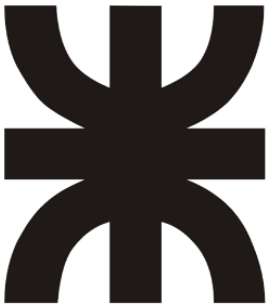
UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Córdoba



Ingeniera de Software

PRÁCTICO 6

Documento de estilo de código - Implementación de User Story

CURSO: 4K2 **TURNO:** Tarde

INTEGRANTES DEL GRUPO N° 2

Apellido y Nombres: Arriaga, Martin	**Legajo:** 81572
Apellido y Nombres: Cordero, Valentina	**Legajo:** 78802
Apellido y Nombres: Delavalle, Facundo	**Legajo:** 78274
Apellido y Nombres: Olivera, Nehuen	**Legajo:** 79470
Apellido y Nombres: Pacciarioni, Sergio Ariel	**Legajo:** 76344
Apellido y Nombres: Zapata, Mauricio	**Legajo:** 76034

Fecha de Entrega: 14-09-2021

Documento de estilos

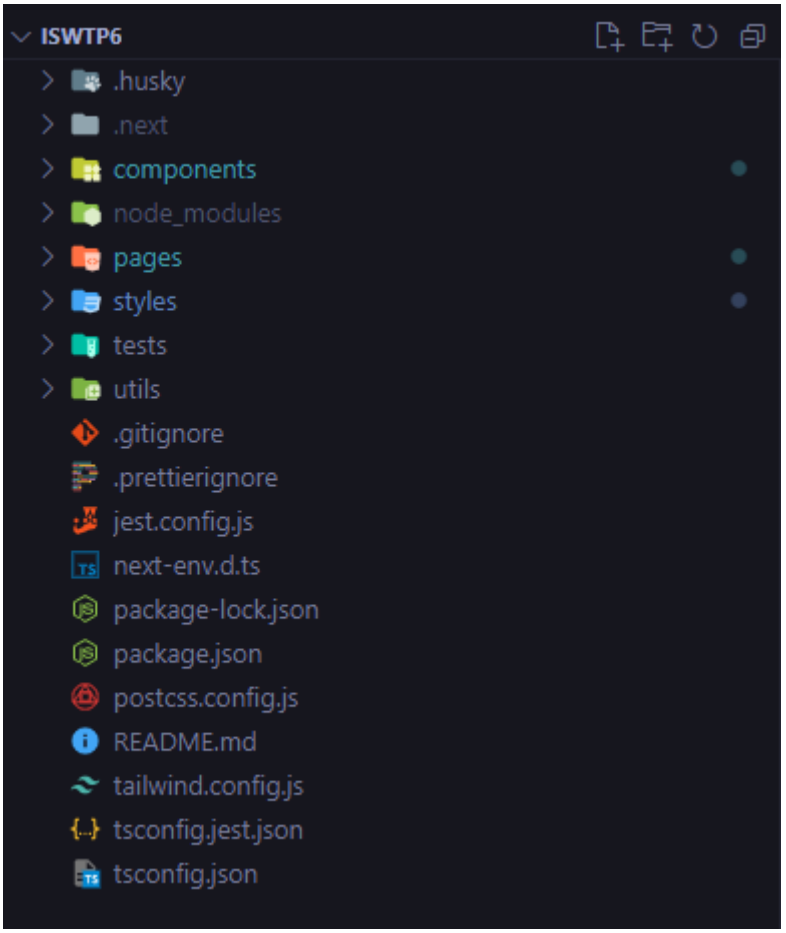
Antes de comenzar a leer este documento se recomienda ver una guía de estilos de JavaScript en:

https://standardjs.com/rules.html

donde se describe un conjunto de reglas y normas estandarizadas a seguir. Se usará React 17 para el desarrollo de la implementación de la User Story.

Estructura de archivos

Next y React nos proveen una forma de crear un nuevo proyecto con una estructura bastante cómoda y práctica:



Generalmente se define la estructura de acuerdo a la complejidad del proyecto. Entre más grande sea, más orden y modularidad requiere.

Una de las mayores ventajas que proporciona React es la flexibilidad lo cual nos brinda la posibilidad de adaptarse a diversas estructuras de proyecto dependiendo de su complejidad.

React maneja el concepto de componentes. Cada componente representa una pieza de UI (interfaz de usuario) en la aplicación web. Ese componente puede ser sin lógica o con lógica, depende el uso que se le quiera dar.

Tener una estructura bien definida nos ayudará a pensar en escalabilidad y permitirá ubicar fácilmente los distintos archivos a medida que el proyecto crezca. Como no hay una estructura bien definida a seguir en React, proponemos usar Screaming Architecture

Nombres claros

Uno de los mayores problemas a la hora de entender el código, suele ser el nombre que se le asigna a los métodos, variables o parámetros. Mientras más claro sea uno a la hora de definir los nombres mientras escribe el código, mejor será su entendimiento a futuro o para alguien que intente leerlo y comprenderlo posteriormente facilitando el mantenimiento del mismo.

Regla de los 5 segundos

Podemos aplicar la regla conocida como “5 segundos” a nuestro código. Si nos toma más de 5 segundos entender un bloque de código, es mejor considerar refactorizar.

La idea es que el código se entienda en el menor tiempo posible. Eso se puede lograr creando diferentes funciones pequeñas y haciendo composición en funciones más complejas. En caso de que el código falle, es más fácil encontrar el error y corregirlo sin afectar otras funciones.

Código organizado y legible

Algunas formas de tener un código más organizado y legible son:

- Lo más importante debe ir arriba.
- Primero propiedades, después métodos.
- Cada archivo debería contener solamente un componente, al igual que los servicios.
- Los componentes deben ser lo más atómicos posibles, de esta forma, son más fáciles de testear, más fáciles de refactorizar y se pueden reutilizar en diferentes partes de la aplicación.
- Las propiedades y métodos deberían escribirse en camelcase (ej: getPaymentMethod), en cambio, las clases deben usar uppercamelcase (ej: PaymentMethod).
- Las clases css deben escribirse en kebab-case.
- Los imports de archivos externos van primero, luego van los internos.

Proveer claridad

Es importante que a la hora de escribir código este sea lo más legible posible, es probable que alguien más lea el mismo en algún momento.

Por esto, el código debe ser autodescriptivo, es decir, no es necesario escribir comentarios si el código es tan claro que se explica solo y si además tanto las funciones como los métodos están bien nombrados.

Solo es necesario comentar cuando se trata de explicar por qué se hizo lo que se hizo o las consecuencias del código que se escribió.

Servicios

Es importante que a la hora de crear servicios se tenga en cuenta lo siguiente:

- Crear los servicios como Injectables para lograr un código menos acoplado.

- Es responsabilidad de los servicios recolectar la información necesaria, ya sea haciendo uso del backend o del localstorage. Los componentes nunca deben encargarse de resolver como traer información, estos sólo deberían encargarse de llamar al servicio que contiene todo lo necesario.

Seguir todas estas buenas prácticas a la hora de escribir código permite obtener un código mucho más mantenible y legible.

Complemento: Tailwind CSS

Para la implementación de la User Story se utilizará Tailwind CSS. Elegimos utilizar este framework debido a que es altamente personalizable, permitiendo la creación de diseños complejos y responsive de forma libre. Nos brinda la posibilidad de crear diseños personalizados sin tener que estar atados a utilizar componentes ya predefinidos.