




TRABAJO PRÁCTICO N°6

“REQUERIMIENTOS ÁGILES - Implementación de user stories”



Universidad Tecnológica
Nacional
Facultad Regional Córdoba



GRUPO N°6:
Calvo, Miguel
Candusso, Tatiana
Cesar, Manuel
Méndez, Ignacio
Otero, Gastón
Villafuerte, Jorge

Contenido

Historial de cambios	3
Objetivo	4
Definición del IDE a utilizar	4
Estándares de Programación	4
Comentarios	4
Sentencias de importación	5
Declaración de clases	5
Métodos	5
Variables	6
Constantes	6

Historial de cambios

VERSIÓN	FECHA	AUTOR	DESCRIPCIÓN / COMENTARIOS
1.0	14/09/2021	Grupo nº6	Versión inicial

Objetivo

El presente documento tiene por objetivo definir los estándares y normas de trabajo para el proceso de desarrollo.

Se determinan a continuación normas de trabajo para la escritura y el mantenimiento del código fuente de la aplicación. Desde elementos simples como la declaración de clases y variables hasta comentarios, sentencias de importación, métodos entre otros.

Framework

El framework que se utilizó es React Native, el cual permite desarrollar aplicaciones nativas para iOS y Android. Es muy flexible y los desarrolladores pueden crear componentes guiados por su imaginación. Cada componente debe estar empaquetado en una sola función y usar sus propios métodos para cumplir con su parte.

Definición del IDE para utilizar

Se propone utilizar como entorno de desarrollo “Microsoft Visual Studio” que es una distribución de la empresa “Microsoft”. La decisión respecto de utilizar este IDE frente a los numerosos existentes se basa en que el mismo es una opción gratuita, versátil (gracias a los numerosos plugin que pueden instalarse), liviano, con una interfaz de usuario intuitiva y con un estilo claro que facilita la comprensión del texto. Además, posee una amplia variedad de complementos fáciles de instalar que permite escribir mejor código con mayor facilidad. El mismo cuenta con IntelliSense para el lenguaje elegido y una integración con nuestro debugger, lo cual facilitaba aún más el proceso de desarrollo.

Estándares de Programación

A continuación, se definen los estándares o convenciones de programación que se emplearán durante todo el desarrollo.

Estas normas son muy útiles por muchas razones, entre las que destacan:

- Facilitan el mantenimiento de una aplicación. Dicho mantenimiento constituye aproximadamente el 80% del coste del ciclo de vida de la aplicación.
- Permiten que cualquier programador entienda y pueda mantener la aplicación. En muy raras ocasiones una misma aplicación es mantenida por su autor original.
- Los estándares de programación mejoran la legibilidad del código, al mismo tiempo que permiten su comprensión rápida.

Comentarios

Para realizar los comentarios en el código se utiliza JSDoc, dentro del comentario se puede utilizar etiquetas. Las etiquetas son las siguientes:

Etiqueta	Descripción
@author	Nombre del desarrollador
@constructor	Marca una función como constructor
@deprecated	Marca un método como obsoleto
@exception	Sinónimo de @throws
@exports	Identifica un miembro exportado por el módulo.
@param	Documenta un parámetro de método; se puede agregar un indicador de tipo de datos entre llaves
@private	Significa que un miembro es privado
@returns	Documenta un valor de retorno
@return	Sinónimo de @returns
@see	Documenta una asociación a otro objeto.
@todo	Documenta algo que falta / está abierto
@this	Especifica el tipo de objeto al que se <i>this</i> refiere la palabra clave dentro de una función.
@throws	Documenta una excepción lanzada por un método
@version	Proporciona el número de versión de una biblioteca.

A continuación, se muestra un ejemplo de comentario:

```
/**
 * Generates a secure URL from Cloudinary from the selected event picture.
 * @param imageData Picture information coming from Expo Camera selected Image.
 * @returns URL from Cloudinary to be stored in MongoDB
 */
```

Sentencias de importación

Al comienzo de cada archivo se incluirán las sentencias de importación de los paquetes necesarios. En este proyecto se prioriza utilizar exportaciones del tipo const, para poder importar solamente los componentes requeridos en cada módulo, aunque para las constantes del módulo 'constants', si se utilizan los imports y exports defaults.

"EJEMPLO"

```
export const StyledButton = ...
import {StyledButton} from '../components/buttons'

export default headerColors = ...
import headerColors from '../constants/colors'
```

Variables

Las variables se escribirán en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra en mayúscula.

Las variables nunca podrán comenzar con el carácter “_” o “\$”. Los nombres de variables deben ser cortos y sus significados tienen que expresarse con suficiente claridad.

Se prioriza el uso de variables del tipo ‘let’ o ‘const’ de la versión ES6 de JavaScript sobre el tipo ‘var’ de versiones anteriores.

```
“ejemplo”  
let paymentType = “cash”
```

Constantes

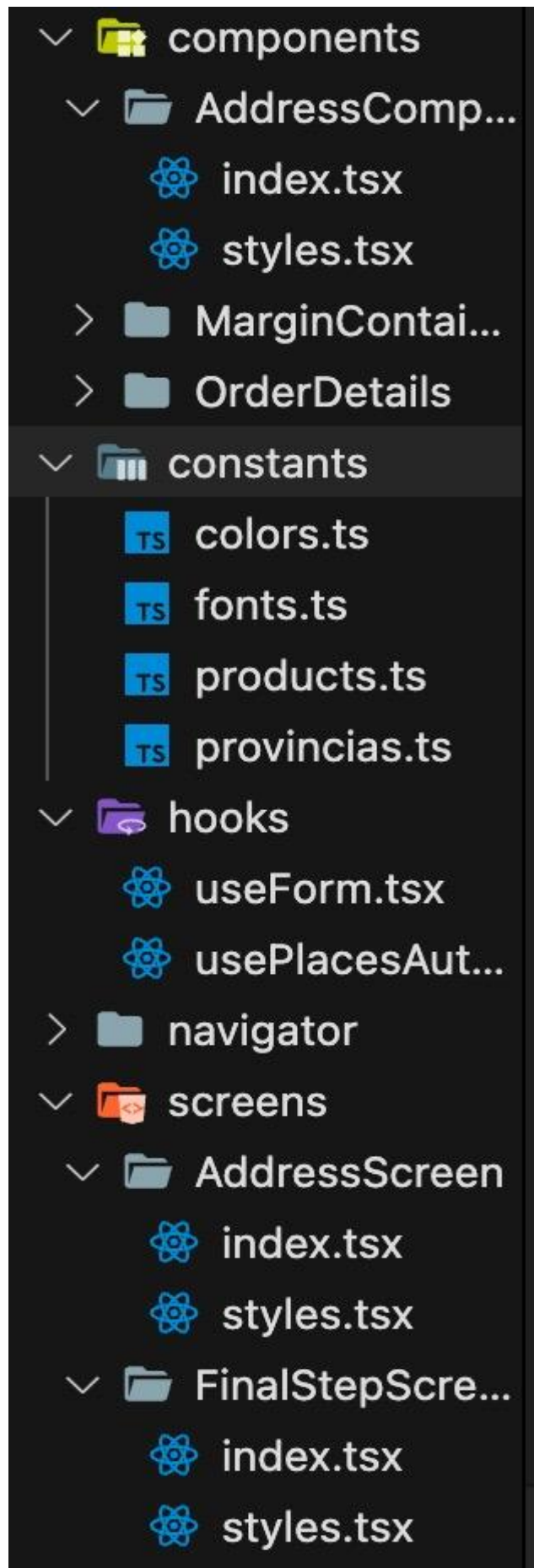
Todos los nombres de las constantes tendrán que inscribirse en camelCase comenzando con la primera palabra en minúscula.

```
“ejemplo”  
const paymentTypes = [“cash”, “visaCard”]
```

```
const orderPrice = products.reduce((acc, item) => {  
  return (acc += item.productPrice);  
}, 0);
```

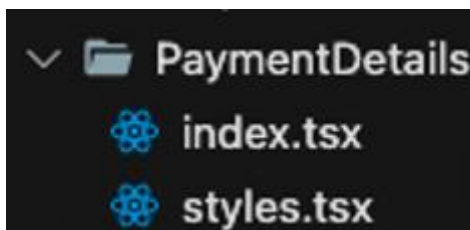
Estructura de archivos

Todos los archivos relacionados a un componente están en una misma carpeta, nombrada de manera entendible según su responsabilidad en el código, escritas en camelCase.



Componentes

- Extensión: Usamos la extensión **.tsx** para los componentes de React
- Nombre de directorios: Escritos en camelCase, comenzando con mayúsculas. Dentro de cada directorio de componente se encuentran dos archivos:
 - index.tsx en donde se encuentra la lógica del componente
 - style.tsx en donde se encuentra el estilo del componente
- Nombre de archivos: Usamos camelCase comenzando con minúscula para los mismos



Formatter

Utilizamos el formatter prettier, con las reglas default para el formateo de código.

Linter

Utilizamos ESLint como linter para encontrar errores y malas prácticas siguiendo una combinación de normas que incluyen el propio lenguaje, y el linter.

Declaración de clases

Los nombres de las clases deben ser sustantivos y deben tener la primera letra en mayúscula. Si el nombre es compuesto, cada palabra componente deberá comenzar en mayúscula.

Los nombres serán simples y descriptivos. Debe evitarse el uso de acrónimos o abreviaturas.

Las interfaces se nombran siguiendo los mismos criterios que los indicados para las clases. Como norma general toda interfaz se nombrará con el prefijo “I” para diferenciarla de la clase que la implementa

Métodos

Los métodos deben ser verbos escritos en minúsculas. Cuando el método está compuesto por varias palabras, cada una de ellas tendrá la primera letra en mayúscula, excepto la primera palabra.

GitFlow

