



PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito final para obtener el Título
de INGENIERO DE SISTEMAS

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA WEB PARA EL SIMULADOR DE EVENTOS DISCRETOS GALATEA.

Por

Br. Erik Velásquez

Tutor: Dr. Jacinto Dávila

Tutor: Prof. Kay Tucci

Noviembre 2016

©2016 Universidad de Los Andes Mérida, Venezuela

Diseño e implementación de un Sistema Web para el Simulador de Eventos Discretos GALATEA.

Br. Erik Velásquez

Proyecto de Grado — Sistemas Computacionales, 68 páginas

Resumen: Se buscó implementar y desplegar como servicio, una aplicación utilizada para modelar y simular eventos discretos en sistemas distribuidos, utilizando como medio los servicios web. El objetivo principal de este proyecto consistió en buscar, mediante el uso de un sistema web, una solución al problema existente entre los usuarios y usuarias del Centro de Simulación y Modelado (CESIMO) de la Universidad de los Andes, que suelen tener muchas dificultades para ejecutar los modelos de simulación en el simulador GALATEA, especialmente al momento de configurar y activar todas sus funcionalidades. Mediante el patrón de diseño de software modelo vista controlador (MVC) y usando el framework de desarrollo web Django; se desarrolló el sistema web, que usando como motor principal a GALATEA, permite crear, controlar y simular proyectos de simulación. Al igual se creó un sistema de gestión y administración de archivos, la cual permite compartir y editar archivos entre los distintos usuarios y usuarias que hacen uso del sistema web de simulación.

Palabras clave: Servicios Web, Sistemas Distribuidos, Web Semántica, Arquitectura Orientada a Servicios, Simulación.

Nadie.

Nadie.

Nadie.

Índice

Índice de Tablas	vi
Agradecimientos	viii
1 Introducción	1
1.1 Antecedentes	2
1.2 Planteamiento del problema	3
1.3 Justificación	4
1.4 Alcance	4
1.5 Objetivos	4
1.5.1 Objetivo general	4
1.5.2 Objetivos específicos	5
1.6 Metodología	5
1.7 Estructura del documento	6
2 Marco teórico-funcional	7
2.1 Simulación Distribuida	7
2.2 Simulación de Eventos Discretos	8
2.3 GALATEA	8
2.3.1 La arquitectura de la plataforma Galatea	9
2.4 Servicio Web	10
2.5 Herramientas usadas para el desarrollo del sistema	11
2.5.1 Modelo Vista Controlador (MVC)	11
2.5.2 Django	15
2.5.3 Bases de Datos	16
2.5.4 JSON	18
2.5.5 AJAX	18

3	Desarrollo del Sistema WEB	21
3.1	Diseño de la aplicación WEB	22
3.1.1	Arquitectura	22
3.1.2	Capa 1: Presentación	23
3.1.3	Capa 2: Servidor Web	23
3.1.4	Capa 3: Integración de Procesos	24
3.1.5	Capa 4: Datos	24
3.1.6	Diseño de la Base de Datos	25
3.1.7	Diseño de Pantallas	30
3.1.8	Diseño de URLs	33
3.2	Administrador de Archivos	34
3.2.1	Control de Archivos	36
3.2.2	Control de Carpetas	36
3.2.3	Control de Permisos	38
4	Integración de GALATEA	40
4.1	Subprocesos	40
4.2	Sockets	42
4.2.1	Programa Cliente	44
4.2.2	Programa Servidor	44
4.2.3	Protocolo de Comunicación	49
5	Pruebas	51
5.1	Metodología de las pruebas	52
5.2	Condiciones de las pruebas	52
5.3	Resultados obtenidos	53
5.4	Análisis de los resultados	53
6	Conclusiones	54
6.1	Recomendaciones	55
	Bibliografía	57
A	Historias de usuario de la aplicación web	59
B	Casos de uso de la aplicación web	61
B.1	Usuario Administrador	63
C	Casos de uso de la integración con GALATEA	65
D	Casos de uso de administración de archivos	67

Índice de Tablas

3.1	Tabla de enrutamiento del servidor web.	23
3.2	Entidades relacionadas al manejo de usuarios.	29
3.3	Entidades relacionadas al manejo de archivos y carpetas.	30
3.4	Entidades relacionadas al manejo de archivos y carpetas compartidos.	30
3.5	Permisología de Archivos y Carpetas.	36
5.1	Especificaciones del servidor web.	53
A.1	Historia de Usuario - Control de Usuarios en el sistema.	59
A.2	Historia de Usuario - Control de Archivos y carpetas en el sistema.	60
A.3	Historia de Usuario - Integración con GALATEA.	60
B.1	Caso de uso - Login Usuario.	61
B.2	Caso de uso - Logout de Usuario.	61
B.3	Caso de uso - Control de Contraseña.	61
B.4	Caso de uso - Manejo de Perfil.	62
B.5	Caso de uso - Manejo de Archivos.	62
B.6	Caso de uso - Manejo de Carpetas.	62
B.7	Caso de uso - Control de Simulación.	62
B.8	Caso de uso - Control de Amigos.	63
B.9	Caso de uso - Control de Archivos Compartidos.	63
B.10	Caso de uso - Control de Usuarios.	63
B.11	Caso de uso - Observación de Logs.	63
B.12	Caso de uso - Control de Espacios en Disco.	64
C.1	Caso de uso - Traducir Archivos.	65
C.2	Caso de uso - Compilar Archivos.	65
C.3	Caso de uso - Ejecutar Archivos.	66
C.4	Caso de uso - Mostrar Simulación.	66
D.1	Caso de uso - NOMBRE.	67

D.2	Caso de uso - Control de Archivos.	67
D.3	Caso de uso - Control de Carpetas.	68

Agradecimientos

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Capítulo 1

Introducción

Este documento es una descripción del cómo se ha desarrollado el servicio Web que prestará una aplicación basada en simulación de eventos discretos, basada en GALATEA (Uzcátegui et al., 2011) , el cual es un software para simulación de sistemas multi-agentes producto de dos líneas de investigación: lenguajes de simulación basados en la teoría de simulación de Zeigler y agentes basados en lógica. La simulación es un herramienta que poseen los investigadores, ingenieros y profesionales para modelar sus experimentos y probarlos en un ambiente controlado. En el campo de la ingeniería la simulación se ha convertido en una herramienta fundamental y muy útil para cualquier tipo de trabajo. El desarrollo de la computación ha permitido que el campo de la simulación crezca de manera muy apresurada y ha permitido que se cuente con herramientas muy avanzadas, de fácil acceso y manipulación para simular.

Con el desarrollo del internet y el acceso que la mayoría de personas tenemos a él de manera sencilla, debemos pensar que desarrollar un software de simulación ejecutado en la web seria una herramienta muy poderosa e importante, que ayudaría en el trabajo a muchísimas personas de distintos campos y áreas. El propósito de éste trabajo es que el sistema de simulación web sirva para proveer un servicio de simulación de eventos discretos al usuario desde cualquier punto de mundo con una conexión a internet. Así también, se han previsto características de seguridad mediante las cuales, ante una petición al computador (request), ésta sea procesada sólo si ha sido previamente validada la identidad y autorización de los solicitantes.

Actualmente existen muchas herramientas que permiten diseñar y desarrollar aplicaciones web, como Foundation (2015). Estas herramientas están muy bien documentadas y son muy flexibles, lo cual permite la integración con distintos softwares. Esta flexibilidad nos permite la integración entre un simulador de eventos discretos como GALATEA y una aplicación web.

1.1 Antecedentes

Como trabajos similares a esta propuesta se pueden mencionar los descritos en (Uzcátegui et al., 2011) el donde se relata la fase de desarrollo, cómo surge y hacia donde se proyecta GALATEA. Implementada como una plataforma libre para simulación de sistemas multi-agente, que incorpora estrategias de simulación bien conocidas con la que cualquier modelista o simulista puede ensayar dichas estrategias en problemas de simulación de sistemas complejos.

Otro precedente (Rengifo, 2011), trata de una tesis de pre-grado que expone el desarrollo de un servicio web para la Modeloteca de un Sistema Nacional de Simulación. La misma consistió en buscar, mediante el uso de una aplicación web, la solución al problema existente en el Centro de Simulación y Modelado (CESIMO) de la Universidad de Los Andes, en el cual había dificultades para mantener un registro referente a los proyectos que allí se desarrollaban, lo que como resultado provocaba que a menudo se perdiera información referente a los mismos, ó que incluso se llevara a cabo proyectos de forma innecesaria, ya que trabajos parecidos habrían sido realizado antes. Resultados que, debido a la falta de un repositorio institucional compartido, resultan imposibles de reusar o integrar. El objetivo de este proyecto de pre-grado fue adaptar un CMS, pero dicha adaptación no logró integrar a GALATEA como parte de su funcionalidad.

En su tesis de pre-grado (Marcano, 2015), se buscó implementar y desplegar como servicio, una aplicación utilizada para modelar y simular eventos discretos en sistemas distribuidos, utilizando como medio los servicios web. Mediante el uso del paradigma de arquitectura orientada a servicios (SOA: Service Oriented Architecture. Pero ese trabajo no alcanzó a resolver la complejidad de un servicio basado en SOA y tampoco de lograr la integración con GALATEA.

A partir de estos antecedentes lo que se pretende es, tomando como base GALATEA, adaptarlo como un servicio web en el cual el usuario pueda interactuar con el simulador, sin tener las limitaciones tales como las que pueden suelen surgir al usar sistemas operativos o navegadores diferentes, incluso ante arquitecturas distintas. A su vez, de (Rengifo, 2011), se buscó obtener la experiencia ganada al desarrollar un Servicio Web en sí. Al igual se toma en cuenta a (Marcano, 2015), buscando implementar y desplegar como servicio, una aplicación utilizada para modelar y simular eventos discretos en sistemas distribuidos, utilizando como medio los servicios web. A pesar de que ambas experiencias desarrollaron soluciones, aún no es posible contar con la experiencia de uso remoto de GALATEA como un servicio Web configurado por expertos administradores, pero con todas las facilidades y características del simulador al alcance de cualquier usuario registrado.

Nuestro enfoque ha sido desarrollar una app web con un diseño MVC a implementar sobre/con el framework Django. El modelo vista controlador (MVC) (Krasner y Pope, 1988) es un patrón de diseño de software, que separa los datos y la lógica de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Este patrón de diseño es muy popular en el marco de aplicaciones web ya que su abstracción permite escribir software altamente desacoplado y fácil de mantener y escalar.

Django es un framework de desarrollo web de software libre, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–controlador. La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, Don't Repeat Yourself) (Foundation, 2015). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. Django permite construir aplicaciones web rápidamente, gracias a su filosofía de baterías incluidas, es decir, que incluye una inmensa gama de características comunes a la mayoría de las aplicaciones web como las validaciones, autenticación de usuarios, manejo de sesiones entre muchos otros (Foundation, 2015).

Podemos mencionar también a (Pérez, 2016), que en su proyecto de grado y utilizando MVC y Django como herramientas fundamentales desarrolló un sistema de monitoreo distribuido de enlaces de redes a través de un servicio web centralizado. Este servicio además hace énfasis en la visualización de los datos recolectados a partir de pruebas periódicas.

1.2 Planteamiento del problema

La preocupación por los sistemas distribuidos y de cómo diferentes máquinas podían comunicarse entre sí surgió en la década de los 90. Hasta ese momento, era suficiente con que las aplicaciones de un mismo ordenador pudieran establecer una comunicación.

En la medida que se ha avanzado en términos tecnológicos, se percibe un crecimiento proporcional de la información que se genera en todos los ámbitos, sea científico, humanístico, económico etc. Estos fenómenos no suceden como acontecimientos aislados, sino que son posibles gracias al conocimiento adquirido y al esfuerzo de muchos, de ponerlo a disposición de quien desee acceder a ellos. Teniendo en cuenta el acceso libre y continuo de la información, surge el planteamiento del problema, el cual tiene dos vertientes, la primera, consiste en que no se dispone de una plataforma web acondicionada para que, de manera fácil y rápida se pueda hacer uso del simulador GALATEA. Los usuarios y usuarias del Centro de Simulación y Modelado (CESIMO) de la Universidad de los Andes, suelen tener muchas dificultades para ejecutar los modelos de simulación en el simulador GALATEA, especialmente al momento de configurar y activar todas sus funcionalidades.

Si bien se posee un servidor académico en CESIMO, es necesario desarrollar mecanismos que posibiliten el uso de las herramientas que éste ofrece, por ello la necesidad de servicios web que permitan que las funcionalidades que posee el simulador estén disponibles tanto a nivel local como a nivel externo. La plataforma desarrollada en éste trabajo permitirá y estimulará el trabajo colaborativo y cooperativo entre los usuarios de simulación, lo que se puede llamar "Simulación 2.0".

En cuanto al otro aspecto, consiste en verificar que la integridad de la información estén garantizados, en especial cuando el usuario desee enviar algún dato, o realizar alguna consulta, por lo cual el sistema debe validar a dicho usuario previamente.

1.3 Justificación

Dado que las organizaciones protegen sus redes mediante firewalls -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados. Es importante señalar que los servicios web se pueden utilizar sobre cualquier protocolo, sin embargo, TCP es el más común.

La principal motivación para realizar este trabajo, es la de desarrollar un sistema web que sirva como base para la simulación de eventos discretos, esperando que un sistema web amplíe la base de usuarios y usuarias del simulador liberándolos del trabajo de configuración y permitiendo acceder a los modelos de simulación, al igual ponerlos en contacto con diferentes expertos que trabajan en el CESIMO y así validar sus modelos de simulación y compartir conocimientos. La justificación, además, es hacer que el simulador sea amigable a la web, para así, aprovechar todas las ventajas que nos proporciona.

1.4 Alcance

El proyecto culmina con el diseño e implementación de un sistema web para el uso de GALATEA. Se desarrolló un prototipo de esa arquitectura, implementando el sistema web para la ejecución de los modelos de simulación. El prototipo cubre todo lo referente a la edición y compilación de los modelos, además de ejecución de simulaciones. Adicionalmente, la arquitectura del mismo permite adaptarse al contexto del usuario, y reaccionar en base a la interacción del usuario con el sistema.

1.5 Objetivos

1.5.1 Objetivo general

Diseñar e implementar un sistema web para los usuarios y usuarias, modelistas y simulistas del simulador de eventos discretos GALATEA, que les permita realizar todas las tareas habituales de modelado, codificación y análisis en sus computadores y en la forma que prefieran, pero permitiéndoles realizar las tareas automáticas de compilación, gestión de archivos, simulación y gestión de salidas, en el espacio virtual y con los recursos compartidos de un servidor Web.

1.5.2 Objetivos específicos

1. Desarrollar un sistema web que permita el control de usuarios junto con los roles a ser utilizados en el sistema.
2. Diseñar e implementar una arquitectura de software que permita la comunicación entre el software de simulación y el sistema web.
3. Instalar y configurar en un servidor la arquitectura de software para el sistema de simulación.
4. Incorporar el simulador GALATEA como servicio para el sistema web.
5. Diseñar y desarrollar un cliente GUI/controlador para un modelo que se pueda gestionar archivos y simular con GALATEA a través del sistema web desarrollado.
6. Sistematizar la experiencia de uso del sistema web para simulación.
7. Analizar el sistema web desarrollado y establecer las conclusiones.

1.6 Metodología

Para el desarrollo del proyecto, se utilizó una metodología en espiral. El modelo en espiral es un modelo del ciclo de vida del software donde el esfuerzo del desarrollo es iterativo (Boehm, 1988). Cada ciclo de la espiral representa una fase del desarrollo de software, cada uno de los ciclos consiste de los siguientes pasos:

1. Determinar o fijar los objetivos. En este paso se definen los objetivos específicos para posteriormente identificar las limitaciones del proceso y del sistema de software, además se diseña una planificación detallada de gestión y se identifican los riesgos.
2. Análisis del riesgo. En este paso se efectúa un análisis detallado para cada uno de los riesgos identificados del proyecto, se definen los pasos a seguir para reducir los riesgos y luego del análisis de estos riesgos se planean estrategias alternativas.
3. Desarrollar, verificar y validar. En este tercer paso, después del análisis de riesgo, se elige un paradigma para el desarrollo del sistema de software.
4. Planificar. En este último paso es donde el proyecto se revisa y se toma la decisión si se debe continuar con un ciclo posterior al de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

Se realizaron cuatro ciclos, el primero correspondió a la realización de un sistema web simple que permitiera el control y administración de usuarios (Registrar, logearse y editar información).

El segundo ciclo se enfoco en el desarrollo de un administrador de archivos y carpetas dentro de nuestro sistema web, permitirle a los usuarios crear, modificar y eliminar archivos y carpetas.

El tercer ciclo estuvo definido por la integración entre el motor de simulación GALATEA y nuestro sistema web. Diseñando y desarrollando los enfoques necesarios para su correcto funcionamiento dentro del sistema, al igual su fácil manejo por parte del usuario.

El cuarto ciclo consistió en hacer análisis del rendimiento del sistema y hacer las optimizaciones necesarias para ofrecer una calidad de servicio apropiada.

1.7 Estructura del documento

Capítulo I. Introducción.

En esa sección se describe brevemente las características del problema abordado en el proyecto, analizando los antecedentes, delimitando los objetivos y estableciendo la metodología con que se procedió al estudio y resolución del mismo.

Capítulo II. Marco teórico.

En esa sección se exponen algunos conceptos relacionados con el problema planteado en el proyecto. Se da una breve introducción sobre los términos y herramientas utilizados para desarrollar el mismo.

Capítulo III. Desarrollo del Sistema WEB.

Contiene el diseño general del sistema de simulación, así como la arquitectura de la aplicación web, sus sub-sistemas, componentes relevantes y casos de uso.

Capítulo IV. Integración de GALATEA.

Contiene la arquitectura, el diseño y desarrollo de los programas de integración con el motor de simulación GALATEA que será ejecutado como una aplicación en un servidor.

Capítulo V. Pruebas.

En esta sección se exponen las distintas pruebas realizadas a la aplicación web, definición, implementación y el resultado, así mismo hacemos un estudio de los resultados obtenidos.

Capítulo VI. Conclusiones y Recomendaciones.

Conclusiones y Recomendaciones, contiene las conclusiones obtenidas a partir del diseño e implementación del sistema, así como recomendaciones y posibles mejoras a incluir en trabajos futuros.

Finalmente se presentan los apéndices y las referencias bibliográficas.

Capítulo 2

Marco teórico-funcional

La simulación por computadora se ha convertido en una parte útil del modelado de muchos sistemas naturales en física, química y biología, y sistemas humanos como la economía y las ciencias sociales (sociología computacional) (Gilbert y Troitzsch, 2005), su comportamiento cambiará cada simulación según el conjunto de parámetros iniciales supuestos por el entorno. En éste capítulo se explican en detalle las herramientas, tecnologías y conceptos usados para la consecución del proyecto que se propone. Definimos la simulación de eventos discretos DEVS (Discrete Event Simulation) (Banks et al., 2010), así mismo nos introducimos en GALATEA, con el fin de proporcionar una base inicial para la codificación del nuevo sistema. Explicamos las ventajas de un servicio web y las herramientas empleadas para el desarrollo del mismo, como lo serian el Patrón de diseño Modelo Vista Controlador (MVC) (Krasner y Pope, 1988) y Django (Fundation, 2015).

2.1 Simulación Distribuida

Las simulaciones distribuidas pueden hacerse de forma que se acelere una simulación o que se acelere la ejecución de un conjunto de simulaciones. Los sistemas de simulación basados en agentes se pueden utilizar con éxito en ambos tipos de simulaciones distribuidas (Santana et al., 2004).

La Simulación Distribuida permite conducir modelos de simulación a través de múltiples hosts o computadoras separados por redes y se ocupa de cuestiones que surgen de la distribución de un programa de simulación de eventos discretos en varias computadoras (Banks et al., 2010). La simulación paralela de eventos discretos se refiere a la ejecución en plataformas multi-procesador que contienen múltiples unidades de procesamiento que interaccionan entre sí con frecuencia, por ejemplo, miles de veces por segundo.

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta (Microsystem,

2009). La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

2.2 Simulación de Eventos Discretos

La simulación de eventos discretos es la "imitación" de un proceso de operación o de un sistema del mundo real construido sobre la base del tiempo (Banks et al., 2010). Esta codifica el comportamiento de sistemas complejos como una secuencia de eventos ordenados y bien definidos. En simulación de eventos discretos (DEVS por sus siglas en inglés), el funcionamiento de un sistema se representa como una secuencia cronológica de los acontecimientos. Cada evento tiene lugar en un instante de tiempo y marca un cambio de estado en el sistema. Frente a su homóloga, la simulación de tiempo continuo, esta se caracteriza por un control en la variable del tiempo que permite avanzar a éste a intervalos variables, en función de la planificación de ocurrencia de tales eventos a un tiempo futuro. Estos sistemas se caracterizan por mantener un estado interno global del sistema, que puede no obstante estar física o lógicamente distribuido, y que cambia parcialmente debido a la ocurrencia de un evento. La ejecución de un evento puede desencadenar la generación de nuevos eventos futuros. Cada uno está marcado por su tiempo, por lo que el orden de generación puede no coincidir con el orden de ejecución.

2.3 GALATEA

Galatea es una plataforma para simulación de eventos discretos, DEVS, con una semántica basada a una red de nodos como metáfora del sistema a simular, la misma semántica matriz del sistema Glider, el proyecto materno local, que fuera formalizada como parte del nuevo proyecto y luego generalizada para luego re-acomodar la simulación continua, la simulación de sistemas multi-agentes y la simulación distribuida, simulación con autómatas celulares y simulación con modelos explícitos de espacios urbanos o arquitectónicos (Uzcátegui et al., 2011).

Podemos explicar de forma breve y concisa en que consiste la misma, según (Uzcátegui et al., 2011):

Galatea es una plataforma libre para simulación de sistemas multi-agente que incorpora estrategias de simulación bien conocidas con la que cualquier modelista o simulista puede ensayar esas estrategias en problemas de simulación de sistemas complejos. La historia de Galatea comienza mucho antes que se planteara formalmente el proyecto con ese nombre. En 1993, nuestro muy joven Centro de Simulación y Modelos, CeSiMo, propone un proyecto para explorar la re-implementación de la plataforma de simulación Glider sobre

una plataforma orientada a objetos dando origen a un prototipo experimental. El problema del cambio estructural, inspirado por investigaciones en economía, se había convertido entonces en uno de los objetivos de investigación fundamentales del CeSiMo y vendría a dictar también la pauta para Galatea. La noción de agente hizo su aparición en algunos reportes internos en los que se enfatizaba su importancia para modelar sistemas complejos como una economía nacional. En 1998 se planteó la posibilidad de integrar Glider con herramientas de inteligencia artificial para modelar agentes. En el 2000, un proyecto vendría a combinar aquel prototipo de 1993, con una teoría de agentes basada en lógica computacional que se planeaba integrar en una nueva teoría de simulación de sistemas multi-agentes. Allí nació Galatea. El logro fundamental para el proyecto, sin embargo, llegaría con las aplicaciones. En 2004, Galatea fue incorporada al banco de pruebas de un proyecto en biocomplejidad. Y en 2009-2011 fue usada como cama de pruebas de un sistema de agentes que aprenden (Dávila, 2011).

2.3.1 La arquitectura de la plataforma Galatea

La arquitectura de GALATEA está basada en objetos. Tanto los agentes como el simulador principal están desarrollados de acuerdo a dicho diseño (orientado a objetos OOD), para apoyar la distribución, la modularidad, la escalabilidad y la interactividad como lo exige la especificación HLA (High Level Architecture) (Fujimoto, 2015). El lenguaje base usado para su desarrollo es el Java (Uzcátegui et al., 2011). Usando como lenguaje base Java, se especificó un nuevo lenguaje de simulación al que denominan lenguaje Galatea. La sintaxis del lenguaje Galatea es, de hecho, una mezcla de la sintaxis Glider, las estructuras básicas de Java y las reglas de conductas de los agentes escritas en los lenguajes de programación lógica Actilog (Dávila, 2003) y Openlog (Dávila, 1999). La compleja semántica de Galatea establece que el código Java sea compilado y ejecutado por el motor de simulación de eventos discretos, mientras que las reglas de los agentes son interpretadas por un motor de inferencia implementado sobre una máquina Prolog (Uzcátegui et al., 2011). Es una plataforma mixta, que corre sobre máquinas virtuales Java y Prolog.

En la figura 2.2 podemos observar la ejecución del modelo "SimpleTeller", que se encuentra entre los ejemplos básicos de GALATEA. En este modelo define un taquilla de un banco, la cual posee una entrada "Gate (I)" y una salida "Exit (E)". Las personas que entran llegan a un cajero de banco a intervalos de tiempo tomadas al azar de una distribución exponencial con la media "InArrTime". Se sirven uno a la vez. Los tiempos de servicio se toman de una distribución gaussiana con media "MeSerTime" y desviación estándar "DeSerTime".

```

TITLE
    Simple Teller
NETWORK
    Gate (I){
        IT=EXP0(InArrTime);
        WRITE("ARRIVE:\t" + TIME);
        ACT(ShowLength,0);
        SENDTO(Teller);
    }
    Teller (R){
        STAY=GAUSS(MeSerTime,DeSerTime);
        RELEASE
        SENDTO(Exit);
    }
    Exit (E) {
        WRITE("EXIT:\t" + TIME);
        ACT(ShowLength,0);
    }
    ShowLength (A) {
        WRITELN("\tthere are " + LL(Teller.el) + " clients waiting ");
    }
INIT

```

Figura 2.1: Estructura del modelo SimpleTeller.

```

[erikvelasquez@localhost 1stDemo]$ javac -cp ../../galatea.jar *.java
[erikvelasquez@localhost 1stDemo]$ java -cp ../../galatea.jar SimpleTeller
ARRIVE: 0.0      there are 0 clients waiting
ARRIVE: 0.35297042343458224      there are 1 clients waiting
ARRIVE: 0.8880618422822584      there are 2 clients waiting
ARRIVE: 1.883912101800299      there are 3 clients waiting
ARRIVE: 2.7537697751706314      there are 4 clients waiting
ARRIVE: 3.5924340327328794      there are 5 clients waiting
EXIT: 4.108969502395377      there are 4 clients waiting
ARRIVE: 5.47806042440434      there are 5 clients waiting
EXIT: 6.274008250407743      there are 4 clients waiting
ARRIVE: 9.481019837013019      there are 5 clients waiting
EXIT: 9.565752362418394      there are 4 clients waiting
[erikvelasquez@localhost 1stDemo]$
[erikvelasquez@localhost 1stDemo]$

```

Figura 2.2: Ejecución del ejemplo SimpleTeller.

2.4 Servicio Web

Un servicio web (Web Service) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La World Wide Web Consortium lo define como “...un sistema de software diseñado para soportar interacción interoperable máquina a máquina sobre una red. Este tiene una interface descrita en un formato procesable por una máquina (específicamente WSDL). Otros sistemas interactúan con el servicios web en una manera prescrita por su descripción usando mensajes SOAP, típicamente enviados usando HTTP con una serialización XML en relación con otros estándares relacionados con la web” (Machuca, 2011).

2.5 Herramientas usadas para el desarrollo del sistema

A continuación se describen las herramientas usadas para el desarrollo del sistema web de simulación de eventos discretos GALATEA:

2.5.1 Modelo Vista Controlador (MVC)

El modelo vista controlador (MVC) es un patrón de diseño de software, que separa los datos y la lógica de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Este patrón de diseño es muy popular en el marco de aplicaciones web ya que su abstracción permite escribir software altamente desacoplado y fácil de mantener y escalar. La figura 2.3 muestra la relación entre los módulos de MVC.

Los modelos son los componentes de la aplicación del sistema que realmente hacen el trabajo. Se mantienen muy distintos de las vistas, que muestran aspectos de los modelos. Los controladores se utilizan para enviar mensajes a los modelos, y proporcionar la interfaz entre el modelo con sus vistas asociadas y los dispositivos de interfaz de usuario interactivas (por ejemplo, teclado, ratón). Cada vista se puede pensar que está estrechamente vinculada con un controlador, que tienen cada uno exactamente un modelo, sino un modelo puede tener muchos pares vista / controlador (Krasner y Pope, 1988).

Modelo

El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, así por ejemplo un sistema de administración de datos climatológicos tendrá un modelo que representará la temperatura, la humedad ambiental, el estado del tiempo esperado, etc. sin tomar en cuenta ni la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, sin tener relación con ninguna otra entidad dentro de la aplicación (Pantoja, 2004).

```
1  """folder model"""
2  class Folder(models.Model):
3      """Attributes"""
4      user = models.ForeignKey(User, on_delete=models.CASCADE)
5      name = models.CharField(max_length=500)
6      path = models.CharField(max_length=1000)
7      father = models.IntegerField(default=0)
```

```

8     active = models.BooleanField(default=True)
9     permission = models.IntegerField(default=0, choices=STATUS_PERMISSION)
10    """file model"""
11    class File(models.Model):
12        """Atributes"""
13        folder = models.ForeignKey(Folder, on_delete=models.CASCADE)
14        name = models.CharField(max_length=500)
15        active = models.BooleanField(default=True)
16        permission = models.IntegerField(default=0, choices=STATUS_PERMISSION)

```

Los modelos dentro de nuestro sistema son representados en clases, las cuales a su vez representan los objetos a ser almacenados dentro de nuestra base de datos. Los modelos de datos "Folder" y "File", se representan junto con sus atributos, tipos de datos y relaciones.

Vista

Las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo; así por ejemplo, se puede tener una vista mostrando la hora del sistema como un reloj analógico y otra vista mostrando la misma información como un reloj digital (Pantoja, 2004).

```

1  {% extends 'website/base.html' %}
2  {% load staticfiles %}
3
4  {% block title_page %}Tesis| Dashboard{% endblock %}
5
6  {% block title_content %}
7      Dashboard
8      <small>Index</small>
9  {% endblock %}
10
11 {% block level %}
12     <li><a href="{% url 'website:admin_index' %}"><i class="fa fa-dashboard"></i>
13         Dashboard</a></li>
14     {% if root_folder.father != 0 %}
15         <li><a href="{% url 'website:admin_show_folder' root_folder.father %}"><i class="fa
16             fa-folder"></i> Folder UP</a></li>
17     {% endif %}
18     <li class="active">Index</li>
19 {% endblock %}
20
21 {% block content %}

```

```

20     <div class="row">
21         <div class="col-xs-6">
22             <a href="#" data-toggle="modal" data-target="#myModal" class="btn btn-block
                btn-warning">New Folder</a>
23         </div>
24         <div class="col-xs-6">
25             <a href="#" data-toggle="modal" data-target="#myModalEdit" class="btn btn-block
                btn-primary">Edit Folder</a>
26         </div>
27         <div class="col-xs-6">
28             <a href="#" data-toggle="modal" data-target="#myModalDelete" class="btn btn-block
                btn-danger">Delete Folder</a>
29         </div>
30         <div class="col-xs-6">
31             <a href="#" data-toggle="modal" data-target="#myModalNewFile" class="btn
                btn-block btn-success">New File</a>
32         </div>
33         <div class="col-xs-6">
34             <a href="#" data-toggle="modal" data-target="#myModalUploadFile" class="btn
                btn-block btn-primary">Upload File</a>
35         </div>
36         <div class="col-xs-6">
37             <a href="#" data-toggle="modal" data-target="#myModalMove" class="btn btn-block
                btn-warning">Move Folder</a>
38         </div>
39         <div class="col-xs-6">
40             <a href="#" data-toggle="modal" data-target="#myNewProject" class="btn btn-block
                btn-warning">New Project</a>
41         </div>
42     </div>
43     <div class="box">
44         <div class="box-header">
45             <h3 class="box-title">Admin Dashboard</h3>
46         </div>
47         <div class="box-body">
48             {% for folder in folders %}
49                 <a class="btn btn-app" href="{% url 'website:admin_show_folder' folder.id %}">
50                     <i class="fa fa-folder"></i> {{folder.name}}
51                 </a>
52             {% endfor %}
53             {% for file in files %}

```

```

54         <a class="btn btn-app" href="{% url 'website:admin_show_file' file.id %}">
55         <i class="fa fa-file"></i> {{file.name}}
56     </a>
57     {% endfor %}
58 </div><!-- /.box-body -->
59 </div><!-- /.box -->
60 {% endblock %}

```

Esta vista es un archivo html, la cual define la forma en que los datos van a ser mostrados al usuario. La vista "show folder" muestra la información almacenada dentro de la carpeta.

Controlador

El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador (Pantoja, 2004).

```

1  @login_required(login_url="/login/")
2  @has_role_decorator('system_admin')
3  def admin_folder_show(request, folder_id):
4      user = User.objects.get(id = request.user.id)
5      root_folder = Folder.objects.get(id=folder_id)
6      folders = Folder.objects.filter(father=root_folder.id)
7      files = File.objects.filter(folder = root_folder.id)
8      all_folders = Folder.objects.filter(user = user.id)
9      if root_folder.father == 0:
10         return redirect("/admin/")
11     return
        render(request, 'admin/show_folder.html', {'root_folder':root_folder, 'folders':folders, 'files':files,
        'all_folders':all_folders})

```

Este controlador es el encargado de enlazar la petición realizada por el usuario sobre un modelo de datos, y ser relacionada con su vista correspondiente, para su posterior visualización por parte del usuario. En la fracción de código mostrada vemos el controlador asociado a "show folder", el cual busca el modelo "folder" y lo asocia a su vista correspondiente "show_folder.html".

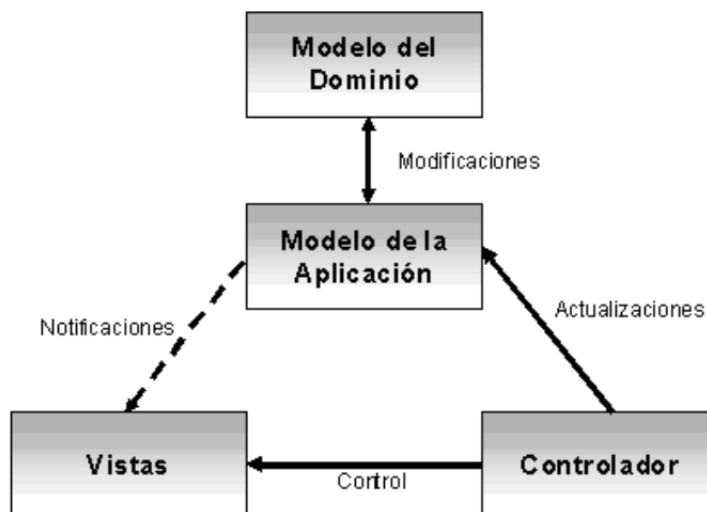


Figura 2.3: Relaciones entre los módulos del patrón MVC.

2.5.2 Django

Django (Foundation, 2015) es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–controlador. La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, Don't Repeat Yourself) (Foundation, 2015). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. Django permite construir aplicaciones web rápidamente, gracias a su filosofía de baterías incluidas, es decir, que incluye una inmensa gama de características comunes a la mayoría de las aplicaciones web como las validaciones, autenticación de usuarios, manejo de sesiones entre muchos otros (Foundation, 2015).

Funcionamiento de Django

Cuando Django recibe una petición ésta pasa por un despachador de URLs (URL Dispatcher), cuya tarea es emparejar el URL con una vista y delegar a la vista el manejo de la petición. La vista contiene la lógica necesaria para atender la petición entrante. Generalmente esto consiste en retirar o actualizar algunos datos del modelo, que a su vez se comunica con el manejador de base de datos, finalmente la vista combina los datos retirados de la base de datos, la petición y la sesión activa con una plantilla (template) para generar la respuesta de será devuelta.

Django ha demostrado ser escalable y flexible (Pérez, 2016), se sabe de instancias de Django atendiendo ráfagas de cincuenta mil peticiones por segundo (Pérez, 2016), además cuenta con una extensa comunidad de colaboradores y amplia documentación.

2.5.3 Bases de Datos

Una base de datos o banco de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Según (Oracle, 2015), una base de datos es una colección de datos estructurados. Puede ser cualquier cosa desde una simple lista de compras. una galería de fotos o las bastas cantidades de información en una red corporativa. Para agregar, acceder y procesar los datos almacenados en una base de datos, se necesita un sistema manejador de base de datos. Ya que los computadores hacen un muy buen trabajo manejando grandes cantidades de datos, los sistemas manejadores de bases datos juegan un papel central en la computación como utilidades independientes o partes de otras aplicaciones.

El sistema web de simulación permite la administración de usuarios, archivos y carpetas. De los cuales debemos guardar y consultar toda la información necesaria continuamente, para ello usamos una base de datos la cual definimos en la sección 3.1.6.

Debido a su alta confiabilidad y estabilidad, se utilizaron sistemas manejadores de bases de datos relacionales como: MySQL y PostgreSQL. Ambos manejadores fueron usados durante el desarrollo del sistema, para probar la portabilidad y funcionalidad del mismo. Debido a que Django utilizada MVC, permite variar el manejador de base de datos, sin modificar ningún elemento entre los modelos de nuestros datos.

MySQL

Mysql es un sistema manejador de bases de datos relacionales (RDBMS por sus siglas en ingles) de código abierto bajo la licencia GPL (GNU General Public License). Mysql se caracteriza por ser rápido, confiable escalable y fácil de usar, es posible instalar Mysql tanto en una maquina junto a otras aplicaciones como servidores web o también instalarlo en máquinas dedicadas para que use todo el poder de cómputo disponible. Mysql posee características para ejecutarse en clusters de máquinas junto con un motor de replicación para obtener una alta escalabilidad (Oracle, 2015).

PostgreSQL

PostgreSQL es un sistema de base de datos de gran alcance, de código abierto objeto-relacional. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de fiabilidad, integridad de datos y correctitud. Se ejecuta en todos los principales sistemas operativos, incluyendo Linux, UNIX, y Windows. Es totalmente compatible con ACID, tiene soporte

completo para claves foráneas, combinaciones, vistas, triggers y procedimientos almacenados (en varios idiomas). Incluye más SQL: 2008 tipos de datos, incluyendo entero, numérico, Boolean, CHAR, VARCHAR, DATE, INTERVALO y TIMESTAMP. También es compatible con el almacenamiento de objetos binarios grandes, como imágenes, sonidos, o de vídeo. Tiene interfaces de programación nativo de C / C ++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre otros, y la documentación excepcional (PostgreSQL, 2016).

Mapeo Objeto-Relacional

El Mapeo Objeto-Relacional (ORM por sus siglas en inglés) es un método para interactuar con bases de datos relacionales desde el paradigma de la programación orientada a objetos, de esta manera es posible aprovechar conceptos como herencia y polimorfismo. Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

```

1 users = User.objects.all() #obtener todos los usuarios registrados en la DB
2 user = User.objects.get(id=user_id) #obtener la información de un usuario específico
3 folders_count = Folder.objects.filter(user_id = user_id).count() #contar las carpetas que
   pertenecen al usuario
4 folders = Folder.objects.filter(user_id = user_id) #obtener todas las carpetas asociadas al
   usuario
5 files = File.objects.filter(folder__user_id = user_id) #obtener todos los archivos asociados
   al usuario

```

Todas estas llamadas están escritas en Python y usando la definición de ORM, este manejador encapsula las llamadas a la base de datos y genera automáticamente los SQLs específicos de cada llamada.

```

1 select * from users #users = User.objects.all()
2 select * from users where id = "user_id" #user = User.objects.get(id=user_id)
3 select count(*) from folders where user_id = "user_id" #folders_count =
   Folder.objects.filter(user_id = user_id).count()
4 select * from folders where user_id = "user_id" #folders = Folder.objects.filter(user_id =
   user_id)
5 select * from files where #files = File.objects.filter(folder__user_id = user_id)

```

El uso de un ORM simplifica enormemente el manejo de la estructura de datos subyacente ya que permite al programador manejar los datos a un mayor nivel de abstracción como si fueran objetos, sin necesidad de general manualmente las consultas SQL, además ésta capa de abstracción permite

desacoplar el código de la aplicación de los detalles específicos de cada RDBMS (Pérez, 2016).

2.5.4 JSON

JSON (JavaScript Object Notation) es un formato textual de intercambio y almacenamiento de datos no estructurados, JSON posee un formato que es fácil de leer para humanos y fácil de interpretar para máquinas y más ligero que XML por lo que se ha popularizado para el desarrollo de APIs. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza las convenciones que son familiares para los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen JSON un lenguaje ideal de intercambio de datos.

```
1 @login_required(login_url="/login/")
2 @has_role_decorator('system_admin')
3 def admin_compile_file(request, file_id):
4     file = File.objects.get(id=file_id)
5     com_java = compile_java(file.id)
6     return JsonResponse({"data":com_java})
7
8 @login_required(login_url="/login/")
9 @has_role_decorator('system_admin')
10 def admin_execute_file(request, file_id):
11     #get arguments for compilation
12     arguments = request.GET['arguments']
13     file = File.objects.get(id=file_id)
14     com_java = execute_java(file.id, arguments)
15     return JsonResponse({"data":com_java})
```

Las llamadas de compilación y ejecución de los modelos de simulación, hacen uso del formato JSON para el retorno de la información a la vista del usuario.

2.5.5 AJAX

AJAX (Asynchronous JavaScript And XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones. Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página

se detiene hasta la espera de la respuesta por parte del servidor (Fuentes, 2009).

```
1 <script type="text/javascript">
2 var editor = CodeMirror.fromTextArea(document.getElementById("info_file"), { lineNumbers:
    true, matchBrackets: true, mode: "text/x-java" });
3 $( "#compile_file" ).click(function() {
4     $("#code_file").text("Compiling file");
5     $.ajax({
6         url : "{% url 'website:admin_compile_file' file.id %}", // the endpoint
7         type : "GET", // http method
8         data : { }, // data sent with the post request
9
10        // handle a successful response
11        success : function(json) {
12            console.log(json); // log the returned json to the console
13            console.log("success"); // another sanity check
14            $("#code_file").text(json.data);
15            $("#code_file").text("Compiled file");
16        },
17
18        // handle a non-successful response
19        error : function(xhr,errmsg,err) {
20            console.log("Error"); // provide a bit more info about the error to the console
21        }
22    });
23 });
24
25 $( "#execute_file" ).click(function() {
26     $("#code_file").text("Compiling file");
27     $.ajax({
28         url : "{% url 'website:admin_execute_file' file.id %}", // the endpoint
29         type : "GET", // http method
30         data : { arguments: $('#arguments').val() }, // data sent with the post request
31
32        // handle a successful response
33        success : function(json) {
34            console.log($('#arguments').val());
35            console.log(json); // log the returned json to the console
36            console.log("success"); // another sanity check
37            $("#code_file").text(json.data);
38            $("#data_execution").val(json.data);
```

```
39         $("#export_data").removeAttr("disabled");
40     },
41
42     // handle a non-successful response
43     error : function(xhr,errmsg,err) {
44         console.log("Error"); // provide a bit more info about the error to the console
45     }
46 });
47 });
48 </script>
```

Para crear la comunicación asíncrona con el servidor, nuestras llamadas de compilación y ejecución de modelos de simulación se realiza mediante una llamada AJAX. Esto nos permite continuar realizando tareas sobre nuestros modelos de simulación, mediante el servidor ejecuta esos procesos de compilación y ejecución.

Capítulo 3

Desarrollo del Sistema WEB

El sistema de simulación de eventos discretos mediante la web esta conformado por tres elementos principales: (1) La aplicación Web que funciona como coordinador del sistema, control de usuarios y roles; (2) Administración de archivos que permite llevar el control de acceso, creación y modificación de los archivos y carpetas, (3) El Sistema de Integración con GALATEA, que ofrece las funcionalidades de simulación, conexión y control con el motor de simulación.

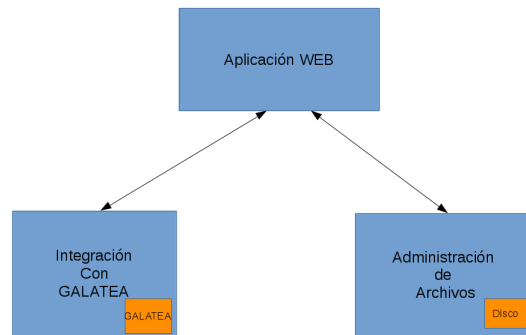


Figura 3.1: Sistema de Simulación Web GALATEA.

En la figura 3.1 se observa la relación que existe entre los componentes principales que conforman el sistema. El punto de entrada principal es la aplicación Web, que es la encargada de manejar el acceso de los usuarios a las distintas funcionalidades. Esto lo realiza mediante la implementación de roles de usuarios y manejo de permisologías. La aplicación web controla la visualización, acceso y modificación de los distintos archivos y carpetas, mediante peticiones sobre el módulo de administración de archivos. Otro aspecto fundamental de la aplicación web es el control y visualización de las simulaciones,

mediante el acceso al sistema de Integración con GALATEA.

Para el módulo de Administración de Archivos se desarrolló una API, la cual permite llevar el control de creación, modificación y eliminación de archivos y carpetas dentro del sistema. Esta API está escrita en python y maneja el control de excepciones u errores que pueden aparecer durante su ejecución, al igual permite verificar el espacio en disco usado por cada usuario. En la sección 3.2 se especifica con mas detalles la funcionalidad de este módulo.

El sistema de Integración con GALATEA es una API de interconexión entre nuestra aplicación web y el simulador GALATEA. Como ambos sistemas están desarrollados en ambientes distintos, con códigos distintos, se buscó desarrollar un mecanismo de comunicación entre ambos sistemas. La API permite crear una instancia del Simulador GALATEA y ejecutarlo, controlar la compilación y los parámetros de ejecución de la simulación, esto lo desarrollamos en python mediante la implementación de control de procesos y socket. En el capítulo siguiente especificaremos a detalle en funcionamiento de este sistema de integración.

3.1 Diseño de la aplicación WEB

Para el desarrollo de esta aplicación se usó el framework de desarrollo web Django que implementa un patrón de diseño MVC.

3.1.1 Arquitectura

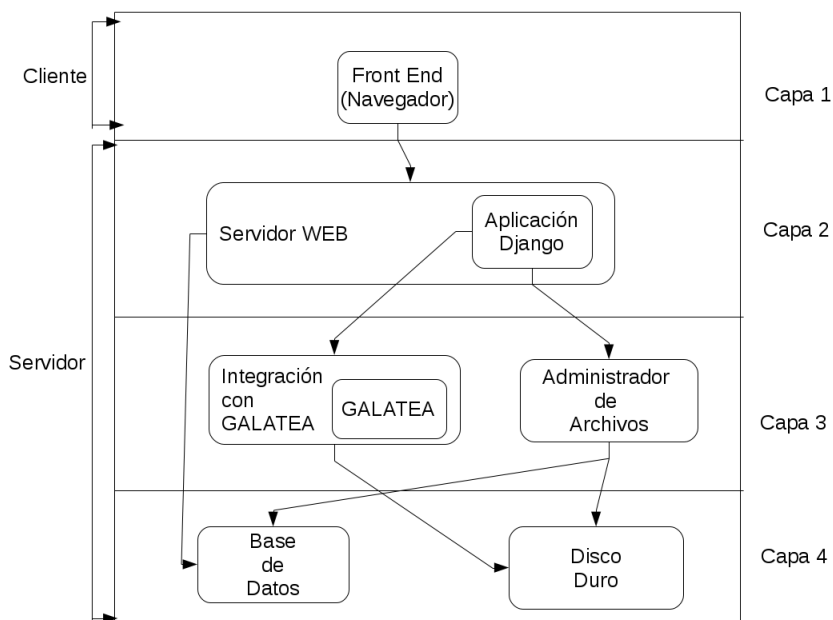


Figura 3.2: Arquitectura de la aplicación web.

La arquitectura del sistema web de simulación GALATEA consiste en cuatro capas, la capa superior o capa de presentación se ejecuta en el navegador del cliente, para ver los resultados obtenidos de la simulación, visualización de archivos, entre otras acciones que puede realizar el usuario; esta capa se comunica con la capa dos o capa de servicio, que es ejecutada por el servidor web y es la responsable de atender las peticiones de los usuarios del sistema; la capa tres es la encargada de las peticiones a los otros módulos que conforman el sistema, comunicación con el módulo de manejo de archivos e integración con el simulador GALATEA; la capa cuatro o capa de datos aloja los datos de la aplicación como usuarios, archivos, carpetas, modelos, trazas y datos de simulaciones, entre otros.

3.1.2 Capa 1: Presentación

La capa de presentación es la interfaz gráfica que permite la interacción del usuario con el sistema de simulación Web. Esta capa consiste en los despliegues de navegadores en cualquier dispositivo conectado a Internet y su propósito es ofrecer la usuario una interfaz gráfica para ingresar el sistema, configurar y manejar simulaciones, observar resultados, entre otros.

La capa de presentación puede ser extendida para la implementación de distintos front-ends como aplicaciones nativas para sistemas operativos móviles o de escritorio a través de APIs programáticas.

3.1.3 Capa 2: Servidor Web

El servidor web es el punto de entrada a la aplicación, éste responde a las peticiones realizadas desde el front-end a través del protocolo HTTP, el enfoque del servidor web es manejar tareas ligeras de la forma mas rápida posible y retornar respuestas para ofrecer al usuario baja latencia en su interacción con el sistema.

Ruta	Acción
/static/	Servicio de archivo estático
/media/	Servicio de archivo estático
/	re-dirección a la aplicación

Tabla 3.1: Tabla de enrutamiento del servidor web.

El servidor web debe manejar dos tipos de peticiones: aquellas que requieren respuestas dinámicas ajustadas a los datos específicos que pertenecen a cada usuario en particular y peticiones de archivos estáticos como CSS, JavaScript, imágenes o archivos HTML pre-definidos que cambien poco o rara vez. A pesar de que la aplicación web es capaz de manejar ambos tipos de peticiones, cada una de ellas tendrá que pasar por todo el pipeline de Django (middlewares, resolución de URLs, procesamiento de la vista, etc). Esta complejidad es innecesaria cuando se trata de archivos estáticos y va a sobrecargar el servidor web en entornos de producción.

Para manejar eficientemente todo tipo de peticiones, debemos implementar un mecanismo que sea capaz de servir los archivos estáticos rápidamente y que sirva como proxy entre el front-end y la

aplicación web. Hay una gran variedad de servidores como Apache¹, Nginx² o Lighttpd³ diseñados especialmente con este propósito, éstos tendrán una sencilla tabla de enrutamiento que decidirá cómo manejar las peticiones. La aplicación web podrá correr entonces en un proceso separado y sólo atenderá peticiones dinámicas, así se asegurará la eficiencia en el uso de los recursos computacionales disponibles.

3.1.4 Capa 3: Integración de Procesos

En esta capa integramos los procesos fundamentales para nuestro sistema, como lo son: La integración con el simulador GALATEA y la administración de archivos y carpetas. Se encarga de gestionar todas peticiones realizadas por el usuario ya sea para el manejo de archivos/carpetas o simulaciones, al igual controlar su correcto funcionamiento y resultado.

3.1.5 Capa 4: Datos

Incluye todos los sistemas encargados de almacenar o alojar los datos en forma persistente ya sea a través de bases de datos o sistemas de archivos.

Bases de datos

La base de datos es el almacén principal de los datos estructurados de la aplicación web, almacena todos tipo de datos de información como datos del usuario, referencias de los archivos, logs de información, entre otras cosas.

El diseño de la base de datos es una de los pilares fundamentales del diseño del sistema, el correcto diseño de las tablas de la base de datos tiene repercusiones importantes en la escalabilidad y tiempo de respuesta del sistema.

Archivos de usuario

Los archivos de usuario son de gran importancia a ser manejados en el sistema, estos pueden ser de configuración de perfiles como imágenes, y los archivos de los modelos de simulación que varían desde archivos Java, archivos GALATEA y archivos de texto con información de trazas y ayudas del modelo, entre otros.

Todos estos archivos son almacenamos en un sistema de almacenamiento secundario, el cual es configurado dentro de la aplicación y controlado para su correcto funcionamiento y acceso. Cada usuario cuenta dentro del sistema con una porción de almacenamiento para su uso.

¹Apache <https://www.apache.org/>

²Nginx <https://nginx.org/>

³Lighttpd <https://www.lighttpd.net/>

3.1.6 Diseño de la Base de Datos

Puesto que Django hace uso de un ORM para manejar la base de datos, desde el punto de vista de la aplicación web, cada tabla de la base de datos es una clase, por lo que es posible aprovechar todas las características de la programación orientada a objetos como: herencia de clases, clases abstractas, implementación de métodos específicos a un modelo y sobre carga de métodos.

A continuación se describen los modelos que forman parte de la base de datos:

Cada usuario registrado está representado por el modelo *user*, el cual contiene la información básica del usuario y su información de ingreso al sistema. Para información de perfil e imagen se maneja el modelo *UserImage*.

El manejo de roles se define en el modelo *group*, en el cual manejamos dos roles principales: el de *administrador* y el de *usuario*.

Las relaciones de usuarios se maneja mediante *Relationship*, lo cual nos permite definir la relación de *amistad* entre dos usuarios. Esto nos permitirá compartir archivos entre usuarios.

Los archivos son manejados por *File*, que permite definir la información básica y permisos de acceso a los mismos.

Las carpetas están definidas mediante *Folder*, que contiene la información básica y jerárquica de carpetas. Otro modelo muy relacionado es *DiskSpace*, que nos permite llevar el control de espacio en disco usado por el usuario.

Para compartir un archivo usamos el modelo *ShareFile*, que contiene la información del archivo y sus permisos compartidos.

Las carpetas compartidas en *ShareFolder*, que contienen la información de la carpeta compartida y sus permisos de acceso.

Hay una gran variedad de logs que se manejan los cuales se encuentran en distintas tablas como: *logsfile* para logs de archivos, *logsfolder* para las modificaciones de carpetas, entre otros.

Diagrama de entidad-relación

La figura 3.3 muestra las entidades principales y sus relaciones dentro de la base de datos.

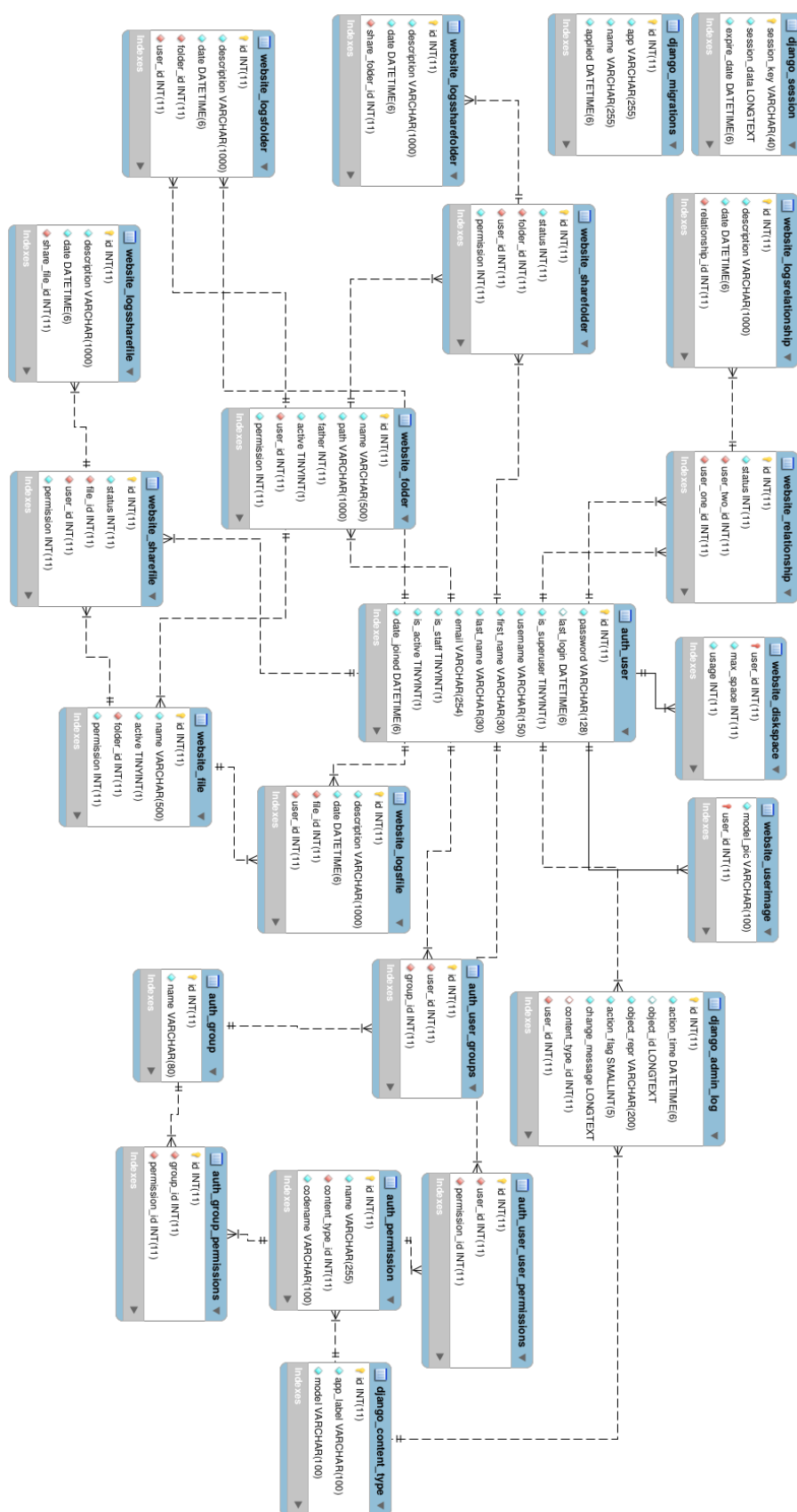


Figura 3.3: Entidades principales de la base de datos.

La figura 3.4 muestra las entidades relacionadas al control y manejo de usuarios, sus roles y perfiles

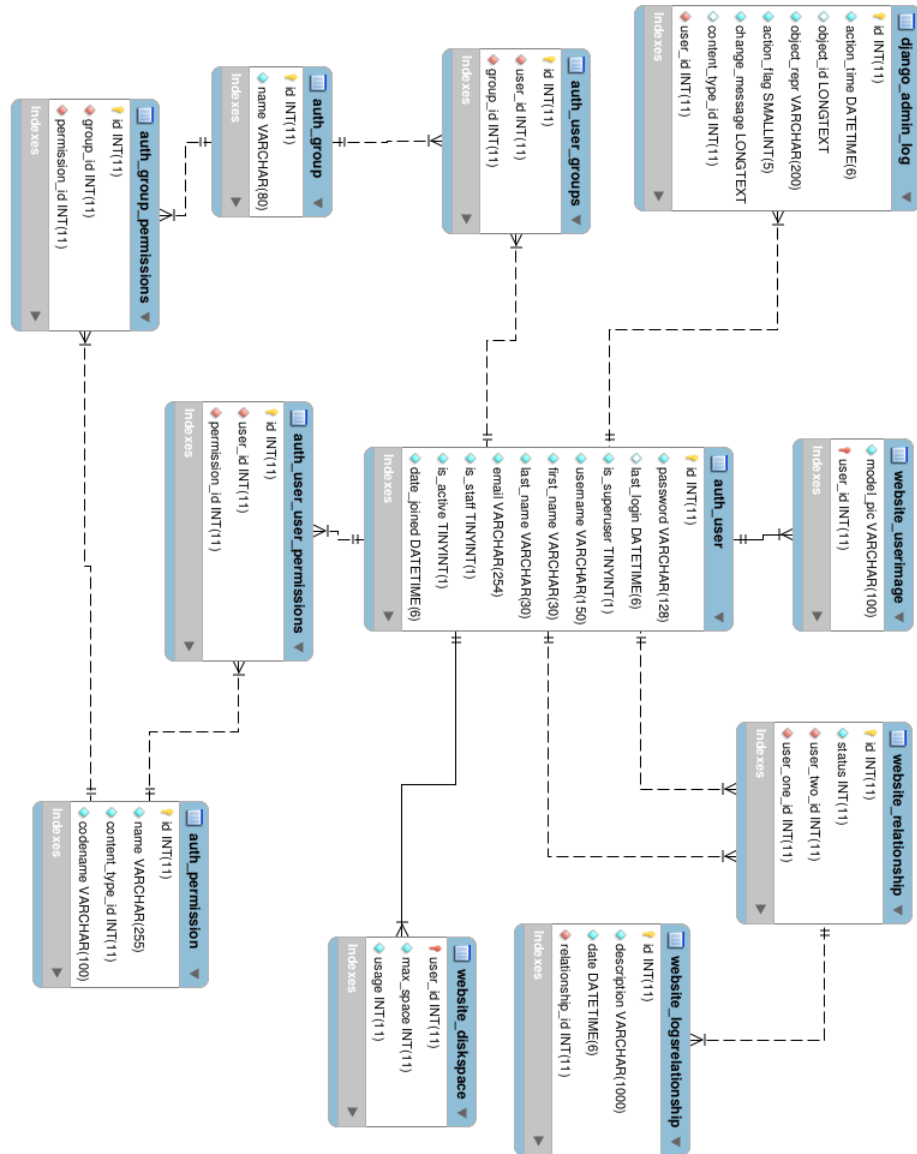


Figura 3.4: Entidades manejo de usuarios.

La figura 3.5 muestra la relación para el control de archivos y carpetas que puede manejar el usuario.

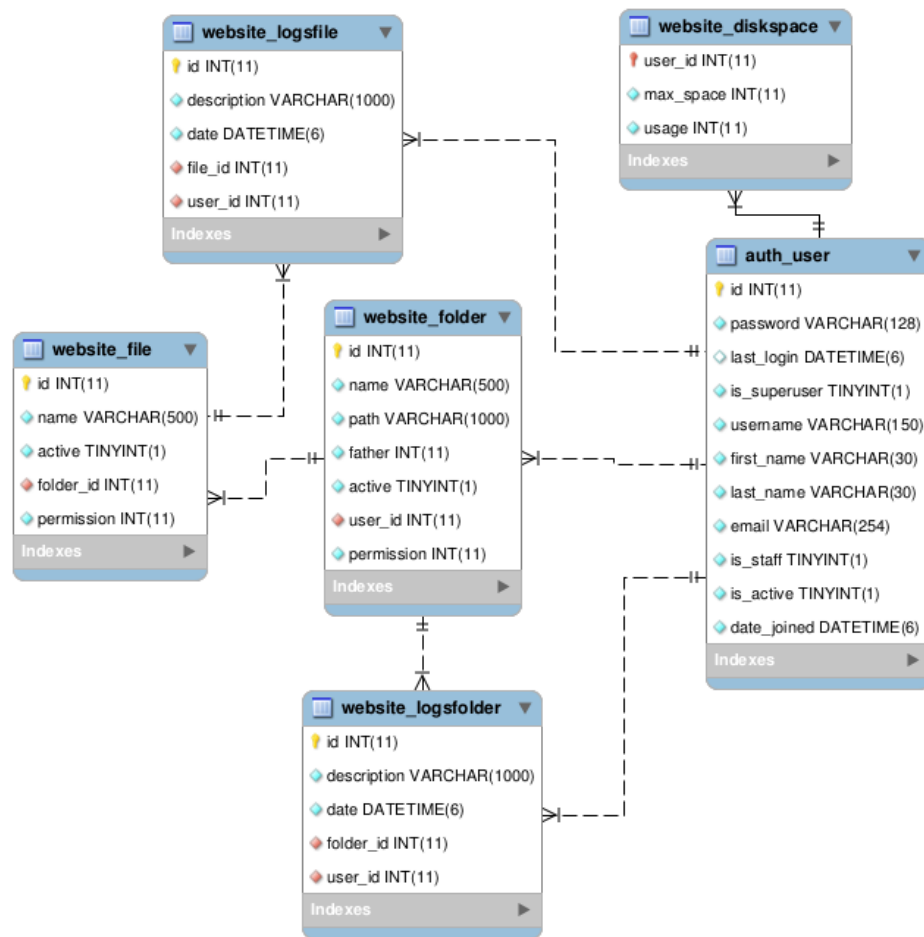


Figura 3.5: Entidades manejo archivos y carpetas.

La figura 3.6 muestra la relación para el control de archivos y carpetas compartidas entre otros usuarios.

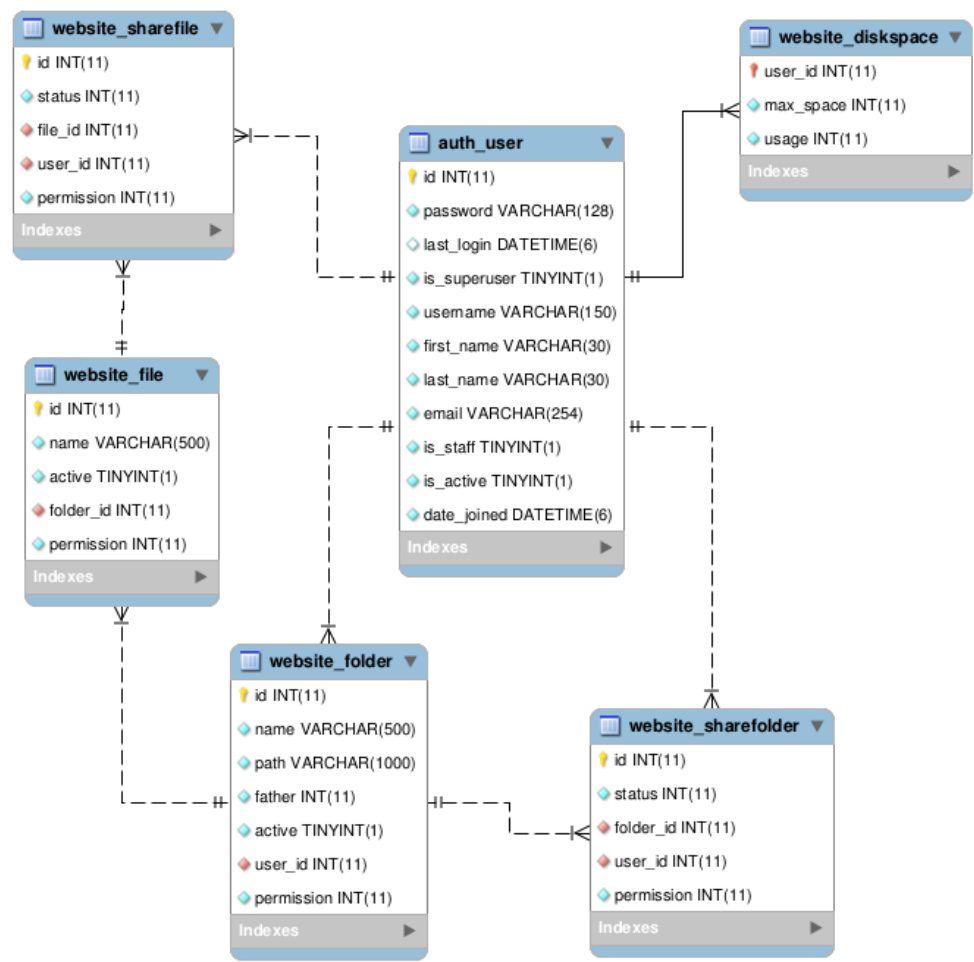


Figura 3.6: Entidades manejo archivos y carpetas compartidos.

Entidades

Las siguientes tablas resumen las entidades y la función de las mismas dentro del sistema web:

Nombre de la entidad	Función
<i>User</i>	Almacena la información básica de usuario y de acceso al sistema
<i>UIImage</i>	Guarda la imagen del perfil de cada usuario
<i>Group</i>	Define los roles a ser usados dentro del sistema (Administrador y Usuario)
<i>Relationship</i>	Define las relaciones entre usuarios

Tabla 3.2: Entidades relacionadas al manejo de usuarios.

Nombre de la entidad	Función
<i>File</i>	Almacena la información básica de archivo y su permisología de acceso
<i>Folder</i>	Información básica de una carpeta junto a su permisología de acceso
<i>DiskSpace</i>	Controla el espacio usado en disco por cada usuario

Tabla 3.3: Entidades relacionadas al manejo de archivos y carpetas.

Nombre de la entidad	Función
<i>ShareFolder</i>	Llevar control de la carpeta compartida
<i>ShareFile</i>	Almacenar la información del archivo compartido

Tabla 3.4: Entidades relacionadas al manejo de archivos y carpetas compartidos.

Relaciones

- **User - UserImage (1-1)**: Un usuario posee una imagen e información de perfil.
- **User - Folder (n-1)**: Un usuario puede poseer una o mas carpetas, pero una carpeta pertenece a un solo usuario.
- **User - DiskSpace (1-1)**: Un usuario tiene un solo espacio en disco asociado y configurado.
- **Folder - File (n-1)**: Una carpeta puede tener muchos archivos almacenados, pero un archivo tiene una carpeta asociada.
- **User - ShareFile (n-1)**: Un usuario puede poseer uno o mas archivos compartidas, pero un archivo pertenece a un solo usuario.
- **File - ShareFile (n-1)**: Un archivo puede estar compartido con muchos usuarios.
- **User - ShareFolder (n-1)**: Un usuario puede poseer una o mas carpetas compartidas, pero una carpeta pertenece a un solo usuario.
- **Folder - ShareFolder (n-1)**: Un carpeta puede estar compartida con muchos usuarios.
- **User - Relationship (n-1)**: Un usuario puede poseer una o mas relaciones de amistad con otros usuarios, pero la relación se define con un solo usuario.

3.1.7 Diseño de Pantallas

Todas las pantallas del sistema fueron creadas bajo el paradigma del diseño web adaptable, es decir, que pueden ajustarse a cualquier resolución de pantalla sin reducir la usabilidad o sacrificar la experiencia del usuario, el servidor no tiene que decidir entre un conjunto de plantillas para distintas resoluciones, sino que envía al cliente sólo documento HTML y este combina las reglas de presentación de archivos CSS y la lógica de archivos JavaScript para desplegar una página web adaptada al dispositivo del usuario.

Ya que una aplicación web está compuesta por decenas de vistas que comparten componentes como: barras de navegación, cabeceras y pie de páginas, Django incluye un micro-lenguaje de plantillas con

funcionalidades de herencia e inclusión, estructuras de repetición y decisión. Esto hace que sea posible tener una taxonomía de plantillas, de manera que sólo es necesario escribir los elementos comunes a un conjunto de plantillas una vez y crear nuevas pantallas en sobre otras. Se le llama rendering al proceso de combinar los datos de la base de datos y la petición del usuario, junto con una plantilla predefinida en la aplicación para generar páginas web personalizadas.

Todas las plantillas heredan de una plantilla base, ésta incluye la declaración del archivo HTML, la cabecera y define bloques que pueden ser sobrescritos por las plantillas "hijas". Los bloques principales definidos en nuestro diseño son: **(1) Bloque de título** permite definir el título de la página, **(2) Bloque de navegación** permite un acceso rápido a los sitios accedidos previamente, **(3) Bloque de contenido** bloque principal con el contenido de la página, **(4) Bloque de JavaScript** bloque para incluir archivos y métodos JavaScript. La plantilla general hereda de la *plantilla base* y esta a su vez incluye la información de la *barra lateral* y el *header* de la página.

A continuación se muestran las principales pantallas de la interfaz de usuario del servicio web:

- **Pantalla del Dashboard (Inicio):** Esta es la pantalla de inicio de un usuario registrado, en ella encontramos la información principal y las distintas opciones de navegación. Como eje central se encuentran los archivos y carpetas de nuestro sistema raíz.



Figura 3.7: Pantalla del Dashboard (Inicio).

- **Pantalla Observación de Carpeta:** En ella encontramos todos los archivos y carpetas que se encuentran dentro de nuestra carpeta, al igual observamos todas las opciones que podemos ejecutar dentro de ella.

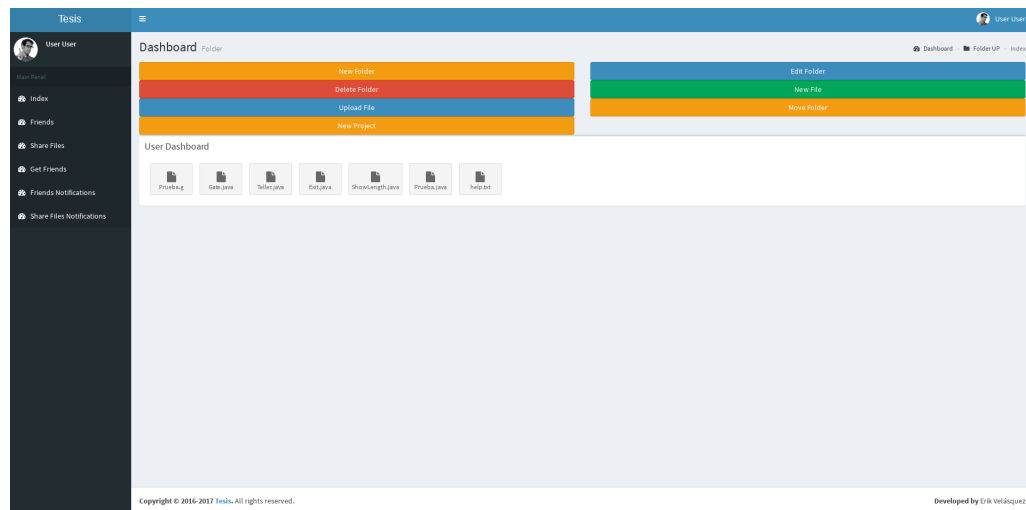


Figura 3.8: Pantalla Observación de Carpeta.

- **Pantalla Observación de Archivo:** Observamos la información contenida en el archivo, poseemos un editor web para editarlo, al igual todas las opciones que podemos manejar sobre el mismo.

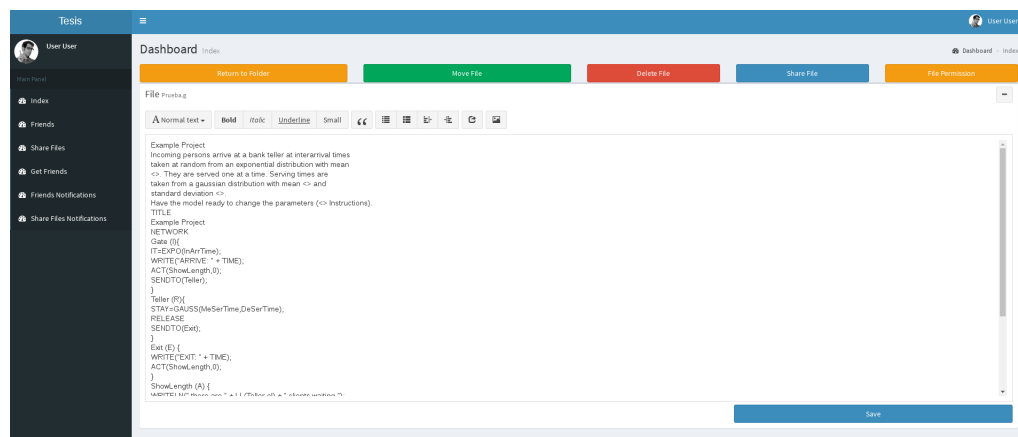


Figura 3.9: Pantalla Observación de Archivo.

- **Pantalla Compilación y Ejecución:** Aquí podemos acceder a las opciones de compilación y ejecución de nuestro archivo, junto con los resultados obtenidos de esos procesos.

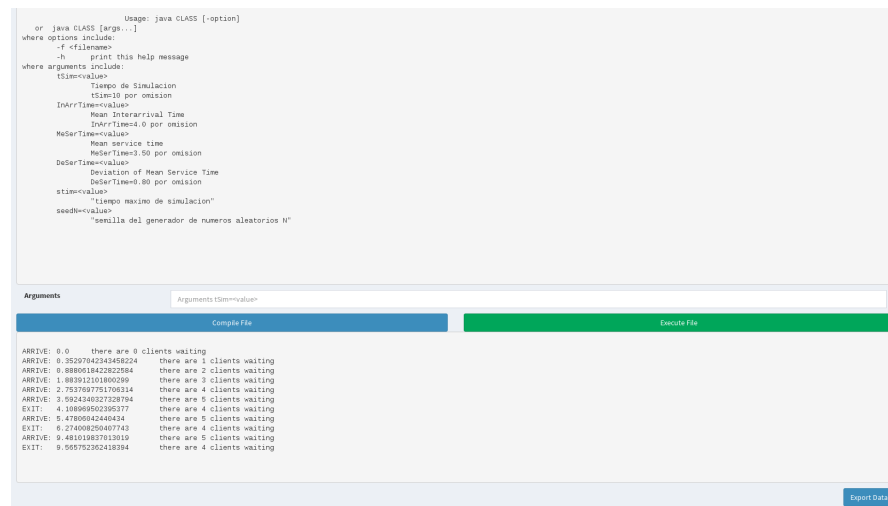


Figura 3.10: Pantalla Compilación y Ejecución.

- **Pantalla Control de Usuarios:** Esta pantalla solo es accedida por usuarios administradores del sistema, en ella tenemos la visualización de usuarios y podemos acceder a las distintas opciones de creación y edición de usuarios.

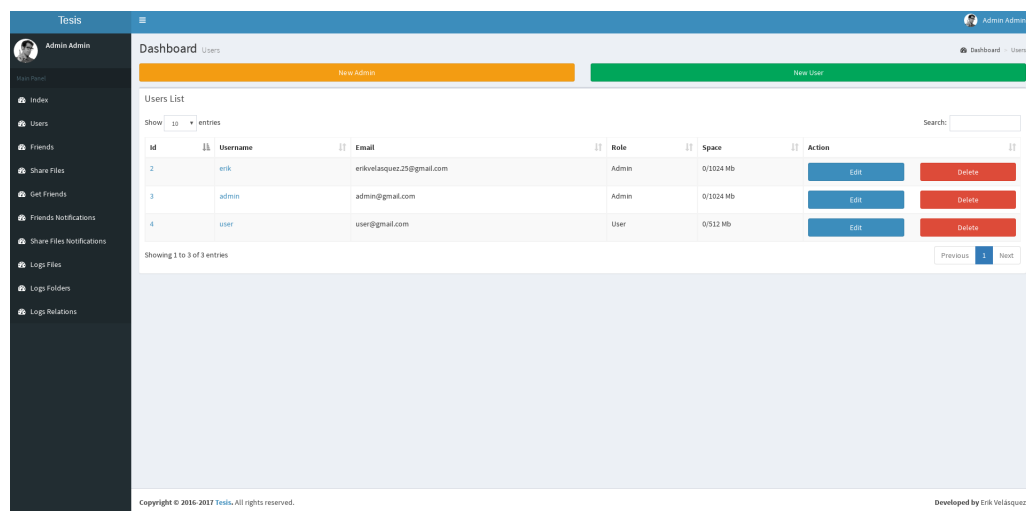


Figura 3.11: Pantalla Control de Usuarios.

3.1.8 Diseño de URLs

Ya que la aplicación web consiste en un conjunto grande de vistas, debemos diseñar URLs que apunten a cada una de ellas de forma ordenada y lógica. Para esto, se ha usado una convención en la que la unidad principal de operación como archivos, carpetas, usuarios anteceden a la operación real que se esta llamando; a continuación se muestran ejemplos de las reglas usadas para diseñar las URLs:

- Operaciones básicas de acceso:
 - **GET** `/admin/` accedemos al dashboard de un usuario administrador. Todas las rutas para acceso de un usuario administrador se definen con esta ruta inicial. Ejemplo `"http://localhost:8000/admin/users"`, retorna la lista de usuarios registrados en el sistema.
 - **GET** `/user/` accedemos al dashboard de un usuario registrado. Todas las rutas para acceso de un usuario registrado se definen con esta ruta inicial. Ejemplo `"http://localhost:8000/user/"` accede al dashboard del usuario.
 - **POST** `/user/create.file/` Ruta para la creación de un nuevo archivo.
 - **POST** `/user/create.folder/` Ruta para la creación de una nueva carpeta.
 - **GET** `/user/file/-id-` acceder a la información del archivo cuyo id dentro del sistema es -id-. Ejemplo `"http://localhost:8000/user/file/2/"`, accedemos a la información del archivo cuyo id dentro del sistema es 2.
 - **GET** `/user/folder/-id-` acceder a la información de una carpeta cuyo id dentro del sistema es -id-. Ejemplo `"http://localhost:8000/user/folder/15/"`, accedemos a la carpeta cuyo id en el sistema es 15, y pertenece a el usuario.
 - **GET** `/user/compile/-id-` compila el archivo cuyo id es -id- y retorna la información de compilación. Ejemplo `"GET /user/compile/file/2/ HTTP/1.1"` 200 12, compilamos el archivo cuyo id es 2, mediante una petición Ajax.
 - **GET** `/user/execute/-id-` ejecuta la simulación definida en el archivo cuyo id es -id- y retorna el resultado de dicha simulación. Ejemplo `"GET /user/execute/file/2/?arguments= HTTP/1.1"` 200 634, ejecutamos el modelo de simulación, mediante una petición Ajax.

3.2 Administrador de Archivos

Un administrador de archivos, gestor de archivos o explorador de archivos es un programa que otorga al usuario la capacidad de trabajar, gestionar o administrar archivos y carpetas. Esto lo realiza mediante una interfaz de usuario que entre muchas operaciones permiten al usuario abrir, crear, editar, reproducir, ya sean archivos o carpetas. En el sistema de simulación web GALATEA se desarrolló un administrador de archivos web que permite crear, editar, visualizar proyectos de simulación entre algunas otras opciones. Los administradores de archivos basados en la web son típicamente scripts escritos en cualquiera de los lenguajes para desarrollo web como PHP, Ajax, Perl, JSP u otro lenguaje de servidor.

Como todo administrador de archivos existente, éste debe trabajar bajo un sistema de archivos el cual dependerá del sistema operativo en el que se encuentre instalado. Sus principales funciones son la asignación de espacio a los archivos, la administración del espacio libre y del acceso a los datos resguardados. Estructuran la información guardada en un dispositivo de almacenamiento de

datos o unidad de almacenamiento (normalmente un disco duro de una computadora), que luego será representada ya sea textual o gráficamente utilizando un gestor de archivos.

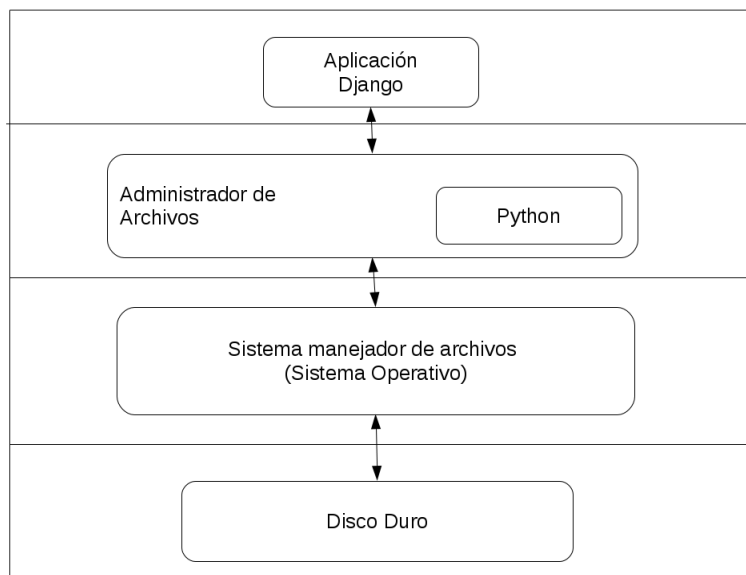


Figura 3.12: Estructura del Administrador de Archivos y Carpetas.

El framework de desarrollo web Django usado para desarrollar este proyecto nos permite tener una interacción directa con los archivos y carpetas. Esto debido a el lenguaje principal es que se encuentra escrito (Python), el cual es un lenguaje de programación potente y versátil para desarrollar cualquier tipo de tareas. Por ende, nuestro administrador de archivos y carpetas esta escrito en python, haciendo uso de las librerías de control archivos y carpetas que el lenguaje nos proporciona.

En la figura 3.12 podemos observar la estructura y diseño de nuestro administrador de archivos, como capa base tenemos a python y sus librerías de manejo de archivos, el cual interactua con el sistema de archivos de nuestro sistema operativo y ejecuta las operaciones a nivel físico en nuestra maquina o servidor. Como interfaz de usuario tenemos a Django, al ser un framework web nos permite una interacción simple y elegante con el usuario para controlar nuestros archivos y carpetas. En el punto central se encuentra nuestro código en python que sirve como una APIs de interconexión entre nuestra interfaz de usuario y el sistema manejador de archivo.

Una configuración de nuestro sistema es la integración de permisologías a los archivos y carpetas, lo cual nos permite compartirlos entre los distintos usuarios del sistema. Esto lo logramos mediante la definición de un conjunto de permisologías como lo observamos en la tabla 3.5. Estas configuraciones de permisos están agregados a los modelos en la base de datos, controlados y configurados por el usuario mediante la interfaz de usuario.

Permiso	Definición
<i>Private (0)</i>	Solo el dueño puede acceder a él
<i>Show (1)</i>	Otros usuarios pueden acceder al él solo para lectura
<i>Edit (2)</i>	Otros usuarios pueden acceder al él para lectura y escritura
<i>Public (3)</i>	Otros usuarios pueden acceder al él y tienen control total

Tabla 3.5: Permisología de Archivos y Carpetas.

3.2.1 Control de Archivos

El administrador de archivos tiene como modelos de datos principales para el manejo de archivos a File, LogsFile y DiskSpace, los cuales fueron explicados en la sección 3.1.6. Para el manejo de archivos se han desarrollado cinco métodos principales los cuales son:

- Create File: el cual nos permite crear un archivo en una carpeta determinada.
- Show File: nos permite acceder a los datos dentro del archivo para ser visualizados por el usuario.
- Edit File: edita la información que se encuentra en el archivo.
- Move File: permite mover el archivo desde su carpeta origen a una nueva carpeta destino.
- Delete File: elimina el archivo de nuestro sistema.

Todos los archivos desde su creación tienen su dueño y permisología asociada. El manejo de archivos dentro del sistema tiene las funcionalidades necesarias para permitir una gran variedad de acciones a realizar.

3.2.2 Control de Carpetas

El control de carpetas es uno de los puntos mas importantes dentro de nuestro administrador de archivos, debido a que todos los archivos están relacionados con una carpeta y las carpetas en sí tienen una relación jerárquica entre ellas. En la figura 3.13 observamos esa relación jerárquica en nuestro manejo de carpetas.

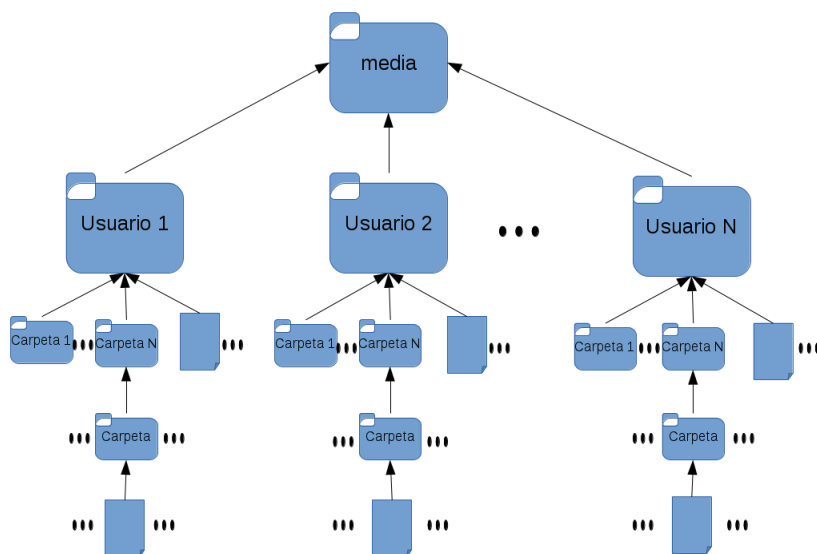


Figura 3.13: Jerarquía de Carpetas.

Cómo raíz de nuestro sistema de carpetas tenemos a la carpeta *media*, la cual es el punto de partida para todo nuestro manejo de archivos y carpetas. En el segundo nivel de jerarquía tenemos las carpetas de *usuari@s*, las cuales funcionan como punto de raíz para su sistema interno de archivos y carpetas. El nombre que identifica a cada carpeta de *usuari@s* es propiamente dicho el "username" que identifica como único a cada usuario dentro del sistema.

Para el correcto manejo de carpetas se definieron una serie de métodos:

- Create Folder: el cual nos permite crear una carpeta bajo cierta jerarquía.
- Show Folder: nos permite acceder a los archivos y carpetas internos.
- Edit Folder: editar la información básica de la carpeta.
- Move Folder: esta es una función delicada, debido a los archivos y carpetas internos, para ello se definió el método de manera recursiva y así modificar todo sin problemas.
- Delete Folder: elimina toda la información de la carpeta junto con sus archivos y carpetas internos.

Debido a que el espacio en disco físico es limitado, cada usuario tiene una porción de espacio asociado para su manejo de archivos y carpetas. Este espacio es constantemente monitoreado por el administrador de archivos y es una de las restricciones principales para el correcto funcionamiento del servicio. Esta porción de espacio en disco es fácilmente administrable por el usuario administrador de nuestro sistema web.

El sistema manejador de archivos y carpetas se desarrolló usando llamadas a los métodos de manejo de archivos y carpetas que proporciona el sistema operativo donde el sistema se encuentre instalado. Esto permite que dependiendo el sistema operativo usado y los recursos de hardware disponibles podemos hacer uso de las nuevas tecnologías disponibles como: "hot swap", modificar tamaños de las particiones en tiempo real y otras tecnologías que los nuevos servidores nos permiten para el manejo de archivos y carpetas.

3.2.3 Control de Permisos

Mediante el sistema web de simulación GALATEA se busca que los usuarios tengan interacción entre ellos y puedan trabajar en conjunto, para así fortalecer y mejorar sus proyectos de simulación. Para ellos se implementó un sistema de permisos de acceso y modificación tanto para los archivos como carpetas dentro del sistema.

Cómo lo muestra la tabla 3.5 existen cuatro permisos bien definidos, pero debemos recalcar que estos permisos se enfocan en dos tipos de usuarios propiamente dichos:

- Usuarios Amigos: estos son como su nombre lo indica *amigos* en el sistema (conocidos). El privilegio principal de tener el *status amigo*, es que permite compartir archivos y carpetas entre ellos.
- Usuarios NO Amigos: son usuarios que no poseen relación en el sistema.

Esto nos permite definir ambas permisologías para cada tipo de usuario. Para los usuarios *NO Amigos*, mantenemos el mismo tipo de permisos y depende del usuario dueño cambiar o no los permisos de sus archivos para darle acceso a este tipo de usuarios. Un *usuario amigo* tiene una relación directa con el usuario dueño, lo cual nos permite compartir nuestros archivos con ellos; este compartir se define en el modelo de datos *ShareFile*, el cual contiene la permisología con la que definimos el acceso que el *usuario amigo* puede tener sobre el archivo. El sistema permite que el usuario dueño en cualquier momento pueda cambiar la permisología que poseen cada uno de sus archivos y carpetas tanto para los *usuarios amigos*, como los *usuarios no amigos*.

El sistema de simulación web cuenta con una gama de funcionalidades para el control de usuarios, como: Registro de usuarios, Activación y desactivación de usuarios, Edición y eliminación de usuarios, Control de espacio en disco asignado para cada usuario, así mismo los usuarios pueden ingresar al sistema libremente y realizar las funciones que su rol dentro del sistema les permite. El control del modelo de simulación esta fuertemente relacionado con el manejo de archivos y carpetas dentro del sistema, debido a que se le permite al usuario agregar, modificar y eliminar modelos de simulación. Para ello es necesario un sistema administrador de archivos y carpetas, que junto a las permisologías asociadas, permita un optimo manejo. Al igual se cuenta con una interfaz de usuario amigable y portable para ejecutar esas tareas de administración. Siendo los elementos explicados en este

capítulo la aplicación Web que funciona como coordinador del sistema, control de usuarios y roles y la Administración de archivos que permite llevar el control de acceso, creación y modificación de los archivos y carpetas. En el capítulo 4 se definirá a detalle la integración con GALATEA.

Capítulo 4

Integración de GALATEA

Debido a que GALATEA esta desarrollado en Java y nuestro sistema web esta escrito en Python, necesitamos un mecanismo de comunicación para ambos lenguajes. Para esto se desarrolló dos mecanismos de comunicación como lo son: (1) Subprocesos y (2) Sockets. El primero de ellos lo implementamos realizando invocación a un subproceso o proceso externo, enviándole los parámetros necesarios para su ejecución, este enfoque es el actualmente implementado en nuestro sistema web y esta bien especificado en la sección 4.1. La segunda implementación es una comunicación entre procesos vía sockets, el cual atiende las peticiones en un puerto de comunicación específico, lo cual nos permite comunicar nuestro sistema Python con el motor de simulación escrito en Java, como se especifica en la sección 4.2.

4.1 Subprocesos

El módulo de subproceso en Python permite ejecutar nuevos procesos, tomando en cuenta las operaciones de entrada/salida/error y así obtener la información de retorno. Lo utilizan para ejecutar un comando shell o un archivo ejecutable en Python, (Astrand, 2003).

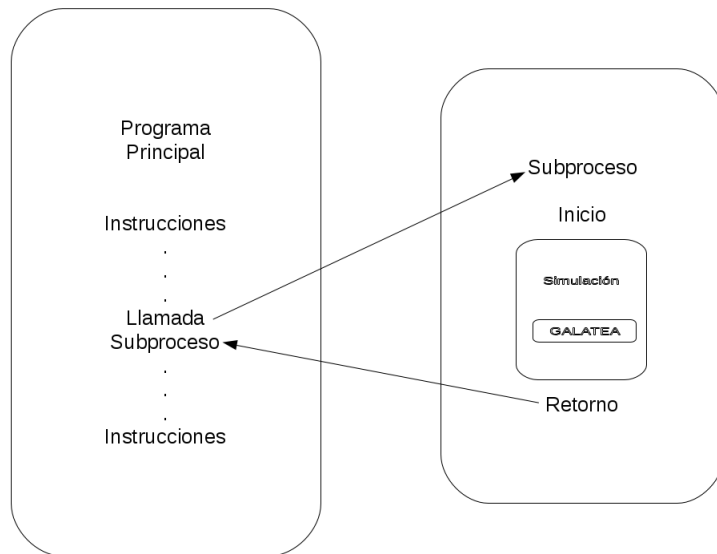


Figura 4.1: Ejecución de un Subproceso.

El proceso de GALATEA para ejecutar simular definido en tres pasos:

- Traducir el lenguaje GALATEA: este primer paso consiste en traducir el modelo escrito en lenguaje GALATEA, a lenguaje Java y Prolog.
- Compilar el modelo: en este paso compilamos el código Java de la simulación (Java a bytecode).
- Ejecutar la simulación: Finalmente ejecutamos la simulación con sus diferentes parámetros de entrada. Es decir, activamos el modelo de simulación con ciertos parámetros.

Cada uno de estos pasos es ejecutado mediante una llamada a un subproceso dentro de nuestro sistema. El resultado de cada una de los pasos es obtenido como valor de retorno de la misma y es mostrado al usuario.

```

1  #run the program
2  #java -cp ../.././galatea.jar SimpleTeller
3  def execute_java(file_id, arguments):
4      #get file
5      file = File.objects.get(id=file_id)
6      #first compile
7      compile_java(file_id)
8      folder = file.folder
9      galatea_code = GALATEA + "galatea.jar "
10     code = "cd "+file.folder.path+" && java -cp .:" + galatea_code +
           os.path.splitext(file.name)[0] + " " + arguments

```

```

11     value = subprocess.check_output([code], shell=True)
12     return value

```

4.2 Sockets

Socket designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada (IBM, 2015). Los sockets de Internet constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto (Yadav, 2007).

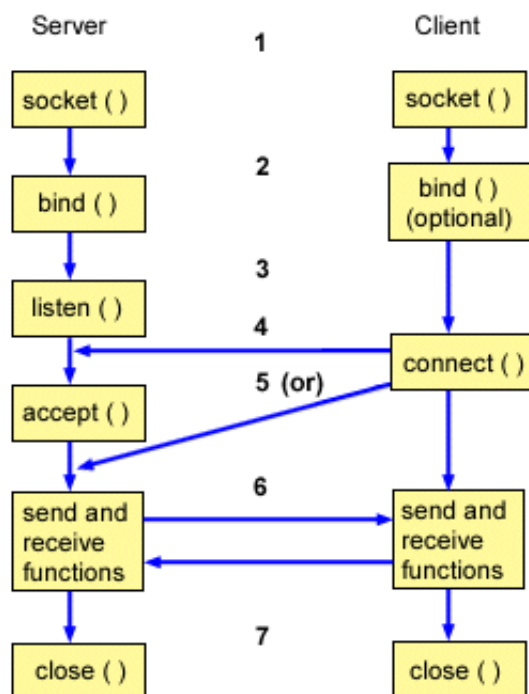


Figura 4.2: Típico flujo usando sockets.

Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos: Primero que un programa sea capaz de localizar al otro y segundo que ambos programas sean capaces de intercambiarse cualquier dato relevante entre ellos. Para ello son necesarios los dos recursos que originan el concepto de socket: un par de direcciones del protocolo de red (dirección IP, si se utiliza el protocolo TCP/IP), que identifican la computadora de origen y la remota, y un par de números de puerto, que identifican a un programa dentro de cada computadora (Yadav, 2007). Los sockets

permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente". El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor" (IBM, 2015).

La implementación de la integración con GALATEA mediante sockets como se observa en la figura 4.3, consiste en lo siguiente:

- **El programa cliente:** es un programa generado y ejecutado por nuestro sistema web, este programa está escrito en Python.
- **El programa servidor:** este programa está escrito en Java para su completa integración con GALATEA, es iniciado desde nuestro sistema web y se crea una comunicación con nuestro programa cliente.
- **Protocolo de comunicación:** este es el corazón de nuestra integración mediante este método. Es importante poder definir un protocolo de comunicación entre nuestro programa cliente y servidor, debido que GALATEA permite diferentes opciones de simulación y controles. Este protocolo de comunicación debe definir todas las posibles acciones a realizar para la simulación como: Iniciar, pausar, terminar, datos de entrada, datos de salida, entre muchos otros.

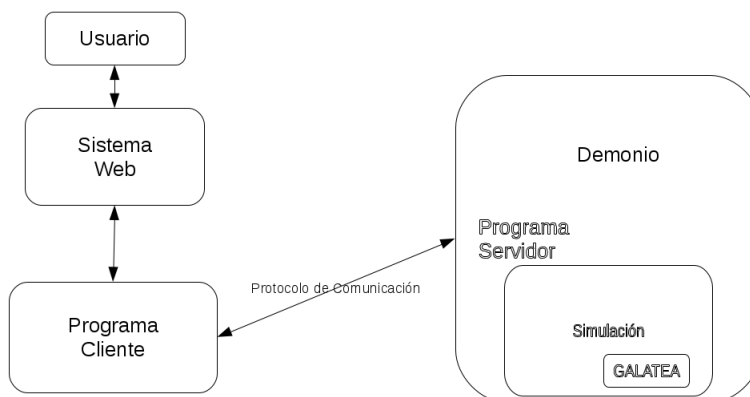


Figura 4.3: Estructura Integración mediante Sockets.

Toda nuestra implementación de sockets se hizo utilizando el protocolo de comunicación TCP. Con el uso de protocolo TCP, las aplicaciones pueden comunicarse en forma segura (gracias al de acuse de recibo -ACK- del protocolo TCP) independientemente de las capas inferiores. Esto significa que los

routers sólo tiene que enviar los datos en forma de datagrama, sin preocuparse con el monitoreo de datos porque esta función la cumple la capa de transporte (Yadav, 2007).

4.2.1 Programa Cliente

El programa cliente está escrito en Python y es llamado por nuestra aplicación web. Su principal función es servir de mecanismo de comunicación entre el proceso de simulación y el usuario. Este programa permite recibir las distintas peticiones del usuario, identificarlas y comunicárselas al proceso de simulación, al igual recibir la comunicación del programa servidor (la simulación) y comunicárselas al usuario. Hay que notar que pueden desarrollarse muchas variantes del programa cliente, pero para uso de este trabajo se desarrolló un cliente que mantuviera una comunicación simple con el programa servidor.

El programa cliente lo podemos estructurar en tres bloques principales:

- **Iniciar Conexión:** Lo principal es crear una comunicación directa con el programa servidor, por lo que, verificamos la conexión y nos conectamos al socket específico. Garantizar la conexión es importante para poder mantener un control continuo de la simulación.
- **Envío de instrucciones:** utilizando el protocolo de comunicación definido en la sección 4.2.3, enviamos al programa servidor una serie de instrucciones de prueba para verificar el correcto funcionamiento de la simulación.
- **Finalizar conexión:** luego de ejecutar toda la simulación y obtener los valores deseados, finalizamos la conexión con el programa servidor.

4.2.2 Programa Servidor

El programa servidor está escrito en Java y es ejecutado por nuestra aplicación web. Justo con su ejecución el programa define un puerto de comunicación específico, que a su vez permite al programa cliente la comunicación. Su función principal es recibir las distintas peticiones del programa cliente, transmitir las al proceso de simulación y devolver los resultados al programa cliente. Este proceso debe llevar control integral de la simulación, así como atender las posibles eventualidades que puedan ocurrir durante la simulación. Este programa mantiene comunicación continua con GALATEA, ya que encapsula en su interior a GALATEA y al modelo que está siendo ejecutado.

Este programa está definido por tres hilos de ejecución como observamos en la figura 4.4:

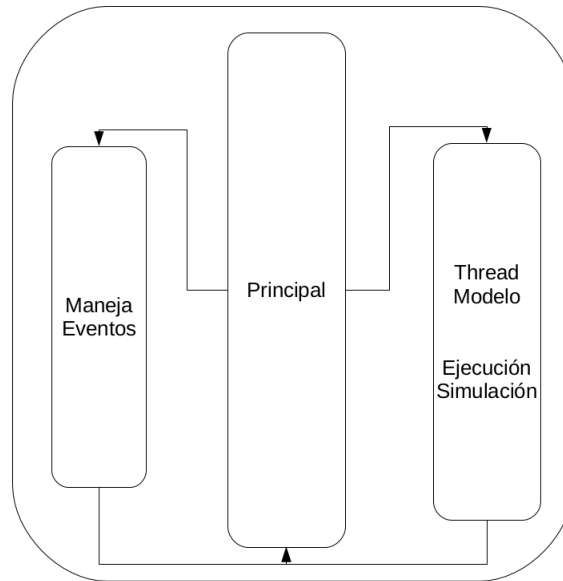


Figura 4.4: Estructura del Programa Servidor.

Manejador de Eventos

Debido a, que en los nuevos desarrollos de GALATEA para cuando este trabajo se realizó, se esta desarrollando el manejo de eventos dentro de GALATEA, los cuales son disparados cuando situaciones particulares dentro de la simulación ocurren. En nuestra integración se tomó en cuenta y se implementó ese manejo de eventos.

La función de el manejador de eventos es capturar todos los eventos que son creados dentro de la simulación, procesarlos y enviarlos nuevamente al programa cliente, para su posterior observación. Los eventos y su procesamiento ocurren en tiempo real, es decir, a penas el evento es creado se captura y se procesa.

```

1 public class ModelEventListener implements ModelListener
2 {
3     Socket _connected;
4     PrintWriter outToClient;
5     public ModelEventListener(Socket connected)
6     {
7         try
8         {
9             _connected = connected;
10            //Buffer out for client data
11            outToClient = new PrintWriter( connected.getOutputStream(),true);

```

```

12     }
13     catch (IOException e) {
14         System.out.println(e);
15         System.exit(1);
16     }
17 }
18 public void modelReceived( ModelEvent event )
19 {
20     System.out.println ("Event " + event.type() + " description: " + event.description() );
21     String data = "Event " + event.type() + " description: " + event.description() ;
22     outToClient.println(data);
23 }
24 }

```

Thread de Simulación

Éste módulo es el modelo de simulación implementado como un Thread de Java (Oracle, 2016a). Este thread encapsula toda la información del modelo de simulación (variables y métodos), al igual todos los métodos de comunicación con GALATEA.

Su función es procesar la simulación y disparar los eventos que serán capturados por el manejador de eventos.

```

1 class ModelThread extends Thread
2 {
3     // Atributes
4     public static String _args[];
5     public static double InArrTime;
6     public static double MeSerTime;
7     public static double DeSerTime;
8     // Nodes
9     static Gate gate = new Gate();
10    static Teller teller = new Teller();
11    static Exit exit = new Exit();
12    static ShowLength showLength = new ShowLength();
13    private ArrayList _listeners = new ArrayList();
14    public int valores;
15    public int jodedera;
16    public void setArg(String args[])
17    {
18        _args = args;
19    }

```

```
20     public void run()
21     {
22         try
23         {
24             main(_args);
25             valores = 40;
26             prueba = 100;
27             System.out.println ("Start Running Thread");
28             this._fireModelEvent(1,"Start Running Thread");
29             for(int i = 0; i < 200; ++i)
30             {
31                 System.out.println ("Running Thread ---> Time " + i);
32                 this._fireModelEvent(2,"Running Thread ---> Time " + i);
33                 Thread.sleep(100);
34             }
35             System.out.println ("Finish Running Thread");
36             this._fireModelEvent(1,"Finish Running Thread");
37         }
38         catch(InterruptedException ex)
39         {
40             Thread.currentThread().interrupt();
41         }
42     }
43     public synchronized void addModelListener( ModelListener l )
44     {
45         _listeners.add( l );
46     }
47     public synchronized void removeModelListener( ModelListener l )
48     {
49         _listeners.remove( l );
50     }
51     private synchronized void _fireModelEvent( int type, String description )
52     {
53         ModelEvent mood = new ModelEvent( this, type, description );
54         Iterator listeners = _listeners.iterator();
55         while( listeners.hasNext() )
56         {
57             ( (ModelListener) listeners.next() ).modelReceived( mood );
58         }
59     }
60     public static void setExperiment(String[] args)
```

```

61  {
62      Glider.setTsim(Galatea.doubleArg(args, "tSim", "10", "Tiempo de Simulacion"));
63      InArrTime = Galatea.doubleArg(args, "InArrTime", "4.0", "Mean Interarrival Time");
64      MeSerTime = Galatea.doubleArg(args, "MeSerTime", "3.50", "Mean service time");
65      DeSerTime = Galatea.doubleArg(args, "DeSerTime", "0.80", "Deviation of Mean Service
        Time");
66      Galatea.checkArgs(args);
67      Glider.addPath(Galatea.getExperDir());
68  }
69  public static void main(String args[])
70  {
71      //ModelThread obj = new ModelThread();
72      //obj.start();
73      Glider.setTitle("Simple Teller");
74      Glider.act(0, gate);
75      setExperiment(args);
76      Glider.process();
77  }
78  }

```

Cuerpo Principal

Es el encargado de controlar todos los aspectos principales del programa servidor. Primero inicia el programa en un puerto específico, e inicializa el thread de simulación y el manejador de eventos.

```

1  //init de the server on port number
2  ServerSocket Server = new ServerSocket (portNumber);
3  System.out.println ("TCPServer Waiting for client on port " + portNumber);
4  //wait for client connected
5  Socket connected = Server.accept();
6  System.out.println( " THE CLIENT " + " " + connected.getInetAddress() +" :
        "+connected.getPort()+" IS CONNECTED ");
7  //Buffer get info to send
8  BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
9  //Buffer in for client data
10  BufferedReader inFromClient = new BufferedReader(new InputStreamReader
        (connected.getInputStream()));
11  //Buffer out for client data
12  PrintWriter outToClient = new PrintWriter( connected.getOutputStream(),true);
13  //create thread
14  ModelThread obj = new ModelThread();

```



```
15 //create ModellListener
16 ModellListener modListener = new ModelEventListener(connected);
17 //ModelThread assing listeners
18 obj.addModellListener(modListener);
```

Se encarga de recibir todas las solicitudes del programa cliente, mediante el protocolo de comunicación definido en la sección 4.2.3, y ejecutar esas peticiones sobre el thread de simulación, al igual controlar todo el manejo de excepciones que puedan ocurrir.

4.2.3 Protocolo de Comunicación

La comunicación es la herramienta mas importante de esta implementación, por ende definir un buen protocolo que permita controlar las distintas opciones y acciones a ser ejecutadas en nuestra simulación nos garantiza una buena integración.

Como primera aproximación funcional, hemos definido un protocolo de control del simulador a partir de las funcionalidades que ofrece la plataforma Java para el control de subprocesos (o hebras, como son llamadas en esa plataforma). Esto permitirá que, sin importar cuál sea la solución, subproceso o sockets, del lado del simulador se tenga acceso a toda la funcionalidad de la plataforma base del simulador (máquina virtual Java, JVM, en este caso y, eventualmente la máquina Prolog, que actualmente es controlada desde la JVM).

A estas funcionalidades de control de hebras se agregarían, en casos particulares, otras funcionalidades específicas, como las que son necesarias para el control interactivo de una animación a partir de una simulación. Otros clientes seran necesarios en esos casos, desde luego.

El API Java para el manejo de Hebras (Threads) (Oracle, 2016a), define un conjunto de métodos para el control de hebras que han venido evolucionando a lo largo de los años y que se combina con los métodos de sincronización y comunicación entre procesos Java. Métodos como stop(), suspend() y resume() han sido retirados (Oracle, 2016b) al ser considerados inherentemente inseguros a bajo nivel. Basándonos en esa experiencia y que este protocolo opera a un nivel superior con menor riesgo de seguridad, hemos propuesto los siguientes comandos de control de la simulación:

- start: inicia la simulación(el simulador debe estar en estado de espera).
- sleep(ms): suspende la simulación (por una cantidad de ms).
- stop: detiene definitivamente la simulación (si el simulador esta programado para detección temprana dela orden esto puede ocurrir en cualquier momento).
- yield: le indica al procesador que debe bajar su prioridad de uso de procesador (corresponde al yield() en Java).
- pause: suspende la ejecución del simulador temporalmente. El simulador queda a la espera de otras órdenes que no impliquen terminar la simulación.

- `set var = value`: es un comando para cambiar el valor de parámetro de simulación llamado *var* asignándole el valor *value* que será usado cuando la simulación se reinicie.
- `get var`: es un comando para recuperar el valor del parámetro *var* que será impreso por la misma vía de comunicación y leído por el cliente como una línea de texto.

Capítulo 5

Pruebas

Ya que la aplicación web está pensada en atender a una cantidad arbitraria de usuarios, deseamos conocer el número máximo de usuarios concurrentes a los que se le puede dar servicio con el hardware disponible. Llevamos a cabo pruebas de carga en las cuales se simulará una ráfaga de peticiones variables para determinar la cantidad de peticiones concurrentes que el servidor puede manejar antes de que ocurra una degradación del servicio o el servidor sea incapaz de responder mas peticiones.

Consideramos que ocurre una degradación del servicio cuando el tiempo de espera del usuario aumenta mas allá de un umbral máximo, según (Nielsen, 1994), existen tres umbrales importantes a la hora de evaluar el tiempo de respuesta de una aplicación:

- **0.1 segundo:** Límite en el cual un usuario siente que está manipulando los objetos desde la interfaz de usuario.
- **1 segundo:** Límite en el cual un usuario siente que está navegando libremente.
- **10 segundos:** Límite en el cual se pierde la atención del usuario, es buena idea proveer barras de cargas y medios para cancelar la operación cuando se trata de operaciones de larga duración.

Para las siguientes pruebas consideramos las siguientes métricas de rendimiento de la aplicación web:

- Tiempo de respuesta promedio.
- Tiempo de respuesta mediana.
- Tiempo de respuesta mínimo.
- Tiempo de respuesta máximo.
- Porcentaje de error.
- Rendimiento en términos de peticiones por segundo durante la duración de la prueba.
- Rendimiento en términos de bytes por segundo durante la duración de la prueba.

Consideramos una prueba como exitosa cuando el porcentaje de error sea igual a 0 y el tiempo máximo de espera de una petición no supere los 10,000ms, tomando en cuenta los umbrales definidos por (Nielsen, 1994). Se considerará óptima aquella prueba que además de cumplir con ser exitosa, tenga el mayor rendimiento en términos de peticiones por segundo (Req/s) entre todas las pruebas realizadas.

La herramienta elegida para ejecutar las pruebas fue (JMeter, 2014), esta herramienta permite simular casos de uso específicos de una aplicación web como ráfagas de peticiones que pueden ser configuradas para seguir cualquier patrón dado. En nuestro caso, Jmeter se usará para generar una ráfaga de peticiones con un intervalo constante entre cada petición durante un tiempo de un minuto. Jmeter provee un componente llamado *Summary Report* que totaliza y promedia los resultados obtenidos de todas las peticiones enviadas, del que podemos extraer todas las métricas de interés.

5.1 Metodología de las pruebas

Para determinar la carga máxima que la aplicación web es capaz de soportar en términos de las peticiones por minuto, simularemos una ráfaga de peticiones en un intervalo de tiempo de un minuto, variando el numero de peticiones hasta obtener un valor óptimo, la metodología a seguir es la siguiente:

1. Se comenzará ejecutando una primera prueba con un número prudencialmente bajo de peticiones durante 60 segundos, éste número debe preferiblemente ser múltiplo de dos.
2. Se varía el número de peticiones enviadas en la ráfaga multiplicando por dos cada vez que una prueba es exitosa, así si la primera prueba se hace con 16 peticiones. la siguiente tendrá 32, 64, 128 y así sucesivamente.
3. Tras la primera prueba fallida, se calculará el número de peticiones de la siguiente prueba (P_{sig}) como el punto medio entre la cantidad de peticiones de la prueba previa (P_{prev}) y la prueba actual (P_{act}).

$$P_{sig} = \begin{cases} P_{prev} + |P_{act} - P_{prev}|/2 \\ P_{prev} - |P_{act} - P_{prev}|/2 \end{cases} \quad (5.1)$$

4. El paso 3 se repite hasta no se obtenga mayor precisión con pruebas sucesivas.

5.2 Condiciones de las pruebas

Todas las pruebas se ejecutaron sobre una red local entre dos máquinas conectadas en un mismo enrutador: una de ellas ejecuta el servidor web, y la otra ejecuta las pruebas simulando las peticiones y totalizando los resultados observados; la latencia de la red se considera menor a 5ms. Las características del servidor web se pueden observar en la tabla 5.1:

Procesador	Memoria	Sistema Operativo
Intel Pentium 4 CPU @3.10GHz	1 GB RAM	Debian 8

Tabla 5.1: Especificaciones del servidor web.

5.3 Resultados obtenidos

1. **Archivos estáticos con Apache:** Definir.
2. **Páginas dinámicas con Django sobre Apache:** Definir.
3. **Páginas dinámicas con Django y archivos estáticos sobre el mismo servidor Apache:** Definir.

5.4 Análisis de los resultados

Definir

Capítulo 6

Conclusiones

Durante el desarrollo de este trabajo, se construyó un sistema web de simulación cuya base principal es GALATEA. Al igual se implementó un sistema de administración de archivos y carpetas, el cual facilita al usuario el manejo de las simulaciones. La interacción de usuarios dentro del sistema es un elemento importante, por lo que se permitió a los usuarios compartir proyectos de simulación (archivos y carpetas), lo cual permite a varios usuarios colaborar entre sí para desarrollar mejores simulaciones y validar simulaciones ya realizadas. Se integraron las nuevas tecnologías y herramientas de software como lo son: nuevos lenguajes de programación adaptados a la web usando Python mediante su framework web Django; herramientas para el control de tareas como Trello¹; repositorios para almacenar los proyectos como Github² y manejadores de control de versiones como Git³ que junto con su integración con Github se convierte en una herramienta de desarrollo muy poderosa para cualquier proyecto de software; al igual el manejo y control de servidores web como Nginx⁴ y Apache⁵, siempre sin olvidar el manejo de Base de datos MySQL⁶ y PostgreSQL⁷.

El objetivo principal de este proyecto era desarrollar una integración entre GALATEA como software de simulación y el sistema web. Se desarrollaron y presentaron dos enfoques de integración que permitieron al sistema web comunicarse con el motor de simulación GALATEA, simular y mostrar los resultados al usuario. Se planteó el desarrollo de una plataforma web para que, los usuarios y usuarias del Centro de Simulación y Modelado (CESIMO) pudieran ejecutar los modelos de simulación en el simulador GALATEA de manera fácil y rápida, sin tener que pasar dificultades al momento de configurar y activar todas sus funcionalidades de simulador y así enfocarse únicamente en sus modelos de simulación y en los resultados obtenidos de los mismos.

El código completo de sistema web de simulación se encuentra en el siguiente repositorio publico

¹Trello <https://trello.com/>

²Github <https://github.com/>

³Git <https://git-scm.com/>

⁴Nginx <https://www.nginx.com/>

⁵Apache <https://www.apache.org/>

⁶MySQL <https://www.mysql.com/>

⁷Postgresql <https://www.postgresql.org/>

de Github <https://github.com/velasquezerik/django-roles-test>. Al igual se utilizó este repositorio <https://github.com/velasquezerik/tesis-doc>, para almacenar todo éste documento. Para permitir que la comunidad de usuarios y usuarias aporten y desarrollen nuevas mejoras en el sistema web, se optó por utilizar la licencia de software GNU AFFERO GENERAL PUBLIC LICENSE⁸. La Licencia Pública General Affero esta basada en la GPL de GNU, pero contiene una cláusula adicional que permite a los usuarios interactuar con el programa licenciado a través de una red para recibir el código fuente de ese programa.

Se espera que con este trabajo el universo de usuarios y usuarias de simulación crezca, ya no solo a personas con conocimientos avanzados en el tema de simulación, conocimientos de programación y de sistemas operativos, sino a la comunidad en general. Permitiéndole a estudiantes de cursos iniciales, y estudiantes de otras universidades, al igual cualquier persona que quiera colaborar en los procesos y modelos de simulación, trabajar, analizar y compartir los resultados obtenidos desde cualquier parte del mundo.

Los logros mas importantes desde el punto de vista computacional se nombran a continuación:

- Se desarrolló una aplicación web que permite el control de usuarios y usuarias, que pueden desde cualquier parte del mundo con una conexión a internet, ingresar en el sistema y desarrollar sus modelos de simulación.
- Se creó un sistema administrador de archivos y carpetas, al igual la correcta administración de espacio en disco que puede utilizar cada usuario y usuaria.
- Se implementó un sistema de permisos para cada archivo y carpeta, lo que permite a los usuarios colaborar entre ellos en diferente proyectos y modelos de simulación.
- Se integró el simulador de eventos discretos GALATEA a la aplicación web, lo cual se hizo mediante dos enfoques: Subprocesos y Sockets, dejando abierta la posibilidad a otros desarrollos específicos en el futuro.

Además del diseño y desarrollo del sistema, también se ejecutaron pruebas para determinar el rendimiento, estabilidad y escalabilidad de la aplicación web así como las funcionalidades básicas de los modelos de simulación existentes en GALATEA.

6.1 Recomendaciones

El presente proyecto permite un universo de opciones que se pueden continuar desarrollando, pero a continuación presentaré los trabajos más importantes a mi parecer que pueden desarrollarse como una continuación a este trabajo:

- Integrar un sistema de educación a distancia para simulación.

⁸Affero <https://www.gnu.org/licenses/agpl-3.0.en.html>

- Desarrollar nuevas mejoras en la api de integración de GALATEA para no solo garantizar una comunicación con el sistema web, sino con otros sistemas.
- Mejorar el sistema de comunicación de usuarios dentro del sistema para de alguna manera desarrollar una red social de simulación.
- Definir e implementar un sistema de control de eventos en tiempo real para GALATEA, lo cual permita al usuario interactuar y obtener los resultados en tiempo real con el proceso de simulación.

La colaboración de una comunidad de desarrolladores siempre ayuda a depurar y mejorar el software, por lo que se invita a la comunidad ULA a continuar el desarrollo de éste proyecto. Es un proyecto libre que puede ayudar no solo a la comunidad universitaria en general, sino a la comunidad educativa, científica y social de nuestro territorio.

Bibliografía

- Astrand, P. (2003). subprocess - new process module.
- Banks, J., Carson, J. S., Nelson, B. L., y Nicol, D. M. (2010). *Discrete-Event System Simulation*. Pearson, 5 edition.
- Boehm, B. W. (1988). A spiral model of software development and enhancement.
- Dávila, J. (2011). *Lógica Práctica y Aprendizaje Computacional*. Editorial Académica Española. ISBN-13: 978-3-8465-6233-8 ISBN-10: 3846562335 EAN: 9783846562338.
- Dávila, J. (1999). Openlog: A logic programming language based on abduction, en ppdp'99. *Principles and Practice of Declarative Programming, Lecture Notes in Computer Science*, (1702/1999):278–293.
- Dávila, J. (2003). Actilog: An agent activation language. *Lecture Notes in Computer Science*, (2562/2003):194–207.
- Fuentes, J. M. (2009). *Manual de AJAX*. 2 edition.
- Fujimoto, R. M. (2015). The high level architecture.
- Fundation, D. S. (2015). *The web framework for perfectionists with deadlines*. 5 edition.
- Gilbert, N. y Troitzsch, K. G. (2005). *SIMULATION FOR THE SOCIAL SCIENTIST*. McGraw - Hill Education, 2 edition.
- IBM (2015). Socket programming.
- JMeter (2014). Apache jmeter.
- Krasner, G. E. y Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Technical report, ParcPlace Systems, Inc., Mountain View.
- Machuca, C. A. M. (2011). Estado del arte: Servicios web. Technical report, Universidad Nacional de Colombia, Bogotá, Colombia.
- Marcano, G. (2015). Desarrollo de un servicio web para el simulador de eventos discretos galatea. Technical report, Universidad de Los Andes, Merida, Venezuela.

- Microsystem, S. (2009). Distributed application architecture.
- Nielsen, J. (1994). *Usability Engineering*. Elsevier, 2 edition.
- Oracle (2015). Mysql 5.7 :: Reference manual 1.3.1 what is mysql?
- Oracle (2016a). Class thread. <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>.
- Oracle (2016b). Java thread primitive deprecation. <https://docs.oracle.com/javase/7/docs/technotes/guides/concurrent.html>.
- Pantoja, E. B. (2004). El patrón de diseño modelo-vista-controlador (mvc) y su implementación en java swing. Technical report, ParcPlace Systems, Inc.
- PostgreSQL (2016). Postgresql about.
- Pérez, J. A. G. (2016). Diseño e implementación de un sistema de monitoreo de redes orientado a la recolección masiva de datos. Technical report, Universidad de los Andes, Mérida, Venezuela.
- Rengifo, B. (2011). Desarrollo de un servicio web para la modeloteca del sistema nacional de simulación. Technical report, Universidad de Los Andes, Merida, Venezuela.
- Santana, Y. C., Ángel Mateo Pla, M., y Terol, J. P. (2004). Herramienta de simulacion distribuida mediante agentes moviles jade. Technical report, Conferência IADIS Ibero-Americana.
- Uzcátegui, M., Dávila, J., y Tucci, K. (2011). Galatea: una historia de modelado y simulación. *Revista Ciencia e Ingeniería*, (1316-7081):85–94.
- Yadav, R. (2007). Client / server programming with tcp/ip sockets.

Apéndice A

Historias de usuario de la aplicación web

Historia	Control de Usuarios en el sistema
Id	UH-001
Descripción	Los usuarios y usuarias del sistema web de simulación deben hacer correcto uso del mismo, por lo que se les debe permitir ingresar al sistema con sus credenciales, salirse del sistema, al igual editar su perfil con los datos de su preferencia (Imagen, contraseña, entre otros). Solo el administrador del sistema tiene la permisología para crear usuarios, editar la información de manejo de espacio de archivos y permitirles el acceso.
Cómo probarlo	El administrador del sistema crea un usuario o usuaria, le envía un correo con la notificación. EL usuario visita el sistema, ingresa sus datos y accede al mismo. EL usuario o usuaria modifica la información de su perfil y se observan los cambios de manera permanente en el sistema. El administrador modifica el espacio asignado en disco para un usuario o usuaria en particular, ese usuario o usuaria llena su espacio con información y una vez esto ocurra se observa que no pueda agregar mas archivos en su cuenta.

Tabla A.1: Historia de Usuario - Control de Usuarios en el sistema.

Historia	Control de Archivos y carpetas en el sistema
Id	UH-002
Descripción	Los usuarios y usuarias del sistema web de simulación deben controlar los archivos y carpetas que se encuentren dentro del sistema, para desarrollar un mejor modelo de simulación. Para ello, el sistema debe permitir la creación, modificación y eliminación de archivos y carpetas en la sesión de cada usuario o usuaria. Los archivos y carpetas deben ser compartidos entre los usuarios y usuarias.
Cómo probarlo	El usuario o usuaria crea, modifica y elimina archivos y carpetas usando el sistema de simulación web, el administrador del sistema verifica la creación dentro del sistema web y verifica su correcta creación, modificación o eliminación dentro del sistema de almacenamiento usado por el servidor donde se encuentra funcionando el sistema.

Tabla A.2: Historia de Usuario - Control de Archivos y carpetas en el sistema.

Historia	Integración con GALATEA
Id	UH-003
Descripción	Los usuarios y usuarias del sistema web de simulación deben poder ejecutar los modelos de simulación mediante el sistema web. La ejecución de la simulación debe permitir definir los parámetros iniciales, mostrar las opciones de ayuda, al igual permitirle al usuario o usuaria la visualización de la salida luego de que el modelo de simulación finalice su ejecución.
Cómo probarlo	Ejecutar un modelo de prueba existente en GALATEA (Por ejemplo SimpleTeller), tanto en el sistema web de simulación, como en la ejecución normal de GALATEA. Variar los parámetros de entrada de la simulación y comparar ambas salidas para verificar que sean iguales.

Tabla A.3: Historia de Usuario - Integración con GALATEA.

Apéndice B

Casos de uso de la aplicación web

U-01	Login de Usuario
Precondición	Usuario no autenticado y usuario registrado
Descripción	Usuario no autenticado se autentifica en el sistema web.
Secuencia Normal	Usuario click en login. Ingresa sus credenciales. Dar click en el botón login. Usuario es autenticado por el sistema.
Excepciones	Usuario o contraseña incorrectos.
Postcondición	Inicia en su pantalla de inicio.

Tabla B.1: Caso de uso - Login Usuario.

U-02	Logout de Usuario
Precondición	Usuario autenticado
Descripción	Usuario deja de estar autenticado en el sistema.
Secuencia Normal	Click en el botón logout. Sistema olvida la sesión del usuario.
Excepciones	Error en el control de sesiones.
Postcondición	Usuario no autenticado e inicio en pantalla de Login.

Tabla B.2: Caso de uso - Logout de Usuario.

U-03	Control de Contraseña
Precondición	Usuario autenticado en el sistema.
Descripción	Usuario puede cambiar su contraseña.
Secuencia Normal	Click en cambiar contraseña. Ingresa la nueva contraseña y la repite. Click en botón actualizar. Sistema actualiza contraseña.
Excepciones	Contraseñas no coinciden. Campos vacíos.
Postcondición	Contraseña actualizada.

Tabla B.3: Caso de uso - Control de Contraseña.

U-04	Manejo de Perfil
Precondición	Usuario autenticado.
Descripción	Cambiar la información básica del perfil de usuario.
Secuencia Normal	Click en perfil. Ingresar nuevos datos del perfil. Click en botón actualizar datos. Notificación de datos actualizados.
Excepciones	Campos vacíos.
Postcondición	Datos de perfil actualizados en el sistema.

Tabla B.4: Caso de uso - Manejo de Perfil.

U-05	Manejo de Archivos
Precondición	Usuario autenticado. Espacio en disco libre.
Descripción	Se puede crear, editar, mover y eliminar archivos en el sistema.
Secuencia Normal	Click en crear Archivo. Se ingresa el nombre. Click Aceptar; Click en eliminar Archivo. Se acepta la notificación. Click Aceptar; Click en mover Archivo. Se ingresa el nombre de la carpeta destino. Click Aceptar;
Excepciones	Espacio en disco insuficiente. Nombres no válidos.
Postcondición	Archivo creado, editado, eliminado y movido con éxito.

Tabla B.5: Caso de uso - Manejo de Archivos.

U-06	Manejo de Carpetas
Precondición	Usuario autenticado. Espacio en disco libre.
Descripción	Se puede crear, editar, mover y eliminar carpetas en el sistema.
Secuencia Normal	Click en crear Carpeta. Se ingresa el nombre. Click Aceptar; Click en eliminar Carpeta. Se acepta la notificación. Click Aceptar; Click en mover Archivo. Se ingresa el nombre de la carpeta destino. Click Aceptar;
Excepciones	Espacio en disco insuficiente. Nombres no válidos.
Postcondición	Carpeta creada, editada, eliminada y movida con éxito.

Tabla B.6: Caso de uso - Manejo de Carpetas.

U-07	Control de Simulación
Precondición	Usuario autenticado. Archivo de simulación válido.
Descripción	Se compila y ejecuta un archivo de simulación. La información resultante se observa en el sistema.
Secuencia Normal	Se visualiza un archivo de simulación válido. Click en compilar. Archivo compilado. Se muestra resultado; Se visualiza un archivo de simulación válido. Click en ejecutar. Archivo compilado y ejecutado. Se muestra resultado de la simulación.
Excepciones	Archivo no válido. Error en compilación. Error en Archivo.
Postcondición	Resultado de la compilación y ejecución por pantalla.

Tabla B.7: Caso de uso - Control de Simulación.

U-08	Control de Amigos
Precondición	Usuario autenticado.
Descripción	El usuario puede enviar solicitudes de amistad a otros usuario, al igual puede recibir notificaciones, aceptarlas o rechazarlas.
Secuencia Normal	Click obtener amigos. Selecciona un usuario de la lista. Enviar solicitud. Solicitud enviada; Click en notificaciones de amigos. Aceptar/Negar/Bloquear solicitud.
Excepciones	Solicitud ya enviada. Usuario bloqueado.
Postcondición	Estatus de amigo actualizado.

Tabla B.8: Caso de uso - Control de Amigos.

U-09	Control de Archivos Compartidos
Precondición	Usuario autenticado. Al menos un amigo en el sistema.
Descripción	El usuario puede compartir archivos entre sus amigos.
Secuencia Normal	Pantalla de visualización de archivo. Click en compartir. Selecciona un amigo. Se envía la notificación; Click en notificaciones de archivos. Click en aceptar/negar solicitud de archivo.
Excepciones	No amigos asociados.
Postcondición	Estatus de archivo actualizado.

Tabla B.9: Caso de uso - Control de Archivos Compartidos.

B.1 Usuario Administrador

A-01	Control de Usuarios
Precondición	Usuario Administrador autenticado.
Descripción	Permite registrar nuevos usuarios al sistema.
Secuencia Normal	Click en usuarios. Click registrar nuevo usuario o administrador. Ingresar datos del nuevo usuario. Click en guardar.
Excepciones	Datos repetidos en el sistema.
Postcondición	Usuario registrado en el sistema.

Tabla B.10: Caso de uso - Control de Usuarios.

A-02	Observación de Logs
Precondición	Usuario Administrador autenticado
Descripción	Observación de los logs de carpetas, archivos y usuarios del sistema.
Secuencia Normal	Click en logs. Muestra listado de logs del sistema.
Excepciones	No logs en el sistema
Postcondición	Lista de logs con descripción.

Tabla B.11: Caso de uso - Observación de Logs.

A-03	Control de Espacios en Disco
Precondición	Usuario Administrador autenticado
Descripción	Permite editar el espacio en disco asociado al usuario.
Secuencia Normal	Pantalla de listado de usuarios. Click editar espacio. Modificar espacio. Click actualizar.
Excepciones	Espacio en disco insuficiente.
Postcondición	Modificación de la cantidad de espacio en disco asociado.

Tabla B.12: Caso de uso - Control de Espacios en Disco.

Apéndice C

Casos de uso de la integración con GALATEA

G-01	Traducir Archivos
Precondición	Archivo de simulación válido.
Descripción	Traduce un archivo de simulación en las distintas clases Java necesarias para su compilación y ejecución.
Secuencia Normal	Buscar el archivo de simulación. Ejecuta el comando de traducción.
Excepciones	Archivo de simulación no válido. Errores en la traducción.
Postcondición	Archivos Java necesarios creados.

Tabla C.1: Caso de uso - Traducir Archivos.

G-02	Compilar Archivos
Precondición	Archivo de simulación válido. Traducción de simulación realizada.
Descripción	Compila las diferentes clases Java necesarias para crear sus .Class
Secuencia Normal	Buscar los archivos Java. Crea los .Class de cada archivo Java a ser ejecutado.
Excepciones	Errores en los archivos Java.
Postcondición	Archivos .Class creados.

Tabla C.2: Caso de uso - Compilar Archivos.

G-03	Ejecutar Archivos
Precondición	Archivo de simulación válido. Compilación de simulación realizada.
Descripción	Ejecuta el archivo de simulación en el ambiente de GALATEA y muestra su resultado.
Secuencia Normal	Busca el archivo .Class principal lo enlaza a GALATEA. Ejecuta el programa de simulación.
Excepciones	Errores de lógica de simulación
Postcondición	Simulación realizada.

Tabla C.3: Caso de uso - Ejecutar Archivos.

Código	Mostrar Simulación
Precondición	Archivo de simulación válido. Ejecución de simulación realizada.
Descripción	Muestra por pantalla el resultado de la ejecución de la simulación.
Secuencia Normal	Ejecuta la simulación. Guarda resultados. Envió resultados al sistema de visualización.
Excepciones	Errores en la simulación.
Postcondición	Resultados de la simulación visualizados.

Tabla C.4: Caso de uso - Mostrar Simulación.

Apéndice D

Casos de uso de administración de archivos

Código	Nombre
Precondición	Con diez cañones por banda, viento en popa, a toda vela, no corta el mar, sino vuela un velero bergantín...
Descripción	Con diez cañones por banda, viento en popa, a toda vela, no corta el mar, sino vuela un velero bergantín...
Secuencia Normal	Volverán las oscuras golondrinas, en tu balcón sus nidos a colgar, y otra vez con el ala, a sus cristales jugando llamarán...
Excepciones	Volverán las oscuras golondrinas, en tu balcón sus nidos a colgar, y otra vez con el ala, a sus cristales jugando llamarán...
Postcondición	Volverán las oscuras golondrinas, en tu balcón sus nidos a colgar, y otra vez con el ala, a sus cristales jugando llamarán...

Tabla D.1: Caso de uso - NOMBRE.

C-01	Control de Archivos
Precondición	Espacio en disco suficiente. Permisología de escritura en disco.
Descripción	Permite crear, mover, eliminar y editar archivos en el disco.
Secuencia Normal	Verificar Archivo. Crear/Editar/Mover/Eliminar archivo en el disco. Respuesta de la operación.
Excepciones	Errores en el disco.
Postcondición	Archivo Creado/Editado/Eliminado.

Tabla D.2: Caso de uso - Control de Archivos.

C-02	Control de Carpetas
Precondición	Espacio en disco suficiente. Permisología de escritura en disco.
Descripción	Permite crear, mover, eliminar y editar carpetas en el disco.
Secuencia Normal	Verificar Carpeta. Crear/Editar/Mover/Eliminar carpeta en el disco. Respuesta de la operación.
Excepciones	Errores en el disco.
Postcondición	Carpeta Creado/Editado/Eliminado.

Tabla D.3: Caso de uso - Control de Carpetas.