DESARROLLO DE APLICACIONES DE ESCRITORIO HÍBRIDAS CON JAVASCRIPT, CSS Y HTML

AUTORES: Carlos Roberto Sampedro Guamán[1]
Silvio Amable Machuca Vivar[2]
Edison Israel Patrón Sabando[3]

DIRECCIÓN PARA CORRESPONDENCIA: carlosrs7@gmail.com

Fecha de recepción: 20-06-2016

Fecha de aceptación: 17-07-2016

RESUMEN

Las aplicaciones de escritorio híbridas se desarrollan con lenguajes propios de las webapps: Javascript, CSS y HTML, siendo estas una combinación de la app nativa y web app incluyendo las mejores características de cada una de ellas, por lo que permiten su uso en diferentes plataformas. El análisis fue realizado con el app runtime NW.js y el Node.js con el que se puede acceder al hardware, sistemas de archivos, y redes funcionales; facultando su uso en una amplia gama de campos, desde la implementación de los servidores web para la creación de software de control para los robots. Al desarrollar las aplicaciones híbridas, el front-end y back-end, el usuario ejecuta de forma explícita la aplicación y en un contexto local. Estas opciones permiten a los desarrolladores combinar los beneficios de las aplicaciones nativas y las webs. El enfoque de esta investigación está inmerso en el desarrollo de software híbrido.

PALABRAS CLAVE: App; Aplicaciones híbridas; Javascript; NW.js; Node.js.

APPLICATIONS DEVELOPMENT OF HYBRID DESKTOP WITH JAVASCRIPT, CSS AND HTML

ABSTRACT

The hybrid desktop applications develop with own languages of webapps: Javascript, CSS and HTML, being these a combination of the native app and web app including the best features of each one, so allow its use on different platforms. The analysis was performed with the app runtime Node.js and NW.js with this we can access to the hardware, file systems, and functional networks; entitling its use in a wide range of fields, from the implementation of web servers for creating control software for robots. In developing hybrid applications, the front-end and back-end, the user runs the application explicitly and in a local context. These options allow developers to combine the benefits of native applications and websites. The focus of this research is involved in developing hybrid software.

[1] Ingeniero en Sistemas e Informática, Magister en Ingeniería y Sistemas de Computación, Docente a Tiempo Completo de la Carrera de Sistemas, Administrador de Seguimiento a Graduados a nivel nacional. Universidad Autónoma Regional Autónoma de los Andes – Extensión Santo Domingo. Ecuador.
[2] Ingeniero en Sistemas e Informática, Magister en Educación Superior, Docente a Tiempo Completo de la Carrera de Sistemas. Coordinador de Investigación. Universidad Autónoma Regional Autónoma de los Andes – Extensión Santo Domingo. Ecuador. E-mail: samv1978@hotmail.com
[3] Ingeniero en Sistemas e Informática. Responsable de Telemática. Universidad Autónoma Regional Autónoma de los Andes – Extensión Santo Domingo. Ecuador. E-mail: sanjisan29@gmail.com

KEYWORDS: App; Applications hybrid; Javascript; NW.js; Node.js.

INTRODUCCIÓN

Gigantes de la tecnología como: Adobe Systems (Virtual Ubiquity), Microsoft (Microsoft Office Live Workspace), Google (Gears), han propuesto desde el 2007 un software en el que se combinan las características de las aplicaciones basadas en Internet y del software de escritorio para crear un modelo híbrido que ofrece lo mejor de ambos mundos. Este tipo de aplicaciones difieren de aplicaciones nativas y web tanto en las tecnologías de desarrollo, rendimiento y costo. El objetivo de esta investigación está en explicar el desarrollo de software híbrido para su uso en sistemas.

El desarrollo de software híbrido permite construir un software mediante tecnologías web, el cual puede ejecutarse en cualquier tipo de dispositivos, tiene un grado mayor de accesibilidad a las funciones de los dispositivos dependiendo del framework en comparación de las aplicaciones web, las aplicaciones híbridas usando tecnologías web (HTML, CSS, EcmaScript), vienen evolucionando de manera significativa, brindando más soporte, accesibilidad y compatibilidad a los dispositivos y sistemas operativos. La construcción de este tipo de software tiene la ventaja de generar un menor costo que un software nativo, debido a que se puede usar el mismo código en cualquier tipo de sistema operativo y dispositivos, aplicando pequeños cambios dependiendo de la API's del framework que se usa, también brinda mejores experiencias de uso que una aplicación web. (Raona, 2014).

El software híbrido se enfoca en aplicaciones que contienen en su interior un navegador web del dispositivo, basado en los lenguajes de programación web (HTML, CSS, Javascript), en la actualidad existen varios frameworks que brindan una gran integración con el SO (Electrón, NW.js), y otros frameworks que están obsoletos (tideskd, appjs) debido a que la integración con el sistema operativo no es óptima y brindan un menor grado de experiencia de usuario del software. Tideskd (Obsoleto), es un kit de desarrollo de software que permitía crear aplicaciones multiplataforma usando HTML, CSS y Javascript, también brindaba las ventajas de usar lenguajes de programación de script tales como: python, php, ruby, permitiendo explotar las habilidades del desarrollador que ya poseen en estos lenguajes de programación. (Pratt, 2012).

AppJS (Desatendido) es un SDK para crear aplicaciones usando Node.js en combinación con chromium como núcleo. Con AppJS se pueden desarrollar herramientas de escritorio y aplicaciones, las cuales usan las mismas bibliotecas y conocimientos usados para crear sitios web, y se las puede ejecutar en los distintos sistemas operativos tales como: Windows, Linux, OSX.

DESARROLLO

Las innovaciones tecnológicas están evolucionando de manera muy acelerada, lo cual es beneficioso tanto para los desarrolladores como para los usuarios finales que usan estas tecnologías. Es importante mencionar que en este documento se enfocará en el desarrollo de software, en lo que respecta a este tipo de desarrollo, existen varios tipos; aplicaciones nativas, aplicaciones web, y aplicaciones híbridas. Este tipo de aplicaciones constan de varias diferencias, tanto en las tecnologías de desarrollo, rendimiento y costo, el enfoque de esta investigación está inmersa en el desarrollo de software híbrido.

Como se ha dicho, el software híbrido permite ejecutarse en cualquier tipo de dispositivos porque se construye mediante tecnologías web y el código para cualquier tipo de sistema operativo y

dispositivos lo cual brinda más soporte, accesibilidad y compatibilidad a ellos. Tiene un grado mayor de accesibilidad a las funciones de los dispositivos dependiendo del framework en comparación de las aplicaciones web, las aplicaciones híbridas usando tecnologías web (HTML, CSS, EcmaScript), vienen evolucionando de manera significativa, brindando más soporte, accesibilidad y compatibilidad a los dispositivos y Sistemas Operativos.

El enfoque híbrido combina desarrollo nativo con tecnología Web. Usando este enfoque, los desarrolladores escriben gran parte de su aplicación en tecnologías Web para múltiples plataformas, y mantienen el acceso directo a APIs nativas cuando lo necesitan. La porción nativa de la aplicación emplea APIs de sistemas operativos para crear un motor de búsqueda HTML incorporado que funcione como un puente entre el navegador y las APIs del dispositivo; este puente permite que la aplicación híbrida aproveche todas las características que ofrecen los dispositivos modernos. (IBM, 2012)

Las apps híbridas se desarrollan con lenguajes propios de las webabpp, es decir, HTML, Javascript y CSS por lo que permite su uso en diferentes plataformas, pero también dan la posibilidad de acceder a gran parte de las características del hardware del dispositivo. La principal ventaja es que a pesar de estar desarrollada con HTML, Java o CSS, es posible agrupar los códigos y distribuirla en app store. (LanceTalent, 2016)

La construcción de este tipo de software tiene la ventaja de generar un menor costo que una software nativo, debido a que se puede usar el mismo código en cualquier tipo de SO y dispositivos, aplicando pequeños cambios dependiendo de la API's del framework que se usa, también brinda mejores experiencias de uso que una aplicación web. (Raona, 2014).

En la actualidad existen varios frameworks que brindan una gran integración con el SO (Electron, NW.js), y otros frameworks que están obsoletos (tideskd, appjs) debido a que la integración con el sistema operativo no es óptima y brindan un menor grado de experiencia de usuario del software.

TideSKD (Obsoleto), es un kit de desarrollo de software que permitía crear aplicaciones multiplataforma usando HTML, CSS y Javascript, también brindaba las ventajas de usar lenguajes de programación de script tales como: python, php, ruby, permitiendo explotar las habilidades del desarrollador que ya poseen en estos lenguajes de programación. (Pratt, 2012).

AppJS (Desatendido) es un SDK para crear aplicaciones usando Node.js en combinación con chromium como núcleo. Con AppJS se pueden desarrollar herramientas de escritorio y aplicaciones las cuales usan las mismas bibliotecas y conocimientos usados para crear sitios web, y se las puede ejecutar en los distintos sistemas operativos tales como: Windows, Linux, OSX.

Los orígenes de NW.js y Electron

Roger Wang (2011) logró encontrar una manera de combinar Webkit (el motor del navegador detrás de Safari, Konqueror y Google Chrome en el momento) con Node.js, por lo que se podía acceder a los módulos de Node.js el código JavaScript que se ejecuta dentro de una página web. Este módulo Node.js se le dio el nombre de nodo Webkit. Él continuó trabajando en el proyecto en el Centro de Tecnología de Código Abierto de Intel en China, que dieron su apoyo al proyecto dejando a Roger trabajar en él por completo, pero no sólo eso, él fue capaz de contratar a otros para trabajar en este también.

En el verano de 2012, un estudiante universitario de alto nivel llamado Cheng Zhao se unió a Intel para trabajar en el nodo Webkit. Él trabajó con Roger para ayudar a mejorar la arquitectura interna del Nodo Webkit, que involucró a cambiar la forma en se combinaron Node.js y Webkit. A medida que evolucionó el código, el Nodo Webkit dejó de ser sólo un módulo de Node.js, y comenzó a ser un framework para aplicaciones de escritorio. Nodo Webkit comenzó a encontrar usos interesantes dentro de 3 de las aplicaciones. El editor de mesa de iluminación fue el primero en hacer uso de Nodo Webkit para entregar su funcionalidad, y ayudó a promover el framework de otros desarrolladores.

En diciembre del mismo año, Cheng dejó Intel para trabajar en Github como contratista. Él tuvo la tarea de ayudar al editor de Atom puerto de Github el uso de infraestructura incorporada de cromo y los enlaces Javascript nativas para el uso de Webkit Nodo.

Aunque NW.js fue la primera estructura de aplicaciones Node.js escritorio, Electron ha convertido rápidamente en un framework popular que ha eclipsado NW.js, aunque ambos han sido muy trabajados por el mismo promotor en diferentes puntos en el tiempo, y compartiendo mucho de código común en términos de cómo los usuarios utilizan su API para la creación de funciones de la aplicación. Ambos han desarrollado diferentes enfoques para su arquitectura interna, y han dado lugar a comunidades separadas que de forma natural acojan las ventajas del uso de su framework sobre el otro.

Introducción a NW.js

NW.js es un framework para la creación de aplicaciones de escritorio con HTML, CSS y JavaScript. Fue creado en noviembre de 2011 por Roger Wang en el Centro de Tecnología de Código Abierto de Intel en China. La idea era que mediante la combinación de Node.js con Webkit (el motor del navegador Web detrás de cromo, una versión de código abierto de Google Chrome), puede crear aplicaciones de escritorio utilizando las tecnologías web. Esta fue la base para el nombre original del framework, Nodo Webkit.

Mediante la combinación de cromo con Node.js, Roger encontró una manera de crear aplicaciones que no sólo se pudo cargar un archivo HTML con CSS y JS dentro de una ventana de la aplicación, sino que también podría interactuar con el sistema operativo a través de una API de JavaScript. Esta API JavaScript podría entonces controlar aspectos visuales como las dimensiones de las ventanas, barras de herramientas y elementos de menú, así como facilitar el acceso a los archivos locales en el escritorio; cosas que las aplicaciones web no podían hacer.

Actualmente hay dos app runtime que están revolucionando las perspectivas del desarrollo de aplicaciones de escritorio, ya que brindan una gran API permitiendo crear software robusto, confiable y escalable:

Electron (creado por Github): Permite desarrollar aplicaciones usando las APIs de HTML5 y también CSS y Javascript, todo esto embebido en el navegador web Chromium con la gran ventaja de usar el conjunto completo de módulos que brinda Node.js y los respectivos módulos de Electron para acceder a los recursos del sistema operativo del usuario.

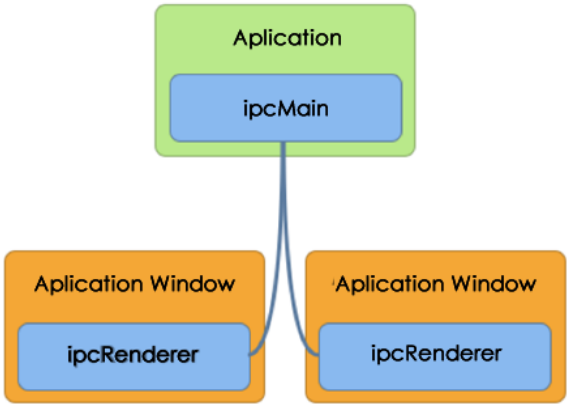

Ilustración 1. Electrón pasa del estado a través de mensajes hacia y desde las ventanas de las aplicaciones. Fuente: (Jensen, 2015)

Electron ejecuta las aplicaciones un tanto diferentes a las de NW.js, en el cual los proceso de las vistas o ventanas de la aplicación como también del back-end son ejecutados de forma separada, donde existe un intercomunicador entre estos dos procesos con la finalidad de enviar los datos entre las ventanas y el back-end.

NW.js (creado por Intel): Al igual que Electron permite crear aplicaciones de escritorio con HTML, CSS y Javascript también basado en Node.js en combinación con Chromium, brindando una gran capacidad de acceder a los recursos del hardware mediante sus respectivas APIs.

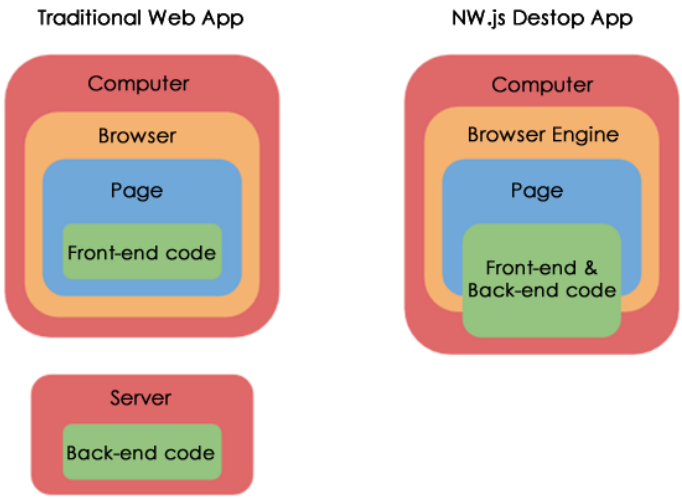

Ilustración 2. Estructura de aplicación de escritorio con NW.js. Fuente: (Jensen, 2015)

La estructura de una aplicación de escritorio realizada con NW.js es que un archivo linkeado dentro de un archivo html permite interactuar con el sistema operativo. Recursos locales del hardware y también con las páginas de la aplicación todo esto mediante las APIs de NW.js de forma unificada sin dividir el servidor de la aplicación renderizada como lo hacen los sitios web.

El siguiente es un diagrama simple que muestra cómo Node.js se ha combinado con WebKit con el fin de dar acceso a las aplicaciones NW.js tanto a la interfaz gráfica de usuario y el sistema operativo:

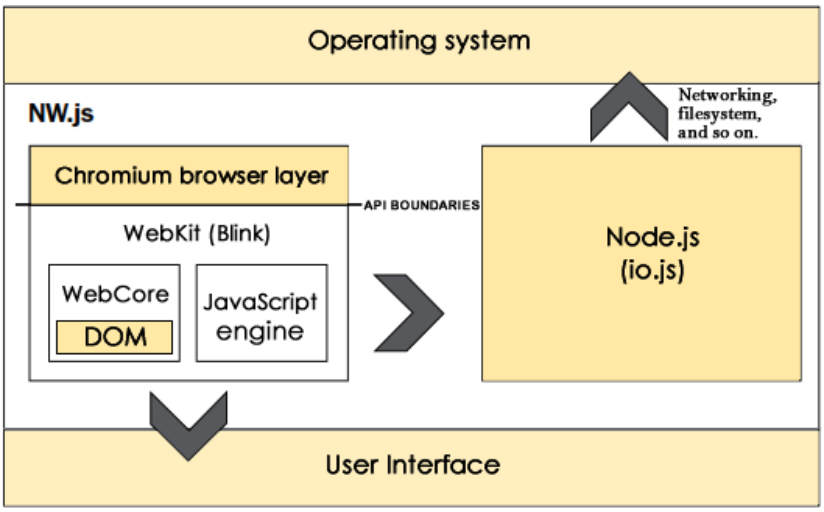

Ilustración 3. Diagrama que muestra cómo Node.js se ha combinado con WebKit. Fuente: (Benoit, 2015)

¿Qué puede hacer NW.js?

NW.js se basa en Chromium y Node.js. Se le permite llamar a los módulos de código y Node.js directamente desde el navegador y también utilizar las tecnologías Web en su aplicación. Además, puede empaquetar fácilmente una aplicación web para una aplicación nativa. (Nwjs, 2016).

Características de NW.js:

- NW.js permite realizar las aplicaciones de escritorio modernos usando HTML5, CSS3, JS, WebGL, y todo el potencial de Node.js, incluyendo el uso de módulos de terceros.
- La API nativa de interfaz de usuario le permite implementar aplicaciones nativas de aspecto similar con el apoyo de los menús, iconos de la bandeja portapapeles, y la unión de archivos.
- Desde Node.js y WebKit se ejecutan dentro del mismo hilo, NW.js tiene un rendimiento excelente.
- Con NW.js, es fácil migrar las aplicaciones web hacia las aplicaciones de escritorio.
- Gracias a la CLI y la presencia de herramientas es muy fácil depurar, empaquetar y desplegar aplicaciones en Microsoft Windows, Mac OS y Linux; y también es capaz de construir los ejecutables de la aplicación para cada sistema operativo de un solo código base.

Acceso a sistema operativo nativo de interfaz de usuario y API a través de JavaScript

Una buena aplicación de escritorio se integra muy bien en el sistema operativo del usuario; una aplicación de música trabajará con las combinaciones de teclas en el teclado multimedia de un usuario, una aplicación de chat puede tener un icono de menú en el área de la bandeja del sistema operativo y una aplicación de productividad puede proporcionar notificaciones cuando las acciones se han completado.

NW.js ofrece un gran API para obtener acceso a las funciones del sistema operativo, que haga lo siguiente:

- Controlar el tamaño y el comportamiento de la ventana de la aplicación.

- Mostrar una barra de herramientas nativo en la ventana de la aplicación con los elementos del menú.
- Añadir los menús de contexto en el área de la ventana de aplicación del botón derecho.
- Añadir un elemento de uso de la bandeja en el menú de la bandeja del sistema operativo.
- Acceder al portapapeles del sistema operativo leyendo y reestableciendo los contenidos, así como: abrir el archivo, carpetas y URLs en el ordenador utilizando sus aplicaciones por defecto, notificaciones de inserción a través del sistema de notificación del sistema operativo.

Construcción de app para sistemas operativos múltiple a partir de código base

Una de las características más útiles de NW.js es que a partir de un código base de su aplicación de escritorio, puede crear aplicaciones ejecutables nativos para Windows, Mac OS X y Linux. Esto es un ahorro de tiempo cuando se trata de desarrollar una aplicación que tiene que trabajar a través de múltiples plataformas. También significa que se puede tener un mayor control sobre cómo la aplicación se ve y se siente, en lo posible cuando se trata de apoyar a un sitio web para múltiples navegadores web.

El ejecutable nativo es capaz de ejecutar por sí mismo, y no requiere que el usuario tenga cualquier otro software instalado en su ordenador. Esto hace que sea fácil de distribuir la aplicación a los usuarios, incluso en tiendas de la aplicación, como la tienda de Apple y la tienda Steam donde se venden algunas aplicaciones y juegos NW.js.

El proceso de creación de una aplicación para un sistema operativo específico implica unos pocos argumentos de línea de comandos, pero hay algunas herramientas que simplifican el proceso para usted, tales como nw-builder toll.

Ejemplo de la herramienta nw-constructor puede construir ejecutables nativos de una aplicación NW.js para las versiones de 32 bits y 64 bits de Mac OS X y Windows.

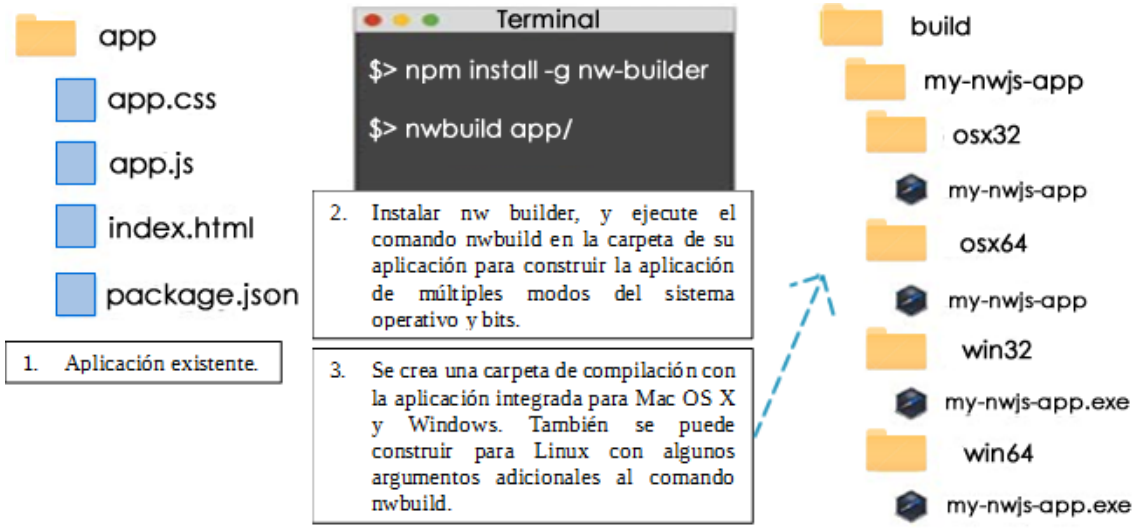

Ilustración 4. nw-constructor construye ejecutables de una aplicación NW.js para las vers. de 32 y 64 bits de Mac OS. Fuente: (Jensen, 2015)

¿Cómo funciona NW.js?

Para un usuario de NW.js, la idea de tener tanto en el front-end y back-end el código existente y que funcione en el mismo context JavaScript puede parecer un poco mágico. En la vista de un desarrollador, NW.js es una combinación de un framework de programación (Node.js) con el motor del navegador de Chromium a través del uso común de V8 (es un motor de JavaScript creado por Google para su navegador web), Google Chrome. Está escrito en C++ y se diseñó con el objetivo de acelerar la ejecución de JavaScript en el navegador web.

Cuando Node.js fue lanzado un año más tarde en el 2009, combinaba un soporte multiplataforma biblioteca llamada libuv con el motor V8, y proporcionaba una manera de escribir programas de servidor asíncronas en JavaScript. Debido a que tanto Node.js y Chromium V8 se utilizan para ejecutar su JavaScript, proporcionan una manera de combinar las 2 piezas de software juntos, esto es lo que Roger llegó a comprender y entender. A continuación se muestra un ejemplo de cómo esos componentes se combinan entre sí:

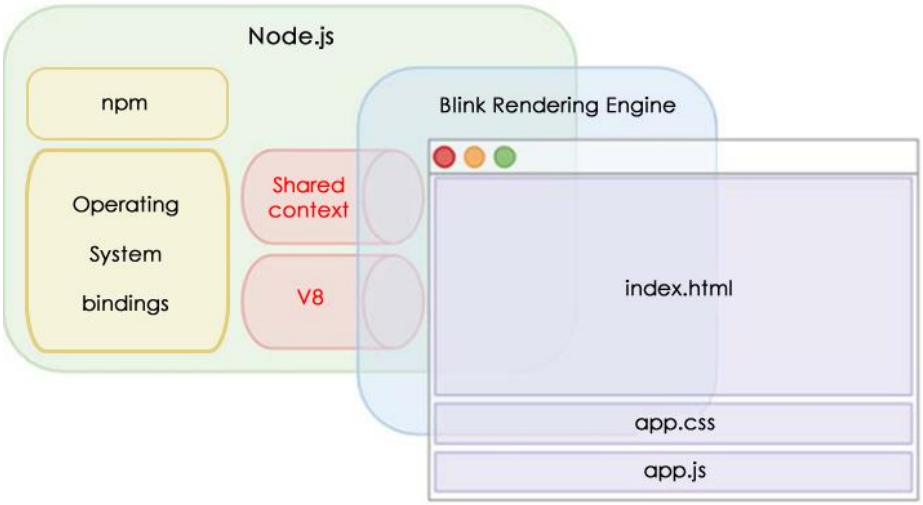

Ilustración 5. Arquitectura del componente NW.js en relación con la carga de una aplicación. Fuente: (Jensen, 2015)

Actividades para obtener Node.js y Chromium para trabajar juntos:

- Hacer Node.js y Chromium utilicen la misma instancia de V8.
- Integrar el bucle de eventos principales.
- Realizar un puente en el context JavaScript entre el nodo y Chromium.

Tanto Node.js y Chromium utilizan V8 para la ejecución de JavaScript. Tiene que realizar 2 cosas en orden para trabajar juntos. Lo primero que hace NW.js está cargando Node.js y Chromium, por lo que ambos tienen sus contexts JavaScript cargados en el motor V8. Nodo context JavaScript expondrá objetos globales y funciones tales como "module", “process”, y “require”. Context Chromium de JavaScript expondrá objetos globales y funciones como “windows”, “document”, y “console”.

Futuro de NW.js

NW.js evolucionará hacia arriba la arquitectura "app shell", en el futuro se va a dividir los componentes del navegador en forma de módulos que se pueden cargar por separado, por lo que el tamaño binario puede ser reducido significativamente. De esta manera se mejorará el esfuerzo

realizado para mover el mecanismo de extensión hacia la Content layer, así como los módulos que componen el navegador.

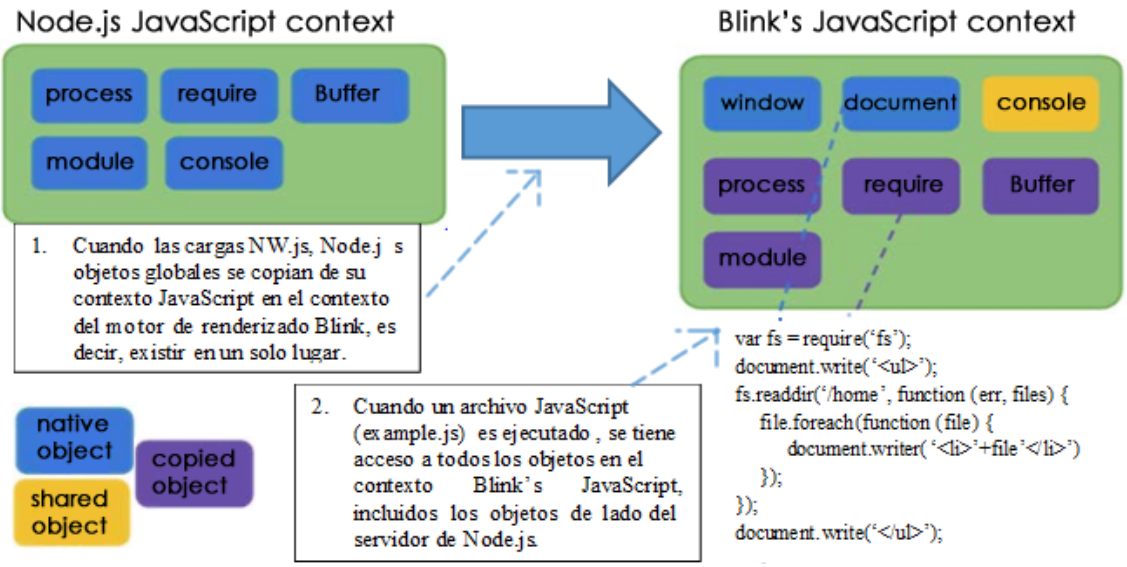

Ilustración 6. NW.js copia el context JavaScript para Node.js en context JavaScript de Chromium's. Fuente: (Jensen, 2015)

También se está usando para refactorizar la implementación de la biblioteca "nw.gui" en 0.12. El mecanismo de extensión proporciona una solución ligera y elegante para la unión JS API. Con él seremos capaces de eliminar la sobrecarga de mensajes adicionales IPC utilizados en la versión anterior. (NWJA, What's new in NW.js v0.13, 2016).

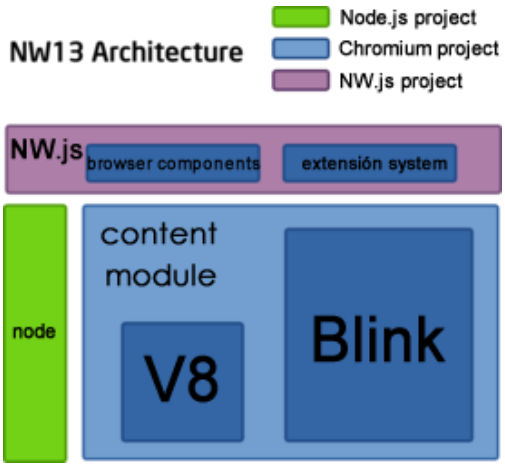

Ilustración 7. NW13 architecture. Fuentes: (Nwjs, What's new in NW.js v0.13, 2016)

CONCLUSIONES

El desarrollo de aplicaciones híbridas con tecnologías web, permite que los programadores de aplicaciones de este tipo no tengan que aprender nuevos lenguajes de programación, donde solo es necesario conocer la estructura, el contexto y las API's de los framework que brindan accesibilidad a los recursos del hardware y software, dando la posibilidad de reducir costos de creación de software, el empaquetado de las aplicaciones y su ejecución en las diferentes

plataformas. En comparación con las aplicaciones nativas y las aplicaciones web, las aplicaciones híbridas combinan las mejores características de estos dos tipos de software.

NW.js combina el front-end JavaScript y back-end JavaScript contexts, lo que permite que el código del lado del servidor y el código del lado del cliente accedan a los contextos de cada uno dentro del mismo archivo y environment. Esta es una principal diferencia de las aplicaciones web, donde los contextos se mantienen separados y los datos tienen que ser pasados a través de HTTP y WebSockets.

NW.js es una gran herramienta para el desarrollo de aplicaciones de escritorio, así como el uso del popular Node.js framework y npm ecosystem, y la forma en que podría proporcionar ejecutables nativos para los diferentes sistemas operativos desde una única base de código. Por último, hemos explorado un par de ejemplos de la vida real de NW.js en la naturaleza, y ©Manning Publications Co.

REFERENCIAS BIBLIOGRÁFICAS

Benoit, A. (2015). *NW.js Essentials.* Birmingham: Packt Publishing Ltd.

IBM, C. (01 de 04 de 2012). *El desarrollo de aplicaciones móviles nativas,*. Obtenido de ftp://ftp.software.ibm.com/la/documents/gb/commons/27754_IBM_WP_Native_Web_or_hybrid_2846853.pdf

Jensen, P. B. (2015). *Cross-Platform Desktop Applications.* Nueva York: Manning Publications Co.

LanceTalent. (20 de 02 de 2016). *Los 3 tipos de aplicaciones móviles: ventajas e inconvenientes*. Obtenido de https://www.lancetalent.com/blog/tipos-de-aplicaciones-moviles-ventajas-inconvenientes/

Lingras, P., Triff, M., & Lingras, R. (2016). *Building Cross-Platform Mobile and Web Apps for Engineers and Scientists: An Active Learning Approach.* Boston: Cengage Learning.

Nwjs, C. (23 de 03 de 2016). *NW.js*. Obtenido de http://docs.nwjs.io/en/latest/

Nwjs, C. (23 de 03 de 2016). *What's new in NW.js v0.13*. Obtenido de http://nwjs.io/blog/whats-new-in-0.13/

Pratt, D. (19 de 11 de 2012). *Getting Started with TideSDK*. Obtenido de http://tidesdk.multipart.net/docs/user-dev/generated/#!/guide/getting_started

Raona. (05 de Septiembre de 2014). *¿App nativa, web o híbrida?* Obtenido de http://blog.raona.com/app-nativa-web-o-hibrida

Towaha, S. O. (2016). *JavaScript Projects for Kids.* Birmingham: Packt Publishing Ltd.