*FUNDACIÓN UNIVERSITARIA TECNOLÓGICO COMFENALCO*

*INGENIERÍA DE SISTEMAS*

*ING. Carlos Garcia Castro*

*Presentado por:*

*JUAN ANDRES WELLMAN AMAYA*

*CARLOS DANIEL ZULBARAN SARMIENTO*

*DARWIN PINTO*

*CARTAGENA DE INDIAS D.T.C*

*2022*

**1. Marco Teórico Sobre lenguajes y expresiones regulares**

**Lenguaje JavaScript**

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas​ y JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

**Framework React**

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. En el proyecto hay más de mil desarrolladores libres.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; React es la Vista en un contexto en el que se use el patrón MVC (Modelo-Vista-Controlador) o MVVM (Modelo-vista-modelo de vista). También puede ser utilizado con las extensiones de React-based que se encargan de las partes no-UI (que no forman parte de la interfaz de usuario) de una aplicación web.

React te ayuda a crear interfaces de usuario interactivas de forma sencilla. Diseña vistas simples para cada estado en tu aplicación, y React se encargará de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien.

Las vistas declarativas hacen que tu código sea más predecible, por lo tanto, fácil de depurar.

**Expresiones Regulares**

Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayores, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para ver las coincidencias. El objetivo de las expresiones regulares es representar todos los posibles lenguajes definidos sobre un alfabeto Σ, en base a una serie de lenguajes primitivos, y unos operadores de composición.

Dado un alfabeto Σ, las expresiones regulares sobre Σ se definen de forma recursiva por las siguientes reglas:

1. Las siguientes expresiones son expresiones regulares primitivas:

∅

Λ

a, siendo a ∈ Σ

2. Sean α y β expresiones regulares, entonces son expresiones regulares derivadas:

α+β (unión)

α.β (o simplemente αβ) (concatenación)

α\* (cierre)

(α)

3. No hay más expresiones regulares sobre Σ que las construidas mediante estas reglas

**2. Justificación de la elección del lenguaje de programación**

La web funciona con este lenguaje

Una de las grandes ventajas que ofrece JavaScript es que la mayor parte de las aplicaciones web funcionan con este lenguaje, y se debe en gran medida a que los navegadores permiten la interacción entre aplicaciones.

Este atributo le ha valido a JavaScript el reconocimiento de ser conocido en todo el orbe como el “lenguaje de programación de la web”, ojo, no es la única aplicación para hacer estos menesteres pero sí la más usada y popular de todas las existentes.

Se trata de uno de los lenguajes más fáciles para iniciarse en el mundo de la programación. Por su sintaxis, que es fácil de entender y porque te permite hacer muchas cosas con un solo lenguaje y es más compleja para las expresiones regulares.

**3. Objetivo de las principales estructuras de datos utilizadas**

Principalmente trabajamos con variables tipo String: para poder capturar los datos que ingresa el usuario en el formulario, cada vez que pulsa una tecla en tiempo real se captura la información que a futuro será validada.

Object: para poder exportar en un esquema de tipo json, con las cabeceras predefinidas en cualquiera de los formatos disponible, por ejemplo, { data, fileName, fields, exportType }

Array de String, la razón es para poder controlar los campos a importar y exportar, por ejemplo, let fields = ["nombres","codigoEstudiante","fechaIngreso", "direccion","telefono","celular","correoElectronico"];

usamos variables y constantes para poder manejar de manera óptima el scope de cada una de ellas.

**4. Objetivo de los principales procedimientos y funciones**

Nos basamos en programación funcional y no orientado a objetos ni el uso de clases, usamos mucho las famosas “arrow functions” para ejecutar los procesos y el funcionamiento, de leer y exportar, ya sea en excel, xml, json o txt.

Para renderizar el html en la web, usamos funciones de React para poder ejecutar una función dinámica y que me retorne algo en el DOM (document object model)

Básicamente usamos librerías para poder exportar a estos formatos, sin embargo se implementó una lógica a la medida de lo que se estaba requiriendo, por ejemplo, se usó un solo botón para importar y el programa es capaz de determinar en qué formato viene y representarlo en una tabla que se genera. Para eso usamos ciclos como “forEach” que es un ciclo “for” mas cómodo para poder recorrer las filas de un excel o de la compleja estructura de un xml, que se representa de una manera totalmente distinta a cualquiera de los otros formatos.

Nos hemos basado en componentes se crean componentes encapsulados que manejen su propio estado, y se convierte en interfaces de usuario complejas.

Ya que la lógica de los componentes está escrita en JavaScript y no en plantillas, puedes pasar datos de forma sencilla a través de tu aplicación y mantener el estado fuera del DOM.

Los componentes de React implementan un método llamado render() que recibe datos de entrada y retorna qué mostrar. Este ejemplo utiliza una sintaxis similar a XML llamada JSX. Puedes acceder a los datos de entrada que se pasan al componente mediante render() a través de this.props.

JSX es opcional y no es requerido para usar React. Prueba Babel REPL para ver el código JavaScript sin procesar la compilación de JSX.

**5. Descripción de las expresiones regulares utilizadas**

Estos patrones se utilizan con los métodos exec() y test() de RegExp, y con match(), matchAll(), replace(), replaceAll() (en-US), search() y split() métodos de String.

Un patrón de expresión regular se compone de caracteres simples, como /ABC/, o una combinación de caracteres simples y especiales, como /ab\*c/ o /Capítulo (\d)\.\d\*/. El último ejemplo incluye paréntesis, que se utilizan como dispositivos de memoria.

Estos métodos se explican en detalle en la [referencia de JavaScript](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference).

| **Método** | **Descripción** |
| --- | --- |
| [exec()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec) | Ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un arreglo de información o null en una discrepancia. |
| [test()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test) | Prueba una coincidencia en una cadena. Devuelve true o false. |
| [match()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/match) | Devuelve un arreglo que contiene todas las coincidencias, incluidos los  grupos de captura, o null si no se encuentra ninguna coincidencia. |
| [matchAll()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll) | Devuelve un iterador que contiene todas las coincidencias, incluidos  los grupos de captura. |
| [search()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/search) | Prueba una coincidencia en una cadena. Devuelve el índice de  la coincidencia, o -1 si la búsqueda falla. |
| [replace()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/replace) | Ejecuta una búsqueda por una coincidencia en una cadena y reemplaza la subcadena coincidente con una subcadena de reemplazo. |
| [replaceAll() (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replaceAll) | Ejecuta una búsqueda de todas las coincidencias en una cadena y  reemplaza las subcadenas coincidentes con una subcadena de reemplazo. |
| [split()](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/split) | Utiliza una expresión regular o una cadena fija para dividir una cadena en un arreglo de subcadenas. |

las expresiones regulares usadas fueron estas:

**nombres:** *Debe empezar por letra y terminar por letra Las palabras deben separarse como mucho por un espacio, busca al inicio que empiece por una palabra (letras repetidas 1 o más veces) a continuación, se repite el siguiente conjunto: ( un espacio + una palabra ) de cero a varias veces. Finalmente, debe terminar con fin de línea/string.*

"^[a-zA-ZáéíóúÁÉÍÓÚñÑ]+(?: [a-zA-ZáéíóúÁÉÍÓÚñÑ]+)\*$"

**codigo estudiante:** *debe contener solo numeros y estrictamente 8 caracteres*

"^[1-9]{8}$"

**fecha ingreso:** *dd/mm/yyyy*

/^(?:(?:(?:0?[1-9]|1\d|2[0-8])[/](?:0?[1-9]|1[0-2])|(?:29|30)[/](?:0?[13-9]|1[0-2])|31[/](?:0?[13578]|1[02]))[/](?:0{2,3}[1-9]|0{1,2}[1-9]\d|0?[1-9]\d{2}|[1-9]\d{3})|29[/]0?2[/](?:\d{1,2}(?:0[48]|[2468][048]|[13579][26])|(?:0?[48]|[13579][26]|[2468][048])00))$/

**dirección:** *admite letras o numeros y simbolos como # - para las direcciones*

"^[A-Z a-z0-9ÑñáéíóúÁÉÍÓÚ\\-\\#]+$"

**teléfono:** *admite solo numeros con una longitud de 7 caracteres*

"^[0-9]{7}$"

**celular:**  *admite solo numeros y tiene que empezar por 3 con una longitud de 10 caracteres*

"^3[0-9]{9}$"

**correo electrónico:** *En general, un correo electrónico debe tener los siguientes elementos Empezar por el identificador o nombre del usuario ^\w+([.-\_+]?\w+)\* Seguido por el símbolo de la arroba @ Por último, el nombre del dominio del correo \w+([.-]?\w+)\*(\.\w{2,10})+$*

/^\w+([.-\_+]?\w+)\*@\w+([.-]?\w+)\*(\.\w{2,10})+$/

**6. Si tuvo dificultades, cómo las resolvió.**

Realizando nuestro programa tuvimos dificultades al momento de leer el XML al Importar y Exportar También al momento de hacer las validaciones del correo electrónico, con el grupo de trabajo pudimos solucionar investigando con la fuentes open source gracia a esa fuente encontramos toda la soluciones de nuestras dificultades.

ya que la documentación en este lenguaje usando este framework (react) es muy amplia y ya se ha trabajado con anterioridad este tipo de problemas con xml.

se tuvo una dificultad para crear la expresión regular para las fechas sin embargo en internet un repositorio en la nube explicaba como se tenía que elaborar y que tipo de validaciones en diferentes formatos, y así, se pudo sacar esa validación de fecha.