Revista Tecnocientífica URU
Universidad Rafael Urdaneta
No. 18 Enero - Junio 2020
Depósito Legal: PPI 201402ZU4464
ISSN: 2343-6360
CC BY-SA 3.0 VE

Arquitectura basada en componentes web de cliente con múltiples capas, para el control de interfaces de aplicación

Luis Alfredo Acurero Quintero, Emilio Andrés Barboza Soto y Jubert Peréz

Escuela de Ingeniería de Computación. Facultad de Ingeniería. Universidad Rafael Urdaneta. Maracaibo, Venezuela.

Correo electronico: 13luismb@gmail.com eabs291197@gmail.com

Recibido: 11-10-2019 Aceptado: 12-01-2020

Resumen

En la presente investigación se desarrolló una arquitectura basada en componentes web de cliente con múltiples capas, para el control de interfaces de aplicación, lo cual se logró estableciendo las características y la especificación de requisitos del mismo, elaborando su arquitectura, desarrollando sus componentes y realizando pruebas de funcionamiento, implementando éste en el desarrollo de una aplicación simple demostrativa. El tipo de investigación fue de carácter descriptivo con diseño de campo o no experimental, apoyándose de la modalidad de Proyecto Factible. Se implementó la metodología ágil Scrum se utilizó en el proceso de análisis, diseño e implementación del proyecto. Esta metodología permite el control de las tareas necesarias para cumplir los objetivos. Como resultado de esta investigación se obtuvo una librería web basada en la arquitectura de múltiples capas que permite el envío estructurado de datos, cambiar el ambiente, registrar componentes, hacer ping a una ruta, activar un modo de depuración, entre otras características resaltantes de la librería. Se recomendó desarrollar futuras investigaciones relacionadas, utilizando ésta como antecedente o fuente de información.

Palabras clave: múltiples capas, componente web, desarrollo web.

Architecture based on client web components with multiple layers, for the control of application interfaces

Abstract

In the present investigation, an architecture based on client web components with multiple layers was developed, for the control of application interfaces, which was achieved by establishing the characteristics and the specification of requirements of the same, elaborating its architecture, developing its components and realizing performance tests, implementing this in the development of a simple demonstrative application. The type of research was descriptive with a field or non-experimental design, based on the Feasible Project modality. The agile methodology Scrum was used in the process of analysis, design and implementation of the project. This methodology allows the control of the tasks necessary to meet the objectives. As a result of this investigation, a web library was obtained based on the multilayer architecture that allows structured data sending, changing the environment, registering components, pinging a route, activating a debug mode, among other highlights of the bookshop. It was recommended to develop future related investigations, using this as a background or source of information.

Keywords: multiple layers, web component, web development.

Introducción

El desarrollo de software se ha convertido en una de las disciplinas más importantes en la actualidad, y avanza a grandes pasos, pues a medida que aumenta el consumo de productos de software por parte de la sociedad y la necesidad de proporcionar soluciones a los problemas del día a día, la tecnología se vuelve esencial.

Bajo esta premisa nace la necesidad de poder satisfacer la demanda de soluciones de software y cada día se crean nuevas herramientas que permitan cumplir con ese propósito; estas herramientas son pensadas para disminuir el tiempo de desarrollo, y en consecuencia, multiplicar la oferta disponible de soluciones.

En el caso de la programación web, hay varios marcos utilizados para simplificar ciertas tareas de programación, es decir, para simplificar el trabajo involucrado en el desarrollo tanto del código del lado del cliente como del lado del servidor web. Estos marcos de trabajo, mejor conocidos como *frameworks,* son herramientas útiles, pero poseen una falla fundamental: la falta de escalabilidad y mantenibilidad que presentan para grandes sistemas.

Una de las técnicas principales en el desarrollo de aplicaciones web *full stack* es el modelo vista controlador, el cual modulariza los sistemas en varias etapas para facilitar el mantenimiento y mantener la individualidad de cada uno de ellos como componentes independientes. La propuesta de esta investigación es realizar una implementación de una arquitectura de múltiples capas en formato de librería web para el lado del cliente, que permita disminuir los tiempos de desarrollo, permitiéndole al programador enfocarse explícitamente en el desarrollo de los componentes visuales y el mantenimiento de la lógica de negocio, conservando la separación por capas que la caracteriza como herramienta fundamental para una programación simple y eficiente, pensada para los programadores que necesitan de instrumentos de resolución de problemas intuitivos, de forma que puedan optimizar su tiempo de desarrollo.

Metodología de la investigación

La presente investigación es de tipo descriptiva debido a que es necesaria la recolección de datos e información para comprender cómo debe de funcionar un framework o librería web, basado en el uso de una arquitectura de múltiples capas.

El presente trabajo de investigación se realizó bajo la modalidad de proyecto factible, debido a que se desarrolló una librería con el fin de satisfacer la necesidad de una herramienta para la codificación de componentes mediante el uso de una arquitectura de múltiples capas.

El diseño de investigación es de campo, debido a que los datos fueron recolectados directamente de expertos en el área de desarrollo web, con énfasis en framework o librerías web, sin manipular o controlar la variable de alguna forma.

La metodología de desarrollo planteada para ejecutar esta investigación fue Scrum.

Análisis de los resultados

Se llevaron a cabo las fases de análisis, diseño, programación y pruebas.

Fase de análisis.

Para establecer las características del framework web se realizaron varias entrevistas durante las primeras semanas del desarrollo investigativo. Los entrevistados fueron los ingenieros Jubert Pérez y Oscar Colmenares; ambos profesionales altamente capacitados dentro del área de conocimiento de tec-

nología de información, y con conocimientos suficientes sobre el objeto de estudio o unidad de análisis de la presente investigación.

Las primeras entrevistas, las cuales se realizaron al ingeniero Pérez, permitieron comprender, de forma general, el comportamiento que debía tener cualquier implementación de la arquitectura de múltiples capas. Posteriormente, en las siguientes entrevistas, esta vez realizadas al ingeniero Colmenares, se asentaron las características de la librería propuesta en el presente estudio. Sin embargo, dichas características sufrieron cambios a lo largo de las entrevistas y fueron plasmadas en un cuaderno de notas.

Finalmente, se concluyó que la librería que implementa la arquitectura de múltiples capas debía permitir enviar de forma estructurada los datos, permitir cambiar entornos visuales, mantener registro de los componentes usados en cada vista, gestionar de manera inteligente el tiempo de procesamiento, ofrecer un componente base sobre el cual el usuario pueda plasmar sus elementos y añadir el uso de una consola.

Fase de diseño.

Luego de haber especificado los requerimientos funcionales y no funcionales de la librería web, se inició la fase de diseño de la misma; elaborando los distintos diagramas UML, esto con la finalidad de plasmar las características que debía poseer la aplicación.

Diagramas de casos de uso.

Los diagramas de caso de uso reflejan el comportamiento esperado de la aplicación ante la interacción con el usuario.

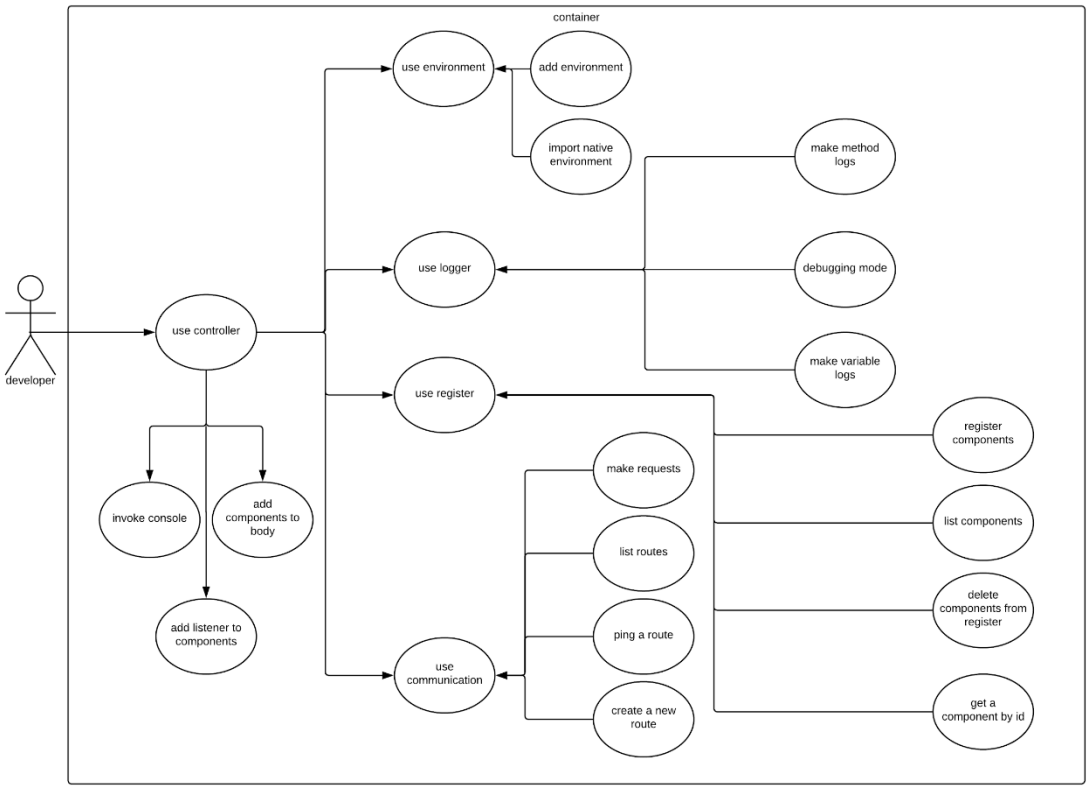

Figura 1. Diagrama de Casos de uso de la librería.

El diagrama de caso de uso que representa el comportamiento de un usuario respecto a la librería se representa en la figura 1. La librería se le presenta al usuario con dos componentes principales: el controlador y el componente visual que será la base de todos los componentes que el usuario desarrolle.

El componente visual será entonces extendido al resto de los componentes, donde el usuario deberá sobreescribir los métodos referentes a la UI del objeto, a su referencia en el DOM, sus opciones a la hora de enviar peticiones de ser necesario, y el método *callback* que será ejecutado al recibir una respuesta correcta desde el servidor.

El usuario luego es recibido por el controlador, una herramienta pensada para realizar toda la creación de componentes de la capa de presentación. En primera instancia, el usuario puede activar la consola nativa de la librería, donde puede ejecutar comandos; esto funciona primordialmente para evaluar el desarrollo de los componentes en tiempo real, y sin necesidad de modificar constantemente los scripts de la lógica.

El componente controlador se relaciona con el componente de comunicación, el cual es un componente encargado de crear las rutas a las cuales se dirigen las peticiones, así como listar todas las rutas disponibles, hacer las peticiones o sencillamente corroborar que la ruta especificada en el servidor funciona, sin necesidad de insertar un *listener* a ningún objeto. Este componente también se vale de su propio componente interno, que es el componente de estructura, encargado de estructurar el cuerpo de las peticiones de las formas que se desea, las cuales en este caso son: filas o columnas. El usuario hace uso de este componente mediante el controlador y añade un evento en el componente visual a una ruta listada y designada por el componente de comunicación.

Diagramas de componentes.

El diagrama de componentes que se presenta en la figura 2 representa la estructura deseada la librería. Éste describe la jerarquía de los niveles de los componentes, así como la manera en que los componentes interactúan entre sí. Se procedió a subdividir cada elemento de la arquitectura en unidades lógicas o componentes que pudiesen interactuar entre sí, para facilitar el proceso de desarrollo, la escalabilidad del sistema y reducir las dependencias.

La librería fue modelada de forma tal que los módulos tuvieran un alto grado de dependencia con el resto, utilizando sus capacidades de librería para forzar al usuario a la utilización correcta de los componentes de la misma. La librería puede descomponerse en tres grandes componentes, que a su vez contienen en total seis elementos fundamentales para el funcionamiento apropiado de la herramienta.

Los componentes de alto nivel son:

- Componente Visual
- Controlador
- Comunicación

Mientras tanto, ellos contienen los componentes de bajo nivel, los cuales son:

- Componente
- Registro

- Logger
- Consola
- Entorno
- Estructura

Los componentes visuales deben ser extendidos de una clase padre, que contiene métodos específicos que deben ser sobreescritos para el uso de todas sus capacidades. Los componentes son los bloques de construcción de cualquier aplicación que implemente esta librería y una aplicación típica de esta índole tendrá muchos de estos. En pocas palabras, un componente es una clase o función de JavaScript que acepta entradas, es decir, propiedades y devuelve un elemento que describe cómo debería aparecer una sección de la interfaz de usuario (UI).

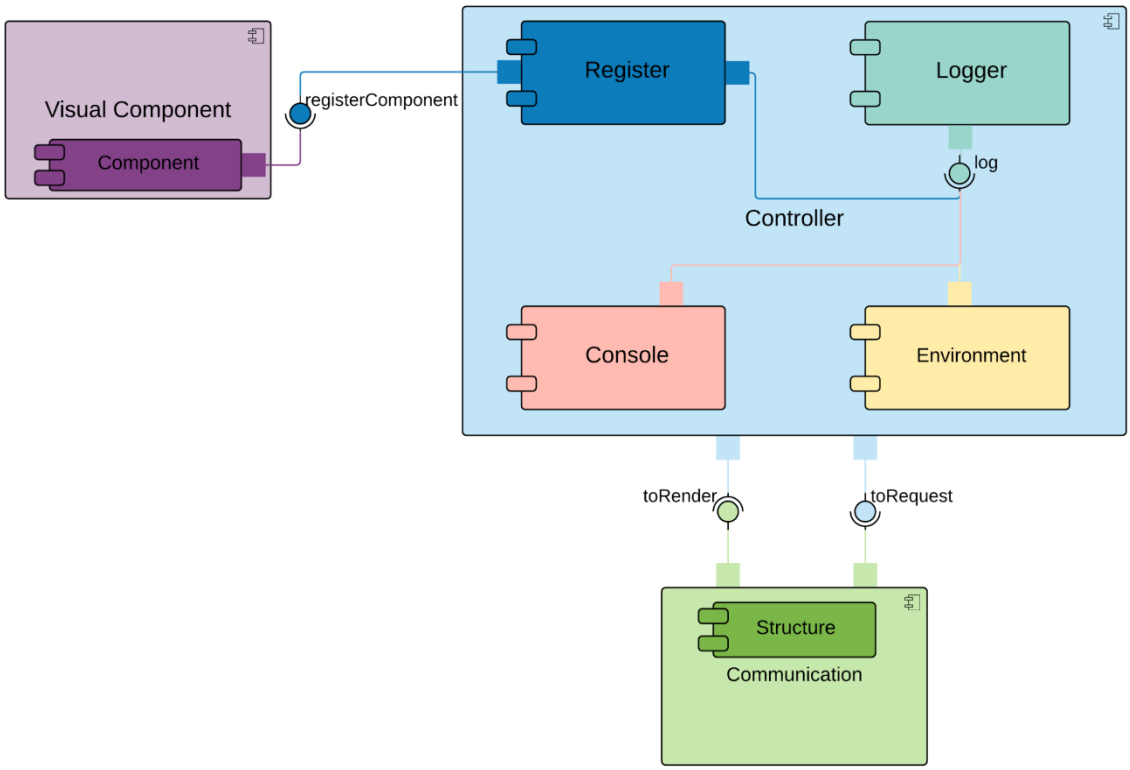

Figura 2. Diagrama de Componentes de la librería.

El componente controlador se encarga de manejar la mayoría de la lógica, pues actúa como la capa de negocio en un modelo vista controlador, sólo que en esta instancia se realiza en el lado del cliente. Hay dos razones principales para crear un componente controlador: para desacoplar el estado de la presentación, de modo que el estado de un componente se pueda almacenar incluso mientras no se muestra al usuario y para facilitar la reutilización de la lógica de negocio. Este elemento sólo se encarga de validar los componentes e insertarlos en el DOM de la página donde se llama, así como manejar el estado de los mismos.

El componente de comunicación será el encargado de realizar peticiones al servidor, proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas. También provee un método global que proporciona una forma fácil y lógica de obtener recursos de for-

ma asíncrona por la red. Funciona como contenedor de la API Fetch, de forma que retorna promesas ya resueltas por el método *fetch()*.

Diagrama de clases.

El diagrama de clases describe la estructura de la librería que implementa la arquitectura basada en componentes web de cliente con múltiples capas, ya que en este se representan directamente las clases del software con sus interacciones e interfaces. Por lo que, la elaboración de este diagrama fue crucial en la fase de análisis, sirviendo como guía para el desarrollo de la librería.

En este diagrama, representado en la figura 3, se plasman las clases que conforman la librería, las cuales son:

• Controller: es la clase encargada de integrar el resto de los componentes para realizar múltiples funciones en las cuales se encuentran el registro de los componentes, cambio de ambientes, creación de la consola, inserción de componentes en el documento, entre otras cosas.

• Component: es la clase que funciona como interfaz principal para los componentes visuales.

• Communication: es la clase que tiene la responsabilidad de realizar todo el procesamiento de solicitudes generadas desde el cliente.

• Structure: es la clase encargada de la deserialización de los cuerpos de las solicitudes.

• Environment: es la clase encargada de controlar todas aquellas acciones relacionadas al entorno de la aplicación.

• Logger: es la clase encargada de funcionar como mecanismo de bitácora para ayudar al desarrollador a la hora de la depuración.

• Register: es la clase encargada de hacerle seguimiento a los componentes de la aplicación.

• Console: es una clase que extiende de la interfaz componente y emula el funcionamiento de una command line interface de Linux.

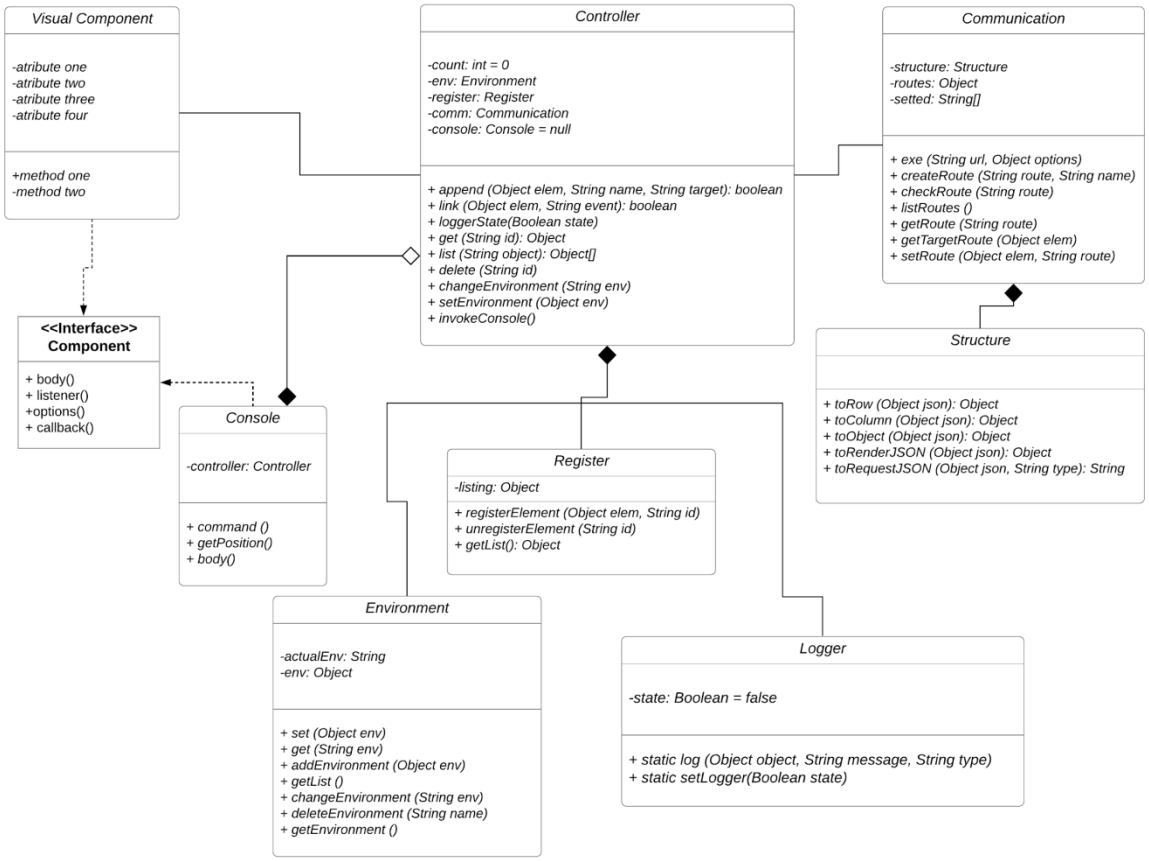

Figura 3. Diagrama de Clase de la librería.

Fase de programación.

El cumplimiento de este objetivo se constituye por la implementación de las herramientas de desarrollo para codificar las instrucciones que llevarán el diseño a cabo. La metodología de desarrollo designada recomienda codificar los módulos de la librería de manera tal que el cliente pueda apreciar algunas interfaces del sistema antes de culminar el proceso de desarrollo.

Desarrollo de los módulos que contienen los componentes visuales de la librería.

El inicio del desarrollo se realiza en el área de los componentes visuales, donde se definió un componente base como interfaz necesaria para la creación de elementos personalizados compatibles con el resto de los componentes de la librería. Adicional a eso, se codificaron cuatro componentes visuales implementando la interfaz del componente base como demostración de su funcionamiento, además añadiéndolos como elementos nativos de la librería, perfectamente utilizables por usuarios nuevos en el uso de la misma.

Los componentes visuales mencionados se definen a continuación:

Componente base (Component).

Este componente, a pesar de que el concepto en JavaScript es inexistente, fue pensado para funcionar como una interfaz con métodos que requieren implementación al heredarlo a otros componentes. Su uso se definió como obligatorio para todos los otros componentes que pudiera tener la aplicación, restringiendo al usuario de formas alternas de crear componentes visuales.

Component, el nombre de la interfaz por conveniencia, contiene tres métodos principales: body, listener y callback. El método *body()*, en el cual se definirá el cuerpo del componente, su presencia visual en el HTML, permitiendo uso de las funciones nativas del manejo del DOM de JavaScript, tales como *innerHTML* o *createElement* en el retorno del mismo.

Por otra parte, el método *listener()* se encarga únicamente de devolver la referencia del DOM del elemento que extiende. Y finalmente, *callback()* será el método que se ejecute luego de que el servidor responda una petición a una ruta a la que apunte el componente.

```
export class Component {
  constructor() {}

  body() {
    throw new Error("body method must be implemented on child component");
  }

  callback() {
    throw new Error("callback method must be implemented on child component");
  }

  listener(){
    return document.getElementById(this.id)
  }
}
```

Figura 4. Componente base (Interfaz: Component)

Adicionalmente, se crearon los componentes Input, Card, Form y List utilizando como base el componente principal de la librería.

Desarrollo de los módulos relacionados con la comunicación de la librería.

El módulo de comunicación representa el intercambio principal de datos entre la librería y el servidor, mientras mantiene en sí misma y asigna a los componentes una ruta definida. Al describir este componente, sólo se define un componente interno: el componente de estructura. Éste se encarga no sólo de estructurar el cuerpo de la petición a ser enviadas al servidor, sino que también se le encomienda la tarea de estructurar la respuesta del mismo a conveniencia del programador. A continuación, se explica el proceso de desarrollo de los mismos:

Componente de Estructura (Structure).

El componente de estructura nace de la necesidad de automatización del análisis y reconversión de los JSON que se intercambian con el servidor. En él se definen métodos que convierten los cuerpos de la petición, o respuestas del servidor, en objetos JavaScript con tres presentaciones distintas.

Este elemento está constituido por un método pensado exclusivamente para la renderización, y otro método diseñado para el envío de datos al servidor. En ellos se implementan las técnicas de reconversión mencionadas anteriormente, los cuales son verificados en las figuras 5 y 6.

```
const body = {
    name: 'john',
    lastname: 'doe',
    age: 32,
    email: 'johndoe@foobar.com',
    password: '123456'
}

JSON.stringify(body);
```

```
import { Structure } from "./lib/communication/structure/structure";

const body = {
    name: 'john',
    lastname: 'doe',
    age: 32,
    email: 'johndoe@foobar.com',
    password: '123456',
    gender: true
}

const structure = new Structure()
requestBody = structure.toRequestJSON(body, 'column');

//output
    requestBody = {
        objectName: 'foo',
        methodName: 'bar',
        params: ['john', 'doe', 32, 'email', '123456', true],
        types: ["string", "string", "number", "string", "string", "boolean"]
    }
```

Figura 5. Envío tradicional de body (izquierda) vs. Envío de body del componente de estructura (derecha)

```
> const JSON = {
      status: 200,
      msg: 'got data',
      fields: ['id', 'name', 'lastname'],
      data: [[1,2,3],['luis', 'emilio', 'andres'],['acurero', 'emilio', 'mora']]
  }
< undefined
> toObject(JSON)
< ▼{id: Array(3), name: Array(3), lastname: Array(3)}
    ▶id: (3) [1, 2, 3]
    ▶lastname: (3) ["acurero", "emilio", "mora"]
    ▶name: (3) ["luis", "emilio", "andres"]
    ▶__proto__: Object
```

Figura 6. Reconversión de la respuesta del servidor

Componente de Comunicación (Communication).

El objetivo del componente de comunicación es encargarse de toda la comunicación con el servidor. Esto se logra partiendo de la idea de que sea el mismo componente el responsable de tener dentro de su estado todas las rutas que serán utilizadas por el usuario.

Al momento de la instancia del componente, es posible añadirle un archivo JSON, del cual extraerá rutas generales para todas las páginas donde se utilice; sin embargo, el elemento posee la capacidad de añadir o eliminar rutas dentro de su estado en cualquier momento, también dándole la potestad de relacionarlo con un componente, de forma que dicho elemento no tenga ningún tipo de contacto con la configuración de la ruta a la cual se dirige.

Este componente, además, tiene la capacidad de evaluar el estado de las rutas que contiene, para comprobar su funcionamiento antes de asociarlo a algún elemento. Dentro de él, posee una instancia del componente de estructura, necesario para transformar los cuerpos de las peticiones así como las respuestas del servidor al momento de utilizar el método que finalmente lo comunica con el servidor para realizar una operación.

Desarrollo de los módulos relacionados con el control de la librería.

La codificación de los módulos de la librería concluye por el componente principal: el controlador de la aplicación; en el cual se basará la mayoría del funcionamiento de las aplicaciones que implementen

la misma. Como ya se ha explicado con anterioridad, este componente es el que presenta el mayor grado de granularidad, por lo cual, a este punto del desarrollo, su realización representaría la finalización del mismo.

Este elemento posee cuatro componentes con los que realiza gran parte de las operaciones definidas en la fase de diseño, más el componente que representa su definición propia, el cual es explicado a continuación:

Componente controlador (Controller).

La unidad final de la etapa de desarrollo es el elemento que funciona como núcleo de la librería, y es la representación de la capa de negocios en la arquitectura de múltiples capas; y en él se sostiene el óptimo funcionamiento de la implementación de la misma.

El componente controlador es una fachada donde trabajan internamente los elementos que lo componen, convergiendo entre ellos para realizar el procesamiento de datos. El objeto descrito ofrece dos servicios principales: el primero, permite tomar una nueva instancia de un componente visual para luego asociar una propiedad que lo identifique como único en el documento, la cual puede ser generada automáticamente por el controlador o manualmente por el usuario y durante ese proceso, registrarlo para poder manipular o sólo hacer seguimiento a la referencia del mismo.

Por otra parte, el segundo servicio recibe el componente registrado, al cual le añade una función que le permite realizar peticiones a un endpoint del servidor, valiéndose de las rutas previamente definidas y asociadas al mismo por el componente de comunicación, evaluando el estado de la respuesta del servidor para ejecutar el *callback* propio del componente ante las peticiones o simplemente lanzar un error.

Sin embargo, el controlador ofrece servicios adicionales pertinentes a la construcción de las aplicaciones que serán realizadas utilizándolo como herramienta principal para optimizar el desarrollo en términos de horas hombre involucradas. Entre estas acciones, se encuentran la capacidad de activar las bitácoras para depurar, obtener los estados actuales del entorno y la lista de componentes, eliminar un componente del documento, asociar un json con el estado inicial del entorno de la aplicación, añadir y cambiar entornos e invocar y cerrar una instancia de la consola.

```
const controller = new Controller();
const card = new Card();
const list = new List();
const input = new Input();
const form = new Form();
controller.append(form)
          .append(card)
          .append(card)
          .append(consoledev, "console-1")
          .append(list, "list-1")
          .append(input, "input-1")
controller.loggerState(true);
controller.send(card);
controller.setEnvironment({green:'./green.css'})
```

Figura 7. Código asociado al uso del componente controlador.

Fase de pruebas.

Las pruebas al final del proceso de desarrollo son una práctica recomendada para determinar la efectividad y alcance del software desarrollado, las pruebas se realizan con la finalidad de detectar imperfecciones, fallas y posibles errores a tiempo y poder corregirlos. Es por esto que el cumplimiento del objetivo se basa únicamente en la realización de las pruebas y la representación de los resultados de las mismas.

Pruebas unitarias.

Se realizaron pruebas unitarias al momento del desarrollo de cada componente para corroborar su funcionamiento y garantizar su estabilidad y eficiencia, de modo que cada elemento funciona por separado sin ningún problema, verificando que el código hace lo que tenía planificado, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, y que si el estado inicial es válido, entonces el estado final es válido también, y en base a todo eso, concluir que dichas pruebas fueron superadas satisfactoriamente. En la tabla 1 se describen las pruebas de unidad realizadas a los módulos pertinentes.

Tabla 1. Pruebas unitarias de la librería.

Componente	Pruebas
Registro	Registrar, anular registro de componentes
Entorno	Inicialización de entorno y cambio de entorno
Logger	Activación de bitácora
Consola	Invocación y comandos de consola
Controlador	Inserción de componentes al DOM y de listeners
Estructura	Métodos de reconversión de JSON (objeto, fila, columna)
Comunicación	Peticiones al servidor de forma estructurada

Pruebas de aceptación.

Al final de todo el desarrollo y de las pruebas, se realizaron pruebas de aceptación junto al tutor académico, con el fin de determinar que la librería cumple con los requerimientos y necesidades planteados al momento de la investigación, la prueba realizada arrojó los resultados esperados y sin encontrar ningún error en tiempo de ejecución. El desarrollo de la aplicación de prueba fue extremadamente sencilla gracias a las herramientas proporcionadas por la librería, permitiendo una óptima capacidad de mantenimiento y escalabilidad debido a su manera amigable de operar con ella, permitiéndole al programador enfocarse únicamente en el desarrollo de los componentes, ya que *Padoru.js* logró mantener eficientemente el control general de la aplicación, así como el procesamiento de peticiones al servidor e incluso, el entorno visual del usuario.

Conclusiones

El desarrollo de este proyecto tiene su origen en la necesidad de conocer y difundir información sobre la manera en cómo se desarrollan las arquitecturas de múltiples capas con un propósito. La intención de esta investigación es promover el desarrollo de innovadores librerías basadas en la arquitectura de múltiples capas que faciliten el trabajo a los desarrolladores para reducir su tiempo de elaboración y complejidad a la hora de codificar. Entonces, se concluye que ese es el motivo por el cual existe esta investigación.

En primer lugar, se cumplió con un proceso de captura de información con la finalidad de definir el alcance de la arquitectura propuesta, es decir, las características, especificaciones y requisitos necesa-

rios para lograr el resultado final. Se procedió con la elaboración del diseño de la misma, en la cual se realizaron los componentes que la conforman y las interacciones entre sí.

Se desarrollaron cada uno de los componentes en lenguaje JavaScript, implementando el estándar ECMAScript 6. A su vez, es importante indicar que cada componente fue creado para cumplir con una tarea específica requerida para el funcionamiento de la arquitectura. Ésto para alcanzar el objetivo de brindarle al desarrollador una herramienta sencilla que le facilite realizar operaciones repetitivas, típicas de la programación, ofreciéndole la oportunidad de enviar de forma estructurada los datos, cambiar el ambiente, registrar componentes, hacer ping a una ruta, activar un modo de depuración, entre otras características resaltantes del modelo, mediante la implementación de una nueva librería basada en la arquitectura de múltiple capas para el desarrollo web.

Una vez culminado el proceso de desarrollo de la librería, se implementó éste en la creación de una aplicación simple que permitiese la subida de fotos, su visualización, filtrado por tipo y su modificación de ser el creador del contenido. Como era de esperarse, los resultados lograron llenar las expectativas. Se demostró la sencillez de su uso, el aumento de productividad al reducir los tiempos de trabajo y el logro de los objetivos planteados y por medio de múltiples pruebas realizadas se corroboró que los componentes cumplieran las funciones para las cuales fueron diseñados, así como el límite de procesamiento actual que posee esta herramienta.

Referencias Bibliográficas

Alegsa, L. (2016). Obtenido de Alegsa: http://www.alegsa.com.ar/Dic/arquitectura_de_sistemas.php

Sanz, A. (2016). Obtenido de GlobalCC: https://www.globalcc.es/blog/la-historia-del-diseno-web/

Schmuller, J. (1999). Aprendiendo UML en 24 Horas. Naucalpan de Juárez: Pearson Educación Latinoamérica.

Staff, P. E. (17 de Mayo de 2018). Obtenido de Packt Hub: https://hub.packtpub.com/what-is-multi-layered-software-architecture/

Tebar, E. (2018). Obtenido de We Are Marketing: https://www.wearemarketing.com/es/blog/frameworks-en-el-desarrollo-web-las-mejores-practicas-para-tu-negocio-online.html