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

Octubre 2016

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

Br. Erik Velásquez

Proyecto de Grado — Sistemas Computacionales, 39 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 una implementación de gestión de archivos, la cual permite compartir y controlar archivos entre los distintos usuarios y usuarias del CESIMO.

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

Nadie.

Nadie.

Nadie.

Índice

Índice de Tablas

Agradecimientos

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Nadie.

Capítulo 1

Introducción

El siguiente extracto, introduce una descripción de cómo se ha desarrollado el servicio que prestará una aplicación basada en simulación de eventos discretos, que estará fundamentado 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. El propósito es que sirva para proveer un servicio de simulación de eventos discretos al usuario desde cualquier punto. Así también, se pretende analizar conceptos de seguridad mediante el cual, ante una petición al computador (request), ésta sea procesada sólo si ha sido previamente validada.

1.1 Antecedentes

Como trabajos similares a esta propuesta se puede mencionar los realizado por (Uzcátegui et al., 2011) el cual relata la fase de desarrollo, cómo surge y hacia donde se proyecta GALATEA. Implementada como una plataforma libre de código abierto 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. Por otra parte, otro precedente tomado en cuenta ha sido (Rengifo, 2011), el cual trata de una tesis de pre-grado que expone el desarrollo de un servicio web para la Modeloteca del 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. A partir de estos dos 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. 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. Django 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). 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 , autentificación de usuarios, manejo de sesiones entre muchos otros (Fundation, 2015). En su proyecto de grado (Pérez, 2016), 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. Los servicios Web son muy prácticos, pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no debe afectar al otro. 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 (Sistema Distribuido). 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.

1.4 Alcance

Se desea culminar en este proyecto el diseño de una arquitectura de software que permita desarrollar un sistema web para el uso de GALATEA. Se desarrollará un prototipo de esa arquitectura, implementando el sistema web para la ejecución de los modelos de simulación. El prototipo debe cubrir todo lo referente al modelado de los eventos discretos, para el funcionamiento adecuado del sistema. Adicionalmente, la arquitectura del mismo debe permitirles 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 el método SCRUM de desarrollo de software. El mismo "es una metodología de desarrollo muy simple, que requiere trabajo duro, porque la gestión no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto". Entre sus características se encuentra que:

1. Es un método de desarrollo de carácter adaptable.
2. Es orientado a las personas, antes que a los procesos.
3. Emplea desarrollo ágil, interactivo e incremental.

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 estudió y resolución del mismo.

Capítulo II. Marco teórico.

En ese 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.

Capitulo 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.

Capitulo IV. Integración de GALATEA.

Explicar aquí.

Capítulo V. Pruebas.

Explicar aquí.

Capítulo VI. Conclusiones.

Explicar aquí.

Capítulo 2

Marco teórico

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), así como en dirigir para ganar la penetración (profundidad) 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), 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) y Django.

2.1 Simulación Distribuida

Los sistemas de simulación distribuidos son herramientas muy útiles en las labores de investigación porque permiten reducir el tiempo necesario para ejecutar experimentos. 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, es la que permite conducir modelos de simulación a través de múltiples hosts o computadoras separados por redes, 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 central que interaccionan entre sí con frecuencia, por ejemplo, miles de veces por segundo.

2.2 Simulación de Eventos Discretos

La simulación de eventos discretos es la "imitacion" 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 ingles), 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 matríz del sistema Glider, el proyecto paterno 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, segun (Uzcátegui et al., 2011):

> Galatea es una plataforma libre de código abierto 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.

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). El lenguaje base usado para su desarrollo es el Java, bien especificado en (Uzcátegui et al., 2011). Usando como lenguaje base Java, se especifico 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. 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.

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). Se puede definir de manera más sencilla como un conjunto de tecnologías estándares de software para el intercambio de datos entre aplicaciones tales como SOAP, WDSL y UDDI. Estos pueden ser desarrollados en una gran variedad de lenguajes para ser implementados sobre muchos tipos de redes de computadores. El éxito de la interoperabilidad se consigue gracias es la adopción de protocolos y estándares abiertos.

2.5 Arquitectura Orientada a Servicios

Una arquitectura orientada a servicios (SOA, Service Oriented Architecture), es una arquitectura de estilo técnico que proporciona los medios para integrar sistemas dispares y descubrir las funciones de negocio reutilizables. Es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos. Las soluciones SOA han sido creadas para satisfacer los objetivos de negocio las cuales incluyen facilidad y flexibilidad de integración con sistemas legados, alineación directa a los procesos de negocio reduciendo costos de implementación, innovación de servicios a clientes y una adaptación ágil ante cambios incluyendo reacción temprana ante la competitividad (Quiroga, 2011).

La definición de SOA por el W3C es: "Conjunto de componentes, los cuales pueden ser invocados y cuyas descripciones de interfaces pueden ser invocadas y descubiertas".

Gartner: "SOA es una arquitectura de software que comienza con una definición de interfaz y construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamadas a interfaces. Sería más adecuado llamarla "arquitectura orientada a interfaces". SOA es una relación de servicios y consumidores de servicios, ambos suficientemente amplios para representar una función de negocios completa".

SOA no es sólo una arquitectura de servicios visto desde una perspectiva de la tecnología, si no las políticas, prácticas y frameworks con los que se garantiza la forma correcta de que los servicios sean provistos y consumidos. Es importante que si un servicio no va a ser usado por múltiples consumidores, la especificación sea generalizada; los servicios necesitan ser extraídos de la implementación y los desarrolladores de las aplicaciones del consumidor no deberían necesitar saber nada sobre los modelos de bajo nivel y sus reglas (Quiroga, 2011).

2.6 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:

2.6.1 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 2.1 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).

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).

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).

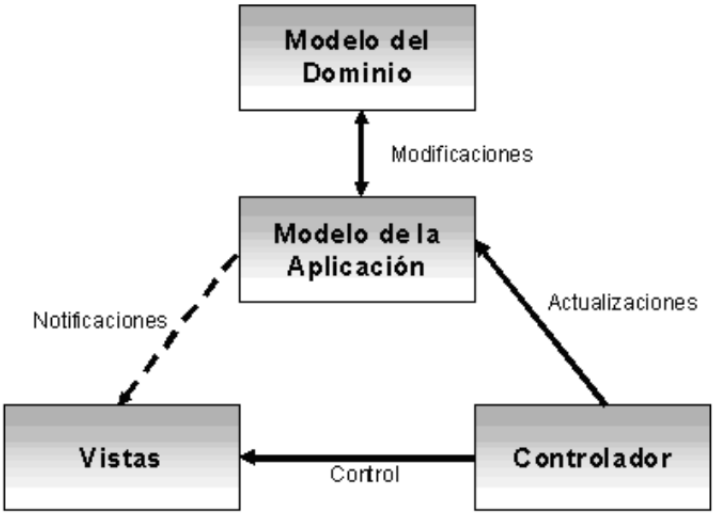

Figura 2.1: Relaciones entre los módulos del patrón MVC.

2.6.2 Django

Django 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). 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 , autentificación de usuarios, manejo de sesiones entre muchos otros (Fundation, 2015).

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 pa 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, se sabe de instancias de Django atendiendo ráfagas de cincuenta mil peticiones por segundo, además es de código abierto, gratuito y cuenta con una extensa comunidad de colaboradores y amplia documentación.

2.6.3 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.

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 ingles) es un método para interacturar 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).

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).

2.6.4 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.

2.6.5 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).

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.

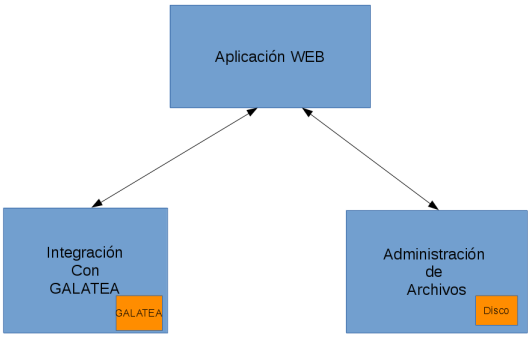

Figura 3.1: Sistema de Simulación Web GALATEA.

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 simulación, mediante

el acceso al sistema de Integración con GALATEA.

Para el módulo de Administración de Archivos se desarrolló una pequeña 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 secciones posteriores se especificará 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

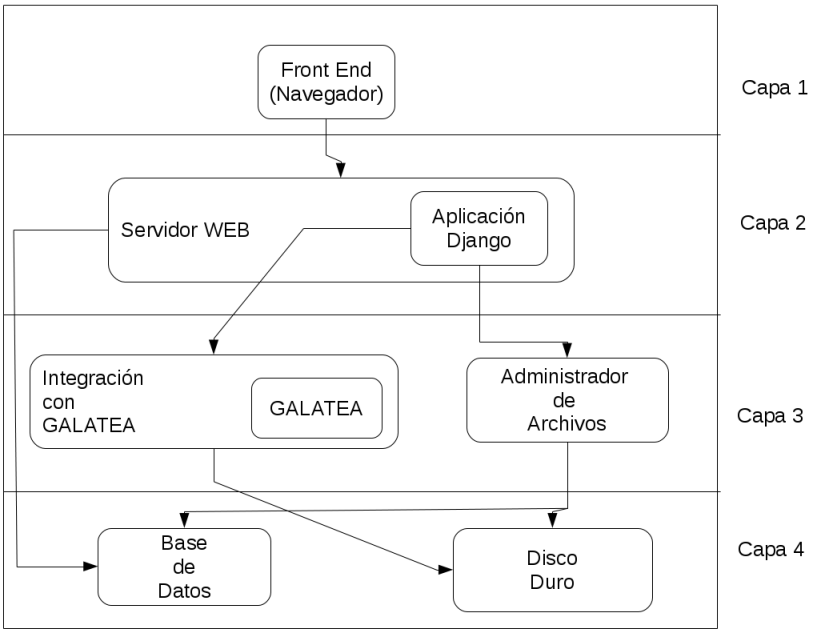

Figura 3.2: Arquitectura de la aplicación web.

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 pude 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, 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 Apache[1], Nginx[2] o Lighttpd[3] 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 de bases de datos o disco duro.

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 de simulaciones que varían desde archivos Java, archivos GALATEA y archivos de texto con información de simulaciones, 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.

[1] Apache https://www.apache.org/
[2] Nginx https://nginx.org/
[3] Lighttpd 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 le 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 esta 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 permisología 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.

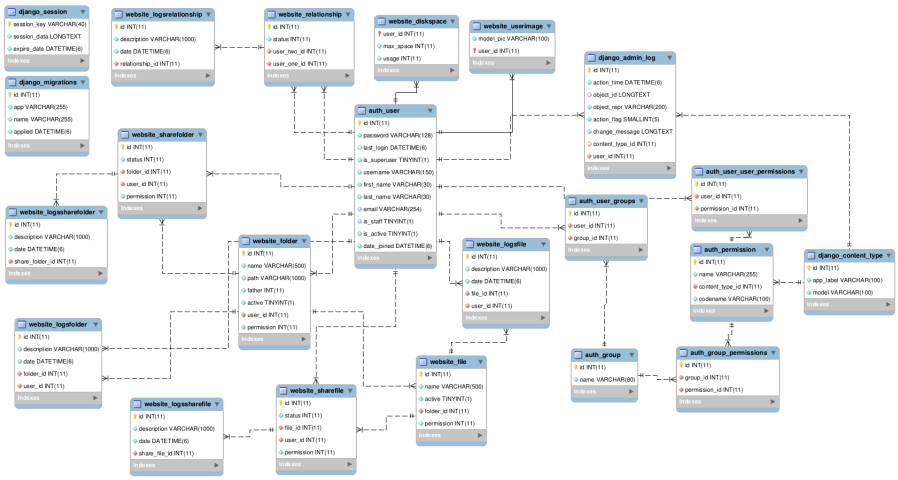

Figura 3.3: Entidades principales de la base de datos.

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

.

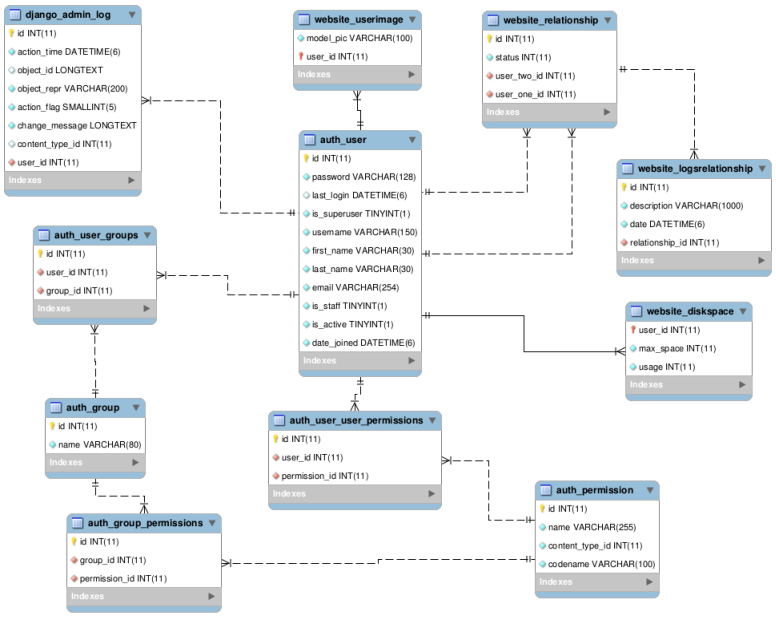

Figura 3.4: Entidades manejo de usuarios.

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

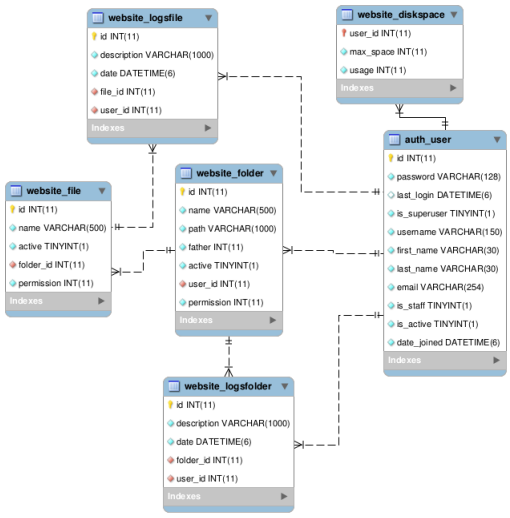

Figura 3.5: Entidades manejo archivos y carpetas.

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

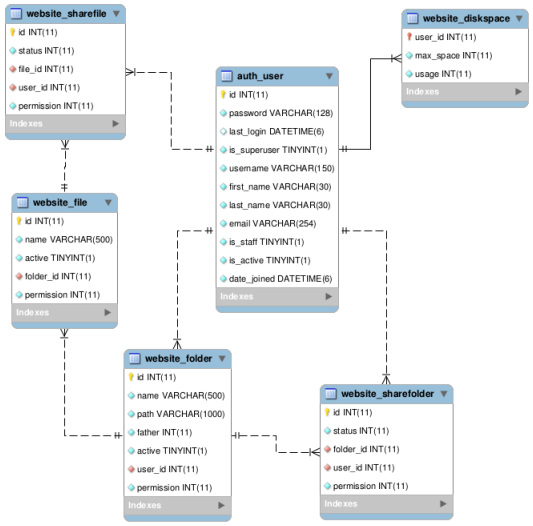

Figura 3.6: Entidades manejo archivos y carpetas compartidos.

Entidades

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

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

Tabla 3.2: Entidades relacionadas al manejo de usuarios.

Nombre de la entidad	Función
File	Almacena la información básica de archivo y su permisología de acceso
Folder	Información básica de una carpeta junto a su permisología de acceso
DiskSpace	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
ShareFolder	Llevar control de la carpeta compartida
ShareFile	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 el tope de 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 capetas de nuestra carpeta de usuario raíz.

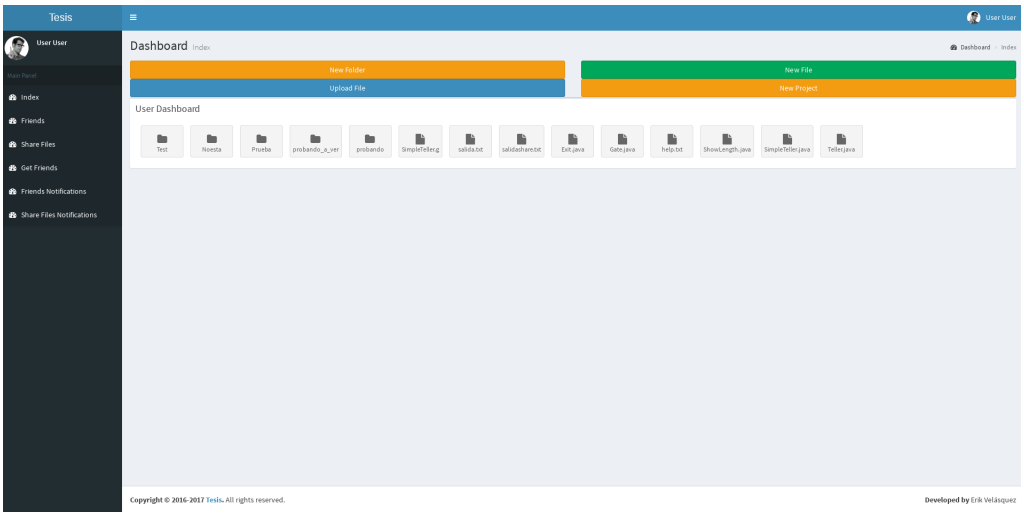

Figura 3.7: Pantalla del Dashboard (Inicio).

- **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.

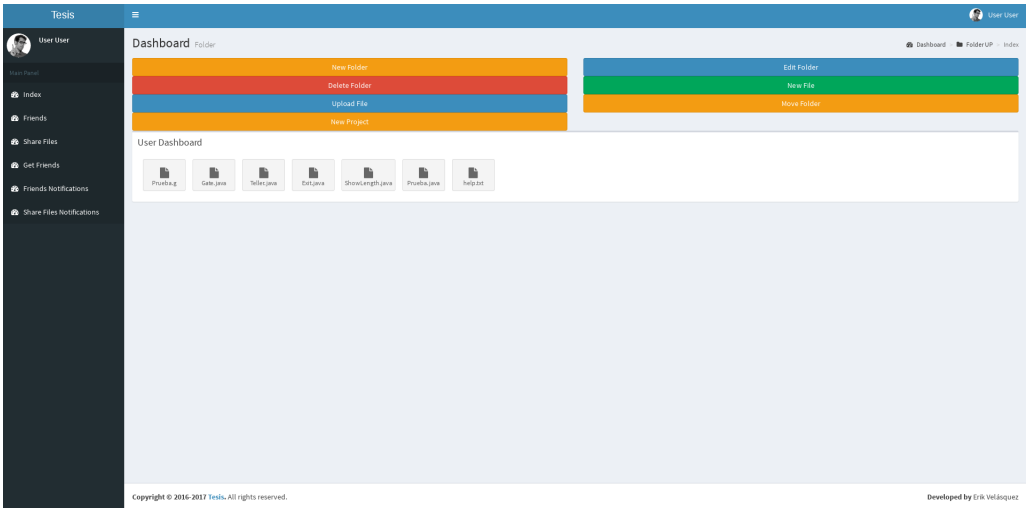

Figura 3.8: Pantalla Observación de Carpeta.

- **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.

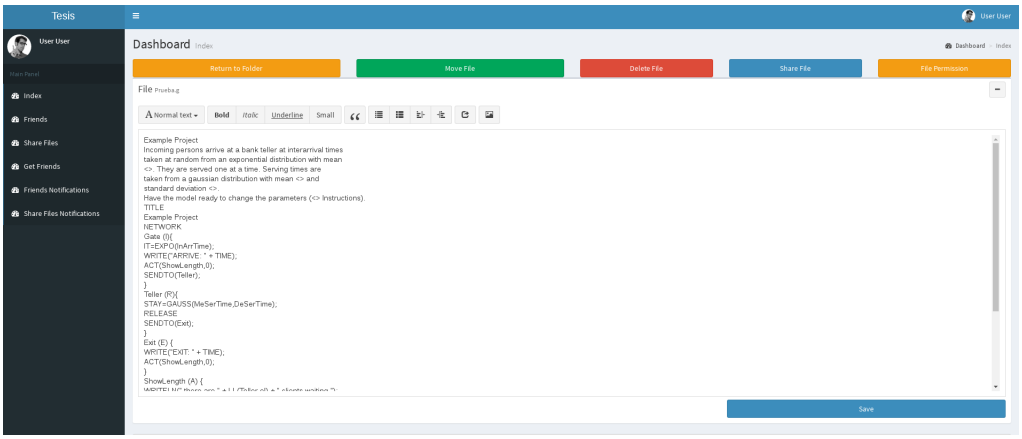

Figura 3.9: Pantalla Observación de Archivo.

- **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.

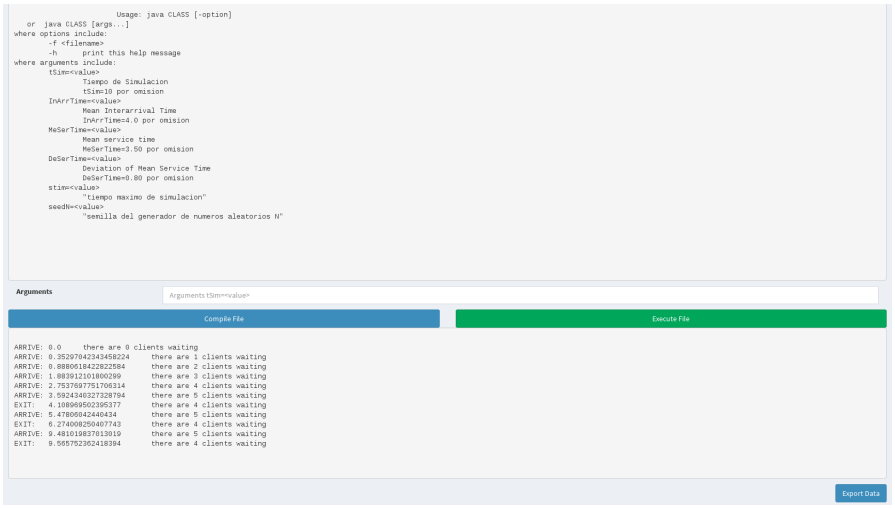

Figura 3.10: Pantalla Compilación y Ejecución.

- **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.

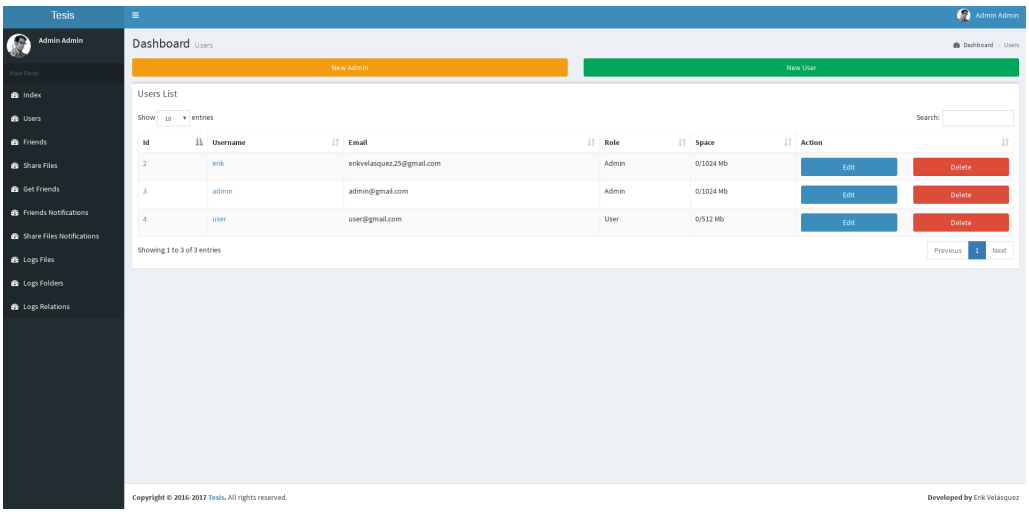

Figura 3.11: Pantalla Control de Usuarios.

3.1.8 Diseño de URLs

Ya que la aplicación web consiste en un largo conjunto 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; 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.
  - **GET /user/** accedemos al dashboard de un usuario registrado. Todas las rutas para acceso de un usuario registrado se definen con esta ruta inicial.
  - **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-.
  - **GET /user/folder/-id-** acceder a la información de una carpeta cuyo id dentro del sistema es -id-.
  - **GET /user/compile/-id-** compila el archivo cuyo id es -id- y retorna la información de compilación.
  - **GET /user/execute/-id-** ejecuta la simulación definida en el archivo cuyo id es -id- y retorna el resultado de dicha simulación.

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 realizan 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 coomo PHP, Ajax, Perl, ASP 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.

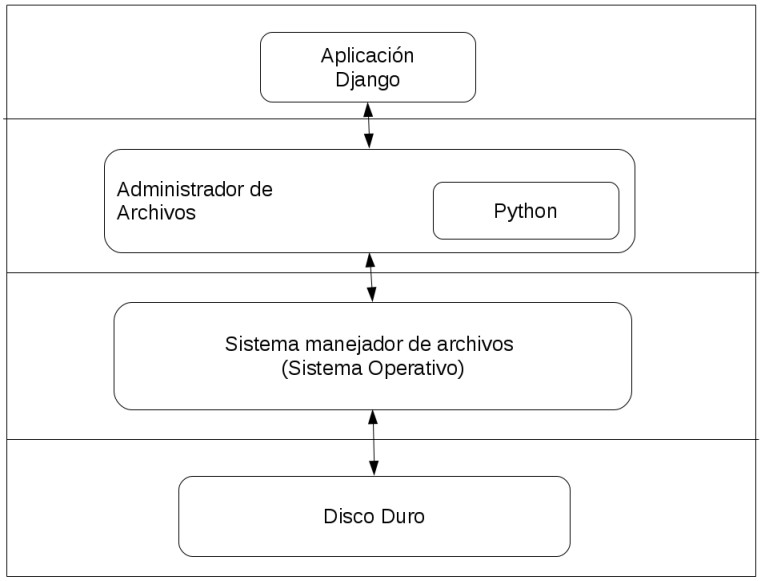

Figura 3.12: Estructura del Administrador de Archivos y Carpetas.

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
Private (0)	Solo el dueño puede acceder a él
Show (1)	Otros usuarios pueden acceder al él solo para lectura
Edit (2)	Otros usuarios pueden acceder al él para lectura y escritura
Delete (3)	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 una de puntos mas importantes dentro de nuestro administrador de archivos, debido a que todos los archivos están relacionados con una carpeta y las carpetas en si tienen una relación jerárquica entre ellas. En la figura 3.13 observamos esa relación jerárquica en nuestro manejo de carpetas.

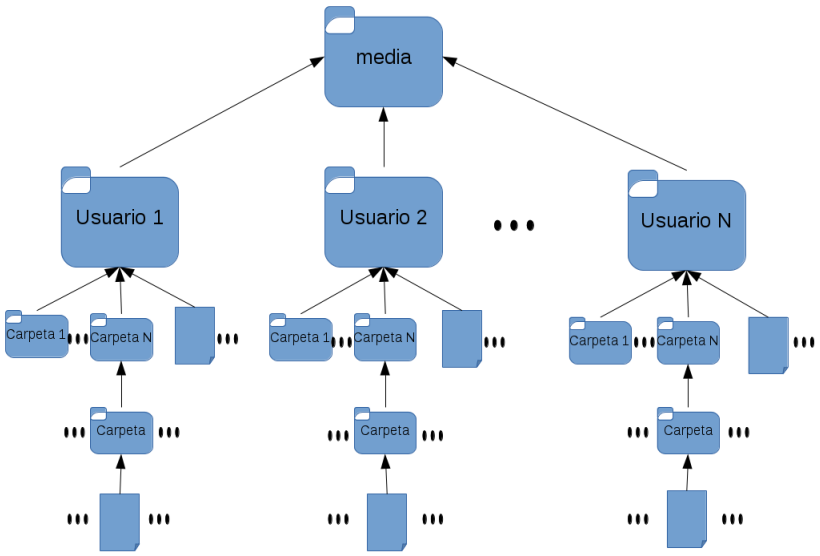

Figura 3.13: Jerarquía de Carpetas.

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 *usuarios*, las cuales funcionan como punto de raíz para su sistema interno de archivos y carpetas.

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 no es ilimitado, 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 su correcto funcionamiento. Esta porción de espacio en disco es fácilmente administrable por el usuario administrador de nuestro sistema web.

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*.

Capítulo 4

Integración de GALATEA

La integración de nuestra aplicación web con su núcleo de simulación GALATEA es una de las principales características que posee nuestro sistema. 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 matríz del sistema Glider, el proyecto paterno 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). El lenguaje base usado para su desarrollo es el Java. Usando como lenguaje base Java, se especifico 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. 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).

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) Llamada a Procesos y (2) Sockets. El primero de ellos lo implementamos llamando a una instancia de GALATEA que ejecuta la simulación, 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 ejecución de la simulación como si fuera un servidor, el cual atiende las peticiones en un puerto de comunicación especifico, 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 Llamada a Procesos

Definir

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).

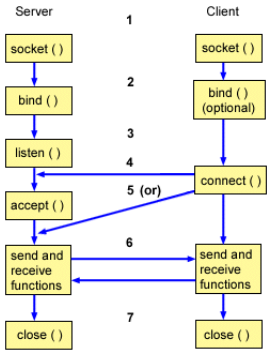

Figura 4.1: Típico flujo usando sockets.

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 4.2, consiste en lo siguiente:

- **El programa cliente**: es un programa generado y ejecutado por nuestro sistema web, este programa esta escrito en Python.
- **El programa servidor**: este programa esta escrito en Java para su complete 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 a acciones a realizar para la simulación como: Iniciar, pausar, terminar, datos de entrada, datos de salida, entre muchos otros.

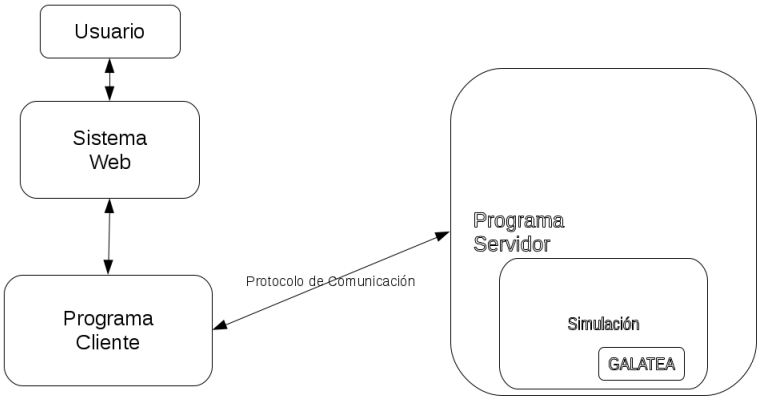

Figura 4.2: Estructura Integración mediante Sockets.

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 esta 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.

4.2.2 Programa Servidor

El programa servidor esta escrito en Java y es ejecutado por nuestra aplicación web. Justo con su ejecución el programa define un puerto de comunicación especifico, que a su vez permite al programa cliente la comunicación. Su función principal es recibir las distintas peticiones del programa cliente, transmitirlas 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 toda la simulación a ser ejecutada.

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.

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}).

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

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

Definir

6.1 Recomendaciones

Definir

Bibliografía

Banks, J., Carson, J. S., Nelson, B. L., y Nicol, D. M. (2010). *Discrete-Event System Simulation*. Pearson, 5 edition.

Fuentes, J. M. (2009). *Manual de AJAX*. 2 edition.

Fundation, D. S. (2015). *The web framework for perfectionists with deadlines*. 5 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.

Nielsen, J. (1994). *Usability Engineering*. Elsevier, 2 edition.

Oracle (2015). Mysql 5.7 :: Reference manual 1.3.1 what is mysql?

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.

Quiroga, M. G. (2011). Estado del arte: Servicios web. Technical report, Universidad Politecnica de Catalunya, Barcelona, España.

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 mediantes 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.