Universidad de las Ciencias Informáticas

Sistema de Publicación Avanzada de *tweets*

(TweetPub)

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Yoandri Rodolfo Díaz Ruíz

Harold Aroche Hernández

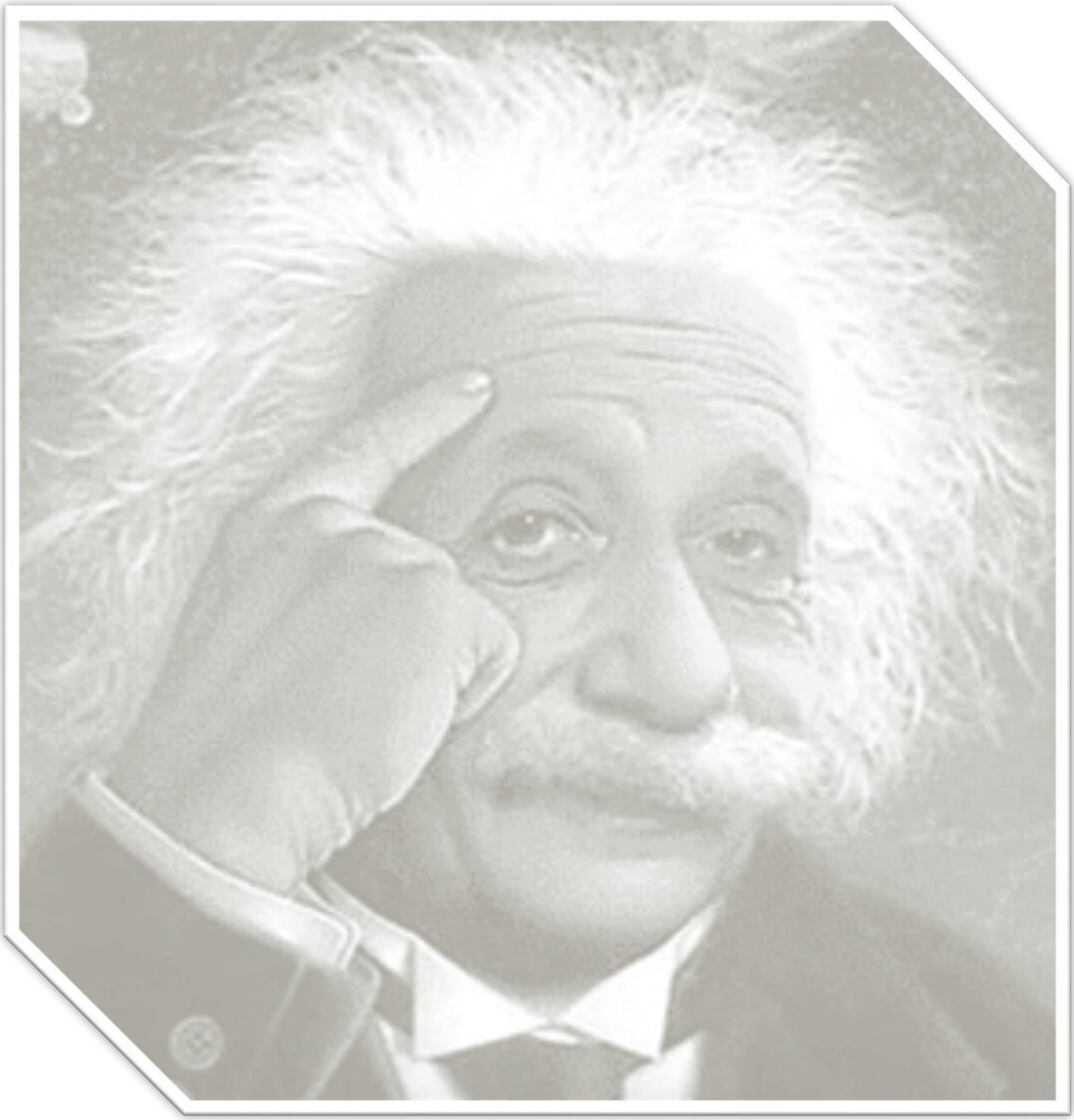
Tutores: Ing. Aliander Capdezuñer González

Ing. Eyeris Rodríguez Rueda

Ing. Yurisledy Hernández Moya

La Habana 2015

Pensamiento



"La religión sin la ciencia estaría ciega, y la ciencia sin la religión estaría coja también."

Albert Einstein

Declaración de Autoría

Declaramos ser los autores del presente trabajo de diploma y autorizamos al Centro de Ideoinformática (CIDI) de la Facultad 1 de la Universidad de las Ciencias Informáticas, para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____	_____
Harold Aroche Hernández	Yoandri Rodolfo Díaz Ruíz
_____	_____
Ing. Aliander Capdezuñer Gonzalez	Ing. Eyeris Rodríguez Rueda

_____

Ing. Yurisleidy Hernández Moya

Datos de Contacto

Tutor: Ing. Aliander Capdezuñer González

Formación académica: Ingeniero en Ciencias Informáticas, 2013.

Centro laboral: CIDI

Correo electrónico: acapdezuner@uci.cu

Tutor: Ing. Eyeris Rodríguez Rueda

Formación académica: Ingeniero en Ciencias Informáticas, 2013.

Centro laboral: CIDI

Correo electrónico: erueda@uci.cu

Tutor: Ing. Yurisleidy Hernández Moya

Formación académica: Ingeniera en Ciencias Informáticas, 2013.

Centro laboral: CIDI

Correo electrónico: ymoya@uci.cu

Agradecimientos

Quiero agradecer primeramente a mi Dios y Señor, al que me ha hecho la persona que soy, al que me ha dado sentido y propósito, al que murió por mí y se levantó de entre los muertos y me sentó junto a Él en los lugares celestiales, a Jesucristo. A Él por darme de sus fuerzas cuando las mías decaían; por acompañarme en los momentos en que sentía estar sólo en esto; por levantar mi cabeza y mostrarme una esperanza cuando todo parecía no tener salida. Y quiero dar gracias al Espíritu Santo, a mi más fiel compañero, al que ha sido mi pastor durante estos cinco años, a Él por enseñarme, consolarme y corregirme en todo tiempo.

A mi mamá Iraida, por ser un modelo de madre, por darme siempre su amor que aunque es finito parece no tener límites, por su sacrificio y entrega por mí; por su aliento en todos los momentos difíciles de mi vida y por su comprensión y apoyo en cada una de mis decisiones. A mi papá Rodolfo, porque aunque a veces no lo demuestro estoy orgulloso de que sea mi padre, porque fue la persona que Dios escogió para traerme a este mundo, a él muchas gracias por estar conmigo. A mi padrastro Pepito, porque me ha cuidado durante todos estos años como si fuese mi propio padre, gracias porque realmente es especial. A mi abuela Ulla, por amarme tanto y comprenderme, por consentirme en todas las cosas que siempre he querido, le agradezco porque 'aguantó', porque ella siempre me decía que Dios se la llevaría antes de yo graduarme y yo le decía que ella tenía que 'aguantar' y así fue 'aguantó'. A mi abuela Elsidia, por estar siempre conmigo y porque mientras he crecido Dios me ha enseñado que es un tesoro para mí. A mis dos hermanas Itsmaray y Raisa, porque las quiero y a pesar de la distancia son importantes para mí. A mi tío Frank por cuidarme tanto desde que estaba en el Círculo Infantil, a Pille y a Chin porque siempre están pendientes de mí y me han ayudado en todas las cosas que han podido, a Marta, Illa, Tío Rafael; a mis primos Javier, Mailín, Abelín, Beatriz, Kiki, Idelsis, Orlandito, Naila, Nailé y a todos los vecinos de Cabaiguán que siempre han estado pendientes de mí, lleguen mis agradecimientos. A mi prima Loli porque a pesar de la distancia ha contribuido y sido de bendición para mi vida y la de mi familia. A mis pastores Giorge, Mirlay, Priscila y Adalberto a quienes amo mucho, y a todos mis hermanos cristianos de Cabaiguán, muchas gracias por sus oraciones y por todo su apoyo espiritual y material que desde que vine a Cristo ellos me han dado. A Celso, Cary, Cristian y Lian, a esa segunda familia que el Señor me ha regalado lleguen mis más profundos agradecimientos por su incansable ayuda. Agradezco a mi grandísima familia cristiana de la UCI, gracias al pueblo del que Dios me ha permitido ser parte durante estos cinco años, gracias a todos por haber marcado mi vida y porque esta familia es una de las razones por las que he permanecido firme en este lugar. Gracias a mis hermanos más cercanos, algunos están y ya otros han partido: A Victor, mi mejor amigo, gracias por estar siempre conmigo, por soportarme estos seis años, por aconsejarme y por corregirme cuando ha sido necesario; a Yaneisi, por soportarme también

desde el IPI, por los momentos compartidos; a Luna, mi padre espiritual, gracias por todos sus consejos; a Angel Lambertt, porque fue bueno compartir junto a él tantas cosas buenas y otras no tan buenas; a José Orlando porque he aprendido muchas cosas con él durante estos años, gracias porque somos un buen equipo; a Kiri muchas gracias, porque con ella aprendí una gran lección para mi vida; a Marcos, que además de ser mi hermano en Cristo fue quien me ayudó con todas las cosas de la programación de la tesis cuando nadie más sabía de Python, muchas gracias por todo el tiempo que me dedicó; a todos mis demás hermanos en Cristo, que no acabaría si los menciono a cada uno, gracias por estar ahí siempre, por sus oraciones y por ser parte de aquellos que han aportado a mi crecimiento espiritual. A mis compañeros de aula, a los que aún siguen en la UCI y a los que ya se han ido, a los soportables y a los insoportables, a todos muchas gracias por estar ahí juntos desde el principio. Al jurado, muchas gracias por su continua comprensión con las situaciones ocurridas en la tesis, a los tres tutores, muchas gracias por ser parte de este trabajo y por su apoyo; a José Gabriel gracias por su tiempo y su ayuda; a la cuarta tutora, la oponente, muchas gracias por su ayuda para las correcciones de esta tesis, fueron muy útiles sus aportes. A todos aquellos que han ayudado de una u otra forma, lleguen estos agradecimientos.

Y quiero terminar como mismo comencé, dando gracias a Aquel que es el Primero y el Último, el Alfa y la Omega, a Jesucristo, al que me ha dado la victoria por medio de su sacrificio en la cruz; al que me dio una nueva vida, perdonó mis pecados y al que puede perdonar los de aquellos que hoy se acerquen a Él. A Dios sea la gloria, la honra, la alabanza y el dominio por siempre.

Yoandri Rodolfo Díaz Ruíz

Quiero agradecer de manera general a todas las personas que me ayudaron de una forma u otra durante todos estos años, de verdad gracias, solo les puedo decir que aquí estoy, dispuesto a ayudar en lo que se pueda a la hora que sea.

Harold Aroche Hernández

Dedicatoria

Al más fiel de todos mis amigos, a Aquel por quien esta tesis ya culminó, a Jesucristo; la dedico primeramente a Él, porque todos mis éxitos le pertenecen, y porque mi verdadero y más grande éxito es permanecer junto a Él.

A mi madre Iraida, la más especial de todas las mujeres; por brindarme siempre su amor durante toda mi vida, y por sus incansables oraciones por mí.

A mi abuela Ulla, a quien amo con todas las fuerzas de mi corazón; porque el saber que todavía la tengo a mi lado en este mundo y que puedo contar con ella, me ha dado mucho ánimo para terminar este trabajo.

A toda mi familia, porque sé que este momento es de mucha alegría y orgullo para todos.

A mi familia en Cristo en la UCI; porque esta tesis es la respuesta de Dios a cada una de sus oraciones.

Yoandri Rodolfo Díaz Ruíz

Dedico esta tesis de manera especial a mis padres María Victoria Hernández Borrote y Elvio Marlon Aroche Durán por apoyarme siempre, por brindarme todo lo que estaba a su alcance a cambio de nada, porque son las dos cosas más grandes que me ha dado la vida.

A todos los que me han brindado su ayuda, a toda mi familia, a mi abuela Tata, muchas gracias por ser una madre más, a mi tías Lourdes e Inés un millón de gracias por todo, a mis tías de nuevitas Mimita y Virna, a mi tío Papito a mi tío Bromnis, a mi hermana un millón de gracias también.

Harold Aroche Hernández

Resumen

El crecimiento acelerado de las Tecnologías de la Información y las Comunicaciones (TIC), específicamente en el área de las redes sociales, ha trascendido el mundo digital haciéndose más tangible. Entre los sistemas de redes sociales de mayor crecimiento se encuentra Twitter. Este es un servicio gratuito de microblogging, que permite a sus usuarios enviar microentradas basadas en textos cortos, llamados *tweets*, de una longitud máxima de 140 caracteres. Para garantizar una presencia activa, efectiva, eficiente y además lograr un menor consumo de ancho de banda de Internet en esta red social no es suficiente con las facilidades que brinda este sistema mediante su interfaz web. Para esto se requiere utilizar otros canales de comunicación con Twitter que permiten de cierta forma automatizar la interacción con la red social. Utilizando estos canales se pretende desarrollar un sistema que garantice estos aspectos mencionados anteriormente que es el objetivo fundamental de este trabajo.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Con el paso del tiempo el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) ha constituido un factor indispensable en el avance de las diferentes esferas de la sociedad. De manera conjunta con este desarrollo surge la Web Social o Web 2.0, la cual ha influido considerablemente en el avance de estas tecnologías, poniendo al alcance de todas las personas la posibilidad de integrarse en el mundo de la Internet, manteniendo un intercambio de información libre y gratuito. Con el progreso de la Web 2.0 surgen las redes sociales, causando un gran impacto en la sociedad.

Cuba diariamente está expuesta al continuo asedio y tergiversación de la información en los medios internacionales, los cuales utilizan las redes sociales para ello, pues a través de ellas se puede divulgar información y acceder a cualquier parte del mundo instantáneamente. Para contrarrestar esta tergiversación, se hace necesario utilizar estas plataformas online para defender el país.

En la Universidad de las Ciencias Informáticas (UCI) se encuentra el Centro de Ideoinformática adscrito a la Facultad 1. Dicho Centro tiene entre sus misiones proveer soluciones, productos y servicios relacionados con las tecnologías de Internet, en función de la defensa de la ideología socialista en la red de redes. Para lograr con éxito esta misión, el Centro hace uso de la red social Twitter que no es más que un servicio gratuito de *microblogging*, que permite a sus usuarios enviar micro entradas basadas en textos cortos, llamados *tweets*, de una longitud máxima de 140 caracteres; a esta red social acceden profesores y estudiantes para realizar un seguimiento de los principales temas generados en ella relacionados con Cuba, y a su vez para divulgar información sobre la verdad cubana.

Con el fin de lograr una presencia activa y eficiente en la red social Twitter, cada persona debe entrar a través de un navegador web a la misma, lo que trae consigo, un alto consumo del ancho de banda de Internet. Este alto consumo se debe al contenido de esta página, la constante actualización en tiempo real de vídeos e imágenes. Además, esta tarea en ocasiones implica el esfuerzo de permanecer despierto a altas hora de la noche, puesto que si se quiere lograr una mayor efectividad en el envío de *tweets*, lo más conveniente es enviarlo en el horario que más frecuenten la red social el público al que se le quiere hacer llegar dichos mensajes; esto se debe a la diferencia de horarios en todo el mundo, es decir, si se pretende que estos *tweets* sean leídos por usuarios españoles hay que enviarlos en un horario donde los españoles hagan un mayor uso de la red social Twitter.

Por todo lo abordado hasta el momento queda claro que la Universidad necesita realizar la tarea de publicar en la red social Twitter de manera eficiente, con el mayor número de usuarios posibles y a su vez con un bajo consumo de ancho de banda de Internet. En la actualidad existen herramientas que facilitan el proceso de planificación y publicación de *tweets* en la red social Twitter; sin embargo no son propias del país o de la UCI, y por el hecho de ser herramientas web publicadas en Internet trae sus riesgos utilizarlas, pues en cualquier momento pueden dejar de estar accesibles ya sea por problemas de mantenimiento o porque sean eliminadas de la red. Por ello, se necesita una herramienta propia de la Universidad, que permita la planificación y publicación de *tweets* a toda hora para realizar de manera eficiente este proceso de publicar en la red social Twitter y lo que se desea lograr con el mismo.

Ante esta dificultad se plantea como **problema de investigación** ¿Cómo lograr la divulgación de información sobre Cuba en la red social Twitter de manera automatizada?

Se define como **objeto de estudio** las tecnologías informáticas para la gestión en las redes sociales el cual se enmarca en el siguiente **campo de acción**: las tecnologías informáticas para la automatización de la publicación de *tweets* en la red social Twitter.

Como una alternativa de solución al problema de investigación antes mencionado, se plantea como **objetivo general** desarrollar un sistema que facilite la planificación y publicación de información en la red social Twitter.

Los **objetivos específicos** derivados del objetivo general son los siguientes:

1. Estudiar el marco teórico conceptual y el estado del arte respecto a las tecnologías y funcionalidades de los sistemas de publicación de *tweets*.
2. Diseñar una aplicación que permita la interacción con la red social Twitter.
3. Implementar un sistema que permita la planificación y publicación de *tweets* en Twitter.
4. Validar el correcto funcionamiento del Sistema de Publicación Avanzada de *tweets*.

Las **tareas de investigación** definidas para darle respuesta al objetivo de la investigación son:

1. Estudio de los conceptos asociados al marco teórico de la investigación.
2. Caracterización de los sistemas de publicación de *tweets* en Twitter.
3. Caracterización de los mecanismos de intercambio de información que provee Twitter.
4. Definición de la arquitectura del sistema.
5. Desarrollo de las funcionalidades que permitan la planificación de *tweets*.

6. Desarrollo de las funcionalidades que permitan la publicación de los *tweets* planificados.
7. Validación del Sistema desarrollado teniendo en cuenta su funcionamiento.

Con el fin de desarrollar las tareas que se plantearon anteriormente fueron empleados **métodos de investigación** que permitieron la obtención de información. A continuación se exponen con qué fines fueron utilizados dichos métodos:

Métodos Teóricos:

Histórico-lógico: Se utilizó para realizar el estudio del estado del arte de los sistemas de publicación de *tweets* y las herramientas y bibliotecas actuales utilizadas para la publicación de *tweets*. Además, se aplicó para el estudio de las diferentes tecnologías y herramientas a utilizar para el desarrollo del presente Sistema.

Analítico-Sintético: Utilizado para consultar la bibliografía relacionada con el tema de la presente investigación y poder obtener de esta los conocimientos necesarios para el desarrollo del Sistema.

Modelación: Con el objetivo de lograr un mejor entendimiento del problema planteado, gracias a la modelación de los diagramas de diseño de la solución.

Métodos Empíricos:

Observación externa, no incluida y cerrada: Se observa cómo otros sistemas realizan la planificación y publicación de *tweets* y qué herramientas, técnicas y bibliotecas utilizan.

El contenido del presente trabajo de diploma está estructurado en tres capítulos de la manera siguiente:

Capítulo 1. Fundamentación Teórica de la Publicación Avanzada de *tweets*.

En el presente capítulo se abordarán los conceptos fundamentales que permitirán entender las características de los servicios web y de las diferentes formas de interactuar con la red social Twitter. Así como la fundamentación de las herramientas, técnicas, tecnologías y metodologías que se emplearán en el desarrollo del trabajo de diploma.

Capítulo 2. Características del Sistema.

En este capítulo se definen los principales requisitos de la propuesta de solución, el modelo de dominio, los casos de uso y el refinamiento del análisis y el diseño. Se describen las clases que se utilizan en la implementación del sistema y las relaciones entre ellas. Se explica además, la arquitectura del Sistema, haciendo énfasis en los patrones arquitectónicos y de diseño utilizados. Se

realiza además, el diseño del sistema mediante los distintos artefactos propuestos por la metodología de desarrollo de *software* seleccionada.

Capítulo 3. Implementación y pruebas.

En este capítulo se describen las principales clases del sistema, así como los artefactos correspondientes a la fase de implementación de la metodología seleccionada. También se aplican los tipos de casos de prueba adecuados para validar la aplicación que permiten detectar y corregir los posibles errores a través de los resultados obtenidos en cada una de las iteraciones.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA DE LA PUBLICACIÓN AVANZADA DE *TWEETS*.

A lo largo de este capítulo se abordan numerosos conceptos relacionados con la Investigación, con el objetivo de lograr una mejor comprensión de cómo interactuar con esta. Se explican, además, varios de los principales servicios que brinda la red social; así como algunas herramientas web, usadas en el mundo, para la publicación avanzada de *tweets*. Se analizan también diferentes herramientas, metodologías y lenguajes de programación, con el objetivo de escoger las más adecuadas para el desarrollo del Sistema a realizar.

Definiciones de conceptos más usados:

A continuación, se definirán varios elementos con el fin de lograr la adquisición de conocimientos previos, necesarios para entender mejor algunos temas futuros con mayor nivel de complejidad encontrados en el presente trabajo.

Definición de Web 2.0

El término Web 2.0 hace referencia a la evolución que ha experimentado el servicio web. Su constante progresión ha pasado de unas páginas estáticas HTML, conocidas como Web 1.0, a un nivel más alto, caracterizado por la creación de documentos dinámicos (Web 1.5). Pero los cambios que se incluyen ahora son más profundos y complejos. Dicho de otra forma, se puede considerar como Web 2.0 todas aquellas utilidades y servicios de Internet que se sustentan en una base de datos, la cual puede ser modificada por los usuarios del servicio; cuando permita procesos de interactividad de contenidos, el usuario pueda añadir y compartir información con otros usuarios (Ribes, 2007).

Definición de Red Social en Internet

Con el desarrollo de la Web 2.0, la historia de los sitios web llega a una segunda generación, cuyo denominador común es que están basados en el modelo de una comunidad de usuarios, introduciéndose en la informática el nuevo término de redes sociales en Internet (Dougherty, 2004).

Las redes sociales, son formas de interacción social, definidas como un intercambio dinámico entre personas, grupos e instituciones. Estas, son un sistema abierto y en construcción permanente, involucrando a conjuntos de personas que se identifican con las mismas necesidades y problemáticas y que se organizan para potenciar sus recursos (Martínez, y otros, 2011)

Todos estos sitios utilizan la inteligencia colectiva para proporcionar servicios interactivos en la red donde el usuario tiene control para publicar sus datos y compartirlos con los demás.

Definición de API

La Interfaz de Programación de Aplicaciones (API por sus siglas en inglés). Esta se refiere a una biblioteca que implementa cierta funcionalidad mediante una interfaz con la cual el programador puede acceder a dicha funcionalidad (Stefan Zerbst, 2004).

Se puede concluir que una API es una llave de acceso a funciones que permiten hacer uso de un servicio web provisto por un tercero, dentro de una aplicación web propia, de manera segura. Es decir, es una interfaz implementada por una aplicación que permite a otras aplicaciones comunicarse con ella (Stefan Zerbst, 2004).

Twitter

Twitter, es una red de microblogging que permite escribir y leer mensajes en Internet de hasta 140 caracteres. Estas entradas o mensajes son conocidas como *tweets*. Los *tweets* además, pueden contener etiquetas, que permiten establecer el tema o enfoque que se le pretende dar a la publicación, o relacionarlo con un tema de conversación que se encuentra en las tendencias del momento. Como red social, Twitter se basa en el principio de los seguidores, es decir, cuando se elige seguir a un usuario, los *tweets* de ese usuario aparecerán en orden cronológico inverso, en la página principal de Twitter. Por ejemplo, si alguien sigue a 20 personas, este verá una mezcla de *tweets* de esas 20 personas en el orden en que vayan publicando mensajes, siendo el primero que vea, el último que ellos hayan escrito. Esta red social es utilizada para la divulgación de información, para el intercambio de ideas entre sus usuario, en tiempo real e instantáneamente.

API de Twitter

Twitter tiene publicada a disposición de sus usuarios su propia API, de forma tal que se puedan crear aplicaciones que se comuniquen con estas, teniendo en cuenta ciertas restricciones. En el presente epígrafe, se tratará sólo lo referente al API *Rest* que es con la cual interactuará la aplicación a desarrollar debido a que es la que permite el acceso a leer y escribir datos en Twitter. La API *Rest* identifica las aplicaciones de Twitter y usuarios utilizando OAuth y brinda la información en formato json (Twitter, 2014).

Sistema de autenticación basado en token

El sistema de autenticación basado en token es una de las vías que existen para autenticarse con la API de Twitter, el cual es el utilizado y abordado en la siguiente investigación debido a la seguridad que ofrece. Al usar este tipo de autenticación, no hay que preocuparse por la falsificación de petición en sitios cruzados (CSRF por sus siglas en inglés) ya que no se dispone de ninguna sesión sensible de ser suplantada. Existen dos métodos de autenticación:

1. *OAuth:* Este método permite realizar todo tipo de funciones con la API de Twitter. Tiene el inconveniente de que es el más complejo de usar. Este método de autentificación requiere el uso de “tokens de acceso” que para conseguirlos se pueden emplear varias vías (Twitter, 2014).
2. *Application-only authentication*: esta es una autentificación únicamente para aplicaciones, es un poco más sencilla que la OAuth, pero tiene el gran inconveniente de que no permite publicar en Twitter. Un ejemplo del uso de este tipo de autenticación sería en una aplicación de escritorio que mostrase el muro de Twitter de un usuario especificado, el cual es más conocido como ‘*timeline’* (Twitter, 2014).

En esta investigación la vía utilizada para obtener los tokens de acceso para la aplicación a desarrollar poder publicar en Twitter fue la siguiente:

Se consultó la red social Twitter, se creó la aplicación, directamente en el apartado *‘Details’* de la aplicación Twitter se generan pulsando el botón *‘Create my acess token’*, ya que por defecto no se crean.

Soluciones internacionales

Existen numerosas aplicaciones en línea que permiten la comunicación con la red social Twitter. Como parte del estado del arte de la presente investigación, se estudiaron algunas de estas herramientas, enfatizando en su funcionamiento de manera general y en los servicios que brindan. Este estudio se realizó con el objetivo de resumir sus características y observar si ellas brindan algún servicio de interés para el desarrollo del presente Sistema. Se estudiaron estas porque eran las que permitían su uso puesto que eran libres. A continuación se explican las herramientas estudiadas.

Buffer

Buffer es reconocido como la forma más inteligente de publicación de *tweets*. Cuenta con dos versiones una una gratuita y otra de pago. Analizando la versión gratuita sobre la cual se realizó un estudio, interactuando directamente con la aplicación, se observó que la misma permite programar *tweets* de forma muy simple. Esta herramienta posibilita a los usuarios hacer publicaciones sin tener que acceder directamente a la red social donde tenga previamente creada una cuenta. El usuario puede registrarse mediante su cuenta de Twitter, Facebook y LinkedIn, ya que la herramienta acepta las principales redes sociales. Su funcionamiento únicamente se da en la web, por lo cual no posee opción de descarga para el ordenador. Una desventaja de la misma es que no hay una forma de configurar los *tweets* para que se ejecuten recurrentemente.

Twuffer

Twuffer, la cual permite componer una lista de *tweets* futuros y planificar su publicación. Una vez que el usuario haya iniciado sesión y autorizado el servicio, puede introducir el contenido a difundir y ver una lista con los *tweets* programados y con los enviados. Es fácil y rápida de usar para planificar la publicación delos *tweets*. Su interfaz es muy sencilla. Como desventaja esta no posee sistema acortador de enlaces, estos enlaces se copian tal como se hace en Twitter o reducir en otro servicio acortador. Además, no facilita la medición e impacto de los *tweets* enviados.

HootSuite

Hootsuite es una herramienta de gestión de medios de comunicación social, esta les permite a sus usuarios programar mensajes con antelación y proporcionar una función de planificación para ayudar a que publique en el momento óptimo de días, y de esta forma llegar al público más amplio. Tiene características similares a *TweetDeck* en términos de crear columnas para organizar su actividad de Twitter. Esta funcionalidad llamada auto-cronograma (*autoschedule*), hace el trabajo por el usuario cuando se trata de elegir el mejor momento para publicar un *tweet*, basándose en los horarios en los que frecuentan la red social Twitter los seguidores de dicho usuario. Esta funcionalidad elige un tiempo para que cuando los *tweets* lleguen al momento en que podrían ser mejor recibidos, entonces se publiquen. Esta información le permitirá planificar cuándo programar sus mensajes en el futuro.

TweetDeck

TweetDeck es otra poderosa herramienta de administración utilizada por Twitter. Tiene características similares a *HootSuite* en términos de crear columnas para organizar su actividad de Twitter. Permite registrarse con el usuario de Twitter o mediante correo electrónico. Se pueden enviar mensajes con más de 140 caracteres, para los cuales se crea una URL acortada donde se puede ver el mensaje competo. Esta herramienta cuenta, entre otras muchas funcionalidades, con la facilidad de planificación de los *tweets*. La misma además permite seleccionar a qué persona específica enviarle el *tweet* o mensaje.

FutureTweets

FutureTweets es un servicio gratuito que permite planificar mensajes para luego ser publicados. Permite enviar el *tweet* a una hora determinada en el futuro o enviar un *tweet* diaria, semanal, mensual o anualmente. Con esta herramienta no se pudo interactuar directamente por estar fuera de servicio por cambios en la API de Twitter.

Aportes de las herramientas estudiadas

Luego de un estudio de las diferentes características que presentan las herramientas internacionales analizadas, se llega a la conclusión de que lo ideal sería no depender de ellas. Lo aconsejable es tener una aplicación propia de la Universidad que cumpla con las necesidades del cliente, las cuales se resumen a modo de requisitos en el capítulo siguiente, y que la aplicación además pueda ser modificada en caso de ser necesario para que cumpla con necesidades futuras. Se sugiere esto debido a que durante el estudio de estas aplicaciones, en ocasiones algunas no estaban disponibles por diversas razones, ya sea por mantenimiento o por problemas con la API de Twitter.

A pesar de todo lo expresado anteriormente, estas herramientas aportaron un sin número de ideas que facilitarán el desarrollo de este trabajo, aportaron conocimientos de cómo comunicarse con la red social Twitter, además de algunos elementos a tener en cuenta a la hora de realizar el diseño de las interfaces web con las que interactuarán los usuarios que accedan al sitio.

Soluciones nacionales

En la investigación efectuada para el estado del arte no se encontró disponible en Cuba ninguna plataforma nacional que interactúe con la red social Twitter. A pesar de que en el año anterior, en la Universidad de las Ciencias Informáticas se realizó una investigación sobre el tema para implementar una plataforma que brindase los servicios de Twitter a terceras aplicaciones, conocida como la Plataforma para la Red Social Twitter (PRST), se puede señalar que la misma sólo está orientada a servicios. No presenta una interfaz para que los usuarios puedan interactuar con ella, además, tampoco permite la publicación de *tweets* en Twitter de forma programada. Por estas razones, el desarrollo de la solución propuesta por los autores de la presente investigación cobra aún mayor significado.

Metodologías de desarrollo

Las metodologías empleadas en el desarrollo de *software* son un conjunto de procedimientos, técnicas, herramientas, y una documentación que ayuda a los desarrolladores a realizar nuevos productos informáticos. Según (Acuña, 2009), la comparación y clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, la información disponible y alcance de cada una de ellas. Si se toma como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, se pueden clasificar las metodologías en dos grupos, metodologías estructuradas y metodologías orientadas a objetos. Por otra parte, si se considera su filosofía de desarrollo se pueden diferenciar en metodologías tradicionales y ágiles.

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de metodologías tradicionales o pesadas (Acuña, 2009).

Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del *software*, con el fin de conseguir un *software* más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto *software*. Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar. Entre las metodologías tradicionales o pesadas se pueden citar las metodologías: Proceso Unificado del Rational (RUP, por sus siglas en inglés), MSF (del inglés *Microsoft Solution Framework*), la metodología *Win-Win Spiral Model* y por último *Iconix*.

Por otra parte las metodologías ágiles se caracterizan por su atención continua a la calidad técnica de un buen diseño, por la colaboración entre desarrolladores y la gente del negocio. Su prioridad es satisfacer al cliente, la simplicidad es esencial, acepta cambios durante el desarrollo del sistema, permite la existencia de motivación y confianza en los individuos, además de la entrega frecuente de *software*, entre otras características (Acuña, 2009).

Teniendo en cuenta las características antes expuestas de los diferentes tipos de metodologías, se decidió utilizar una metodología de tipo ágil puesto que la misma como su nombre lo indica permite un ágil desarrollo, su prioridad es satisfacer al cliente, la simplicidad es esencial, acepta cambios durante el desarrollo del Sistema etc.

Luego de seleccionado el tipo de metodología ágil, sólo falta seleccionar de entre estas metodologías, la más eficiente para la realización de la presente investigación, teniendo en cuenta las características del proyecto, del equipo y la complejidad del Sistema a desarrollar. Se decidió valorar dos de las metodologías ágiles existentes: Programación Extrema (XP por sus siglas en inglés) y Proceso Unificado Abierto (OpenUP por sus siglas en inglés). A continuación se abordan las principales características de las metodologías recién mencionadas.

XP

La clave del proceso de desarrollo de XP es la comunicación. La gran mayoría de los problemas en los proyectos de desarrollo son provocados por falta de comunicación en el equipo, así que se pone un gran énfasis en facilitar que la información fluya lo más eficientemente posible. XP es una de las metodologías de desarrollo de *software* más exitosas en la actualidad, utilizada para proyectos de corto plazo y para equipos pequeños. La metodología consiste en una programación rápida o

extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (Robles, 2002).

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas (Robles, 2002).

OpenUP

OpenUP es una metodología de *software* ágil y unificado, que contiene el conjunto mínimo de prácticas que ayudan a los equipos a ser más eficaces en el desarrollo de *software*. OpenUP se basa en una filosofía pragmática y ágil que se centra en la naturaleza colaborativa de desarrollo de *software*. Es un proceso iterativo que es mínimo, completo y extensible, que puede utilizarse en este tipo de proyecto o ampliarse para tratar una gran variedad de tipos de proyectos. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Está organizada dentro de cuatro áreas principales de contenido, comunicación y colaboración, intención, solución y administración (Balduino, 2007).

Está organizado en dos dimensiones diferentes pero interrelacionadas, el método y el proceso. El contenido del método es donde los elementos del método, es decir los roles, las tareas, los artefactos y los lineamientos son definidos, sin tener en cuenta cómo son utilizados en el ciclo de vida del proyecto. El proceso es donde los elementos del método son aplicados de forma ordenada en el tiempo. Muchos ciclos de vida para diferentes proyectos pueden ser creados a partir del mismo conjunto de elementos del método (Balduino, 2007).

Según (Hernández, 2013) esta metodología se caracteriza por cuatro principios básicos que se soportan mutuamente, los cuales son la colaboración para alinear los intereses y un entendimiento compartido; el balance para confrontar las prioridades, es decir las necesidades y costos técnicos, para maximizar el valor para los stakeholders (persona o entidad que es afectada por las actividades de una organización); el enfoque en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra; y por último la evolución continua para reducir riesgos y demostrar resultados.

Justificación de la metodología escogida

Como parte del proceso de esta investigación se seleccionó como metodología para el desarrollo del *software*, OpenUP. Para ello se tienen en cuenta algunas características de este proyecto, entre las cuales se halla que el mismo cuenta con pocos integrantes, además de que es un proyecto

pequeño. Al analizar esta metodología se observó que la misma brinda un marco ideal para el desarrollo de proyectos pequeños y con un número reducido de integrantes, como es el caso. Esta metodología toma lo mejor de RUP, y en cuanto al diseño establece un desarrollo ágil, iterativo e incremental. Presenta además un mejoramiento continuo centrado en el desarrollo colaborativo del *software*, donde interactúan programador y cliente en cada una de las iteraciones, reduciéndose así “en gran medida” los riesgos en la construcción de la propuesta de solución.

Bibliotecas para la conexión con Twitter

Después de analizar las herramientas existentes nacional e internacionalmente, se tiene como punto de partida investigar cuáles pueden ser las bibliotecas que faciliten la comunicación con la API de Twitter y así consumir los servicios de publicación de *tweets* que brinda dicha red social. Para ello se presenta a continuación un estudio de algunas bibliotecas que brindan estas facilidades y la selección realizada para el desarrollo de la tesis.

Python-Twitter

Python-Twitter es una biblioteca que proporciona una interfaz de Python puro para trabajar con la API de Twitter de una manera sencilla. Permite realizar acciones usando sencillos métodos. Pero no se cuenta con mucha bibliografía disponible acerca de la misma (HitHup, 2014).

Twython

Según la descripción mostrada en (HitHup, 2014) Twython es la biblioteca Python premier que provee una manera fácil de acceder a información de Twitter. Esta biblioteca se encuentra en constante actualización por lo que puede ser una gran alternativa para integrar en proyectos que requieran de una interacción con los datos de Twitter. (HitHup, 2014).

Twython soporta búsqueda dentro de la información de usuario, la lista de Twitter, el *timeline* o el muro del usuario, en los mensajes directos y busca además, cualquier cosa que se encuentre en la documentación de la API de Twitter. En cuanto a su relación con las imágenes, esta biblioteca permite la publicación de un *tweet* con una imagen, brinda al usuario la posibilidad de cambiar la imagen de perfil. Brinda soporte de sólo lectura para OAuth2, para la API de Stream de Twitter y para Python 3.

Tweepy

Tweepy es una biblioteca fácil de usar para acceder a la API de Twitter desde Python, sirve para conocer y administrar mejor la cuenta de Twitter del usuario. Al utilizar la biblioteca Tweepy es posible realizar un análisis puntual. La herramienta permite seguir o dejar de seguir por grupos de 40 usuarios de forma rápida. Tweepy tiene como ventaja ser gratuita (HitHup, 2014).

Selección de la biblioteca para la conexión con Twitter

Se decide utilizar Twython debido a que cuenta con un gran número de funcionalidades que facilitarán y agilizarán la interacción con la API de Twitter. Además, es una poderosa biblioteca que constantemente se actualiza, lo cual permite mejoras de la misma con mucha más rapidez. Está programada en el lenguaje de programación Python. Esta servirá de vínculo entre la aplicación y la API Rest de Twitter, facilitando el proceso de consulta y de gestión de la información del servicio de la red social, de forma transparente para la aplicación.

Framework de desarrollo

En el desarrollo de *software*, un marco de trabajo o *framework*, es una estructura de soporte definida, mediante la cual un proyecto de *software* puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto (*Framework*, 2014). A continuación, se presentan algunos *frameworks* estudiados y la selección del utilizado para desarrollar el presente Sistema.

Django

Django es un *framework* web de código abierto escrito en Python, que permite construir aplicaciones web de manera sencilla. Fue inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias del sitio web *The World Online* (El Mundo en Línea), más tarde se liberó bajo licencia BSD. Django se centra en automatizar todo lo posible y se adhiere al principio ‘No repetirse a sí mismo’ (DRY por sus siglas en inglés) (Bennett, 2009).

Django, además, está inspirado en la filosofía de desarrollo Modelo Vista Controlador, sus desarrolladores declaran públicamente que no se sienten especialmente atados a observar estrictamente ningún paradigma particular y en cambio prefieren hacer lo que les parece correcto (Bennett, 2009).

Symfony2

Symfony2 proporciona un conjunto de herramientas para desarrollar aplicaciones web. Es una colección de más de veinte bibliotecas independientes que se pueden utilizar dentro de cualquier proyecto PHP. Estas bibliotecas, llamadas componentes de Symfony2, contienen algo útil para casi cualquier situación, independientemente de cómo se desarrolle el proyecto. La plataforma Symfony2 proporciona una selección de componentes y bibliotecas de terceros; y proporciona configuración sensible. El objetivo de la plataforma es integrar muchas herramientas independientes; es un paquete que se puede configurar o sustituir completamente (Pacheco, 2011).

Una característica muy importante de Symfony2 es su flexibilidad. Para el formato de los archivos de configuración usa PHP, Yaml y XML y con el fin de tener un buen rendimiento, tanto Yaml como XML se transforman en PHP; las anotaciones del código se compilan y se ejecutan a PHP (Potencier, 2009).

Selección del *framework* de desarrollo

Entre los dos *frameworks* analizados, es válido observar que Django presenta características representativas que llevan a su elección. Entre ellas está que posee un sistema de redirección de URL con una amplia documentación incorporada, y presenta un sistema de base de datos robusto que facilitará la conexión con la API de Twitter mediante la biblioteca Twython anteriormente seleccionada. Tiene además como lenguaje de programación base Python. Por estas razones se selecciona el *framework* Django en su versión 1.6.2 para el desarrollo de este Sistema.

Sistema Gestor de Base de Datos (SGBD)

En este epígrafe se procede a estudiar algunos de los sistemas gestores de bases de datos más utilizados en la actualidad (Donostiarra, 2014) y a seleccionar aquel que mejor se ajuste a resolver las necesidades del Sistema a desarrollar.

Los SGBD son un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información, proporcionan además, herramientas para añadir, eliminar, modificar y analizar los datos. El propósito general de los SGBD es el de manejar de manera sencilla, clara y ordenada un determinado conjunto de datos (Martinez, 2009).

Las bases de datos relacionales cumplen con el modelo relacional, el cual es el modelo más utilizado en la actualidad para implementar bases de datos ya planificadas. Permiten establecer conexiones entre los datos que están guardados en tablas y a través de dichas conexiones relacionar los datos de ambas tablas (Martínez, 2010).

Dentro de los SGBD relacionales usados a nivel internacional se encuentran PostgreSQL, MySQL, Oracle y SQL-Server. Sin embargo, entre los almacenes de datos No SQL que más sobresalen en la actualidad se encuentran MongoDB, Cassandra y BigTable, todos usados en aplicaciones que manejan grandes volúmenes de datos como son Twitter, Facebook y Google respectivamente.

PostgreSQL

Este es un SGBD objeto-relacional, con su código fuente disponible libremente. Es uno de los SGBD de código abierto más potente del mercado. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (Martínez, 2010).

MongoDB

Un estudio enfocado en las base de datos no relacionales arroja que MongoDB constituye una solución escalable y de alto rendimiento de almacenes de datos No SQL. Es de código abierto y escrito en C++, este sistema tiene orientado el almacenamiento de datos en documentos al estilo json con esquemas dinámicos, que ofrecen potencia y simplicidad. Se destaca por conservar los índices de todos los atributos y hacer mucho más flexible la agregación y procesamiento de datos (MongoDB, 2011).

Las colecciones en MongoDB son análogas a las tablas en una base de datos relacional. Cada colección contiene varios documentos. Como se mencionó anteriormente estos documentos pueden ser grandes. No requiere que en una colección todos los documentos tengan la misma estructura, sin embargo, en la práctica, la mayoría de las colecciones son altamente homogéneas (MongoDB, 2011).

Selección del Sistema Gestor de Bases de Datos

Dadas las características anteriores y valorando las necesidades del Sistema a desarrollar, se determina que se deberá utilizar PostgreSQL como SGBD de la aplicación. El mismo permitirá el almacenamiento de los *tweets* que se planifiquen para ser publicados en Twitter y la información asociada a estos, dígase los usuarios a nombre de los que se desea publicar, la fecha planificada y en caso de seleccionar una imagen para publicar, su dirección URL. Se escoge este gestor ya que el mismo logra un buen manejo de la información frente a un volumen considerable de datos.

Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un programa informático. Proporciona a los desarrolladores múltiples herramientas que permiten crear aplicaciones tanto para la web como para dispositivos portátiles. Está compuesto por un conjunto de herramientas de programación. Puede dedicarse únicamente a un solo lenguaje de programación o bien puede utilizarse para varios. Según (Rodríguez Galván, J y otros) un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de Interfaz Gráfica de Usuario (GUI por sus siglas en inglés) (Alonzo Velázquez, 2010).

PyCharm

PyCharm es utilizado para la programación en Python. Proporciona análisis de código, depurador gráfico, probador de unidad integrada, VCS / integración DVCS y apoya el desarrollo web con Django. Este IDE es desarrollado por la empresa checa JetBrains. Es multiplataforma ya que opera

sobre Windows, Mac OS y Linux. PyCharm ha sido liberado bajo una licencia propietaria y también bajo la licencia Apache (JetBrains, 2011).

Este IDE brinda una buena asistencia de codificación con la finalización de código inteligente, inspecciones de código, resaltado de error en la marcha, auto-correcciones, junto con refactorizaciones de código automatizados y abundantes capacidades de navegación por las principales implementaciones de Python (JetBrains, 2011).

Spyder

Spyder es un IDE para Python con edición avanzada, pruebas interactivas, introspección y algunas otras características. Está especialmente recomendado para computación científica gracias a NumPy (álgebra lineal), SciPy (procesamiento de imágenes y señales), matplotlib (ploteo interactivo en 2D/3D) y soporte a mlab de MayaVi (visualización 3D interactiva). Tiene un workflow (flujo de trabajo) especializado para personas no avanzadas en la programación, aunque puede ser muy útil también para programadores con mayor experiencia. Spyder también puede ser usado como una biblioteca, ya que provee poderosos widgets (reproductores) de PyQt4 relacionados con la consola (Gómez, y otros, 2011).

Selección del IDE

Para el desarrollo de este Sistema, se ha escogido como IDE PyCharm en su versión 3.4, ya que el mismo tiene una versión libre y además es potente. Proporciona un gran número de herramientas y funciones para realizar el presente trabajo, recogidas en una interfaz que lo hace fácil y agradable de usar. La integración con la biblioteca y el *framework* seleccionados lo convierten en una valiosa opción para el desarrollo del Sistema.

Herramientas CASE

La Ingeniería de *Software* Asistida por Ordenador (CASE por sus siglas en inglés) no son más que herramientas utilizadas para la administración del desarrollo de *software*. CASE proporciona al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería (Pressman, 2005).

Rational Rose Enterprise Edition

Rational Rose brinda soporte al Lenguaje Unificado de Modelado (UML por sus siglas en inglés) y a la vez ofrece distintas perspectivas del sistema. Propone un diseño dirigido por modelos que favorece en productividad a los desarrolladores, admitiendo el lenguaje de modelado UML y técnicas de modelado de objetos.

En la definición de sistemas esta herramienta permite que el equipo de desarrollo entienda mejor el problema, que identifique las necesidades del cliente en forma más efectiva y comunique la solución propuesta de forma más clara. Según (Rose, 2007), además de poseer capacidades de ingeniería inversa, permite completar una gran parte de las disciplinas de RUP tales como captura parcial de requisitos; completo análisis y diseño; implementación, como ayuda; y control de cambios y gestión de configuración parcialmente.

***Visual Paradigm* 8.0**

Visual Paradigm es una herramienta CASE profesional que soporta todo el ciclo de vida del *software*: análisis y diseño orientado a objetos, construcción, prueba y despliegue. Dicha herramienta ayuda a una rápida construcción de la aplicación con alta calidad y a un menor costo. Permite modelar todos los tipos de diagrama de clases, código inverso, generar código desde diagramas y generar documentación (UML, 2010).

La herramienta también proporciona abundantes tutoriales, demostraciones y proyectos UML. La misma presenta una interfaz de uso intuitiva y con muchas facilidades a la hora de modelar los diagramas que soportan la ingeniería de requerimientos (UML, 2010).

La herramienta está diseñada para una amplia gama de usuarios, incluidos los ingenieros de *software*, analistas de sistemas y analistas de negocio, o para cualquiera que esté interesado en la construcción de forma fiable a gran escala de sistemas de *software*, utilizando un enfoque orientado a objetos. Además, presenta licencia gratuita y comercial (UML, 2010).

Selección de la herramienta CASE

Luego de estudiar estas dos herramientas se llega a la conclusión de que la más apropiada es Visual Paradigm en su versión 8.0, ya que es un potente generador de informes en formato PDF/HTML, permite realizar ingeniería inversa y directa, soporta varios usuarios trabajando sobre un mismo proyecto y permite agilidad en el trabajo del analista. No se escoge *Rational Rose*, pues no contiene el ciclo de vida completo del desarrollo del *software*.

Lenguaje de Programación

Un lenguaje de programación es un convenio entre personas que puede definirse como un conjunto de reglas o normas que permiten asociar a cada programa correcto un cálculo que será llevado a cabo por un ordenador, sin ambigüedades. Por tanto, un lenguaje de programación es un convenio acerca de cómo se debe de interpretar el significado de los programas de dicho lenguaje ya que en ocasiones se confunden los lenguajes con los compiladores, intérpretes o con los entornos de desarrollo de *software* (Ureña, 2011).

Una vez escogidas todas las herramientas anteriores que se utilizarán en el desarrollo del Sistema de Publicación Avanzada de *tweets*, se procede a elegir el lenguaje de programación. Teniendo en cuenta la biblioteca para la conexión con la API de Twitter, el *framework* de desarrollo a utilizar y el IDE seleccionado, se puede definir que el lenguaje a utilizar será Python, ya que el mismo se integra adecuadamente con cada una de las herramientas anteriormente explicadas.

Python es un lenguaje de programación interpretado, orientado a objetos, multiplataforma y de sintaxis sencilla. Permite dividir un programa en módulos reutilizables desde otros programas en Python. El lenguaje incorpora una gran colección de módulos estándar que se pueden utilizar como base de los programas, o como ejemplos para empezar a aprender Python. Permite además, escribir programas muy compactos y legibles. Estos programas son normalmente mucho más cortos que sus equivalentes en C o C++, es ampliable o conocido como Lenguaje Integrador. Otra característica de Python es que sintácticamente escribir un programa en este lenguaje es muy sencillo, y la razón de esto es que usa tabulación o espaciado para mostrar estructura de bloques (Corso, 2007).

Lenguaje de Modelado

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) provee un sistema de arquitecturas trabajando con objetos, análisis y diseño, con una buena consistencia del lenguaje para especificar, visualizar, construir y documentar los artefactos de un sistema de *software*.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones. Según (Romero Guillén, 2004) los objetivos que se establecieron al desarrollar el UML fueron los siguientes:

1. Proporcionar a los usuarios un Lenguaje de Modelado Visual de tal forma que sea posible intercambiar información de los modelos.
2. Proporcionar mecanismos de extensibilidad y especialización para ampliar los conceptos básicos.
3. Ser independiente de un lenguaje en particular y del proceso de desarrollo.
4. Proporcionar bases formales para la comprensión del Lenguaje de Modelado.
5. Integración en una mejor práctica.

Es fácil predecir que UML será el lenguaje de modelado de *software* de uso universal. Las principales razones para ello son que en su desarrollo han participado investigadores de reconocido prestigio, ha sido apoyado por prácticamente todas las empresas importantes de informática, se ha aceptado como un estándar por la OMG y prácticamente todas las herramientas CASE y de desarrollo la han adaptado como lenguaje de modelado.

En resumen, UML resuelve de forma satisfactoria un viejo problema del desarrollo de *software* como es su modelado gráfico. Además, se ha llegado a una solución unificada basada en lo mejor que había hasta el momento, lo cual lo hace todavía más excepcional (Hernández Orallo, 2010).

Conclusiones Parciales

Con el resultado de la presente investigación se puede concluir que con la realización de este capítulo se abordaron conceptos fundamentales relacionados con el problema planteado. También se realizó un estudio sobre los sistemas existentes que permiten la publicación de *tweets* en Twitter, determinando que ninguno cumplía con las características requeridas como se explicaron anteriormente. Se definió la utilización de Twython como biblioteca para la conexión con Twitter, Django como *framework* de desarrollo, OpenUp como metodología de desarrollo, PyCharm como entorno de desarrollo integrado, Visual Paradigm como herramienta de modelado, Python como lenguaje de programación y PostgreSQL como Sistema Gestor de Base de Datos.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

Este capítulo tiene como propósito presentar la propuesta de solución del Sistema de Publicación Avanzada de *tweets*. En el mismo se definen los requisitos funcionales y no funcionales que deben complementar el Sistema para su aplicación, así como las clases y el modelo de dominio. Conjuntamente, se identifican y describen los actores, los casos de uso del Sistema, los patrones de arquitectura y diseño, diagramas de diseño y diagrama de despliegue; utilizando la metodología de desarrollo OpenUP junto al lenguaje de modelado UML.

Modelo de dominio

El modelo del dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Facilita en gran medida la comprensión del problema mediante un vocabulario común a los usuarios, clientes, desarrolladores e interesados en el desarrollo del Sistema. Muestra la relación entre los principales conceptos y contribuye a una correcta y eficiente captura de requisitos (Jacobson, 2000). En la Figura 1 se muestra el diagrama de modelo de dominio que representa los conceptos fundamentales y las relaciones que existen entre ellos.

Descripción de Clases del Modelo del Dominio

En este epígrafe se procede a describir cada una de las clases existentes en el Sistema de Publicación Avanzada de *tweets*.

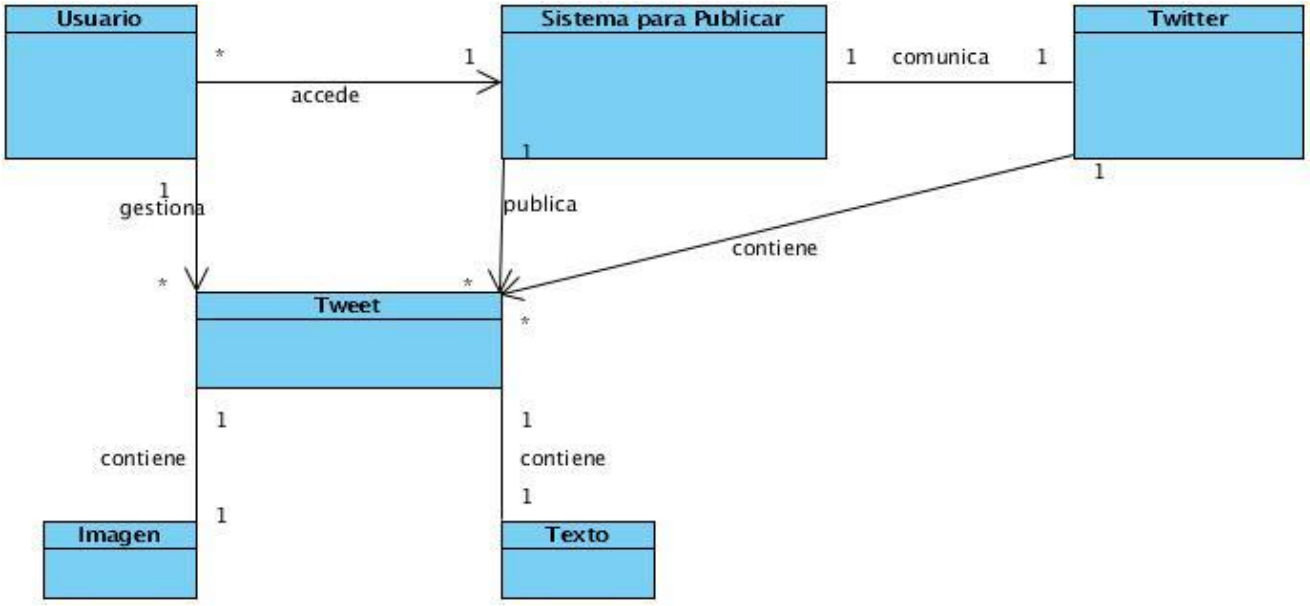


Figura 1: Modelo del Dominio

Seguidamente se puntualiza cada concepto del modelo de dominio utilizado en el desarrollo de este proyecto.

- Sistema para publicar: Sistema que almacena los *tweets* de los usuarios y los envía a la red social Twitter para su publicación.
- Twitter: Plataforma donde se visualizarán los *tweets* enviados por el Sistema para publicar.
- Usuario: Usuario que accede al Sistema para publicar.
- *Tweets*: Mensajes creados por los usuarios en el Sistema para publicar.
- Texto: Texto que contiene el *tweet* a publicar.
- Imagen: Imagen incluida por el usuario en el *tweet* a publicar.

Especificación de los requisitos del Sistema

El proceso conocido como obtención de requisitos, inicia la interacción con el cliente. Durante este proceso se realiza una relación de las necesidades que tiene el usuario y se define lo que debe hacer el Sistema.

Se describen las condiciones que se necesitan para que los requisitos funcionales puedan cumplirse y se detallan los usuarios que van a interactuar con la aplicación.

Requisitos funcionales

Estas son las declaraciones de los servicios que el Sistema debe proporcionar, cómo este debe reaccionar ante entradas particulares y cómo debe comportarse en situaciones específicas. En algunos casos, los requisitos funcionales también pueden declarar explícitamente lo que el Sistema no debe hacer (Sommerville, 2007). La Tabla 1 muestra el número de requisitos funcionales del Sistema de acuerdo con la prioridad del cliente.

Prioridad de cliente	Cantidad de requisitos funcionales
Alta	13
Media	2
Baja	0
Total	15

Tabla 1: Requisitos funcionales del Sistema

Listado de requisitos funcionales:

RF1. Autenticar usuario.

RF2. Mostrar lista de usuarios registrados en el Sistema.

RF3. Seleccionar usuarios para la planificación.

FR4. Insertar la hora y fecha de planificación.

RF5. Modificar la selección de los usuarios para la planificación.

RF6. Crear *tweet*.

RF7. Modificar la hora y fecha de planificación.

RF8. Modificar *tweet* planificado.

RF9. Eliminar *tweet* planificado.

RF10. Planificar *tweet*.

RF11. Mostrar *tweets* con error.

RF12. Enviar *tweets* con error.

RF13 Eliminar *tweets* con error.

RF14. Mostrar *tweets* planificados.

RF15. Publicar *tweet*.

Requisitos no funcionales

Según (Sommerville, 2007) los requisitos no funcionales son las limitaciones en los servicios o funciones que ofrece el sistema. Incluyen restricciones de tiempo, en el proceso de desarrollo y en las normas a utilizar. Estos a menudo se aplican al sistema como un todo. Por lo general, no sólo se aplican a características o servicios del sistema individuales. Seguidamente se realiza una clasificación de los requisitos no funcionales atendiendo a varios parámetros como son la eficiencia, la seguridad y el hardware.

Requerimientos de Usabilidad

RNF 1. El Sistema debe permitir a usuarios con pocos conocimientos de informática poder interactuar con el mismo.

RNF 2. La aplicación sólo podrá ser utilizada por los usuarios que estén registrados en el Sistema.

Requerimientos de Confiabilidad

RNF 3. La información no podrá ser modificada por usuarios no autorizados, protegiendo así la integridad de los datos.

Requerimientos de Soporte

RNF 4. El Sistema debe dar la posibilidad de ser actualizado, así como de incorporarle nuevas funcionalidades en caso de ser necesarias.

Eficiencia

RNF 5. Cantidad de peticiones concurrentes: el Sistema debe ser capaz de soportar un mínimo de 300 peticiones concurrentes.

Seguridad

RNF 6. El Sistema debe tener protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

Hardware

RNF 7. Para que el servidor realice todas las funcionalidades especificadas se debe poseer como requisitos mínimos para el mismo:

1. Memoria RAM de 2 GB.
2. Procesador de la familia INTEL u otro con 2.10 GHz de velocidad.
3. Espacio en disco duro de 40 GB.

RNF 8. Para el acceso a la aplicación, la estación de trabajo debe contar como mínimo con:

1. Memoria RAM de 256 MB.
2. Procesador de la familia INTEL u otro con una velocidad superior a 1.10 GHz.
3. Espacio en disco duro de 1 GB.

Software

RNF 9. Para acceder al Sistema se necesita tener instalado un navegador web.

Descripción del Sistema a desarrollar

El Sistema permitirá hacer publicaciones en la red social Twitter de manera automatizada, de ahí el título a este trabajo (Sistema de Publicación Avanzada de *tweets*). Este Sistema permitirá planificar las publicaciones de *tweets*, dada una fecha y hora determinada, la aplicación debe de interactuar

con los servicios que brinda Twitter y hacer dichas publicaciones de manera satisfactoria, de modo que se alcance una mayor eficiencia en el uso de las tecnologías y los recursos humanos.

Modelo de casos de uso del Sistema

El Sistema contará con un conjunto de actores que serán capaces de interactuar con él teniendo en cuenta los permisos que tengan asignados dichos actores. Para la realización de un sistema se debe hacer una correcta identificación de los requisitos que este debe cumplir. Esto se debe a que un proyecto no puede tener éxito sin una correcta especificación de los requerimientos que debe cumplir el sistema que se desea desarrollar. Definido más formalmente, un actor es algo que comunica con el producto final y que es externo al sistema en sí mismo (Pressman, 2005). A continuación se muestran los actores del Sistema a desarrollar (Ver Tabla 2).

Actor del Sistema	Descripción
Usuario	Es el usuario que accede al Sistema de Publicación Avanzada de *tweets* con su cuenta de Twitter.
Administrador	Es el usuario autorizado a realizar la gestión de la planificación, los *tweets* con error y el manejo de los usuarios registrados en la aplicación.

Tabla 2: Descripción de los actores del Sistema

Diagrama de Casos de Uso del Sistema

Los casos de uso sirven para especificar el comportamiento de un sistema mediante la interacción con los actores ya sean usuarios u otros sistemas. Los Diagramas de Caso de Uso son utilizados para ilustrar los requerimientos del sistema al mostrar cómo reacciona ante determinados eventos que se producen en su ámbito o en él mismo (Pressman, 2005), (Ver Figura 2).

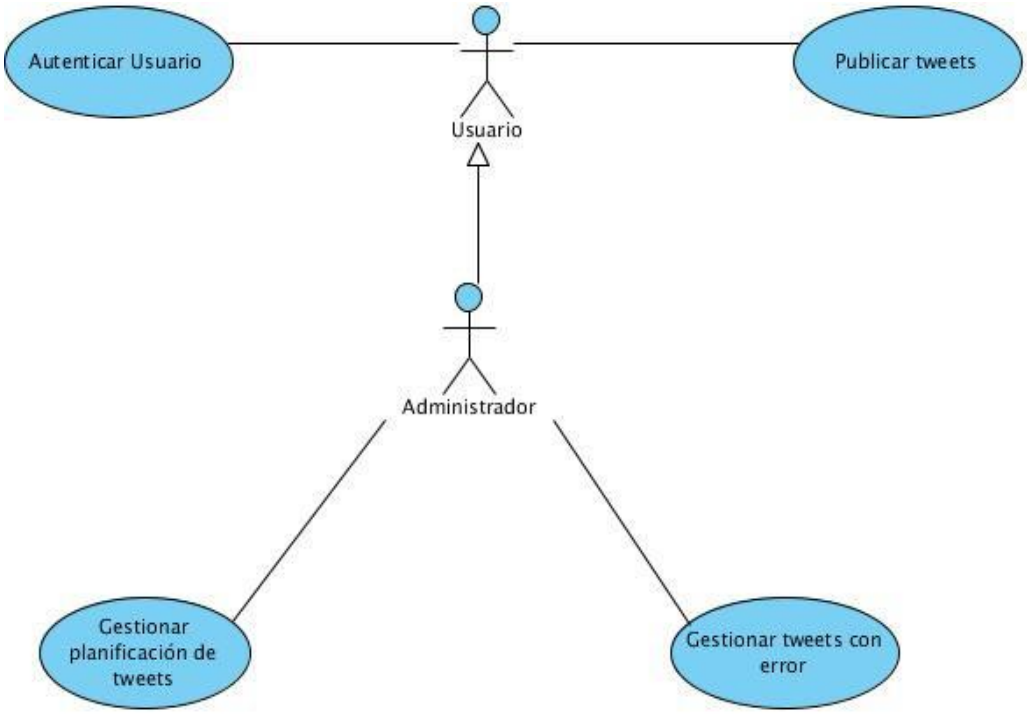


Figura 2: Diagrama de Casos de Usos del Sistema

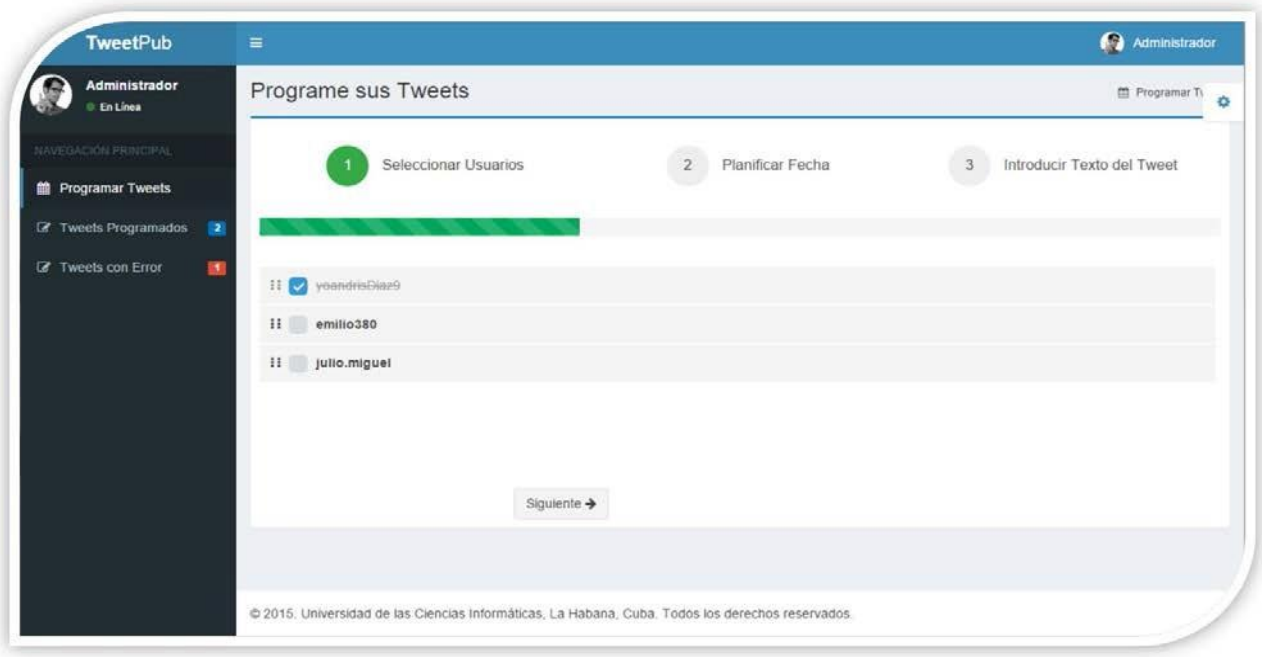
Descripción de Casos de Uso del Sistema

Al utilizar la metodología de desarrollo de *software* OpenUP, los casos de uso del Sistema son descritos en el artefacto Especificación de Casos de Uso. A continuación se describe uno de prioridad alta para el cliente (Ver Tabla 3).

CU Gestionar planificación de *tweets*

Objetivo	Gestionar la planificación de los *tweets*.	
Actores	Administrador.	
Resumen	El CU se inicia cuando el usuario quiere gestionar la planificación de los *tweets* a enviar, selecciona los usuarios a nombre de los cuales va a enviar el *tweet*. El Caso de Uso termina cuando se planifica dicho *tweet*.	
Complejidad	Alta	
Prioridad	Alta	

Precondiciones	El usuario está autenticado en el Sistema.	
Postcondiciones	Se realiza satisfactoriamente la planificación del *tweet*.	
Requisitos funcionales		
Flujo de eventos		
Flujo básico		
	Actor	Sistema
		Muestra listado de los usuarios registrados en el Sistema.
Sección 1: Programar *tweet*.		



Flujo básico		
	Actor	Sistema
1	El administrador selecciona los usuarios a nombre de quien va a publicar y da clic en la opción “Siguiente”.	
		El Sistema muestra una interfaz donde brinda las opciones de insertar hora, fecha, día de la publicación, eliminar estos datos o ir hacia “Atrás”.
2	El actor inserta hora, fecha, día de la publicación y da clic en la opción “Siguiente”.	
		El Sistema muestra una interfaz donde brinda las opciones de crear el *tweet*, permitiéndole solo un número máximo de 140 caracteres, dándole la opción además de subir una imagen, ir hacia “Atrás” o “Planificar”.
3	El actor crea el *tweet* con su imagen si desea y da clic en la opción “Planificar”.	
		El Sistema guarda los datos y re-direcciona a la plantilla donde se muestra la lista de usuarios para comenzar el proceso.
Flujos alternos		
	Actor	Sistema

1	El actor selecciona la opción siguiente sin seleccionar ningún usuario de la lista.	
		El Sistema muestra un mensaje de error “El formulario contiene errores” y no te permite continuar hasta que se haga correctamente el paso 1.
Flujos alternos		
	Actor	Sistema
2	El actor selecciona la opción “Atrás”	
		El Sistema muestra la interfaz para modificarlos usuarios seleccionados.
Flujos alternos		
	Actor	Sistema
3	El actor selecciona la opción “Atrás”	
		El Sistema muestra la interfaz para modificar la hora y fecha de planificación.
Flujos alternos		
	Actor	Sistema
		En caso de que al dar en la opción “Siguiente” y los datos sean incorrectos el Sistema muestra un mensaje de error “El formulario contiene errores” y no te permite continuar hasta que se haga correctamente el Paso 2.
Sección 2: Modificar *tweets* programados.		
Flujo básico		
	Actor	Sistema

	El administrador da clic en la opción "*Tweets* Programados".	
		El Sistema muestra la interfaz donde están los *tweets* planificados.
	Selecciona el *tweet* que desee y ejecuta las opciones "Modificar".	
		El Sistema re-direcciona a la interfaz anterior.
Sección 3: Eliminar *tweets* programados.		
Flujo básico		
	Actor	Sistema
	El administrador da clic en la opción "*Tweets* Programados".	
		El Sistema muestra la interfaz donde están los *tweets* planificados.
	Selecciona la opción "Eliminar".	
		El Sistema elimina el *tweet* de la lista.
Relaciones	RF2, RF3, FR4, RF5, RF6, RF7 RF8, RF9, RF10.	

Tabla 3: Descripción del Caso de Uso Gestionar planificación de *tweets*

Modelo de Diseño

El Modelo de Diseño ofrece una perspectiva de especificación o implementación, como quiere el modelador. Este modelo representa clases del Sistema a desarrollar (Larman, 2000).

Diagrama de Clases del Diseño

El Diagrama de Clases de Diseño (DCD) representa las especificaciones de las clases de *software* y de las interfaces de alguna aplicación. Entre la información que contienen de manera general se encuentran clases, asociaciones, atributos, métodos y dependencias (Larman, 2000), (Ver Figura 3).

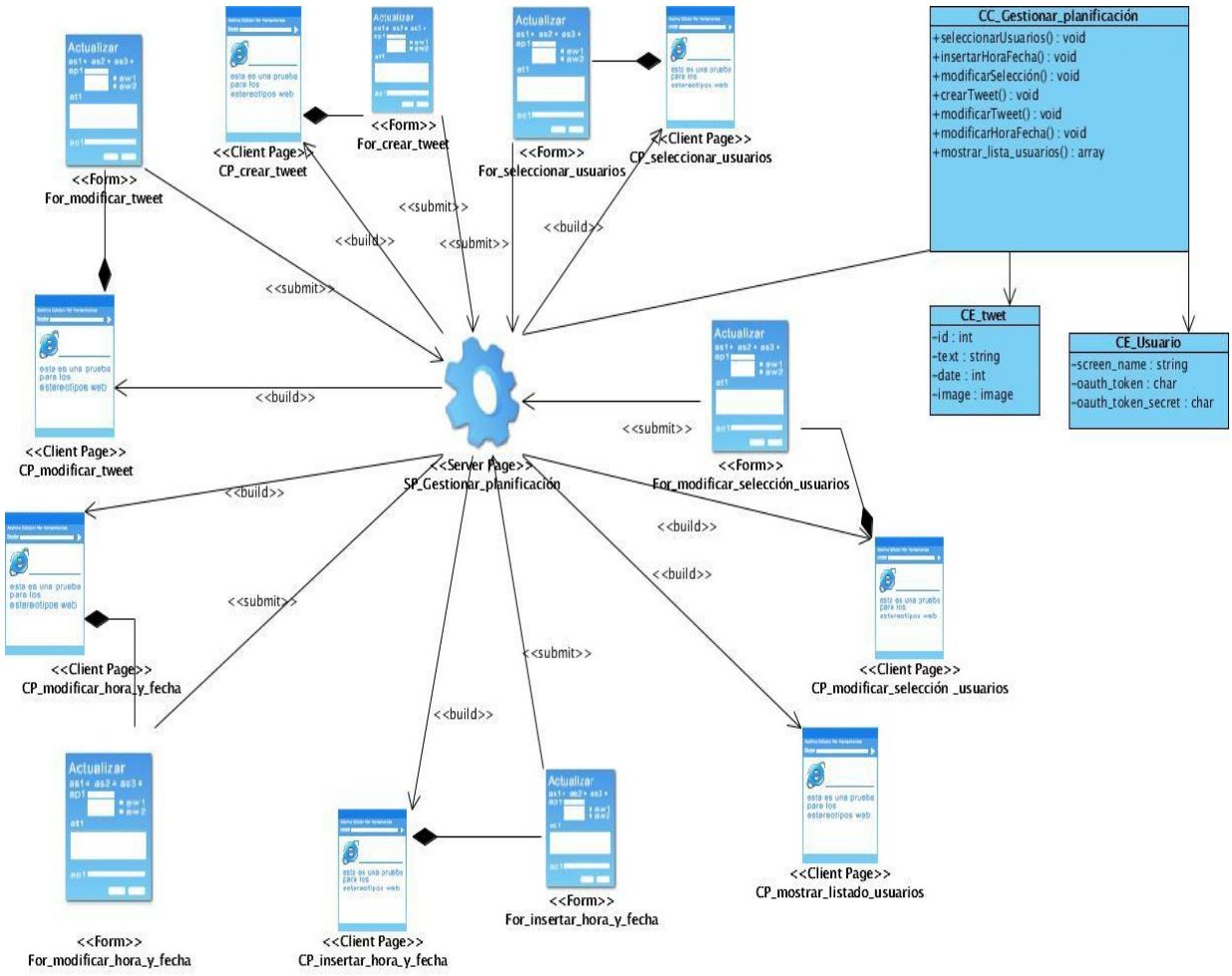


Figura 3: Diagrama de Clases del Diseño para el Caso de Uso Gestionar planificación de *tweets*

Patrones de diseño

Los patrones de diseño de *software* son los que permiten describir fragmentos de diseño y reutilizar ideas de diseño, ayudando a beneficiarse de la experiencia de otros. Los patrones dan nombre y forma a heurísticas abstractas, reglas y buenas prácticas de técnicas orientadas a objetos. Los patrones de diseño comunican los estilos y soluciones consideradas como buenas prácticas (Larman, 2000).

Como parte de los patrones de diseño utilizados para el desarrollo de TweetPub, se encuentran los patrones GRASP y GOF, que a continuación se explica cada uno de estos.

Patrones GRASP

Los Patrones Generales de *Software* para Asignación de Responsabilidades (GRASP por sus siglas en inglés) se consideran más que patrones propiamente dichos, son vistos como una serie de buenas prácticas de aplicación recomendable en el diseño de *software*.

Creador: Este patrón es el que crea, el que ayuda a identificar quién debe ser el creador de objetos y clases. Una clase B tiene la responsabilidad de crear un objeto de una clase A cuando la clase está contenida en la clase B, la clase B en una agregación o composición de la clase A, la clase B almacena a la clase A, la clase B inicializa los datos de la clase A y la clase B usa la clase A (Larman, 2000). En la presente investigación la clase *View* hace uso de las implementaciones de la clase Task, a la cual se accede mediante su respectiva instancia.

Bajo acoplamiento: El acoplamiento es el grado de relación que tienen los elementos de un sistema, dígase clases, módulos u otro elemento. El bajo acoplamiento es la idea de tener estos elementos o clases lo menos ligadas entre sí, de modo que de ocurrir una modificación en una de ellas se tenga la mínima repercusión posible en el resto de las clases. Significa que debe haber pocas dependencias entre las clases para ser más entendidas cuando estén aisladas, porque no tendrían tantas dependencias de otras clases (Larman, 2000). En la aplicación este patrón se evidencia en las clases del negocio, ya que cada una de ellas tiene la menor dependencia posible.

Alta cohesión: La cohesión es una medida de cuán relacionadas están las responsabilidades de una clase. Una alta cohesión permite a las clases que están muy relacionadas no realizar un enorme trabajo. Las clases que presentan baja cohesión son difíciles de comprender, reutilizar y conservar (Larman, 2000). En la aplicación cada clase tiene bien definidas sus responsabilidades, este patrón se hace evidente en la clase Task ya que esta hace uso de los métodos de la clase Service para evitar la sobrecarga de funcionalidades.

Experto: Se basa en la asignación de responsabilidades. La responsabilidad de realizar una labor es de la clase que tiene o puede tener la información necesaria. Se le asigna la responsabilidad al experto en información. Esto garantiza que el sistema sea más fácil de entender (Larman, 2000). En la aplicación este patrón se ve reflejado en las clases del negocio pues cada una tiene una función acorde con los datos que ella conoce.

Patrones GOF

El término Pandilla de Cuatro (GOF por sus siglas en inglés) son patrones generales de diseño que se utilizan en situaciones frecuentes, debido a que se basan en la experiencia acumulada al resolver problemas reiterativos. Con la aplicación de estos patrones se puede obtener un mejor diseño, trabajo con objetos y asignación de responsabilidades. Favorecen la reutilización de código. Ayudan a desarrollar *software* basados en la reutilización y a implementar clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar (Larman, 2000).

Descripción de la arquitectura

En la arquitectura de un *software*, los estilos definen la forma de organización arquitectónica de los subsistemas de información que conforman una aplicación. Estos indican los tipos de componentes y conectores involucrados, las restricciones de interconexión entre ellos, y el empleo de buenas prácticas para el diseño de un *software* (Torre, y otros, 2010). El estilo arquitectural de MTV (Modelo Template Vista) seleccionado para el desarrollo de este Sistema se basa en una modificación del MVC (Modelo Vista Controlador), debido a que los desarrolladores no tuvieron la intención de seguir algún patrón de desarrollo, sino hacer el *framework* lo más funcional posible.

Para empezar a entender MTV hay que fijarse en la analogía con MVC.

- El modelo en Django sigue siendo modelo (*Model*).
- La vista en Django se llama Plantilla (*Template*).
- El controlador en Django se llama Vista (*View*).

En la Figura 4 se muestra el funcionamiento de esta relación:

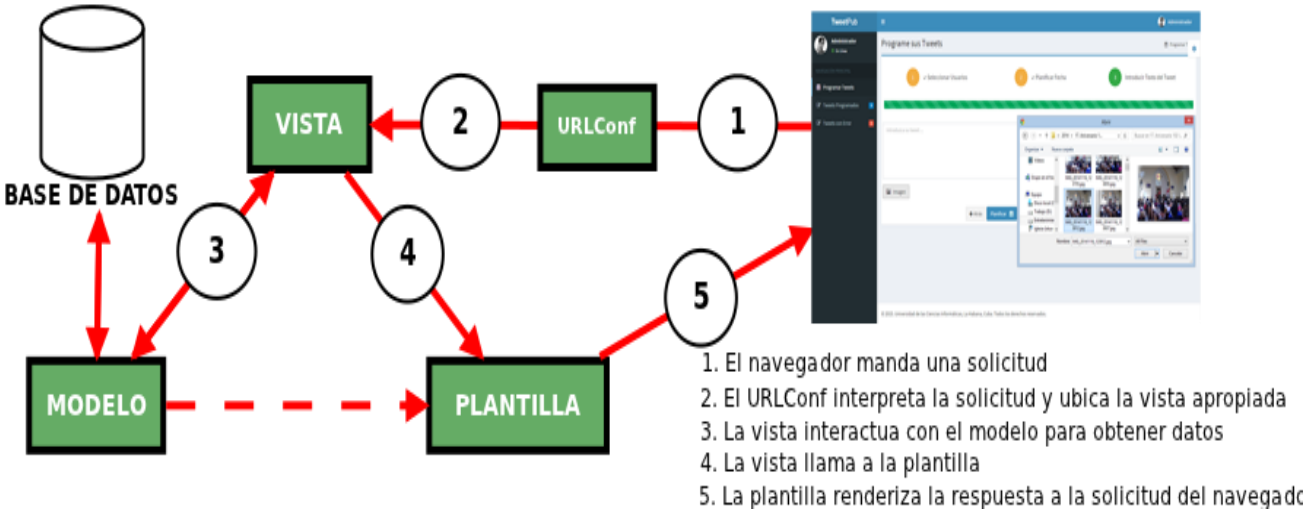


Figura 4: Funcionamiento de la arquitectura Modelo Plantilla Vista

Seguidamente se muestra qué hace cada uno de ellos con un poco más de detalles y algunos conceptos adicionales.

El modelo:

El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, y además posee varios métodos. Todo esto permite indicar y controlar el comportamiento de los datos.

La vista:

La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo más importante a entender respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

La plantilla:

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc). La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del *frontend*, incluso tiene estructuras de datos como *if*, por si es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python. Esto permite que la lógica del sistema siga permaneciendo en la vista.

Diseño de la Base de Datos

En la Figura 5 se muestra el Modelo de Base de Datos del Sistema que se desea desarrollar, que de una manera física y estructurada, muestra cómo se almacenan, relacionan, organizan y manipulan los datos del Sistema.

El Modelo de Base de Datos está compuesto por 5 tablas, donde 2 de ellas representan las relaciones mucho a mucho entre dos tablas, de las cuales su nombre comienza con el identificador ‘twtpub_’; una tabla donde se almacena los datos de los usuarios y una tabla para almacenar los datos de los *tweets* contenidos en la base de datos. Hay otras 2 tablas cuyo nombre empieza con el identificador auth_, una de ellas guarda los datos del administrador y la otra los datos asociados con los permisos del mismo.

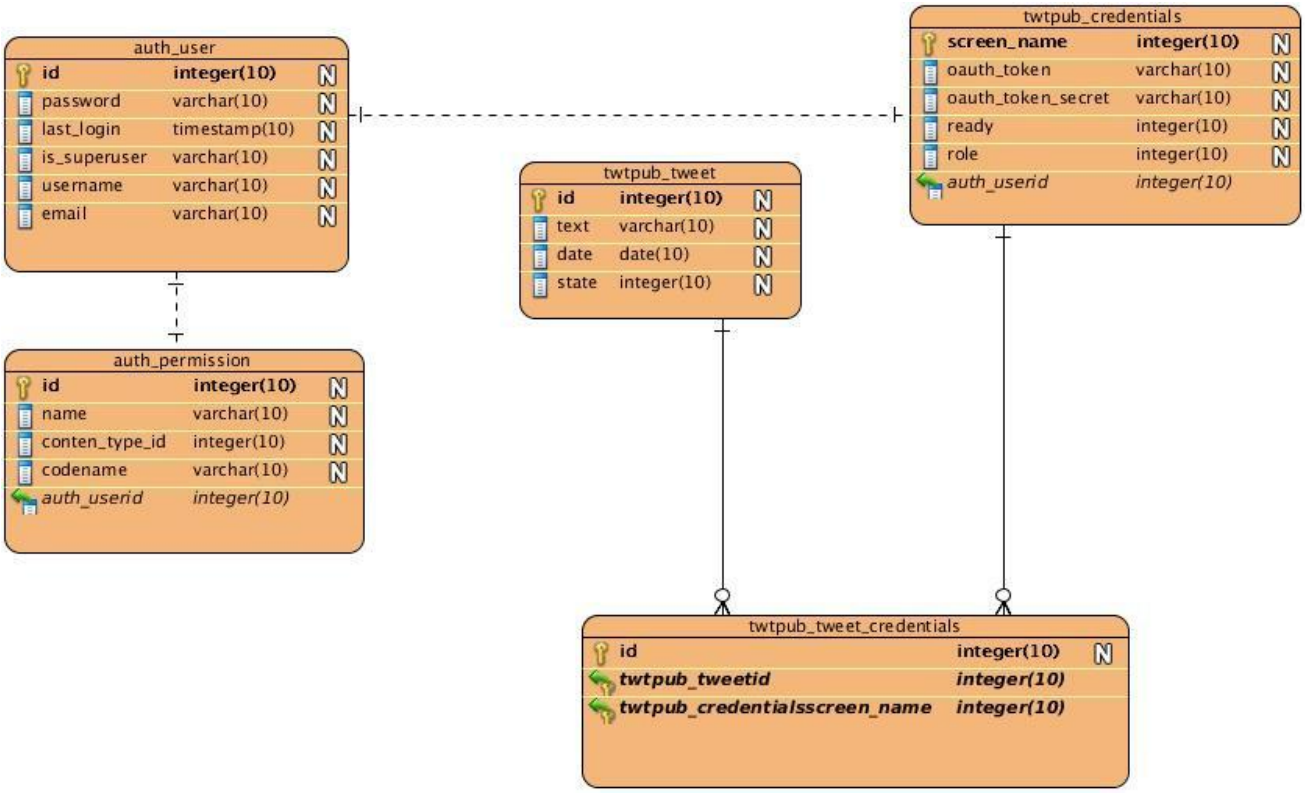


Figura 5: Colecciones de la base de datos PostgreSQL

Diagrama de despliegue

UML define varios diagramas que se pueden utilizar para ilustrar los detalles de implementación. El que se utiliza más comúnmente es un diagrama de despliegue, para ilustrar el despliegue de los componentes y procesos en los nodos de proceso. Muestran la disposición física del hardware que conforman la composición de un sistema y la distribución de los programas ejecutables en dicho hardware. Todo sistema se describe por un pequeño número de diagramas de despliegue y normalmente, uno solo es suficiente. Cada hardware es identificado como un nodo. Un nodo es un elemento donde se ejecutan los componentes, entre ellos existen relaciones que son representados como medios de comunicación tales como HTTP, TCP/IP, entre otros (Larman, 2000), (Ver Figura 6).

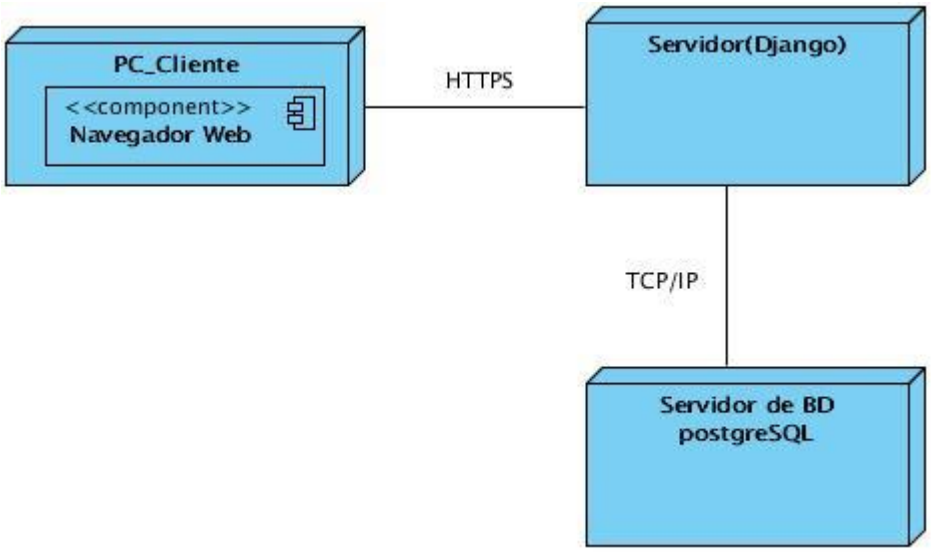


Figura 6: Diagrama de Despliegue

Descripción del diagrama de despliegue

PC Cliente: Representa la computadora a través de la cual el usuario puede consultar y actualizar la información que se encuentra en el Servidor Web. La comunicación entre las PC Clientes y el Servidor Web se establece utilizando el protocolo de comunicación HTTPS.

Servidor Django: Es el nodo que funciona de intermediario entre las PC clientes que hacen peticiones y el servidor donde se encuentra la información. Este Servidor Web toma los datos, realiza sus funciones y presenta la información a las PC Clientes. La comunicación entre el Servidor Web y el Servidor de Bases de Datos se establece utilizando el protocolo de comunicación TCP/IP.

Servidor de BD: Es el nodo que contiene toda la información del Sistema.

Conclusiones Parciales

En este capítulo se describió la propuesta de solución, así como las características del Sistema de Publicación Avanzada de *tweets*. Se realizó el modelo de dominio lo que facilitó en gran medida la comprensión del problema, logrando además, una mejor identificación de los requisitos funcionales y no funcionales. Se realizó el diagrama de casos de uso del Sistema, los diagramas de clases del diseño brindando una mejor comprensión del Sistema a desarrollar. Se seleccionó la arquitectura MTV y los patrones de diseño GRASP y GOF a utilizar durante la implementación para lograr una solución acorde a las necesidades del cliente. Todo esto permitió lograr una alta cohesión entre las capas lo cual posibilitará en caso de ser necesario modificar de forma sencilla las diferentes implementaciones.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

En este capítulo se podrá observar el diagrama de componentes del Sistema. Se exponen las características de la implementación del *software* a realizar. Se mostrarán algunas interfaces de la aplicación. También se describirán algunas pruebas realizadas al Sistema de Publicación Avanzada de *tweets*, pruebas que se realizan con el objetivo de detectar posibles no conformidades.

Diagrama de componentes

Los diagramas de componentes modelan la vista estática de un sistema, describiendo sus elementos físicos y sus relaciones. Los componentes representan todos los tipos de *software* que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc. Los elementos se agrupan en paquetes con vista a simplificar el modelado. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo que cada diagrama describe un apartado del sistema (Universidad de castilla-La mancha, 2014), (Ver Figura 7).

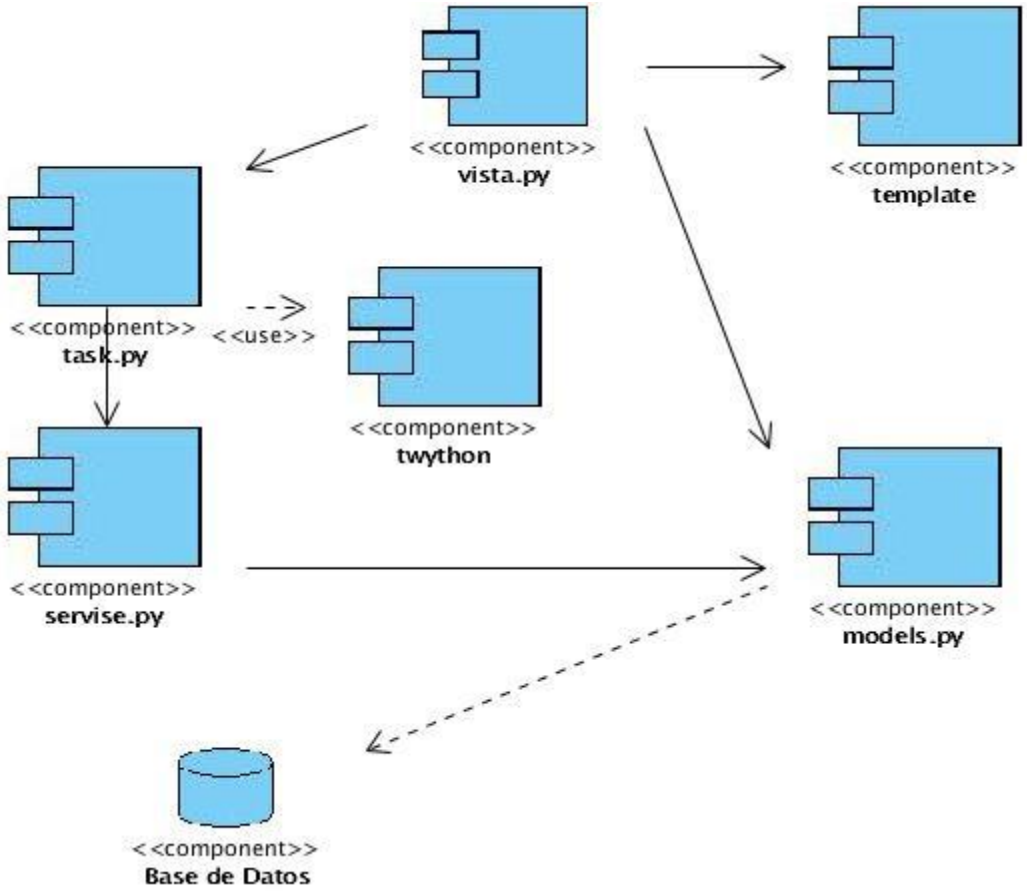


Figura 7: Diagrama de Componentes

En la siguiente tabla se describen los componentes modelados en el diagrama anterior:

Componentes	Descripción
Twython	En este paquete se encuentra la librería Twython que servirá de nexo entre la aplicación y la REST API de Twitter, para gestionar el intercambio de información entre ellas.
Tasks.py	En este paquete se encuentra la clase *Task*, una clase que contiene todas las tareas que serán ejecutadas por Celery, se encarga de manejar el flujo de datos desde y hacia Twitter haciendo uso de la librería Twython así como del manejo de las credenciales de los usuarios.
Services.py	Paquete que contiene la clase *Services* que es usada por la clase *Tasks* para el manejo de las credenciales y la usa también la clase *Views* para la conexión con la base de datos en PostgreSQL.
Models.py	Su función es la de gestionar los datos de las credenciales de los usuarios.
Views.py	Paquete que contiene la clase *Views* que hace función de controladora, es la que controla la lógica del negocio de la aplicación, en ella se capturan los datos enviados por los sistemas externos en formato json, seguido manda la petición a la clase Tasks para que ejecute una tarea determinada, según el servicio que sea solicitado y a su vez es la encargada de enviarle la respuesta en formato json a la aplicación que hizo la petición.
Base de Datos	Su función es la de almacenar los datos que maneja el Sistema TweetPub.

Tabla 4: Descripción de componentes del Sistema

Estándares de código

Los estándares de codificación establecen un conjunto de reglas que los desarrolladores deben seguir para escribir el código fuente de un *software*. Su utilización posibilita que el Sistema sea más legible y más fácil de mantener. Son pautas de programación que no están enfocadas a la lógica del programa, sino a la estructura y apariencia física, haciéndolo más reutilizable.

Seguidamente se verán varios estándares de codificación que presenta Python, y que además se ponen en práctica en el desarrollo del Sistema.

Estándares:

Tabuladores o espacios: Se utiliza la forma más popular de indentar en Python sólo espacios. El código indentado con una mezcla de tabuladores y espacios no es aconsejable.

Tamaño máximo de línea: Todas las líneas están limitadas a un máximo de 79 caracteres.

Líneas en blanco: Separa las funciones no anidadas y las definiciones de clases con dos líneas en blanco. Las definiciones de métodos dentro de una misma clase se separan con una línea en blanco. Se usan líneas en blanco extras, de forma reservada para separar grupos de funciones relacionadas.

Indentación: Consiste en insertar espacios en blanco o tabuladores en determinadas líneas de código para facilitar su comprensión. En Python la indentación usa 4 espacios por cada nivel de indentación.

Interfaces de la aplicación TweetPub

Figura 8: Interfaz de autenticación de usuarios

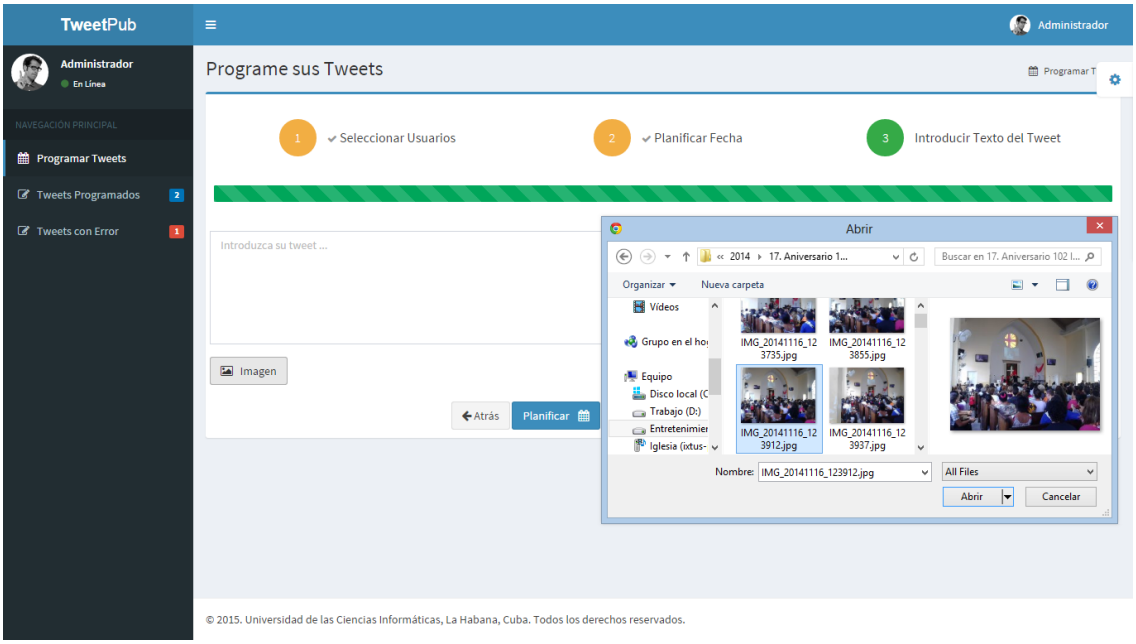


Figura 9: Interfaz administrativa de planificación de *tweets*

Pruebas

Las pruebas son un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática. Por tanto, se debe definir una plantilla para las Pruebas de *Software*. Las pruebas constituyen un factor importante en el proceso de desarrollo de un *software*. Estas permiten aumentar la calidad de los sistemas reduciendo el número de errores no detectados y el tiempo demorado en detectar los errores que puedan existir (Pressman, 2005).

Validación del Sistema

El flujo de trabajo de prueba de un sistema tiene como objetivo verificar el *software* para comprobar si cumple con sus requisitos. Además, se desarrollan distintos tipos de pruebas en función de los objetivos del Sistema. Los tipos de pruebas pueden ser funcionales, de aceptación, de seguridad, de rendimiento, entre otras. Las pruebas verifican que el Sistema ofrece a los actores las funcionalidades especificadas.

Pruebas de funcionalidad

Es una prueba que se aplica con el objetivo de detectar errores en el funcionamiento del sistema. Al realizarse las pruebas de funcionalidad para todos los casos de uso con entradas diferentes, se comprueba el cumplimiento o no de cada uno de los requisitos de *software* definidos en el capítulo anterior (Pressman, 2005). El objetivo final de estas pruebas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable.

A continuación se describe uno de los escenarios y los casos de prueba de prioridad alta para el cliente.

CP Publicar *tweet*.

Escenario	**Descripción**	**Respuesta del Sistema**	**Flujo central**
EC1.1 Publicar *tweet*.	El Sistema debe permitir al usuario publicar *tweets*.	El Sistema verifica las credenciales del usuario.	1- El Sistema brinda la posibilidad de autenticarse. 2- El usuario una vez autenticado, el Sistema le brinda la posibilidad de publicar. 3- El usuario introduce el texto del *tweet* y/o la imagen que desea subir y escoge la opción “Publicar”.
EC1.2 Publicar *tweet* con el campo de texto vacío.	El Sistema no publica el *tweet* si se deja en blanco el campo de texto.	Si el área donde va la fecha está vacía, el Sistema alerta al usuario con un mensaje de error “El formulario contiene error”.	

Tabla 5: Descripción del Caso de Prueba Publicar *tweet*

Estas pruebas fueron realizadas a la aplicación web, arrojando como resultado un total de 50 no conformidades en 3 iteraciones distribuidas de la forma que se presenta en la Figura 10. Las no conformidades encontradas se agruparon fundamentalmente en cuatro tipos: de funcionalidad, de validación y de idioma. La Figura 11 muestra el porcentaje que representaron las cuatro clasificaciones de no conformidades con respecto al total.

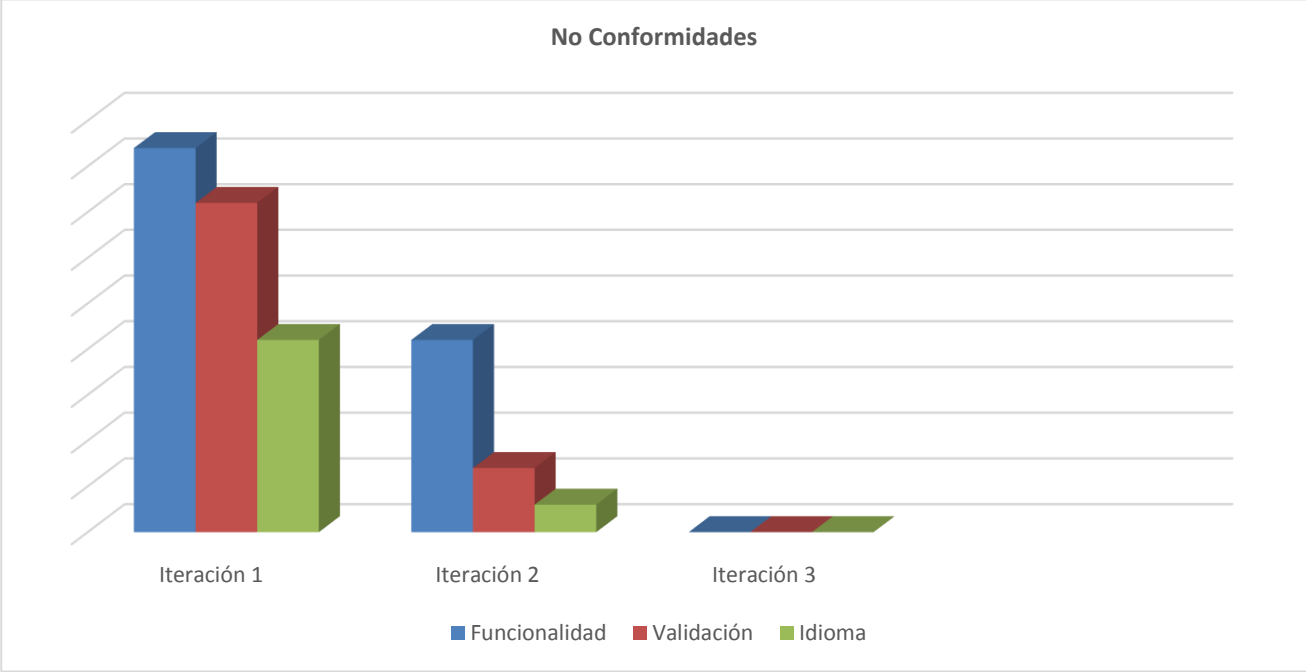


Figura 10: No Conformidades encontradas en la aplicación web distribuidas en tres iteraciones

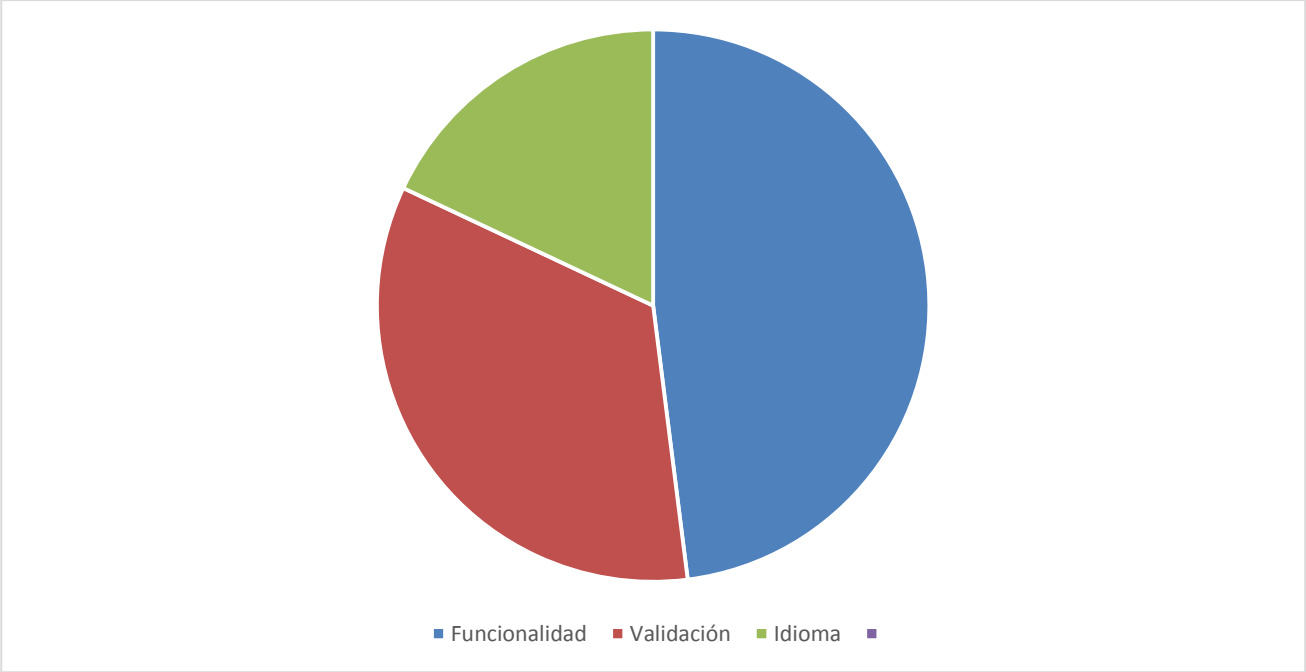


Figura 11: Porcentaje de las No Conformidades encontradas por clasificación

Pruebas de rendimiento

Mediante las pruebas de rendimiento es posible hallar tendencias y comportamientos para los elementos de una aplicación, los cuales generan bajo rendimiento. Este tipo de pruebas permiten identificar cuellos de botella, capacidad de concurrencia de usuarios, tiempos de respuesta de

operaciones de negocio a nivel de sistema, establecer un marco de referencia para pruebas futuras, determinar el cumplimiento de los objetivos de rendimiento y requerimientos no funcionales, entre otros (V&V Quality, 2013).

Pruebas de carga

Mediante la ejecución de las pruebas de carga es posible identificar la capacidad de recuperación de un sistema cuando es sometido a cargas variables, tanto de usuarios como de procesos. Al realizar las pruebas de carga se puede determinar el tiempo de respuesta de todas las transacciones críticas del sistema y encontrar cuellos de botella de la aplicación (V&V Quality, 2013).

Pruebas de estrés

Mediante las pruebas de estrés es posible identificar la capacidad de respuesta de un sistema bajo condiciones de carga extrema, representadas por una alta concurrencia de usuarios y/o procesos. Una vez realizadas las pruebas de estrés se podrá conocer el punto de quiebre del aplicativo en términos de capacidad de respuesta, con lo cual será posible establecer acciones de optimización en diferentes niveles para asegurar una mejor capacidad de concurrencia de usuarios y procesos, que se verá reflejada en una óptima operación de negocio (V&V Quality, 2013).

La realización de las pruebas de rendimiento se llevó a cabo con la herramienta JMeter, estas pruebas fueron divididas en test de 50, 100 y 150 hilos cada uno, los cuales simulan la cantidad de usuarios que acceden a las funcionalidades concurrentemente. Se tiene en cuenta el escenario donde se encuentra la aplicación y las condiciones del hardware y *software* de la computadora donde es ejecutado el muestreo; atendiendo a las siguientes características:

- Hardware: Posee una memoria RAM de 2 GB, procesador Intel Core 2 duo a 2.10 GHz y una capacidad de disco duro de 160 GB.
- *Software*: Tiene instalado como plataforma de desarrollo Windows en su versión 8.0.

Se utilizó el componente “Informe Agregado", el cual permite visualizar los resultados del muestreo a algunas funcionalidades del Sistema de Publicación Avanzada de *tweets*, obteniendo resultados más precisos.

Para una concurrencia de 50 usuarios se realizaron 200 peticiones al servidor obteniéndose un tiempo mínimo de respuesta de 35 segundos, con un margen de error del 0.0 % y un rendimiento promedio de 24.5 segundos (ver Figura 10 en los anexos). Para 100 usuarios concurrentes se ejecutaron 400 peticiones al servidor con un tiempo mínimo de respuesta de 32 segundos, obteniendo un margen de error de 0.0 % y un rendimiento promedio de 23.7 segundos (ver Figura 11 en los anexos). Y para un total de 150 usuarios concurrentes se efectuaron 600 peticiones al servidor con un tiempo mínimo de

respuesta de 25 segundos, un margen de error de 0.0 % y un rendimiento promedio de 17.8 segundos para el número de peticiones realizadas (ver Figura 12 en los anexos).

Conclusiones del Capítulo

En este capítulo se modelaron los diagramas de componentes, lo que mejoró la comprensión de cómo está integrada la aplicación. Se logró una mayor legibilidad, limpieza y organización del código, todo esto gracias a los estándares de codificación utilizados. También se realizaron una serie de pruebas que permitieron la detección de errores que posteriormente fueron solucionados, todo esto arrojó como resultado un mejor desempeño de la aplicación.

Conclusiones Generales

El estudio de los sistemas existentes de publicación de *tweets* en Twitter, facilitó la selección de las herramientas y las tecnologías a utilizar para el desarrollo del Sistema de Publicación Avanzada de *tweets*.

El análisis de las funcionalidades del Sistema y la utilización de los diferentes patrones de diseño y arquitectura permitieron diseñar una solución considerando las buenas prácticas de desarrollo de *software*.

El producto funcional obtenido a partir de la implementación de las clases del diseño permitió publicar en la red social Twitter de manera eficiente, logrando un menor consumo de ancho de banda de internet.

Se realizaron las pruebas a nivel de desarrollador que permitieron identificar y corregir los errores del Sistema, así como verificar la correcta implementación de las funcionalidades del mismo.

El Sistema implementado permitirá publicar *tweets* en la red social Twitter. Permitirá además, planificar dichos *tweets* para ser publicados en una fecha determinada, a nombre de varios usuarios, dándole cumplimiento así a los requisitos propuestos por el cliente.

Recomendaciones

Continuar el estudio de la API de Twitter ya que frecuentemente las mismas son actualizadas pudiendo influir de manera directa en el funcionamiento del presente Sistema.

Continuar el desarrollo del Sistema de Publicación Avanzada de *tweets*, pues al mismo se le pueden agregar otras funcionalidades que mejoren su apariencia y rendimiento.

Publicar el sitio en Internet para facilitar la comunicación con Twitter.

Implementar la planificación de *tweets* para el usuario, ya que actualmente se encuentra disponible sólo para el administrador.

REFERENCIAS

Acuña, Karenny Brito. 2009. *Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos.* Cienfuegos : s.n., 2009.

Alonzo Velázquez, José Luis. 2010. *Lenguaje de Programación: Introducción a C/C++(IDE).* 2010.

Balduino, Ricardo. 2007. [En línea] 2007. [Citado el: 11 de diciembre de 2014.] http://www.eclipse.org/epf/general/OpenUP.pdf.

Bennett, James. 2009. *Practical Django Projects.* 2009.

Corso, Cynthia Lorena. 2007. *Lenguaje de Programación Python.* 2007.

Donostiarra, Editorial. 2014. Editorial Donostiarra. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] http://www.editorialdonostiarra.com/pdf/389_UD-11_TIC.pdf.

Dougherty, Dale. 2004. educaLAB. [En línea] 2004. [Citado el: 10 de diciembre de 2014.] http://www.ite.educacion.es/formacion/materiales/155/cd/modulo_1_Iniciacionblog/concepto_de_web20. html.

Gómez, Sigifredo Escobar y Zapata, Omar Andrés. 2011. *Programación Científica en Phyton.* 2011.

Hernández Orallo, Enrique. 2010. *El Lenguaje Unificado de Modelado.* 2010.

Hernández, Yulainne Alonso. 2013. [En línea] 2013. [Citado el: 11 de diciembre de 2014.] http://www.somosjovenes.cu/sites/default/files/cidi_configuracion_de_la_metodologia_open_up.pdf.

HitHup. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/bear/python-twitter.

—. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/ryanmcgrath/twython.

—. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/tweepy/tweepy.

Jacobson. 2000. *El Proceso Unificado de Desarrollo de Software.* 2000.

JetBrains. 2011. JetBrains s.r.o. [En línea] 2011. [Citado el: 20 de enero de 2015.] http://www.jetbrains.com/pycharm/index.html.

Larman, Craig. 2000. *UML y Patrones.* Primera. 2000.

—. 2003. *UML y Patrones.* Segunda. 2003.

Martínez, Antonia Cascales, García, José Julio Real y Benito, Benedicto Marcos. 2011. *LAS REDES SOCIALES EN INTERNET.* s.l. : EDUTEC, 2011.

Martinez, Rafael. 2009. PostgreSQL-es. [En línea] 2009. [Citado el: 10 de diciembre de 2014.] http://www.postgresql.org.es/.

Martínez, Rafael. 2010. PostgreSQL-es. [En línea] 2 de octubre de 2010. [Citado el: 11 de diciembre de 2014.] http://www.postgresql.org.es/sobre_postgresql.

MongoDB. 2011. MongoDB. [En línea] 2011. [Citado el: 20 de enero de 2015.] https://www.mongodb.org/.

Pacheco, Nacho. 2011. *Manual de Symfony2.* 2011.

Potencier, Fabien. 2009. *El tutorial Joobeet.* 2009.

Pressman, Roger. 2005. *Ingeniería de Software: Un enfoque práctico.* [ed.] Concepción Femández Madrid. España : s.n., 2005.

Ribes, Xavier. 2007. *La Web 2.0. El valor de los metadatos y de la inteligencia colectiva.* s.l. : TELOS, 2007.

Robles, Gregorio. 2002. Programación eXtrema y Software Libre. [En línea] 10 de octubre de 2002. [Citado el: 11 de diciembre de 2014.] http://es.tldp.org/Presentaciones/200211hispalinux/ferrer/robles-ferrer-ponencia-hispalinux-2002.html.

Romero Guillén, Paola. 2004. *Análisis y Diseño Orientado a Objetos.* 2004.

Rose, Rational. 2007. La Plataforma de Desarrollo Rational. [En línea] 2007. [Citado el: 20 de enero de 2015.] http://www.rational.com.ar/herramientas/herramientasrational.html.

Sommerville, Ian. 2007. *Software Engineering.* 2007. ISBN.

Stefan Zerbst, Oliver Düvel. 2004. *3D Game Engine Programming.* 2004. pág. 860.

Torre, César de la, y otros. 2010. *Guía de Arquitectura N-Capas orientada al Dominio con .NET.* 2010. ISBN.

Twitter. 2014. Twitter Developers. [En línea] 2014. [Citado el: 10 de diciembre de 2014.] https://dev.twitter.com/rest/public.

UML, Visual Paradigm for. 2010. Visual Paradim. [En línea] 2010. [Citado el: 20 de enero de 2015.] http://www.visual-paradigm.com.

Universidad de castilla-La mancha. 2014. Departamento de Sistemas Informáticos. [En línea] 2014. [Citado el: 2 de febrero de 2015.] http://www.dsi.uclm.es/.

Ureña, Carlos. 2011. *Lenguajes de Programación.* 2011.

V&V Quality. 2013. V&V Quality. [En línea] 2013. [Citado el: 2 de febrero de 2015.] http://www.vyvquality.com.

BIBLIOGRAFÍA

Acuña, Karenny Brito. 2009. *Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos.* Cienfuegos : s.n., 2009.

Alonzo Velázquez, José Luis. 2010. *Lenguaje de Programación: Introducción a C/C++(IDE).* 2010.

Balduino, Ricardo. 2007. [En línea] 2007. [Citado el: 11 de diciembre de 2014.] http://www.eclipse.org/epf/general/OpenUP.pdf.

Bennett, James. 2009. *Practical Django Projects.* 2009.

Corso, Cynthia Lorena. 2007. *Lenguaje de Programación Python.* 2007.

Donostiarra, Editorial. 2014. Editorial Donostiarra. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] http://www.editorialdonostiarra.com/pdf/389_UD-11_TIC.pdf.

Dougherty, Dale. 2004. educaLAB. [En línea] 2004. [Citado el: 10 de diciembre de 2014.] http://www.ite.educacion.es/formacion/materiales/155/cd/modulo_1_Iniciacionblog/concepto_de_web20. html.

Frentzen, Jeff y Sobotka, Henry. 2005. *Superutilidades para Java Script.* 2005. pág. 551.

Gómez, Sigifredo Escobar y Zapata, Omar Andrés. 2011. *Programación Científica en Phyton.* 2011.

Hernández Orallo, Enrique. 2010. *El Lenguaje Unificado de Modelado.* 2010.

Hernández, Yulainne Alonso. 2013. [En línea] 2013. [Citado el: 11 de diciembre de 2014.] http://www.somosjovenes.cu/sites/default/files/cidi_configuracion_de_la_metodologia_open_up.pdf.

HitHup. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/bear/python-twitter.

—. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/ryanmcgrath/twython.

—. 2014. HitHup. [En línea] 2014. [Citado el: 11 de diciembre de 2014.] https://github.com/tweepy/tweepy.

Hogl, Hubert. 2011. *Intro to Python.* 2011. pág. 32.

Jacobson. 2000. *El Proceso Unificado de Desarrollo de Software.* 2000.

JetBrains. 2011. JetBrains s.r.o. [En línea] 2011. [Citado el: 20 de enero de 2015.] http://www.jetbrains.com/pycharm/index.html.

Larman, Craig. 2000. *UML y Patrones.* Primera. 2000.

—. 2003. *UML y Patrones.* Segunda. 2003.

—. 2004. *UML y Patrones.* 2004. pág. 499.

Martínez, Antonia Cascales, García, José Julio Real y Benito, Benedicto Marcos. 2011. *LAS REDES SOCIALES EN INTERNET.* s.l. : EDUTEC, 2011.

Martinez, Rafael. 2009. PostgreSQL-es. [En línea] 2009. [Citado el: 10 de diciembre de 2014.] http://www.postgresql.org.es/.

Martínez, Rafael. 2010. PostgreSQL-es. [En línea] 2 de octubre de 2010. [Citado el: 11 de diciembre de 2014.] http://www.postgresql.org.es/sobre_postgresql.

MongoDB. 2011. MongoDB. [En línea] 2011. [Citado el: 20 de enero de 2015.] https://www.mongodb.org/.

Pacheco, Nacho. 2011. *Manual de Symfony2.* 2011.

Palach, Jan. 2014. *Parallel Programming with Python.* 2014. ISBN.

Pérez, Javier Eguíluz. 2008. *Introduccion Javascript.* 2008. pág. 134.

Postgresql, El Equipo De Desarrollo De. 2001. *Guia del Administrador de PostgreSQL.* 2001. pág. 147.

Potencier, Fabien. 2009. *El tutorial Joobeet.* 2009.

Powell, Thomas A. 2004. *Diseño de sitios web.* 2004. pág. 857.

Pressman, Roger. 2005. *Ingeniería de Software: Un enfoque práctico.* [ed.] Concepción Femández Madrid. España : s.n., 2005.

Ribes, Xavier. 2007. *La Web 2.0. El valor de los metadatos y de la inteligencia colectiva.* s.l. : TELOS, 2007.

Robles, Gregorio. 2002. Programación eXtrema y Software Libre. [En línea] 10 de octubre de 2002. [Citado el: 11 de diciembre de 2014.] http://es.tldp.org/Presentaciones/200211hispalinux/ferrer/robles-ferrer-ponencia-hispalinux-2002.html.

Romero Guillén, Paola. 2004. *Análisis y Diseño Orientado a Objetos.* 2004.

Rose, Rational. 2007. La Plataforma de Desarrollo Rational. [En línea] 2007. [Citado el: 20 de enero de 2015.] http://www.rational.com.ar/herramientas/herramientasrational.html.

Sampieri, Roberto Hernández, Collado, Carlos Fernández y Lucio, Pilar Baptista. 2005. *Metodologia de la Investigacion Cientifica.* 2005. pág. 517.

Sommerville, Ian. 2007. *Software Engineering.* 2007. ISBN.

Stefan Zerbst, Oliver Düvel. 2004. *3D Game Engine Programming.* 2004. pág. 860.

Torre, César de la, y otros. 2010. *Guía de Arquitectura N-Capas orientada al Dominio con .NET.* 2010. ISBN.

Twitter. 2014. Twitter Developers. [En línea] 2014. [Citado el: 10 de diciembre de 2014.] https://dev.twitter.com/rest/public.

UML, Visual Paradigm for. 2010. Visual Paradim. [En línea] 2010. [Citado el: 20 de enero de 2015.] http://www.visual-paradigm.com.

Universidad de castilla-La mancha. 2014. Departamento de Sistemas Informáticos. [En línea] 2014. [Citado el: 2 de febrero de 2015.] http://www.dsi.uclm.es/.

Ureña, Carlos. 2011. *Lenguajes de Programación.* 2011.

V&V Quality. 2013. V&V Quality. [En línea] 2013. [Citado el: 2 de febrero de 2015.] http://www.vyvquality.com.

GLOSARIO DE TÉRMINOS

***API*:** Interfaz de Programación de Aplicaciones.

***CASE*:** *Computer Aided Software Engineering*, por sus siglas en inglés, en español se refiere a la Ingeniería de Software Asistida por Ordenador.

Celery: *Celery* es una aplicación que permite crear tareas de trabajo asíncronas gestionadas por un gestor de colas que está basada en el envío de mensajes de manera distribuida. Se focaliza en operaciones en tiempo real pero también soporta la calendarización de tareas, es decir, puede lanzar tareas que se tengan que ejecutar en un momento determinado o de manera periódica.

***Framework*:** Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

HTTP: *Hypertext Transfer Protocol* por sus siglas en Inglés, en español Protocolo de Transferencia de Hipertexto. Es el protocolo usado en cada transacción de la *World Wide Web*.

HTTPS: *Hypertext Transfer Protocol Secure* por sus siglas en Inglés, en español Protocolo Seguro de Transferencia de Hipertexto. Es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP.

Informática: Disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales.

OAuth2.0: *Open Authorization*, por sus siglas en Inglés, en español Abrir Autorización. Es un protocolo de autorización que permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario.

PHP: *Hypertext Pre-Processor*, por sus siglas en Inglés, en español Preprocesador de Hipertexto.

RUP: *Rational Unified Process*, por sus siglas en inglés, en español Proceso Unificado del Rational. Es una metodología de desarrollo de software tradicional, actualmente propiedad de IBM. Junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

***Software*:** Se refiere al equipamiento lógico o soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas.

TCP/IP: Se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron dos de los primeros en definirse.

***UML*:** *Unified Modeling Language* dadas sus siglas en Inglés, en español Lenguaje Unificado de Modelado. Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.

***Yaml*:** *Yet Another Markup Language* sus siglas en inglés, en español Otro Lenguaje de Marcado más. YAML es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML.

ANEXOS

Especificación de Casos de Uso del Sistema.

CU Autenticar Usuario

Objetivo	Entrar al sistema	
Actores	Usuario.	
Resumen	El CU se inicia cuando el Sistema le pide los datos del usuario y el usuario se autentica. El Caso de Uso termina cuando el Sistema valida los datos y permite el acceso al usuario.	
Complejidad	Media	
Prioridad	Alta	
Precondiciones		
Postcondiciones	Que el usuario acceda el sistema satisfactoriamente.	
Prototipo		

<table>
<tr><td colspan="3"> 
</td></tr>
<tr><td>Flujo de eventos</td><td></td><td></td></tr>
<tr><td colspan="3">Flujo básico</td></tr>
<tr><td></td><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>El usuario introduce sus datos.</td><td></td></tr>
<tr><td></td><td></td><td>El Sistema valida los datos para verificar que sean correctos, si son válidos muestra la interfaz principal en caso de que sea administrador y termina el Caso de Uso. En caso de que sea usuario común el Sistema le muestra una interfaz donde le da la opción de dar clic en un enlace “Aquí” para que obtenga el pin y la opción de introducir el pin en el formulario.</td></tr>
<tr><td colspan="3">Flujos alternos</td></tr>
<tr><td></td><td>Actor</td><td>Sistema</td></tr>
</table>

		Si las credenciales del usuario son incorrectas el Sistema muestra un mensaje “credenciales no válidas”.
Flujo básico		
	Actor	Sistema
	El usuario da clic en la opción “Aquí”.	
		El Sistema redirecciona hacia una interfaz para que se autentique.
	El usuario se autentica.	
		Muestra un pin.
	Copia el pin que se le muestra en la interfaz, lo pega en la anterior y da clic en el botón “Autorizar la Aplicación”.	
		El Sistema muestra la interfaz para crear el *tweet*. Termina el Caso de Uso.
Relaciones	RF1	

CU Publicar *tweets*

Objetivo	Publicar un *tweet.*	
Actores	Usuario.	
Resumen	El CU se inicia cuando el usuario quiere crear un *tweet*. El Caso de Uso termina cuando el usuario da clic en el botón “Publicar”.	

Complejidad	Alta	
Prioridad	Alta	
Precondiciones	El usuario debe estar autenticado en el Sistema.	
Postcondiciones	Que el mensaje (*tweet*) se publique satisfactoriamente.	
Prototipo		
		
Flujo de eventos		
Flujo básico		
	Actor	Sistema
	El usuario una vez en la interfaz de crear *tweets*, introduce el texto en al capo y selecciona la opción “Publicar”.	

		El Sistema valida los datos para verificar que sean correctos, si son válidos y no hay ningún otro problema, muestra un mensaje “El *tweet* se ha publicado satisfactoriamente”.
Relaciones	RF15.	

CU Gestionar *tweets* con error

Objetivo	Gestionar los *tweets* con errores.	
Actores	Administrador.	
Resumen	El CU se inicia cuando el administrador quiere hacer alguna tarea con los *tweets* que no se han publicado por errores y selecciona la opción “*Tweets* con error”. El Caso de Uso termina cuando el Sistema ejecuta la orden sin mostrar ningún mensaje de error.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	Que haya *tweet* con errores.	
Postcondiciones	El Sistema ejecuta la orden sin mostrar ningún mensaje de error.	
Prototipo		
Flujo de eventos		

Flujo básico		
	Actor	Sistema
	El administrador da clic en la opción "*Tweets* con Error".	
		El Sistema muestra la interfaz donde está la lista de *tweets* con errores.
Sección 1: Enviar *tweet*.		
Flujo básico		
	Actor	Sistema
	Selecciona la opción "Enviar".	
		Envía el *tweet*.
Sección 2: Eliminar *tweet*.		
Flujo básico		
	Actor	Sistema
	Selecciona la opción "Eliminar".	
		Elimina el *tweet* de la lista de *tweets* planificados.
Relaciones	RF12, RF13	

Diagramas de Clases de Diseño con Estereotipos Web.

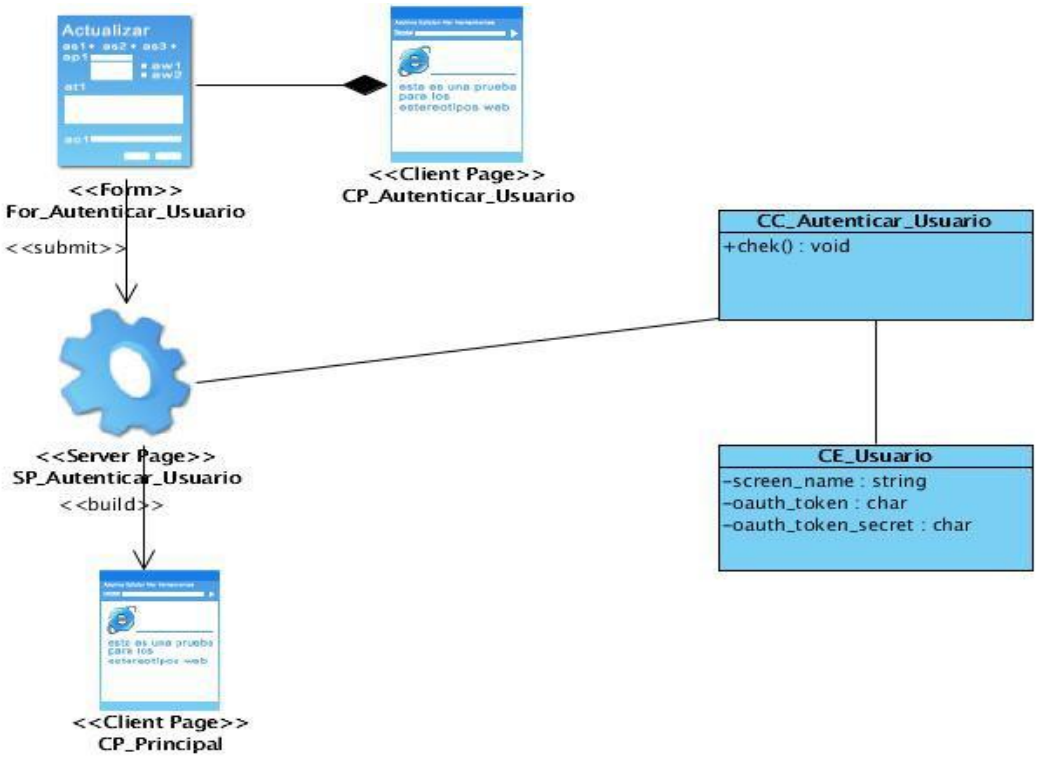


Figura 1: CU Autenticar Usuario

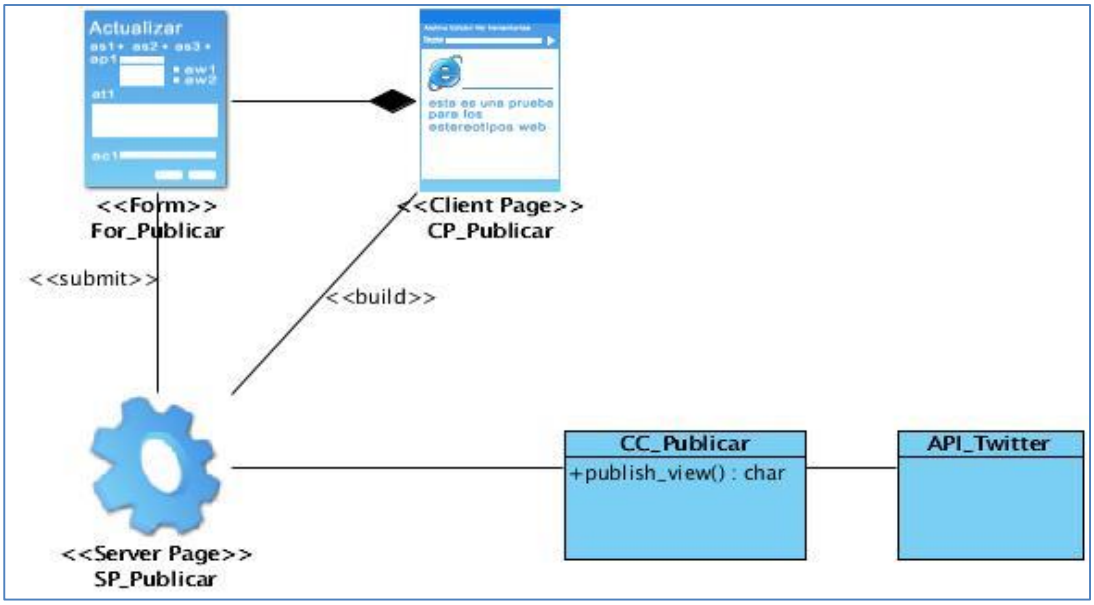


Figura 2: CU Publicar *tweet*

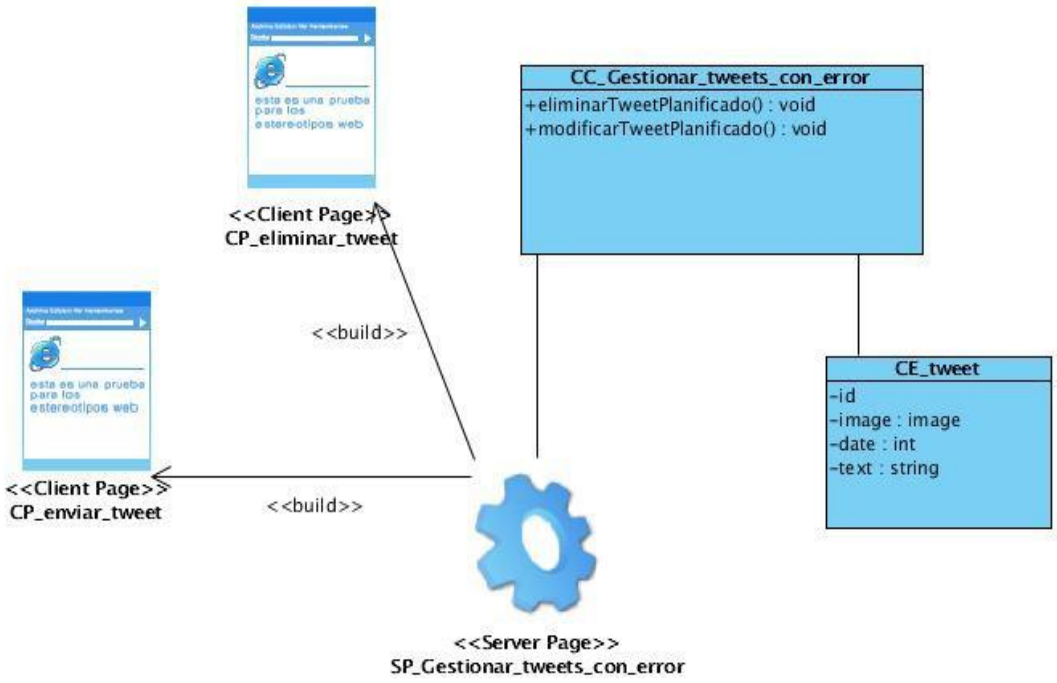


Figura 3: CU Gestionar *tweets* con error

Vistas del Sistema de Publicación Avanzada de *tweets*.

Figura 4: Vista de solicitud del PIN de autorización

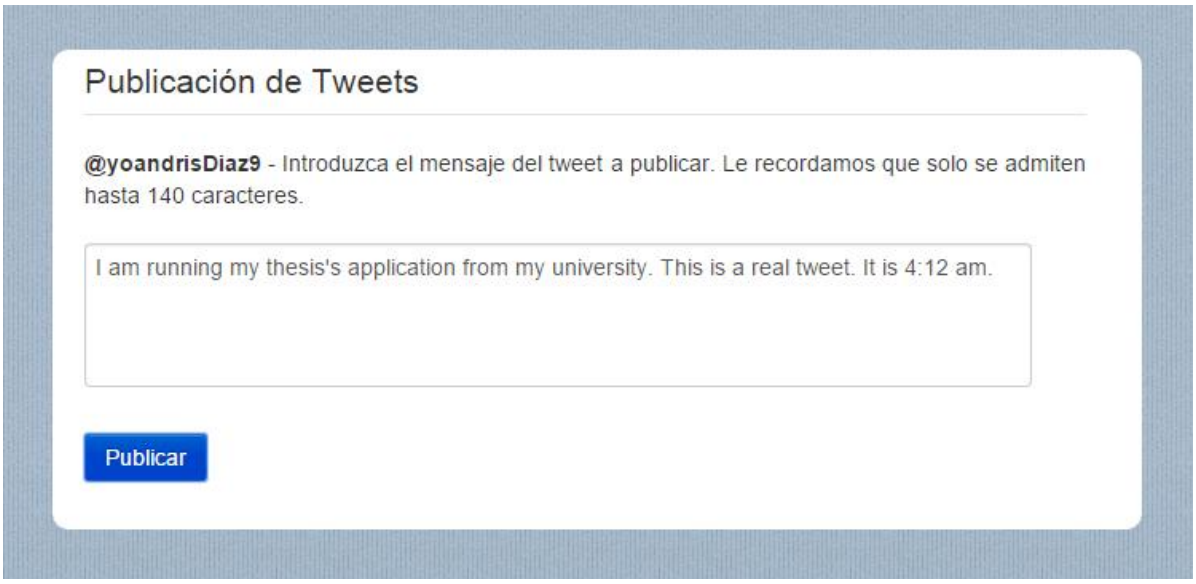


Figura 5: Vista de publicación de *tweets* para el usuario autenticado con Twitter

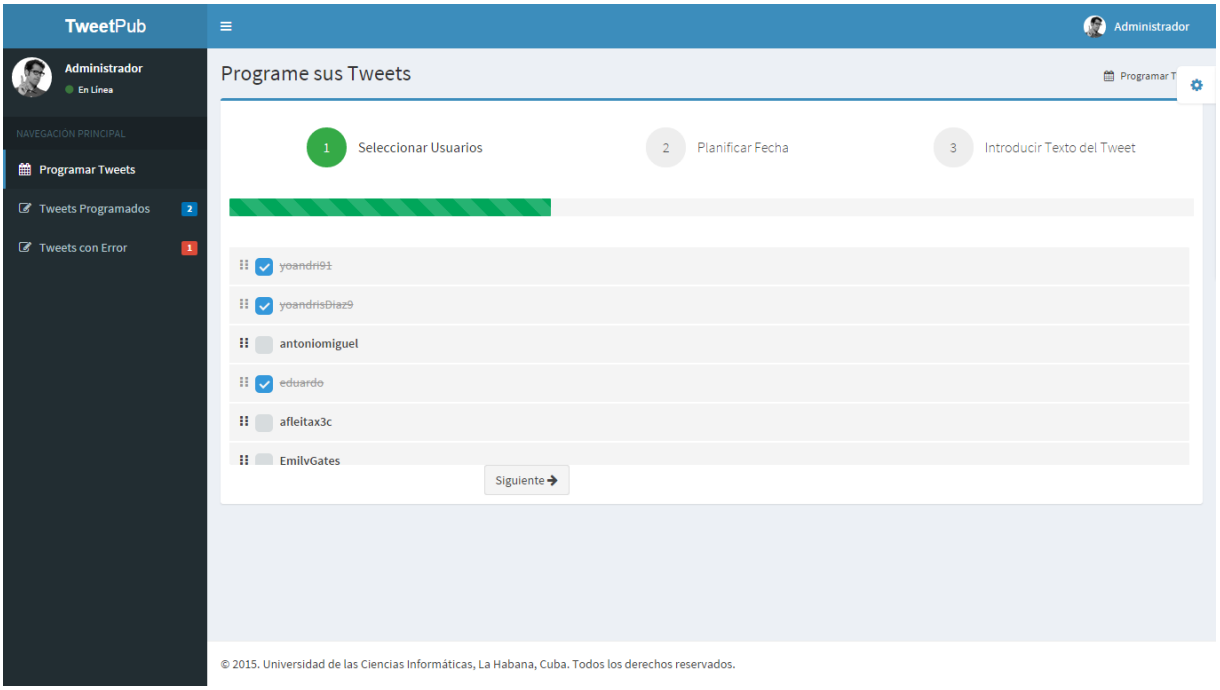


Figura 6: Vista de gestión de usuarios para la planificación de *tweets*

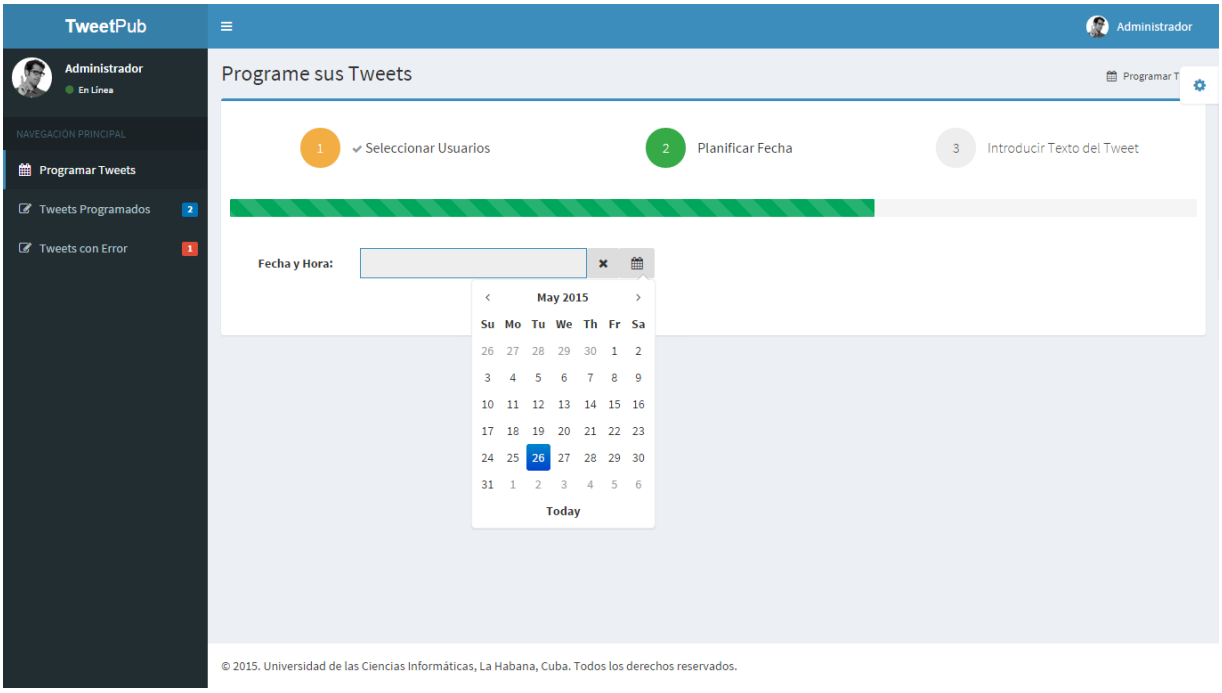


Figura 7: Vista de gestión de fecha para la planificación de *tweets*

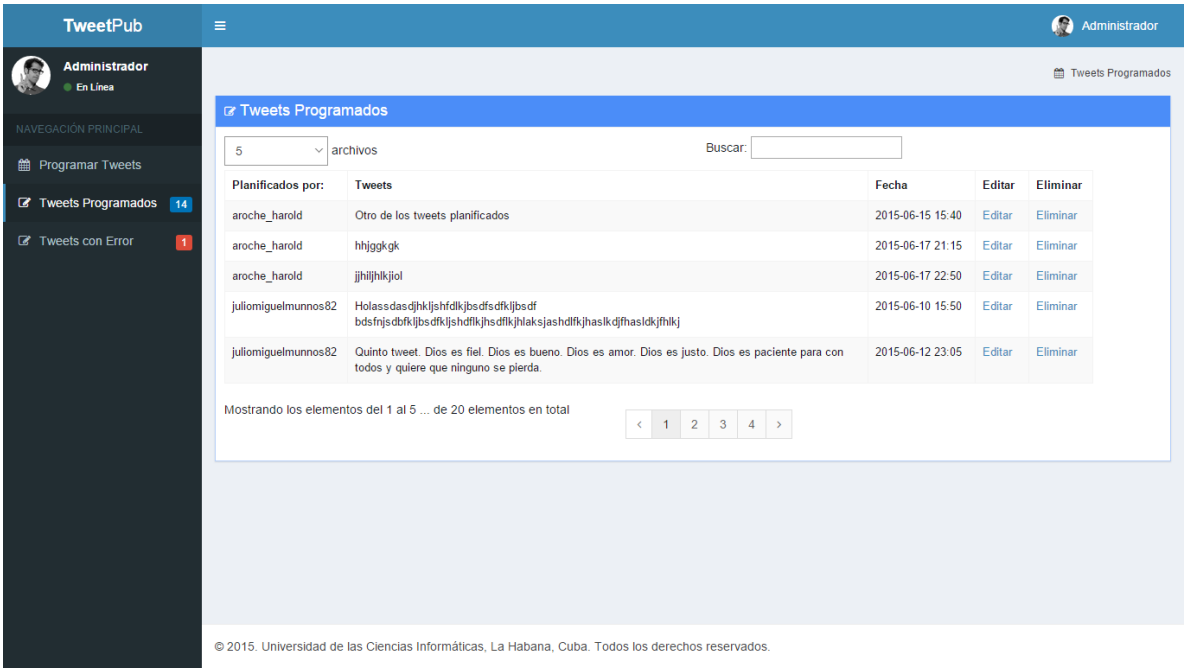


Figura 8: Vista de los *tweets* planificados en el Sistema

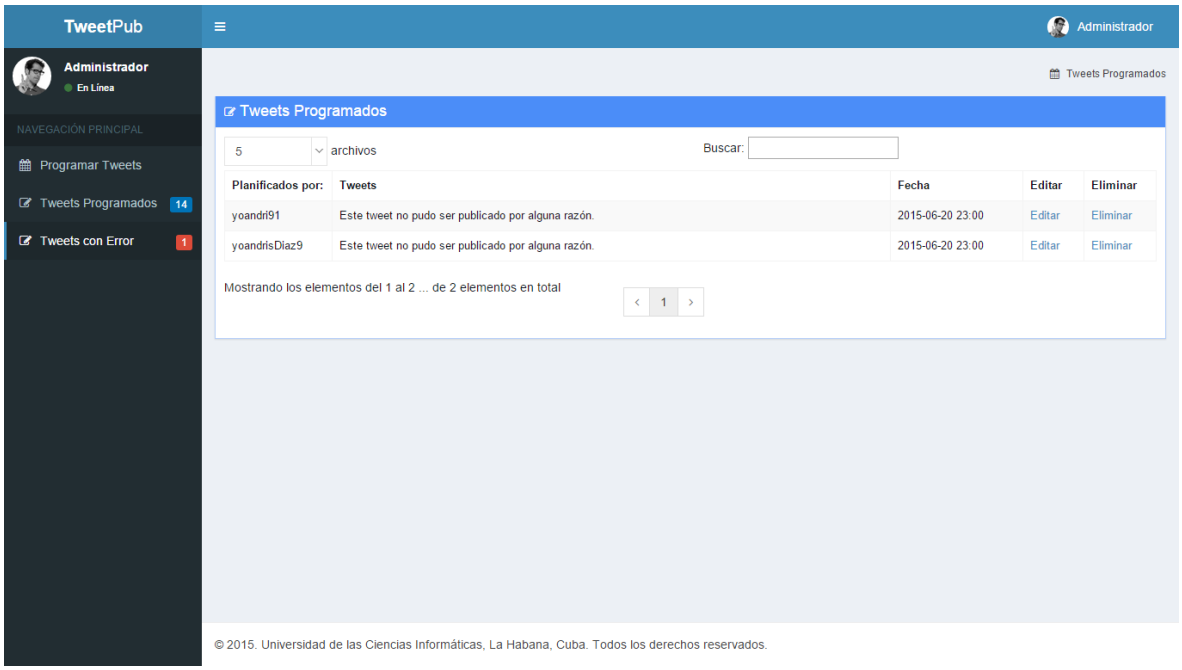
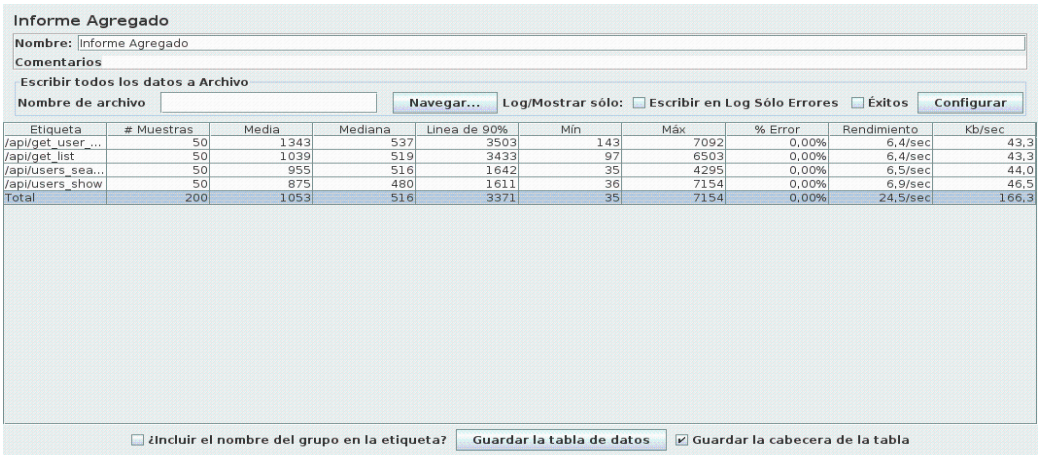
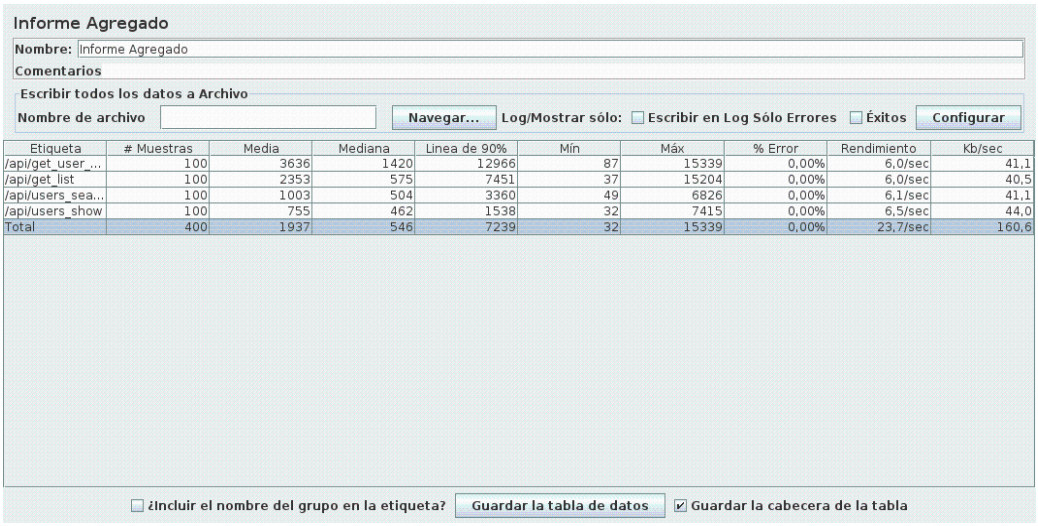


Figura 9: Vista de los *tweets* que no se publicaron a pesar de haberse planificado



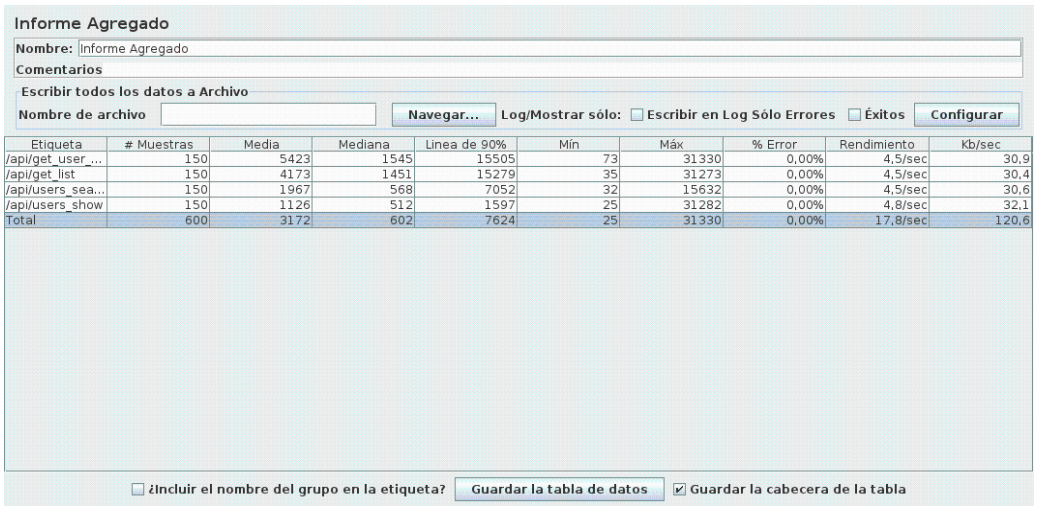
Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
/api/get_user_...	50	1343	537	3503	143	7092	0,00%	6,4/sec	43,3
/api/get_list	50	1039	519	3433	97	6503	0,00%	6,4/sec	43,3
/api/users_sea...	50	955	516	1642	35	4295	0,00%	6,5/sec	44,0
/api/users_show	50	875	480	1611	36	7154	0,00%	6,9/sec	46,5
Total	200	1053	516	3371	35	7154	0,00%	24,5/sec	166,3

Figura 10: Informe resultante de la herramienta JMeter para 50 usuarios



Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
/api/get_user_...	100	3636	1420	12966	87	15339	0,00%	6,0/sec	41,1
/api/get_list	100	2353	575	7451	37	15204	0,00%	6,0/sec	40,5
/api/users_sea...	100	1003	504	3360	49	6826	0,00%	6,1/sec	41,1
/api/users_show	100	755	462	1538	32	7415	0,00%	6,5/sec	44,0
Total	400	1937	546	7239	32	15339	0,00%	23,7/sec	160,6

Figura 11: Informe resultante de la herramienta JMeter para 100 usuarios



Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
/api/get_user_...	150	5423	1545	15505	73	31330	0,00%	4,5/sec	30,9
/api/get_list	150	4173	1451	15279	35	31273	0,00%	4,5/sec	30,4
/api/users_sea...	150	1967	568	7052	32	15632	0,00%	4,5/sec	30,6
/api/users_show	150	1126	512	1597	25	31282	0,00%	4,8/sec	32,1
Total	600	3172	602	7624	25	31330	0,00%	17,8/sec	120,6

Figura 12: Informe resultante de la herramienta JMeter para 150 usuarios