Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

Frameworks para el Desarrollo de Aplicaciones Web en Haskell: Una Comparación de Yesod, Snap y Happstack

Renny Hernández

RT 2014-01

Centro de Investigación en Sistemas de Información (CISI)
Caracas, mayo 2014

Frameworks para el Desarrollo de Aplicaciones Web en Haskell: Una Comparación de Yesod, Snap y Happstack

Renny A. Hernández G.
renny.hernandez@ciens.ucv.ve
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Resumen

Debido al procesamiento de grandes datos, Haskell y otros lenguajes funcionales como Erlang, Scala y Scheme han ganado un auge considerable en la industria del desarrollo de software gracias a las ventajas que brinda al procesar listas, su correctitud gramatical y robustez en el sistema de tipos de datos. En los ambientes web, Haskell cuenta con una lista de frameworks que permiten la automatización y generación de funcionalidades comunes en las aplicaciones, reduciendo los costos y tiempos de programación. En este documento se comparan los frameworks Yesod, Snap y Happstack, enfocándose en sus características técnicas e identificando sus ventajas y deficiencias más relevantes que permiten generar recomendaciones e implicaciones en el uso de cada de uno de estos.

1. Introducción

El desarrollo de aplicaciones web en la actualidad se soporta en la creación de vistas de manera dinámica, ambientes asíncronos y servicios web que brindan ubicuidad y extensibilidad al permitir el despliegue en ambientes móviles y navegadores tradicionales de manera transparente.

Los frameworks o marcos de trabajo web permiten agilizar el proceso de desarrollo de una aplicación web a través de librerías y herramientas de generación de código para crear un andamiaje o esqueleto que contiene un conjunto de funcionalidades básicas. El objetivo de estos marcos de trabajo es el de reducir los tiempos de programación y la cantidad de errores al brindar una gama de patrones configurables y extensibles.

Haskell es uno de los lenguajes funcionales puros más conocidos en la actualidad y cuya utilidad ha sido demostrada ampliamente en ambientes educativos. Pese a esto, sus características de robustez, y eficiencia en el procesamiento de datos tanto en ambientes secuenciales como paralelos y concurrentes han permitido industrializar su uso [1] . Gracias a este auge, en los últimos años se han desarrollados frameworks y librerías que permiten el desarrollo de aplicaciones web en el lenguaje Haskell. Algunos de los más utilizados en la actualidad son Yesod, Snap y Happstack.

El objetivo de este documento es estimar la viabilidad de uso entre los frameworks yesod, snap y happstack, a través de un análisis comparativo y de la realización de pruebas de desempeño.

El documento se organiza de la siguiente manera. La sección 2 describe el lenguaje Haskell y sus frameworks a analizar junto a sus características técnicas más relevantes. La sección 3 muestra el análisis comparativo, las pruebas de desempeño aplicadas a cada uno de los frameworks y los resultados obtenidos. Finalmente, la sección 4 presenta un conjunto de recomendaciones derivadas de la discusión de resultados obtenidos.

2. Frameworks Web en Haskell

Un framework o marco de trabajo permite reducir los tiempos de desarrollo a través de la implementación de distintos patrones diseñados para solucionar problemas específicos. Los frameworks web reducen los tiempos de codificación mediante la generación de código y reutilización de módulos y funciones. Entre los frameworks para el desarrollo de aplicaciones web utilizados hoy en día son Cake PHP (PHP), Ruby On Rails (Ruby) y Django (Python). En esta sección se describe Haskell como lenguaje funcional, su aplicación en entornos web y finalmente, se definen los frameworks para el desarrollo web en Haskell: Yesod, Snap y Happstack.

2.1. Haskell

Haskell es un lenguaje funcional puro de propósito general, creado por Simon Peyton Jones y John Hughes con una primera versión presentada en 1990. En 1997 se publicó Haskell 98 con miras a ser un lenguaje portable y minimal para fines académicos. En la actualidad, el lenguaje se encuentra en la versión 7 y ha sobrepasado los ambientes educativos para ser parte del grupo de lenguajes funcionales utilizados en la industria del desarrollo de software por sus características de evaluación perezosa (retrasar el cálculo de una expresión hasta que el valor sea necesario), desarrollo por patrones, clases de tipos y polimorfismo de tipos. El compilador GHC (*Glasgow Haskell Compiler*) es el más utilizado en la actualidad para interpre-

tar y compilar código Haskell en múltiples plataformas. Este compilador cuenta con `Cabal`, un sistema que permite construir y compartir librerías a través de un repositorio web de acceso libre.

Los lenguajes funcionales puros permiten asegurar inmutabilidad y persistencia de los valores a través de funciones puras. Una función es pura si su valor sólo depende de los argumentos con los cuales es evaluada. Por ejemplo, las funciones $\cos(x)$, $\exp(-x)$ son funciones puras, ya que al ser evaluada con un valor de x arbitrario siempre se obtendrá el mismo resultado. Por otro lado, una función aleatoria $random()$ es impura, debido a que debe retornar un valor distinto cada vez que es invocada. En el caso de las funciones impuras, *Haskell* utiliza el concepto de mónadas, extraído de la teoría de categorías. Una mónada permite representar funciones no seguras, que puedan generar efectos secundarios al ser invocadas. Como otros lenguajes funcionales, *Haskell* es una implementación del cálculo lambda [2], el uso de funciones anónimas y de estrategias de evaluación perezosa.

2.2. Haskell en Ambientes Web

El uso de los lenguajes funcionales para entornos industriales y en particular en el desarrollo de aplicaciones web ha incrementado en los últimos años. Uno de éstos lenguajes es Erlang, creado por la compañía de comunicaciones Ericsson para crear y mantener hilos de procesos a nivel de usuario, en un ambiente funcional, concurrente y con manejo de tipos de datos dinámico [3]. Muchos lenguajes no funcionales utilizados hoy en día como Ruby y Python, implementan conceptos propios de la programación funcional, como el uso de funciones anónimas y los conocidos procedimientos *map* y *reduce*.

Empresas como AT&T, Barclays y Bank of America han incursionado en el desarrollo de aplicaciones en *Haskell*: sistemas de seguridad, módulos de Extracción/-Transformación/Carga (ETL) para almacenes de datos y para la definición de lenguajes de propósito específico, aplicaciones web, entre otros. Entre las aplicaciones web más relevantes, se encuentra el servicio de transferencia de archivos entre dispositivos móviles **Bump** [4], el cual permite enviar archivos entre dispositivos cercanos utilizando geolocalización y redes móviles. Por su parte, la conocida empresa **Facebook**, también cuenta con herramientas para el análisis de código fuente desarrolladas en *Haskell* [5].

En la actualidad, Haskell cuenta con distintas librerías para el desarrollo de aplicaciones y que han sido recopiladas en frameworks o marcos de trabajo, los cuales agilizan el proceso de desarrollo y brindan un ambiente de alto desempeño a través de la reutilización y generación automática de código. Algunos de ellos son: *Yesod*, *Happstack* y *Snap*. Cada uno de ellos será revisado a continuación.

2.3. Yesod

Yesod [6] es un framework web con características de seguridad de tipos, precisión y trabajo modular. Permite desarrollar aplicaciones bajo la arquitecura MVC (Modelo Vista Controlador) e implementar servicios web REST (*Representational State Transfer*). Al estar desarrollado en la capa más alta de la arquitectura WAI (*Web Architecture Infrastructure*), puede ejecutarse en los servidores de aplicación más utilizados en *Haskell* e incluso como aplicación de escritorio [7].

El sistema de tipos en *Haskell* permite definir nuevos tipos sobre los ya existentes, definiendo nuevas reglas. Por ejemplo, el tipo de datos `Html` (del lenguaje HTML: Hypertext Markup Language) existente en el framework Yesod, se define como una cadena de caracteres (derivado del tipo `String`) y contiene definiciones de reglas propias del lenguaje de hipertexto, como por ejemplo, la transformación del caracter '>' a su equivalente en el lenguaje de marcado '`>`'.

Yesod soporta aplicaciones capaces de implementar servicios web REST definir recursos que poseen un conjunto de operaciones para su acceso, creación, modificación y eliminación a través de peticiones HTTP (HyperText Transfer Protocol) y los métodos `GET`, `POST`, `PUT` y `DELETE` respectivamente. Este sistema permite además definir más de un formato para la respuesta del servidor. Por ejemplo, la petición `http://example.com/productos/1.xml` utiliza el método `GET` y retorna el producto de identificador "1" en formato XML. Para mayor detalle sobre los servicios RESTful, se recomienda consultar [8].

En la arquitectura MVC, existen tres niveles de responsabilidades que permiten responder a una petición de usuario. El nivel modelo gestiona la persitencia de los datos y define los recursos junto a laógica asociada (atributos, métodos y fuentes de datos). La capa controlador se encarga de recibir y responder a peticiones HTTP e instancia los modelos necesarios para cumplir con la lógica de la aplicación. La capa de vista, mantiene una comunicación directa con el usuario a través de las interfaces gráficas y las presentaciones de datos.

Considere el ejemplo de la figura 1: en este esquema existe el modelo `Producto` que puede contener atributos donde cada uno corresponde a una columna de la tabla Producto en la base de datos. El controlador `Productos` mantiene la lógica de la aplicación al definir los métodos de creación, lectura, actualización y eliminación (CRUD: *Create, Read, Update, Delete*) de las instancias del modelo `Producto`. En cada uno de estos métodos se definen los formatos de respuesta y se redirecciona a la vista asociada con la petición. Finalmente, en la capa de vistas, se definen las presentaciones a través de lenguajes como HTML y XML (del eXtensible Markup Language, Lenguaje de Marcado Extensible).

Usualmente, el enrutador es quien se encarga de redirigir las peticiones al controlador pertinente. La función principal del enrutador es la de analizar la petición HTTP y enviar los parámetros de entrada al acción del controlador correspondiente.

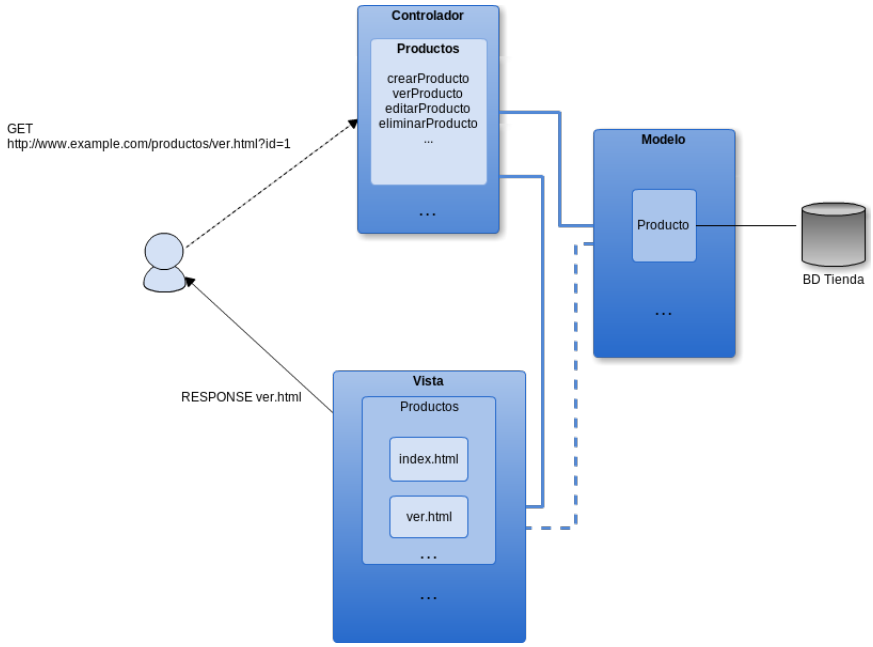

Figura 1: Flujo de una Petición en una Aplicación MVC

En el ejemplo de la figura 1, la petición `http://www.example.com/productos/1.html` puede descomponerse y obtener el controlador (`productos`). el enrutador infiere la acción por medio del método GET (`verProducto`), el identificador único del producto solicitado (`1`) y finalmente, el formato de respuesta (`html`). El significado de las partes en la petición pueden variar según la implementación del enrutador en el framework.

Yesod agiliza el desarrollo de las aplicaciones a través de la reutilización y generación de código mediante técnicas de metaprogramación, lenguajes de propósito específico y el uso de libreías reutilizables. A continuación se mencionan algunas características:

- Al igual que muchos frameworks web, Yesod incorpora rutinas que generan la estructura de archivos y el código necesario para comenzar una nueva aplicación. Este proceso se conoce como andamiaje o *scaffolding*).
- Yesod cuenta con los lenguajes Hamlet, Cassius y Julius, que permiten generar documentos HTML, CSS y Javascript, respectivamente, partiendo de un código que puede contener rutinas de Haskell.
- Yesod implementa el uso de *widgets*. Un widget es un bloque de código que encapsula una vista total o parcial.
- Yesod promueve la reutilización de código. Por ejemplo, la librería `Yesod.Form`,

permite crear formularios mediante patrones de soluciones comunes. Por otro lado, la librería `Persistent` serializa automáticamente los objetos extraídos de la base de datos.

Yesod está compuesto por varios paquetes que pueden ser reutilizados de manera independiente. Del mismo modo, todas las librerías y programas desarrollados en Haskell pueden ser utilizados en un ambiente Yesod. Por ejemplo, si se desea manejar datos del tiempo siguiendo el formato UTC (Tiempo Universal Coordinado), es posible cargar el módulo `time` en el ambiente de la aplicación.

2.4. Snap

Snap [9] es un framework para el desarrollo de aplicaciones web de alto desempeño en Haskell, basado en el uso de módulos o extensiones interoperables e independientes que permiten escalar las aplicaciones desarrolladas. Posee una documentación bien definida y está enfocado en su implementación bajo sistemas *nix. Su primera versión fue lanzada al público en mayo del 2010 y en la actualidad es un proyecto de desarrollo activo.

En general, el núcleo del marco de trabajo consiste en un servidor HTTP, un paquete de módulos para la programación basada en mónadas y un sistema de plantillas para la creación de vistas basado en lenguajes de propósito espécifico DSL (del inglés *Domain Specific Language*). Los paquetes que componen el framework son:

snap-core . El paquete contiene el núcleo del framework. Es una librería que posee las primitivas para el manejo de peticiones y respuestas HTTP, así como para la gestión de las *cookies*. En ella también se encuentra la definiciones de tipos, funciones de I/O y de la mónada Snap que permite el acceso a los objetos recibidos por HTTP así como gestionar los fallos de aplicación.

snap-server . Es un servidor de aplicaciones minimal escrito en Haskell. Actúa en conjunto con el núcleo del framework para servir de interfaz por medio del protocolo HTTP.

snap Es una librería de alto nivel que incluye la interfaz de programación, los archivos ejecutables para iniciar y detener los servidores y los módulos (snaplets) de sesiones, autenticación y creación de vistas.

heist . Heist es un lenguaje de propósito específico que permite la creación de vistas en HTML5 y XML. Se enfoca en la usabilidad orientada al diseñador, facilidad de modificación sin necesidad de reiniciar los servidores y proveer separación entre la capa lógica y la capa de vistas.

xmlhtml . Es una librería que posee intérpretes y generadores de XML y HTML5. Permite interpretar documentos HTML5 y XML y convertirlos en objetos para su uso en Haskell.

2.5. Happstack

Happstack [10] es uno de los frameworks web escritos en Haskell que se conocen en la actualidad. Es el resultado de la prosecución del proyecto HAppS (Haskell Application Server) y al igual que Yesod y Snap, integra el uso de componentes independientes que permiten la creación de vistas de manera dinámica, comunicación mediante el protocolo HTTP y persistencia de datos.

Los componentes principales de Happstack son:

happstack-server . Servidor HTTP del framework Happstack. Permite enrutar peticiones, manejar consultas parametrizadas y generar respuestas, además de encargarse de la generación de cookies y el servicio de transferencia de archivos.

happstack-hsp . Módulo para la creación de vistas mediante plantillas XML y el lenguaje HSP. Básicamente, HSP es un lenguaje XML/HTML que permite la incrustración directa de código Haskell.

reform . Permite la generación de formularios HTML con tipos seguros y validación de campos.

acid-state . Módulo de gestión de la persistencia de los datos. Este paquete brinda características de Atomicidad, Consistencia, Aislamiento y Durabilidad a las estructuras de datos en Haskell.

web-routes . Permite generar las rutas de acceso a la aplicación y validarlas.

happstack-clientsession . Almacena la sesión del usuario en el lado del cliente utilizando cookies.

Al igual que muchos frameworks en Haskell, Happstack permite integrar librerías interoperables que sirven de componentes principales de otros marcos de trabajo como Yesod y Snap. Por ejemplo, las creación de vistas en Happstack (happstack-hsp) puede realizarse utilizando el módulo para la creación de vistas propio de Yesod (hamlet) y viceversa.

3. Pruebas y Resultados

Con el fin de conocer las diferencias entre los frameworks descritos anteriormente, se realizaron distintas pruebas y comparaciones. Las versiones utilizadas para la

prueba fueron Yesod 1.2.5, Snap 0.9.4 y Happstack 7.0.1. Las siguientes pruebas fueron realizadas mediante aplicaciones simples que permiten mostrar un documento HTML. En estas pruebas no fue evaluado el desempeño de los sistemas de bases de datos subyacentes para evitar costos de comunicación y de acceso a ellos.

3.1. Características Básicas

La siguiente tabla (Tabla 1) muestra la comparación de un conjunto de características extraídas de la documentación de cada uno de los frameworks evaluados.

Tabla 1: Comparación de las Características Básicas de los Frameworks Yesod, Snap y Happstack

	yesod	**snap**	**happstack**
Licencia	BSD3	BSD3	BSD3
Servidor de aplicación	warp	snap-server	happstack-server
Lenguaje de creación de vistas	Hamlet (Html), Julius (Javascript), Cassius (CSS)	Heist (Html) , Fay (Javascript)	Hsp, Blaze-Html (Html), Fay (Javascript)
Soporte a Persistencia	MySQL, MongoDB, PostgreSQL, Sqlite	Snaplets (HDBC, AcidState, MySQL, PostgreSQL, MongoDB, Sqlite	acid-state (NoSQL), HDBC (*Haskell Database Controller*)
Soporte integrado a servicios REST	Sí	No	No
Procesamiento de Formularios	yesod-form (formularios monádicos y aplicativos)	digestive-forms (basados en functores)	reform (formularios aplicativos)
Documentación	Libro, Wiki, Grupos de Discusión	Wiki, Grupos de Discusión	Wiki, Grupos de Discusión
Sistemas Operativos Soportados	Linux/Windows (a través de haskell-platform)	Linux/Windows 7 (a través de haskell-platform)	Linux

3.2. Instalación y Despliegue

La instalación es el proceso en el cual los distintos paquetes y ejecutables del framework se albergan en memoria secundaria y se configuran para su uso.Para evitar conflictos con los paquetes instalados en Haskell, se recomienda el uso del comando `cabal sandbox init`. Este comando permite almacenar paquetes localmente sin interferir con otros y así evitar problemas de dependencia conocidos [11] siempre y cuando la instalación se realice dentro del directorio del proyecto (es decir, la ejecución del comando `cabal install <paquete>` debe realizarse en el directorio raíz

generado anteriormente). Finalmente y ya que la aplicación a realizar con yesod es a su vez un ejecutable, éste debe instalarse a través del comando `cabal install` desde la raiz del sistema de directorios creado. Esto permite crear un archivo binario que contiene el servidor web junto a la aplicación creada, lista para ser desplegada.

La instalación de yesod, snap y happstack se realiza de manera similar. Para ello, en el caso de Yesod es necesario ejecutar la instrucción
`cabal install yesod-platform yesod-bin`.
Para Snap y happstack se utilizan `cabal install snap`
y `cabal install happstack-server` respectivamente.

El comando `cabal` permite instalar los paquetes y ejecutables necesarios para el uso del framework. Entre los ejecutables se encuentra el comando `yesod init`, el cual permite realizar el andamiaje de una aplicación web de manera automática luego de ingresar el nombre del proyecto y el tipo de persistencia.

El despliegue de la aplicación permite generar un ejecutable para su ejecución en ambientes de producción. En yesod, el despliegue se realiza usando la herramienta `cabal`. Específicamente mediante el comando `cabal clean && cabal build`. Esto genera un paquete ejecutable que debe ser desplegado junto a los directorios `config` y `static`. La documentación muestra una manera de realizar el despliegue utilizando los servidores web apache y nginx [12]. Utilizando Happstack, el despliegue se realiza mediante la compilación del código y su ejecución. Finalmente, las aplicaciones web desarrolladas con Snap pueden ser desplegadas mediante la instrucción `cabal install`. El paquete ejecutable que resulta del proceso de despliegue con Yesod puede trabajar en conjunto con los servidores web Apache y Nginx [7].

3.3. Manejo de la Persistencia

La persistencia de los datos es una parte esencial de cualquier sistema informático. En la arquitectura Modelo Vista Controlador MVC, la capa de Modelos alberga la definición y manejo de las estructuras de datos no volátiles. En la mayoría de los casos, el modelo de datos es de tipo relacional y se soporta mediante el uso de un Sistema Manejador de Bases de Datos Relacional SMBDR (MySQL, PostgreSQL y Sqlite son los más utilizados). Comúnmente, este esquema relacional se comunica con el framework a través de un conjunto de librerías de correspondencia Objeto-Relacional que llevan los datos del esquema relacional a un enfoque orientado a objetos (ORM: Object Relational Mapping) junto a abstracciones que permiten la conexión a distintos sistemas de bases de datos. Mediante estas librerías, se pueden manipular los datos a nivel de lenguaje de tal modo que las tuplas de una tabla sean instancias de clases, que a su vez poseen atributos y métodos. En otros casos, la persistencia es soportada bajo sistemas de almacenamiento de datos semiestructurados (sistemas basados en archivos JSON (*JavaScript Object Notation*) o XML, como por ejemplo, MongoDB).

Yesod cuenta con un sistema de definición de entidades a almacenar. En el andamiaje del framework, el archivo `config/models` se definen todas las estructuras, sus atributos y restricciones tales como la existencia de atributos únicos, opcionales, compuestos, relaciones entre tablas, entre otras. El framework realiza la traducción de código a través de la librería ORM `Database.Persist` y genera nuevas consultas que modifican el esquema cada vez que el archivo `models` se modifique. El siguiente código es un ejemplo de la sintaxis utilizada en el archivo `config/models`.

```
-- config/models
Usuario
  nombre Text
  apellido Text
  email Text
  password Text Maybe
  UniqueUsuario email
  NombreCompleto nombre apellido
  deriving Typeable

Entrada
  titulo Text
  contenido Text
  fecha_creacion UTCTime
  propietario Usuario
```

Para el manejo de la comunicación con los SMBDR, Snap cuenta con el módulo `snap-hdbc`, el cual presenta una serie de utilidades para el uso del conector de bases de datos HDBC *Haskell Database Connectivity*. Las bases de datos soportadas por HDBC son SQLite, PostgreSQL, MySQL y ODBC (Microsoft SQLServer). Adicionalmente, existen librerías que proveen soporte tanto a los SBMDR mencionados anteriormente como al sistema de almacenamiento MongoDB. La diferencia de estos enfoque radica en la cantidad de operaciones que brinda cada uno de los módulos: el módulo `snap-hdbc` permite realizar consultas y retornar estructuras de datos que representan una o más filas de los resultados sobre el sistema de base de datos configurado. Las librerías adicionales como `snaplet-mysql` poseen un conjunto de operaciones reducidas y enfocadas sólo al manejador MySQL.

En contraste con Yesod, Snap y Happstack cuentan con la librería Acid-State para el manejo de persistencia a nivel de lenguaje. Este enfoque permite mantener la persistencia en un ambiente funcional puro, donde los objetos de datos poseen características de atomicidad, consistencia, aislamiento y durabilidad propias de un SMBDR. A diferencia de Snap, Happstack sólo posee este tipo de almacenamiento nativo.

3.4. Desempeño del Servidor de Aplicaciones

Las herramientas utilizadas para medir el desempeño del servidor fueron: httperf, autobench y ab (ApacheBench). Las aplicaciones web fueron desplegadas en producción sobre un computador con procesador Intel I3 (2.8 GHz, 4 núcleos), 4 GB de memoria principal y 500 GB de disco duro. Con el fin de orientar las medidas al desempeño de los servidores de aplicación y evitar los costos de comunicación a través de la red, las pruebas se realizaron de manera local sobre el servidor. La respuesta de la petición HTTP GET sobre el URI `http://localhost:<puerto>/hello` que fue utilizada para las pruebas consiste en el siguiente archivo HTML (con un tamaño de 1KB aproximadamente):

```
<html>
  <head>
    <!-- -->
  </head>
  <body>
    <p>
      Hello, World!
    </p>
  </body>
</html>
```

Autobench es una herramienta de medición de desempeño para servidores web basada en httperf. Principalmente, es un script escrito en Perl que permite automatizar la ejecución de httperf por lotes y retorna un archivo tabulado con los resultados de la prueba. El comando ejecutado para la ejecución de la prueba es el siguiente:

```
$> autobench --host1=localhost --uri1=/hello --port1=3000
--quiet --single_host --file=yesod-benchmark.tsv
```

En el ejemplo anterior, se especifica el puerto 3000 para el servidor Warp (Yesod). Las pruebas para Snap y Happstack se realizaron en los puertos 4000 y 8000, respectivamente.

La prueba consistió en el envío de una cantidad específica peticiones por segundo, comenzando con 20 conexiones, cada una con 10 peticiones por conexión y con aumentos de 10 conexiones por cada segundo hasta llegar a 2000. Del resultado de esta ejecución interesa conocer el promedio de tiempo de respuesta por segundo. La siguiente tabla (2) muestra el resultado de la ejecución de la prueba para los servidores de Yesod (Warp), Snap y Happstack.

Para evaluar la cantidad de peticiones atendidas por segundo, se utilizó la herramienta AB (Apache Benchmark). Se ejecutaron 10000 peticiones por cada una de 10 conexiones concurrentes. El promedio de peticiones atendidas por segundo para cada servidor se muestra en la tabla 3.

Tabla 2: Tiempo de Respuesta por Segundo #

Peticiones	Yesod	Snap	Happstack
200	0.3	0.3	0.2
300	0.3	0.3	0.2
400	0.2	0.3	0.2
500	0.3	0.3	0.5
600	0.2	0.3	0.3
700	0.2	0.3	0.3
800	0.2	0.3	0.3
900	0.2	0.3	0.3
1000	0.2	0.3	0.6
1100	0.3	0.3	0.4
1200	0.3	0.3	0.4
1300	0.2	0.3	0.5
1400	0.2	0.3	0.5
1500	0.3	0.3	0.7
1600	0.2	0.3	0.4
1700	0.2	0.3	0.5
1800	0.2	0.3	0.5
1900	0.2	0.3	0.6
2000	0.2	0.3	0.7
Promedio	0.231	0.3	0.426

ǫnocer la cantidad aproximada de peticiones solicitadas, aceptadas y rechazadas, se utilizó la herramienta `weighttp`. Esta utilidad de prueba de desempeño permite enviar peticiones a un servidor HTTP a través de hilos y simulando una cantidad de conexiones por usuario. Para la prueba realizada, se simularon 10 hilos de 40 usuarios con 15000 peticiones por cada uno. La instrucción que configura los parámetros descritos se muestra en la Tabla 4.

```
$> weighttp -n 15000 -t 10 -c 40 http://localhost:3000/hello
```

Finalmente, los resultados de la ejecución de las pruebas para cada uno de los servidores son:

3.5. Uso de Memoria

El valor de asignación y uso de memoria fue calculado mediante la herramienta `valgrind`, que permite evidenciar el espacio de memoria utilizado al finalizar el proceso, además de la espacio asignado. Los parámetros utilizados para esta prueba

Tabla 3: Número de Peticiones Atendidas por Segundo

	Yesod	Snap	Happstack
# Peticiones	2319	2279	3324

Tabla 4: Número de Peticiones Solicitadas, Aceptadas y Rechazadas

	Yesod	Snap	Happstack
Solicitadas	15000	15000	15000
Aceptadas	15000	15000	7978
Rechazadas	0	0	7022

fueron similares a los utilizados para conocer la cantidad de peticiones finalizadas mediante `weighttp` (10 hilos, 40 usuarios y 15000 peticiones por cada usuario). La cantidad de memoria asignada para cada uno de los frameworks se muestra en la Tabla 5.

Tabla 5: Espacio de Memoria Asinada por Framework en MBytes

	Memoria asignada (bytes)
Yesod	236461157
Snap	1576196
Happstack	3527509

El uso de la memoria asignada para las pruebas fue inferior al 1 % debido a la cantidad de peticiones y al hecho de que no se evalúan peticiones de carga de archivos grandes.

4. Discusión y Recomendaciones

En este documento se presentó una revisión a nivel teórico y técnico de los frameworks Yesod, Snap y Happstack. De los resultados de la evaluación características básicas presentes en la documentación, se puede notar que:

- Yesod es el framework con mayor documentación y soporte de la comunidad disponible.

- Algunos paquetes (como Fay y acid-state) pueden ser importados correctamente en Snap y Happstack.
- Snap y Happstack poseen un manejo de la persistencia no relacional como opción predeterminada (Acid-State).
- El soporte a servicios REST de Snap y Happstack no está presente como parte de su instalación. Sin embargo, puede brindarse a través del uso de librerías adicionales.
- Según la documentación, los tres frameworks (al igual que sus respectivos servidores) pueden ejecutarse desde ambientes Windows.

Con respecto al análisis del proceso de instalación, es importante recalcar el uso de la herramienta `cabal sandbox` para evitar errores de dependencia al momento de instalar cualquiera de los tres frameworks. La instalación de dos o más de estos frameworks en el mismo ambiente de trabajo podría ocasionar el conflicto de dependencias conocido como *cabal hell* [11]. El despliegue de las aplicaciones es similar para los tres framworks. Sin embargo, Snap y Yesod utilizan el gestor de paquetes `cabal` a diferencia de Happstack que sólo necesita ser compilado. El proceso de despliegue bajo Yesod utilizando los servidores web Apache o Nginx está ampliamente documentado.

Con respecto al análisis de la persistencia de los frameworks, se recalca lo siguiente:

- Snap y Yesod proveen soporte nativo a Sistemas Manejadores de Bases de Datos Relacionales. Resulta viable utilizar estos frameworks si se requiere el uso de estos sistemas de almacenamiento.
- A pesar de que Happstack no posee un acceso nativo a SMBDR, puede implementarse a través de la librería HDBC.
- Para aplicaciones donde la consistencia de los datos sea importante (sistemas transaccionales), no se recomienda el uso de almacenamiento no relacional (Acid-State).

Tomando en cuenta lo anteriormente mencionado, podría inferirse que las aplicaciones desarrolladas utilizando Happstack son más eficientes debido a la ausencia de un sistema de bases de datos relacional y todo el costo de cómputo que representa. Sin embargo, esta situación no es generalizada. En muchos casos, el costo de procesamiento inherente al uso de Sistemas Manejadores de Bases de Datos Relacionales ha obligado a escalar a modelos NoSQL, mientras que en otros, la cantidad de peticiones, el volumen de datos y la naturaleza transaccional del sistema son

características que podrían hacer esta migración inviable. Se recomiendan estudios exhaustivos con grandes cantidades de datos y la evaluación de necesidades en cuanto a características que brindan los sistemas relacionales para tomar la decisión de utilizar un modelo de datos relacional o un esquema propio de los sistemas NoSQL.

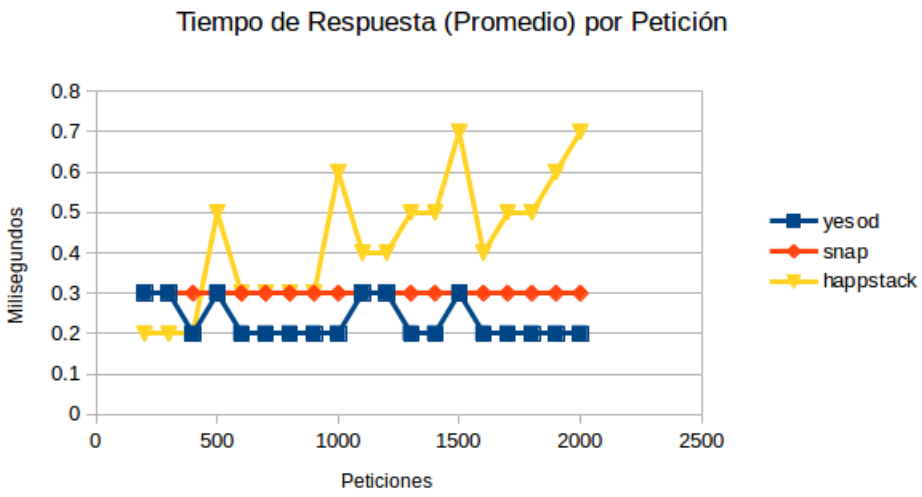

Figura 2: Promedio de Tiempos de Respuesta por Servidor

El desempeño de los frameworks fue evaluado tomando en cuenta principalmente el tiempo de respuesta a peticiones solicitadas por lote y la memoria asignada en este proceso. Con respecto al tiempo de respuesta (Figura 2) el resultado de esta prueba evidencia que tanto Snap como Yesod poseen valores similares y mucho más reducidos que Happstack. Esto podría variar según la complejidad lógica de la aplicación.

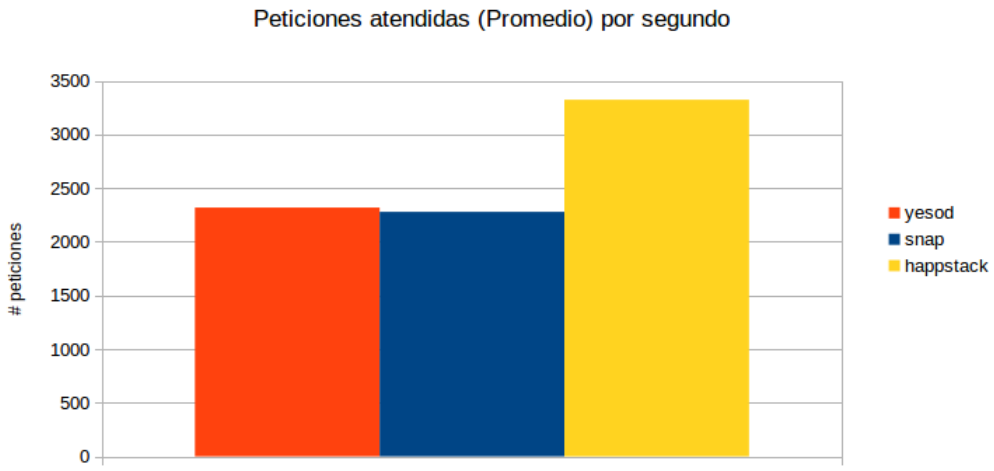

Figura 3: Promedio de Peticiones Atendidas por Segundo

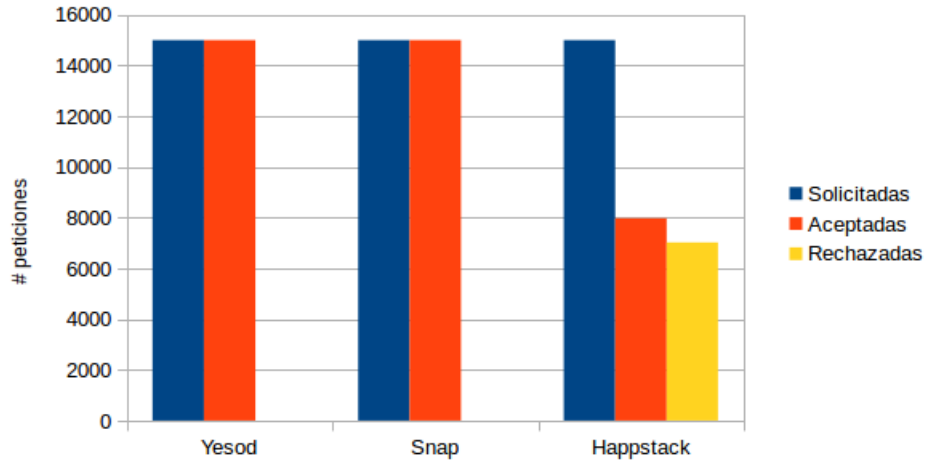

Figura 4: Cantidad de Peticiones Solicitadas, Aceptadas y Rechazadas por Servidor

Con respecto a la cantidad de peticiones atendidas por segundo (Figura 3), Happstack resultó ser el framework con más peticiones atendidas. Sin embargo, Happstack obtuvo un total aproximado de 46 % de peticiones rechazadas (Figura 4).Por otro lado, tanto Yesod como Snap obtuvieron el 100 % de peticiones aceptadas. Este comportamiento se evidenció en las distintas ejecuciones de la prueba de desempeño. Los registros del servidor arrojaron error de Entrada/Salida. El incidente fue reportado al grupo de desarrollo del proyecto Happstack a través de la lista de correos.

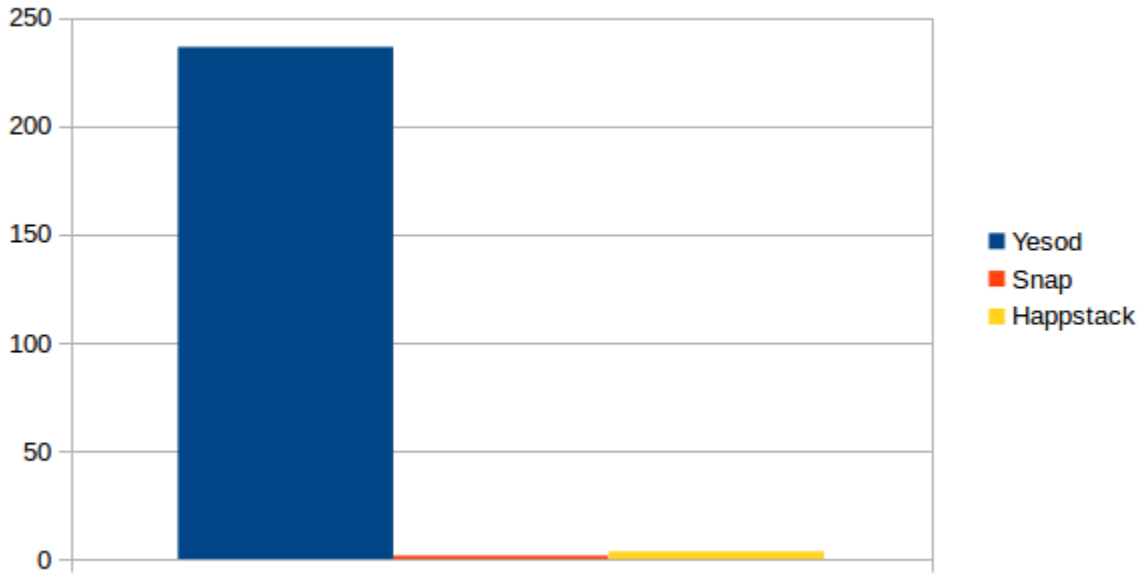

Figura 5: Promedio de Memoria Asignada a los Servidores Yesod, Snap y Happstack

Al evaluar la cantidad de memoria asignada, es evidente que Yesod consume muchos más recursos que Snap y Happstack 5. La razón podría inferirse por la complejidad del servidor y del framework además de la presencia del conector para la base de datos utilizada por defecto al generar el andamiaje del framework (En este caso, se utilizó SQLite), a diferencia de Snap y Happstack que implementan soluciones NoSQL nativas. Es importante también tomar en cuenta que debido a la naturaleza desacoplada de SQLite, gran parte de la memoria es asignada al manejador que se ejecuta dentro del proceso asignado al servidor Yesod.

Tomando en cuenta los resultados obtenidos, se recomienda el uso de los frameworks Yesod y Snap para el desarrollo de aplicaciones web. Yesod representa la solución más viable para el desarrollo de aplicaciones web debido al desempeño de su servidor y flexibilidad en el uso de SMBDR, además de contar con una extensa documentación. Sin embargo, Snap representa una lógica mucho más sencilla, lo que facilita la codificación y reduce los tiempos de entrega.

Referencias

[1] "Is haskell a good choice for web applications?" http://jekor.com/article/is-haskell-a-good-choice-for-web-applications, accedido: 01/02/2012.

[2] E. Moggi, "Computational lambda-calculus and monads." IEEE Computer Society Press, 1988, pp. 14–23.

[3] E. T. Ab, "The development of erlang," in *in Proceedings of the ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 1997, pp. 196–203.

[4] "Bump: Easily transfer photos, files and contacts between your phone and computer," https://bu.mp/, accedido: 30/03/2014.

[5] "lex-pass: manipulate a php codebase using haskell to transform the abstract-syntax-tree http://developers.facebook.com/," https://github.com/facebook/lex-pass, accedido: 01/04/2014.

[6] "Yesod web framework for haskell," http://www.yesodweb.com, accedido: 01/05/2011.

[7] "Deploying your webapp," http://www.yesodweb.com/book/deploying-your-webapp, accedido: 07/05/2014.

[8] R. Fielding, "Architectural styles and the design of network-based software architectures," Doctoral Dissertation, University of California, Irvine, 2000.

[9] "Snap framework," http://www.snapframework.com, accedido: 01/02/2012.

[10] "Happstack," http://www.happstack.com/, accedido: 01/02/2012.

[11] "Solving cabal hell," http://www.yesodweb.com/blog/2012/11/solving-cabal-hell, accedido: 07/05/2014.

[12] M. Snoyman, *Developing Web Applications with Haskell and Yesod*. O'Reilly Media, Inc., 2012.