar instantáneamente su nota una vez que está publicado y también que professores no tienen que estar transfiriendo notas entre sistemas distintas.
- Utilizar servidores de sistemas de control de versiones de la institución para recolectar código en los mismos. Eso permitirá, de una forma transparente y sin mayor interacción del usuario, ver un historial de código escrito en la plataforma y también servir de respaldo continuo del trabajos que realizan los estudiantes.
- Recolectar datos para la toma de decisiones estratégicas. Esta característica permite a la administración de la aplicación sea la universidad o equipo de mantenimiento puede sacar datos y estadísticas de uso de la aplicación para en base

a ello realizar análisis de cómo se está siendo usado, que son los comportamientos de los estudiantes y otras decisiones estratégicas que se pueden tomar en base a los datos de uso del sistema.

1.4. Resultados Esperados

En el presente trabajo de titulación pretende obtener los siguientes resultados:

Sistema para Editar Código en Línea que permite a los estudiantes y sus docentes tener una herramienta para mejorar el proceso de enseñanza, aprendizaje y evaluación de la programación.

Sistema para Ejecutar Código en Línea que se integra con lo anterior y facilita las pruebas que requieren las estudiantes o profesores con respeto a algún código al cual tengan acceso.

Repositorio de datos recolectados para la toma de decisiones estratégicos que permite a la titulación ver cómo se está utilizando el sistema y en base a ello planificar decisiones futuras.

Manual de Programador que sea una ayuda para futuros desarrolladores que deseen extender la funcionalidades del sistema inicial.

Manual de Usuario que apoya a estudiantes y docentes que quieren aprender a usar el nuevo sistema.

Manual de Mantenimiento/Administrador que ayuda al equipo de mantenimiento poder mantener la aplicación al corto, mediano y largo plazo.

1.5. Organización del Documento

Dentro del capítulo uno, se introduce el tema y problemática, se organiza el documento y se plantea la metodología y objetivos que guían el proyecto desde su inicio hasta su fin.

En el capítulo dos, se ve el estado del arte que contenga un análisis de trabajos relacionados y sistemas similares a la que se plantea desarrollar. También dentro del capítulo dos se encuentra un subsistema del estado del arte en la forma de marco teórico, el cual desglosa la situación actual y permite ver la teoría de las tecnologías de apoyo para el sistema final.

En el capítulo tres, se da un análisis profundo del problema para entenderlo de forma completa previa a las fases siguientes. Dentro del mismo capítulo, en base al análisis, se plantea un diseño para el producto final, el cual se refleja más adelante en el desarrollo durante el curso del capítulo cuatro.

A lo largo del capítulo cuatro, se lleva al lector por el proceso de desarrollo. Por lo tanto el mismo se divide en los varios módulos que se necesitan desarrollar para el proyecto; un módulo de autenticación LTI que permite a los estudiantes autenticarse contra sistemas LMS ya existentes, un módulo de editar código en línea, un módulo de persistencia de código en servidores de control de versiones institucionales, un subsistema de ejecución de código en línea, un módulo de pruebas unitarias para calificaciones automáticas y un módulo para sincronizar calificaciones con sistemas LMS institucionales.

El quinto capítulo se basa en la fase anterior de desarrollo para planificar y realizar pruebas de funcionamiento e integración. De esta manera se encuentran las fallas existentes para resolver los mismos antes de seguir con el despliegue en el sexto capítulo.

A continuación, en el capítulo sexto se planifica y se despliega la aplicación final para que pueda empezar su vida útil dentro de la universidad.

Dentro del séptimo capítulo se concluye el trabajo realizado en base al cumplimiento de los objetivos planteados dentro del capítulo uno.

En el octavo capítulo se da recomendaciones e ideas para trabajos futuros en base al proyecto realizado y para la evolución continua del producto final.

Al final de todos los capítulos anteriores se encuentran los anexos con documentos de apoyo como un manual de usuario tanto para profesores como alumnos, un manual de programadores para los que deseen extender el sistema al futuro y un manual de mantenimiento para la administración que se encarga de mantener la aplicación al largo plazo. Adicionalmente se incluye de donde se puede descargar el código de la aplicación como un anexo más al final para asegurar su disponibilidad para futuros desarrolladores que deseen extender las funcionalidades del sistema.

Capítulo 2

Estado del Arte

Sin la ciencia, no podría existir la ingeniería. Sin investigación y curiosidad acerca del entorno, no podría existir innovación y creatividad frente los problemas que se da. Es por tal motivo que el siguiente capítulo trata de un estudio del entorno en cuanto tecnologías en adición a trabajos e investigaciones previos que han sido realizados por terceros para en base a ello dar contexto a la investigación actual y aprender de lo bueno y malo de cada ascendente.

En la sección 2.1 de este capítulo, se presenta el marco teórico donde se realiza una investigación y estudia previa a las tecnologías y otros aspectos de entorno previo al planteamiento de una solución al problema introducido previamente. En la sección 2.2, se sigue con un estudio de trabajos relacionados a las temas abarcados en este trabajo de titulación mientras que en la sección 2.3 se realiza un estudio de sistemas similares en funcionalidad. Y finalmente se cierre el capítulo con la sección 2.4 donde se de una discusion y conclusion a todo lo analizado y presentado del capítulo con comparaciones de los mismos.

2.1. Marco Teorico

Para llevar a cabo de forma exitosa el trabajo de titulación y entender el contexto del ambiente en que sea implementado es importante establecer una línea base de conocimiento que se ve a continuación y dividido de la siguiente forma:

Aspectos de Propiedad Intelectual que se topa con las licencias y culturas de código abierto y libre que se encuentra actualmente en el entorno.

Aspectos Ambientales del Entorno de Desarrollo se trata de dar mejor contexto al entorno en el cual se encuentra la aplicación planteada. Esto incluye pero no está limitado a conceptos teóricos, aplicaciones y protocolos.

2.1.1. Aspectos de Propiedad Intelectual

Hoy en día, en un mundo cada vez más conectado y con cada vez más informacion compartida entre personas distintas, es importante conocer bien temas de derechos de autor y propiedad intelectual, no sólo para desarrollar y despues dar licencia a un trabajo de titulación, si no también para entender lo que se puede y no se puede hacer dentro de un entorno de desarrollo que cada vez involucra más algún codigo o trabajo que fue desarrollado alguna licencia abierta o libre.

Software Libre

Software Libre es un movimiento sociopolítico que busca liberar códigos fuentes. Se divide en dos campos que son: los de Open Source (Fuentes Libres) que quieren

liberar código fuente sólo en base a los méritos de desarrollo colaborativo y los de Free/Libre Software (Software Libre) que buscan liberar código fuente en base a ciertos derechos de compartir, colaborar y tener control de lo que hacen sus dispositivos que se les pertenecen (su propiedad) para todos los usuarios que ocupan el software (Stallman, 2016a) (Stallman, 2016c). En el contexto de esta tesis, Software Libre se refiere al segundo campo, no al primero. Para el mismo, la fundación GNU con su fundador Richard M. Stallman define 4 libertades mínimas que deben ser cumplidas para constituir Software Libre y los cuales se los enumera desde el 0 hasta el 3 (Stallman, 2017a) (Stallman, 2016b):

0. La libertad de ejecutar el programa para cualquier propósito que desee
1. La libertad de estudiar cómo funciona el programa y poderlo modificar como desee. Un requisito para esta libertad es que el usuario tenga acceso al código fuente original.
2. La libertad de distribuir copias del programa original con terceros.
3. La libertad de distribuir copias de sus versiones modificadas a terceros. Un requisito para esta libertad es que el usuario tenga acceso al código fuente original.

Donde el software no cumpla con uno de los anteriores, ya deja de ser considerado libre. Sólo por ser software libre no significa que no puede ser comercializado, únicamente se requiere nuevos modelos de negocio los cuales si se los puede encontrar en uso diario (Stallman, 2016b). Como sociedad, Stallman argumenta, es importante proteger esas libertades porque avanzan la humanidad, como la libertad de expresión, ya que las libertades que se proponen proteger, buscan sostener una sociedad que trabaja por el bien de todos, no de solo unos pocos que saben más de la funcionalidad de ciertos aspectos tecnológicos (Stallman, 2016c). En la época actual de gobiernos y empresas que espían sin vergüenza, se ha revivido el movimiento político y se ha convertido en algo más necesario debido a que todos tenemos y queremos nuestro derecho a la privacidad (Stallman, 2017a).

OAS OAS o Adopción de Software Libre es la tendencia que hay en el mundo empresarial de adoptar soluciones de tecnologías abiertas como los que ofrecen el mundo de software libre ya que las mismas pueden llegar a ser superiores en cuanto su eficacia y costo, que lo que ofrece su competencia comercial. Se estima que más del 78 % de instituciones usan software de Fuentes Abiertas y menos del 3 % indican que no utilizan nada de software de fuentes abiertas. Eso demuestra un enorme mercado creciente y emergente a nivel mundial (*The Next Wave in Software: Open Adoption Software (OAS)*).

GNU El proyecto GNU fue lanzada por el señor Richard Mathew Stallman en el año 1983 con un fin de desarrollar en comunidad un sistema operativo que podría reemplazar los sistemas operativos UNIX de aquella época con algo que sería un 100 % software libre (Stallman, 2014). Pero nunca terminaron todo el proyecto, específicamente el núcleo del sistema operativo, conocido como GNU Hurd, porque antes llegó otro núcleo desarrollado por un joven estudiante de sistemas llamado Linus Torvalds quien llamó su clon libre de Unix, Linux, y de esta forma, unida con todo lo que se había desarrollado el proyecto GNU, se formó un sistema operativo completamente libre (Stallman, 2017b).

GPL Aunque es la licencia más común en proyectos de software libre y usada para una mayoría de ellos, especialmente dentro del proyecto GNU, la GPL o GNU Public License no es la única licencia libre que existe (Free Software Foundation, 2016). Fue diseñada específicamente en la década de los 1980s para proteger la libertad de los usuarios de software (Stallman, 2016c). La última versión, GPLv3 fue publicada en Junio de 2007 y construye encima de versiones anteriores para tratar de asegurar de mejor manera las 4 libertades que tiene que garantizar cualquier licencia que quiere ser software libre (Smith, 2014) (Free Software Foundation, 2007).

2.1.2. Aspectos Ambientales del Entorno de Desarrollo

El contexto dentro del cual se desarrolla cualquier proyecto y sobre todo conocer las tecnologías existentes dentro de un ambiente en los proyectos tecnológicos, son puntos importantes que se debe tomar en cuenta previo a la ejecución del mismo. Es por tal motivo que a continuación se presenta algunas de las tecnologías con que se va a encontrar el proyecto y la teoría detrás de ellos.

Tipos de Recurso de Aprendizaje

CUADRO 2.1: Tipos de Recurso de Aprendizaje.

Concepto	Definición
LA	LA es un activo de aprendizaje el cual representa un archivo que no llega al nivel de un objeto de aprendizaje. También se los guarda en repositorios de objetos de aprendizaje (*About Learning Repository*).
LO	LO es un objeto de aprendizaje y es un recurso que ayuda a los estudiantes con su aprendizaje, por ejemplo un tema (que se puede enseñar con diapositivas, enlaces, documentos, etc), pruebas, exámenes, etc (*About Learning Repository*).
LOR	LOR es un repositorio de objetos de aprendizaje unidos por docentes y otros profesionales educativos para enseñar a sus alumnos de la mejor forma posible (*About Learning Repository*).

MOOC

MOOCs o Cursos Online Masivos Abiertos son cursos abiertos en línea que guían estudiantes paso a paso en su aprendizaje sin límites como costos, fronteras o culturas. Los mismos usan expertos del área en cuestión para dar el mejor calidad de educación possible a través de videos, pruebas, foros, redes sociales, charlas y artículos que promuevan debate y reflexión (*What is a MOOC? Massive Open Online Courses Explained*).

CMS

Un sistema de gestión de cursos es una colección de herramientas que ayudan proveer un ambiente en línea para interacciones entre integrantes de una clase. Esto puede incluir espacio para publicar anuncios, recolección de deberes, gestión de notas tanto para que docentes puede calificar sus alumnos y los mismos alumnos pueden ver sus notas en tiempo real, integración con sistemas de correos institucionales, chat en vivo y foros que permiten responder a entradas previas (*Course Management Systems*).

LMS

Según Mindflash, un vendedor de sistemas educativos (*Mindflash Learning Management*), los LMS, o Sistemas de Gestión de Aprendizaje, son herramientas muy útiles para instituciones educativas y otras entidades económicas que lo apoyen para organizar y gestionar recursos educativos, estudiantes, docentes y cursos. Las mismas son un aspecto crítico de la tendencia que tiene el sector educativo a digitalizarse que se puede ver en la aproximadamente 600 LMS que existen para elegir. Características comunes entre ellos incluyen: (*What is an LMS?*)

- Listas de Estudiantes para tomar asistencia entre otras interacciones que puede haber entre estudiantes y sus profesores.
- Matrículas
- Gestión de Documentos
- Soporte para Educación Distribuido; por ejemplo alumnos y/o docentes a distancia
- Calendarios de Curso para dar a conocer cronogramas, fechas para pruebas, exámenes y deberes.
- Interacción Estudiantil a través de correo electrónico, foros y chat.
- Comprobación de Conocimiento de Alumnos con pruebas y exámenes.
- Sistemas de Calificación que pueden ser automatizados (en casos objetivos), manuales por parte del docente (para casos más subjetivos) y híbridos cuando un profesor ofrece ambos casos en un curso.

Moodle Moodle es un LMS de software libre, licenciada bajo el GPLv3 (Dougiamas y Free Software Foundation, 2016), que ofrece los caracteristicas típicas de un LMS como herramientas de colaboración (Wikis, Foros, Glosarios, etc), gestión de y interacción con estudiantes (notificaciones, gestión de progreso y notas, etc), en adición a gestión de usuarios y matrículas (Moodle Community, 2016). Actualmente es lo que usa la UTPL con sus Docentes y Alumnos a través del Entorno Virtual de Aprendizaje (López, 2017).

OpenCampus OpenCampus es un sistema para instituciones educativas para ayudar a gestionar sus estudiantes y procesos a través de una combinación de OAS y software propietario con un fin de proveer mejor acceso a recursos educativos a todos (*Technology*), el mismo integra sistema de gestión de campus (CMS), gestión de escuelas de postgrado (GSM), sistema de gestión de aprendizaje (LMS), gestión de escuelas de medicina, sistema de aprendizaje electrónica, gestión de salud en el lugar de trabajo, gestión de objetivos, sistema de gestión de información estudiantil, gestión de clases, sistema de gestión de dato de investigación, sistema de gestión de ensayos clínicos, gestión de becas de investigación y gestión de laboratorios (*OpenCampus*). De esta manera ofrece características comunes de los LMS como exámenes, cursos en línea, rastreo académico para docentes, alumnos y administradores (*Technology*) (*For Universities*). La UTPL está ocupando esta tecnología actualmente para ofrecer "cursos en línea... de forma abierta y libre" (UTPL, 2017c).

Open edX Una organización sin fines de lucro famosa por sus cursos abiertos masivos (MOOCs), EdX, se ha liberado con una licencia libre su plataforma bajo el nombre Open edX. El mismo ofrece:

- CMS para gestionar varios cursos
- LMS para presentar contenido a estudiantes y profesores
- Open edX Studio para el diseño de cursos desde su contenido y políticas de calificación hasta el horario que seguirá y el equipo de docentes.
- Un plataforma para MOOCs
- XBlock para construir arquitecturas de componentes donde docentes mismos pueden agregar funcionalidad a su curso sin meterse mucho al resto del sistema.
- Foros
- Recolección de Datos para la toma de decisiones estratégicas

Según la documentación del mismo, Open edX utiliza una arquitectura de NGinx con Gunicorn y Django. Para el despliegue del mismo en entornos de desarrollo se ofrece una máquina virtual preconfigurada denominado "DevStack" (*About Open edX*). Como tecnología de backend, el OpenCampus de la UTPL utiliza Open edX (López, 2017).

LTI

LTI es un estándar para asegurar interoperabilidad entre sistemas educativos. Fue diseñada y es gestionado por IMS Global Learning Consortium (*Learning Tools Interoperability® Background*), un conjunto de instituciones y organizaciones que desean mejorar procesos educativos (*Contributing Members, Affiliates, and Alliance Participants*), con el fin de simplificar el proceso de interconección de aplicaciones de aprendizaje (conocidos en el estándar como Herramientas [de aprendizaje]), como los que saben ofrecer terceros, con ambientes educativos (conocidos en el estándar como Consumidores de Herramientas [de aprendizaje]) que pueden ser sistemas de gestión de aprendizaje (LMS), portales de aprendizaje, repositorios de objetos de aprendizaje (LOR), o otros ambientes educativos (*Learning Tools Interoperability® Background*). Moodle ofrece una implementación estándar para comunicarse con sistemas de terceros (Moodle Community, 2016).

Sistema de Control de Versionamiento

Un sistema de control de versiones mantiene y organiza un historial de versiones de uno o más archivos con el fin de poder intercambiar entre versiones rápidamente. Son muy usados para quienes trabajan con código fuente y también artistas (Chacon y Straub, 2014a) que necesitan llevar un historial de sus obras de arte, pero su utilidad no termina allí ya que son diversos los grupos que pueden aprovechar los beneficios que ofrecen estos sistemas. El sistema màs común que sabe usar los seres humanos actualmente es copiar archivos que deseen versionar, típicamente dándoles una nueva ubicación o nombre dado que es una solución muy simple para lograr versionar, pero eso puede ser peligroso ya que es muy fácil trabajar sobre una versión incorrecta, por lo que se inventaron los sistemas de control de versionamiento,

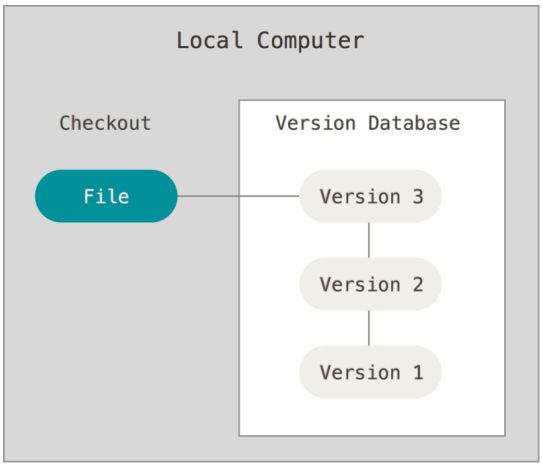

FIGURA 2.1: Sistema de Control de Versionamiento Local. Fuente: (Chacon y Straub, 2014a).

programas especializados en ayudar los seres humanos llevar una historial de versiones en uno o más archivos.

Inicialmente esos eran sumamente locales, como RCS, que mantiene una base de datos con todas las versiones (almacenadas como los cambios que hubieron en comparación con la versión anterior, una práctica conocida como respaldo incremental) y en base a eso permitir acceso a la versión actual o cualquiera de los otros guardadas previamente, mira figura 2.1. (Chacon y Straub, 2014a).

Después se dieron cuenta que seria bueno poder colaborar sobre las mismas versiones entonces se inventó lo que se conoce como Sistemas de Control de Versionamiento Centralizados los cuales permiten un alto grado de control y seguridad sobre quién puede editar y cuando puede editar que dentro de la base de datos de contenido ya que las mismas siguen un modelo arquitectónico cliente-servidor (Chacon y Straub, 2014a). Eso ha resultado que en empresas grandes que necesitan desarrollar sistemas grandes en paralelo con un cierto grado de seguridad, les resulta bien usar un sistema de control de versionamiento de esta clase aunque esto está cambiando ya que las empresas están dándose cuenta que pueden optar por una solución híbrida (Rose, 2015). Ejemplos de estos CVCS son CVS, Subversión y Perforce. Pero el problema viene a ser que todo el repositorio tiene un solo punto de falla porque si cae el servidor o le pasa algo, se pone en riesgo la productividad continua de quien colabora y en el peor de los casos, la integridad del repositorio. Esta propiedad se puede ver en la figura 2.2 (Chacon y Straub, 2014a).

Con la explosión de desarrollo de software libre, hubo una necesidad de poder trabajar grandes cantidades de personas desconocidas en paralelo (Raymond, 1999), y para resolver este problema y también la de integridad de datos que presenta el único punto de falla de los Sistemas de Control de Versionamiento Centralizado, se inventó Sistemas de Control de Versionamiento Distribuidos como Git, Mercurial, Bazaar y Darcs. Cada repositorio es un espejo de los demás en una arquitectura distribuida o red de iguales. Con esta arquitectura (mira figura 2.3), ningún nodo viene a ser un punto de falla ya que se puede espejar entre cualquiera de ellos debido al hecho de que todos tienen una copia local de todo lo que tienen los demás (Chacon y Straub, 2014a).

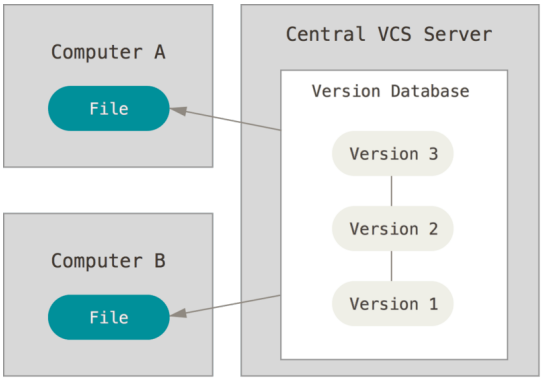

FIGURA 2.2: Sistema de Control de Versionamiento Centralizado. Fuente: (Chacon y Straub, 2014a).

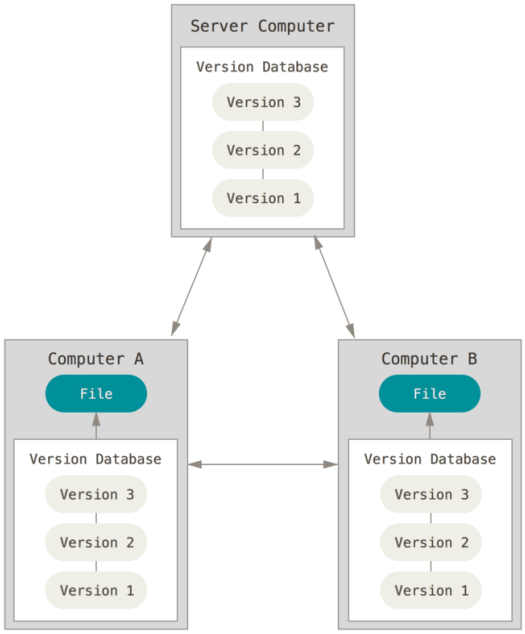

FIGURA 2.3: Sistema de Control de Versionamiento Distribuido. Fuente: (Chacon y Straub, 2014a).

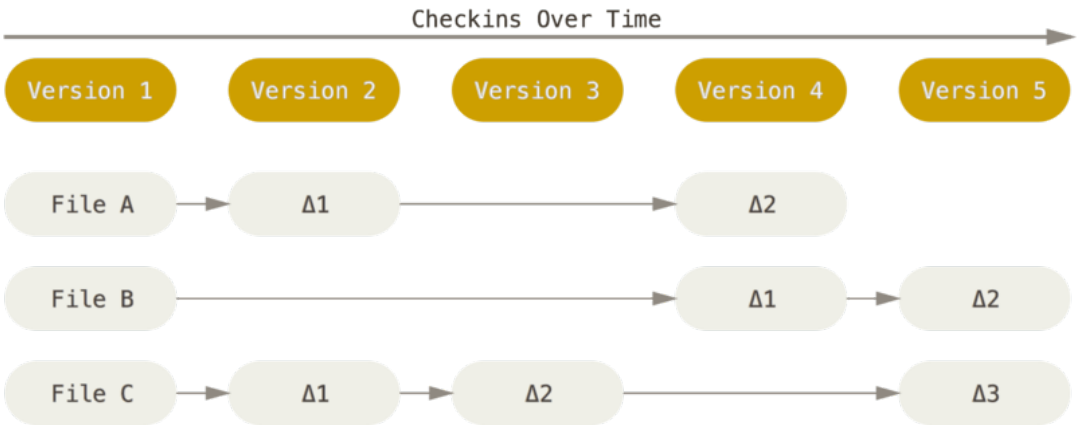

FIGURA 2.4: Figura (VCS-Incremental). Un Sistema de Control de Versiones llevado de forma incremental (la manera en que muchos VCS funcionan). Fuente: (Chacon y Straub, 2014b).

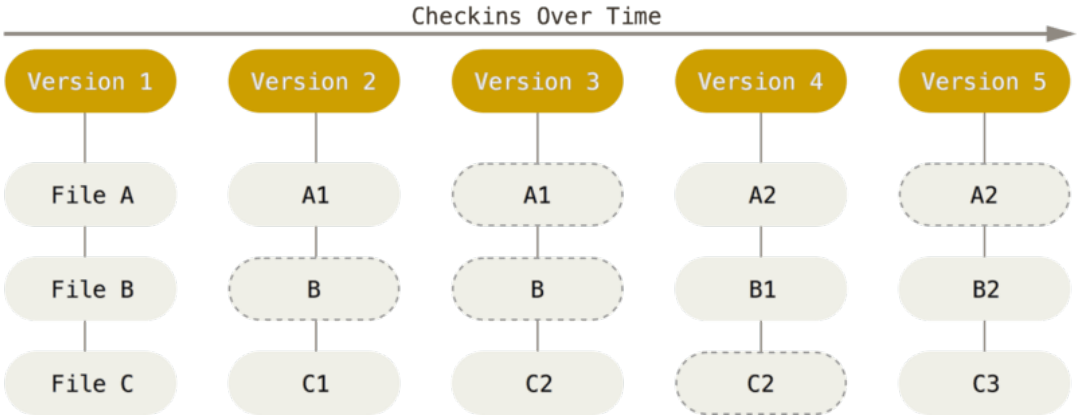

FIGURA 2.5: Un Sistema de Control de Versiones llevado con archivos enteros (la manera en que funciona Git). Fuente: (Chacon y Straub, 2014b).

Git

Git es un sistema de control de versiones distribuido que a diferencia de otros VCS guarda con cada versión una copia entera de los archivos modificados (otros sistemas de control de versionamiento saben guardar solo las diferencias entre versiones de forma incremental, mira la diferencia entre la figura 2.4 y la figura 2.5) (Chacon y Straub, 2014b).

Esto hace que Git se convierte más en como un pseudo sistema de ficheros que un simple VCS. Además significa mayor integridad de los datos guardados a costo de mayor consumo de almacenamiento. Trabaja de esta manera distribuida, es decir el repositorio existe en esta forma en todo lado donde se encuentra y por lo tanto la mayoría de operaciones de Git se puede realizar de manera local sin ninguna conexión de red (Chacon y Straub, 2014b). Volviendo al tema de integridad de datos, Git utiliza SHA-1 como algoritmo criptográfico para calcular los hash de cada cambio que se realiza en base a una combinación de qué información contiene el cambio,

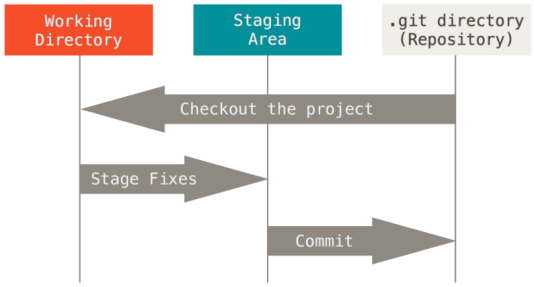

FIGURA 2.6: Los tres fases de Git y sus relaciones. Fuente: (Chacon y Straub, 2014b)

su metadata como autor, fecha, hora y fecha en adición a los SHA-1 de su(s) cambio(s) padre(s)[1] de tal forma que es difícil si no prácticamente imposible[2] para un atacante modificar el historial sin dejar evidencia de lo que se ha hecho. Por defecto funciona así (Chacon y Straub, 2014b), pero si uno desea asegurar la integridad del historial aún más, se integra Git con GPG para ofrecer mayor seguridad ya que con eso se puede firmar digitalmente cambios realizados para demostrar su autenticidad (Chacon y Straub, 2014c).

La mayoría de operaciones normales que se hacen en Git no eliminan datos ya guardados en el repositorio previamente lo cual lo hace una buena herramienta para personas que recien esta entrando al mundo de Sistemas de Control de Versionamiento y permite que se pueda recuperar de la mayoría de errores que se puede cometer y una recuperación casi garantizado de datos "perdidos". Eso es gracias en parte a la filosofía que se refleja en el diseño del mismo ya que una prioridad primordial era la integridad de los datos (Chacon y Straub, 2014b).

Cada repositorio de Git trabaja con tres fases que son el directorio de trabajo donde se realiza cambios, una área de preparación donde se marca cambios listos para guardar y una base de datos junto un sistema de ficheros donde se guardan los cambios. Al saltarse entre versiones, se sobreescribe los contenidos del directorio de trabajo con lo que se extrae del sistema de ficheros que tiene los cambios guardados. Esas tres fases y su relación se puede ver en la figura 2.6 (Chacon y Straub, 2014b).

GitLab GitLab es un servidor de Git ofrecido por GitLab Inc. Se ofrece en dos versiones: una versión gratis (Freyd y col., 2017) de código abierto liberado bajo el nombre GitLab Community Edition (GitLab CE) y una versión pagada que forma la línea principal de negocio de GitLab Inc. que lo venden bajo el nombre GitLab Enterprise Edition (GitLab EE) (Sijbrandij y col., 2017). También proporcionan hosting de las mismas versiones alojados por ellos en GitLab.com (alli se puede usar GitLab EE gratis (Nierop y col., 2017) (Freyd y col., 2017)) y en GitHost.io (donde se puede arrendar servidores para alojar instancias de GitLab CE o EE y su respectivo infraestructura de apoyo (GitLab Inc., 2017)) (Sijbrandij y col., 2017). Actualmente la Universidad Técnica Particular de Loja utiliza un servidor de GitLab CE como su instancia institucional de Git (UTPL, 2017d).

[1] La mayoría de commits solo tienen un padre, pero aquellos que unen varios historiales, tienen dos padres

[2] Un equipo de investigación de Google y CWI Amsterdam ha encontrado una vulnerabilidad que recién publicaron y les permite atacar SHA-1 y encontrar colisiones hash (Stevens y col., 2017)

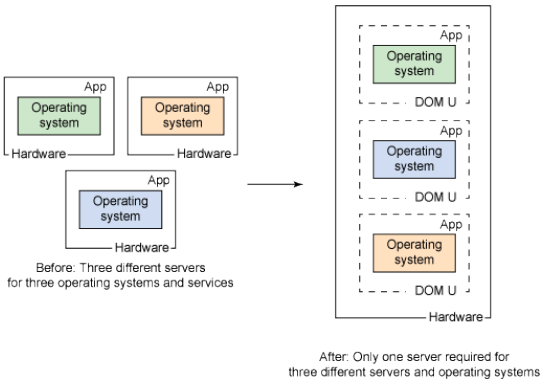

FIGURA 2.7: Antes y después de virtualizar aplicaciones. Fuente: (Tholeti, 2011).

Virtualización

Virtualización es la creación de recursos virtuales en software para usar los mismos en lugar de un recurso físico. Este puede ser usado en áreas diversas como aplicaciones, servidores, almacenamiento y redes. Según la empresa VMware, que se dedica a vender productos de virtualización, la misma es la estrategia más efectiva para reducir costos de TI mientras que se aumenta la eficiencia y agilidad de un negocio de cualquier tamaño. Llegan a esta conclusión porque es tan común que se ha convertido en estándar industrial que solo se ocupa en entre 5 % y 15 % de la capacidad de los servidores debido a que se les pone solo un sistema operativo y aplicación a la vez. Virtualización permite resolver estas deficiencias en permitir la división lógica de equipos y distribuir estas partes donde más se los necesitan para aumentar la tasa de eficiencia y reducir la cantidad de recursos físicos necesarios. Con esta clase de tecnología, se puede reducir y en algunos casos eliminar tiempo fuera de servicio. Las facilidades de portabilidad y agilidad de ajustar recursos hace que se puede reducir los tiempos y costos de administración. Se domina hipervisor un software que gestiona virtualización (*What is Virtualization?*).

Las Máquinas Virtuales Una Máquina Virtual es una sistema computacional entera con sistema operativo y aplicación corriendo sobre recursos virtuales. Por lo tanto cada máquina virtual es completamente aislada e independiente de los demás de tal forma que se puede tener varios sistemas operativos y aplicaciones corriendo sobre un solo servidor físico. Eso permite tener varias características importantes:

División de Recursos Se puede dividir recursos físicos entre varios sistemas operativos corriendo sobre la misma máquina.

Aislamiento Proveer mayor seguridad en prevenir o restringir acceso entre las varias máquinas virtuales.

Encapsulación de Persistencia Todo el estado de la máquina virtual se lo puede guardar en archivos, los mismos que pueden ser sincronizados entre varios equipos o ubicaciones.

Independencia de Hardware Se puede usar la máquina virtual en cualquier hipervisor que lo soporte.

Todo estas caracteristicas anteriores permiten la consolidación de servidores para utilizar menos servidores y cortar costos (*What is Virtualization?*).

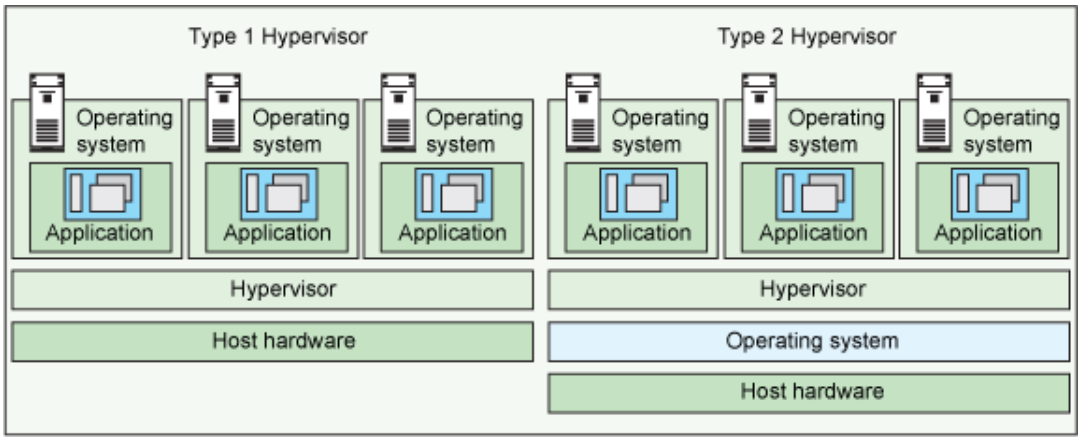

FIGURA 2.8: Tipos Clásicos de Hipervisor.

Fuente [IBM-Hypervisors].

Virtualización de Servidores Con virtualización de servidores, se puede reducir el número de servidores necesarios dentro de una granja, especialmente cuando los mismos son miembros de un cluster que realiza balanceo de recursos frente su carga para optimizar la eficiencia de toda la granja (*What is Virtualization?*).

Virtualización de Redes Con virtualización de redes se puede simular todo el tráfico y funcionamiento de una red sin mayor retardo. Se puede virtualizar interfaces de red, los switch, routers, cortafuegos, balanceadores de carga, VPNs y más. Como los otros tipos de virtualización, esta ofrece los mismos beneficios de independencia, costos y escalabilidad (*What is Virtualization?*).

Virtualización de Escritorios Virtualización de Escritorios se trata de usar recursos centralizados para dar escritorios de trabajo a usuarios remotos. Esto permite a cualquier organización reducir sus gastos tecnológicos debido a que el mismo permite menor inversión en tecnología para la misma cantidad de usuarios mientras al mismo tiempo facilita temas de administración de los equipos de los empleados en una organización y seguridad de la información de la misma (*What is Virtualization?*).

Hipervisores

Un hipervisor es una capa intermedia de software que gestiona la interacción entre lo virtualizado y hardware o otro software que ayuda realizar las peticiones de lo virtualizado (*What is Virtualization?*). De la misma manera, el mismo hipervisor se encargan de seguridad para asegurar que lo que sea virtualizado no pueda salir de su ambiente y atacar otras partes virtualizados o no virtualizados (VMWare, Inc., 2014) (Wilson, Day y Taylor, 2011).

Hipervisores de Tipo 1 Un Hipervisor de Tipo 1 se ejecuta directamente sobre hardware física (Tholeti, 2011) y por lo tanto puede llegar a conseguir mejor rendimiento y seguridad pero a un costo de requerir en muchos casos mayor configuración y administración para poder alcanzar su rendimiento óptimo.

Xen Xen es un hipervisor de tipo 1, que corre directamente sobre hardware. Es el único hipervisor de su clase que es completamente disponible bajo una licencia abierta y forma una base por muchas aplicaciones comerciales y no comerciales como virtualización de servidores, IaaS, virtualización de escritorios, aplicaciones de seguridad, y dispositivos embebidos. Dentro del mercado, es la tecnología detrás de las nubes más grandes (Xen Community, 2016).

Para el mismo, se considera las siguientes características:

- La implementación y uso de un micronúcleo que ocupa un mínimo de memoria y tiene interfaz externa limitado el cual ayuda proporcionar mayor seguridad y estabilidad en comparación con otros hipervisores. El hipervisor en si esta escrito en menos de 150 KLOCs que para un sistema operativo es muy liviano. Puede ser tan liviano porque no necesita tener conocimientos de operaciones de entrada/salida como redes o almacenamiento debido a que Dom0 se encargará de estos tareas. Al mismo tiempo, las demás máquinas virtuales no son privilegiados y siempre tienen que comunicarse con Dom0 para realizar cualquier operación con respecto al hardware físico. Por lo tanto la muerte del Dom0 es fatal para todas las máquinas que están en el Hipervisor.
- La arquitectura de Xen ocupa una maquina virtual que se domina Dom0 el cual contiene todos los drivers para el hardware y también la herramientas necesarias para ejercer control sobre todo el hipervisor. Estás funcionalidad son independientes del sistema operativo motivo por el cual que el sistema operativo residente del Dom0 puede ser Linux, una BSD, OpenSolaris, etc...
- Como los drivers que controlan hardware física existen dentro de una máquina virtual, se los consideran drivers aislados. En caso de que pase algo, solo hay necesidad de reiniciar la máquina afectada, el resto del sistema puede seguir funcionando normalmente.
- Con una tecnología llamada paravirtualización, en lugar de optimizar al nivel de hipervisor se optimiza los sistemas operativos que son virtualizados para de esta manera mejorar su rendimiento incluso en hardware que no tiene soporte para virtualización. (Xen Community, 2016).

Hipervisores de Tipo 2 Un Hipervisor de Tipo 2 se ejecuta por encima de un sistema operativo que le ayuda con su gestión interna como si fuera una aplicación más (Tholeti, 2011) lo que hace este tipos de hipervisores más lentos que hipervisores de tipo 1 (por tener un mayor número de capas y también por virtualizar partes del hardware) pero a su vez más fáciles de configurar y administrar. El mismo hecho de ejecutar por encima de un sistema operativo también vulnera la seguridad de todo el sistema en proveer un superficie de ataque más grande para actores maliciosos.

Qemu-KVM QEMU es un proyecto de software abierto con el fin de crear un emulador y virtualizador de procesadores de varias arquitecturas para una variedad de sistemas operativos. Permite el uso de backends como KVM o Xen para dar rendimiento casi nativo (*QEMU*). Aunque su despliegue normal requiere un sistema operativo, basado en Linux, intermedio debido a que es un módulo que se integra directamente en el núcleo de la misma sistema operativo, Wilson et. al. consideran KVM como un hipervisor de tipo 1 debido a sus características y implementación a bajo nivel cerca al nivel hardware física, mientras que su arquitectura lo hace aparecer como un hipervisor de tipo 2, que es como se lo está considerando aquí dentro

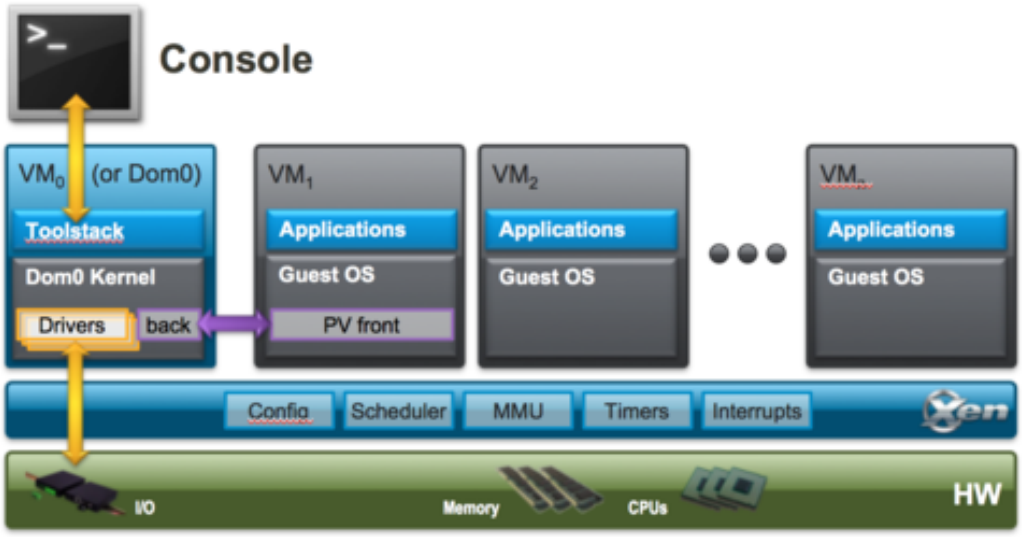

FIGURA 2.9: La arquitectura de Xen. Fuente: (Xen Community, 2016).

del contexto de este trabajo (mientras que otros autores lo consideran un hipervisor de tipo 1.5, ni uno ni el otro si no algo híbrido dentro de la taxonomía de hipervisores). KVM es la "hipervisor estratégica para IBM" debido a (1) su bajo costo por ser tecnología completamente abierta, (2) su alta evolución y madurez, (3) alta evolución y madurez de Linux (su tecnología fundamental atras), (4) eficiencia y rendimiento alto, (5) una comunidad de desarrollo muy activa y responsable frente fallos de seguridad, (6) la idea de que muchos ojos viendo código lo hace más seguro[3], (7) varias tecnologías de seguridad que integra que falta la competencia, (8) mayor flexibilidad que ofrece frente otros hipervisores competidores y (7) control que IBM puede ejercer sobre el proyecto de KVM (Wilson, Day y Taylor, 2011).

VirtualBox VirtualBox es un hipervisor para virtualización de arquitecturas estándar x86 y x86_64 tanto para uso casero como uso empresarial. Para ello ofrece muchas características, rendimiento optimizado, y un gran número de sistemas operativos que soporta tanto para virtualizar como para ser virtualizados todo bajo una licencia de software libre, la GPL versión 2 (*VirtualBox*). Su arquitectura de ser montado encima de cualquier sistema operativo y apoyarse en ello para realizar operaciones por parte de un sistema operativo y hardware virtualizado hacen este hipervisor un ejemplo clásico de un hipervisor de tipo 2.

Virtualización a Nivel de Sistema Operativo (Contenerización) Desde la antes de la incepción de los sistemas Linux, los sistemas de la familia Unix han tenido un mecanismo de aislamiento conocido como chroot el cual genera un espacio aislado de usuario para contener alguna aplicacion o varias aplicaciones mientras que todos siguen compartiendo el mismo sistema operativo o kernel por detrás. La tendencia de contenerización que existe hoy en día sigue por la misma línea pero lo implementa de una forma aún más avanzada. Eso permite a muchos usuarios que se desconfían mutuamente unos en otros utilizar el mismo hardware mientras que al mismo tiempo quedan completamente aislados unos de otros con menor impacto al rendimiento

[3]Uno de las características principales de Código Abierto

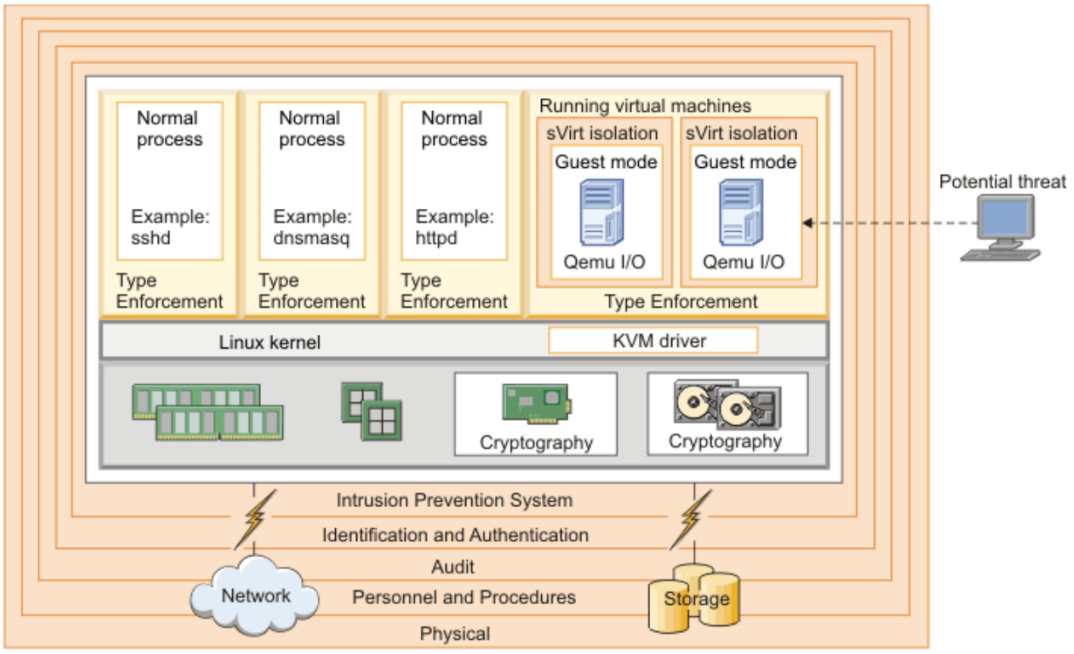

FIGURA 2.10: Arquitectura de KVM/QEMU. Fuente: (Wilson, Day y Taylor, 2011).

o una necesidad de hardware que soporta la carga adicional que virtualizacion genera (Teimouri, 2016), mira la figura 2.11. Algunos sistemas de este tipo de aislamiento que son populares hoy en día son:

- chroot
- Docker
- LXC
- LXD
- Linux-VServer
- OpenVZ
- Solaris Containers
- FreeBSD jail
- Hyper-V containers (Microsoft)
- Photon (VMware)
- vSphere Integrated Container (VMware)
- Windows Containers (a partir de Windows Server 2016)

Debido al hecho de que muchas veces cada máquina virtual necesita su propio hardware virtual y también sistema operativo, se termina consumiendo mucho más RAM y ciclos de CPU de tal manera que un servidor típicamente puede soportar dos a tres veces más servicios si están en contenedores en lugar de si son virtualizados (Teimouri, 2016). La diferencia se puede ver en la figura 2.12.

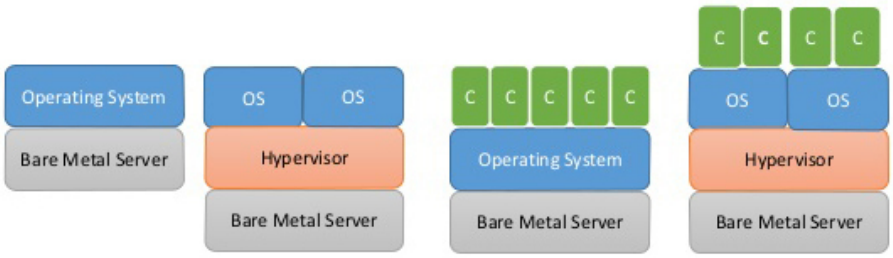

FIGURA 2.11: Un concepto básico de que son los contenedores (Teimouri, 2016).

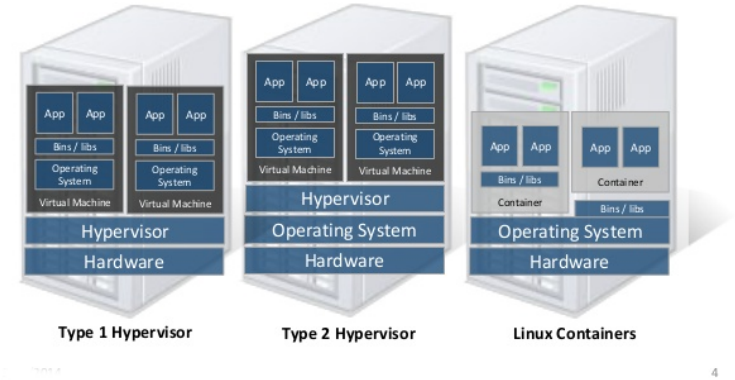

FIGURA 2.12: La diferencia que abarca contenedores de los tipos de hipervisores (Teimouri, 2016).

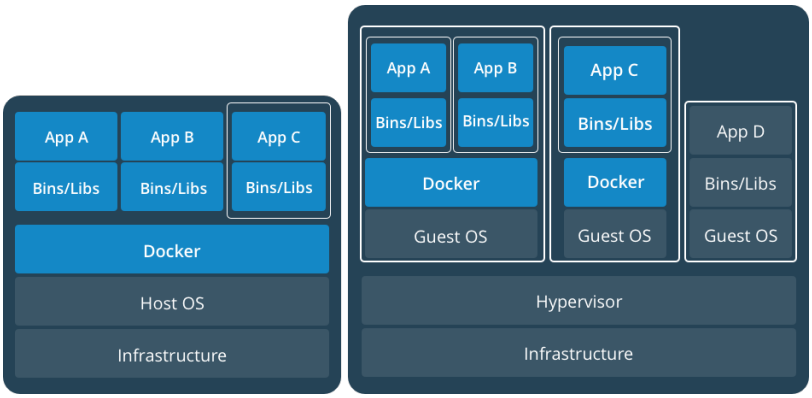

FIGURA 2.13: La diferencia entre contenedores nativas y dentro de una máquina virtual. (Docker Inc., 2017b)

Entre las ventajas de contenedores está que permiten tener ambientes aislados livianos para prevenir conflictos de dependencias y estandarizar versiones utilizadas en ambientes de desarrollo y producción mientras que al mismo tiempo requieren mucho menos recursos que una máquina virtual. Las mismas se pueden levantar casi instantáneamente a diferencia de máquinas virtuales que requieren tiempo para arrancarse. Como se puede ver en la figura (contenedores-vms), también se puede tener un motor de contenedores dentro de una máquina virtual, el cual ofrece beneficios como mayor aislamiento (seguridad) y flexibilidad al momento de migrar aplicaciones entre servidores (Docker Inc., 2017b).

La estandarización que ofrece la contenerización ayuda abstraer sistemas operativos que están en los servidores (permitiendo mayor flexibilidad de despliegue), permite mayor escalabilidad si la aplicación es diseñada para eso, es fácil de versionar dado que los archivos de configuración que describen los contenedores muchas veces son de texto plano y es muy orientado a arquitecturas compuestas por servicios como SOA o las arquitecturas de Microservicios debidos a que estas ya tienen componentes donde es fácil saber dónde poner la frontera de cada contenedor (Ellingwood, 2015).

Para tener seguridad adicional, también se puede ocupar contendores dentro de máquinas virtuales aisladas como se puede observar en la figura 2.13 (Docker Inc., 2017b).

Docker Docker es una plataforma más popular hoy en dia para contenedores. Se puede usar para eliminar problemas de dependencias entre equipos de desarrolladores, para aumentar la capacidad computacional de equipos y también para desplegar de una forma más ágil, rápida y segura. Con contenedores de Docker, toda la complejidad está dentro del contenedor, facilitando su compilación, la forma en que son compartidos y ejecutados, lo que hace fácil que cualquier persona que tenga los archivos de configuración necesarias puede levantar la aplicación en minutos en lugar de en horas (Docker Inc., 2017c).

La arquitectura de Docker permite funcionar con cualquier tecnología. La estandarización y facilidad que ofrece permite mejor colaboración entre miembros de un equipo. Por defecto Docker se trata de ser lo más seguro, flexible y extensible posible mientras que al mismo tiempo no necesitar cambios para que un proveedor de software no necesita realizar cambios ni casarse con la tecnología (Docker Inc., 2017a).

Protocolo de Túnel

Un túnel es una técnica de permitir acceso remoto a recursos en una red a los cuales normalmente no se tendría acceso desde afuera. Se puede realizar esto en la capa 2 de redes con los protocolos como L2TP, PPTP y L2F quienes buscan encapsular paquetes previo a su viaje entre redes y desencapsularlos en ambos lados del túnel. Puede ser una solución económica para todos los involucrados ya que permite montar varios VPNs en la misma infraestructura física (*Tunneling*). Los protocolos de túnel saben transmitir dos tipos de mensajes, aquellos paquetes que encapsulan datos y otros paquetes auxiliares que definen las reglas de reenvío (*What is a Tunneling Protocol?*). Por esta razón y también la tendencia de estos protocolos de proteger los datos que transmiten con encriptación resulta en una velocidad de transferencia de datos menor que simple envío directo. Algunos otros protocolos de encapsulación buscan reemplazar TCP y UDP en la capa 4 pero su funcionalidad de transportar paquetes entre redes, que de ninguna otra forma serían conectados, es igual y forma un

tipo de VPN. Los protocolos SSH y IPSec también tienen esas funcionalidades. Así se puede lograr reenviar puertos para utilizar servicios internos como portales internos de una institución o convertir una máquina con Linux/UNIX en un servidor gráfico para "terminales tontos" (Schluting, 2006) como los días clásicas de la infancia de UNIX (Geerling, 2014). Aunque su poder y seguridad lo ha hecho un herramienta común entre profesionales que trabajan con redes (Schluting, 2006), también por su misma naturaleza se puede ocultar tráfico y por lo tanto ahora es común que malware y actores maliciosos utilizan esos mismos protocolos para esconder su actividad y realizar actos malvados con un menor riesgo de ser detectados en tiempo real. Por lo tanto, para usuarios normales, se recomienda limitar el uso de estos protocolos (*What is a Tunneling Protocol?*).

Criptografía

Criptografía es una ciencia de proteger datos comunicados que han sido utilizado por miles de años para comunicarse de tal forma que el emisor y receptor pueden enviar y recibir mensajes pero personas intermedias no pueden entender los mensajes en tránsito. Tiene aplicaciones militares, financieros y también en el ámbito tecnológico, sobre todo para el internet, entre otras aplicaciones para los cuales ha sido usado a lo largo de la historia. Cifrar es el acto de ocultar datos y decifrar es la acción opuesta para devolverlos a su estado original (Cruise, 2012b).

Criptografía Asimétrica Con la introducción de redes y la necesidad de encriptar comunicaciones entre personas quienes nunca se ha habían reunido para llegar a un acuerdo de que usar de llave criptográfica para comunicaciones, se necesitaba una manera de llegar a este acuerdo sin que ningún agente intermedio también sepa la llave criptográfica resultante del acuerdo y por lo tanto se introdujo la necesidad de la criptografía asimétrica (Cruise, 2012a). Para cumplir con este requisito, el receptor genera una llave privada y llave pública, los cuales son operaciones inversas y publica su llave pública para todos que deseen comunicarse con el mismo. Ahora quienes quieren comunicar con el dueño de la par de llaves ocupan su llave pública para cifrar el mensaje y lo envían. Solo con la llave privada se puede descifrar el mensaje y así como el dueño es el único que tiene esta llave, sólo él puede leer los mensajes enviados (Cruise, 2012a).

SSH

En la infancia de la computación, solo habían computadores centralizados en instituciones grandes. Aún no existía el Internet ni las redes, estaban en su infancia. Poco a poco se introdujo la idea del terminal tonto y con ello el acceso por varios usuarios a estas mismas computadoras centralizados sobre un LAN institucional. De aquí proviene el concepto de control remoto y en aquella época de redes bien controlados y cableados, no hubo mucho riesgos de seguridad entonces por lo tanto el protocolo de esos tiempos fue Telnet, donde todos los datos, incluyendo usuarios y contraseñas son enviados en texto plano. Con el tiempo se volvió más económico y ubicó la tecnología de tal forma que, junto al Internet, se estaba empezando a usar Telnet en redes de desconocidos y generando situaciones potencialmente maduros para ataques clásicos de análisis o incluso modificación de paquetes en tránsito. Fue frente esta realidad que un Finlandés diseñó e implementó el protocolo SSH (Geerling, 2014). OpenSSH es una implementación abierta y libre (*OpenSSH Features*) del protocolo SSH, que como se define el protocolo, cifrar su tráfico para proteger contra

espionaje, robo de conexión y otros ataques que afectan otros protocolos de una naturaleza similar. El proyecto OpenSSH, para ofrecer la mayor funcionalidad posible tiene integrado funcionalidades de túneles seguros, varios métodos de autenticación, y opciones avanzadas de configuración. Adicional a esto ofrece una variedad de herramientas para la gestión de sus funcionalidades con clientes como los comandos ssh, scp y sftp, gestión de llaves con comandos como ssh-add, ssh-keysign, ssh-keyscan, y ssh-keygen, en adición a servicios en segundo plano como sshd, sftp-server y ssh-agent (*OpenSSH*). Hoy en día se puede utilizar SSH para cifrar conexiones en redes que de ninguna otra forma serían cifradas, reenviar tráfico de un servidor gráfico X11 (para tener varios terminales gráficos en una sola máquina servidor (Schluting, 2006)), reenvío de puertos sobre un canal seguro, fuerte mecanismos de autenticación como claves de único uso y llaves de criptografía asimétrica, reenvío de agentes de autenticación, interoperabilidad con otras versiones y implementaciones del protocolo SSH, y la opción de activar compresión y descompresión de datos en tiempo real para optimizar el consumo de recursos en red (*OpenSSH Features*) (OpenBSD, 2016).

Túnel de SSH SSH es un protocolo de túnel muy popular (*What is a Tunneling Protocol?*) ya que ofrece mucha flexibilidad para las mismas (*OpenSSH*) y alta seguridad desde la autenticación para iniciar la conexión y para los datos transferidos en dentro de la conexión (*OpenSSH Features*). Su facilidad de uso permite con rapidez levantar mini VPNs de un puerto para acceder a puertos de otros equipos los cuales normalmente no serían disponibles, por ejemplo acceder servicios que solo son disponibles en un intranet (Schluting, 2006).

SSL

SSL es un estándar tecnológico para cifrar tráfico (SSL.com Corp, 2016) (*What is an SSL Certificate?*) de protocolos que normalmente no son cifrados por ejemplo HTTP, SMTP e IMAP. Esto se hace con un fin de proteger datos sensitivos transferidos (en adición a asegurar su integridad (SSL.com Corp, 2016)) entre un cliente y servidor, los cuales normalmente se transfieren en texto plano y podrían ser leídos por cualquier agente intermedio. El proceso de comunicación se puede encontrar en la figura 2.14. Para iniciarse la conexión cifrada, primero el cliente pide que el servidor se identifica (1) con la llave pública que forma parte de su certificado (2). Una vez que lo tiene, el cliente procede a validar el certificado para ver si confía en su origen (a través de firmas digitales por autoridades centrales) y en caso de que haya confianza, se procede a generar una llave criptográfica para la sesión, lo cual se lo transmite cifrada al servidor con la llave pública del mismo (3). El servidor descifra la clave de sesión con su llave privada de su certificado, avisa el cliente que la transacción ha sido exitosa (4) y ambos proceden a transferir los datos adecuados, cifrados con la llave de sesión que tanto cliente como servidor comparten (5) (*What is an SSL certificate?*).

Para ayudar con el proceso de validación, el certificado SSL sabe contener detalles como la llave pública del servidor, el nombre de dominio, nombre de empresa, dirección, ciudad, estado o provincia y país. Es firmado por un autoridad de certificados para dar confianza a los clientes que revisarán el certificado (SSL.com Corp, 2016). Antes de firmar un certificado, una autoridad de certificados primero valida la identidad del servidor que ha realizado la petición para el certificado para que más adelante puede confirmar a clientes que es el mismo servidor (*What is an SSL Certificate?*).

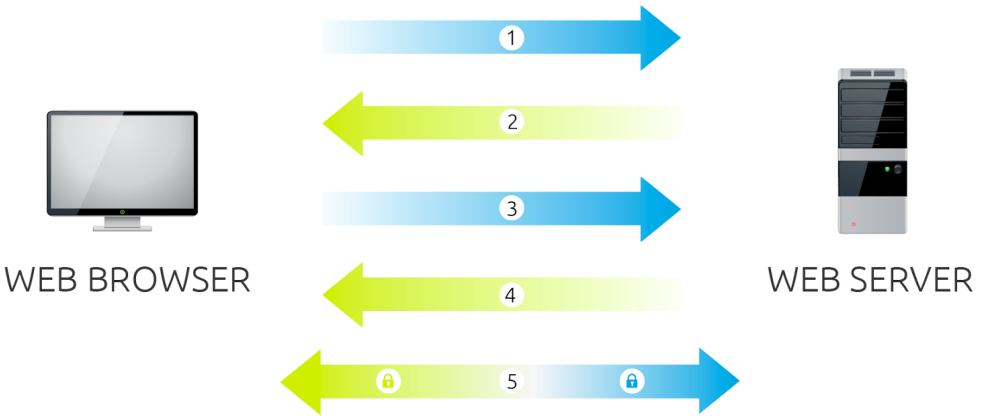

FIGURA 2.14: El proceso para comunicación con el protocolo SSL.
Fuente: (*What is an SSL certificate?*)

TLS Cuando llegó la versión 4.0 de SSL, se cambió de nombre a TLS versión 1.0 (*What is an SSL certificate?*) para señalar un fuerte cambio de seguridad en cómo se manejan los casos de uso que pueden causar fallos de seguridad en antiguas versiones de SSL como POODLE, DROWN y BEAST que dejan la seguridad que promete SSL inútil y peligroso (ya que las mismas prometen seguridad falsa) (Kangas, 2016) (GlobalSign, 2016).

2.1.3. Tecnologías de la Web

El cuadro 2.2 presenta conceptos de varias tecnologías que existen en la web hoy en dia.

CUADRO 2.2: Algunos de las tecnologías que son importantes en la funcionamiento de la web.

Concepto	Definición
URI	Una URI identifica de forma única un recurso que existe en una red como el Internet (*URI - Uniform Resource Identifier*). Como se define en el RFC 3986, debe consistir de forma genérica en un: [protocolo de acceso][usuario][clave][dominio][ruta de recurso][parámetros de acceso al recurso] Donde usuario, clave son opcionales y la ruta del recurso y sus parámetros también pueden ser opcionales o innecesarios en casos de protocolos que no requieren el mismo (Berners-Lee, Fielding y Masinter, 2005).
HTTP	HTTP es el protocolo con el cual se intercambia información como peticiones y datos con páginas web en el World Wide Web. Usa un esquema de códigos de error para comunicar tipos de sucesos que pueden ocurrir con el uso del protocolo (ComputerHope, 2017). Dentro del protocolo HTTP, se usa verbos que son comandos que dicen que tipo de operación quiere hacer un cliente con relación a un servidor web. HTTP es conocido como un protocolo sin estados ya que dentro del protocolo se desconoce acciones anteriores que se ha tomado en la interacción cliente/servidor (*HTTP - HyperText Transfer Protocol*). Algunas tecnologías adicionales de la web se muestran en la tabla (tecnologias-web).
HTTPS	HTTPS utiliza el estándar SSL y ahora TLS para cifrar conexiones HTTP entre un cliente y servidor (ComputerHope, 2017).
API REST	Un API REST es un estilo arquitectónico donde se construye encima del protocolo HTTP para orientar el mismo protocolo a manejar estados. Este mismo estilo arquitectónico se orienta a escalabilidad y sistemas de software grandes (*REST - Representational State Transfer*).
HTML	HTML es el lenguaje de autoría para páginas en la web. Usa una combinación de cientos de etiquetas predefinidas para definir estructura, formato y posición de contenido (*HTML - HyperText Markup Language*).
CSS	CSS es un lenguaje que permite definir estilos de cómo se presentará HTML. Se dice que son hojas de estilo en cascada porque se puede aplicar varios de ellos "en cascada" a una página de HTML (*CSS - Cascading Style Sheets*).
JavaScript	JavaScript es un lenguaje de programación que tiene un fin de permitir que páginas de HTML sean dinámicas y pueden interactuar con el usuario en tiempo real (*JavaScript*)..

2.2. Trabajos Relacionados

Dentro de la temática de este trabajo de titulación es importante entender trabajos relacionados los mismos que son analizados para en base a ellos entender investigaciones que ya se han realizado y de esta forma aprender de ellos. A continuación se mencionan los siguientes:

2.2.1. GitEduERP

En un trabajo reciente del autor con sus colegas, Marcelo Bravo y Priscilla Vargas, en la Universidad Técnica Particular de Loja, y frente el problema de necesitar ofrecer una ambiente de programación a estudiantes en línea, se realizó un editor de código en línea con sistema de permisos y la capacidad de compartir entre usuarios en un backend de Django y utilizando una librería ACE liberado por Cloud9 IDE que guardaba código editado en una instancia de GitLab CE. Además ofrece chat en línea con una liberia TogetherJS. El resultado era un prototipo que guardaba snippets en GitLab y permite compartir código de una manera muy primitiva entre usuarios. Se llevó el codigo editado en un sistema de control de versiones interna, usando MongoDB. Entre los problemas que se dieron era una falta de tiempo para el proyecto y una alta nivel de complejidad para el mismo lo cual no permitía que se lo termina según su alcance propuesta (Bravo, Earley y Vargas, 2017).

2.2.2. Sistema de Encuestas Online

En la Universidad Técnica Particular de Loja, para mejorar temas de business analytics, dos autores, Diana Ortega y Juan Carlos Sánchez, se trataron de diseñar procesos de negocio para realizar colección de datos en tiempo real a través de encuestas y analizar las mismas con un fin de ayudar la toma de decisiones estratégicas de negocio en tiempo real. Plantearon soluciones con un fin de optimizar tiempos y recursos a través del uso de soluciones tecnológicas (Salinas y Castillo, 2007). Aunque no se trató de enseñar ni de programar en línea se considera importante este trabajo relacionado debido al hecho de que llevaron a cabo temas de business analytics y uno de las finalidades de este trabajo de titulación.

2.2.3. Metodología de Enseñanza con la Web 2.0

En la Universidad Técnica Particular de Loja, Mauricio Castillo buscó utilizar la manera en que la moda de lectura-escritura en la Web 2.0 se podría crear cursos interactivos con estudiantes online como base de una nueva metodología de enseñanza. Destaca esos temas desde el punto de vista de ingeniería en sistemas para proporcionar soluciones netamente técnicas y estratégicas para dar el mejor aporte posible a quienes deseen implementar un sistema de este tipo. Estos conocimientos fueron aplicados en una materia ”Diseño de Paginas Web dinamicas con PHP” en la modalidad abierta y a distancia de la misma universidad para comprobar la validez de la metodología desarrollado lo cual se encontró que al aplicar la metodología de integrar tecnologías de la web 2.0 y aumentar la interacción estudiantil dentro del curso, se pudo aumentar un grado significativo el porcentaje de estudiantes que aprobaron la materia (Torres, 2008).

2.2.4. Xen Web-based Terminal for Learning

Los autores, Abdullah Almurayh y Sudhanshu Semwal de IAENG, propusieron e implementaron una arquitectura de cliente-servidor para la distribución de recursos educativos y en el proceso enseñar a estudiantes de Linux, programación orientada a la nube y gestión de nubes/servidores. Su aplicación web ofrece terminales SSH donde cada estudiante y docente disponía de una máquina virtual de tal forma que tenían su propio ambiente con permisos de superusuario y al mismo tiempo eran aislados de la infraestructura real y de los demás usuarios, motivo por el cual se podía dar una solución flexible y a su vez segura.

Como hipervisor ocuparon un sistema de Xen que controlan remotamente con su servidor de aplicación (servidor web). El enfoque del trabajo era garantizar seguridad mientras que al mismo tiempo dar ambientes con permisos completos donde estudiantes pueden experimentar y aprender sin restricciones mientras al mismo tiempo ser aislados y prevenir el mal uso o de alguna forma maliciosa.

La parte de seguridad se implementó con un validacion de comandos y eliminación de caracteres especiales que se podrían usar para escapar las restricciones establecidas previamente. De esta forma se logró realizar un sistema útil para administradores, docentes y estudiantes dentro del ámbito educativo. El sistema solo se implemento un comando, xe, para gestión del hipervisor Xen atrás y los autores dejaron como trabajo futuro mejorar la seguridad a futuro debido a que no lo vean como algo completo la seguridad actual (Almurayh y Semwal, 2014).

2.3. Sistemas Similares

De la misma manera que se analiza la parte de trabajos relacionados, es importante conocer también el contexto actual del mercado para ver las alternativas que ofrece la competencia y saber si ya hay alguna solución que sea de mayor beneficio a la universidad que el mismo sistema que se plantea (y por tal razón sería más factible implementar dicha solución en lugar de desarrollar algo nuevo). Se presenta a continuación las iniciativas investigadas:

2.3.1. Repl.it

Repl.it es una plataforma en línea, desarrollado por Neoreason Inc., para escribir y probar código en tiempo real como un IDE en línea. Su producto tiene un componente educativo que permite integración con otros sistemas de aprendizaje, organización por aulas, texto que guía las tareas, calificación automática de tareas a través de pruebas unitarias entre otras características que mejoran la interacción entre docentes y sus alumnos en su aprendizaje de código. Actualmente usa “máquinas completas de linux” para soportar “más de 30 lenguajes” de una manera “fiable y segura”. Como sistema completa ofrece muchas de las funcionalidades que se esperan de un sistema de este tipo como integración LTI, ejecucion de codigo en linea, soporte para muchos lenguajes, calificación basado en pruebas unitarias entre otros. Pero tanta funcionalidad viene con un precio alto que reduce la accesibilidad al mismo para las instituciones educativos (Repl.it y Neoreason, Inc., 2017).

2.3.2. io.livecode.ch

Io.livecode.ch, desarrollado por Nada Amin, es un plataforma prototipo que convierte repositorios públicos de GitHub en tutoriales interactivos y documentados de programación. Permite la ejecución de código en contenedores de Docker en tiempo real. El mismo utiliza máquinas virtuales alojados en DigitalOcean y permite la extensión por terceras personas con más tutoriales (Amin, 2017). El problema del mismo es que no lleva estados, como ambiente de programación en línea estática, solo lleva los datos del momento sin persistirlos y por lo tanto no ofrece factibilidad dentro del alcance del proyecto actual. El mismo hecho de que lleva Docker como su tecnología de virtualización, según Amin, también baja la seguridad de la aplicación y genera graves vectores de ataque dentro del mismo.

2.3.3. Cloud9 IDE

Cloud9 IDE, desarrollado por una empresa incorporado con el mismo nombre, ofrece un espacio de trabajo personal rápido y escalable (pero administrado por la misma empresa) basado en contenedores de Ubuntu encima de Docker para cada usuario, con soporte para 40 lenguajes de programación. Permite ver aplicaciones web en tiempo real con una variedad de navegadores y sistemas operativos. También permiten conectarse a servidores privados del usuario por SSH para utilizar estos en lugar de los servidores que ellos proveen. También tiene la capacidad de compartir código entre varios usuarios bajo permisos de lectura o lectura y escritura, permitiendo que los mismos editan en tiempo real (visible a todos) y que pueden comunicarse a través de chat. El mismo código, una vista previa de ello o su versión completa también se puede compartir públicamente con usuarios que no pertenecen a la plataforma. Todo esto es con un fin de reemplazar todo el ambiente de desarrollo local con un entorno completamente en la nube; ofrece un terminal, editores avanzados de código, división de pantalla, un debugger, temas, personalización de atajos, comandos communes, modo vim y modo sublime para el editor y un editor de imágenes. Cloud9 IDE está orientado más a profesionales y por lo tanto aún no ofrece tan buena integración con sistemas educativas (*Your development environment, in the cloud*).

2.3.4. GitLab

GitLab, desarrollado por GitLab Inc. con un conjunto de miembros de una comunidad de software abierta, en adición a ser un servidor de Git, y por lo tanto llevar un control de versiones de los datos que aloja, también ofrece otras características asociados con entornos profesionales de desarrollo como seguimiento de incidentes, revisión de código, un IDE para editar código en línea, la capacidad de tener wikis asociado con proyectos, un sistema de integración continua para probar, compilar y desplegar código en una variedad de ambientes. Su versión de pago también soporta el consumo de un servidor de LDAP para autenticar usuarios, hooks de Git para tomar acciones personalizadas en respuesta a eventos de Git y capacidades para auditoria por parte de administradores (Hughes y col., 2017). Además, GitLab puede importar proyectos de otros plataformas y servidores de Git a través de una URI, gestión de snippets, ramas protegidas, un API que permite controlar al servidor de GitLab y un registro de contenedores de Docker asociados con cada proyecto. Por tener un enfoque sumamente de guardar código no ofrece caracteristicas tan fuertes ni para desarrollar en linea ni para ejecutar en línea aunque si soporta otros plataformas de integración continua (Pipinellis y col., 2017).

2.3.5. OverLeaf

Overleaf es una plataforma en línea, desarrollado por Writelatex Limited en Londres del Reino Unido, para editar y publicar de forma colaborativo documentos de LaTeX. Permite editar LaTeX directamente o usar un editor WYSIWG para quienes no conozcan bien el sistema TeX. La plataforma compila el documento de LaTeX en tiempo real como se lo va editando para disponer una vista previa con los últimos cambios y de la misma forma avisa de errores que genera al compilador. Como es una plataforma en línea, se puede editar el mismo documento varias personas al mismo tiempo desde distintos clases de dispositivos y cómo el sistema que respalda los documentos es un motor completo de LaTeX/TeX, permite una gran variedad de tipos de documentos y contenido en ellos (*Collaborative Writing and Publishing*). También

se integra Overleaf con Git ya que el mismo dispone de un servidor interno de Git para interactuar con sus proyectos a través de este sistema de control de versiones (Lees-Miller, 2015). Obviamente aunque es un plataforma muy potente para lo que hace también es completamente especializado para el fin de realizar documentos de LaTeX.

2.3.6. Google Drive

Google Drive como servicio de la misma empresa Google ofrece 15 Gigabytes de Almacenamiento gratis para guardar cualquier tipo de archivo, los mismos que pueden ser compartidos con cualquier persona para facilitar colaboración entre personas. Tiene para editar en línea documentos, hojas de cálculo, presentaciones, formularios (para hacer encuestas), dibujos y más con una API que permite mayor expansión por terceros. También controla las versiones de forma automática para que se puede volver a revisar versiones anteriores y quienes han introducido cambios y como fueron los mismos cambios. Además en caso de requerir acceder ciertos archivos sin internet, se puede señalar a la plataforma de Google Drive que se deben sincronizar en segundo plano cada vez que hay internet para también tener la disponible y actualizada cada vez que se encuentre sin conexión. Está orientado completamente al desarrollo de documentos como paquete de ofimática y por lo tanto ofrece poca utilidad para escribir codigo (*Explore the Storage Features of Drive*).

2.4. Discusion

A continuación, se recapitula los trabajos relacionados y sistemas similares previo al fase de análisis y diseño de una solución en el próximo capítulo.

2.4.1. Comparación de Trabajos Relacionados

En el cuadro 2.3 se ve las diferencias entre los trabajos relacionados y los campos del trabajo actual en donde se aplica cada uno de ellos. Tanto GitEduERP como Xen Web-based Terminal for Learning Virtualization permiten a estudiantes editar y persistir código en línea a diferencia del Sistema de Encuestas Online que solo era para recolectar y analizar datos y la metodología de enseñanza online con la web 2.0 que no implementa ninguna parte de enseñar la parte práctica de programación en línea. Para enseñar a estudiantes en línea solo dos trabajos relacionados se tratan de eso; Metodología de Enseñanza con la Web 2.0 que buscaba aplicar tecnologías como los blogs para aumentar la participación estudiantil dentro de las materias a distancia y con ello aumentar su aprendizaje medido a través de la métrica de sus notas en una materia de programación y Xen Web-based Terminal for Learning Virtualization que a diferencia de los demás trabajos relacionados ocupó máquinas virtuales (de Xen) para dar ambientes completos y no tan restringidos a estudiantes para permitir su aprendizaje libre en línea con aspectos de virtualización, programacion y Linux.

CUADRO 2.3: Comparación de Trabajos Relacionados.

	Editar Código en Línea	Persistir Código en Línea	Recolección y Análisis de Datos para decisiones estratégicas en tiempo real	Metodología de Enseñanza Online	Ambientes Virtualizados
GitEduERP	x	x			
Sistema de Encuestas Online			x		
Metodología de Enseñanza con la Web 2.0				x	
Xen Web-based Terminal for Learning Virtualization	x	x		x	x

2.4.2. Comparación de Sistemas Similares

En el cuadro 2.4 se ve las características básicas que se considera para un sistema de este tipo. Dentro de esta comparacion, se ve que Repl.it es la mejor alternativa debida a que soporta todas las características menos la compilación en tiempo real (que podría venir a ser una desventaja que empeora el rendimiento del sistema.

En el cuadro 2.5 se ve vuelta aspectos sociales de cada sistema, entre ellos cómo funcionan las dinámicas de compartir código y interactuar entre usuarios. Aqui Repl.it sale perdiendo con muy pobre suporte para compartir entre varios usuarios. Los sistemas que son mejores en estos aspectos, por ejemplo GitLab, Overleaf y Google Drive no son tan optimizados a escribir código en línea o a ejecutar el mismo.

CUADRO 2.4: Comparación de Características Generales entre Sistemas Similares.

	Escribir Código Online	Probar Código Online	Integración LTI	Sistema de Autocalificación	Compilación en Tiempo Real
Repl.it	Si	Si	Si	Si	No
io.livecode.ch	Si	Si	No	No	No, pero si permite ejeccución de código en linea
Cloud9 IDE	Si	Si	No	No	Si para ciertos lenguajes
GitLab	Si	No	No	No	No
Overleaf	Solo LaTeX	Solo LaTeX	No	No	Si, LaTeX
Google Drive	No, solo se lo considera como texto	No	No	No	No

CUADRO 2.5: Comparación de Características Sociales entre Sistemas Similares.

	Control Interno de Versiones	Integración de algún sistema de control de versiones	Sistema de Permisos	Sistema de Compartir	Compartir / Editar en Tiempo Real
Repl.it	No	No	Si, basado en docente / alumno	No	No
io.livecode.ch	No	No	No	Todo es Público	No
Cloud9 IDE	Si	No	Si, basado en usuarios	Si, basado en usuarios	Si
GitLab	Si	Si, Git	Si, basado en usuarios y grupos	Si, basado en usuarios y grupos	No
Overleaf	Si	Si, Git	Si, basado en usuarios	Si, basado en usuarios	Si
Google Drive	Si	No	Si, basado en usuarios	Si, basado en usuarios	Si

En el cuadro 2.6, se compara las diferencias entre los aspectos más avanzados de cada uno de los sistemas similares. Aquí no hay ni ganadores ni perdedores. Repl.it ofrece una solucion robusta pero que falta alguna forma de acceso fuera de línea. io.livecode.ch ofrece un sistema basico de documentacion y ejecución basado en HTML, pero no maneja estados y no ofrece interactividad en tiempo real.

Adicionalmente, Cloud9 sufre de las mismas problemas de faltar usabilidad sin conexión y de no disponer de una solución robusta de documentación. GitLab y Overleaf ofrecen acceso a través de Git de sus contenidos, abriendo la posibilidad de trabajar sin conexión, pero Overleaf no es para código exactamente. GitLab ofrece el mejor solucion de documentación a través de sus Wikis, pero esos no se visualizan al mismo tiempo que el código. Y Google Drive ofrece la mejor solución para trabajar

CUADRO 2.6: Comparación de Características Avanzadas entre Sistemas Similares.

	Control Completo sobre ambiente de ejecución de código	Acceso Remoto al ambiente de Ejecución de Código	Sistema de Documentación	API sobre HTTP(S)	Acceso fuera de línea
Repl.it	Si para instalación de dependencias	No	Si, documentación del ejercicio	Si, pero está cerrado a nuevos clientes	No
io.livecode.ch	Si, script de Bash	No en tiempo real	Si, HTML	No mantiene estados	No
Cloud9 IDE	Si, Terminal	Si, SSH	No, fuera de documentos en el servidor	Si	No
GitLab	No	No	Si, Wikis con GitHub Markdown	Si	Si, si es que se ha bajado todo anteriormente con Git
Overleaf	No	No	No, todo el sistema es de documentación	Si	Si, si es que se ha bajado todo anteriormente con Git
Google Drive	No	No	No, todo el sistema es de documentación	Si	Si

sin conexion, pero el plataforma es para documentos, no para código.

Al final, ninguna de los sistemas similares alcanza el nivel de funcionalidad que se requiere y por lo tanto, aunque se puede estudiar los mismos para ver que hacen bien y que hacen mal, es necesario, a partir del próximo capítulo, analizar y diseñar e implementar una solución nueva que satisface las necesidades institucionales que hay actualmente.

Capítulo 3

Analisis y Diseño

3.1. Analisis

La resolución de los problemas siempre debe empezar con un análisis acerca de su alcance y características. En base a aquello análisis se empieza a salir las ideas de soluciones para en base a ellas diseñar una resolución del mismo problema. Para la fase de análisis se ha realizado un Documento de Visión (Apéndice A) y una Especificación de Requerimientos (Apéndice B).

3.1.1. Vision

El resto del documento de visión se puede encontrar en el Apéndice A. Aquí se ha incluido la parte de mayor consideración.

Proposito

Ayudar a mejorar los métodos de enseñanza que ofrece la Universidad Técnica Particular de Loja en cuanto a la programación y uso de base de datos para las carreras de Sistemas y Electrónica.

Alcance

Se propone un sistema de editar código en línea que a su vez integra LMS externos (para autenticación y notas), un servidor de control de versiones externo (para la persistencia de código), y un servicio web de ejecución de código en línea de una forma segura, eficaz y eficiente (para dar un ambiente de ejecución y pruebas tanto para los usuarios del sistema como para calificar de una forma automática).

Oportunidad de Negocios

Con el avance continuo de la tecnología y su introducción en mas aspectos de la vida diaria de cada uno, hay una necesidad creciente de ingenieros en sistemas e electrónica que pueden programar e entender el software que hace todo funcionar en adición a los bases de datos que están por detrás de estos mismos sistemas.

Es por aquella razón que ahora está de moda ofrecer plataformas en línea para la enseñanza dinámica de la programación. GitEDU espere ofrecer las mismas funcionalidades a un costo institucional menor a travez de integración con sistemas existentes e innovación para proveer una mejor experiencia de usuario, tanto estudiantes como profesores para llevarse a cabo un mejor proceso de aprendizaje.

Demográfica del Mercado

En el año 2011, la Universidad Técnica Particular de Loja contaba con aproximadamente 4000 estudiantes presenciales y 24000 estudiantes a distancia con una tendencia creciente (UTPL, 2017a). En la experiencia personal del autor, las carreras de Sistemas y Electrónica, por lo menos en la modalidad presencial, juntos representan aproximadamente un 10 % de todos los estudiantes en la universidad lo cual daría un mercado de estudiantes afectados por un nuevo sistema de aproximadamente un mínimo 2800 estudiantes. Según el directorio de docentes de la universidad, son 60 profesores en el departamento de Ciencias de la Computación y Electrónica (UTPL, 2017b). Con eso se puede estimar un mínimo de 2860 usuarios lo los cuales el sistema propuesto podría llegar a afectar.

Es precisamente la parte de la población, de usuario potenciales mencionado anteriormente, que está en el proceso de enseñar, evaluar y aprender habilidades de programación y consultar bases de datos que forman la base de usuarios de la aplicación.

3.1.2. Especificacion de Requerimientos

El resto del documento de especificación de requerimientos se puede encontrar en el Apéndice B. Aquí se ha incluido la parte de mayor consideración.

Proposito

El siguiente documento espera dar una descripción detallado de los requisitos del Sistema Git Education (GitEDU) con las finalidades de definir la intención de la misma, declarar de forma completa todos los componentes a ser desarrollados del sistema, y también explicar limitaciones del sistema además de sus interacciones y interfaces con otros sistemas. Se destina el siguiente documento para revisión por el equipo de asesores y como una referencia de alcance para el equipo de desarrollo.

Alcance

GitEDU es un sistema de integración para promover la programación estudiantil y el sistema educativo que soporta el mismo. El sistema final debe disponer de buena documentación y alta calidad de código para sostener su futuro desarrollo y mantenimiento por parte de una comunidad de profesionales y profesionales en formación.

Docentes de las carreras que involucran la enseñanza de programación, actualmente tanto la carrera de Sistemas Informáticos y Computación como la carrera de Electrónica y Telecomunicaciones, pueden crear dentro de la plataforma, deberes, talleres, pruebas y exámenes, y a través de los LMS institucionales, compartir las mismas con sus alumnos. Los alumnos pueden entrar a través de un enlace que les comparte su docente dentro del LMS institucional y con un canal de comunicación LTI, la plataforma GitEDU les identifica y autentica sin interacción del usuario para que el estudiante puede empezar directamente con el trabajo que tiene que realizar sin digitar sus credenciales. En el curso de su actividad, se va guardando su progreso periódicamente contra un servidor de control de versiones institucional (GitLab CE) y en cualquier momento el estudiante puede probar e interactuar con su código a través de un terminal virtual de Linux que se encuentra dentro de la misma interfaz. Una vez que se termine su trabajo y desee enviarlo, se lo envía y el código escrito se auto califica contra un conjunto de pruebas unitarias ocultas que el docente agregó a

la actividad al crearlo. La nota que se genera se lo envía al LMS institucional a través del mismo canal LTI. En caso de que el docente decide no utilizar pruebas unitarias o calificar cada trabajo a mano o realizar una calificación híbrida entre las pruebas unitarias y la parte a mano, no se reflejarán las notas en el LMS hasta que el docente se ha terminado de calificar el código en la plataforma de GitEDU.

Perspectiva de Producto

La plataforma GitEDU se descompone en dos subsistemas: el subsistema de editar código y el subsistema de ejecutar código. El subsistema de editar código consiste en la integración de autenticación y notas por el lado del LMS, el uso de persistencia del código por el servidor de control de versiones por otro lado, el consumo de un servicio de ejecución de código por un tercer lado y un editor de código en línea. El subsistema de ejecutar código consiste en esperar llamadas del subsistema anterior, validar la autenticidad, establecer permisos, limitaciones y parámetros de ejecución, gestionar recursos para ejecución, la ejecución interactiva y devolución de resultados al otro subsistema.

Debido a la necesidad de mantener seguridad y prevenir la ejecución indebido de código, el subsistema para lo mismo debe ser aislado de todos los usuarios y solo accedido desde el otro subsistema a través de un API que permite su interoperabilidad. La solución para la ejecución de código debe ser liviana debido a número grande de usuarios concurrentes que puede tener a la vez.

Funcionalidades de Producto

Para entrar al subsistema de editar código en línea, se debe autenticarse contra el LMS institucional. El editor de código en línea debe soportar una variedad grande de lenguajes y estar abierto a la agregación de más lenguajes en un futuro. La persistencia de código en un servidor de control de versiones debe ser transparente y no requerir ningún interacción del usuario. El interfaz para interactuar con el ambiente virtual debe ser simple y fácil de manejar sin mayor conocimiento de las tecnologías que le respaldan. Se debe poder acompañar el editor de código con documentación que guía el estudiante en explicar o resolver el problema actual. La manera en que docentes pueden generar nuevos ejercicios, su documentación y pruebas unitarias para la calificación debe ser intuitivo.

Caracteristicas de Usuario

Los tres tipos de usuarios que interactúan con el sistema son Docentes, Alumnos y Administradores. Cada tipo de usuario tiene sus propias necesidades y requerimientos.

Docentes necesitan un sistema que sea fácil de manejar y que les simplifica y automatiza el proceso de enseñar y evaluar a sus estudiantes. Eso significa tener una interfaz intuitiva para docentes que les permite escribir, documentar y generar pruebas unitarias para probar funcionalidad de código que escribe estudiantes para los ejercicios.

Alumnos necesitan un sistema que les brinde una buena experiencia de aprendizaje y les proporciona todas las herramientas que requieren para desarrollar en linea. Igual que sus profesores, deben disponer de una interfaz intuitiva que les ayuda cumplir con sus responsabilidades de estudiar, aprender y demostrar conocimientos en una cantidad de tiempo óptimo.

Administradores necesitan un sistema que sea seguro y cuyos componentes sean fáciles de mantener y extender a lo largo del tiempo. Para ello se necesita tener una buena documentación del funcionamiento del sistema y todos sus componentes para tener la misma como referencia.

Limitaciones

Se requiere de hardware con alta capacidad de virtualización y conectividad al internet y la intranet institucional para poder brindar la mejor experiencia al usuario.

Suposiciones y Dependencias

Se supone que siempre habrá la conectividad necesaria para poder usar todos los componentes del sistema.

División de Requerimientos

En caso de que falta recursos de tiempo o algún otro recurso, se limitará el alcance del proyecto para enfocarse en los requerimientos de una prioridad más alta y dejar los demás requerimientos para trabajos futuros.

3.2. Diseño

Previo a la implementación de cualquier sistema es importante analizar su funcionamiento y en base a lo mismo realizar un diseño que provee estas funcionalidades que se requiere. Por lo tanto, a continuación se demuestra las fases del diseño realizado desde la definición de sistemas, subsistemas y módulos y la interacción entre ellos, hasta la arquitectura interna y de despliegue.

3.2.1. Sistemas, Subsistemas y Módulos

Debido a la naturaleza del alcance de este trabajo de titulación, se considera que se debe dividir la plataforma en dos sistemas que sean capaces de interactuar entre sí. El primer sistema es la que ofrece el editor de código en línea, integración con los LMS institucionales y a su vez integración con un servidor de control de versiones. El segundo sistema solo se encarga de ejecución de código en línea a través de algún tipo de virtualización y aislamiento de procesos. De esta forma se puede aislar el sistema que está bajo mayor riesgo de ser atacado (el de ejecución de código arbitrariamente) del sistema con mayor riesgo de impacto negativo (el sistema de escribir código en línea, el cual también se encarga de las calificaciones). A continuación se presenta la interacción entre los sistemas y actores en el Diagrama de Contexto de Sistemas.

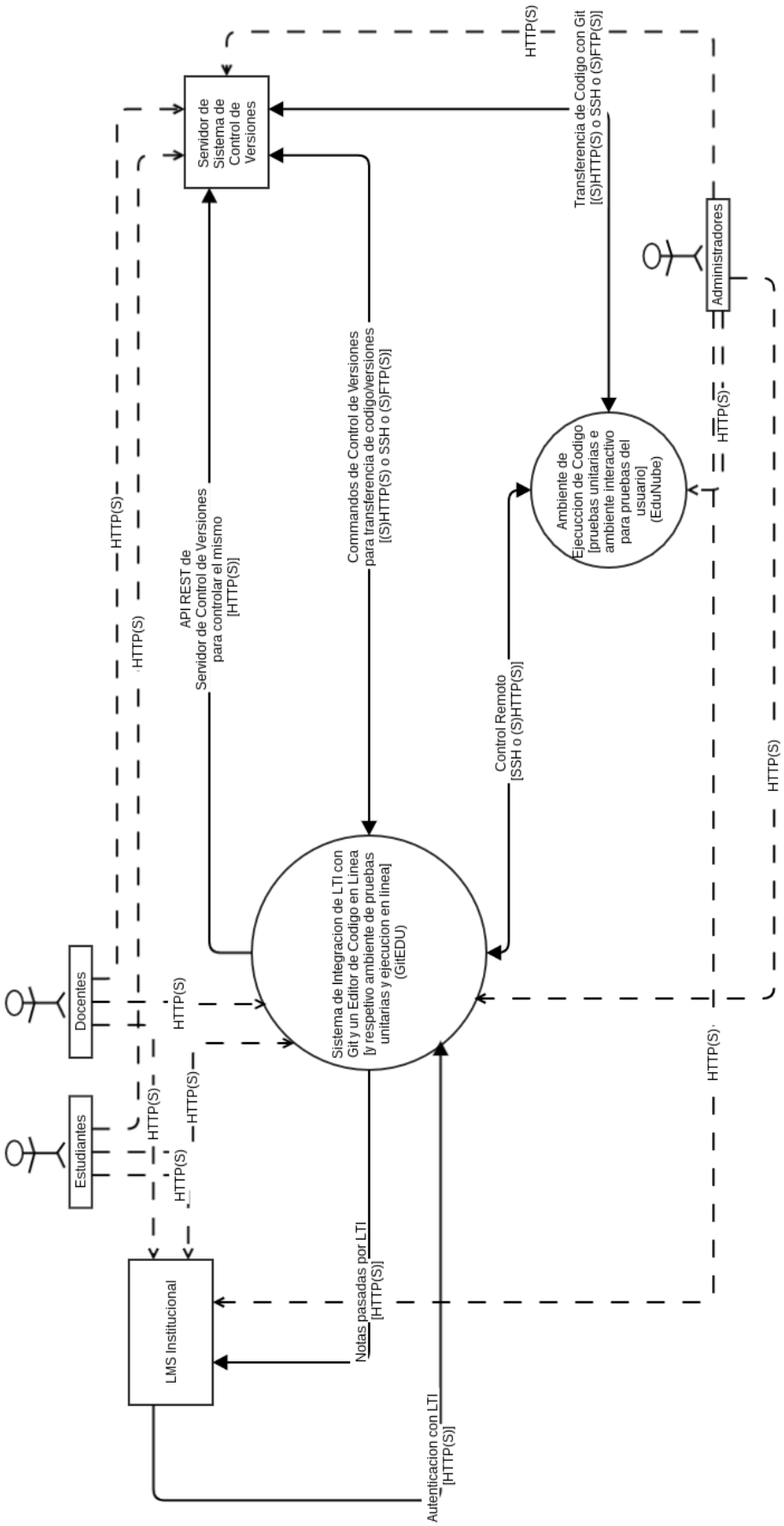

FIGURA 3.1: Diagrama de Contexto para GitEdu y EduNube.

El diagrama de contexto visualizado en la figura 3.1 demuestra la relación entre los tres clases de usuarios, sistemas existentes, los sistemas que se plantea desarrollar y los protocolos de interacción entre sistemas distintas y también entre humanos y sistemas.

Los tres tipos de usuarios que se considera para el sistema son administradores, quienes administran y mantienen todo el sistema en su ambiente de despliegue, docentes, quienes generan contenido didactica para sus estudiantes y califican los mismos motivo por el cual que tienen acceso a los LMS, el sistema de editar codigo (que les permite generar nuevas tareas / exámenes / pruebas / talleres para sus estudiantes y ver el progreso y calificaciones de los mismos) y el sistema de control de versiones (donde puede ver y bajar codigo que escriben estudiantes y ver a un nivel más profundo el historial y evolución del mismo), y estudiantes, quienes tengan acceso a los mismos tres sistemas de los docentes. Los estudiantes acceden al LMS donde vean los tareas / exámenes / pruebas / talleres nuevos que hay y al abrirlos se les lleva, identifica y autentica el sistema de editar código en línea mediante LTI. Los estudiantes realizan sus obligaciones de código, mientras que los mismos se respaldan de forma automática en un servidor de control de versiones independiente de tal forma que se lo puede revisar tanto los estudiantes involucrados como su docente a futuro. Aquello control adicional podría llegar a ser una evidencia importante para la resolución de conflictos entre estudiantes y/o docentes a futuro.

El sistema de editar codigo, apartir de aqui conocido simplemente bajo el nombre GitEDU, controla remotamente por medio de comandos, mediante posiblemente una mezcla de protocolos (SSH, HTTP sobre SSH [SHTTP] y HTTPS), dependiendo de cómo se da el caso, un sistema de ejecucion de código, conocido a partir de aquí en adelante como EduNube, para señalar acciones que debe tomar el segundo y/o datos que necesita sincronizar con la primera. Al mismo tiempo, GitEDU comunica con el servidor de control de versiones mediante su API externa (una API REST) sobre HTTP o HTTPS para controlar el mismo y prepararle para flujos continuos de código editado que puede viajar por una variedad de protocolos como SHTTP, HTTPS, HTTP, SFTP, FTP o FTPS como requiere la situación y ambiente de despliegue. Siempre y cuando EduNube requiere de codigo de algun usuario para realizar una calificación o algún otro tipo de interacción con un usuario que requiere probar el mismo, primero se procede a descargar el código involucrado desde el servidor de control de versiones sobre uno de los protocolos mencionado previamente para el mismo. En base a la naturaleza de las operaciones que está haciendo, comunica sus resultados con GitEDU para que puede manejar las mismas de la forma adecuada. En el caso de notas, GitEDU se encarga de sincronizar las mismas con los LMS institucionales con los cuales está integrado mediante el protocolo LTI.

GitEdu

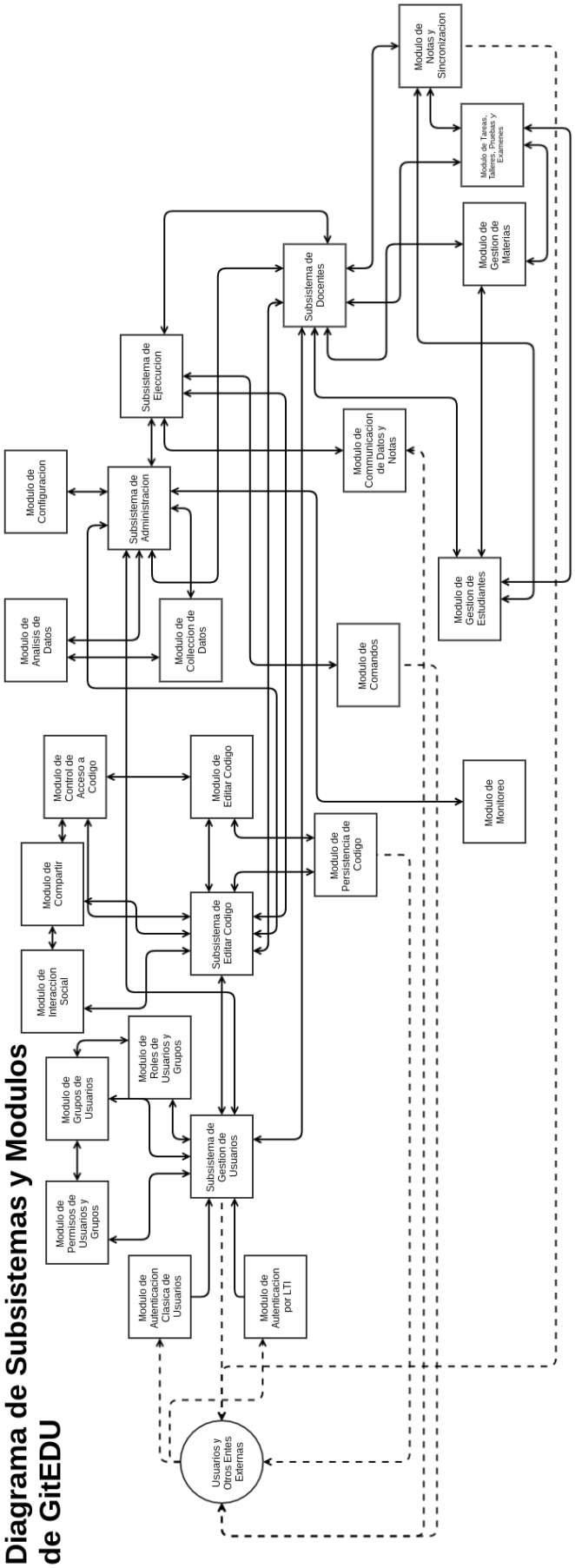

FIGURA 3.2: Diagrama de Subsistemas y Modulos de GitEdu.

La figura 3.2 presenta todos los componentes en forma de subsistemas y módulos que forman parte del diseño propuesta para el sistema GitEdu.

Subsistema de Gestión de Usuarios El subsistema de gestión de usuarios de encarga de temas de seguridad y acceso por usuarios a varios componentes de la aplicación. Para lo mismo autentica usuarios mediante dos módulos (uno clásico y el otro mediante el protocolo LTI) y lleva un sistema de permisos orientado a usuarios, grupos y roles.

Módulo de Autenticación Clásica de Usuarios El módulo de autenticación clásica de usuarios permite que los usuarios finales pueden iniciarse session mediante un usuario y contraseña.

Módulo de Autenticación por LTI El módulo de autenticación por LTI permite que usuarios finales puede iniciar su sesión mediante una sesión existente en algún sistema externa que soporta LTI.

Módulo de Grupos de Usuarios El módulo de grupos de usuarios permite definir grupos específicos de usuarios cuyo membresía les cambia el nivel y naturaleza del acceso que tengan a la aplicación.

Módulo de Permisos de Usuarios y Grupos El módulo de permisos de usuarios y grupos define la naturaleza de acceso que tengan usuarios autorizados y los grupos a los cuales pertenecen dentro de la aplicación.

Módulo de Roles de Usuarios y Grupos El módulo de roles de usuarios y grupos define las responsabilidades de varios grupos de usuarios y usuarios específicos y en base a los mismos les asigna permisos adecuados tanto para proteger la aplicación contra uso y accesos indebidos y al mismo tiempo permitir a los usuarios poder realizar sus actividades normalmente sin impedimentos.

Subsistema de Editar Código El subsistema de editar codigo forma el núcleo de la funcionalidad del sistema GitEdu. Eso lo realiza a través de funcionalidades de editar codigo, persistirlo y permitir la interacción adecuada y social entre usuarios.

Modulo de Editar Codigo El módulo de editar código en línea ofrece funcionalidades básicas de un editor de código (IDE) en línea.

Modulo de Persistencia de Codigo El módulo de persistencia de código persiste codigo versionado en servidores de control de versiones externas que usuarios escriben y editan, y al momento de necesitar abrir un código previo, recupera el mismo de los servidores de control de versiones externas para que puede seguir editando lo mismo.

Módulo de Control de Acceso de Código El módulo de control de acceso de código permite a usuarios definir y modificar los permisos que tiene su código para controlar la naturaleza del acceso que tienen otros usuarios, grupos y roles.

Modulo de Compartir El módulo de compartir permite compartir el acceso al código con otros usuarios y grupos.

Módulo de Interacción Social El módulo de interacción social permite que usuarios pueden interactuar en tiempo real mientras que editan juntos el código.

Subsistema de Docentes El subsistema de docentes ofrece muchas de las funcionalidades que requieren los docentes para llevar exitosamente su materia de programación. Estas funcionalidades incluyen gestión de estudiantes, materias y notas. La gestión de materias y notas también involucra un gestión y calificación de talleres, deberes, pruebas y exámenes. Además por el hecho de que se genera nuevas notas y calificaciones a través de GitEdu, también se encarga de sincronizar notas con otras sistemas externas como los LMS a través de LTI.

Modulo de Gestion de Estudiantes El módulo de gestión de estudiantes permite que docentes pueden ver notas de un estudiante suyo. Además estudiantes pueden revisar sus notas y tanto administradores como docentes pueden asignar estudiantes a aulas o cambiar o eliminar asignaciones previas.

Módulo de Gestión de Materias El módulo de gestión de materias permite a docentes y administradores ver estadísticas de una materia, la creación de grupos de estudiantes dentro de una materia (por ejemplo para un proyecto grupal) tanto por parte de un docente o por parte de un estudiante, modificar grupos de estudiantes (por ejemplo botar un integrante), y asignar talleres, deberes, pruebas o exámenes a un estudiante, grupo o materia. Además dentro de la gestión de materias, docentes pueden establecer el peso de cada taller, deber, prueba y examen para en base a ello calcular los promedio de un estudiante.

Módulo de Talleres, Deberes, Pruebas y Exámenes El módulo de talleres, deberes, pruebas y exámenes permite a docentes definir nuevos talleres, deberes, pruebas, exámenes, etc y los criterios de evaluación de los mismos. Involucrado en eso son pruebas unitarias (si desee calificación automatizada) y documentación que explica lo que hay que hacer al estudiante.

Módulo de Notas y Sincronización El módulo de notas y sincronización gestiona las notas que mantiene GitEdu y se encarga de sincronizar cambios en las mismas. Aunque realiza este trabajo por detrás normalmente, también proporciona una interfaz gráfica para interacción con los docentes respectivos y adecuados. Además por el tema de seguridad y integridad de las notas, se lleva las mismas versionado y con un registro preciso de cuando fueron modificados y por quien.

Subsistema de Ejecución de Código El subsistema de ejecución de código se encarga de comunicar con el sistema externa EduNube con un fin de aprovechar las funcionalidades del mismo y proveer funcionalidades de ejecución en línea, aplicación de pruebas unitarias y calificación automática de código sin exponer el sistema EduNube a usuarios finales. Para aumentar la seguridad que logra llevar EduNube, GitEdu communica con lo mismo a través de canales seguros como HTTPS y SSH utilizando métodos de autenticación orientados a una schema de llaves públicas y privadas (criptografía asimétrica) para asegurarse de la identidad de ambos previo al envío de comandos y comunicación de datos y notas.

Modulo de Comandos El módulo de comandos se encarga de controlar el sistema de EduNube a través de su API externa para indicar al mismo cuando debe bajar, ejecutar o calificar algún código.

Módulo de Comunicación de Datos y Notas El módulo de comunicación de datos y notas permite la sincronización de los mismos cuando sea necesario entre EduNube y GitEdu.

Subsistema de Administración El subsistema de administración ayuda a administradores de GitEdu recolectar datos acerca del uso del mismo y analizar lo datos en adición a ser capaces de configurar y monitorear el rendimiento del sistema.

Módulo de Configuración El módulo de configuración permite la configuración en tiempo real del sistema GitEdu, por ejemplo conexiones con sistemas externas y gestión de usuarios.

Modulo de Monitoreo El módulo de monitoreo permite ver en tiempo real el consumo del sistema.

Módulo de Recolección de Datos El módulo de recolección de datos lleva un registro continuo del uso del sistema para su revisión y análisis por parte de operadores humanos después tanto para temas de auditoría como toma de decisiones estratégicas.

Módulo de Análisis de Datos El módulo de análisis de datos proporciona las herramientas necesarias para que un operador humano puede interpretar los mismos y sacar sus conclusiones.

EduNube

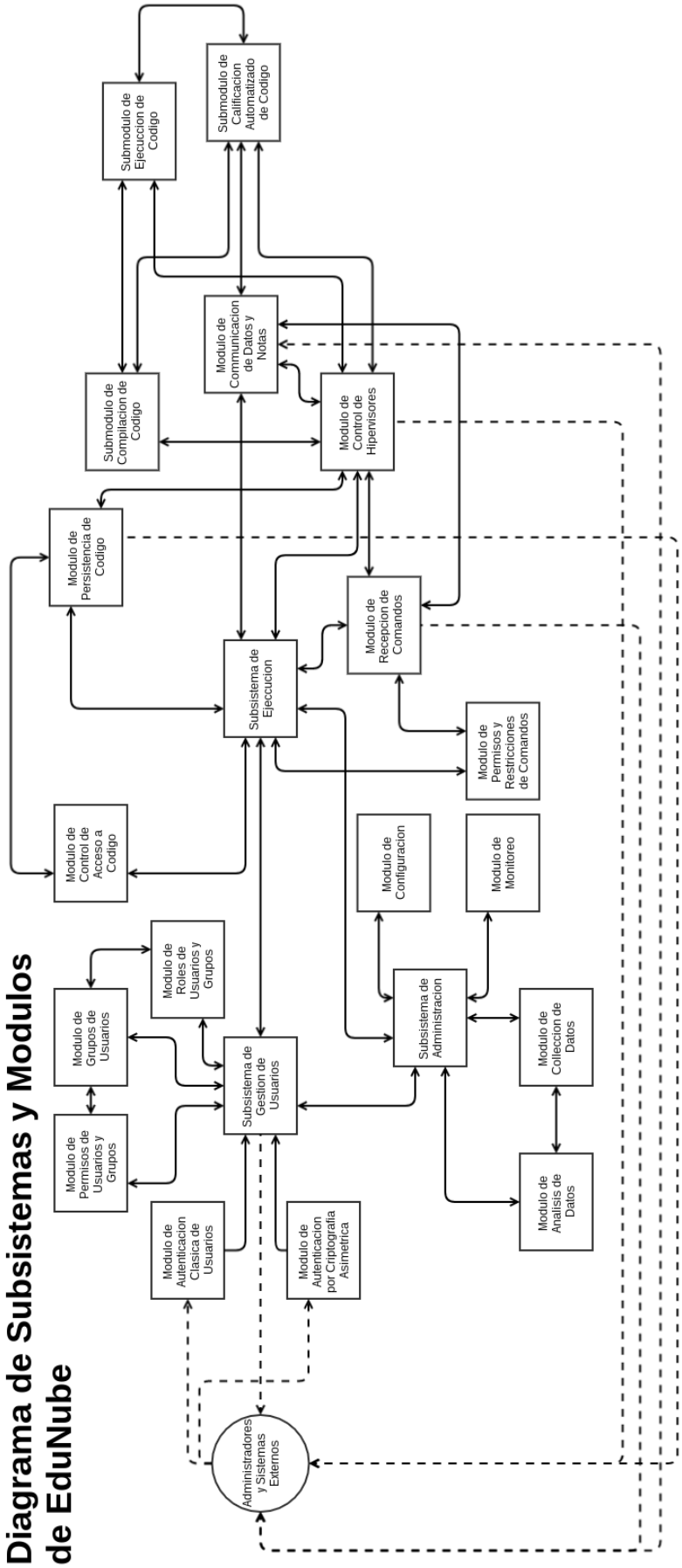

FIGURA 3.3: Diagrama de Subsistemas y Modulos de EduNube.

La figura 3.3 presenta todos los componentes en forma de subsistemas y módulos que forman parte del diseño propuesta para el sistema EduNube.

Subsistema de Gestión de Usuarios El subsistema de gestión de usuarios de encarga de temas de seguridad y acceso indirecto por usuarios (a través de GitEdu) y acceso directo por parte de administradores a varios componentes de la aplicación. Para lo mismo autentica usuarios mediante dos módulos (uno clásico solo para administración del mismo y el otro mediante llaves públicas/privadas [criptografía asimétrica] para aplicaciones externas) y lleva un sistema de permisos orientado a usuarios, grupos y roles.

Módulo de Autenticación Clásica de Usuarios El módulo de autenticación clásica de usuarios permite que administradores pueden iniciarse sesión mediante un usuario y contraseña.

Módulo de Autenticación por Criptografía Asimétrica El módulo de autenticación por criptografía asimétrica permite que usuarios finales, a través de GitEdu puede iniciarse sesión mediante para la ejecución controlada y remoto de comandos y código.

Módulo de Grupos de Usuarios El módulo de grupos de usuarios permite definir grupos específicos de usuarios cuyo membresía les cambia el nivel y naturaleza del acceso que tengan a la aplicación.

Módulo de Permisos de Usuarios y Grupos El módulo de permisos de usuarios y grupos define la naturaleza de acceso que tengan usuarios autorizados y los grupos a los cuales pertenecen dentro de la aplicación.

Módulo de Roles de Usuarios y Grupos El módulo de roles de usuarios y grupos define las responsabilidades de varios grupos de usuarios y usuarios específicos y en base a los mismos les asigna permisos adecuados tanto para proteger la aplicación contra uso y accesos indebidos y al mismo tiempo permitir a los usuarios poder realizar sus actividades normalmente sin impedimentos.

Subsistema de Ejecución El subsistema de ejecución de código se encarga de recibir y procesar comandos del sistema externa GitEdu con un fin de aprovechar los hipervisores que tiene a su cargo y proveer funcionalidades de ejecución en línea, aplicación de pruebas unitarias y calificación automática de código sin exponer dichos hipervisores a usuarios finales. Para aumentar la seguridad que logra llevar EduNube, GitEdu communica con lo mismo a través de canales seguros como HTTPS y SSH utilizando métodos de autenticación orientados a una schema de llaves públicas y privadas (criptografía asimétrica) para asegurarse de la identidad de ambos previo al envío de comandos y comunicación de datos y notas.

Módulo de Recepción de Comandos El módulo de recepción de comandos se encarga de recibir comandos externos enviados a través del API externa que provee para que otras aplicaciones pueden indicar al mismo cuando debe bajar, ejecutar o calificar algún código.

Módulo de Permisos y Restricciones de Comandos El módulo de permisos y restricciones de comandos se encarga de limitar o permitir comandos externos recibidos por el sistema EduNube. Se basa en políticas establecidos previamente que definen los horarios de trabajo de ciertos procesos y dependencias de los mismos (por ejemplo no se debe calificar los exámenes de un parallelo hasta que todos terminan). Junto al modulo de control de acceso a codigo, eso permite a docentes y administradores tener control refinado sobre cuando y que está permitido hacer usuarios que están usando EduNube a través de GitEdu.

Módulo de Control de Hipervisores El módulo de control de hipervisores se encarga de cumplir con la funcionalidades críticos del sistema EduNube de compilar, ejecutar y calificar código. Esto lo hace a través de tres respectivos submódulos para cada fase. De manera más general, este módulo también controla hipervisores conectados al sistema EduNube con un fin de poder realizar las operaciones listados previamente.

Submódulo de Compilación de Código El submódulo de compilación de código se encarga de compilar codigo que se le entrega en base a procedimientos que define el profesor anteriormente y reporta resultados del mismo, sea exitosa o sea errores de compilación. Para la implementación del mismo submódulo, se aprovecha de las máquinas virtuales que provee el hipervisor con un fin de realizar la compilación en un ambiente controlado y aislado.

Submódulo de Ejecución de Código El submódulo de ejecucion de codigo recibe de entradas codigo compilado y comandos que se desee ejecutar en el ambiente de ejecución y pruebas con un fin de proveer, a través de las máquinas virtuales controladas y aisladas, un ambiente interactivo donde estudiantes, tanto estudiantes como docentes pueden trabajar de una forma interactiva con código desarrollado dentro de la plataforma.

Submódulo de Calificación Automatizado de Código El submódulo de calificación automatizado de código se basa en pruebas unitarias escritas previamente por un docente y que van contra la funcionalidad que se pide en el código de los estudiantes con un fin de calificar el código de una forma autónoma. Una vez que termina de calificar un código, reporta los resultados a gestores de los mismos.

Módulo de Persistencia de Código El módulo de persistencia de código recupere codigo versionado en servidores de control de versiones externas que usuarios han escrito y editado con un fin de permitir la compilación, ejecución y calificación del mismo dentro del sistema EduNube.

Módulo de Control de Acceso a Código El módulo de control de acceso a codigo se encarga de restringir y permitir en base a reglas las operaciones de compilación, ejecución y calificación automatizada de códigos que usuarios editan. De la misma forma permite definir flujos de trabajo y dependencias para el procesamiento de proyectos en adición a establecer horarios y restricciones de procesamiento para los mismos.

Módulo de Comunicación de Datos y Notas El módulo de comunicación de datos y notas permite la sincronización de los mismos cuando sea necesario entre EduNube y GitEdu.

Subsistema de Administración El subsistema de administración ayuda a administradores de EduNube recolectar datos acerca del uso del mismo y analizar lo datos en adición a ser capaces de configurar y monitorear el rendimiento del sistema.

Módulo de Configuración El módulo de configuración permite la configuración en tiempo real del sistema EduNube, por ejemplo conexiones con sistemas externas, gestión de llaves públicas y privadas, y gestión de usuarios.

Modulo de Monitoreo El módulo de monitoreo permite ver en tiempo real el consumo del sistema y sus hipervisores respectivos.

Módulo de Recolección de Datos El módulo de recolección de datos lleva un registro continuo del uso del sistema para su revisión y análisis por parte de operadores humanos después tanto para temas de auditoría como toma de decisiones estratégicas.

Módulo de Análisis de Datos El módulo de análisis de datos proporciona las herramientas necesarias para que un operador humano puede interpretar los mismos y sacar sus conclusiones.

Ventajas del Diseño de Sistemas, Subsistemas y Módulos

La ventaja de un diseño orientado a módulos como se lo presenta aquí es la división de funcionalidad para ayudar en la extensibilidad y mantenibilidad de los sistemas al largo plazo. Por ejemplo este diseño permite agregar funcionalidad adicional a futuro como autenticación por otros medios (por ejemplo LDAP) o la integración de mecanismos anti-plagio. Esta propiedad puede dar una vida útil mayor a los sistemas desarrolladas porque muchas veces no se conoce todo lo que puede requerir un software a futuro para el diseño e implementación inicial. A continuación se presenta la arquitectura de estos dos sistemas, tanto GitEdu como su ambiente de ejecución de código virtualizado, EduNube.

3.2.2. Arquitectura

Como se puede observar en la figura 3.4, a un nivel global, el sistema GitEdu consiste en 5 capas que son:

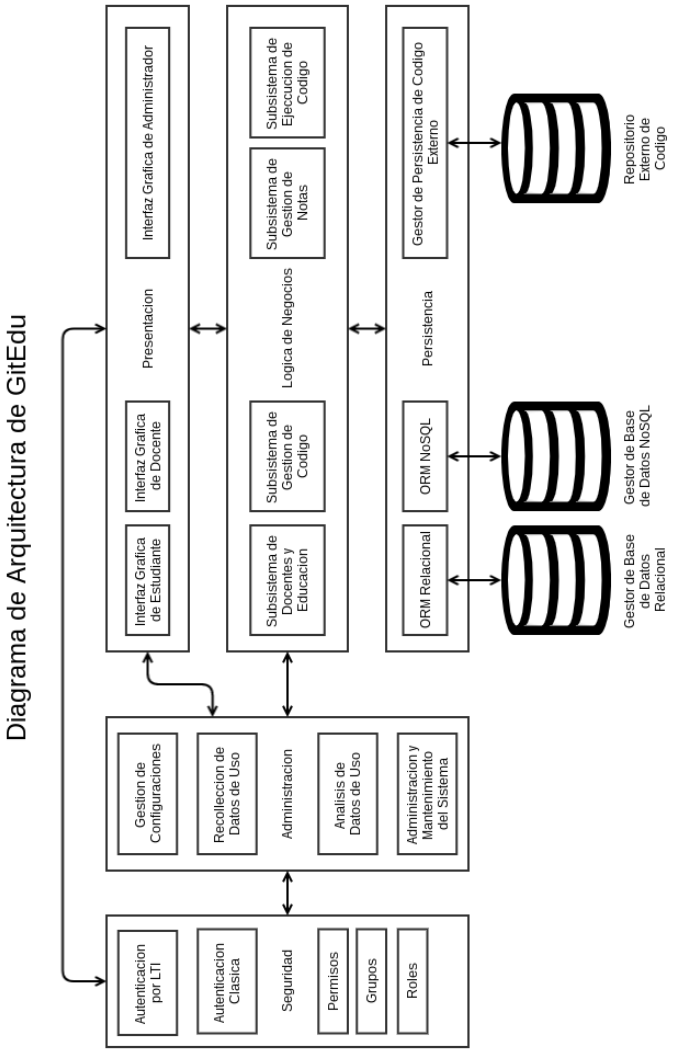

FIGURA 3.4: Diagrama de arquitectura para el sistema GitEdu.

- **Seguridad** auténtica usuarios y controla el acceso que tiene cada uno de ellos. Esto se realiza con los siguientes componentes:
  - **Autenticación por LTI** usa interfaces LTI de aplicaciones externas como LMS institucionales para autenticar usuarios.
  - **Autenticación Clásica** permite el login (autenticación) de usuarios mediante la forma clásica de un usuario y contraseña.
  - **Permisos** permite otorgar o revocar partes específicas del sistema a usuarios específicos. Ejemplos de permisos podrían ser acceso a la administración de una materia, por ejemplo el control que se da a un docente o grupo de docentes sobre una materia.
  - **Grupos** permite aplicar el sistema de permisos a grupos de usuarios creados bajo la discreción de usuarios suficiente privilegiados dentro del sistema. Ejemplos de grupos podrían ser miembros de una cierta materia o tipo de materia, por ejemplo "Base de Datos".
  - **Roles** permite definir usuarios por su propósito del sistema con un fin de aplicarles un conjunto global de permisos asociados por el tipo de usuario es. Ejemplos de roles podrían ser Estudiante, Docente y Administrador.
- **Administración** permite la administración, mantenimiento y recolección de datos por usuarios dentro del sistema. Para cumplir con estas responsabilidades se divide en:
  - **Subsistema de Gestión de Configuraciones** que permite a administradores cambiar de forma rapida y facil las configuraciones internas del sistema.
  - **Subsistema de Recolección de Datos de Uso** que collecta automáticamente datos del uso del sistema.
  - **Subsistema de Análisis de Datos de Uso** que permite a administradores visualizar y analizar los datos que ha recolectado el Subsistema de Recolección de Datos de Uso.
  - **Subsistema de Administración y Mantenimiento** que permite a administradores ver en tiempo real y también registros históricos de consumo de recursos. Este subsistema también proporciona a administradores las herramientas que necesita para gestionar y controlar problemas como usuarios maliciosos, sistemas caídos entre otros problemas detectados.
- **Presentación** que se encarga de presentar interfaces web a varias tipos de usuario, incluyendo:
  - **Estudiantes**
  - **Docentes**
  - **Administradores**
- **Lógica de Negocios** que se encarga de llevar a cabo los procesos de negocio por el cual la aplicación tiene propósito. Esto se subdivida en cuatro subsistemas:
  - **Subsistema de Docentes y Educación** que se lleva a cabo procesos de negocio que tienen que ver con enseñanza y evaluación. Dentro de esta subsistema también se lleva las tareas de administración asociadas con un docente y sus aulas de alumnos.

- **Subsistema de Gestión de Código** lleva el control del código y los procesos asociados en cuanto.
- **Subsistema de Gestión de Notas** maneja los procesos de negocio asociados con la sincronización y auditoría de notas generadas dentro del sistema.
- **Subsistema de Ejecución de Código** que lleva a cabo procesos de negocio que tienen que ver con ejecución arbitraria de código y por lo tanto requiere un cierto nivel de seguridad y aislamiento elevado a través de entornos virtualizados. Dentro de los procesos que se incluyen aquí, hay compilación, ejecución y calificación de código automatizado mediante pruebas unitarias.

- **Persistencia** que se encarga de asegurar que se persisten datos importantes para uso futuro y también la recuperación de los mismos. Esto se subdivide en tres clases de persistencia:
  - Persistencia a través de un **ORM Relacional** que guarda y recupera datos estructurados de la aplicación en bases de datos relacionales y orienta los mismos datos a objetos dentro de la aplicación para permitir su facil manipulacion.
  - Persistencia a través de un **ORM NoSQL** que guarda y recupera datos no estructurados de la aplicación en bases de datos no relacionales y orienta los mismos datos a objetos dentro de la aplicación para permitir su facil manipulacion.
  - Persistencia de código a través de un **Gestor de Persistencia de Código Externo** que guarda y recupera codigo de usuarios para su manipulación dentro de la aplicación y persistencia fuera del mismo.

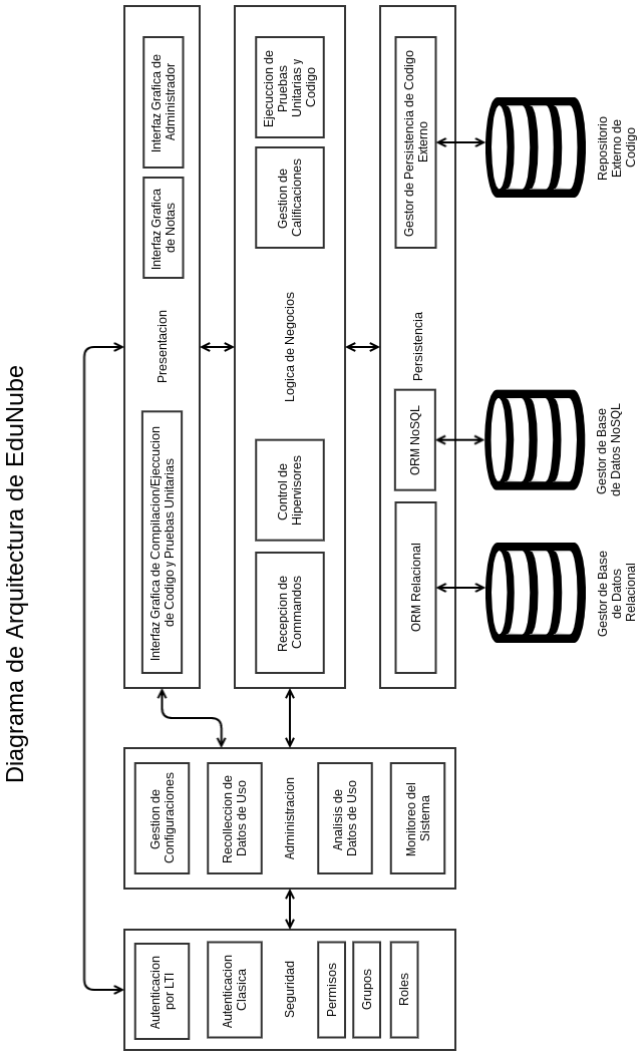

FIGURA 3.5: Diagrama de arquitectura para el sistema EduNube.

Como se puede observar en la figura 3.5, a un nivel global, el sistema EduNube consiste en 5 capas que son:

- **Seguridad** auténtica administradores y sistemas externos y controla el acceso que tiene cada uno de ellos. Esto se realiza con los siguientes componentes:
  - **Autenticación por Criptografía Asimétrica** usa pares de llaves públicas y privadas suficientemente grandes para ser difíciles de falsificar matemáticamente para autenticar sistemas externas.
  - **Autenticación Clásica** permite el login (autenticación) de administradores mediante la forma clásica de un usuario y contraseña.
  - **Permisos** permite otorgar o revocar partes específicas del sistema a usuarios específicos. Ejemplos de permisos podrían ser quien puede crear máquinas virtuales o ver el consumo de recursos del sistema.
  - **Grupos** permite aplicar el sistema de permisos a grupos de usuarios creados bajo la discreción de administradores suficiente privilegiados dentro del sistema. Ejemplos de grupos podrían ser ciertos tipos de usuario que representa algun tipo de estudiante en algún tipo de materia, por ejemplo para la materia de "Base de Datos".
  - **Roles** permite definir usuarios por su propósito del sistema con un fin de aplicarles un conjunto global de permisos asociados por el tipo de usuario es. Ejemplos de roles podrían ser Sistema Externa y Administrador.
- **Administración** permite la administración, mantenimiento y recolección de datos por usuarios dentro del sistema. Para cumplir con estas responsabilidades se divide en:
  - **Subsistema de Gestión de Configuraciones** que permite a administradores cambiar de forma rapida y facil las configuraciones internas del sistema.
  - **Subsistema de Recolección de Datos de Uso** que collecta automáticamente datos del uso del sistema.
  - **Subsistema de Análisis de Datos de Uso** que permite a administradores visualizar y analizar los datos que ha recolectado el Subsistema de Recolección de Datos de Uso.
  - **Subsistema de Monitoreo** que permite a administradores ver en tiempo real y también registros históricos de consumo de recursos. Este subsistema también proporciona a administradores las herramientas que necesita para gestionar y controlar problemas como usuarios maliciosos, sistemas caídos entre otros problemas detectados.
- **Presentación** que se encarga de presentar interfaces web a varias tipos de usuario, incluyendo:
  - **Compilación/Ejecución de Código y Pruebas Unitarias** proporciona las interfaces gráficas utilizadas durante la compilacion y ejecucion de codigo en adición a la aplicación de pruebas unitarias contra un codigo para ayudar en la calificación del mismo.
  - **Notas** permite a docentes y sistemas externos ver las notas asignados a un código.

    - **Administradores** proporciona las interfaces gráficas necesarias para ayudar un administrador cumplir con sus responsabilidades de monitoreo, control y mantenimiento del sistema en ambientes de producción.

- **Lógica de Negocios** que se encarga de llevar a cabo los procesos de negocio por el cual la aplicación tiene propósito. Esto se subdivida en cuatro subsistemas:

    - **Subsistema de Recepción de Comandos** que se lleva a cabo procesos de negocio que tienen que ver con recibir comandos de sistemas externas y procesar las mismas.
    - **Subsistema de Control de Hipervisores** lleva el control de los hipervisores conectados y les guía en el proceso de virtualizar y ejecutar código dentro de máquinas virtuales de los mismos hipervisores.
    - **Subsistema de Calificaciones** maneja los procesos de negocio asociados con la calificación, sincronización y auditoría de notas generadas dentro del sistema.
    - **Subsistema de Ejecución de Código y Pruebas Unitarias** que lleva a cabo procesos de negocio que tienen que ver con ejecución arbitraria de código en entornos virtualizados. Dentro de los procesos que se incluyen aquí, hay compilación, ejecución y calificación de código automatizado mediante pruebas unitarias.

- **Persistencia** que se encarga de asegurar que se persisten datos importantes para uso futuro y también la recuperación de los mismos. Esto se subdivide en tres clases de persistencia:

    - Persistencia a través de un **ORM Relacional** que guarda y recupera datos estructurados de la aplicación en bases de datos relacionales y orienta los mismos datos a objetos dentro de la aplicación para permitir su facil manipulacion.
    - Persistencia a través de un **ORM NoSQL** que guarda y recupera datos no estructurados de la aplicación en bases de datos no relacionales y orienta los mismos datos a objetos dentro de la aplicación para permitir su facil manipulacion.
    - Persistencia de código a través de un **Gestor de Persistencia de Código Externo** que guarda y recupera codigo de usuarios para su manipulación dentro de la aplicación y persistencia fuera del mismo.

Ventajas del Diseño Arquitectónico

Se ha considerado para ambos sistemas una arquitectura de n-capas debido a su capacidad de escalabilidad sobre varios niveles físicos. Además la separación lógica entre capas permite mayor mantenibilidad al largo plazo donde solo se necesitan reemplazar los componentes que requieren cambios en lugar de todo el sistema. El nivel de aislamiento entre estas mismas capas brinde mejor seguridad contra usuarios externos sean maliciosos o que quieren dar mal uso (un uso inadecuado) al sistema. A continuación se presenta la arquitectura de despliegue propuesta dentro de este trabajo de titulación.

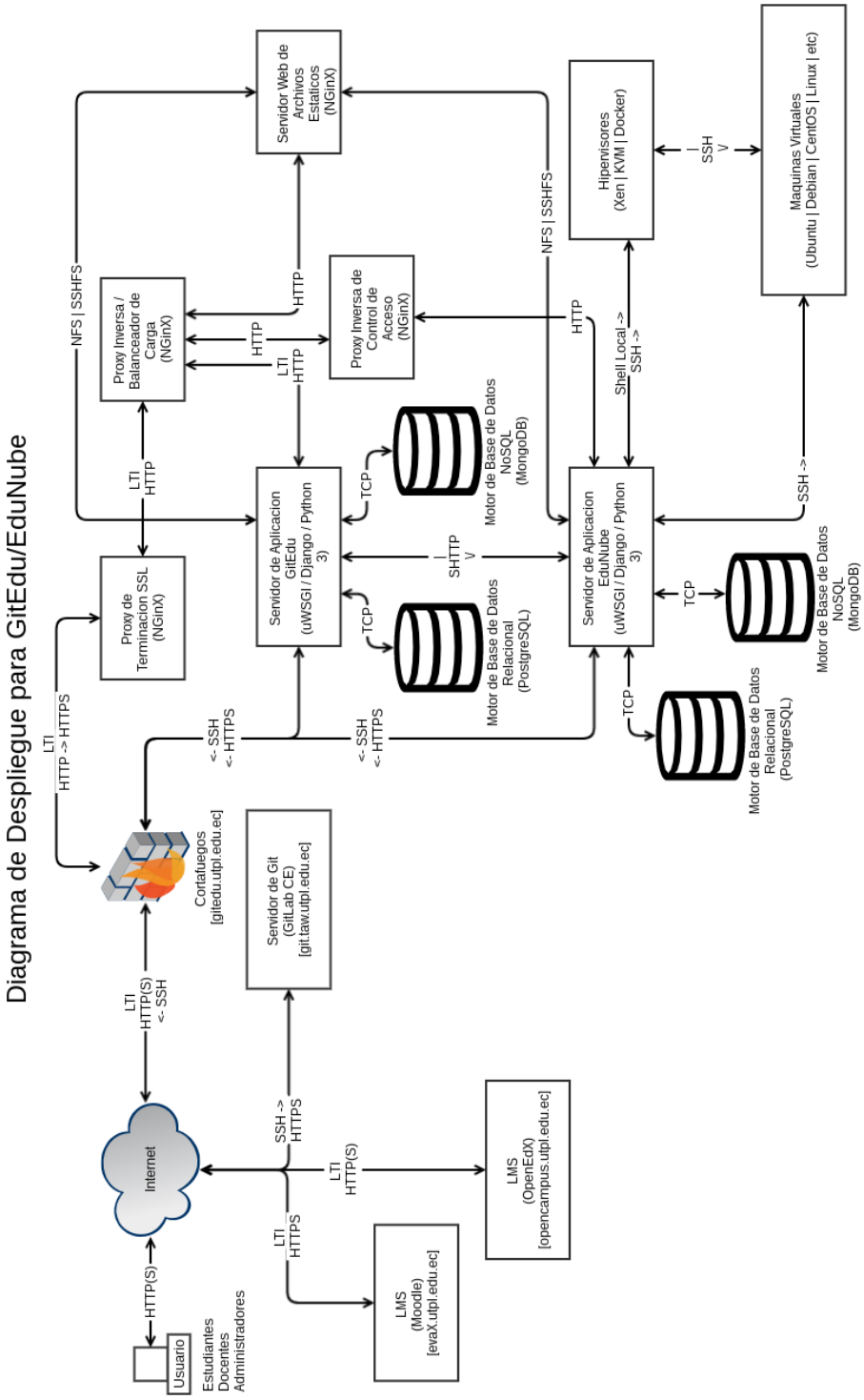

FIGURA 3.6: Diagrama de Despliegue para los sistemas GitEdu y EduNube.

3.2.3. Despliegue

Como se puede observar en la figura 3.6, el despliegue del sistema GitEdu junto al sistema EduNube, consiste en dos redes, una externa que contiene los usuarios finales (Estudiantes, Docentes, Administradores) en adición a sistemas externas como los dos LMS institucionales (Moodle y OpenEdX) y el servidor de Git (GitLab Community Edition), mientras que la red interna contiene las varias componentes que forman el conjunto de sistemas:

- **Cortafuegos** divide la red interna de la red externa y permite únicamente que pasa tráfico HTTP y HTTPS de manera bidireccional (con posibilidad de llamadas LTI sobre estos protocolos) y la salida de SSH al servidor de Git (para ofrecer mayor seguridad que transferencias sobre HTTP(S)).
- **Proxy de Terminación SSL** accepta HTTP y HTTPS, pero para asegurar la seguridad de usuarios finales y aplicaciones externas redirecciona toda uso del protocolo HTTP a HTTPS. Además descifra conexiones HTTPS entrantes para manejarlos por HTTP dentro de la red interna (reduciendo la carga en otros servidores sin abandonar seguridad en la red externa) y cifra conexiones salientes. Se utiliza NGinX por su bajo consumo de recursos en comparación con sus competidores como Apache.
- **Proxy Inversa / Balanceador de Carga** accepta todo el tráfico HTTPS (ahora por el protocolo HTTP dentro de la red interna) entrante y elige en base a la ruta cual servidor se lo puede responder. A futuro este componente permite que se puede escalar el sistema en lo que permite dividir tráfico entre varios servidores del mismo tipo (por ejemplo dos servidores de aplicación: GitEdu1 y GitEdu2). Se considera que NGinX es la solución más óptima para realizar este rol.
- **Servidor Web de Archivos Estáticos** se responsabiliza por peticiones de archivos estáticos que no conviene enviar a los servidores de aplicación (quienes los procesaría con mayor lentitud). Estos mismos tienen una conexión NFS o SSHFS con los mismos servidores de aplicación para que estos sean capaces de actualizar remotamente los archivos estáticos en caso de que los mismos cambien (por ejemplo cuando un usuario actualiza su foto de perfil). Estudios de rendimiento demuestran que NGinX cumple este rol de ser servidor web de archivos estáticos mejor que Apache.
- **Servidor de Aplicación "GitEdu"** se aloja el sistema GitEdu, desarrollado bajo el marco Django y el lenguaje de programación Python (version 3.x). Comunica con el servidor externo de Git sobre SSH y dispone tanto de un base de datos relacional como uno no relacional para persistir los distintos tipos de datos que tiene que manejar. Se conecta directamente al Servidor de Aplicación "EduNube" mediante SHTTP (HTTP sobre un túnel SSH) utilizando un par de llaves (pública y privada) como schéma de criptografía asimétrica. Se considera uWSGI como mejor servidor de aplicación para el sistema debido a que, a diferencia de su competencia (Gunicorn), esta suporta Websockets.
- **Motores de Base de Datos Relacional** se alojan datos estructurados para ambos sistemas, tanto GitEdu como NubeEdu.
- **Motores de Base de Datos NoSQL** se alojan datos no estructurados para ambos sistemas, tanto GitEdu como NubeEdu.

- **Proxy Inversa de Control de Acceso** restringe acceso al servidor de aplicaciones de EduNube cuando se accede al mismo desde la red externa (a través del proxy inversa).
- **Servidor de Aplicación "EduNube"** se aloja el sistema EduNube, desarrollado bajo el marco Django y el lenguaje de programación Python (version 3.x). Comunica con el servidor externo de Git sobre SSH y dispone tanto de un base de datos relacional como uno no relacional para persistir los distintos tipos de datos que tiene que manejar. Utiliza un shell local (dentro del mismo servidor) para manejo de un hipervisor local o en el caso de un hipervisor remoto, maneja el mismo sobre SSH. Las máquinas virtuales que se levanta el hipervisor, también se los controla con SSH. Se considera uWSGI como mejor servidor de aplicación para el sistema debido a que, a diferencia de su competencia (Gunicorn), esta suporta Websockets.
- **Hipervisores** gestionan máquinas virtuales por parte del sistema EduNube quien le envía comandos por SSH o un shell local. A su vez, controla sus máquinas virtuales mediantes SSH. Como tecnologías de hipervisor se considera Xen, KVM y Docker como buenas opciones para diferentes necesidades. Cada sistema de hipervisores soporta el uso de plantillas para la creación masiva de máquinas virtuales iguales para cada estudiante de una materia.
- **Máquinas Virtuales** ejecutan código de usuarios y reportan los resultados.

Ventajas de la Arquitectura de Despliegue

De la misma moda como el diseño de sistemas, subsistemas y módulos en adición al diseño arquitectónico, el diseño de arquitectura de despliegue está orientado a extensibilidad a través de modularidad, mantenibilidad por la naturaleza de sus capas y módulos, además de escalabilidad y seguridad mantenido a través de mecanismos de aislamiento de componentes que sean libres de escalar independientemente entre ellos, la arquitectura de despliegue se orienta a los mismos principios.

La separación de deberes dentro del sistema permite que cada servicio dentro de la arquitectura de despliegue se puede dedicar a hacer bien su parte de la manera más eficiente posible sin responsabilidades que no puede sostener de forma inadecuada, algo importante para empezar a poder garantizar rendimiento óptimo del sistema. El uso de un cortafuegos y varias proxies inversas adelantes de cada sistema también ayuda aislar las mismas del mundo externo a un nivel de protocolos reduce los superficies de ataque contra las mismas para ofrecer seguridad sin sacrificar rendimiento de una manera similar al punto anterior de separación de trabajo en roles y asignación al actor más adecuado para el mismo.

Este mismo uso de trabajadores, especializado cada uno en sus responsabilidades y posible carga de trabajo también sigue ofreciendo escalabilidad a futuro. Esto se realiza con comunicación entre servicios que está sobre protocolos de red para que las mismas o pueden estar en un solo recurso (sea servidor fisica o virtual) o en varios apartados lo cual permite a futuro la asignación de recursos físicos o virtuales como se lo ve la necesidad. Esta capacidad para escalar se puede ofrecer gracias al hecho de que los servicios semi independientes unos de otros, permitiendo n-capas sobre m-niveles físicos/lógicos.

Donde haya dependencias de, por ejemplo, sistema de ficheros compartidos para dos capas distintas se puede implementar con protocolos que permiten acceso remoto para las operaciones necesarios, en este caso que los servidores de aplicación sean

capaces de actualizar los archivos estáticos en el servidor para el mismo (se lo está considerando esta parte con un NFS o SSHFS). Obviamente en el caso de que estén en el mismo servidor, no se da la necesidad de establecer aquello conexión.

En casos donde se considera una necesidad de mayor seguridad en proteger canales de comunicación, se ha considerado que la mejor forma de brindar lo mismo es con tunnels SSH que encapsulan trafico que por si sola ocupa un protocolo adecuado según la necesidad. Por ejemplo para consumir el API de EduNube, GitEdu utiliza HTTP sobre SSH (también conocido como SHTTP). De esta forma se puede aprovechar lo que ya existe y que está comprobada como segura y eficiente (SSH como protocolo, especialmente en su segunda versión, obliga fuertes mecanismos de autenticación y seguridad mientras que al mismo tiempo ofrece menor sacrificio de rendimiento y alta funcionalidad).

A continuación se presenta el alcance del desarrollo, pruebas y despliegue para este trabajo de titulación.

3.2.4. Alcance

Debido al hecho de que se cuenta con tiempo y recursos limitados para realizar y acabar el trabajo de titulación, se considera un alcance limitado en ciertos aspectos con un fin de acabar con los componentes críticos del sistema dentro del tiempo dado.

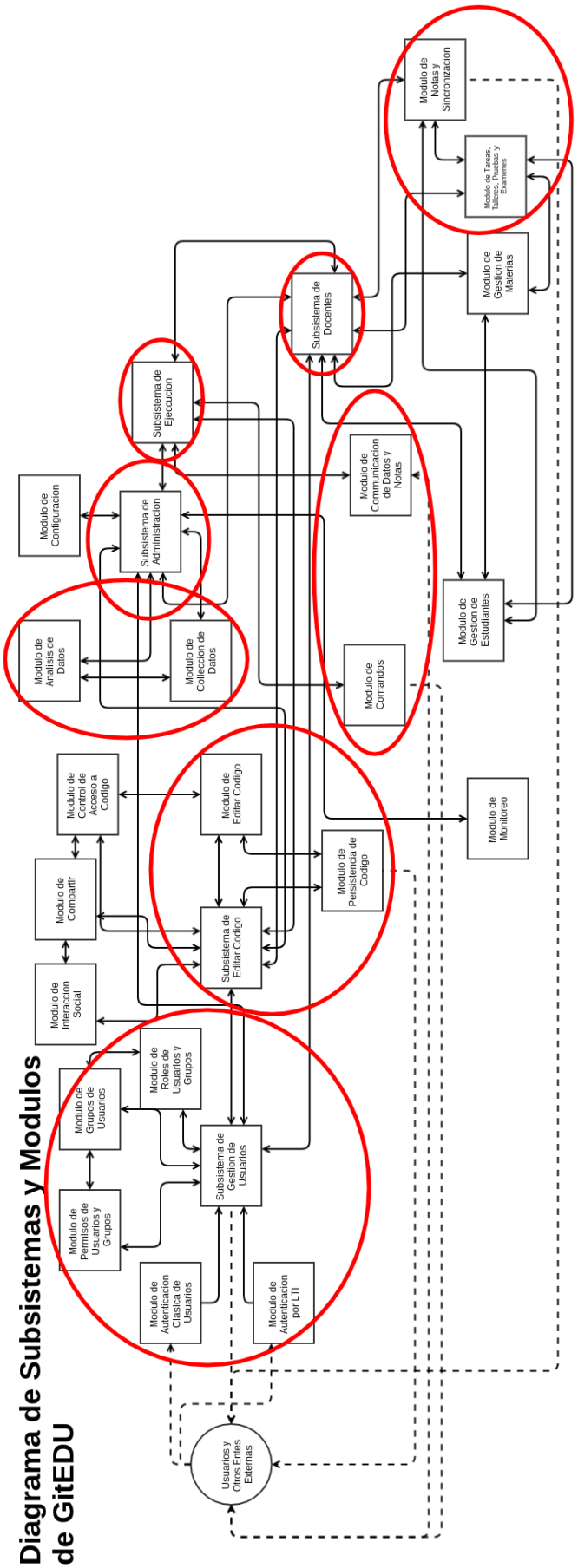

FIGURA 3.7: Alcance de Módulos para GitEdu.

Para el sistema GitEdu se considera los siguientes subsistemas y módulos críticos como se puede observar en la figura 3.7:

- **Todo el subsistema de Gestión de Usuarios.** Todos los módulos de este sistema son críticos porque se encargan de cómo los usuarios se autentican contra el mismo y el control de acceso que tienen cada uno. Por lo tanto el subsistema en sí es una parte importante para garantizar seguridad en el sistema final.
- **Parte del subsistema de Editar Código.** El subsistema de editar codigo se considera crítico los módulos tanto de editar como de persistir codigo. El módulo de editar codigo es crítico porque forma la funcionalidad principal de que se trata el sistema mientras que el modulo de persistir codigo viene a ser critico para sincronizar codigo entre los dos sistemas. No se toma en consideración como críticos los módulos de interacción social, compartir codigo ni control de acceso a codigo porque las mismas se tratan de llevar el sistema más allá de su propósito original y son más adecuados para un futuro trabajo.
- **Parte del subsistema de Docentes.** Dentro del subsistema de docentes, se considera crítico un módulo de crear talleres, pruebas y exámenes en adición a un módulo de notas y su sincronización. Entre estos dos se forma el núcleo de funcionalidad que requieren docentes dentro de la versión inicial. Dentro de trabajos futuros, se podría extender esta funcionalidad crítica con temas más administrativas como los módulos de gestión de materias y gestión de estudiantes que por el momento no se los considera como funcionalidades críticas para una primera versión estable, el producto de este trabajo de titulación.
- **Parte del subsistema de Administración.** Dentro del subsistema de administración, solo se considera que los componentes críticos se tratan de colección de datos y el análisis de los mismos debido a que esos dos módulos respectivamente ayudan en la toma de decisiones estratégicas que tienen que ver con la escuela de Ciencias de Computación y Electrónica y el uso del sistema. Los módulos de monitoreo y configuración, aunque podrían ser muy útiles en un ambiente de producción no caen dentro del alcance inicial de este tesis y por lo tanto no se los considera críticos bajo los parámetros del proyecto. De todas formas su implementación podría ser una funcionalidad adicional interesante para un futuro proyecto.
- **Todo el subsistema de Ejecución.** Todos los módulos de este sistema son críticos porque se encargan de la comunicación con el sistema EduNube. El módulo de comandos prepara comandos para ejecutar a través del módulo de comunicación de datos y notas se los envía en adición a conseguir resultados y sincronizar notas y calificaciones generadas por EduNube.

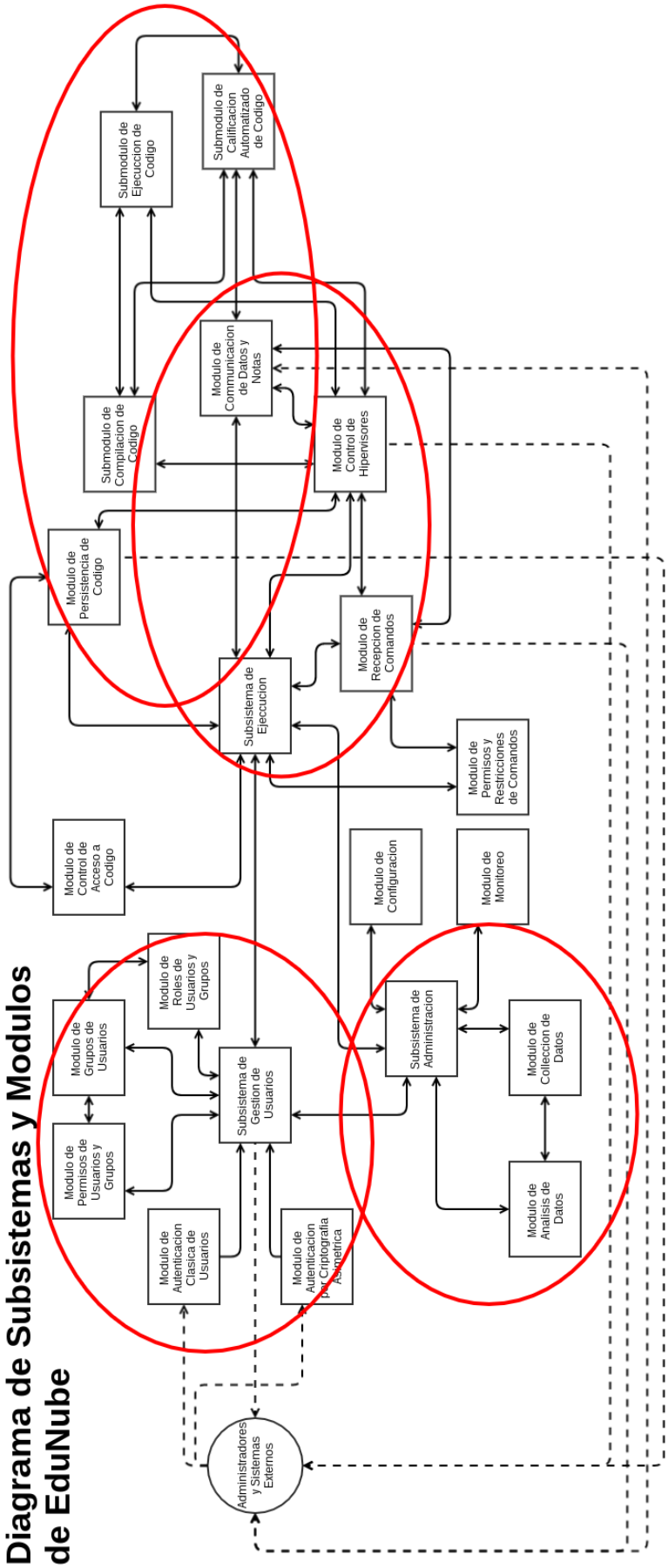

FIGURA 3.8: Alcance de Módulos para EduNube.

Para el sistema EduNube, se considera los siguientes subsistemas y módulos como críticos, de la forma que estan demostradas en la figura 3.8:

- **Todo el subsistema de Gestión de Usuarios.** Todos los módulos de este sistema son críticos porque se encargan de cómo los usuarios se autentican contra el mismo y el control de acceso que tienen cada uno. Por lo tanto el subsistema en sí es una parte importante para garantizar seguridad en el sistema final.
- **Parte del subsistema de Ejecución.** Dentro del subsistema de ejecución se consideran críticos los módulos de comandos (para procesar comandos entrantes), de control de hipervisores (para gestionar recursos virtuales), y de persistencia de código (para recuperar código para su compilación, compilación y ejecución). Además como componentes del módulo de control de hipervisores, se considera crítico los submódulos de compilación de código (para aquellos lenguajes que lo requieren), de ejecución (para ejecutar código), y de calificación automatizado de código (para calificar codigo de estudiantes). Se considera que los módulos de seguridad adicionales dentro de este subsistema no son necesarios por el momento debido al que único sistema que tendrá acceso a esta será la de GitEdu. A futuro, donde más sistemas y servicios empiezan a necesitar consumir el API de EduNube, llegará a ser necesario que tenga aquellos módulos de seguridad para proveer protección adicional.
- **Parte del subsistema de Administración.** Dentro del subsistema de administración, solo se considera que los componentes críticos se tratan de colección de datos y el análisis de los mismos debido a que esos dos módulos respectivamente ayudan en la toma de decisiones estratégicas que tienen que ver con la escuela de Ciencias de Computación y Electrónica y el uso del sistema. Los módulos de monitoreo y configuración, aunque podrían ser muy útiles en un ambiente de producción no caen dentro del alcance inicial de este tesis y por lo tanto no se los considera críticos bajo los parámetros del proyecto. De todas formas su implementación podría ser una funcionalidad adicional interesante para un futuro proyecto.

En cuanto el despliegue (figura 3.9), los sistemas externas que se toma en consideración para el alcance son Moodle como LMS, y GitLab Community Edition como servidor de control de versiones. En relación a los ambientes virtuales, solo se considera un hipervisor y un sistema operativo para las máquinas virtuales dentro de la versión final (durante el desarrollo se presenta los candidatos, comparación y selección de los mismos). También se incluye dentro del alcance el desarrollo y despliegue de los dos sistemas GitEdu y EduNube con sus respectivas bases de datos utilizando uWSGI como servidor de aplicación. No se incluye dentro del alcance OpenEdX o ningún otro LMS externa fuera de la mencionada previamente ni las configuraciones del cortafuego y servidores web, estos quedan como responsabilidades institucionales en cuanto como deseen implementar estos partes del despliegue. Lo que se presenta aquí es una sugerencia del autor basado en su experiencia práctica.

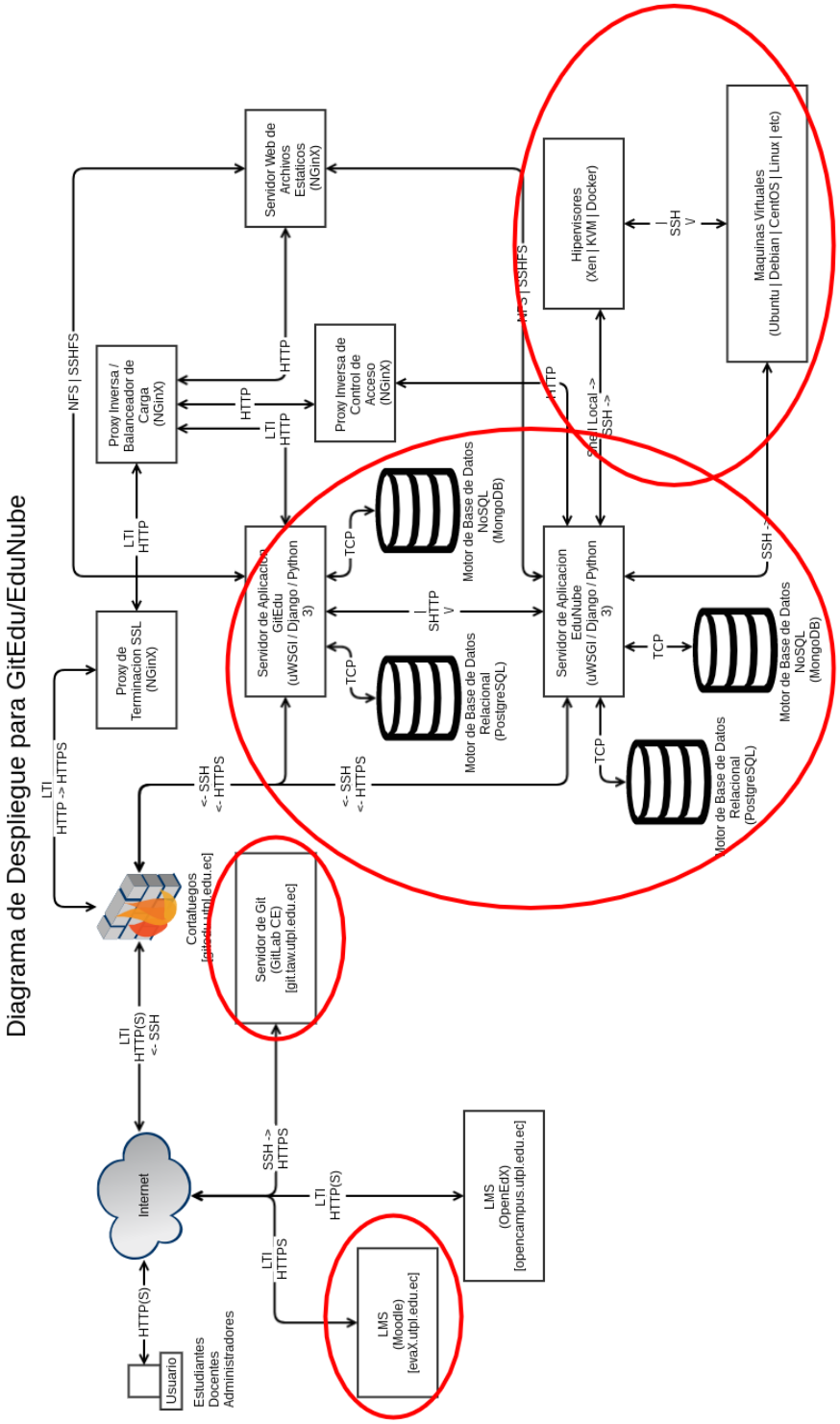

FIGURA 3.9: Alcance de Despliegue.

Apéndice A

Documento de Vision

A.1. Proposito

Ayudar a mejorar los métodos de enseñanza que ofrece la Universidad Técnica Particular de Loja en cuanto a la programación y uso de base de datos para las carreras de Sistemas y Electrónica.

Alcance Se propone un sistema de editar código en línea que a su vez integra LMS externos (para autenticación y notas), un servidor de control de versiones externo (para la persistencia de código), y un servicio web de ejecución de código en línea de una forma segura, eficaz y eficiente (para dar un ambiente de ejecución y pruebas tanto para los usuarios del sistema como para calificar de una forma automática).

A.2. Definiciones, Acrónimos y Abreviaciones

LMS Learning Management System (Sistema de Gestión de Aprendizaje)

LTI Learning Tools Interoperability (estandar de Interoperabilidad entre Herramientas de Aprendizaje)

GitEDU sistema de Git EDUcation

A.2.1. Posición y Oportunidad de Negocios

Con el avance continuo de la tecnología y su introducción en mas aspectos de la vida diaria de cada uno, hay una necesidad creciente de ingenieros en sistemas e electrónica que pueden programar e entender el software que hace todo funcionar en adición a los bases de datos que están por detrás de estos mismos sistemas.

Es por aquella razón que ahora está de moda ofrecer plataformas en línea para la enseñanza dinámica de la programación. GitEDU espere ofrecer las mismas funcionalidades a un costo institucional menor a travez de integración con sistemas existentes e innovación para proveer una mejor experiencia de usuario, tanto estudiantes como profesores para llevarse a cabo un mejor proceso de aprendizaje.

CUADRO A.1: Definición del Problema.

El problema de	enseñar y evaluar programación
afecta a	los estudiantes y docentes de las carreras de Sistemas y Electrónica
el impacto de lo cual es	el uso ineficiente de recursos universitarios en la enseñanza de la programación
Una solución exitosa seria	una aplicación web que ofrece un editor de código en línea, la capacidad de ejecutar este código para proveer mejor interacción con los estudiantes, la capacidad de ejecutar pruebas unitarias para automatizar el proceso de calificaciones, la persistencia de código en un repositorio de control de versiones remoto para su fácil revisión después por parte de profesores y estudiantes, y la integración transparente con sistemas de gestión de aprendizaje externos para la autenticación de usuarios y la respectivo registro de notas.

CUADRO A.2: Posición de Producto.

Para	docentes y estudiantes de la Universidad Técnica Particular de Loja
Quienes	tienen dificultades en la enseñanza y aprendizaje con la programación
GitEDU	es una plataforma web
Que	provee un espacio para la interacción entre profesores y alumnos para la enseñanza y aprendizaje de la programación y uso de las bases de datos
A diferencia de	otras plataformas altamente costosas que no se integran completamente con sistemas existentes de la universidad ni permiten alta interacción entre docentes y alumnos
Nuestro producto	da mayor capacidad para interacción entre estudiantes y sus docentes y se integra bien con las sistemas existentes para dar una mejor experiencia a todos los involucrados a un costo menor.

A.3. Usuarios e Interesados

A.3.1. Demográfica del Mercado

En el año 2011, la Universidad Técnica Particular de Loja contaba con aproximadamente 4000 estudiantes presenciales y 24000 estudiantes a distancia con una tendencia creciente [UTPL-Datos-Estadisticos]. En la experiencia personal del autor, las carreras de Sistemas y Electrónica, por lo menos en la modalidad presencial, juntos representan aproximadamente un 10 % de todos los estudiantes en la universidad lo cual daría un mercado de estudiantes afectados por un nuevo sistema de aproximadamente un mínimo 2800 estudiantes. Según el directorio de docentes de la universidad, son 60 profesores en el departamento de Ciencias de la Computación y Electrónica [UTPL-Directorio-Docentes]. Con eso se puede estimar un mínimo de 2860 usuarios lo los cuales el sistema propuesto podría llegar a afectar.

Es precisamente la parte de la población, de usuario potenciales mencionado

anteriormente, que está en el proceso de enseñar, evaluar y aprender habilidades de programación y consultar bases de datos que forman la base de usuarios de la aplicación.

Nombre	**Descripción**	**Responsabilidades**
Analista	Trabaja con el Asesor Principal y Auxiliar para entender bien las necesidades institucionales para poder llevar un buen ingeniería de requerimientos y diseño del sistema propuesto.	Definir bien el problema para analizarlo, generar requerimientos en base a las necesidades para diseñar y documentar componentes del sistema final para el beneficio del Arquitecto de Software, Programador, Gestor de Proyecto y futuro mano de obra en el proyecto.
Arquitecto de Software	Trabaja con el Asesor Principal y Auxiliar para definir una arquitectura que garantiza que se cumple con los atributos de calidad que se requiere el sistema y será compatible con infraestructura y sistemas institucionales ya existentes.	Diseñar los modelos de interacción entre todos los componentes internos y externos del sistema para con ello lograr un flujo eficaz y eficiente, que también cumple con los parámetros de los requerimientos no funcionales, en el sistema final.
Gestor de Proyecto	Trabaja con el Asesor Principal y Auxiliar para evaluar, estimar y establecer el alcance, los recursos y el cronograma del proyecto.	Distribuir de manera eficiente y eficaz los recursos para ayudar el analista, arquitecto de software, programador y administrador de sistemas y bases de datos cumplir dentro de los recursos, alcance y cronograma preestablecido.
Programador	Trabaja con el Asesor Principal, Asesor Auxiliar, y Analista para implementar soluciones técnicas que cumplen con el diseño dado por el analista.	Desarrollar el sistema en todos sus componentes.
Administrador de Sistemas y Bases de Datos	Trabaja con el Asesor Principal y Auxiliar para desplegar la aplicación en la institución.	El despliegue correcto del sistema con todos sus componentes en la institución respectivo.
Asesor Principal	Trabaja con el Asesor Auxiliar para poder aconsejar de la mejor manera el Analista, Arquitecto de Software, Gestor de Proyecto, Programador y Administrador de Sistemas y Bases de Datos.	La definición de necesidades y aprobación del producto final.

Nombre	Descripción	Responsabilidades
Asesor Auxiliar	Trabaja con el Asesor Principal para poder aconsejar de la mejor manera el Analista, Arquitecto de Software, Gestor de Proyecto, Programador y Administrador de Sistemas y Bases de Datos.	La definición de necesidades y aprobación del producto final.
Asesor de Documentación	Trabaja con el Analista y Gestor de Proyecto para asegurar la calidad de la documentación que se genera a lo largo del proyecto y que se cumple con el cronograma establecido entre el gestor del proyecto y los asesores principales y auxiliares.	La aprobación de los avances en la documentación y la documentación completa al final del proyecto.

CUADRO A.3: Resumen de Interesados.

CUADRO A.4: Resumen de Usuarios.

Nombre	Descripción	Responsabilidades	Interesado
Estudiantes	Usuario Final Primaria del Sistema	Use la aplicación para cumplir con las tareas, pruebas y exámenes que le pone el docente	Los mismos
Professores	Usuario Final Primaria del Sistema	Use la aplicación para dar tareas, pruebas y exámenes a los estudiantes y calificar los mismos	Los mismos
Administradores	Quienes administran el sistema en su ambiente de despliegue	Mantener el sistema	El mismo

A.3.2. Ambiente de Usuario

El sistema será disponible para el uso por usuarios que estén en el campus universitario y en sus casas.

A.3.3. Perfiles de Interesados

Para entender a fondo cada clase de interesado se da a continuacion un analisis de los mismos.

CUADRO A.5: Perfil de Interesado: Analista.

Descripción	El analista del equipo de desarrollo
Tipo	Miembro del Equipo de Desarrollo
Responsabilidades	Definir bien el problema para analizarlo, generar requerimientos en base a las necesidades para diseñar y documentar componentes del sistema final para el beneficio del Arquitecto de Software, Programador, Gestor de Proyecto y futuro mano de obra en el proyecto.
Criteria de Exito	Que se lleva a cabo exitosamente el proyecto bajo todos sus requerimientos funcionales y no funcionales
Involucramiento	En cada fase del proyecto
Entregables	Documentación del Sistema y su funcionamiento
Comentarios / Preocupaciones	Que se cumple con todas las necesidades institucionales

CUADRO A.6: Perfil de Interesado: Arquitecto de Software.

Descripción	El arquitecto de software en el equipo de desarrollo
Tipo	Miembro del Equipo de Desarrollo
Responsabilidades	Diseñar los modelos de interacción entre todos los componentes internos y externos del sistema para con ello lograr un flujo eficaz y eficiente, que también cumple con los parámetros de los requerimientos no funcionales, en el sistema final.
Criteria de Exito	Que se lleva a cabo exitosamente el proyecto bajo todos sus requerimientos no funcionales
Involucramiento	En cada fase del diseño y despliegue
Entregables	Modelos Arquitectónicos del Sistema y su interacción con otros sistemas
Comentarios / Preocupaciones	Que se cumple con todas las atributos de calidad que la institución manda

CUADRO A.7: Perfil de Interesado: Gestor de Proyecto.

Descripción	El gestor de proyecto en el equipo de desarrollo
Tipo	Miembro del Equipo de Desarrollo
Responsabilidades	Distribuir de manera eficiente y eficaz los recursos para ayudar el analista, arquitecto de software, programador y administrador de sistemas y bases de datos cumplir dentro de los recursos, alcance y cronograma preestablecido.
Criteria de Exito	Que se lleva a cabo exitosamente el proyecto bajo todos sus requerimientos y dentro de los recursos y cronograma preestablecido.
Involucramiento	En cada fase del proyecto
Entregables	Documentación de la gestión del proyecto y el adecuado seguimiento y control interno a lo largo del mismo
Comentarios / Preocupaciones	Que se cumple con todo el proyecto dentro de los recursos y cronograma preestablecido

CUADRO A.8: Perfil de Interesado: Programador.

Descripción	El programador del equipo de desarrollo
Tipo	Miembro del Equipo de Desarrollo
Responsabilidades	Desarrollar el sistema en todos sus componentes.
Criteria de Exito	Que se lleva a cabo exitosamente el proyecto bajo todos sus requerimientos funcionales y no funcionales
Involucramiento	En cada fase del desarrollo del proyecto
Entregables	Código del Sistema
Comentarios / Preocupaciones	Que se logra programar segun la especificacion que el analista le da

CUADRO A.9: Perfil de Interesado: Administrador de Sistemas y Bases de Datos.

Descripción	El administrador de sistemas y bases de datos del equipo de desarrollo
Tipo	Miembro del Equipo de Desarrollo
Responsabilidades	El despliegue correcto del sistema con todos sus componentes en la institución respectivo.
Criteria de Exito	Que se lleva a cabo exitosamente el proyecto bajo todos sus requerimientos funcionales y no funcionales
Involucramiento	En cada fase del desarrollo y despliegue del proyecto
Entregables	Documentación de los Servicios Desplegados
Comentarios / Preocupaciones	Que se logra desplegar la aplicación según la especificación del arquitecto de software

CUADRO A.10: Perfil de Interesado: Asesor Principal.

Descripción	El asesor principal para equipo de desarrollo
Tipo	Asesor del Equipo de Desarrollo
Responsabilidades	La definición de necesidades y aprobación del producto final.
Criteria de Exito	La entrega del producto final
Involucramiento	En cada fase del proyecto
Entregables	Aprobación del Producto Final
Comentarios / Preocupaciones	Que se logra realizar la aplicación.

CUADRO A.11: Perfil de Interesado: Asesor Auxiliar.

Descripción	El asesor auxiliar para equipo de desarrollo
Tipo	Asesor del Equipo de Desarrollo
Responsabilidades	La definición de necesidades y aprobación del producto final.
Criteria de Exito	La entrega del producto final
Involucramiento	En cada fase del proyecto
Entregables	Aprobación del Producto Final
Comentarios / Preocupaciones	Que se logra realizar la aplicación.

CUADRO A.12: Perfil de Interesado: Asesor de Documentación.

Descripción	El asesor de documentación para equipo de desarrollo
Tipo	Asesor del Equipo de Desarrollo
Responsabilidades	La aprobación de los avances en la documentación y la documentación completa al final del proyecto.
Criteria de Exito	La entrega de la documentación final
Involucramiento	En cada fase del proyecto
Entregables	Aprobación de la Documentación Final y los respectivos avances
Comentarios / Preocupaciones	Que se logra realizar la documentación.

A.3.4. Perfiles de Usuarios

Para entender a fondo cada clase de usuario se da a continuacion un analisis de los mismos.

CUADRO A.13: Perfil de Usuario: Estudiante.

Descripción	Estudiantes de la Modalidad Presencial y a Distancia de las carreras de Sistemas y Electrónica
Tipo	Usuario Final Primaria
Responsabilidades	Probar el sistema
Criteria de Exito	Que el sistema sea de utilidad para su experiencia educativa
Involucramiento	En la fase de pruebas
Entregables	Ninguno
Comentarios / Preocupaciones	Que la aplicación sea fácil de usar

CUADRO A.14: Perfil de Usuario: Professor.

Descripción	Profesores de Programación y de Bases de Datos de la Modalidad Presencial y a Distancia de las carreras de Sistemas y Electrónica
Tipo	Usuario Final Primaria
Responsabilidades	Probar el sistema
Criteria de Exito	Que el sistema sea de utilidad para su docencia
Involucramiento	En la fase de pruebas
Entregables	Ninguno
Comentarios / Preocupaciones	Que la aplicación les facilita el proceso de enseñanza

CUADRO A.15: Perfil de Usuario: Administrador.

Descripción	Administradores Institucionales de Sistemas y de Bases de Datos
Tipo	Usuario Final Secundaria
Responsabilidades	Mantener el Sistema
Criteria de Exito	Que el sistema sea fácil de mantener
Involucramiento	En las fases de pruebas y despliegue
Entregables	Ninguno
Comentarios / Preocupaciones	Que la aplicación sea lo suficientemente documentado

A.3.5. Necesidades de Interesados y Usuarios Principales

En base a los interesados y usuarios principales definidos, se define las necesidades del sistema a continuacion.

Necesidad	**Prioridad**	**Preocupaciones**	**Solución Seleccionado**	**Soluciones Propuestas**
Seguridad en el Ambiente de Ejecución de Código	Alta	Los partes del sistema que se encargan de ejecución de código, necesitan alta protección contra usuario maliciosos y daños accidentales.	Ver las soluciones propuestas	Virtualización: ▪ Hipervisor de Tipo 1 ▪ Hipervisor de Tipo 2 ▪ Contenerización
Extensibilidad	Alta	El sistema debe soportar funcionalidad agregadas en el futuro.	Aplicación orientado a la Modularidad, ver la solución propuesta para mayor detalle.	Modularidad entre componentes del sistema para facilitar el proceso de agregar componentes nuevo o reemplazar componentes existentes sin tocar los demás
Facilidad en Autenticación	Mediana	El sistema debe ser fácil para autenticar todos sus usuarios.	Autenticación contra LMS existente mediante LTI	Autenticación contra LMS existente mediante LTI

Necesidad	**Prioridad**	**Preocupaciones**	**Solución Seleccionado**	**Soluciones Propuestas**
Facilidad en Notas	Mediana	El sistema debe ser capaz de registrar notas en sistemas externas sin interacción del profesor.	Registro de Notas mediante LTI	Registro de Notas mediante LTI
Facilidad en Persistencia de Código	Mediana	El sistema, sin necesidad de interacción del usuario debe persistir el código escrito en un servidor de algún sistema de control de versiones externa	Transferencia de código mediante sistemas de control de versiones ya existentes	Transferencia de código mediante sistemas de control de versiones ya existentes

CUADRO A.16: Necesidades de Interesados y Usuarios Principales

A.3.6. Alternativas y Competidores

1. Repl.it
2. Cloud9 IDE

A.4. Vista General de Producto

A.4.1. Perspectiva del Producto

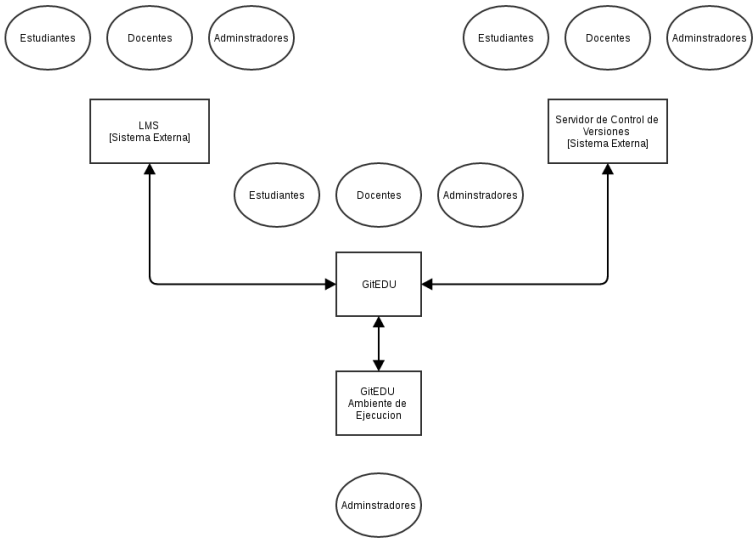

FIGURA A.1: Perspectiva del Producto.

A.4.2. Resumen de Capacidades

CUADRO A.17: Resumen de Capacidades.

Beneficio al Cliente	**Característica de Apoyo**
Facilidad de Autenticación	Autenticación por LTI contra LMS institucional
Gestión de la Configuración para el Código Escrito	Uso de Servidores Externos de Control de Versiones
Ejecución y Pruebas Unitarias en Línea	Sistema de Apoyo con Ambiente de Ejecución de Código
Generación y Sincronización de Notas	Calificación a través de pruebas unitarias, Sincronización de Notas a través de LTI con LMS institucional

A.4.3. Presuposiciones y Dependencias

1. La institución seguirá usando Moodle y Open EDX como sus plataformas y proveedores de LMS.

2. La institución seguirá usando como proveedor de un servidor de control de versiones de la institución, un servidor institucional con GitLab Community Edition.

3. Alternativas como Repl.it y Cloud9 IDE no dan la funcionalidad necesaria o son demasiados costosos para su uso general por la institución.

A.5. Caracteristicas de Producto

1. Caracteristicas de Autenticación
   a) Autenticarse por LTI
   b) Autenticarse con Usuario y Contraseña
   c) Cerrar Session
2. Caracteristicas de Editar Código
   a) Nuevo Proyecto
   b) Nuevo Proyecto basado en otro Proyecto
   c) Ver Proyectos
   d) Ver Detalles de un Proyecto
   e) Editar Detalles de un Proyecto
   f) Gestionar Usuarios de un Proyecto
   g) Gestionar Permisos de un Proyecto
   h) Abrir Proyecto
   i) Editar Archivos en un Proyecto
   j) Crear Nuevo Archivo en un Proyecto
   k) Borrar Archivo de un Proyecto
   l) Interactuar con Usuarios dentro de un Proyecto
3. Caracteristicas de Persistencia de Codigo
   a) Guardar cambios en un Proyecto
   b) Subir cambios en un Proyecto
   c) Bajar cambios en un Proyecto
4. Caracteristicas de Ejecutar y Calificar Código
   a) Ejecutar Codigo en ambiente interactivo en el navegador
   b) Agregar Pruebas Unitarias a un Proyecto
   c) Ejecutar Pruebas Unitarias en un Proyecto
   d) Calificar un Proyecto
   e) Calificiar un Proyecto de forma Automática en base a Pruebas Unitarias
   f) Enviar Calificaciones a un Sistema Externo

A.6. Precedencia y Prioridades

CUADRO A.18: Precedencia y Prioridades.

Prioridad	Característica (Según su número)
Alta	1.a, 1.c, 2.a, 2.b, 2.c, 2.h, 2.i, 3.a, 3.b, 4.a
Media	1.b, 2.f, 2.g, 2.j, 3.c, 4.b, 4.c, 4.d, 4.f
Baja	2.d, 2.e, 2.k, 2.l, 4.e

A.7. Restricciones

A.7.1. Seguridad

Protección contra ataques en la red
Aislamiento de ambientes de ejecución de código de usuarios

A.7.2. Extensibilidad

Soporte al nivel de sistema y documentación para extensión con más lenguajes de programación y motores de base de datos al futuro

A.7.3. Usabilidad

Ser usable
Requerir un mínimo de interacción del usuario para que puede enfocarse en realizar sus responsabilidades en el sistema

A.7.4. Escalabilidad

Tener capacidad para escalar frente mayor carga en el futuro

A.7.5. Rendimiento

Minimizar las requerimientos mínimos de hardware requerido

A.8. Otros Requisitos de Producto

A.8.1. Normas

Ninguna.

A.8.2. Requisitos de Sistema

Ninguno.

A.8.3. Requisitos de Rendimiento

Ninguno.

A.8.4. Requisitos Ambientales

Ninguno.

A.9. Requisitos de Documentación

A.9.1. Manual del Programador

Documentación para explicar el funcionamiento de la aplicación para que futuros desarrolladores pueden extender sin mayor dificultad la aplicación.

A.9.2. Manual de Mantenimiento

Documentación para definir los procesos de mantenimiento de la aplicación para guiar la gobernanza y administración del mismo.

A.9.3. Manual de Usuario

Documentación para enseñar el uso de la aplicación a docentes y alumnos.

Apéndice B

Especificacion de Requisitos de Software

B.1. Introduccion

B.1.1. Proposito

El siguiente documento espera dar una descripción detallado de los requisitos del Sistema Git Education (GitEDU) con las finalidades de definir la intención de la misma, declarar de forma completa todos los componentes a ser desarrollados del sistema, y también explicar limitaciones del sistema además de sus interacciones y interfaces con otros sistemas. Se destina el siguiente documento para revisión por el equipo de asesores y como una referencia de alcance para el equipo de desarrollo.

B.1.2. Alcance

GitEDU es un sistema de integración para promover la programación estudiantil y el sistema educativo que soporta el mismo. El sistema final debe disponer de buena documentación y alta calidad de código para sostener su futuro desarrollo y mantenimiento por parte de una comunidad de profesionales y profesionales en formación.

Docentes de las carreras que involucran la enseñanza de programación, actualmente tanto la carrera de Sistemas Informáticos y Computación como la carrera de Electrónica y Telecomunicaciones, pueden crear dentro de la plataforma, deberes, talleres, pruebas y exámenes, y a través de los LMS institucionales, compartir las mismas con sus alumnos. Los alumnos pueden entrar a través de un enlace que les comparte su docente dentro del LMS institucional y con un canal de comunicación LTI, la plataforma GitEDU les identifica y autentica sin interacción del usuario para que el estudiante puede empezar directamente con el trabajo que tiene que realizar sin digitar sus credenciales. En el curso de su actividad, se va guardando su progreso periódicamente contra un servidor de control de versiones institucional (GitLab CE) y en cualquier momento el estudiante puede probar e interactuar con su código a través de un terminal virtual de Linux que se encuentra dentro de la misma interfaz. Una vez que se termine su trabajo y desee enviarlo, se lo envía y el código escrito se auto califica contra un conjunto de pruebas unitarias ocultas que el docente agregó a la actividad al crearlo. La nota que se genera se lo envía al LMS institucional a través del mismo canal LTI. En caso de que el docente decide no utilizar pruebas unitarias o calificar cada trabajo a mano o realizar una calificación híbrida entre las pruebas unitarias y la parte a mano, no se reflejarán las notas en el LMS hasta que el docente se ha terminado de calificar el código en la plataforma de GitEDU.

Además el sistema requiere una conexión de internet y la disponibilidad de sistemas de apoyo como el servidor de control de versiones, el LMS y el servicio de ejecución de código.

B.1.3. Definiciones, Acrónimos y Abreviaciones

CUADRO B.1: Definiciones, Acrónimos y Abreviaciones para GitEDU

Termino	**Definicion**
Docente / Professor	Alguien quien pertenece a la facultad de la institución y cuyo responsabilidad es enseñar a sus alumnos / estudiantes programación.
Alumno / Estudiante	Un miembro del cuerpo estudiantil de la institución y está en el proceso de aprender la programación.
Administrador	Quien se administra y mantiene el sistema.
Materia / Clase	Un conjunto de Docente(s) y Alumno(s) que están el proceso educativo de programación juntos en una aula física o virtual.
GitLab CE	GitLab Community Edition, un servidor de control de versiones que utiliza el VCS Git
LMS	Learning Management System, un sistema que apoya en la interacción educativa diaria entre Docentes y sus Alumnos
LTI	Learning Tools Interoperability, una estandar y protocolo que permite la comunicación e interacción entre varias herramientas educativas.
VCS	Sistema de Control de Versiones, permite almacenar y visualizar una historia de versiones de archivos o código para dar trazabilidad para lo mismo.

B.1.4. Referencias

Como fuente de la plantilla para el ERS, se ocupó: `http://www.cse.chalmers.se/~feldt/courses/reqeng/examples/srs_example_2010_group2.pdf`

B.2. Descripción General

B.2.1. Perspectiva de Producto

La plataforma GitEDU se descompone en dos subsistemas: el subsistema de editar código y el subsistema de ejecutar código. El subsistema de editar código consiste

en la integración de autenticación y notas por el lado del LMS, el uso de persistencia del código por el servidor de control de versiones por otro lado, el consumo de un servicio de ejecución de código por un tercer lado y un editor de código en línea. El subsistema de ejecutar código consiste en esperar llamadas del subsistema anterior, validar la autenticidad, establecer permisos, limitaciones y parámetros de ejecución, gestionar recursos para ejecución, la ejecución interactiva y devolución de resultados al otro subsistema.

Debido a la necesidad de mantener seguridad y prevenir la ejecución indebido de código, el subsistema para lo mismo debe ser aislado de todos los usuarios y solo accedido desde el otro subsistema a través de un API que permite su interoperabilidad. La solución para la ejecución de código debe ser liviana debido a número grande de usuarios concurrentes que puede tener a la vez.

Funcionalidades de Producto Para entrar al subsistema de editar código en línea, se debe autenticarse contra el LMS institucional. El editor de código en línea debe soportar una variedad grande de lenguajes y estar abierto a la agregación de más lenguajes en un futuro. La persistencia de código en un servidor de control de versiones debe ser transparente y no requerir ningún interacción del usuario. El interfaz para interactuar con el ambiente virtual debe ser simple y fácil de manejar sin mayor conocimiento de las tecnologías que le respaldan. Se debe poder acompañar el editor de código con documentación que guía el estudiante en explicar o resolver el problema actual. La manera en que docentes pueden generar nuevos ejercicios, su documentación y pruebas unitarias para la calificación debe ser intuitivo.

B.2.2. Caracteristicas de Usuario

Los tres tipos de usuarios que interactúan con el sistema son Docentes, Alumnos y Administradores. Cada tipo de usuario tiene sus propias necesidades y requerimientos.

Docentes necesitan un sistema que sea fácil de manejar y que les simplifica y automatiza el proceso de enseñar y evaluar a sus estudiantes. Eso significa tener una interfaz intuitiva para docentes que les permite escribir, documentar y generar pruebas unitarias para probar funcionalidad de código que escribe estudiantes para los ejercicios.

Alumnos necesitan un sistema que les brinde una buena experiencia de aprendizaje y les proporciona todas las herramientas que requieren para desarrollar en linea. Igual que sus profesores, deben disponer de una interfaz intuitiva que les ayuda cumplir con sus responsabilidades de estudiar, aprender y demostrar conocimientos en una cantidad de tiempo óptimo.

Administradores necesitan un sistema que sea seguro y cuyos componentes sean fáciles de mantener y extender a lo largo del tiempo. Para ello se necesita tener una buena documentación del funcionamiento del sistema y todos sus componentes para tener la misma como referencia.

B.2.3. Limitaciones

Se requiere de hardware con alta capacidad de virtualización y conectividad al internet y la intranet institucional para poder brindar la mejor experiencia al usuario.

B.2.4. Suposiciones y Dependencias

Se supone que siempre habrá la conectividad necesaria para poder usar todos los componentes del sistema.

B.2.5. División de Requerimientos

En caso de que falta recursos de tiempo o algún otro recurso, se limitará el alcance del proyecto para enfocarse en los requerimientos de una prioridad más alta y dejar los demás requerimientos para trabajos futuros.

B.3. Requisitos Específicos

B.3.1. Interfaces Externos

Interfaces de Hardware

El sistema que interactúa directamente con los usuarios no tiene ningún requisito especial de hardware solo que si se debe contar con memoria, disco y procesador suficiente para un servidor web, servidor de aplicación y servidor de base de datos. El subsistema de virtualización debe contar con los recursos necesarios y el hardware adecuado para soportar la carga del muchos estudiantes a la vez utilizando la tecnología seleccionada.

Interfaces de Software

El sistema que interactúa directamente con los usuarios debe ser capaz de autenticar los mismos contra LMS existentes y sincronizar notas que tiene con el mismo sistema LMS. Además el código que se desarrolla dentro de esta plataforma debe ir contra el servidor del sistema de control de versiones institucional. Y finalmente debe haber comunicación mediante APIs que permite el intercambio de código, entradas y salidas de ejecución en adición a notas resultantes de pruebas unitarias entre el sistema de desarrollo en línea y el sistema de ejecución de código en línea.

Interfaces de Comunicación

La comunicación entre los dos subsistemas es vital para lograr el nivel de funcionalidad que los usuarios finales requieren. Además, como aplicación en línea, se supone que todos los usuarios finales deben tener conectividad previo y durante el uso del sistema.

B.3.2. Requerimientos Funcionales

1. **Estudiantes**

   a) **Autenticación mediante LTI**

   Título: Autenticación mediante LTI

   Descripción: Un estudiante debe poder autenticarse en la aplicación por medio del estándar LTI a través de su sesión abierta en sistemas LMS institucionales.

Razón: Para reducir el número de credenciales que un estudiante necesita recordar y actualizar en adición a promover el uso del sistema en brindar mayor facilidad de acceso.

Dependencias: Ninguno.

b) **Ambiente Completo de Programación**

Título: Ambiente Completa de Programación

Descripción: Un estudiante debe contar con un ambiente de programación completa dentro de la aplicación con el cual puede desarrollar y probar de forma continua.

Razón: Para dar a los estudiantes todas la herramientas necesarias para cumplir con sus responsabilidades, estudio y aprendizaje dentro de un solo entorno.

Dependencias: Ninguno.

c) **Persistencia de Codigo**

Título: Persistencia de Código

Descripción: El código que se escribe dentro del plataforma debe constar respaldado externamente en un servidor de control de versiones independiente.

Razón: Para llevar incrementalmente respaldos del código que escriben estudiantes.

Dependencias: Ambiente Completa de Programación.

d) **Sistema de Documentación**

Título: Sistema de Documentación

Descripción: Tanto estudiantes como sus profesores deben disponer de las herramientas necesarias para documentar y expresar tanto necesidades como funcionalidades de código desarrollado en el plataforma de una manera clara y efectiva.

Razón: Para ayudar profesores y sus alumnos llevar más efectivamente la parte de enseñanza y comunicación que requiere un sistema educativa de esta naturaleza.

Dependencias: Ambiente Completa de Programación.

e) **Sistema de Compartir**

Título: Sistema de Compartir

Descripción: Un usuario del sistema debe poder compartir sus proyectos de programación y la documentación que acompañan los mismos con otros usuarios.

Razón: Para promover cooperación entre usuarios y interacción entre estudiantes y profesores que son importantes para el proceso de enseñanza y aprendizaje.

Dependencias: Ambiente Completa de Programación.

1) **Sistema de Permisos**

Título: Sistema de Permisos

Descripción: Usuarios deben poder controlar el acceso y la naturaleza del mismo de otros usuarios con los mismos se ha compartido el código.

Razón: Para brindar seguridad para el sistema de compartir y para proveer la flexibilidad necesario para tener una gran variedad de usos posibles al mismo sistema, con el suficiente nivel de seguridad (a través de los permisos) para cada uno de ellos.

Dependencias: Sistema de Compartir.

2) **Editar Varios Usuarios al Mismo Tiempo**

Título: Editar Varios Usuarios al Mismo Tiempo

Descripción: Usuarios quienes tengan acceso a un código compartido previamente y tienen permisos de editarlo deben poder editarlo al mismo tiempo de tal forma que los cambios que realizan aparecen a todos en tiempo real.

Razón: Para promover cooperación entre usuarios y interacción entre estudiantes y profesores que son importantes para el proceso de enseñanza y aprendizaje.

Dependencias: Sistema de Compartir.

f) **Sistema de Interacción Social**

Título: Sistema de Interacción Social

Descripción: Un usuario debe poder comunicar en tiempo real con otros usuarios con el cual se ha compartido un código y quienes lo tienen abierto al mismo tiempo.

Razón: Para promover cooperación entre usuarios y interacción entre estudiantes y profesores que son importantes para el proceso de enseñanza y aprendizaje.

Dependencias: Sistema de Compartir.

2. **Professores**

a) **Organizar Estudiantes y Aulas**

Título: Organizar Estudiantes y Aulas

Descripción: Un profesor debe poder organizar sus estudiantes y aulas para poder revisar los mismos y llevarlos del mejor forma.

Razón: Para reducir la cantidad de tiempo que necesita dedicar a la parte administrativo de sus materias para maximizar su tiempo para enseñanza de estudiantes.

Dependencias: Ninguno.

b) **Crear Pruebas, Exámenes, Proyectos, Talleres y Deberes para Estudiantes**

Título: Crear Pruebas, Exámenes, Proyectos, Talleres y Deberes para Estudiantes

Descripción: Un profesor debe poder crear dentro de la aplicación pruebas, exámenes, proyectos, talleres y deberes de los cuales se les puede asignar a sus aulas y estudiantes.

Razón: Para reducir la cantidad de tiempo que necesita dedicar a la parte administrativo de sus materias para maximizar su tiempo para enseñanza de estudiantes.

Dependencias: Organizar Estudiantes y Aulas.

c) **Autocalificación de Código**

Título: Autocalificación de Código

Descripción: Un profesor debe poder escribir pruebas unitarias dentro de la plataforma, los cuales califica de forma automática los pruebas/exámenes/proyectos/talleres y deberes que les asigna a los estudiantes.

Razón: Para reducir la cantidad de tiempo que necesita dedicar a calificar sus estudiantes dentro de sus materias para maximizar su tiempo para enseñanza de los mismos.

Dependencias: Crear Pruebas, Exámenes, Proyectos, Talleres y Deberes para Estudiantes.

d) **Sincronización de Notas**

Título: Sincronización de Notas

Descripción: El sistema debe ser capaz de sincronizar las notas que tiene con sistemas de terceros mediante el protocolo LTI.

Razón: Para reducir la cantidad de tiempo que necesita dedicar a calificar sus estudiantes y a la parte administrativa dentro de sus materias para maximizar su tiempo para enseñanza de los mismos.

Dependencias: Autocalificación de Código.

3. **Administradores**

a) **Integración con Sistemas Existentes**

1) **LMS Institucionales**

Título: Integración con LMS Institucionales Existentes

Descripción: La integración de la plataforma con LMS Institucionales Existentes debe dar las funcionalidades requeridas de autenticación y sincronización de notas de tal forma que su despliegue y mantenimiento sea sencilla y fácil.

Razón: Para reducir los recursos humanos necesarios para asegurar un buen despliegue y mantenimiento de la aplicación.

Dependencias: Autenticación mediante LTI, Sincronización de Notas.

2) **Servidores de Control de Versiones Institucionales**

Título: Integración con Servidores de Control de Versiones Institucionales Existentes

Descripción: La integración de la plataforma con Servidores de Control de Versiones Institucionales Existentes debe dar las funcionalidades requeridas de autenticación y sincronización de notas de tal forma que su despliegue y mantenimiento sea sencilla y fácil.

Razón: Para reducir los recursos humanos necesarios para asegurar un buen despliegue y mantenimiento de la aplicación.

Dependencias: Persistencia de Código.

b) **Recolección de Datos**

Título: Recolección de Datos

Descripción: La plataforma debe recolectar datos acerca de su uso con un fin de ayudar en la toma de decisiones institucionales.

Razón: Para ser un apoyo en la decisiones estratégicos.

Dependencias: Sincronización de Notas.

B.3.3. Requerimientos No Funcionales

1. **Numero de Estudiantes Suportados en Paralelo**

   Título: Número de Estudiantes Suportados en Paralelo

   Descripción: La plataforma debe soportar un número alto de estudiantes usando el mismo en paralelo (a la escala de 90 estudiantes).

   Razón: Para permitir sostener un cierto número de paralelos de una materia a la vez y de esta forma tomar un examen a todos en paralelo.

2. **Tiempo de Respuesta**

   Título: Tiempo de Respuesta

   Descripción: La plataforma debe ser capaz de responder dentro de un tiempo mínimo de un segundo a cualquier consulta de sus usuarios.

   Razón: Para ser útil a sus usuarios finales.

B.3.4. Limitaciones de Diseño

1. **Uso de Disco**

   Se considera que cada repositorio de código que se cree debe tener un tamaño máximo de 1 GB y el entorno virtual asociado con cualquier proyecto no debe exceder 10 GB.

2. **Uso de Memoria**

   El sistema de escribir código en línea no debe necesitar más de una GB de memoria, pero se considera que el subsistema de ejecución de código en línea debe disponer de mínimo 512 MB por cada usuario que espera suportar en paralelo.

3. **Uso de Processador**

   El sistema de escribir código en línea no debe necesitar más de cuatro núcleos físicos de procesador, pero se considera que el subsistema de ejecución de código en línea debe disponer de mínimo un núcleo equivalente a 1 GHz por cada usuario que espera suportar en paralelo.

B.3.5. Atributos del Sistema de Software

1. **Disponibilidad**

   El sistema debe ser disponible para peticiones en todo momento de clases cuando puede ser necesitado.

2. **Seguridad**

   El sistema debe ofrecer niveles óptimos de seguridad para prevenir modificación de notas, daño a equipos físicos y modificación de entornos virtuales de otros usuarios.

3. **Rendimiento**

   No se debe notar la diferencia en tiempos de respuesta y rendimiento entre una carga muy baja y muy alta.

4. **Concurrencia**

   Debe ser capaz de sostener 90 usuarios en paralelo.

5. **Mantenibilidad**

   Debe ser modular para permitir su mantenimiento al largo plazo y el intercambio de nuevos componentes.

6. **Estabilidad**

   A lo largo del tiempo debe presentar un mínimo de fallos que afectan a funcionalidad para el usuario final y de esta manera ofrecer una alta disponibilidad.

7. **Escalabilidad**

   Debe tener capacidad de escalar a números de usuarios mayores con la agregación de hardware adicional.

B.4. Plan de Prioridades y Despliegue

B.4.1. Selección de Método de Prioridades

Se da prioridad a cada requerimiento en base a cuales de los mismos son críticos, es decir funcionalidades básicas del mismo sistema y cuales solo son funcionalidades de valor agregado. En base a estas mismas, se considera también las dependencias entre requerimientos para empezar el desarrollo en las funcionalidades de que dependen otras funcionalidades. Eso se realiza junto con los stakeholders del proyecto para determinar cuales funcionalidades son críticos y cuales se los puede hacer si el tiempo y los recursos se los permiten.

B.4.2. Plan de Entregas de Software

Se plantea desarrollar el sistema en cuatro fases de los cuales cada uno se termina con una entrega de software. En la primera fase se autentica por LTI, en la segunda se realiza un editor de código en línea, en la tercera se realiza la persistencia periódica en el servidor de control de versiones y en la cuarta se realiza el subsistema de virtualización y ejecución de código el mismo que permite a usuarios probar su codigo en linea y califica codigo escrito por estudiantes. Dentro de cada fase se plantea desarrollar las otras características de valor agregado mencionadas previamente siempre y cuando se da el tiempo y los recursos para su cumplimiento. Al final de las cuatro fases y una fase final de pruebas, se procede a trabajar en el despliegue del mismo sistema con los autoridades responsable para el mismo.

Bibliografía

Almurayh, Abdullah y Sudhanshu Semwal (2014). «Xen web-based terminal for learning virtualization and cloud computing management». En: *Proc. World Congress on Engineering and Computer Science*. Vol. 1, págs. 329-333.

Amin, Nada (2017). *io.livecode.ch*. URL: http://io.livecode.ch/ (visitado 30-04-2017).

Beal, Vangie. *CSS - Cascading Style Sheets*. URL: http://www.webopedia.com/TERM/C/CSS.html (visitado 19-03-2017).

— *HTML - HyperText Markup Language*. URL: http://www.webopedia.com/TERM/H/HTML.html (visitado 19-03-2017).

— *HTTP - HyperText Transfer Protocol*. URL: http://www.webopedia.com/TERM/H/HTTP.html (visitado 19-03-2017).

— *JavaScript*. URL: http://www.webopedia.com/TERM/J/JavaScript.html (visitado 19-03-2017).

Berners-Lee, Tim, Roy Fielding y Larry Masinter (2005). *Uniform Resource Identifier (URI): Generic Syntax*. URL: https://tools.ietf.org/html/rfc3986 (visitado 19-03-2017).

Bravo, Marcelo, Nicholas Earley y Pricilla Vargas (2017). *GitEduERP*. URL: https://git.taw.utpl.edu.ec/ArqAppGrpBravoEarleyVargas/GitEduERP (visitado 30-03-2017).

Chacon, Scott y Ben Straub (2014a). *1.1 Getting Started - About Version Control*. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (visitado 24-02-2017).

— (2014b). *1.3 Getting Started - Git Basics*. URL: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics (visitado 24-02-2017).

— (2014c). *7.4 Git Tools - Signing Your Work*. URL: https://git-scm.com/book/tr/v2/Git-Tools-Signing-Your-Work (visitado 24-02-2017).

Cisco. *Tunneling*. URL: http://www.cisco.com/c/en/us/products/ios-nx-os-software/tunneling/index.html (visitado 04-03-2017).

Cloud9 IDE, Inc. *Your development environment, in the cloud*. URL: https://c9.io/ (visitado 19-03-2017).

ComputerHope (2017). *HTTP*. URL: https://www.computerhope.com/jargon/h/http.htm (visitado 19-03-2017).

Cruise, Brit (2012a). *RSA encryption: Step 1*. URL: https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/intro-to-rsa-encryption (visitado 19-03-2017).

— (2012b). *What is cryptography?* URL: https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/intro-to-cryptography (visitado 19-03-2017).

DigiCert Inc. *What is an SSL certificate?* URL: https://www.digicert.com/ssl/ (visitado 19-03-2017).

Docker Inc. (2017a). *Docker*. URL: https://www.docker.com/ (visitado 30-04-2017).

— (2017b). *What is a Container*. URL: https://www.docker.com/what-container (visitado 30-04-2017).

Docker Inc. (2017c). *What is Docker*. URL: https://www.docker.com/what-docker (visitado 30-04-2017).

Dougiamas, Martin y Free Software Foundation (2016). *GNU GENERAL PUBLIC LICENSE*. URL: https://git.moodle.org/gw?p=moodle.git;a=blob;f=COPYING.txt;h=94a9ed024d3859793618152ea559a168bbcbb5e2;hb=HEAD (visitado 17-02-2017).

edX. *About Open edX*. URL: https://open.edx.org/about-open-edx (visitado 25-04-2017).

Ellingwood, Justin (2015). *The Docker Ecosystem: An Overview of Containerization*. URL: https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-overview-of-containerization (visitado 30-04-2017).

Flomenberg, Jake. *The Next Wave in Software: Open Adoption Software (OAS)*. URL: https://www.accel.com/interests/TheNextWaveInSoftwareOAS (visitado 21-02-2017).

Free Software Foundation (2007). *GNU General Public License*. URL: https://www.gnu.org/licenses/gpl.html (visitado 24-02-2017).

— (2016). *Licenses*. URL: https://www.gnu.org/licenses/licenses.en.html (visitado 24-02-2017).

Freyd, Régis y col. (2017). *GitLab Products*. URL: https://about.gitlab.com/gitlab-com/ (visitado 04-03-2017).

Geerling, Jeff (2014). *A brief history of SSH and remote access*. URL: https://www.jeffgeerling.com/blog/brief-history-ssh-and-remote-access (visitado 04-03-2017).

GitLab Inc. (2017). *Host GitLab on GitHost.io*. URL: https://githost.io/ (visitado 04-03-2017).

GlobalSign. *What is an SSL Certificate?* URL: https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/ (visitado 15-03-2017).

— (2016). *SSL vs. TLS - What's the Difference?* URL: https://www.globalsign.com/en/blog/ssl-vs-tls-difference/ (visitado 19-03-2017).

Google. *Explore the Storage Features of Drive*. URL: https://www.google.com/drive/using-drive/ (visitado 23-03-2017).

Hughes, Phil y col. (2017). *The platform for modern developers*. URL: https://about.gitlab.com/ (visitado 23-03-2017).

IMSGlobal. *Contributing Members, Affiliates, and Alliance Participants*. URL: https://www.imsglobal.org/membersandaffiliates.html (visitado 16-02-2017).

— *Learning Tools Interoperability® Background*. URL: https://www.imsglobal.org/activity/learning-tools-interoperability (visitado 16-02-2017).

Kangas, Erik (2016). *SSL versus TLS – What's the difference?* URL: https://luxsci.com/blog/ssl-versus-tls-whats-the-difference.html (visitado 19-03-2017).

Kaspersky Labs USA. *What is a Tunneling Protocol?* URL: https://usa.kaspersky.com/internet-security-center/definitions/tunneling-protocol (visitado 15-03-2017).

Lees-Miller, John (2015). *New: Collaborate Online and Offline with Overleaf and Git (beta)*. URL: https://www.overleaf.com/blog/195-new-collaborate-online-and-offline-with-overleaf-and-git-beta (visitado 23-03-2017).

López, Jorge (2017). Conversaciones en el curso de 2017.

Mindflash. *Mindflash Learning Management*. URL: https://www.mindflash.com/about/ (visitado 17-02-2017).

— *What is an LMS?* URL: https://www.mindflash.com/learning-management-systems/what-is-an-lms/ (visitado 17-02-2017).
Moodle Community (2016). *Features*. URL: https://docs.moodle.org/32/en/Features (visitado 17-02-2017).
Nierop, Ernst van y col. (2017). *Host your projects on GitLab.com*. URL: https://about.gitlab.com/gitlab-com/ (visitado 04-03-2017).
OpenBSD. *OpenSSH Features*. URL: http://www.openbsd.org/openssh/features.html (visitado 15-03-2017).
— (2016). *SSH(1)*. URL: http://man.openbsd.org/OpenBSD-current/man1/ssh.1 (visitado 15-03-2017).
OpenCampus. *For Universities*. URL: https://www.opencampus.net/home/univerities (visitado 21-02-2017).
— *OpenCampus*. URL: https://www.opencampus.net/home/ (visitado 21-02-2017).
— *Technology*. URL: https://www.opencampus.net/home/technology (visitado 21-02-2017).
OpenSSH. *OpenSSH*. URL: https://www.openssh.com/ (visitado 15-03-2017).
Oracle. *VirtualBox*. URL: https://www.virtualbox.org/ (visitado 30-04-2017).
Overleaf y Writelatex Limited. *Collaborative Writing and Publishing*. URL: https://www.overleaf.com/ (visitado 23-03-2017).
Pipinellis, Achilleas y col. (2017). *Next-generation developer collaboration software*. URL: https://about.gitlab.com/features/ (visitado 23-03-2017).
Project, QEMU. *QEMU*. URL: http://www.qemu.org/ (visitado 25-04-2017).
Raymond, Eric (1999). «The cathedral and the bazaar». En: *Philosophy & Technology* 12.3, pág. 23.
Repl.it y Neoreason, Inc. (2017). *repl.it is a cloud coding environment for...* URL: https://repl.it/ (visitado 19-03-2010).
Rose, Scott (2015). *Distributed or Centralized Development?* URL: http://blogs.collab.net/uncategorized/distributed-or-centralized-development (visitado 24-02-2017).
Salinas, Diana Maritza Ortega y Juan Carlos Sánchez Castillo (2007). *Sistema para la estructuración lógica de encuestas online para el centro de asesoría empresarial y social (cades);y call center de la UTPL*. URL: http://dspace.utpl.edu.ec/handle/123456789/14471 (visitado 30-03-2017).
Schluting, Charlie (2006). *Networking 101: Understanding Tunneling*. URL: http://www.enterprisenetworkingplanet.com/netsp/article.php/3624566/Networking-101-Understanding-Tunneling.htm (visitado 15-03-2017).
Sijbrandij, Sid y col. (2017). *About Us*. URL: https://about.gitlab.com/about/ (visitado 04-03-2017).
Smith, Brett (2014). *A Quick Guide to GPLv3*. URL: https://www.gnu.org/licenses/quick-guide-gplv3.html (visitado 24-02-2017).
SSL.com Corp (2016). *FAQ: What is SSL?* URL: http://info.ssl.com/article.aspx?id=10241 (visitado 19-03-2017).
Stallman, Richard Mathew (2014). *About the GNU Operating System*. URL: https://www.gnu.org/gnu/about-gnu.html (visitado 21-02-2017).
— (2016a). *FLOSS and FOSS*. URL: https://www.gnu.org/philosophy/floss-and-foss.en.html (visitado 21-02-2017).
— (2016b). *What is free software?* URL: https://www.gnu.org/philosophy/free-sw.html (visitado 21-02-2017).
— (2016c). *Why Open Source misses the point of Free Software*. URL: https://www.gnu.org/philosophy/open-source-misses-the-point.html (visitado 21-02-2017).

Stallman, Richard Mathew (2017a). *Free Software Is Even More Important Now*. URL: https://www.gnu.org/philosophy/free-software-even-more-important.html (visitado 21-02-2017).

— (2017b). *Linux and the GNU System*. URL: https://www.gnu.org/gnu/linux-and-gnu.html (visitado 23-02-2017).

Stevens, Marc y col. (2017). *The first collision for full SHA-1*. URL: https://shattered.io/static/shattered.pdf (visitado 24-02-2017).

Teimouri, Davoud (2016). *Operating-system-level virtualization*. URL: https://www.teimouri.net/operating-system-level-virtualization/ (visitado 30-04-2017).

Tholeti, Bhanu P (2011). *Learn about hypervisors, system virtualization, and how it works in a cloud environment*. URL: https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/ (visitado 25-04-2017).

Torres, Mauricio Javier Castillo (2008). *Metodología para implementación de cursos virtuales con herramientas Web 2.0*. URL: http://dspace.utpl.edu.ec/handle/123456789/17020 (visitado 30-03-2017).

University of East Anglia. *What is a MOOC? Massive Open Online Courses Explained*. URL: https://www.uea.ac.uk/study/short-courses/online-learning/what-is-a-mooc (visitado 25-04-2017).

University System of Georgia. *About Learning Repository*. URL: https://go.view.usg.edu/shared/Documentation/9.4.1/Instructor/9.4.1%20Instructor%20Course%20Designer%20Help/learningrepository/about_learning_repository.htm (visitado 17-02-2017).

UTPL (2017a). *DATOS ESTADÍSTICOS*. URL: http://www.utpl.edu.ec/utpl/informacion-general/datos-estadisticos (visitado 05-04-2017).

— (2017b). *DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION Y ELECTRONICA*. URL: http://www.utpl.edu.ec/directorio/index.php?ban=2&id_citte=1&id_dep=118&nom_dep=DEPARTAMENTO%20DE%20CIENCIAS%20DE%20LA%20COMPUTACION%20Y%20ELECTRONICA (visitado 05-04-2017).

— (2017c). *OpenCampus Acerca De*. URL: http://opencampus.utpl.edu.ec/about (visitado 21-02-2017).

— (2017d). *UTPL Tecnologías Avanzadas de la Web GitLab Community Edition*. URL: https://git.taw.utpl.edu.ec/ (visitado 04-03-2017).

Vanderbilt University. *Course Management Systems*. URL: https://cft.vanderbilt.edu/guides-sub-pages/course-management-systems/ (visitado 25-04-2017).

VMWare, Inc. *What is Virtualization?* URL: http://www.vmware.com/solutions/virtualization.html (visitado 25-04-2017).

— (2014). *What is Virtualization?* URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/techpaper/vmw-white-paper-secrty-vsphr-hyprvsr-uslet-101.pdf (visitado 25-04-2017).

Webopedia, ITBusinessEdge y QuinStreet Inc. *REST - Representational State Transfer*. URL: http://www.webopedia.com/TERM/R/REST.html (visitado 19-03-2017).

Webopedia Staff. *URI - Uniform Resource Identifier*. URL: http://www.webopedia.com/TERM/U/URI.html (visitado 19-03-2017).

Wilson, George, Michael Day y Beth Taylor (2011). *KVM: Hypervisor Security You Can Depend On*. URL: ftp://public.dhe.ibm.com/linux/pdfs/LXW03004-USEN-00.pdf (visitado 25-04-2017).

Xen Community (2016). *Xen Project Software Overview*. URL: `https://wiki.xen.org/wiki/Xen_Project_Software_Overview` (visitado 30-04-2017).