Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX

Herlindo René Corona Arenales
Asesorado por Ing. Manuel Haroldo Castillo Reyna

Guatemala, octubre de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

HERLINDO RENÉ CORONA ARENALES
ASESORADO POR ING. MANUEL HAROLDO CASTILLO REYNA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton De León Bran
VOCAL IV	Br. Kevin Vladimir Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. Luis Fernando Espino Barrios
EXAMINADORA	Inga. Virginia Victoria Tala Ayerdi
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 30 de noviembre de 2021.

Herlindo René Corona Arenales

Guatemala, 09 de septiembre de 2022

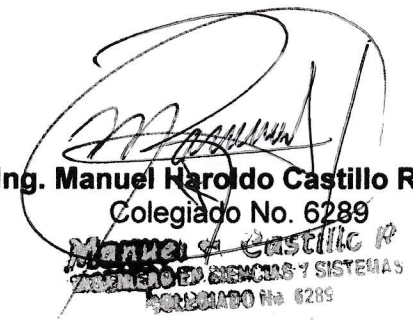
Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **Herlindo René Corona Arenales** con carné **201612219 y CUI 3114 32719 0409** titulado **"Manejo del estado de aplicaciones web desarrolladas con Angular usando NgRx",** luego de corroborar que el mismo se encuentra finalizado, lo he revisado y doy fé de que el mismo cumple con los objetivos propuestos en el respectivo protocolo, por consiguiente, procedo a la aprobación correspondiente.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Manuel Haroldo Castillo Reyna
Colegiado No. 6289

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 13 de septiembre de 2022

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **HERLINDO RENÉ CORONA ARENALES** con carné **201612219** y CUI **3114 32719 0409** titulado **"MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

LNG.DIRECTOR.206.EICCSS.2022

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX**, presentado por: **Herlindo René Corona Arenales**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

Msc. Ing. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, octubre de 2022

Decanato
Facultad de Ingeniería
24189101- 24189102
secretariadecanato@ingenieria.usac.edu.gt

LNG.DECANATO.OI.682.2022

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **MANEJO DEL ESTADO DE APLICACIONES WEB DESARROLLADAS CON ANGULAR USANDO NGRX,** presentado por: **Herlindo René Corona Arenales,** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANA
FACULTAD DE INGENIERÍA

Inga. Aurelia Anabela Cordova Estrada
Decana

Guatemala, noviembre de 2022

AACE/gaoc

ACTO QUE DEDICO A:

Dios	Por ser la fuente de inspiración y fuerza para alcanzar metas en mi vida.
Mis padres	Guicela Arenales y Carlos Corona; por su amor, confianza y apoyo incondicional.
Mis hermanos	Carlos Enrique y José Angel Corona; por los momentos que hemos pasado juntos y su apoyo incondicional.
Mis primos	Por el apoyo incondicional para alcanzar mis metas.
Mis amigos	Por apoyarme a lo largo de toda la carrera y por las experiencias vividas durante nuestra educación universitaria.

AGRADECIMIENTOS A:

Dios	Por darme la fuerza para alcanzar una nueva meta en la vida.
Universidad de San Carlos de Guatemala	Por las oportunidades de aprendizaje y por formarme como un profesional capaz.
Mi asesor	Ing. Manuel Castillo, por brindarme su apoyo y conocimiento para el desarrollo de este trabajo.
Mis amigos	Por el apoyo con sus conocimientos a lo largo de toda la carrera.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
npx	Ejecutor de paquetes de Node.js
V8	Entorno de ejecución de código JavaScript
CSS	Hojas de estilo en cascada
HTML	Lenguaje de marcado de hipertexto
DOM	Modelo de objetos de documento
ng	Módulo empleado para el uso de herramientas de Angular
HTTP	Protocolo de transferencia de hipertexto
%	Símbolo de porcentaje

GLOSARIO

Angular — *Framework* completo para el desarrollo de páginas web basado en el lenguaje TypeScript.

Backend — Parte de una solución de software orientada a la manipulación y transformación de los datos manipulados a través de una interfaz gráfica.

CSS — Lenguaje de estilos utilizado para definir la forma en que se presentan etiquetas HTML en una página web.

DOM — Estándar que permite la representación y manipulación de etiquetas HTML dentro de una página web.

Enrutamiento — Forma en la que la mayoría de las páginas web navegan entre distintas vistas de una misma aplicación.

Framework — Conjunto de herramientas estandarizadas bajo cierto concepto que dan solución a un problema o actividad recurrente.

Frontend — Parte de una solución de software orientada a la interacción directa con el usuario, generalmente se

	trata de páginas web o cualquier otro tipo de interfaz gráfica.
Google	Compañía dedicada principalmente al desarrollo de productos y servicios web.
HTML	Lenguaje de etiquetas utilizado para el desarrollo de páginas web.
Input	Decorador del *framework* Angular que permite compartir valores de un componente padre hacia un componente hijo.
JavaScript	Lenguaje de programación interpretado orientado al manejo dinámico de páginas web.
Librería	Conjunto de funciones que dan solución a uno o varios problemas recurrentes en el desarrollo de una solución.
Mobx	Biblioteca de JavaScript para el manejo del estado de aplicaciones web.
Node.js	Entorno de ejecución multiplataforma orientado a la ejecución de código JavaScript.
Output	Decorador del framework Angular que permite emitir eventos desde un componente hijo hacia un componente padre.

React	Biblioteca de JavaScript desarrollada para facilitar la creación de interfaces web.
Redux	Librería para JavaScript que ofrece la posibilidad de manejar el estado de aplicaciones.
Store	Almacenamiento en donde se encuentra el estado de una aplicación web basada en el patrón Redux.
TypeScript	Lenguaje de programación basado en JavaScript que ofrece la posibilidad del manejo de tipos y objetos en el lenguaje JavaScript.
V8	Motor de código abierto desarrollado en C++ orientado a la ejecución de código JavaScript fuera de un navegador.
Vue.js	Framework de JavaScript que permite el desarrollo de páginas web progresivas.

RESUMEN

Se presenta una propuesta para el manejo del estado de una aplicación web desarrollada con Angular y NgRx. El objetivo de la propuesta es definir una organización de componentes, interfaces, servicios y estado de una aplicación para reutilizar la mayor cantidad de código posible y reducir la complejidad en la comunicación entre los componentes de una aplicación web.

Se hace uso de Angular como *framework* de desarrollo web para esta propuesta debido a que es un framework que ofrece la mayor parte de herramientas necesarias para realizar el desarrollo de casi cualquier aplicación web, esto permite evitar la búsqueda de librerías externas que agreguen complejidad en la compatibilidad entre librerías existentes en la aplicación.

Para el manejo del estado de la aplicación web se hace uso de la librería NgRx. Está librería esta desarrollada específicamente para el *framework* Angular y es capaz de ofrecer una fácil compatibilidad con herramientas y librerías ya ofrecidas por Angular.

La reactividad de la aplicación se realiza empleando el estado de la aplicación generado con NgRx y de la librería RxJS garantizando que la aplicación se comporte de una forma asíncrona basándose en el flujo de datos obtenidos del estado de la aplicación.

OBJETIVOS

General

Implementar una alternativa que facilite en el control del estado de aplicaciones web haciendo uso del *framework* Angular y el framework NgRx.

Específicos

1. Demostrar que el uso del *framework* Angular es una de las mejores alternativas para el desarrollo de aplicaciones web por la cantidad de herramientas que ofrece.

2. Definir la conveniencia del uso del *framework* NgRx en el manejo del estado de una aplicación de Angular.

3. Establecer criterios de referencia sobre la utilidad de combinación entre el *framework* Angular y el *framework* NgRx.

4. Proveer una guía para el control del estado de una aplicación de Angular con el uso de NgRx.

INTRODUCCIÓN

Previo al crecimiento del uso de aplicaciones web, la mayoría de las aplicaciones de software eran entregadas a través de un instalador que instalaba las dependencias necesarias para asegurar el correcto funcionamiento una aplicación, actualmente con los avances en internet, la mayor parte de aplicaciones son desarrolladas en ambientes web, esto se debe a beneficios como el hacer uso de aplicaciones a través de un navegador web sin necesidad de una instalación previa en clientes que necesiten la aplicación.

Una de las problemáticas que surgieron junto con el desarrollo las aplicaciones web es la necesidad de hacer uso de servicios de internet, por lo que se volvió de vital importancia manejar correctamente el estado de una aplicación para que la experiencia del usuario final se sienta fluida y pueda comportarse de mejor manera dependiendo de factores como el estado de la aplicación o errores por parte de servicios externos.

En esta investigación se propone una alternativa para manejar el estado de una aplicación y reutilizar código haciendo uso de Angular y NgRx. Se iniciará repasando algunos conceptos claves relacionados con el desarrollo web y estudiando a grandes rasgos algunas de las herramientas más populares para desarrollar aplicaciones web y para manejar el estado de aplicaciones web.

Finalmente, se propone una estructura para organizar el código de una aplicación desarrollada con Angular, de tal manera que el código se pueda reutilizar fácilmente y que sea capaz de modificar su comportamiento dependiendo del estado de la aplicación.

1. MARCO TEÓRICO

1.1. Desarrollo Web

El desarrollo web es un conjunto de tareas que están relacionadas con el desarrollo de sitios web que son publicados en internet, generalmente el desarrollo web involucra tareas como el diseño de interfaces gráficas, manejo del estado de aplicaciones, seguridad en aplicaciones y comunicación con servidores. Debido a que el desarrollo web involucra muchas tareas, podemos subdividir el desarrollo web en dos áreas: desarrollo frontend y desarrollo backend.

El desarrollo frontend es un tipo de desarrollo que está específicamente relacionado con el desarrollo de interfaces gráficas, en este tipo de desarrollo se lleva a cabo todo lo relacionado con el aspecto visual de un sitio, durante este desarrollo se diseña y crea aspectos como los colores, tipografía, navegación, animaciones y tecnologías a utilizar.

Al contrario del desarrollo frontend el desarrollo backend está dedicado a manejar toda la lógica que ocurre detrás de una página web, durante este desarrollo se validan aspectos como la autenticación, conexiones a bases de datos, comunicación con servicios externos, manipulación de datos y entrega de información.

Actualmente gran parte del desarrollo web está relacionado con frontend, este desarrollo es realizado haciendo uso de HTML, JavaScript y TypeScript. Por el otro lado, el desarrollo backend es mucho más flexible y regularmente está

basado en servicios o microservicios que permiten emplear una gran variedad de lenguajes de programación siempre y cuando los lenguajes permitan comunicarse a través de cualquier medio.

1.1.1. HTML

HTML cuyas iniciales significan HyperText Markup Language es un lenguaje de etiquetas utilizado para el desarrollo de páginas web, HTML funciona a través de un conjunto de elementos que permiten encapsular múltiples elementos para que tengan una apariencia específica. Un elemento de HTML está compuesto por 3 partes: etiqueta de apertura, etiqueta de cierre y contenido.

La etiqueta de apertura es usada para indicar el inicio de un nuevo elemento. La etiqueta de apertura sirve para indicar el tipo de elemento que se creara en la vista y el tipo de contenido que se esperaría que este elemento tenga en dentro de las etiquetas de apertura y cierre.

El contenido de un elemento es lo que encontramos dentro de la etiqueta de apertura y cierre de un elemento, este contenido puede variar dependiendo del tipo de elemento que se está empleando, el contenido dentro de una etiqueta puede ser texto, un conjunto de más elementos o contenido vacío.

Finalmente, la última parte de un elemento en HTML es la etiqueta de cierre, esta etiqueta indica que el elemento ha finalizado y que a partir de ese punto el resto de los elementos ya no serán parte del contenido del elemento generado.

1.1.2. CSS

CSS, cuyas iniciales significan Cascading Style Sheets, son hojas de estilos que nos permiten especificar la presentación de los elementos en una página web. Con CSS podemos definir el tamaño, margen interno, margen externo, color y bordes de un elemento HTML.

La razón por la que se utiliza CSS en el desarrollo de páginas web es porque utilizar HTML puro produce páginas webs poco agradables visualmente, por lo que es casi seguro que a la mayoría de las páginas web se apliquen estilos CSS para generar páginas web mucho más agradables a la vista.

A diferencia de lenguajes de programación y de etiquetas, CSS funciona a través de reglas que son aplicadas a los elementos de HTML. CSS nos permite definir reglas para estilizar elementos haciendo uso de selectores y clases. Un selector selecciona todos los elementos que poseen el mismo tipo del selector creado y aplica el conjunto de reglas a todos los elementos. Una clase permite generar un conjunto de reglas y aplicarlas a elementos específicos del HTML.

1.1.3. JavaScript

JavaScript es un lenguaje de programación para navegadores creado para mejorar la interactividad del contenido HTML y CSS. JavaScript permite desarrollar funcionalidades complejas como animaciones, control en contenido multimedia y presentación de información de forma dinámica.

Al igual que CSS, es muy difícil encontrar páginas web desarrolladas con HTML y CSS puro, en la mayoría de las ocasiones el código HTML y CSS es

desarrollado junto a código JavaScript relacionado con el comportamiento de una página web.

El funcionamiento de JavaScript se centra en la manipulación del Document Object Model de una página web, JavaScript nos permite declarar variables y funciones que nos facilitan modificar el comportamiento de una página web. Una página web es modificada a través de respuestas a eventos disparados desde elementos en el HTML.

Actualmente, JavaScript ha pasado de ser un lenguaje que permite manipular de forma dinámica el comportamiento de una aplicación web a ser un lenguaje de programación que permite desarrollar servicios backend gracias a entornos de ejecución como Node.js.

1.1.4. TypeScript

TypeScript es un lenguaje de programación de código abierto basado en el lenguaje JavaScript, este lenguaje que agrega tipos de datos al lenguaje JavaScript, esta característica es de utilidad para evitar errores que inconscientemente se crean durante el desarrollo de programas.

Actualmente, Node.js ha convertido a JavaScript en un lenguaje referente para el desarrollo de aplicaciones backend. JavaScript ofrece muchas ventajas gracias a su flexibilidad en el desarrollo de soluciones de software, si bien estas ventajas regularmente facilitan el desarrollo de software, es importante tener un patrón de desarrollo claro para que estas ventajas no se conviertan en desventajas y generen código de software desordenado y difícil de mantener.

Debido a las ventajas y desventajas que ofrece JavaScript y Node.js es importante tomar en cuenta el lenguaje TypeScript, este lenguaje ofrece las mejores ventajas de dos mundos, el desarrollo orientado a objetos y la flexibilidad del lenguaje JavaScript.

Se podría decir que TypeScript es un lenguaje que logro cerrar la brecha entre el desarrollo de aplicaciones exitosas orientadas a objetos y aplicaciones flexibles desarrolladas con JavaScript, esta brecha fue cerrada gracias a la posibilidad de establecer tipos dentro del lenguaje de programación JavaScript.

1.1.5. Node.js

Node.js es un entorno de ejecución desarrollado para el lenguaje JavaScript creado por los desarrolladores del lenguaje JavaScript. Node.js se caracteriza por ser capaz de ejecutar cualquier programa escrito en el lenguaje JavaScript sin necesidad de un navegador.

Node.js utiliza el motor de JavaScript V8 desarrollado por Google para lograr la ejecución del código escrito en JavaScript. A grandes rasgos, Node.js hace uso de eventos para lograr la ejecución del código JavaScript, este entorno de ejecución se basa en una pila de hilos que se encarga de distribuir los recursos para asegurarse que una tarea que consume muchos recursos no interrumpa al resto de tareas.

Actualmente, existe una gran cantidad de librerías desarrolladas en JavaScript para uso en Node.js, estas librerías van desde pequeñas funcionalidades que resuelven problemas recurrentes hasta *frameworks* de desarrollo completos que facilitan la creación de aplicaciones en Node.js.

1.2. Aplicaciones reactivas

Las aplicaciones reactivas son aplicaciones desarrolladas bajo el paradigma reactivo, este paradigma está enfocado en el manejo de flujo de datos finitos o infinitos de una forma asíncrona. Este tipo de paradigmas ha sido utilizado principalmente por aplicaciones que manejan grandes cantidades de contenido multimedia. La razón por la que estas aplicaciones han utilizado este paradigma es porque permiten modificar el comportamiento de una aplicación dependiendo del estado en el que se encuentra la aplicación, un ejemplo es la advertencia de errores al ocurrir errores durante una petición a un servicio externo.

1.2.1. Principios de la programación reactiva

Los principios de la programación reactiva fueron desarrollados en 2014 por Jonas Bonér, Dave Farley, Roland Kuhn, Martin Thompson y miembros de la comunidad de desarrollo, estos principios fueron plasmados en el Manifiesto Reactivo, el cual es una guía con conceptos clave acerca de la programación reactiva.

1.2.1.1. Responsividad

Este principio asegura que un sistema sea capaz de responder de la forma más rápida posible, asegurándose de que al ocurrir un error este sea tratado correctamente y que la respuesta que brinde en la aplicación sea confiable, cumpla con la calidad y tiempo establecido con el servicio que se está ofreciendo.

1.2.1.2. Resiliencia

La resiliencia asegura que una aplicación permanezca responsiva aun cuando ocurre errores, esto se logra asegurándose de que se posea replicación, contención y aislamiento de componentes de una aplicación, lo que permite continuar respondiendo al usuario aun cuando un componente de nuestra aplicación está fallando.

1.2.1.3. Elasticidad

La elasticidad permite que una aplicación permanezca responsiva aun cuando la carga de trabajo de la aplicación varia sin importar si es mucho trabajo o poco trabajo, las aplicaciones deben ser capaces de reaccionar y administrar los recursos dependiendo del estado de una aplicación.

1.2.1.4. Basado en mensajes

Las aplicaciones reactivas deben basarse en mensajes asíncronos para poder garantizar un acoplamiento flexible entre los componentes de una aplicación, también permiten enviar mensajes cuando ocurre un error y se sabe cómo se reaccionará ante dicho error.

1.3. Patrón Redux

El patrón Redux es un patrón de desarrollo que permite manejar el estado de una aplicación de una forma predecible, este patrón inicio con la idea de crear una librería sobre el motor JavaScript capaz de reducir el número de relaciones entre los componentes de una aplicación y de esta manera manejar un flujo de datos de una forma sencilla, rápida y predecible.

Debido a los grandes beneficios de este patrón, el patrón Redux ha sido adoptado como base por varios f*rameworks* de desarrollo web como React, Vue.js e incluso utilizado como base para *frameworks* dedicados al manejo del estado de una aplicación como Redux, NgRx y MobX.

Este Patrón funciona bajo 3 grandes principios, el primer principio es el Store el cual es la fuente principal en donde se almacena el estado actual de la aplicación, las acciones encargadas de informar que el estado debe ser modificado y reductores que se encargan de cambiar el estado de una aplicación.

1.3.1. Store

El concepto de Store dentro de Redux se refiere a un objeto que contiene todo el estado de una aplicación, por lo que, si un componente necesita información sobre el estado actual de una aplicación, debe consultar el objeto del Store para comportarse correctamente basado en el estado de la aplicación. Un Store se caracteriza por ser una fuente de datos de una única vía, por lo que si es necesario modificar sus datos deben hacerse a través de acciones disparadas desde componentes que se encargan de realizar cambios en el Store a través de reductores.

1.3.2. Acciones

Las acciones dentro de Redux representan eventos disparados desde componentes y servicios, estas acciones indican que el estado de la aplicación debe de ser mutado, un ejemplo claro es un error ocurrido durante una petición a un servicio externo, este cambio debería ser capaz de disparar una acción que modifique el estado de una aplicación y que el resto de los componentes respondan correctamente basados en el cambio del estado de la aplicación.

1.3.3. Reducciones

Una reducción en Redux es una función pura capaz de cambiar el estado de una aplicación, esta función siempre se basará en el estado anterior de la aplicación y en las propiedades de la acción disparada que definen cuál es el cambio debe realizar dicha mutación.

1.4. *Frameworks* de desarrollo web

Un *framework* de desarrollo web es un conjunto de herramientas y recursos que permiten a los desarrolladores programar de una forma mucho más rápida, sencilla y ordenada. La mayoría de frameworks de desarrollo web se caracterizan por brindar funcionalidades específicas que permiten reutilizar y aplicar buenas prácticas de desarrollo de software dependiendo del patrón de desarrollo que estamos aplicando en nuestros proyectos.

Algunos de los *frameworks* de desarrollo web más utilizados son:

- Angular
- React
- Vue.js
- Ionic
- Flutter

1.5. *Frameworks* para el manejo de estado de aplicaciones

Existen muchos *frameworks* y librerías para el manejo del estado de una aplicación. Estas librerías generalmente están desarrolladas para ser aplicadas en conjunto con un framework de desarrollo web específico, aunque existen

librerías que son aplicables en cualquier tipo de proyecto o framework de desarrollo web.

Entre las más populares podemos encontrar Redux y MobX que son compatibles con una gran variedad de frameworks de desarrollo web y a su vez podemos encontrar alternativas como NgRx que está desarrollado específicamente para funcionar junto al *framework* de desarrollo Angular. Entre los *frameworks* para el manejo de estado de aplicaciones más populares encontramos los siguientes:

- Redux
- NgRx
- MobX
- NgXs
- Vuex

2. ANGULAR

Angular es un *framework* de desarrollo web mantenido por Google y la comunidad de desarrollo de software libre. Angular está construido sobre el lenguaje TypeScript y tiene como objetivo facilitar, ordenar y acelerar el desarrollo de aplicaciones web de una sola página. Angular logra todos estos beneficios a través de un conjunto de módulos que permiten facilitar el enrutamiento, administración de formularios, manipulación del comportamiento de páginas web, manejo de peticiones web y reutilización de código escrito.

Uno de los objetivos principales de Angular es maximizar la cantidad de desarrollo efectivo con el mínimo esfuerzo posible, por esta razón los proyectos de Angular están organizados de tal manera que el desarrollador pueda identificar de una forma sencilla la dependencia que existe entre las directivas y los archivos fuentes que se desarrollan para cada uno de los bloques de código que dan vida a un proyecto desarrollado en Angular.

2.1. Módulos

Un módulo dentro de Angular es la parte encargada de definir el espacio de trabajo para cierta funcionalidad dentro de una aplicación. Los módulos se caracterizan por definir todos los componentes a utilizar en la sección de trabajo, importar los módulos necesarios para el correcto funcionamiento de los componentes declarados y la exportación de funcionalidades que pueden ser reutilizables en otros módulos.

Los módulos en Angular tienen una gran importancia, ya que permiten la importación de módulos dentro de otros módulos, esto permite que el módulo que está importando sea capaz de emplear componentes, directivas y cualquier otra funcionalidad que ha sido previamente exportada en el módulo importado. La importación de módulos permite reutilizar funcionalidades basadas en las necesidades de un módulo, lo que permite tener claro que se está usando en el área de trabajo del módulo.

2.2. Directivas

Las directivas en Angular son clases que agregan un comportamiento adicional a los bloques de código que se desarrollan en Angular, al hacer uso de estas directivas podemos asegurarnos de que la reutilización de código de un proyecto se alcance de una mejor manera basado en las necesidades del proyecto.

Angular ofrece varios tipos de directivas, una de las más utilizadas es la directiva de componentes que permite reutilizar bloques de código de una aplicación, otro tipo de directivas muy utilizada son las directivas para atributos que permiten cambiar la apariencia o comportamiento de otras directivas y finalmente Angular ofrece las directivas estructurales que permiten cambiar el comportamiento de los elementos en archivos HTML.

2.3. Decoradores

Angular ofrece los decoradores como una opción a incluir en un patrón de desarrollo, los decoradores permiten mejorar la reutilización de componentes permitiendo ampliar o modificar una clase preexistente previo a ser utilizada, de

esta manera se evita la necesidad de duplicar la misma lógica de desarrollo en todos los componentes en donde se utilice la clase decorada.

2.4. Componentes

Un componente es un bloque de desarrollo web que está compuesto por un archivo TypeScript que define el comportamiento de nuestro bloque de desarrollo, una plantilla HTML que nos indica como será la interfaz gráfica de nuestro bloque y opcionalmente de una hoja de estilos CSS en caso de ser deseada.

Un componente se encarga de ofrecer la capacidad de poder construir aplicaciones web complejas a través de pequeños bloques de desarrollo que simplifiquen la construcción de páginas web, Angular ofrece la posibilidad de reutilizar dichos componentes a través de selectores que son definidos en el archivo TypeScript de cada componente, de esta manera en caso de tener un bloque de desarrollo repetitivo en múltiples páginas de nuestra aplicación podemos reutilizar este código sin necesidad de reescribir archivos HTML, TypeScript y CSS.

2.5. Inyección de dependencias

Otra herramienta que ofrece Angular es la inyección de dependencias dentro del código TypeScript sin necesidad de crear instancias en el código fuente, Angular ofrece la alternativa de inyectar dependencias en el constructor, permitiendo utilizar estas dependencias en cualquier parte de nuestro código TypeScript.

La inyección de dependencias es de utilidad porque permite inyectar servicios externos que realizan peticiones HTTP e incrustar cadenas de texto dentro de nuestro código HTML. Estas inyecciones son útiles cuando se necesita incrustar direcciones web variables que dependen del servicio externo que se utiliza en el proyecto para el manejo de contenido multimedia, ya que permiten inyectar una misma variable en múltiples componentes que provienen desde un mismo archivo fuente.

2.6. Input

Input es un decorador que permite definir un campo de entrada dentro en una clase con el objetivo maximizar la reutilización de componentes y de modificar el comportamiento de nuestras clases, un ejemplo podría ser ocultar contenido basado en el componente padre, con este decorador se puede crear un campo de entrada que permita saber si el contenido debe ocultarse sin necesidad de crear un componente adicional que no muestre ese contenido.

2.7. Output

Output es la contraparte del decorador Input, se diferencia del decorador Input porque ofrece la capacidad de poder pasar un flujo de datos desde el componente hijo hacia un componente padre, esto es útil para reutilizar componentes que son iguales en su aspecto pero que generan un resultado final distinto, un ejemplo puede ser un formulario reutilizado durante la creación o edición de un elemento, si bien el formulario visualmente es el mismo, la acción final que se lleva a cabo es distinta.

2.8. Interceptores

Tal como su nombre lo indica, los interceptores dentro de Angular permiten realizar intercepciones de peticiones HTTP para que estas sean manipuladas previo o posterior a ser procesadas, esta funcionalidad nos permite incrustar encabezados dentro de las solicitudes previo a ser enviadas lo que maximiza el tiempo de desarrollo al evitar agregar estos encabezados en cada una de las peticiones a realizar.

Por otra parte, los interceptores permiten manipular respuestas a solicitudes HTTP, un ejemplo es el vencimiento de un token de acceso a cierto servicio, por lo que a través de este interceptor podemos refrescar el token de acceso y realizar nuevamente una petición sin necesidad de realizar una implementación repetitiva en todas las solicitudes HTTP a realizar.

2.9. Guard

Si bien el desarrollo aplicaciones web se suele tener una aplicación frontend distinta para cada tipo de rol dentro de una aplicación, existen ocasiones en donde se hace uso de la misma aplicación, aunque existan múltiples roles dentro de la aplicación.

Angular permite el manejo de acceso a pantallas basándose no solo en roles, sino que en cualquier tipo de condiciones que denieguen el acceso a pantallas. Angular permite desarrollar esta funcionalidad a través de Guards en donde se define la lógica de acceso a una pantalla, esto permite no solo redireccionar a un inicio de sesión cuando el usuario no se encuentra autenticado, sino que permite implementar funcionalidades como modales de advertencia al intentar cambiar de pantalla cuando un formulario no ha sido guardado.

3. NGRX

NgRx es un framework para el manejo del estado de aplicaciones desarrolladas con Angular, este framework está inspirado en Redux y ofrece compatibilidad con RxJS lo que permite manipular el estado de una aplicación de una forma mucho más flexible dentro de componentes de Angular.

Muchos de los conceptos usados en NgRx están inspirados en Redux por lo que la mayoría de estos son similares entre sí. NgRx está basado en los conceptos de store, acciones, funciones puras y reductores, la principal variación de NgRx respecto a Redux está en que NgRx combina lo mejor de Redux, RxJS y ofrece herramientas como ngrx/store-devtools para facilitar las tareas de depuración y pruebas de aplicaciones web.

3.1. Store

El Store dentro de NgRx es un contenedor en el cual se almacena toda la información relacionada con el estado actual de la aplicación y acciones realizadas a lo largo de la vida de una aplicación, este contendor está creado con una alta compatibilidad con RxJS para facilitar la transformación de la información del estado de una aplicación haciendo uso de observables.

3.2. Reductores

Los reductores son funciones puras que mutan el estado actual dentro del Store de NgRx, estas reducciones son realizadas con base en acciones disparadas por NgRx, las reducciones pueden o no tener propiedades

dependiendo de su naturaleza. En caso de ser una acción que involucra una interacción con formularios, lo más probable es que la acción incluya propiedades con el valor del formulario para mutar el valor del estado, aunque no siempre es necesaria una propiedad para mutar el estado.

3.3. Acciones

Son eventos únicos que desencadenan cambios dentro del Store de NgRx, generalmente se crean acciones para cada uno de los eventos que ocurren durante la interacción del usuario con una página web, aunque también suelen crearse acciones para tareas repetitivas como el bloqueo temporal de una página web, presentación de páginas por defecto o la llamada a servicios externos.

3.4. Selectores

Un selector es un objeto creado con el objetivo de seleccionar únicamente una porción del estado actual de una aplicación. Si no se utiliza selectores durante el desarrollo de una aplicación web, cada vez que ocurre un cambio toda la información será seleccionada y emitida a los componentes, lo que implica un gasto de recursos innecesario.

3.5. Store Devtools

Store Devtools es un conjunto de herramientas de desarrollo creadas para NgRx. Estas herramientas permiten desarrollar de una manera sencilla al permitir ver el estado actual, previos, acciones disparadas e incluso modificar manualmente el valor del estado de la aplicación, lo que permite realizar pruebas de una forma mucho más sencilla y amigable para un desarrollador.

4. ¿POR QUÉ USAR ANGULAR Y NGRX PARA EL DESARROLLO DE APLICACIONES WEB?

Con el rápido crecimiento en el uso y desarrollo de aplicaciones web, es importante entender por qué Angular y NgRx son una de las mejores alternativas para realizar el desarrollo de una aplicación web. En este capítulo se encuentra estadísticas y comparaciones de frameworks de desarrollo web y librerías para el manejo del estado de aplicaciones *frontend.* Las comparaciones que se encuentran en este capítulo se basan en aspectos claves que deben ser considerados para maximizar la cantidad de desarrollo al crear una aplicación *frontend.*

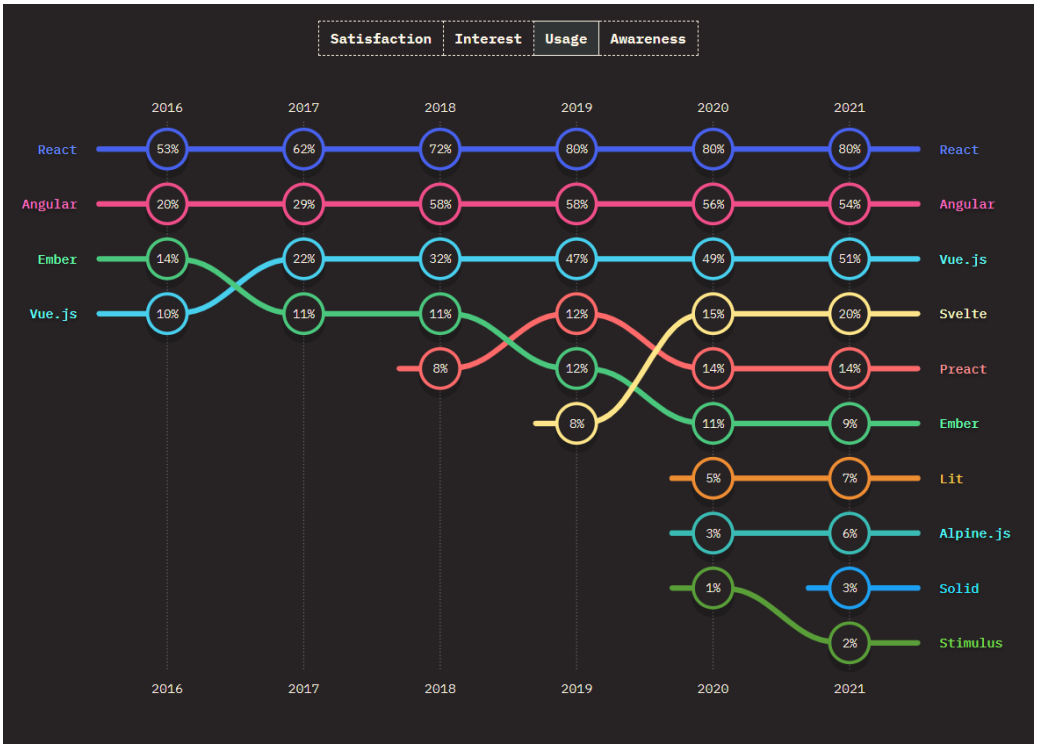
4.1. *Frameworks* de desarrollo *frontend*

En esta sección se encuentra un análisis de los frameworks de desarrollo *frontend* más utilizados por la comunidad de desarrollo. El análisis está basado en los 3 frameworks de desarrollo *frontend* más usados, la cantidad de documentación disponible, cantidad de consultas promedio realizadas sobre el *framework*, orientación del *framework* y herramientas ofrecidas por el *framework.*

4.1.1. *Frameworks* más utilizados

De acuerdo con State of JS, proyecto dedicado a explorar tendencias en herramientas utilizadas en el desarrollo web, los tres frameworks de desarrollo frontend más utilizados por la comunidad de desarrollo son React, Angular y Vue.js, seguidos en un menor porcentaje por Svelte, Preact y Ember.

Figura 1. **Tendencia de uso de frameworks de desarrollo frontend más utilizados**



Fuente: State of JS (2021). FRONT-END FRAMEWORKS. Consultado el 27 de junio de 2022. Recuperado de https://2021.stateofjs.com/en-US/libraries/front-end-frameworks.

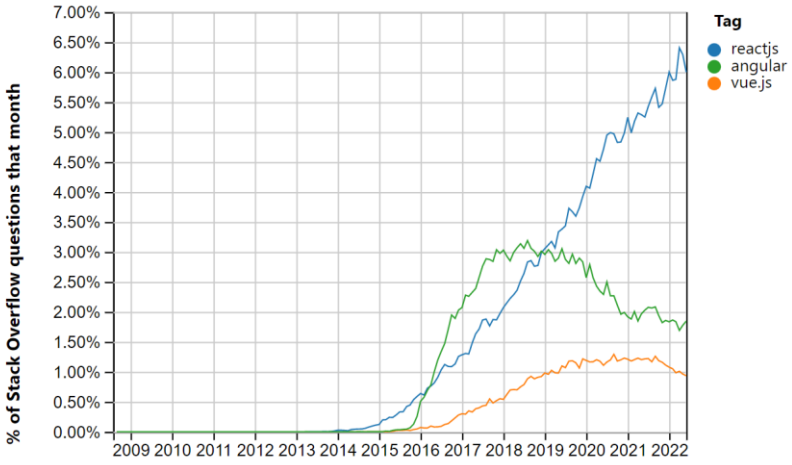
El uso de React, Angular y Vue.js ha cambiado ligeramente en más de los últimos seis años, pero siempre manteniéndose como los frameworks de desarrollo frontend más utilizados por la comunidad de desarrollo, esto asegura que son frameworks de desarrollo robustos, confiables y con suficiente documentación a consultar durante el desarrollo de una aplicación web.

4.1.2. Documentación y consultas en línea

Tanto para React, Angular y Vue.js existe documentación oficial del framework de desarrollo en las páginas oficiales correspondientes para cada framework de desarrollo. Si bien esto suele ser suficiente para desarrollar una aplicación web, es importante saber si los frameworks siguen siendo consultados y poseen información acerca de problemas que pueden surgir a lo largo del desarrollo de la aplicación.

Una de las páginas web con mayor cantidad de preguntas y respuestas a problemas relacionados con cualquier tipo de desarrollo de software es Stack Overflow, razón por la que esta página recopila estadísticas en tendencias en preguntas realizadas en esta plataforma desde 2008.

Figura 2. **Tendencia de preguntas realizadas en Stack Overflow acerca de React, Angular y Vue.js**



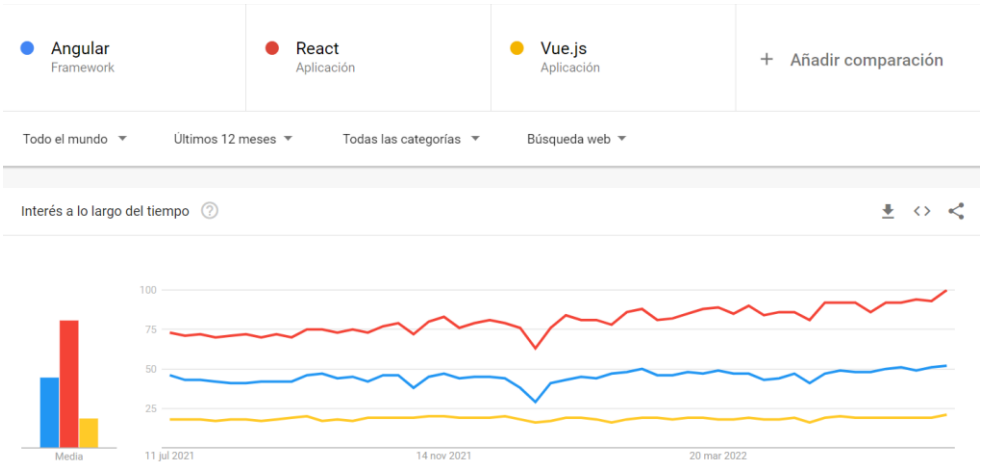
Fuente: Stack Overflow Trends (2022). Stack Overflow Trends. Consultado el 27 de junio de 2022. Recuperado de https://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cvue.js.

De acuerdo con estadísticas de Stack Overflow, los temas React, Angular y Vue.js continúan siendo consultados en la plataforma desde 2014 hasta la actualidad, por lo que adicional a la documentación es muy probable encontrar soluciones a problemáticas recurrentes para cada uno de los *frameworks* de desarrollo.

Es importante tomar en cuenta que, si bien en la actualidad cerca del 6 % de las preguntas mensuales de Stack Overflow están relacionadas con React, no quiere decir que exista más información respecto a React que sobre de Angular. Este comportamiento se debe a que React ha realizado cambios recientes en el manejo de herramientas como por ejemplo el uso de hooks, cambios que, por el contrario, Angular realizo en mayor cantidad entre los años 2016 y 2020 con implementaciones como el compilador Ivy para la mejora del rendimiento en estas aplicaciones.

Si bien Stack Overflow es una de las plataformas más importantes para la consulta de dudas relacionadas con el desarrollo de software, no es la única fuente de consultas del framework por lo que también es importante analizar la cantidad de búsquedas que se realizan relacionadas con estos *frameworks* de desarrollo web.

Figura 3. **Tendencia de búsquedas realizadas en Google acerca de Angular, React y Vue.js**



Fuente: Google Trends (2022). Google Trends. Consultado el 27 de junio de 2022. Recuperado de https://trends.google.com/trends/explore?q=%2Fg%2F11c6w0ddw9,%2Fm%2F012l1vxv,%2Fg%2F11c0vmgx5d.

De acuerdo con Google Trends tanto React, Angular y Vue.js son temas buscados constantemente en línea diariamente, variando ligeramente por 25 búsquedas entre cada uno de los frameworks. Esta estadística nos permite deducir que estos frameworks continúan teniendo una tendencia de búsqueda similar a nivel global no solo en plataformas como Stack Overflow, sino que también en páginas web dedicadas al desarrollo de software lo que da un valor agregado adicional a cualquiera de estos *frameworks.*

4.1.3. Orientación de *frameworks*

Un aspecto muy importante para tomar en cuenta es la orientación que ofrece cada uno de los frameworks de desarrollo frontend, ya que, si bien la mayoría de frameworks hacen uso del concepto de uso de componentes para reutilizar funcionalidades desarrolladas, la orientación puede definir si el framework se adapta o no a nuestras necesidades.

4.1.3.1. React

Tal como la mayoría de frameworks React trabaja bajo el concepto de componentes reutilizables que juntos forman un solo componente que es mostrado en la aplicación. React tiene una orientación estrictamente minimalista, este framework está orientado al desarrollo de interfaces de usuario y al manejo del estado de dichas interfaces de usuario.

Debido a que React tiene una orientación minimalista, es necesario tener en cuenta que todas las funcionalidades necesarias para el desarrollo de una aplicación deben ser desarrolladas desde cero o debe considerarse el uso de librerías de terceros para aumentar la cantidad de desarrollo a realizar en el menor tiempo posible.

4.1.3.2. Angular

Angular es un *framework* que también trabaja bajo el concepto de componentes, pero que a diferencia de la mayoría de frameworks tiene una orientación completamente distinta a la encontrada en la mayoría frameworks. Angular está orientado a ser un framework todo en uno capaz no solo de facilitar la construcción de interfaces de usuario, sino que también capaz de ofrecer la

mayor parte de herramientas necesarias para construir casi cualquier aplicación web.

Gracias a esta orientación, el desarrollo de Angular es maximizado al evitar la búsqueda de librerías externas compatibles con la aplicación, lo que asegura que las herramientas existentes tienen una gran compatibilidad y que adicionalmente han sido aprobadas por el grupo de desarrollo de Angular.

Por el otro lado, el hecho de que Angular sea un *framework* todo en uno no quiere decir que sea ideal para cualquier tipo de aplicaciones, ya que estas herramientas poseen limitaciones y deben seguir un patrón específico para que funcionen correctamente.

Si bien en pocas ocasiones las herramientas que ofrece Angular no cumplen con todos los requisitos para desarrollar alguna funcionalidad, Angular permite modificar estas herramientas a través de decoradores que mejoran el funcionamiento de las herramientas existentes.

4.1.3.3. Vue.js

Como es de esperarse, Vue.js es un framework que también funciona bajo el concepto de uso de componentes para realizar el desarrollo de aplicaciones, pero que al igual que la mayoría de frameworks posee una orientación única que adapta el framework a varios tipos de soluciones deseadas. La orientación de Vue.js está relacionada con el desarrollo de aplicaciones progresivas que cambian constantemente a lo largo del desarrollo de la aplicación.

Gracias a la orientación progresiva de Vue.js podría decirse que Vue.js posee una orientación intermedia entre la orientación ofrecida por Angular y

React, tratando de mantener Vue.js lo más minimalista como sea posible y que deje a discreción del desarrollador la mayor parte de funcionalidades, pero que a su vez ofrezca herramientas cuya necesidad es recurrente.

4.1.4. Herramientas por considerar en la selección de framework de desarrollo frontend

Generalmente, la mayoría de las aplicaciones hacen uso de varias herramientas comunes para llevar a cabo el desarrollo final de una aplicación web, ejemplos de estas herramientas son la manipulación de la interfaz de usuario, manejo de estados, enrutamiento, manejo de formularios y peticiones HTTP a servicios externos.

4.1.4.1. Manipulación de interfaz de usuario

Angular, React y Vue.js ofrecen una manipulación de la interfaz de usuario completa, esta manipulación puede realizarse de forma automática a través del estado de la aplicación o de forma directa a través de herramientas desarrolladas para cada uno de los componentes para manipular de forma directa los elementos de una interfaz de usuario.

Para el caso de Angular, la manipulación de un elemento puede ser realizado a través de la herramienta ViewChild, en React a través de la herramienta createRef y en Vue.js a través de referencias en los elementos de la interfaz de usuario.

4.1.4.2. Manejo de estado

El manejo del estado de una aplicación puede visualizarse a tres grandes niveles: estado de componente, estado de varios componentes y estado de una aplicación, estos estados son manejados de formas distintas por Angular, React y Vue.js.

4.1.4.3. Estado de componente

El estado de variables directas de un componente es manejado de forma automática al hacer uso de Angular, basta con modificar el valor de dicha variable para que los cambios sean reflejados en la interfaz de usuario. En React y Vue el estado de variables de un componente deben ser manejados de forma manual a través de herramientas como useState y referencias para definir cuando el valor del estado de una variable de un componente debe de ser modificada y actualizada en la interfaz de usuario.

4.1.4.4. Estado para múltiples componentes

El manejo del estado entre múltiples componentes es manejado de la misma manera en Angular, React y Vue.js, este estado es manejado a través de la comunicación de componentes en donde se envía el estado desde un componente padre hacia un componente hijo y viceversa. Cada uno de estos frameworks cuentan con herramientas que permiten realizar la comunicación de estos valores.

4.1.4.5. Estado general de una aplicación

El manejo del estado de una aplicación completa se maneja de la misma manera en Angular, React y Vue.js, es necesario de una librería de terceros para el manejo de este tipo de estado, ejemplos de estas librerías pueden ser Redux, MobX y NgRx. Si bien estos frameworks casi siempre recurren a librerías externas, React es el único framework que ofrece la herramienta useContext para manejar el estado general de una aplicación.

4.1.4.6. Enrutamiento

El enrutamiento de distintas páginas web en una sola aplicación es una de las funcionalidades más comunes en una aplicación web, por lo que esta herramienta suele incluirse entre las herramientas ofrecidas por un framework de desarrollo frontend, en este caso herramientas de enrutamiento son incluidas tanto en el framework de Angular como de Vue.js.

4.1.4.7. Manejo de formularios

El manejo de formularios es una herramienta que está incluida únicamente en el *framework* de desarrollo Angular y es necesario desarrollar desde cero o buscar librerías de terceras para el caso de React y Vue.js. Angular ofrece el manejo de formularios reactivos con una gran cantidad de validadores y funcionalidades que permiten manejar casi cualquier tipo de formularios en una aplicación.

Para el caso de React y Vue.js es necesario el desarrollo de funcionalidades reutilizables que permiten el manejo de formularios o recurrir a librerías de terceros como Formik o Vue Formulate para lograr una verificación

de formularios para lograr una buena experiencia de usuario en una aplicación web.

4.1.4.8. Cliente HTTP

Al igual que con el manejo de formularios, el cliente HTTP es una herramienta ofrecida únicamente por Angular y delegada al usuario para el caso de React y Vue.js. El cliente HTTP de Angular ofrece compatibilidad entre las peticiones HTTP a realizar y herramientas ya utilizadas por Angular, como lo son los observables de RxJS de Angular, lo que reduce la cantidad de código a desarrollar.

En el caso de React y Vue.js es necesario recurrir a la API fetch ofrecida por los navegadores para desarrollar una herramienta desde cero capaz de realizar todas las peticiones HTTP necesarias en la aplicación o recurrir a una librería ofrecida por terceros.

4.1.5. Comparación final

Tomando como referencia los aspectos definidos en incisos anteriores, se define una comparación final de los tres *frameworks* más utilizados para el desarrollo frontend, dando una mayor ponderación a los aspectos claves que definen la conveniencia del uso del *framework* de desarrollo y una menor ponderación a las herramientas que pueden ser utilizadas a partir de librerías de terceros indicando al final el porcentaje final obtenido por cada *framework.*

Tabla I. **Tabla comparativa entre Angular, React y Vue.js**

Aspecto	Ponderación	Angular	React	Vue.js
Uso	17	54	80	51
Consultas en línea	17	75	50	25
Orientación	17	95	40	70
Manejo de interfaz de usuario	8	100	100	100
Manejo de estado	17	80	100	80
Enrutamiento	8	100	0	100
Manejo de formularios	8	100	0	0
Cliente HTTP	8	100	0	0
Total	**100 %**	**83.7 %**	**53.9 %**	**54.4 %**

Fuente: elaboración propia.

Basado en las herramientas a considerar, uso y material disponible en línea sobre los distintos frameworks de programación, se llega a la conclusión de que Angular es un framework ideal para el desarrollo de aplicaciones, ya que al ser un *framework* de desarrollo frontend manejado por Google, es seguro que este se mantenga actualizado y continúe ofreciendo gran parte de las herramientas necesarias para el desarrollo de casi cualquier aplicación web.

Por el contrario, si bien Vue.js y React también cuentan con bastante material disponible en línea y tanto Facebook como la comunidad de Vue.js ofrecen una buena continuidad del *framework*, estos no ofrecen tantas herramientas para el desarrollo de aplicaciones web, lo que reduce el tiempo de desarrollo inicial de una aplicación al tener que implementar herramientas que faciliten el desarrollo de funcionalidades en un proyecto.

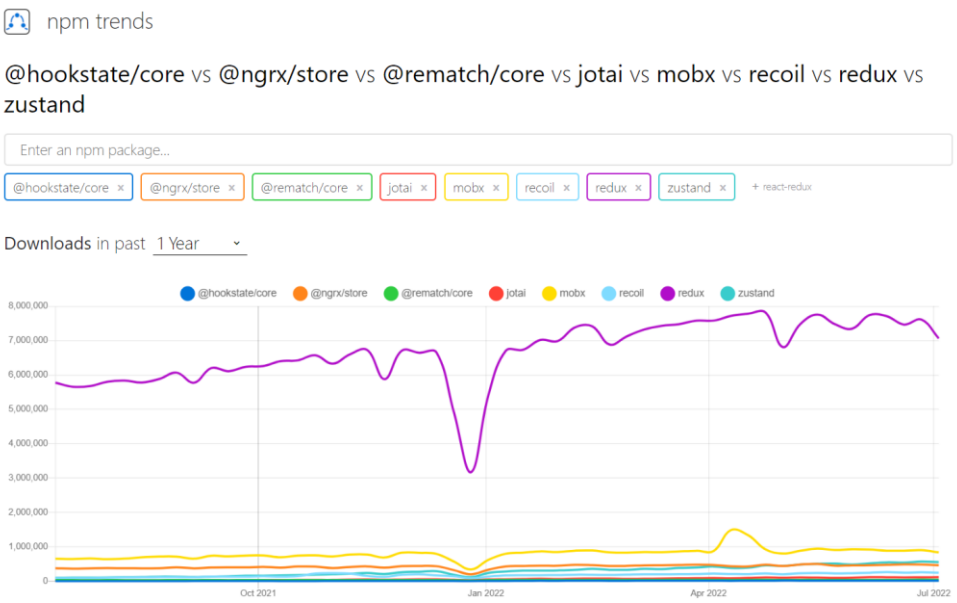
4.2. Librerías para manejo de estado de aplicaciones frontend

Al igual que en la sección de *frameworks* de desarrollo frontend, en esta sección se encuentra un análisis de las librerías para el manejo del estado de aplicaciones frontend más utilizados por la comunidad de desarrollo. El análisis está basado en las 3 librerías más usadas y la documentación disponible, cantidad de consultas promedio realizadas sobre el *framework* y en las ventajas ofrecidas por la librería.

4.2.1. Librerías más utilizadas

Para establecer cuáles son las librerías para manejo de aplicaciones frontend más utilizadas se hizo uso de la página web npm trends en donde se puede encontrar la tendencia de descargas de librerías desde el gestor de paquetes de npm. Para seleccionar las librerías a comparar se consultó blogs relacionados con las librerías para el manejo de estado de aplicaciones más populares para de esta manera identificar las tres librerías más populares y descargadas por la comunidad de desarrollo.

Figura 4. **Tendencia de uso de librerías para manejo de estado de aplicaciones frontend más utilizadas**



Fuente: Npm Trends (2022). Npm Trends. Consultado el 04 de julio de 2022. Recuperado de https://npmtrends.com/@hookstate/core-vs-@ngrx/store-vs-@rematch/core-vs-jotai-vs-mobx-vs-recoil-vs-redux-vs-zustand.

De acuerdo con las tendencias de descarga del último año realizadas a través del gestor de paquetes npm, podemos identificar que las librerías para manejo de estado de aplicaciones frontend más utilizadas son Redux, MobX y NgRx seguidos en menor cantidad por librerías como recoil, zustand y rematch.

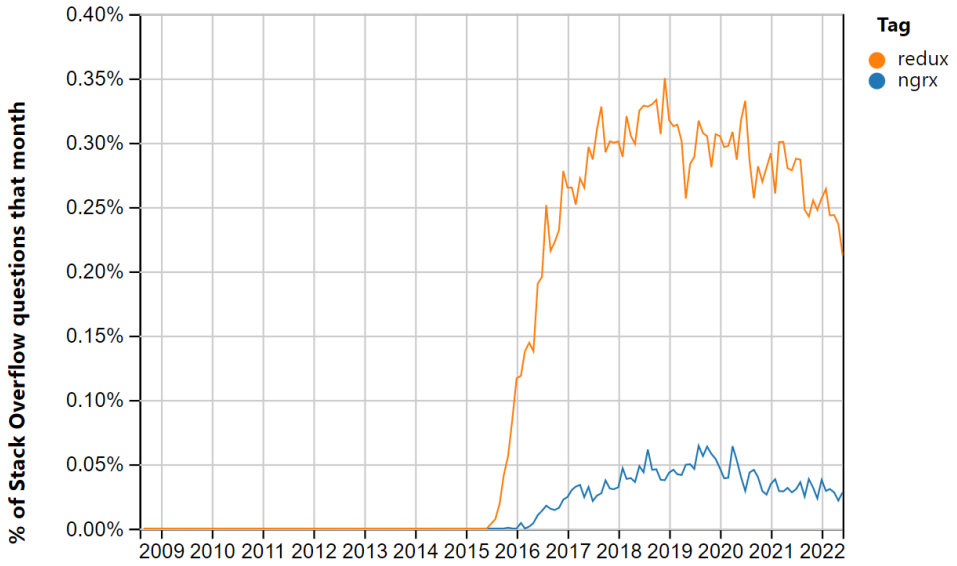
En esta gráfica también podemos identificar que la cantidad de descargas entre la librería Redux y el resto de las librerías varía por más de 5 millones de descargas, este comportamiento se debe a que Redux es una librería robusta que es compatible con la mayoría de frameworks de desarrollo frontend por lo que recurrir a otras librerías es una opción poco frecuente.

Generalmente, librerías distintas a Redux como MobX y NgRx son seleccionadas cuando se necesita compatibilidad con herramientas específicas como por ejemplo los observables de RxJS cuya compatibilidad ya es ofrecida por NgRx o cuando se desea adoptar conceptos que facilitan el desarrollo como lo es el caso de MobX que ofrece la posibilidad de manejar más de un solo almacén para el estado de una aplicación.

4.2.2. Consultas en línea

En esta sección se encuentra información acerca de las tendencias de búsquedas de Redux, MobX y NgRx en Stack Overflow y Google, esta información permite identificar si estas librerías continúan siendo utilizadas y si podemos encontrar respuestas a problemas frecuentes al manejar cualquiera de estas tres librerías.

Figura 5. **Tendencia de preguntas realizadas en Stack Overflow acerca de Redux y NgRx**



Fuente: Stack Overflow Trends (2022). Stack Overflow Trends. Consultado el 04 de julio de 2022. Recuperado de https://insights.stackoverflow.com/trends?tags=redux%2Cngrx.

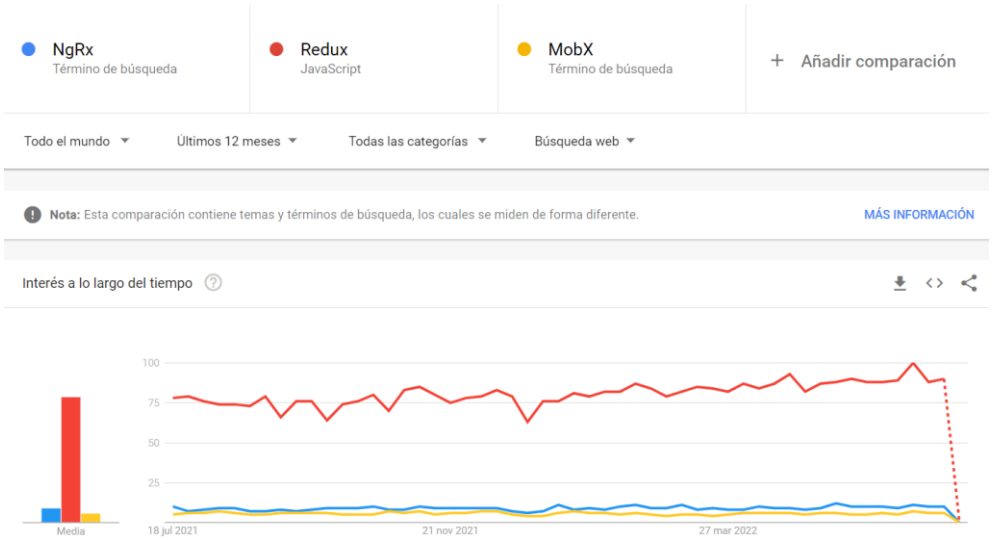
De acuerdo con la tendencia de búsqueda de preguntas relacionadas con Redux, MobX y NgRx en Stack Overflow, podemos identificar que Redux y NgRx presentan valores significativos en las cantidades de preguntas realizadas en Stack Overflow, siendo MobX una excepción al no ser una tendencia en preguntas dentro de Stack Overflow.

Si bien MobX no presenta una tendencia de búsquedas en Stack Overflow, esto no quiere decir que no se encuentren dudas relacionadas con este framework en Stack Overflow, sino que se presentan, pero no como para ser una tendencia en la plataforma Stack Overflow.

Es de esperarse que la tendencia de preguntas de Redux respecto a NgRx sea de casi más del 0.30 %, esto se debe a que la cantidad de descargas de Redux respecto a NgRx varía por más de 5 millones mensualmente, por lo que es más frecuente encontrar preguntas respecto a este *framework.*

De acuerdo con la gráfica de Stack Overflow también puede identificarse que la tendencia de preguntas de Redux y NgRx se ha mantenido a lo largo del tiempo, por lo que adicional a la documentación de cada librería podemos encontrar material de apoyo adicional en la plataforma Stack Overflow para solucionar problemas durante el desarrollo de software.

Figura 6. **Tendencia de búsquedas realizadas en Google acerca de NgRx, Redux, MobX**



Fuente: Google Trends (2022). Google Trends. Consultado el 04 de julio de 2022. Recuperado de https://trends.google.com/trends/explore?q=NgRx,%2Fg%2F11dxf0gf92,MobX.

Basados en las tendencias de búsquedas de NgRx, Redux y Mobx podemos identificar el mismo comportamiento en la tendencia de búsquedas de Google respecto a la tendencia de descargas y búsquedas en Stack Overflow, presentando una diferencia de más de 75 búsquedas semanales de Redux respecto a NgRx y MobX, aunque manteniendo una tendencia regular de búsquedas en las librerías.

De acuerdo con las gráficas de consultas realizadas en línea de las librerías más utilizadas, podemos identificar que gran parte de la información relacionada con Redux, NgRx y MobX puede ser encontrada en sus respectivas documentaciones, blogs en línea y en respuestas en la plataforma Stack Overflow por lo que existe suficiente material de referencia para el desarrollo de estados de aplicaciones con Redux, NgRx y MobX.

4.2.3. Comparativa de manejo de librerías

La mayor parte de librerías dedicadas al manejo del estado de aplicaciones están basadas en el manejo de una o varias fuentes de datos que permitan a múltiples componentes comunicarse entre sí, a pesar de que estas librerías se centran en este concepto, cada una de estas librerías cuentan con características que las hacen únicas para distintos tipos de problemas. En esta sección se encuentra comparaciones de las diferencias principales entre las tres librerías más populares para el manejo del estado de aplicaciones frontend.

Tabla II. **Tabla comparativa con funcionalidades de Redux, MobX y NgrX**

Característica	Redux	MobX	NgRx
Manejo de Store	Un único objeto con la información del estado de la aplicación	Capacidad de manejo de más de un objeto para almacenar el estado de una aplicación	Un único objeto con la información del estado de la aplicación
Compatibilidad con la mayoría de frameworks	Compatibilidad con la mayoría de frameworks de desarrollo frontend	Compatibilidad con la mayoría de frameworks de desarrollo frontend	Compatibilidad reducida al ser desarrollado principalmente para usarse con Angular
Tipo de desarrollo durante uso de librería	Funcional	Orientado a objetos	Funcional
Modificación de estado	Funciones puras	Modificación directa en variables	Funciones puras
Dificultad de aprendizaje	Alta por complejidad en modificación de estado	Media por facilidad en modificación de estado	Alta por complejidad en modificación de estado
Compatibilidad con RxJS	Requiere de librerías adicionales	Requiere de librerías adicionales	Compatibilidad ya integrada

Fuente: elaboración propia.

4.2.4. Comparación final

Tomando como referencia los aspectos definidos en incisos anteriores, se define una comparación entre las librerías Redux, NgRx y MobX. En esta sección se encuentran 2 comparaciones, la primera comparación está orientada al

manejo de la librería con cualquier framework de desarrollo frontend y la segunda comparación orientada al framework de desarrollo Angular.

Tabla III. **Tabla comparativa Redux, MobX, NgRx con orientación a cualquier framework de desarrollo frontend**

Característica	Ponderación	Redux	MobX	NgRx
Uso	25	95	20	25
Consultas en línea	25	90	25	30
Compatibilidad con la mayoría de frameworks	25	80	50	20
Facilidad de aprendizaje	15	60	70	60
Manejo de store	10	80	90	80
Total	**100**	**83.25 %**	**43.25 %**	**35.75 %**

Fuente: elaboración propia.

Desde un punto de vista general, Redux es indudablemente una librería ideal para cualquier tipo de framework de desarrollo frontend, esto se debe a que, gracias a la gran cantidad de uso de la librería, la comunidad de desarrollo, cantidad de recursos en línea y a la gran cantidad de librerías de terceros creadas para esta librería se es capaz de adaptar Redux a la mayoría de *frameworks* de desarrollo frontend.

Tabla IV. **Tabla comparativa entre Redux, MobX y NgRx con orientación al framework Angular**

Característica	Ponderación	Redux	MobX	NgRx
Uso	20	95	20	25
Consultas en línea	20	90	25	30
Compatibilidad con Angular	20	30	20	100
Compatibilidad con RxJS	20	30	20	100
Manejo de store	10	80	90	80
Facilidad de aprendizaje	10	50	40	70
Total	**100**	**62 %**	**30 %**	**66 %**

Fuente: elaboración propia.

Orientando la decisión de selección de la librería de manejo de estado de una aplicación hacia el *framework* de desarrollo frontend Angular, la librería ideal a usar es NgRx, esto se debe gracias a que fue creado para Angular y la compatibilidad que ofrece con las herramientas de Angular es realmente alta, ejemplo de esto es la compatibilidad con RxJs por lo que adicional a la compatibilidad y reducción de tiempo de aprendizaje por la compatibilidad con herramientas de Angular también se logra mantener la cantidad de recursos de referencia para el desarrollo de Angular con NgRx.

5. PROPUESTA DE DISEÑO PARA EL MANEJO DEL ESTADO DE APLICACIONES CON ANGULAR Y NGRX

A pesar de que Angular ofrece la mayoría de las herramientas necesarias para crear una aplicación web, es necesario ser capaces de organizar los módulos y componentes de una aplicación de tal manera que cualquier desarrollador pueda comprender y reutilizar componentes de la mejor manera posible. De esta manera aseguramos una fácil continuidad del desarrollo de una solución web.

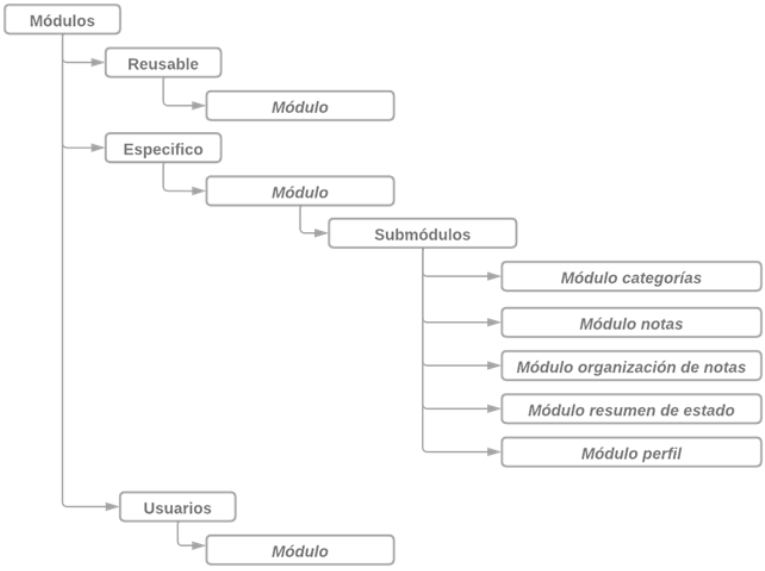
5.1. Organización de módulos

Para el manejo de los módulos se recomienda segmentar los módulos en cuatro grandes grupos, el primer módulo corresponde al módulo creado por defecto en una aplicación de Angular, el segundo módulo orientado al manejo de usuarios en la aplicación, el tercer módulo correspondiente a funcionalidades reutilizables y finalmente el cuarto módulo correspondiente a módulos específicos en una aplicación.

Para organizar los módulos se debe producir un directorio en donde se encuentren todos los módulos de la aplicación, a excepción del módulo generado por defecto para la aplicación de Angular. En este directorio se incluirá un subdirectorio específico para la creación del módulo de trabajo de funcionalidades específicas que no son reutilizadas en la aplicación y por el otro lado también se encontrará el subdirectorio de módulos reutilizables, en este caso contendrá el módulo y todos los componentes reutilizables dentro de la aplicación.

En el directorio de módulos se propone incluir todos los submódulos dependiendo de si su naturaleza es reutilizable o específica, también se debe tomar en cuenta un módulo puede desligarse del directorio de módulos reutilizable y específicos con el objetivo de mantener un mejor orden, un ejemplo de un tipo de módulo que puede ser desligado de módulos específicos y reutilizable es el módulo de usuarios dedicado al inicio de sesión y registro de usuarios.

Figura 7. **Propuesta de organización de módulos**



Fuente: elaboración propia, realizado con Lucid Chart.

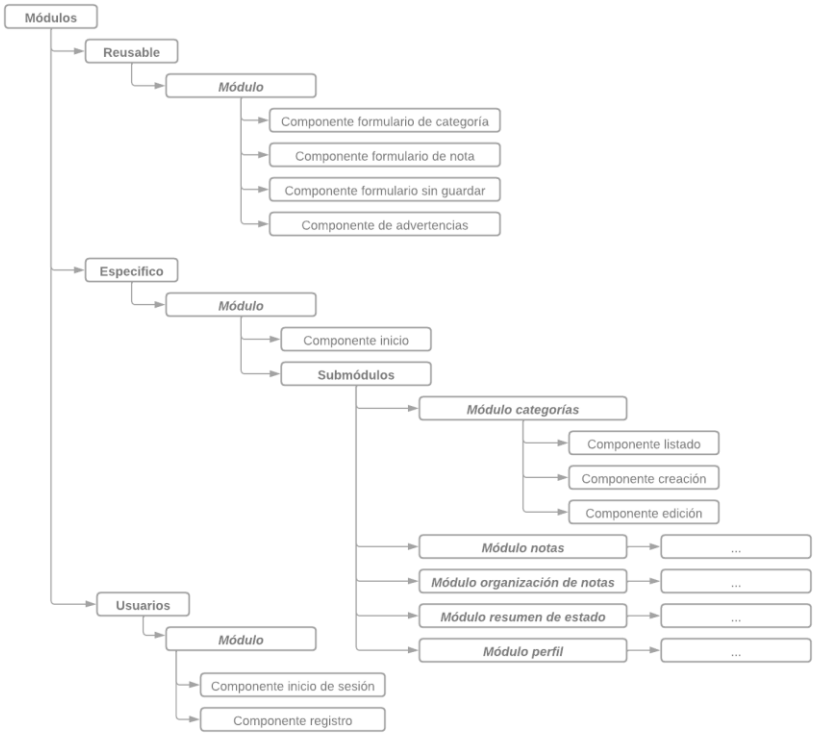
5.2. Presentación de componentes

Para el manejo de los componentes web se recomienda continuar con el patrón creado para la organización de módulos. Al generar componentes se debe identificar la naturaleza del componente respecto a los módulos generados y en

caso de que el componente no se asocie a ninguno de los módulos generados previamente, se procede a generar un nuevo módulo o submódulo para que este componente pueda ser asociado.

Para la propuesta establecida todos los componentes estarán asociados, ya sea al módulo de componentes reutilizables, módulo de usuarios o asociado en alguno de los submódulos creados en el módulo de funcionalidades específicas.

Figura 8. **Ejemplo de organización de componentes**



Fuente: elaboración propia, realizado con Lucid Chart.

5.2.1. Organización de componentes reutilizables

Durante la creación de componentes reutilizables se recomienda incluir todos los componentes que son utilizados de una forma intensiva dentro de una aplicación. Ejemplos claros de estos componentes podrían ser modales de confirmación, alertas y formularios reutilizables.

Se recomienda usar la directiva @Input de Angular para ofrecer una fácil continuidad en el desarrollo de proyectos. Con la directiva @Input se puede ofrecer la posibilidad de manejar información proveniente de componentes padres, un ejemplo puede es la creación de variables que indiquen si un formulario está orientado a creación o edición.

La directiva @Output ofrece la posibilidad de disparar eventos que comuniquen acciones desde un componente hijo hacia un componente padre, un ejemplo es la confirmación o cancelaciones de eventos que ocurren desde un componente hijo, pero que son tratados de distintas maneras desde distintos componentes padres.

Es importante tomar en cuenta que estas directivas deben utilizarse de tal forma que se logre una buena continuidad en el desarrollo de los componentes web, de esta manera se asegura de que, si bien los componentes específicos pueden llegar a cambiar su lógica durante el desarrollo de la solución, los componentes reutilizables siempre mantendrán una independencia a través de este tipo de directivas.

5.2.2. Organización de componentes específicos

A diferencia de los componentes reutilizables, el desarrollo de componentes específicos es más complejo, ya que estos componentes cumplen con cierta lógica de negocio que hace que la reusabilidad de estos componentes sea casi nula, por lo que estos componentes reutilizan otros componentes para generar la lógica final de la aplicación.

Estos componentes deben de encargarse de realizar todas las transformaciones necesarias de la información que requiera los componentes reutilizables, de esta manera si a futuro el comportamiento de los componentes específicos puede cambiar debido a reglas de negocio, en ningún momento estaríamos alterando el comportamiento de los componentes reutilizables lo cual asegura una correcta continuidad en el manejo de componentes específicos y reutilizables.

Algunos ejemplos de estos componentes son los componentes de inicio de sesión, recuperación de contraseñas y formularios. Esta clase de componentes son muy sencillos de reconocer, ya que son vistas que generalmente se encuentran una sola vez dentro de la aplicación y que hacen uso de componentes reutilizables para llevar a cabo el desarrollo final del componente.

Como se mencionó anteriormente, estos componentes son rara vez reutilizados, por lo que es raro utilizar las directivas @Input y @Output durante el desarrollo de estos componentes, por el otro lado estos componentes utilizan selectores del estado de la aplicación para obtener la información necesaria para cumplir con las reglas de negocio, estos componentes contienen los métodos

para navegar entre pantallas y a su vez son los encargados de ejecutar los servicios web para modificar la información dentro de una solución.

5.3. Lógica de datos

Estructurar de forma correcta el manejo de los datos de una aplicación también es una tarea importante para asegurar la continuidad de un proyecto. Se debe asegurar que la obtención información por parte de servicios externos sea accesible desde cualquier componente de nuestra aplicación y que a su vez sepamos cuáles son los atributos obligatorios y opcionales de cada modelo a trabajar dentro de nuestros datos.

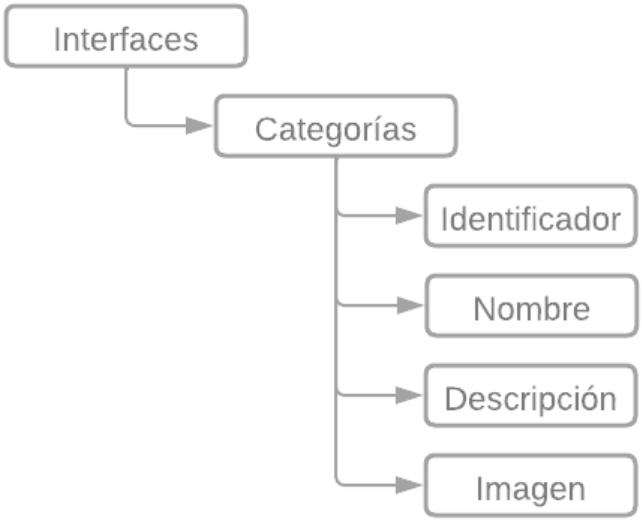
5.3.1. Organización de modelos

Se debe aislar los modelos a utilizar en una aplicación web, de manera que cualquier desarrollador sepa que atributos son obligatorios y que atributos son opcionales dentro de un modelo de datos, esto no solo brinda la ventaja de saber cómo está estructurada la información dentro de nuestra aplicación, sino que también ofrece la ventaja de asignar tipos de datos a variables y peticiones HTTP de servicios externos.

Se recomienda hacer uso de interfaces y enumeradores para la creación de modelos en Angular, de esta manera nos aseguramos de que cualquier desarrollador use siempre el mismo tipo de datos y asegure la continuidad de desarrollo en el caso de agregar futuros tipos de datos o modificar la estructura en los tipos de datos. Se recomienda el uso de interfaces exportables que hagan empleo de enumeradores para que se asignen tipos de datos a cualquier tipo de variables dentro en una solución.

Para organizar los modelos de datos se recomienda un directorio con todos los modelos a utilizar dentro de la solución web, al igual que con los componentes web, con esto se logra que el desarrollador encuentre fácilmente el modelo de datos con el que desea trabajar para saber que campos son requeridos y cuáles no para poder realizar una asignación de tipos correcta a las variables dentro de la solución.

Figura 9. **Ejemplo de organización de modelos**



Fuente: elaboración propia, realizado con Lucid Chart.

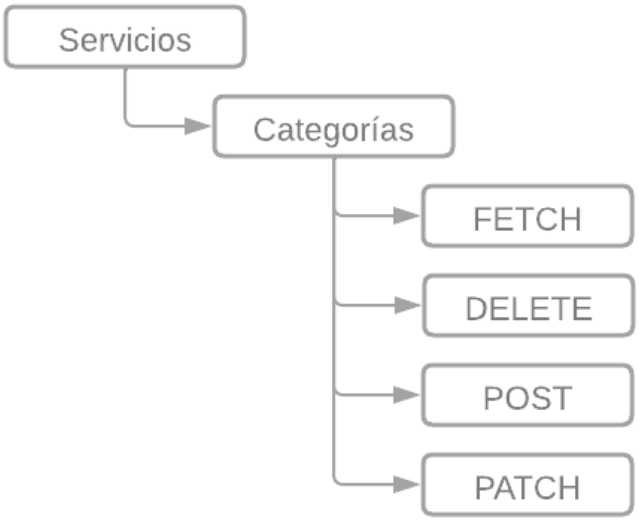
5.3.2. Organización de servicios

Es de gran utilidad separar las peticiones de servicios externos de tal manera que estos puedan ser utilizados en cualquier parte de la aplicación en caso de ser necesario, un ejemplo de esto es la utilización de servicios de usuarios y productos para realizar una petición de compra, al separar las

peticiones de servicios e inyectarlos en distintos componentes se permite que un servicio que fue declarado una sola vez en la aplicación pueda ser reutilizado en cualquier componente de la aplicación sin necesidad de duplicar grandes cantidades de código durante la creación de componentes.

Para la organización de servicios se propone hacer uso de un solo directorio en el que se encuentren todos los archivos con todos los servicios relacionados con el área que se trabaja por cada módulo, de esta forma cualquier desarrollador sabrá en qué directorio se encuentran los servicios relacionados con el desarrollo de la aplicación.

Figura 10. **Ejemplo de organización de servicios**



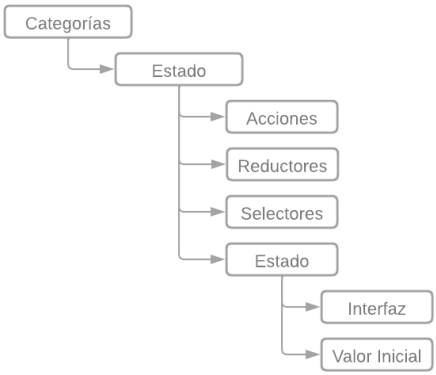
Fuente: elaboración propia, realizado con Lucid Chart.

5.4. Administración del estado de una aplicación

La organización del estado de una aplicación también definirá una fácil continuidad en el desarrollo de un proyecto, ya que de hacerlo de forma incorrecta ocasionará no solo que sea difícil de entender para un grupo de desarrolladores, sino que también la cantidad de recursos que se consuman al hacer uso del estado de la aplicación sean demasiado grandes.

Para administrar el estado de la aplicación se propone un directorio dedicado específicamente al manejo de las acciones, selectores y reducciones del estado de un componente. Esta administración se propone separarla en relación los módulos que se planea desarrollar en la aplicación. En este directorio se propone incluir una interfaz para establecer el tipo de dato, nuestro estado, las acciones que se llevaran a cabo en el componente, los selectores de la información de nuestro estado para ahorrar recursos y las reducciones que mutaran nuestro estado de nuestros componentes.

Figura 11. **Ejemplo de organización de estado**



Fuente: elaboración propia, realizado con Lucid Chart.

5.4.1. Estado

El manejo de los estados de una aplicación con NgRx funciona a través de un Store general en donde se guardan todos los estados creados para el manejo de una aplicación, uno de los inconvenientes con este tipo de manejo es que al establecer un tipo para el Store de NgRx nos enfrentamos al inconveniente de que cuando creamos nuevos estados debemos actualizar el tipo en cada uno de los lugares en donde estamos haciendo uso del Store, esta actualización en el código de la aplicación hace que el desarrollo de la aplicación se convierta en una tarea compleja y genere una difícil continuidad a largo plazo.

Debido al inconveniente que representa establecer un tipo para el manejo de los estados en una aplicación, es importante evaluar alternativas que permitan asegurar fácil continuidad a la aplicación para que cuando nuevos desarrolladores se integren al equipo sean capaces de entender de una forma rápida y sencilla como se está manejando el estado de la aplicación.

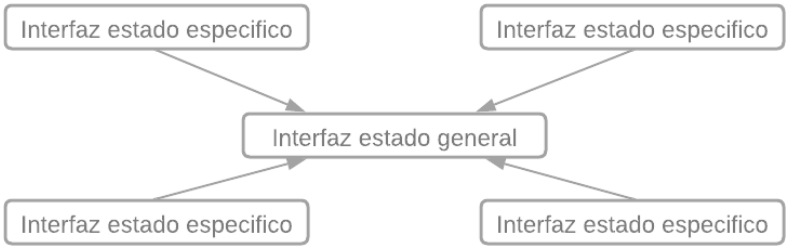
Una posible alternativa es la creación de un estado general que contenga todos los tipos de los estados dedicados a cada componente, de esta manera se creará una fácil continuidad en donde al agregar nuevos estados para nuevos componentes no implica una refactorización de código compleja, sino que al contrario solo implica agregar el nuevo estado creado a la interfaz general de nuestra aplicación.

5.4.1.1. Estado general

Para llevar a cabo el desarrollo del estado general se propone crear una interfaz como un nuevo tipo que haga uso de las interfaces creadas para cada estado de la aplicación, de esta forma se logra crear un tipo general que hace

uso de tipos específicos que permiten asignar un tipo general al Store de NgRx. Esta transformación de tipos específicos a generales nos ofrece una fácil continuidad y comprensión para el desarrollo del estado de una aplicación generada con NgRx.

Figura 12. **Ejemplo de organización de estado general**



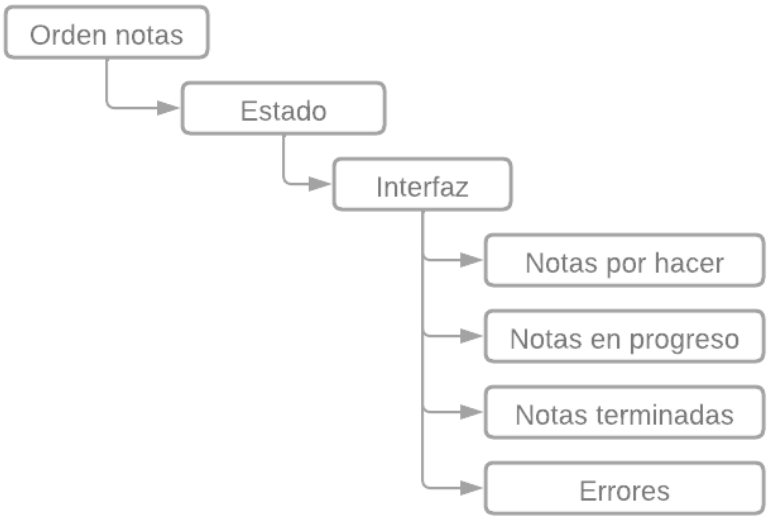
Fuente: elaboración propia, realizado con Lucid Chart.

5.4.1.2. Estado dedicado

Para el desarrollo del estado dedicado de la aplicación se propone crear una interfaz con toda la información relacionada con el estado de un componente, en esta interfaz se incluirían los tipos de información como listado de ítems, errores y cualquier otro tipo de información relacionada con el manejo del estado de un componente.

Para el manejo de errores se propone incluir dentro de la interfaz un arreglo capaz de almacenar todos los errores que ocurrieron durante una solicitud HTTP y este arreglo será de tipo error, el cual corresponderá a una interfaz que incluye el tipo de solicitud y el mensaje ocurrido durante el error.

Figura 13. **Ejemplo de organización de estado dedicado**



Fuente: elaboración propia, realizado con Lucid Chart.

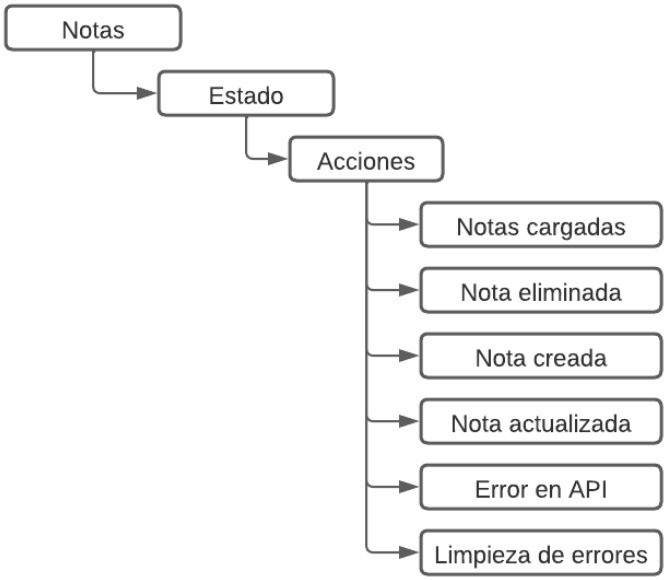
5.4.2. Acciones

El manejo de las acciones es considerablemente más flexible y fácil de implementar, ya que el manejo de las acciones en NgRx se ejecutan a través de un método en el objeto store de NgRx por lo que estas acciones pueden declararse en donde sea, porque las acciones son llamadas cuando hay cambios en un componente por lo que se puede tener todas las acciones en un solo archivo o separadas dependiendo de la orientación que se desee implementar.

Se recomienda separar las acciones con base en el estado de un componente con el objetivo de tener un orden en específico, esto con el objetivo de mantener un orden y que en el caso de modificar alguna de las acciones se pueda encontrar fácilmente basados en el estado del componente. En este

archivo se recomienda incluir todas las acciones que se llevan a cabo en un componente, ejemplos de las acciones podrían ser la carga de ítems, creación de ítem, actualización de ítem, creación de error de API, creación de error de componente y limpieza de estado.

Figura 14. **Ejemplo de organización de acciones**



Fuente: elaboración propia, realizado con Lucid Chart.

5.4.3. Reductores

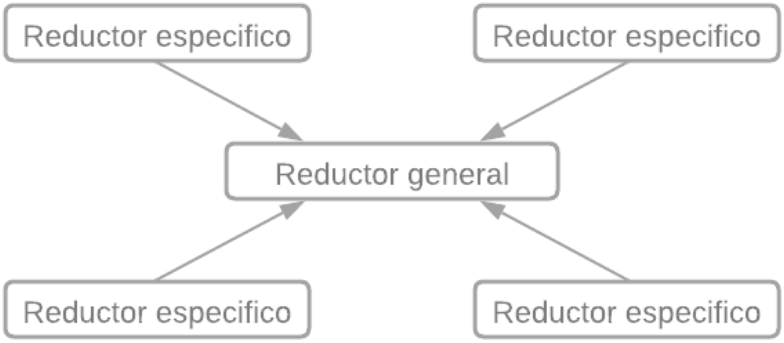
El manejo de los reductores dentro de NgRx se realiza a través de la importación de la función forRoot del módulo de Store dentro de NgRx, al realizar esta importación es necesario enviar como parámetro un objeto con el conjunto de reductores y las reducciones necesarias para manipular el estado de la aplicación que estamos creando en Angular.

Algo muy importante a tomar en cuenta al proceder con la importación de las reducciones de NgRx es el inconveniente de modificar el módulo de la aplicación para generar un nuevo atributo dentro del objeto de reducciones de NgRx. Esto suele ser un inconveniente en aplicaciones demasiado grandes en donde la cantidad de módulos y componentes importados es demasiado grande y la búsqueda de reducciones se vuelve ligeramente compleja, por esta razón es importante separar el objeto de reducciones para posteriormente ser importado.

5.4.3.1. Reductor general

Para mejorar el manejo de las reducciones de la aplicación en NgRx se propone crear una constante exportable en donde se incluya el conjunto de reducciones necesarias para ese módulo en específico. El generar esta constante exportable nos permite fácilmente identificar en donde se encuentran las reducciones correspondientes a cada módulo de la aplicación de Angular y asegurar que cualquier desarrollador pueda darle continuidad al proyecto sin necesidad de tener que realizar modificaciones propias dentro del módulo en donde se modifiquen las reducciones.

Figura 15. **Ejemplo de organización de reductor general**



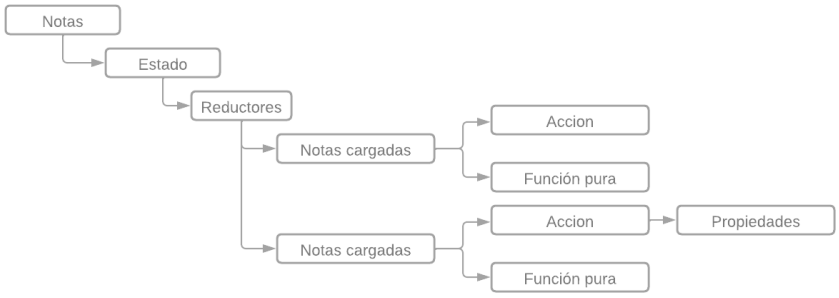
Fuente: elaboración propia, realizado con Lucid Chart.

5.4.3.2. Reductor dedicado

El manejo de reductores dedicados se realiza basados en acciones y funciones puras, es importante tomar en cuenta que por cada acción creada dentro del archivo de acciones es necesario crear una función pura que transforme el estado de nuestra aplicación. Para la creación del reductor dedicado se debe aprovechar las acciones generadas previamente, por lo que dichas acciones deben ser exportable para poder utilizarlas no solo para la creación de reductores, sino que también en otros componentes externos.

En caso de que la aplicación sea demasiado grande puede considerarse separar las funciones puras del área de reducciones y crearlas de una forma exportable, de esta forma se puede mantener un mejor orden entre funciones y reducciones de la aplicación, aunque esta alternativa no es totalmente necesaria, ya que la creación de las reducciones se realiza a través haciendo uso de la función creada y de la acción importada.

Figura 16. **Ejemplo de organización de reductor dedicado**



Fuente: elaboración propia, realizado con Lucid Chart.

6. IMPLEMENTACIÓN DE PROPUESTA PARA EL MANEJODEL ESTADO DE APLICACIONES CON ANGULAR Y NGRX

Una vez se tiene claro el patrón de la organización de los componentes, estados, reducciones, acciones y selectores de la aplicación que ofrecen una fácil continuidad en el desarrollo de una aplicación a largo plazo, es importante convertir de manera correcta el patrón propuesto hacia una aplicación de Angular. Es conveniente aprovechar al máximo la interfaz de comandos de línea de Angular y utilizar correctamente NgRx para llevar a cabo el desarrollo del estado de nuestra aplicación para maximizar la cantidad de desarrollo.

6.1. Implementación de componentes

Para llevar a cabo la creación de los componentes se debe iniciar evaluando el tipo de componente que se debe crear. Dependiendo el tipo de componente se comienza la implementación basada en sí, el componente necesita de directivas para ofrecer la fácil continuidad de su uso o de si necesita de generar un directorio para dichos tipos de componentes.

Para la creación de componente se propone hacer uso de la línea de comandos de Angular, ya que esta línea de comandos ofrece la ventaja de crear los archivos necesarios para la creación de un componente y la importación automática de dichos componentes al módulo de la aplicación de Angular.

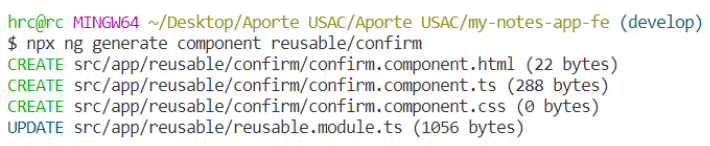
6.1.1. Componentes generales

Al iniciar con la creación de componentes generales es necesario crear un directorio de componentes y un directorio de componentes generales, en caso de no haberlos creado previamente, estos directorios deben de estar dentro del directorio fuente de nuestra aplicación. Una vez se tengan esos directorios se puede pasar a la implementación detallada del componente.

6.1.1.1. Creación de un componente

Realizar la creación del componente haciendo uso de la línea de comandos de Angular es sencillo debido a que los módulos de Angular se encargan de realizar la creación de cada uno de los archivos necesarios para la creación de un componente. Para proceder con la creación se propone hacer empleo del comando de creación de componentes de Angular indicando la ruta en donde se debe de crear dicho componente, para esta propuesta la creación del componente sería dentro del directorio de componentes reutilizables.

Figura 17. **Ejemplo de creación de componente**



Fuente: elaboración propia, realizado con Git Bash.

6.1.1.2. Directivas @Input

Posterior a la creación de un componente reutilizable es importante evaluar la necesidad de hacer uso de directivas @Input para ofrecer una alta reusabilidad de un componente, si se usa de referencia un componente de tipo formulario, se hace evidente la necesidad de crear directivas @Input para permitir la variación en los títulos y elementos a incluir dentro del formulario, de esta forma no se genera la necesidad de crear un componente para la creación y edición de un elemento.

Es importante tomar en cuenta que las directivas @Input pueden ser vistas como variables creadas desde un componente padre, por lo que deben ser utilizadas cuando un componente es utilizado de forma recurrente en múltiples componentes padres en donde lo único que cambia son ciertos objetos o condiciones que alteran levemente el aspecto o comportamiento del componente reutilizado, el ejemplo más claro para este caso es la presentación de información de una lista de elementos en donde lo uno que cambia es la información que se presenta en cada elemento y no la estructura del elemento.

Figura 18. **Ejemplo de uso de directivas @Input**

```
export class CategoryFormComponent implements OnInit, OnDestroy {

  @Input() formType: 'edit' | 'create' = 'create';
  @Input() category: Category | null = null;
  @Output() saveEvent: EventEmitter<Partial<Category>> = new EventEmitter();
  @Output() cancelEvent: EventEmitter<boolean> = new EventEmitter();
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.1.1.3. Directivas @Output

Otro punto para tomar en cuenta durante el desarrollo de un componente reutilizable son las directivas @Output las cuales permiten desencadenar eventos en una aplicación para informar eventos que han ocurrido en componente hijo.

Esta directiva nos ofrece la posibilidad de crear una reusabilidad aún más alta de un componente, ya que acciones de creación, edición, eliminación o cancelación pueden ser informadas directamente desde un componente hijo hacia un componente padre sin verse obligado a definir el comportamiento que de las acciones en el componente hijo, sino que, por el contrario, definir el comportamiento de las acciones desde un componente padre, un ejemplo es la reutilización de un formulario de edición o creación, esta directiva es la encargada de emitir el evento ocurrido y el componente padre el encargado de decidir que debe ocurrir con la información emitida por el evento.

Figura 19. **Ejemplo de uso de directiva @Ouput para confirmar acciones**

```
save(): void {
  this.form.markAllAsTouched();
  if (this.form.valid) {
    this.saveEvent.emit(this.form.value);
  }
}

cancel(): void {
  this.store.dispatch(categoryActions.resetApiErrors());
  this.cancelEvent.emit(true);
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.1.2. Componentes específicos

El desarrollo de componentes específicos generalmente se desarrolla para cumplir cierta lógica de negocios requerida para el desarrollo de una solución, por lo que es muy raro hacer uso de directivas como @Input y @Output durante el desarrollo de estos componentes, si no que, por el contrario, estos componentes reutilizan componentes reutilizables en la aplicación.

El desarrollo de estos componentes se realiza de la misma forma en la que se realiza un componente reutilizable, la única variación se encuentra en la reutilización de componentes y en la evaluación de la necesidad de hacer uso de las directivas @Input y @Output.

6.1.3. Reutilización de componentes

Para llevar a cabo la reutilización de un componente es necesario identificar el componente a reutilizar y hacer uso de su selector para llevar a cabo la reutilización del componente, también es importante verificar si dicho componente hace uso de directivas @Input para que de esta manera desde el componente padre se indique los valores que dicho componente hijo tendrá.

Respecto a las directivas @Output al reutilizar componentes se debe desarrollar las funciones que lleven a cabo las acciones necesarias cuando un evento es emitido dependiendo de las reglas de negocio con las que deba cumplir el componente padre con la información emitida por parte del componente hijo.

Figura 20. **Ejemplo de uso de reutilización de componente**

```
<div class="card mb-3 p-4" *ngFor="let category of categoriesObservable | async">
    <div class="d-flex flex-row align-items-center">
        <div class="me-5 no-display">
            <img [src]="category.image" alt="" width="90" height="90" class="rounded-circle me-2"
            loadImgError="assets/image-error.png">
        </div>
        <div class="me-auto">
            <div class="ellipsis">
                <b>Nombre: </b> {{category.name}}
            </div>
            <div class="ellipsis">
                <b>Descripción: </b>{{category.description}}
            </div>
        </div>
        <div class="d-flex flex-sm-row flex-column">
            <button class="btn btn-link text-decoration-none text-primary me-5 me-i"
            (click)="editCategory(category._id)">
                <i class="bi bi-pencil me-2"></i> Editar
            </button>
            <button class="btn btn-link text-decoration-none text-danger me-4 me-i"
            (click)="deleteCategory(category._id)">
                <i class="bi bi-trash me-2"></i> Eliminar
            </button>
        </div>
    </div>
</div>
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.2. Implementación de lógica de datos

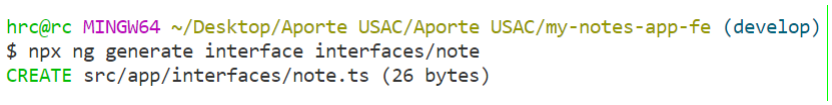
Tener un correcto manejo de los datos a utilizar en el desarrollo de una aplicación de Angular es importante porque de no realizarlo de una forma correcta haremos que el código de la aplicación no sea mantenible y que solo las personas que estuvieron directamente involucradas en el desarrollo de esta parte de la solución sepan cómo funciona mientras que, por el contrario, al implementarlo de forma correcta los modelos y servicios permitirán que cualquier desarrollador nuevo se adapte fácilmente a los datos que se manejan en la aplicación.

6.2.1. Modelo de datos

Teniendo en cuenta que Angular es un framework de desarrollo para el lenguaje TypeScript, debemos aprovecha la ventaja de poder asignar tipos no solo a nuestras variables, sino que también a los datos que obtenemos desde un servicio externo, para esto se propone crear interfaces exportables con el comando de líneas de Angular que contengan los tipos necesarios para el desarrollo de los modelos de datos, de esta manera aseguramos que la interfaz pueda ser utilizada en cualquier componente o servicio de nuestra aplicación.

Otro punto importante para tomar en cuenta es la ventaja de la programación orientada a objetos que nos ofrece el lenguaje TypeScript, por lo que de ser necesario pueden implementarse clases e instancias de objetos que apoyen en el manejo de la información de datos dentro de nuestra aplicación, aunque regularmente basta con implementar interfaces exportables para establecer los tipos de datos a manejar en nuestra aplicación.

Figura 21. **Ejemplo de creación de interfaz con línea de comandos de Angular**



Fuente: elaboración propia, realizado con Git Bash.

Figura 22. **Ejemplo de creación de interfaz exportable**

```
export interface Note {
    _id: string;
    name: string;
    description: string;
    image?: string;
    category: string;
    status: string;
    order: number;
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.2.2. Servicios

Debido a que la mayoría de las aplicaciones web hacen uso de servicios de terceros, no solo para obtener datos que son desplegados en una aplicación web, sino que también hacen uso de servicios para modificar esta información, es importante crear servicios de tal manera que estos puedan ser llamados desde cualquier componente dentro de la solución.

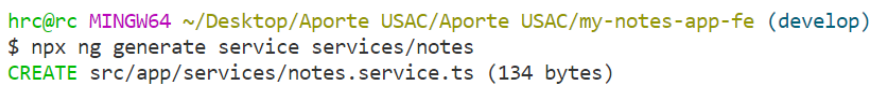
Para lograr que los servicios sean accesibles dentro de cualquier componente de nuestra aplicación y no exista necesidad de generar el mismo servicio en varios componentes, es recomendable hacer uso de inyectores, de tal manera que los servicios se generen una sola vez y estos sean inyectados en los constructores de los componentes que lo necesiten.

Se recomienda hacer empleo del módulo HttpClient de Angular para lograr realizar peticiones HTTP a servicios externos sin importar si estamos obteniendo o modificando información, al igual que el uso del módulo HttpClient se

recomienda hacer uso de la librería RxJs para lograr que las peticiones HTTP realizadas sean reactivas a través de la creación de observables y suscripciones que definan que acciones se deben generar cada vez que se obtiene una respuesta de un servicio externo.

Para facilitar la creación de servicios en Angular también se recomienda hacer uso de la línea de comandos de Angular que crea automáticamente el archivo base con la definición del inyector necesario para definir cada una de las funciones que se encargaran del manejo de las peticiones HTTP.

Figura 23. **Ejemplo de creación de servicio con línea de comandos de Angular**



Fuente: elaboración propia, realizado con Git Bash.

Figura 24. **Ejemplo de servicio inyectable en Angular**

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';
import { Note } from '../interfaces/note.interface';
@Injectable({
  providedIn: 'root'
})
export class NotesService {
  constructor(private http: HttpClient) { }
  fetchNotes(): Observable<Note[]> {
    return this.http.get<Note[]>(`${environment.notes}`);
  }
  deleteNote(id: string) {
    return this.http.delete(`${environment.notes}/${id}`);
  }
  saveNote(note: Partial<Note>) {
    return this.http.post<Note>(`${environment.notes}`, note);
  }
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.3. Implementación de estados

Posterior a la creación de los módulos, componentes y servicios necesarios para el desarrollo de la solución web podemos pasar a la definición de los estados de nuestra aplicación, para este paso se recomienda crear un estado por cada módulo de una aplicación, de tal manera que todos los componentes que influyen significativamente en el cambio del estado ya se encuentren dentro del módulo a modificar, de igual manera en caso de ser usado por un componente reutilizable sabremos donde encontrar los archivos del estado de una forma rápida y sencilla.

Tal como se mencionó durante la propuesta de la estructura para el manejo del estado de una aplicación, es necesario crear un estado general adicional al estado que se crea para cada módulo para lograr asignarle un tipo

de dato al Store de NgRx y asegurar un mejor control del estado actual de la aplicación.

6.3.1. Estado general

Para el desarrollo del estado general es necesario importar cada uno de los estados creados en cada módulo definido y luego crear una interfaz haciendo uso de cada uno de los estados importados para lograr crear un tipo asignable para el estado general de NgRx.

Para hacer uso de este estado general debemos de importar este estado en cada uno de los componentes en donde haremos consultas al estado de la aplicación, este estado se utilizará como tipo durante la inyección del Store de NgRx en el constructor de cada componente donde se hará uso de este.

Figura 25. **Ejemplo de creación de estado general**

```
import { CategoriesState } from "../specific/categories/state/categories.state";
import { LoaderState } from "../specific/loader/loader.state";
import { NotesState } from "../specific/notes/state/notes.state";
import { SortNotesState } from "../specific/sort-notes/state/sort-notes.state";
import { UnsavedFormState } from "../specific/unsaved-forms/unsaved-forms.state";

export interface AppState {
    categories: CategoriesState;
    loader: LoaderState;
    notes: NotesState;
    ['sort-notes']: SortNotesState;
    ['unsaved-forms']: UnsavedFormState;
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 26. **Ejemplo de uso de estado general**

```
export class SummaryComponent implements OnInit, OnDestroy {

  subscriptions: Subscription[] = [];
  state: any = null;
  loadForm: FormGroup = new FormGroup({});
  changesForm: FormGroup = new FormGroup({});
  errorForm: FormGroup = new FormGroup({});

  constructor(
    private store: Store<AppState>,
    private fb: FormBuilder,
    private toastService: ToastService) { }
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.3.2. Estado dedicado

Para el desarrollo del estado de cada módulo es importante tomar en cuenta que cada estado es distinto dependiendo del comportamiento de cada componente, por lo que es importante tener claro que se incluirá dentro de cada estado de un módulo, generalmente un estado incluye el listado de elementos a mostrar en pantalla, elementos seleccionado, listado de categorías necesarias para el manejo de elementos que necesitan otros listados y errores provenientes de un servicio externos utilizados y estado de un formulario.

Durante la definición de errores de un estado es importante tomar en cuenta que los errores que se incluyen en el estado deben estar preferiblemente relacionados con errores generados a partir del consumo de servicios de terceros, ya que los errores generados a partir de un formulario pueden ser manejados directamente dentro de un componente aunque puede darse el caso de que el comportamiento del componente sea tan complejo que se vuelve conveniente incluir errores de formularios en el estado de la aplicación para que

permitan manejar errores no solo desde el componente donde ocurre el error sino que también desde un componente padre.

Figura 27. **Ejemplo de creación de estado dedicado**

```
export interface SortNotesState {
    todoNotes: number;
    inProgressNotes: number;
    doneNotes: number;
}

export const initialState: SortNotesState = {
    todoNotes: 0,
    inProgressNotes: 0,
    doneNotes: 0
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.3.2.1. Implementación de acciones

La implementación de acciones en el desarrollo del manejo del estado de una aplicación es algo muy fácil de identificar, ya que cada acción que se implementa en el desarrollo generalmente está ligada a las acciones que se realizan dentro de un componente, estas pueden ir desde la preparación de la información necesaria para generar el comportamiento deseado hasta acciones generadas por el usuario final al modificar la información que se encuentra dentro del estado de la aplicación.

Implementar acciones con NgRx es realmente sencillo, para ello se hace uso de la función createAction de NgRx que recibe como primer parámetro el nombre de la acción que está siendo disparada por nuestro componente y como

segundo parámetro el conjunto de propiedades que nos permiten saber cómo estas acciones modifican el estado de la aplicación desarrollada, estas propiedades son creadas haciendo uso de la función props de NgRx.

Figura 28. **Ejemplo de creación de acciones**

```
export const startLoading = createAction('Loading Started', props<{ loadingName: string }>());
export const stopLoading = createAction('Loading Stopped', props<{ loadingName: string }>());
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 29. **Ejemplo de uso de acciones**

```
ngOnInit(): void {
  this.store.dispatch(loaderActions.startLoading({ loadingName: 'LOAD_CATEGORIES' }));
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.3.2.2. Implementación de reductores

Una vez definido el estado general, los estados dedicados para cada componente y las acciones disparadas por los componentes de la aplicación, se tiene todo lo necesario para realizar la implementación de los reductores que modifican el estado de una aplicación.

Al igual que la implementación de acciones, la implementación de reductores se realiza haciendo uso de funciones de NgRx, la función que permite crear un reductor en NgRx es createReducer la cual recibe como primer parámetro el estado inicial del estado de dicho reductor y posterior al estado

inicial recibe el conjunto de reducciones encargadas de modificar el estado de la aplicación.

La implementación de reducciones para un reductor se compone de dos partes, la primera parte es la acción encargada de disparar dicha reducción, la cual es implementada a través de la importación de acciones y la segunda parte es la función pura encargada de modificar el estado de la aplicación asegurándonos que la modificación de estos datos se realice a través de una sola vía y que esta información no sea editada desde la consola o desde código que no esté relacionado con la ejecución de acciones de un componente.

Figura 30. **Ejemplo de creación de reductor**

```
const stopLoadingReducer = (state: LoaderState, action: any) => {
    const loadings = [...state.loadings];
    const loadingIndex = loadings.findIndex((s) => s === action.loadingName);
    loadings.splice(loadingIndex, 1);
    return {
        ...state,
        loadings
    }
}

export const loaderReducer: ActionReducer<LoaderState, Action> = createReducer(
    initialState,
    on(startLoading, startLoadingReducer),
    on(stopLoading, stopLoadingReducer),
)
```

Fuente: elaboración propia, realizado con Visual Studio Code.

6.3.2.3. Implementación de selectores

Hacer uso de los selectores dentro de una aplicación web es una de las partes más importantes durante el desarrollo de nuestra aplicación, ya que los selectores permiten darle el comportamiento deseado a cada una de las pantallas que se desarrollan en la aplicación.

NgRx ofrece la ventaja de crear selectores que permiten acceder al estado de una aplicación haciendo empleo de Observables que permiten modificar el comportamiento de un componente cada vez que el valor de un observable cambia, esto permite saber cuál es el estado de lo definido en un estado, esto puede ser el estado un formulario, carga de información e ítems cargados., y de esta manera poder ejecutar acciones con base en el estado que el Observable esté emitiendo.

La creación de selectores con NgRx se realiza haciendo empleo de dos funciones de NgRx, la primera función es createFeatureSelector, esta función se encarga de definir un identificador del estado de la aplicación a la cual se desea acceder, es recomendado acompañar esta función con la interfaz del estado al cual se está accediendo, de esta manera al generar selectores el IDE que utilice se encargara de darnos recomendaciones de los selectores que podemos crear. La segunda función es createSelector la cual es la encargada de generar el selector que se exportara a los distintos componentes de nuestra aplicación, esta función recibe como primer parámetro la propiedad generada con la función createFeatureSelector y recibe como segundo parámetro la función encargada de retornar la propiedad a seleccionar en el estado.

El último paso para llevar a cabo la implementación de un selector es importar los selectores necesarios para el desarrollo de un componente y generar un Observable haciendo uso del selector importado, para generar un Observable a partir del selector es necesario inyectar desde nuestro componente una instancia del Store de NgRx y hacer uso de la función Select que nos permite generar un Observable al indicarle el selector deseado para este observable.

Es recomendado hacer uso de los selectores de NgRx, ya que de acceder directamente al estado de la aplicación implicaría en una carga de toda la

información del estado de la aplicación, lo que generaría una carga adicional en el procesamiento de datos, porque se cargaría todo el estado mientras que es muy probable que para generar el comportamiento del componente solo se necesite de unas pocas propiedades del estado de la aplicación.

Figura 31. **Ejemplo de creación de selector**

```
const categoriesState = createFeatureSelector<CategoriesState>('categories');

export const categoriesList = createSelector(categoriesState, (state) => {
    return state.categoriesList;
})

export const errors = createSelector(categoriesState, (state) => state.errors);
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 32. **Ejemplo de uso de selector**

```
categoriesObservable: Observable<Category[]> = this.store.select(categoriesSelectors.categoriesList);
errorsObservable: Observable<IError[]> = this.store.select(categoriesSelectors.errors);
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7. EJEMPLOS DE DESARROLLO DE FUNCIONALIDADES COMUNES DE UNA APLICACIÓN

7.1. Manejo de sesión

Una de las funcionalidades más recurrentes dentro de una aplicación web es el registro de usuarios, inicio de sesión y manejo de una sesión al interactuar con una aplicación. El estado de una aplicación juega un papel muy importante durante el desarrollo de esta funcionalidad, ya que permite realizar acciones como redirigir pantallas de inicio de sesión, pantallas de inicio de una aplicación, hacer uso de tokens de autorización y actualizar tokens de autorización desde cualquier componente sin necesidad de hacer uso de directivas para mantener una comunicación entre componentes.

7.1.1. Estado

Para el manejo de sesiones se debe incluir una serie de propiedades claves para el manejo de la sesión en una aplicación. La primera propiedad es la información del usuario que ha iniciado sesión dentro de la aplicación, de esta forma se podrá realizar acciones como manejo de roles dentro de la aplicación, actualización de información del usuario que ha iniciado sesión y redireccionamiento adecuado de páginas web.

La segunda propiedad es el token de autorización obtenido durante el inicio de sesión o durante el registro de usuario en la aplicación, de esta manera al realizar peticiones HTTP a servicios externos se hará uso del token de autorización más reciente obtenido por parte del servicio externo.

Finalmente, se debe incluir el token de actualización que permitirá mantener el token de autorización almacenado de una manera actualizada en el estado, de esta manera en caso de vencer un token de autorización podremos actualizar el token de autorización en el estado y reprocesar la solicitud HTTP previo a emitir un valor final de la solicitud a procesar.

Tal como se realizó en las propuestas anteriores, este estado también incluirá un arreglo de errores en caso de ocurrir errores durante peticiones a servicios externos, por lo que se podrá reaccionar con base en estos errores.

Figura 33. **Estado propuesto para el manejo de sesiones**

```
export interface UserState {
    loggedUser: Partial<User> | null;
    accessToken: string | null;
    refreshToken: string | null;
    errors: IError[],
}
```

Fuente: elaboración propia, realizado con Git Bash.

7.1.2. Acciones y reducciones

Para el manejo del estado debe considerarse acciones y reducciones que permitan mantener actualizada la información del usuario que ha iniciado sesión y la información de autorización para el manejo de servicios externos. Debe incluirse acciones y reducciones que permitan definir quien ha iniciado sesión, cuál es el token de autorización para el manejo de servicios externos y cuál es el token de autorización para realizar la actualización del token de autorización.

Para lograr la actualización de toda esta información se debe realizar acciones que informen a los reductores que debe realizarse la actualización de cada uno de estos atributos, por lo que estas acciones deben de incluir la información necesaria para esa actualización.

Al igual que las acciones, es necesario desarrollar funciones puras que actualicen la propiedad afectada por dicha acción y que sea capaz de reemplazar la información que se encuentra en el estado por la información proporcionada a partir de las acciones disparadas por los componentes correspondientes.

Figura 34. **Ejemplo de acciones y reductores para manejo de sesión**

```
export const loggedUserUpdated = createAction('LogedUserUpdated', props<{ loggedUser: Partial<User> | null }>());
export const accessTokenUpdated = createAction('AccessTokenUpdated', props<{ accessToken: string | null }>());
const loggedUserUpdatedReducer = (state: UserState, action: any) => {
    return {
        ...state,
        loggedUser: action.loggedUser
    }
}
const accessTokenUpdatedReducer = (state: UserState, action: any) => {
    return {
        ...state,
        accessToken: action.accessToken
    }
}
export const userReducer: ActionReducer<UserState, Action> = createReducer(
    initialState,
    on(loggedUserUpdated, loggedUserUpdatedReducer),
    on(accessTokenUpdated, accessTokenUpdatedReducer),
    on(refreshTokenUpdated, refreshAccessTokenUpdatedReducer),
    on(saveApiError, saveApiErrorReducer),
    on(resetApiErrors, resetErrorsReducer),
    on(userImageUpdated, userImageUpdatedReducer)
)
```

Fuente: elaboración propia, realizado con Git Visual Studio Code.

7.1.3. Sesión existente y manejo de sesión

Mantener sesiones activas dentro de una aplicación es otra funcionalidad muy recurrente durante el desarrollo de aplicaciones. El Store de NgRx permite almacenar información de la aplicación siempre y cuando la aplicación permanezca abierta, por lo que es necesario inicializar el Store con la información de la sesión en caso de existir una sesión existente.

Para inicializar el estado con una sesión preexistente se propone hacer uso del almacenamiento local del navegador, de esta forma desde el componente inicial de la aplicación se podrá verificar en este almacenamiento si hay una sesión preexistente para inicializar el estado o si, por el contrario, debe mantener su valor inicial. Se propone que para definir si existe una sesión preexiste se deba contar como mínimo con el token de autorización, token de actualización e información de usuario, de lo contrario el almacenamiento local debe ser limpiado y el estado debe permanecer con el valor inicial.

Figura 35. **Ejemplo de manejo de sesión preexistente**

```
ngOnInit(): void {
  const userCredentials = this.userService.getUserCredentials();
  if (userCredentials) {
    this.store.dispatch(userActions.loggedUserUpdated({ loggedUser: userCredentials.user }));
    this.store.dispatch(userActions.accessTokenUpdated({ accessToken: userCredentials.accessToken }));
    this.store.dispatch(userActions.refreshTokenUpdated({ refreshToken: userCredentials.refreshToken }));
    this.subscriptions.push(this.userService.getUserImage(userCredentials.user._id || '').subscribe((image) => {
      this.store.dispatch(userActions.userImageUpdated({ image: image.image }));
    }));
  }
```

Fuente: elaboración propia, realizado con Visual Studio Code.

En caso de no existir una sesión preexistente se debe manejar desde el componente de inicio de sesión la lógica de almacenamiento de una nueva sesión, para ello se hace uso del almacenamiento local del navegador, estado, acciones y reducciones creadas previamente, de esta forma se asegura lograr

una persistencia de la nueva sesión creada. Para almacenar la información relacionada con la sesión debemos realizar una solicitud al servicio externo correspondiente, en caso de autenticarse correctamente podemos proceder con el almacenamiento de la sesión y en caso contrario con la presentación del error correspondiente.

Figura 36. **Ejemplo de inicio de sesión**

```
this.subscriptions.push(this.userService.login(userBody).subscribe((response) => {
  this.store.dispatch(userActions.accessTokenUpdated({ accessToken: response.accessToken }));
  this.store.dispatch(userActions.refreshTokenUpdated({ refreshToken: response.refreshToken }));
  this.store.dispatch(userActions.loggedUserUpdated({ loggedUser: response.user }));
  localStorage.setItem('loggedUser', JSON.stringify(response.user));
  localStorage.setItem('accessToken', response.accessToken);
  localStorage.setItem('refreshToken', response.refreshToken);
  this.store.dispatch(loaderActions.stopLoading({ loadingName: 'START_LOGIN' }));
  this.subscriptions.push(this.userService.getUserImage(response.user._id || '').subscribe((image) => {
    this.store.dispatch(userActions.userImageUpdated({ image: image.image }));
    this.router.navigate(['specific']);
  }));
}, (error) => {
  this.store.dispatch(userActions.saveApiError({ error: { type: 'POST', messages: error.error.messages } }));
  this.store.dispatch(loaderActions.stopLoading({ loadingName: 'START_LOGIN' }));
  this.toastService.show('Ocurrió un error al realizar tu solicitud', { classname: 'bg-danger text-light', del
}));
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 37. **Ejemplo de cierre de sesión**

```
logout(): void {
  this.store.dispatch(userActions.accessTokenUpdated({ accessToken: null }));
  this.store.dispatch(userActions.refreshTokenUpdated({ refreshToken: null }));
  this.store.dispatch(userActions.loggedUserUpdated({ loggedUser: null }));
  localStorage.removeItem('loggedUser');
  localStorage.removeItem('accessToken');
  localStorage.removeItem('refreshToken');
  this.router.navigate(['']);
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.1.4. Acceso a pantallas restringidas

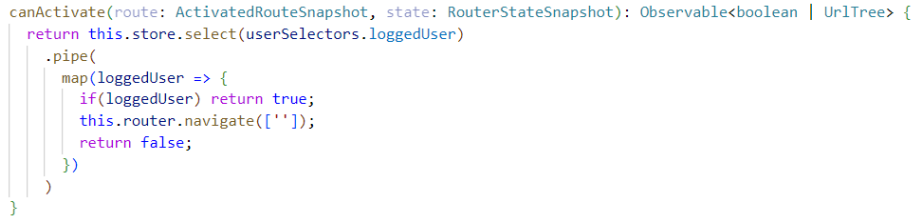
Denegar el acceso a pantallas previo o posterior a tener una sesión activa es una funcionalidad de gran importancia, ya que de no desarrollarse correctamente podría generar el remplazo de sesiones activas o acceso a pantallas que requieren de un usuario para poder utilizarse. Al igual que con el manejo de sesiones, el Store de NgRx apoya de gran manera durante este desarrollo, porque proporciona la información de la sesión activa en la aplicación.

7.1.4.1. Acceso a pantallas con sesión requerida

Para denegar el acceso a pantallas con autenticación requerida se propone hacer uso de Guards de Angular. Para el desarrollo del Guard se recomienda hacer empleo de canActivate de Angular que a través de un observable de tipo booleano permite definir si podemos a la página deseada en donde el Guard haya sido utilizado durante la definición del redireccionamiento de las páginas.

El desarrollo de este Guard se vuelve bastante sencillo al únicamente tener que transformar el observable de la información del usuario que se encuentra en el Store por lo que al realizar una transformación con el operador map de rxjs podemos transformar el usuario que se encuentra en el Store en una respuesta verdadera que permita acceder a dichas pantallas o una respuesta falsa en caso de que no exista ninguna sesión.

Figura 38. **Ejemplo de Guard para acceso a pantallas con sesión requerida**



```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> {
  return this.store.select(userSelectors.loggedUser)
    .pipe(
      map(loggedUser => {
        if(loggedUser) return true;
        this.router.navigate(['']);
        return false;
      })
    )
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Finalmente, posterior al desarrollo del Guard es necesario incluir este Guard en el archivo de redireccionamiento de páginas. Basándose en la estructura propuesta basta con incluir este Guard en el redireccionamiento del componente de categorías específicas, ya que de no poder acceder a este componente no se puede acceder a ninguno de los componentes hijos de este módulo.

Figura 39. **Ejemplo de uso de Guard en archivo de redireccionamiento de paginas**

```
const routes: Routes = [
  {
    path: '',
    component: SpecificComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: 'categories',
        loadChildren: () => import('./categories/categories.module').then((m) => m.CategoriesModule),
      },
```

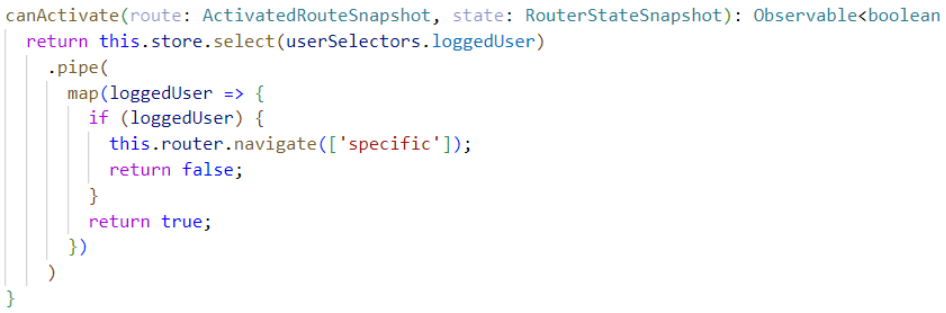
Fuente: elaboración propia, realizado con Visual Studio Code.

7.1.4.2. Acceso a pantallas sin sesión requerida

El desarrollo de acceso a pantallas sin una sesión requerida es bastante similar al Guard propuesto previamente, la variación respecto al Guard anterior radica en que el resultado del observable es opuesto al anterior, por lo que si existe una sesión no se podrá acceder a pantallas que no requieren de una sesión y se redirigirá a la pantalla de inicio de la aplicación.

Para la estructura propuesta, este Guard se utilizaría únicamente en el componente de inicio de sesión, ya que es la única pantalla que no requiere de un inicio de sesión previo para poder acceder a ella.

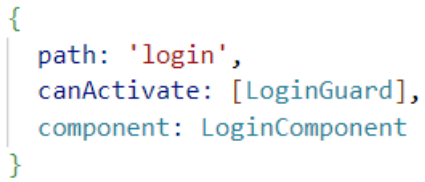
Figura 40. **Ejemplo de Guard para acceso a pantallas sin sesión requerida**



```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean
  return this.store.select(userSelectors.loggedUser)
    .pipe(
      map(loggedUser => {
        if (loggedUser) {
          this.router.navigate(['specific']);
          return false;
        }
        return true;
      })
    )
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 41. **Ejemplo de uso de Guard sin sesión requerida en archivos de redireccionamiento de paginas**



Fuente: elaboración propia, realizado con Visual Studio Code.

7.1.5. Manejo de tokens de autorización en peticiones HTTP

Manejar tokens de autorización en peticiones HTTP es otra funcionalidad muy importante durante el manejo de sesiones en una aplicación, al igual que con el manejo de pantallas restringidas, podemos aprovechar tanto el Store de NgRx con la información de la sesión almacenada como también aprovechar herramientas de Angular para lograr el desarrollo de esta funcionalidad.

Para desarrollar esta funcionalidad se propone hacer uso de Interceptores de solicitudes HTTP de Angular, de esta forma podremos manipular todas las solicitudes HTTP previo a procesarlas y de esta manera evitar la manipulación de todas las funciones encargadas de realizar peticiones a servicios externos. Para este caso se plantea manipular el observable de la sesión activa tal como se realizó durante el desarrollo del Guard, con la diferencia que en lugar de retornar un valor verdadero o falso retornara una solicitud HTTP con los encabezados del token de autorización almacenado en la sesión activa, de esta manera aseguramos que todas las solicitudes posean el token de autorización necesario para procesar la solicitud.

Figura 42. **Ejemplo de interceptor que agrega token de autorización a peticiones HTTP**

```
return this.store.select(userSelectors.accessToken)
  .pipe(
    take(1),
    concatMap(accessToken => {
      if (accessToken && request.url.startsWith(environment.apiUrl)) {
        request = request.clone({
          setHeaders: {
            Authorization: `Bearer ${accessToken}`
          }
        })
      }
      return next.handle(request);
    })
  );
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Este interceptor debe ser incluido en los proveedores del módulo principal de la aplicación, ya que de esta manera se logra todas las peticiones de servicios externos sean interceptadas por el módulo principal y no por submódulos de la aplicación.

Figura 43. **Ejemplo de uso de interceptor en módulo principal**

```
providers: [
  { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
  { provide: HTTP_INTERCEPTORS, useClass: RefreshTokenInterceptor, multi: true }
],
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.1.6. Actualización de token de autorización

Desarrollar la actualización de un token de autorización es una funcionalidad muy similar al manejo de tokens en peticiones HTTP, la diferencia radica en que esta tarea ocurre posterior a realizar una petición HTTP, ya que necesitamos una respuesta del servicio externo para saber si el token de autorización debe ser actualizado y, por tanto, la solicitud debe ser reprocesada.

Para el desarrollo de esta funcionalidad también se propone hacer uso de interceptores que se encarguen de transformar la respuesta del servicio externo previo a emitir un valor para el componente final, en este caso es necesario realizar la solicitud a un servicio externo y verificar si en la respuesta existe un error y verificar el tipo de error, en caso de que el error corresponda a la expiración de un token de autorización se procede a realizar la actualización del token de autorización y reprocesar la solicitud HTTP con el nuevo token de autorización, de esta manera el componente final recibirá la respuesta adecuada a la solicitud del servicio externo en caso de no ocurrir un error distinto al error de un token de autorización vencido.

Figura 44. **Ejemplo de interceptor de tokens vencidos**

```
intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
  return next.handle(request).pipe(
    catchError((err: any) => {
      if (err.error.message === 'JWT_EXPIRED') {
        return this.refreshToken(request, next);
      }
      else {
        return throwError(err);
      }
    })
  );
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 45. **Ejemplo de actualización de token y reprocesamiento de solicitud**

```
refreshToken(request: HttpRequest<unknown>, next: HttpHandler) {
  return this.store.select(userSelectors.refreshToken)
    .pipe(
      take(1),
      concatMap(refreshToken => {
        return this.authService.refreshToken({ refreshToken }).pipe(
          concatMap((refreshResponse: any) => {
            this.store.dispatch(userActions.accessTokenUpdated({ accessToken: refreshResponse.accessToken }));
            localStorage.setItem('accessToken', refreshResponse.accessToken)
            request = request.clone({
              setHeaders: {
                Authorization: `Bearer ${refreshResponse.accessToken}`
              }
            });
            return next.handle(request);
          })
        )
      })
    );
}
```

Fuente: elaboración propia, realizado con Git Bash.

7.2. Carga inicial de datos

Una de las primeras subtareas que se realiza durante el desarrollo pantallas para las funcionalidades de una aplicación es el proceso de la carga inicial de datos necesaria para mostrar de forma correcta la pantalla de la aplicación.

Se propone realizar esta carga de forma independiente en cada uno de los submódulos específicos de la aplicación, de esta forma aseguramos que las peticiones a servicios externos se realicen cuando sean necesarias y que los componentes que hagan uso de esta información reaccionen de forma adecuada en cada cambio.

7.2.1. Estado

Para el desarrollo del estado se propone incluir todas las propiedades necesarias para mostrar el listado de elementos relacionados con el módulo que se está tratando y posteriormente agregar las propiedades relacionadas a funcionalidades que no estén involucradas con el listado de elementos relacionados con el módulo.

Tomando como referencia un módulo relacionado con categorías en el estado, solo se necesitaría un arreglo de categorías para mostrar el listado de categorías y el resto de las propiedades serian manejadas durante el desarrollo de manejo de errores o edición de elementos.

Es importante que la propiedad este correctamente asociado el tipo de dato creado con las interfaces, de esta forma se asegurara una correcta continuidad en el desarrollo al saber cuáles son las propiedades del listado y en caso de verse modificado sea fácil identificar la lógica que debe ser modificada en el desarrollo de la solución.

Figura 46. **Ejemplo de listado de elementos en un módulo**

```
export interface CategoriesState {
    categoriesList: Category[],
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.2.1.1. Acciones y reducciones

Para el desarrollo de la carga de elementos a mostrar en la página principal de un módulo basta con crear una reducción capaz de actualizar el valor del listado del módulo, de esta forma se podrá almacenar los elementos obtenidos a partir de la consulta a servicios externos.

Es importante que la acción encargada de emitir el evento de cambio en los elementos del listado tenga el mismo tipo asociado en el listado del estado del módulo, de esta manera al realizar la asignación del estado del reductor se asegurara que el tipo de datos sea el mismo y no se ingrese información no adecuada al listado de datos.

Figura 47. **Ejemplo de acciones y reducciones para carga inicial de elementos**

```
export const loadCategories = createAction('Categories Loaded', props<{ categories: Category[] }>());

const loadCategoriesReducer = (state: CategoriesState, action: any) => {
    return {
        ...state,
        categoriesList: action.categories
    }
}

export const categoriesReducer: ActionReducer<CategoriesState, Action> = createReducer(
    initialState,
    on(loadCategories, loadCategoriesReducer),
);
```

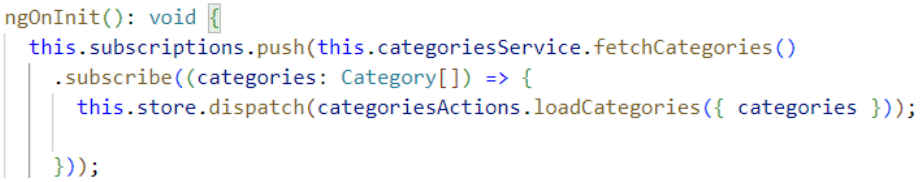
Fuente: elaboración propia, realizado con Visual Studio Code.

7.2.2. Carga de información de elementos en estado

Para realizar la carga de elementos dentro de un estado es necesario hacer uso del servicio externo realizado para el tipo de elemento que se esté cargando y suscribirse a la respuesta emitida por el servicio, en caso de que no

ocurra un error se haría uso de la acción creada para realizar la carga de la información y almacenaría la respuesta recibida por el servicio externo con el mismo tipo creado previamente para el tipo de datos.

Figura 48. **Ejemplo de carga de elementos obtenidos de un servicio externo**



Fuente: elaboración propia, realizado con Visual Studio Code.

7.2.3. Uso de elementos en estado a través de un selector

Hacer uso del listado de elementos guardados en un estado es realmente sencillo, basta con hacer uso del selector creado para la propiedad de elementos y manipular la información desde el archivo TypeScript o HTML. Existen dos maneras totalmente válidas de manejar el observable de datos del estado de la aplicación.

La primera forma propuesta de hacer uso del listado de elementos en el estado de la aplicación es a través de la transformación de datos asíncronos ofrece Angular, con esta forma basta con utilizar directivas cíclicas que recorran el listado de elementos al tener una respuesta asíncrona y mostrar los datos en la página web deseada.

Figura 49. **Ejemplo de uso de listado en componente**

```
<div class="card mb-3 p-4" *ngFor="let category of categoriesObservable | async">
    <div class="d-flex flex-row align-items-center">
        <div class="me-5 no-display">
            <img [src]="category.image" alt="" width="90" height="90" class="rounded-circle me-2"
            loadImgError="assets/image-error.png">
        </div>
        <div class="me-auto">
            <div class="ellipsis">
                <b>Nombre: </b> {{category.name}}
            </div>
            <div class="ellipsis">
                <b>Descripción: </b>{{category.description}}
            </div>
        </div>
        <div class="d-flex flex-sm-row flex-column">
            <button class="btn btn-link text-decoration-none text-primary me-5 me-i" (click)="editCategory(category.
                <i class="bi bi-pencil me-2"></i> Editar
            </button>
            <button class="btn btn-link text-decoration-none text-danger me-4 me-i" (click)="deleteCategory(category
                <i class="bi bi-trash me-2"></i> Eliminar
            </button>
        </div>
    </div>
</div>
```

Fuente: elaboración propia, realizado con Visual Studio Code.

La segunda forma de hacer uso del listado de elementos es a través de la creación de una variable que contenga el listado de elementos, de esta forma podremos suscribirnos al selector de elementos y al tener una respuesta podremos asignarla a la variable que se utilizara para mostrar los datos en pantalla, esta forma ofrece la ventaja adicional de poder manipular los datos previo a mostrarlos en caso de ser necesario.

7.3. Estado de un formulario

Otra funcionalidad recurrente durante el desarrollo de aplicaciones web es el manejo de formularios, es importante brindarle al usuario final información adicional y advertencias de ciertas acciones que implican perdida de información en formularios que no ha sido guardada previo a que el usuario realice la acción.

Para esto se propone el desarrollo de un estado encargado de almacenar el estado inicial y actual de cualquier formulario, de esta manera se puede

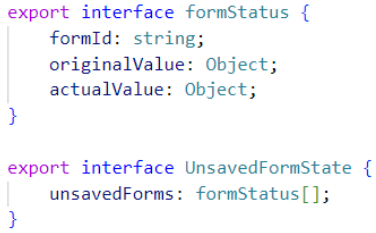
implementar el desarrollo de pantallas de advertencia a formularios no guardados.

7.3.1. Estado

El desarrollo de este estado a diferencia de la mayoría de los estados no debe estar dentro el directorio de algún modulo, esto se debe a que este estado no tiene relación directa con ningún modulo por lo que basta con la creación de un directorio para el manejo de archivos de estado, acciones, reductores y selectores. Este directorio puede ser creado junto con los submódulos de categoría especifica ya que si bien no es considerado un submódulo si es considerado como una funcionalidad especifica.

Para el desarrollo de este estado se propone la creación de una interfaz encargada del manejo de cada formulario, esta interfaz tendría como propiedades un identificador único para formularios, el valor inicial del formulario y el valor final del formulario. A partir de esta interfaz el estado puede creado con una propiedad de formularios no guardados con un arreglo de formularios sin guardar con la interfaz creada previamente lo que permitirá poder saber el estado de cada uno de los formularios.

Figura 50. **Ejemplo de estado de formularios sin guardar**



Fuente: elaboración propia, realizado con Visual Studio Code.

7.3.1.1. Acciones y reductores

Para el desarrollo de esta funcionalidad es necesario la creación de tres acciones y reductores. Las acciones y reductores estarán orientadas a tres funcionalidades específicas, la primera funcionalidad está encargada de inicializar el valor de un formulario, por lo que a partir del identificador y valor del formulario el estado es modificado agregando un nuevo elemento al arreglo del estado con el identificador y valores proveídos.

La segunda funcionalidad está orientada al cambio de valor en un formulario, en esta funcionalidad la acción tendrá la propiedad con el identificador asignado al formulario y el valor actual del formulario, de esta manera la reducción modificará el valor de un formulario basado en el identificador del formulario creado.

Finalmente, se tendrá una acción sin propiedades encargada de realizar la limpieza en el arreglo de los estados de formulario para evitar comportamientos no deseados en las advertencias de la aplicación.

Figura 51. **Ejemplo de acciones y reductores para el manejo de formularios sin guardar**

```
export const formInitialized = createAction('Form Value Initialized', props<{ formId: string, value: Object }>());
export const formValueChanged = createAction('Form Value Updated', props<{ formId: string, value: Object }>());
export const unsavedFormsCleaned = createAction('Unsaved Forms Cleaned');

const formInitializedReducer = (state: UnsavedFormState, action: any) => {
    const newFormValue: formStatus = {
        formId: action.formId,
        originalValue: {...action.value},
        actualValue: {...action.value}
    }
    const forms = [...state.unsavedForms, newFormValue];
    return {
        ...state,
        unsavedForms: forms
    }
}

export const unsavedFormsReducers: ActionReducer<UnsavedFormState, Action> = createReducer(
    initialState,
    on(formInitialized, formInitializedReducer),
    on(formValueChanged, formValueChangedReducer),
    on(unsavedFormsCleaned, unsavedFormsCleanedReducer),
)
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.3.2. Advertencia de formulario sin guardar

Para el desarrollo de advertencias de formularios sin guardar previo a cambiar de páginas dentro de la aplicación se propone hacer uso de Guards como se realizó durante el desarrollo del acceso a pantallas con sesión requerida, la variación de este desarrollo estará en que en lugar de retornar un valor verdadero o falso se retornará un observable con la respuesta que decida el usuario a través de un modal de advertencia.

El desarrollo se realizará a través de la manipulación del observable del selector del estado de formularios, en él se recorrerá el arreglo de formularios y se verificara si existe algún formulario cuyo valor actual sea distinto al valor inicial, en caso de serlo se desplegará el modal de advertencia y se retornara un

observable con la respuesta seleccionada por el usuario para confirmar si el cambio de página es autorizado o no.

Figura 52. **Ejemplo de Guard para formularios sin guardar**

```
canActivate(
  route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
  return this.store.select(unsavedFormsSelectors.unsavedForms).pipe(
    concatMap((unsaved) => {
      if (!unsaved) {
        const modalRef = this.modalService.open(UnsavedFormConfirmationComponent, { centered: true });
        modalRef.componentInstance.title = 'Cambios sin guardar';
        modalRef.componentInstance.message = '¿Estás seguro de que deseas salir de esta página? Aún tienes cambios
        return modalRef.result;
      }
      return of(unsaved);
    })
  )
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.3.3. Uso de Guard para modal de advertencias en archivos de redireccionamiento de páginas

Al igual que con el manejo en el acceso a pantallas con sesión requerida o no requerida, es necesario incluir el nuevo Guard de formularios sin guardar en todos los componentes a donde el usuario pueda redirigirse a partir del formulario que está siendo editado, generalmente este Guard es incluido en todas las pantallas de listado de elementos de un módulo.

Figura 53. **Ejemplo de uso de Guard de formulario sin guardar en archivos de redireccionamiento**

```
const routes: Routes = [
  {
    path: '',
    canActivate: [UnsavedFormsGuard],
    component: CategoriesComponent
  },
  {
    path: 'create',
    canActivate: [UnsavedFormsGuard],
    component: CreateCategoryComponent
  },
  {
    path: ':id',
    canActivate: [UnsavedFormsGuard],
    component: EditCategoryComponent
  }
];
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.4. Lazy Loading

La mayoría de las veces que se utiliza un servicio externo cierto tiempo de espera antes de recibir una respuesta de un servicio externo. Este tiempo de espera puede variar desde la cantidad de procesamiento que el servicio externo deba realizar hasta la velocidad de conexión que se tenga en dicho momento. Por esta razón es muy importante que el usuario final esté informado de que existe una espera de respuesta por parte de un servicio externo y que debe esperar que dicha respuesta sea recibida para poder proceder con cualquier otra acción.

7.4.1. Estado

La propuesta del estado para tener control sobre una pantalla de carga es realmente simple, basta con la creación de un arreglo de tipo cadenas donde se encuentren todos los identificadores de las cargas que se encuentran en progreso, manejar el estado, de esta forma se asegura de que en el caso de que se realicen múltiples cargas al mismo tiempo, al momento de detenerlas estemos deteniendo una carga que ya ha finalizado.

Si al contrario de usar un arreglo se hace uso de una variable cuyo valor fuera verdadero o falso, tendríamos el problema de pantallas de carga cuyo comportamiento no sería adecuado, podría ocurrir un escenario en donde se inician dos pantallas de carga y una pantalla finaliza mucho antes de la segunda pantalla de carga lo que ocasionaría que la pantalla no se muestre, aunque todas las cargas aún no hayan finalizado, por esta razón se propone el uso de un arreglo con identificadores de carga.

Figura 54. **Ejemplo de estado para manejo de pantallas de carga**

```
export interface LoaderState {
    loadings: string[];
}

export const initialState: LoaderState = {
    loadings: []
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.4.1.1. Acciones y reducciones

Al igual que con el estado, las acciones y reductores propuestos para el manejo del estado son sencillos, basta con una acción y reductor para iniciar una pantalla de carga y una acción y reductor para detener una pantalla de carga, es importante volver a recordar que estas acciones siempre tendrán dentro de sus propiedades el identificador único de la pantalla de carga, de esta forma en los reductores se podrá saber cuál es el elemento a eliminar dentro del arreglo de pantallas de carga en curso.

Figura 55. **Ejemplo de acciones y reductores para el manejo de pantallas de carga**

```
export const startLoading = createAction('Loading Started', props<{ loadingName: string }>());
export const stopLoading = createAction('Loading Stopped', props<{ loadingName: string }>());
const startLoadingReducer = (state: LoaderState, action: any) => {
    const loadings = [...state.loadings, action.loadingName];
    return {
        ...state,
        loadings
    }
}
const stopLoadingReducer = (state: LoaderState, action: any) => {
    const loadings = [...state.loadings];
    const loadingIndex = loadings.findIndex((s) => s === action.loadingName);
    loadings.splice(loadingIndex, 1);
    return {
        ...state,
        loadings
    }
}
export const loaderReducer: ActionReducer<LoaderState, Action> = createReducer(
    initialState,
    on(startLoading, startLoadingReducer),
    on(stopLoading, stopLoadingReducer),
)
```

Fuente: elaboración propia, realizado con Git Bash.

7.4.1.2. Selector

Para el manejo de pantallas de carga solamente es necesario el uso de un único selector encargado de hacer saber si hay alguna carga en progreso o no. A diferencia de la mayoría de los selectores en donde simplemente se emite el valor de la propiedad del estado, este selector debe ser transformado a un valor de verdadero y falso para indicar si la pantalla de carga debe mostrarse.

Figura 56. **Ejemplo de selector para el manejo de pantallas de carga**

```
export const loadings = createSelector(loaderState, (state) => {
    return state.loadings.length > 0 ? true : false;
});
```

Fuente: elaboración propia, realizado con Git Bash.

7.4.2. Manejo de pantalla de carga

Realizar el control en el manejo de una pantalla de carga es una tarea mucho más sencilla, basta con ejecutar la acción con el inicio de una pantalla de carga previo al uso de un servicio externo y ejecutar el fin de la pantalla de carga al recibir la respuesta por parte del servicio externo. También es importante ejecutar la acción de fin de la pantalla de carga al ocurrir un error en la respuesta del servicio externo, de esta forma aseguramos de que la pantalla de carga no permanezca congelada, aunque ya se posea una respuesta.

Figura 57. **Ejemplo de inicio y fin de pantallas de carga**

```
ngOnInit(): void {
  this.store.dispatch(loaderActions.startLoading({ loadingName: 'LOAD_CATEGORIES' }));
  this.subscriptions.push(this.categoriesService.fetchCategories()
    .subscribe((categories: Category[]) => {
      this.store.dispatch(categoriesActions.loadCategories({ categories }));
      this.store.dispatch(loaderActions.stopLoading({ loadingName: 'LOAD_CATEGORIES' }));
    }, (error) => {
      this.store.dispatch(loaderActions.stopLoading({ loadingName: 'LOAD_CATEGORIES' }));

    }));
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Una vez en el estado se encuentre las pantallas de carga activas, basta con hacer uso del selector de pantallas activas como condicional para cubrir toda la pantalla con una animación de carga que cubra el componente principal de toda la aplicación.

Figura 58. **Ejemplo de uso de reductor de pantallas de cargas activas**

```
<div class="overlay" *ngIf="overlayEnabled"></div>
<router-outlet></router-outlet>
<div class="spinner-border spinner text-primary" role="status" *ngIf="overlayEnabled"></div>
<my-notes-app-toasts aria-live="polite" aria-atomic="true"></my-notes-app-toasts>
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.5. Manejo de errores

Finalmente, otra de las funcionalidades más recurrentes en aplicaciones reactivas es el manejo de errores, es importante que el usuario que hace uso de la aplicación siempre este al tanto de los errores que ocurren en la aplicación. Generalmente gran parte de los errores pueden ser tratados directamente desde

la aplicación a través de validaciones en valores de formularios, lo cual es sencillo de desarrollar.

Por el otro lado, existen errores que no pueden ser tratados directamente desde la aplicación, ya que dependen de respuestas de servicios externos, ejemplos pueden ser la repitencia en el nombre de un elemento, la denegación de eliminación de un elemento o cualquier otro tipo de error generado desde la lógica de negocios de un servicio externo. Estos errores tienden a ser definidos con códigos de error proveídos por parte de servicios externos para que puedan ser tratados desde la aplicación web, por lo que hacer uso del estado para almacenar estos errores es una gran idea para asegurar que las advertencias necesarias para estos errores sean mostradas cada vez que este error exista en el estado de la aplicación.

7.5.1. Estado

Tal como se propuso durante la organización del estado específico de una aplicación, es necesario incluir un arreglo de errores ocurridos dentro de la aplicación, para esto se propone la creación de una interfaz en donde se almacene el tipo de petición HTTP realizada y un arreglo de códigos de error generados posterior al procesamiento de la petición HTTP.

Este arreglo de errores debe ser incluido en todos los estados en donde se realicen peticiones HTTP, de esta manera los componentes que se suscriban a estos errores podrán mostrar las advertencias necesarias relacionadas con el error generado.

Figura 59. **Ejemplo de estado de errores en módulo de aplicación**

```
export interface IError {
    type: string;
    messages: string[];
}

export interface CategoriesState {
    categoriesList: Category[],
    errors: IError[],
}

export const initialState: CategoriesState = {
    categoriesList: [],
    errors: [],
}
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.5.1.1. Acciones y reducciones

Al igual que el arreglo de errores dentro de los estados de una aplicación, es necesario incluir una serie de acciones y reducciones en la mayoría de los estados de la aplicación para asegurar un fácil manejo de errores ocurridos durante el consumo de servicios externos.

Las dos acciones y reducciones necesarias para el manejo de errores son el guardado de mensajes de error de un servicio externo y la limpieza del arreglo de errores dentro de la aplicación, de esta forma se podrá guardar errores cuando ocurran y limpiar los errores cuando un formulario es editado o cuando se cambia de página en la aplicación.

Figura 60. **Ejemplo de acciones y reducciones para el manejo de errores**

```
export const saveApiError = createAction('Category Error Ocurred', props<{ error: IError }>());
export const resetApiErrors = createAction('Note Error Reset');
const saveApiErrorReducer = (state: CategoriesState, action: any) => {
    return {
        ...state,
        errors: [...state.errors, action.error],
    }
}
const resetErrorsReducer = (state: CategoriesState, action: any) => {
    return {
        ...state,
        errors: [],
    }
}

export const categoriesReducer: ActionReducer<CategoriesState, Action> = createReducer(
    initialState,
    on(saveApiError, saveApiErrorReducer),
    on(resetApiErrors, resetErrorsReducer),
);
```

Fuente: elaboración propia, realizado con Visual Studio Code.

7.5.2. Manejo de advertencia de errores

Teniendo en cuenta que en el estado de la aplicación se encontraran arreglos con todos los errores ocurridos, se propone crear una variable por cada tipo de error a ocurrir en la aplicación, de esta forma se podrá mostrar advertencias distintas para cada tipo de error.

La asignación del valor de la variable estará definida por la suscripción al selector del arreglo de errores deseado, se recomienda hacer uso del operador para manejo de arreglos some para identificar si alguno de los errores posee el código de error establecido por el servicio externo utilizado. La limpieza del valor de esta variable es innecesaria, ya que una vez se limpie los errores, la suscripción será la encargada de desactivar el valor de la variable de error al no encontrar ningún error con el código especificado.

Por último, dentro de la plantilla HTML se hará uso de dichas variables junto con la directiva estructural NgIf para mostrar en pantalla el error deseado cada vez que este sea asignado en el estado de la aplicación.

Figura 61. **Ejemplo de asignación de ocurrencia de error en variables**

```
this.subscriptions.push(
  this.errorsObservable.subscribe((errors) => {
    this.duplicatedCategory = errors.some((e) => e.messages.some((e2) => e2 === 'DUPLICATED_CATEGORY'));
}));
```

Fuente: elaboración propia, realizado con Visual Studio Code.

Figura 62. **Ejemplo de uso de directivas y variables para mostrar errores**

```
<div *ngIf="duplicatedCategory" class="text-danger error-size2 mb-2">
    La categoría ya existe
</div>
```

Fuente: elaboración propia, realizado con Visual Studio Code.

CONCLUSIONES

1. De acuerdo con las estadísticas y análisis expuestos, Angular es un *framework* de desarrollo frontend estable y robusto, capaz ofrecer una gran cantidad de herramientas de desarrollo, lo que lo hace un *framework* de desarrollo frontend ideal para acelerar el desarrollo de software respecto a *frameworks* que se limitan a ofrecer herramientas dedicadas al manejo de interfaces de usuario.

2. Con base en la compatibilidad de la librería de NgRx con Angular y la librería RxJS, se concluye que NgRx es la librería ideal para el manejo del estado general de una aplicación frontend.

3. La orientación del estado de una aplicación, los errores y hacia el estado actual de cada componente permite manejar de forma sencilla y asíncrona peticiones y errores derivados por el uso de servicios de terceros, lo que mejor la experiencia del usuario al estar siempre informado de lo que sucede en la aplicación.

RECOMENDACIONES

1. Establecer cuáles serán las necesidades esenciales para el desarrollo de un nuevo proyecto frontend, ya que la selección del *framework* de desarrollo frontend dependen de la compatibilidad entre las necesidades del proyecto y la orientación ofrecida por el *framework* de desarrollo.

2. Poner en claro las necesidades de un nuevo proyecto frontend y hacer uso de Angular, ya que adicional a ofrecer una gran cantidad de herramientas de desarrollo, permite la modificación de estas herramientas durante la marcha.

3. Orientar la librería para el manejo del estado de una aplicación hacia el *framework* de desarrollo frontend a utilizar ya que, estas librerías ofrecen compatibilidad con el *framework* utilizado sin necesidad de librerías externas.

4. Manejar errores asíncronos a través del estado general de la aplicación, ya que esto permite que los componentes se comporten de forma reactiva basándose en el estado de los errores.

REFERENCIAS

1. Angular. (s. f.). *Introduction to the Angular Docs.* Introduction to the Angular Docs. Recuperado de https://angular.io/docs.

2. Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. (2014, septiembre). *El Manifesto de Sistemas Reactivos*. El Manifesto de Sistemas Reactivos. Recuperado de https://www.reactivemanifesto.org/es.

3. MobX. (s. f.). *MobX Documentation,* MobX Documentation. Recuperado de https://mobx.js.org/README.html.

4. Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda. (2017). *ng-book 2*. San Francisco, California: Fullstack.io.

5. NgRx. (s. f.). *NgRx Docs*. NgRx Docs. Recuperado de https://ngrx.io/docs.

6. React. (s. f.). *React Docs*. React Docs. Recuperado de https://ngrx.io/docs.

7. Redux. (s. f.). *Redux Documentation*. Redux Documentation. Recuperado de https://es.redux.js.org/docs/.

8. Rossi, G., Pastor, O., Schwabe, D., & Olsina. (2008). Web engineering: modelling and implementing web applications. Springer Science & Business Media.

9. Vue.js. (s. f.). *Introduction Vue.js.* Introduction Vue.js. Recuperado de https://v3.vuejs.org/guide/introduction.html.

APÉNDICES

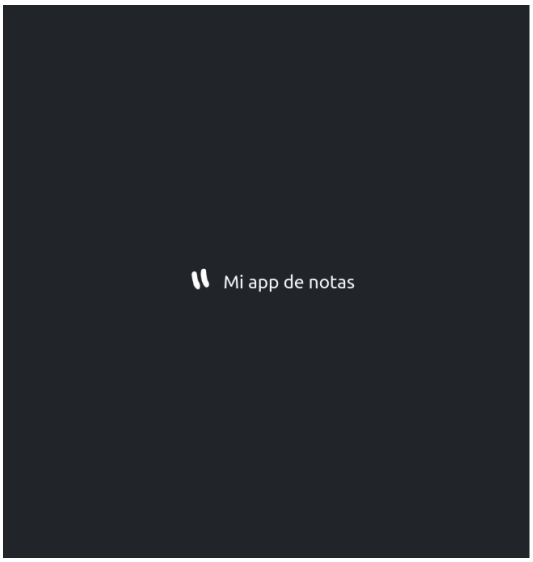
Apéndice 1. **Manual de usuario de aplicación de ejemplo desarrollada con la propuesta planteada**

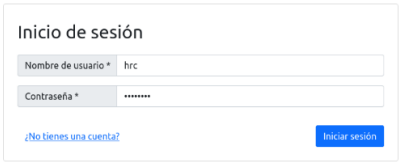
La aplicación de ejemplo se encuentra desarrollada tanto para dispositivos móviles y de escritorio. El desarrollo de la aplicación de ejemplo está basado en la propuesta realizada en la investigación y cuenta con una sección de resumen en donde se puede visualizar y realizar pruebas de uso del estado de la aplicación desarrollada.

- Inicio de sesión

Gracias al desarrollo del Guard de redireccionamiento de páginas y al manejo de sesiones, la página web redirecciona al inicio de sesión siempre que no exista una sesión activa en el navegador, en esta sección se encuentra un formulario en donde deben ser ingresadas las credenciales para hacer uso de la aplicación, en caso de no contar con credenciales para el uso de la aplicación existe una opción para crear una cuenta para poder acceder a la aplicación.

Continuación del apéndice 1.

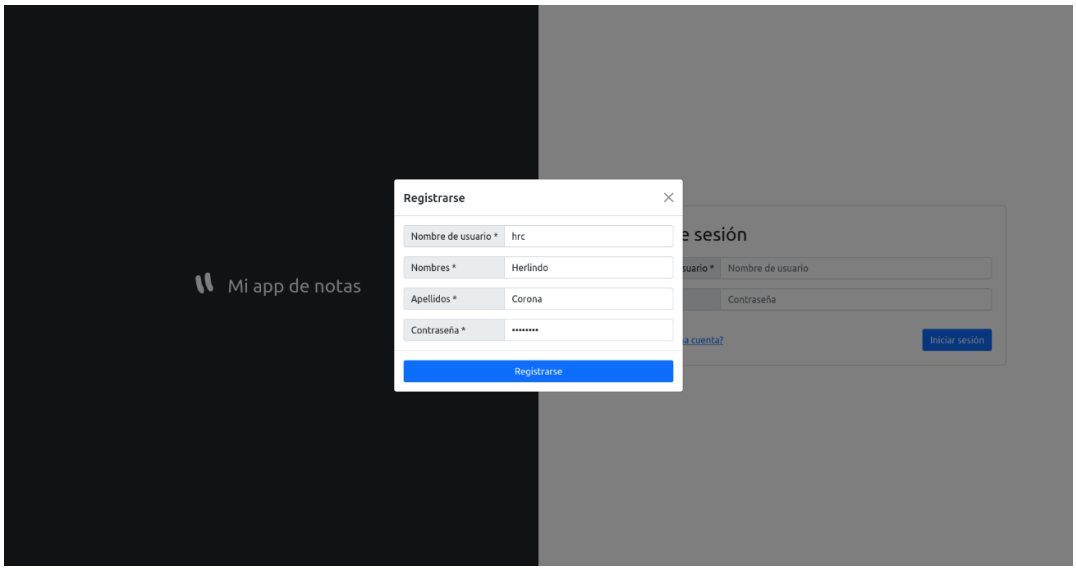




- Registro de usuario

De no contar con credenciales para el uso de la aplicación podemos crearla desde el formulario de registro de la aplicación, este formulario puede ser accedido desde la opción de crear cuenta que se encuentra desde el inicio de sesión. Para realizar el registro es necesario llenar todos los campos marcados con *, ya que son requeridos para poder completar el registro, posterior al registro del usuario podremos contar con acceso a la aplicación y se manejara la sesión como se propuso en el desarrollo del documento.

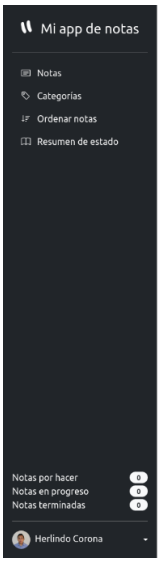
Continuación del apéndice 1.



- Pantalla de inicio

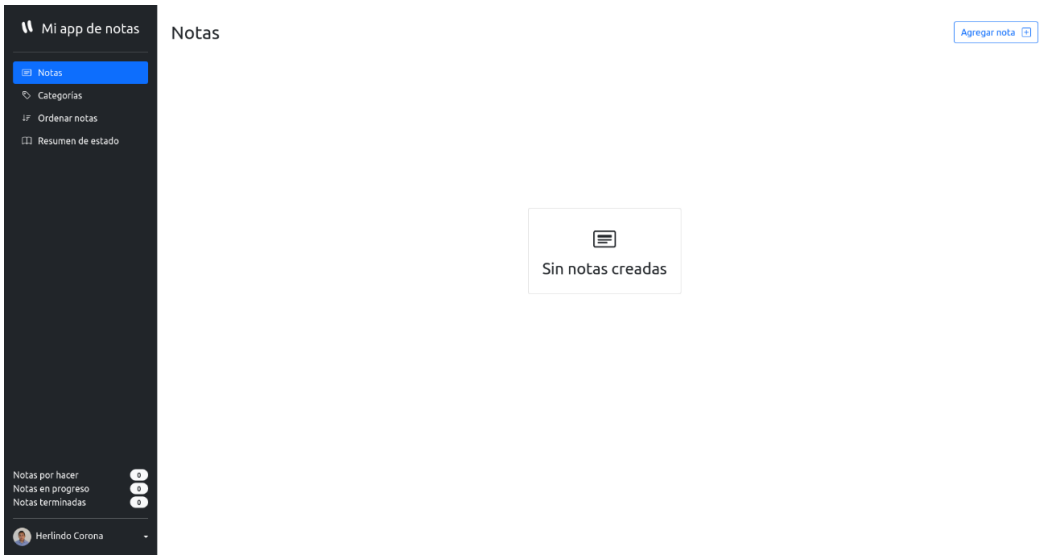
Posterior a realizar un registro de cuenta, realizar un inicio de sesión y en caso de ser existir una sesión previa, la aplicación redirigirá automáticamente a la pantalla de inicio de la aplicación, tal como se planteó en el manejo de Guards durante la propuesta del manejo de sesiones. En esta sección se encontrará la pantalla de bienvenida con el logo de la aplicación y un sidebar con las distintas secciones para el manejo de la aplicación, resumen de notas del usuario y el menú de manejo del usuario.

Continuación del apéndice 1.

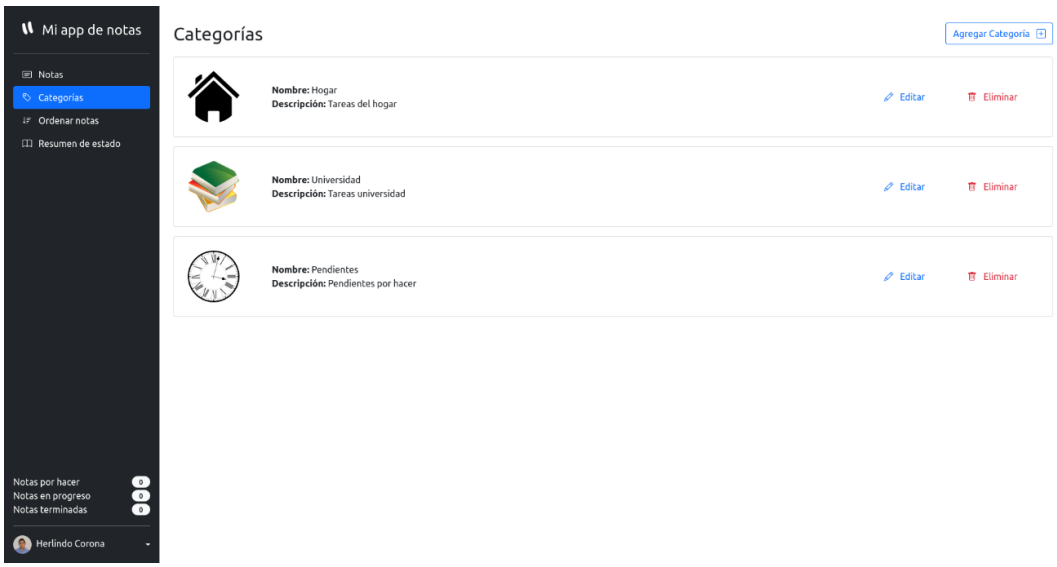


- Listado de notas y categorías

Por defecto el listado de notas y categorías contarán con una pantalla que indique que el listado de elementos está vacío, al igual que la mayoría de las funcionalidades, esta pantalla hace uso del estado de la aplicación para saber si el listado se encuentra o no vacío. En esta pantalla se encontrará también por defecto la opción de agregar un nuevo elemento, ya sea existan o no elementos relacionados con la sección que se está trabajando.



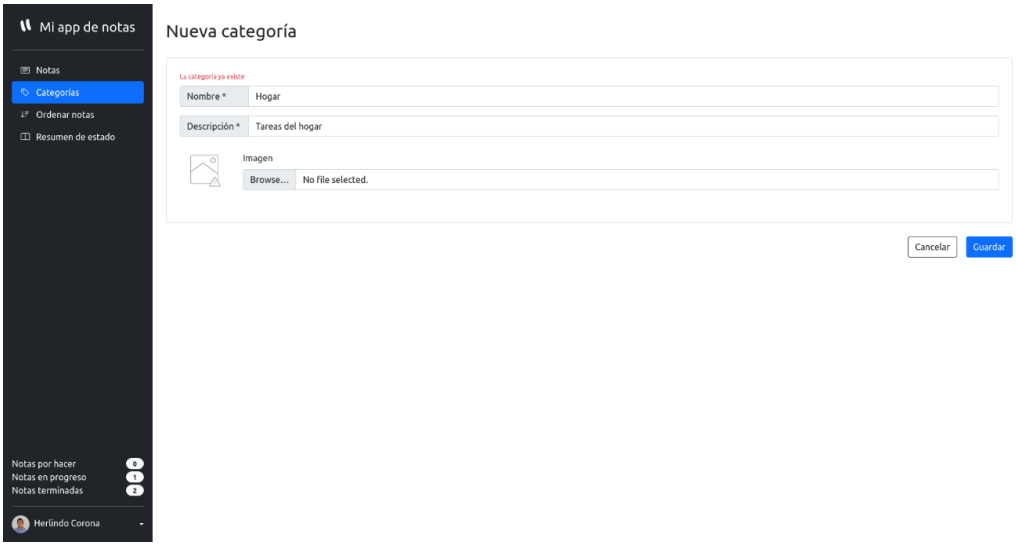
En caso de existir elementos creados en cualquiera de las secciones, se tendrá un breve resumen del elemento y contará con la opción de edición y eliminación del elemento, en esta sección basta con seleccionar la opción de edición para ser redirigido a la pantalla de edición y seleccionar la opción de eliminar para eliminar un elemento.



Continuación del apéndice 1.

- Creación de notas y categorías

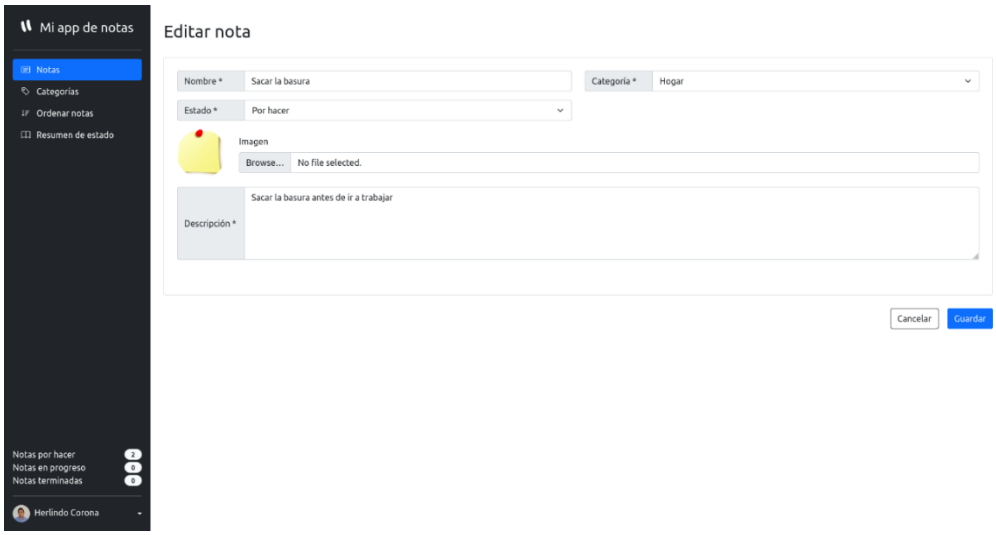
La creación de notas y categorías es llevada a cabo desde la sección del listado de elementos, es necesario seleccionar la opción crear para ser redirigidos a la pantalla de creación del elemento deseado, en esta sección se debe llenar los formularios con todos los campos requeridos marcados con * por la aplicación para poder crear un nuevo elemento, también es necesario resolver todos los errores informados por el componente en caso de existir previo a crear un nuevo elemento.



Continuación del apéndice 1.

- Edición de notas y categorías

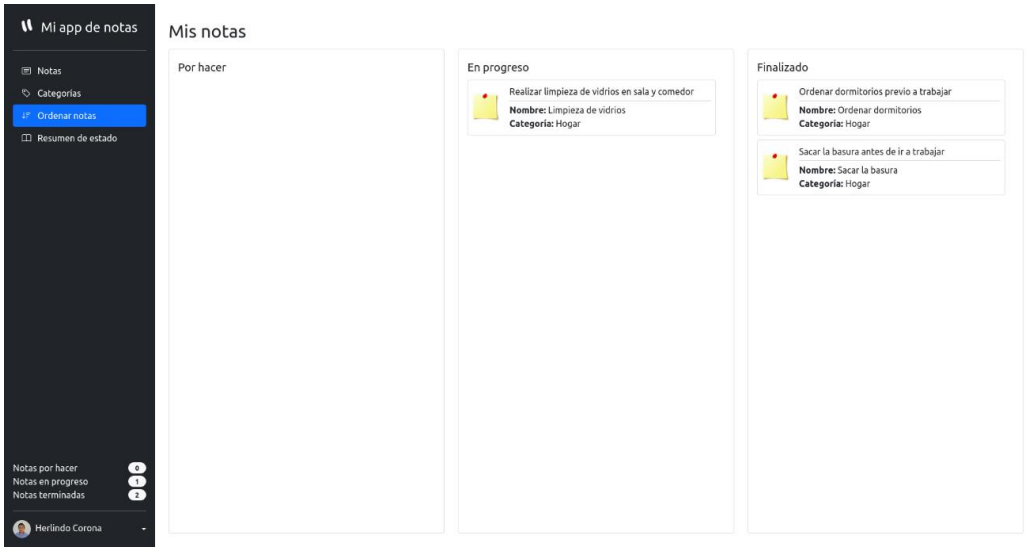
Posterior a seleccionar cualquier nota o categoría, a editar la aplicación, redireccionará a la pantalla de edición de elemento, esta sección funciona exactamente igual que la sección de crear elementos, por lo que es necesario llenar los campos requeridos del formulario y resolver los errores informados por parte del componente para editar un elemento.



Continuación del apéndice 1.

- Ordenar notas

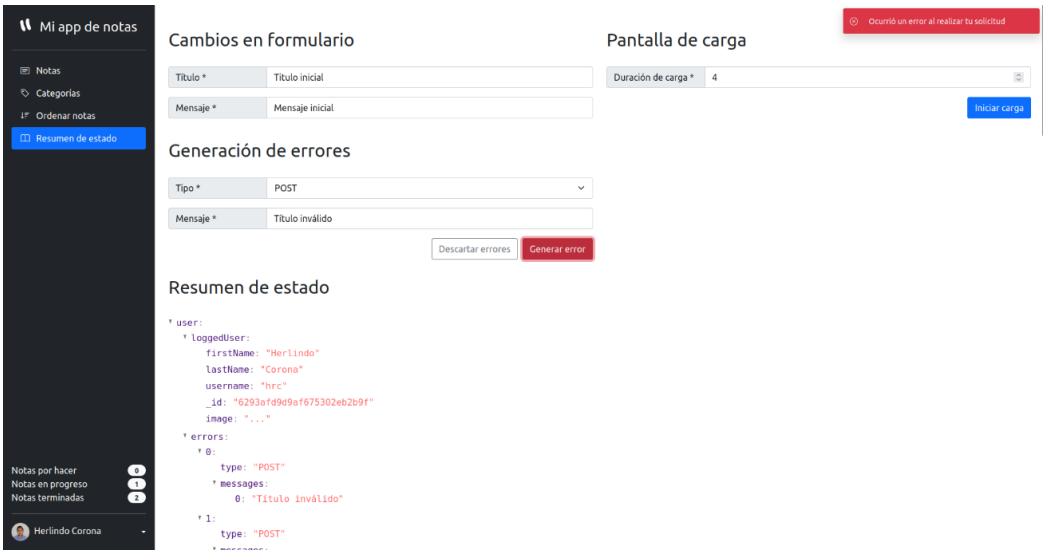
La sección de ordenado de notas funciona de forma muy similar a un tablero Kanban, basta con seleccionar cualquiera de las notas en el tablero y arrastrar dicha nota a la sección del estado en que se desea que la aplicación se encuentre. Esta sección funciona muy de la mano con el estado de la aplicación, ya que al modificar el estado de una nota el resumen de notas que se encuentra en el sidebar es actualizado automáticamente gracias a la modificación del estado al cambiar de estado cualquier nota.



Continuación del apéndice 1.

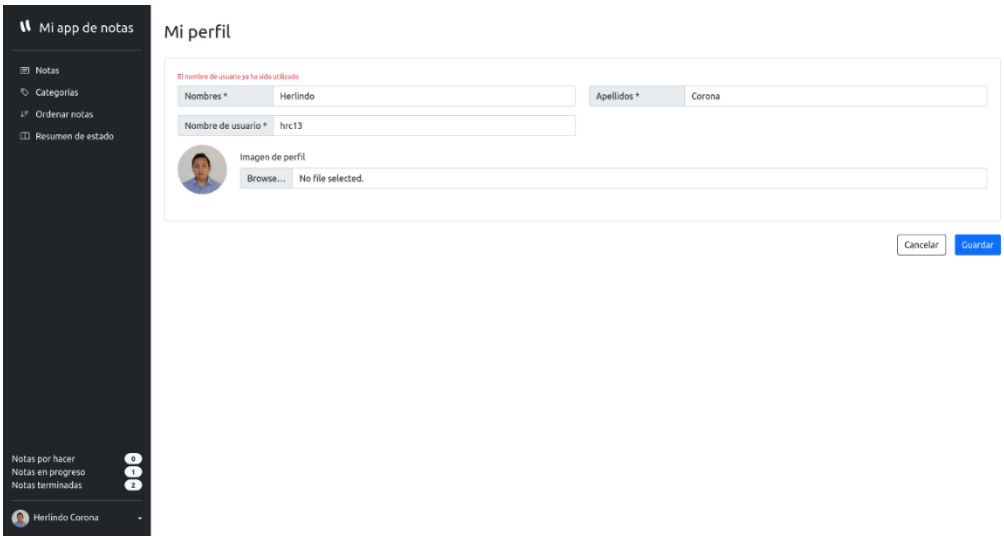
- Simulación de errores

Para visualizar de mejor manera el manejo del estado de una aplicación se incluyó una sección resumen donde se puede simular errores en la aplicación, en esta sección basta con seleccionar el tipo de error a generar y el mensaje de error a generar, también se cuenta con la opción de descartar errores para poder ver como estos errores son modificados desde el resumen del estado.



Adicional a la simulación de errores podemos generar un error de forma manual al crear una categoría con el mismo nombre y descripción de cualquier otra categoría existente. Otras opciones para generar errores desde el servicio externo utilizado para el desarrollo de la aplicación puede ser la utilización de un nombre de usuario ya utilizado, notas con el mismo nombre, categoría y descripción y la creación de notas sin ninguna categoría generada.

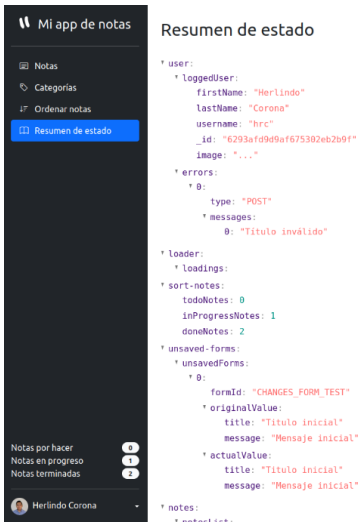
Continuación del apéndice 1.



- Resumen de estado

Adicional a la posibilidad del uso de herramientas externas para visualizar el estado de la aplicación, se incluyó dentro de la aplicación ejemplo un resumen del estado actual de la aplicación, este estado cambia de forma automática con cada una de las acciones tomadas por el usuario desde cualquier parte de la aplicación o desde las acciones que pueden ser encontradas en la sección del resumen de estado.

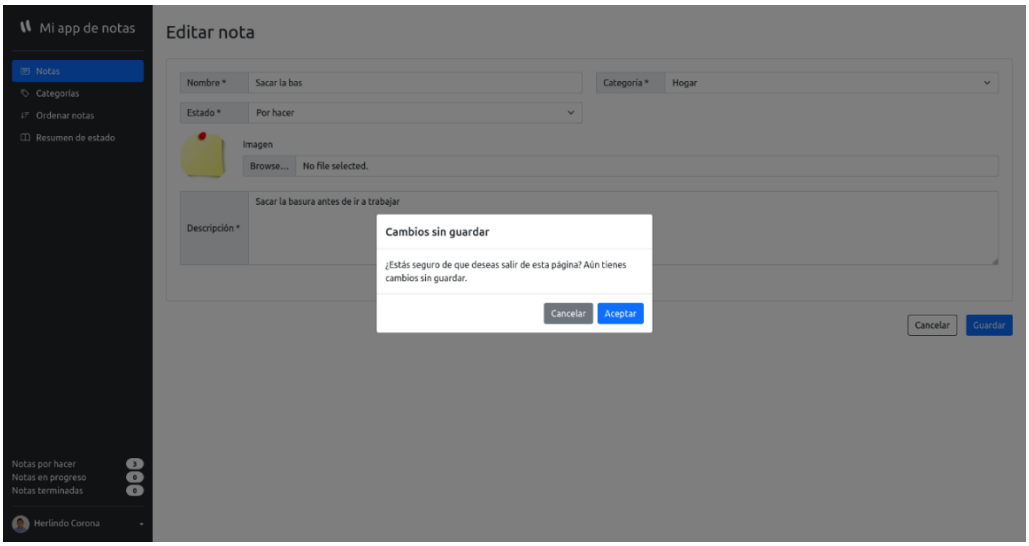
Continuación del apéndice 1.



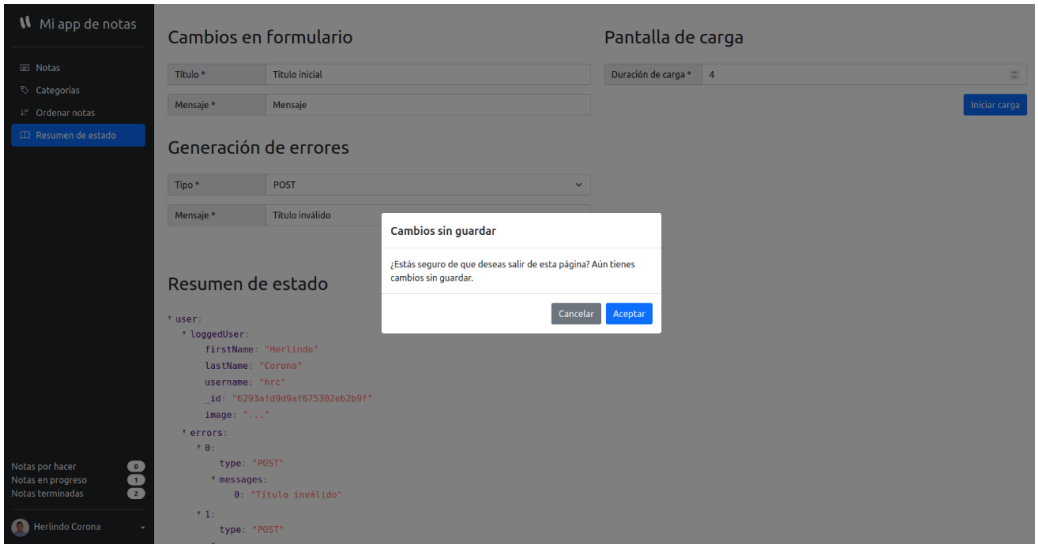
- Formularios sin guardar

Todos los formularios dentro de la aplicación, a excepción de los formularios de inicio de sesión y registro, cuentan con detección de cambios, por lo que si nos encontramos en el formulario de creación y seleccionamos desde el sidebar cualquiera de las secciones de la aplicación veremos la advertencia de cambios en un formulario siempre que este formulario aún no haya sido guardado, en esta sección bastara con que el usuario confirme si desea o no salir de la sección en la que se encuentra.

Continuación del apéndice 1.



Adicional a encontrar esta funcionalidad en todos los formularios, se puede simular esta funcionalidad desde la sección de resumen de estado modificando el valor del formulario y saliendo de la sección de resumen de estado, en esta sección se podrá visualizar de mejor manera como los valores de los formularios de la aplicación son actualizados en el estado al cambiar su valor.

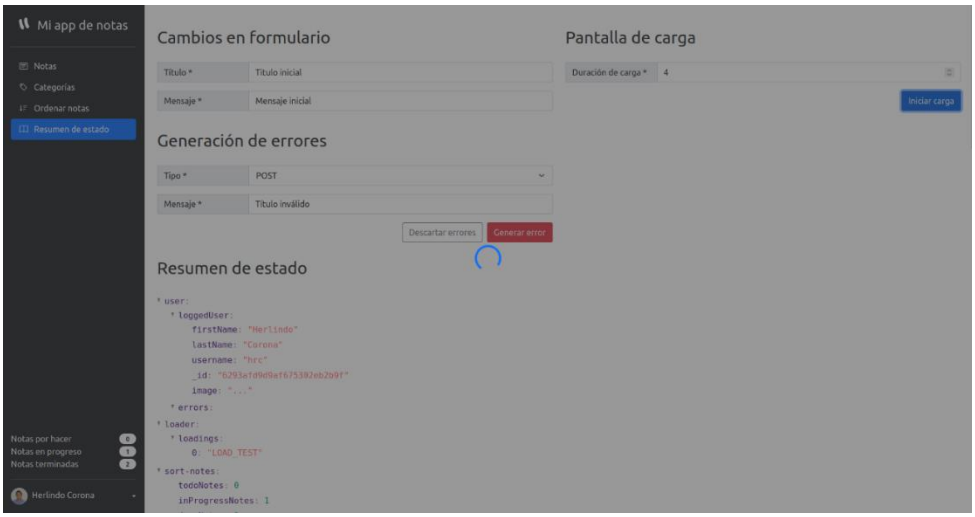


Continuación del apéndice 1.

- Pantalla de carga

Todas las funciones de carga, guardado, edición y eliminación de notas, categorías y manejo de usuario cuentan con pantallas de carga durante la espera de respuesta por parte de un servicio externo, por lo que basta con crear o editar cualquier nota para poder visualizar esta funcionalidad.

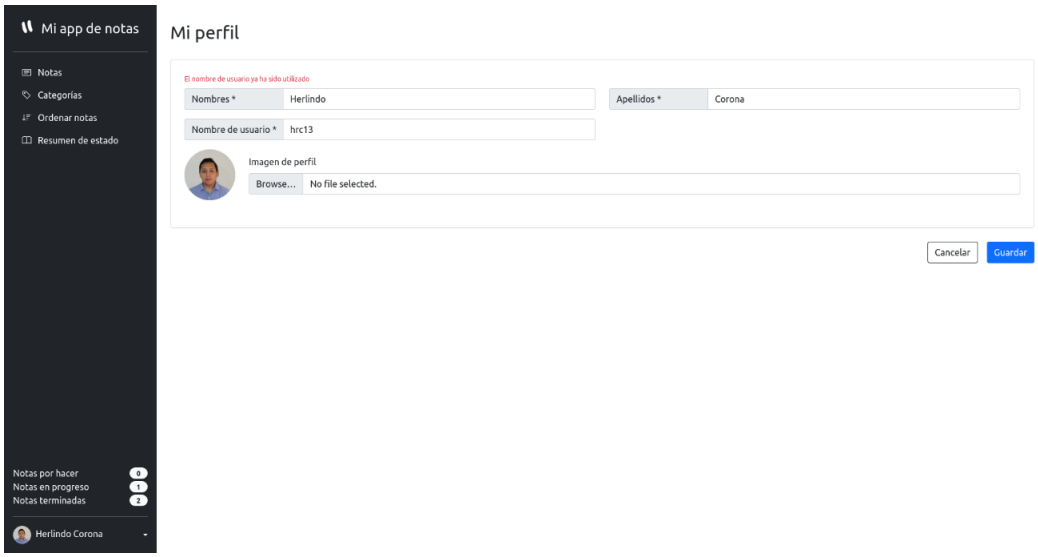
Debido a que esta funcionalidad generalmente depende de la velocidad de internet, se incluyó un formulario de simulación de pantalla de carga en la sección de resumen de estado, en esta sección se puede definir la duración en segundos que se desea que la duración de la pantalla tenga para realizar dicha simulación, también es de utilidad para visualizar como en el estado esta carga es registrada durante la duración especificada en el formulario de simulación.



Continuación del apéndice 1.

- Actualización de información de perfil

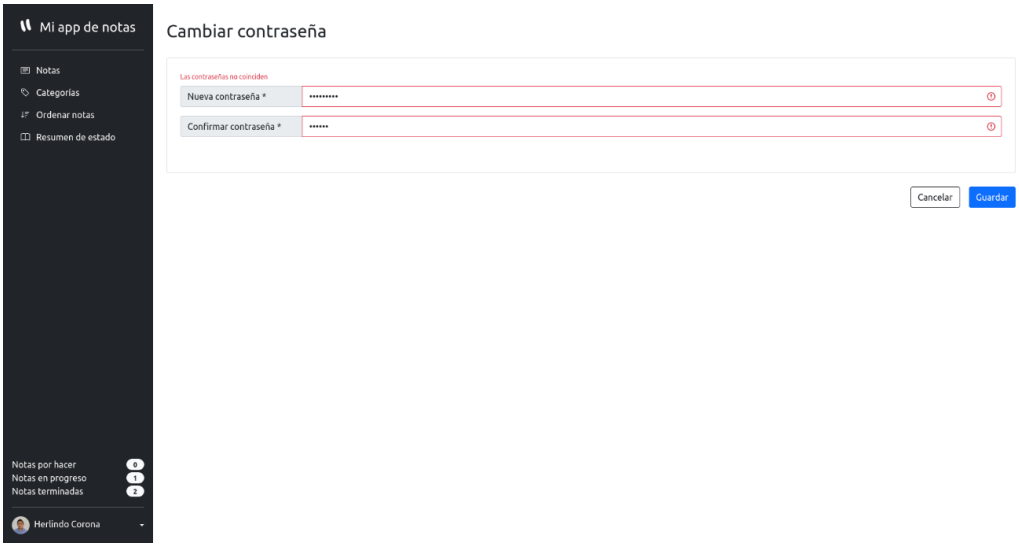
La información de perfil puede ser actualizada desde la sección de usuario encontrada en la parte inferior izquierda del sidebar de la aplicación, al seleccionar la modificación del perfil se redireccionará al formulario de actualización de perfil en donde se encuentra un formulario con los campos obligatorios a llenar y se tendrá la posibilidad de realizar el cambio de imagen de perfil. En caso de existir un error desde el servicio externo, el componente reaccionara ante dicho error e informara el error para que sea solucionado y se pueda proceder con la actualización de la información de perfil.



Continuación del apéndice 1.

- Cambio de contraseña

El cambio de contraseña también podrá ser realizado a partir de la sección de usuario encontrada en la parte inferior izquierda del sidebar de la aplicación, una vez redirigido se encontrara un formulario en donde se ingresara la nueva contraseña, la cual debe ser confirmada para poder realizar el cambio, en caso de que las contraseñas no coincidan el componente informara dicho error para que la contraseña sea confirmada correctamente y se pueda realizar el cambio y pueda actualizarse la sesión activa basándose en una nueva autorización.



Fuente: elaboración propia.

Apéndice 2. **Manual de levantado de aplicación web**

En este manual se encuentra las instrucciones sugeridas para el levantado de la aplicación frontend. En el manual se indica como realizar el levantado de la aplicación en un ambiente de desarrollo, realizar la configuración de las variables de ambiente, construcción de la aplicación para un ambiente de producción y el levantado de la aplicación en un ambiente de producción haciendo uso de Docker.

Adicionalmente se puede usar el siguiente video tutorial: https://drive.google.com/file/d/15A1S6e5-vGihDNPdR2eQZTK86FMyD6F7/view?usp=sharing, como referencia más detallada para realizar el levantado de la aplicación frontend en un ambiente de desarrollo junto con su aplicación backend complemento que se encuentra en el siguiente repositorio: https://github.com/uhcoronaa/my-notes-app-be, aplicación que también cuenta con su propio manual de levantado y videotutorial correspondiente.

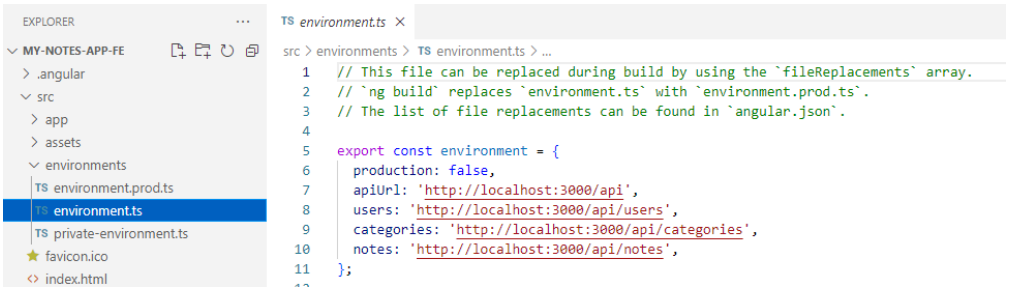
- Configuración de variables de ambiente

Para realizar el levantado de la aplicación se debe especificar en el archivo de variables de ambiente todas las direcciones a servicios externos utilizados en la aplicación, en este caso debe indicarse la dirección de los servicios de usuarios, categorías y notas.

Los archivos de variables de ambiente se encuentran en el directorio environments dentro del directorio src de la aplicación. Por defecto, Angular configura los archivos environment.ts y environment.prod.ts para el manejo de variables de ambientes de producción y desarrollo.

Continuación del apéndice 2.

En el archivo environment.ts debe especificarse las direcciones correspondientes para el ambiente de desarrollo y en el archivo environment.prod.ts las direcciones correspondientes para un ambiente de producción.



- Instalación de módulos

La instalación de módulos necesarios para el levantado de aplicación es realizada con ayuda de Node.js y NPM como gestor de paquetes, contado con estas herramientas basta con hacer uso del comando de instalación “npm install” del gestor de paquetes dentro del directorio de la aplicación para realizar la instalación de todas las dependencias necesarias para el levantado de la aplicación.

```
rc@DESKTOP-888I391 MINGW64 ~/Desktop/APORTE USAC/my-notes-app-fe
$ npm i
npm WARN deprecated source-map-resolve@0.6.0: See https://github.com/lydell/source-map-resolve#deprecated

added 866 packages, and audited 867 packages in 1m

102 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Continuación del apéndice 2.

- Levantado de aplicación en modo desarrollo

Una vez realizada la configuración de variables de entorno e instalación de las dependencias de la aplicación se puede iniciar con el levantado de la aplicación, para esto se recomienda hacer uso del ejecutor de paquetes de npx para evitar cualquier conflicto entre módulos de Angular instalados globalmente y entre los módulos utilizados en la aplicación, para realizar el levantado basta con hacer uso del comando “npx ng serve”.

Al realizar el despliegue en modo desarrollo, la aplicación hará uso de las variables de entorno definidas para el ambiente de desarrollo, adicional a esto Angular hace uso de herramientas proporcionadas por defecto para que los cambios que se generen durante el desarrollo sean reflejados inmediatamente en el navegador durante la ejecución de la aplicación en un ambiente de desarrollo.

```
rc@DESKTOP-888I391 MINGW64 ~/Desktop/APORTE USAC/my-notes-app-fe
$ npx ng serve
⠸ Generating browser application bundles (phase: setup)...Processing legacy "View Engine" libraries:
- ngx-json-viewer [es2015/esm2015] (git+https://github.com/hivivo/ngx-json-viewer.git)
Encourage the library authors to publish an Ivy distribution.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names      |  Raw Size
vendor.js           | vendor     |    3.19 MB |
styles.css, styles.js | styles   |  415.29 kB |
polyfills.js        | polyfills  |  303.38 kB |
scripts.js          | scripts    |  164.58 kB |
```

Continuación del apéndice 2.

- Construcción de aplicación para entorno de producción

Para realizar la construcción de la aplicación para un ambiente de producción debe realizarse todos los pasos previos definidos anteriormente. Posterior a los pasos anteriores basta con hacer uso del comando “npx ng build” junto con banderas de especificación indicando que el ambiente de despliegue para saber que variables deben utilizarse para la generación de los archivos finales de la construcción para ser en cualquier servidor de páginas web.

Al igual que con el despliegue en un entorno de desarrollo, se recomienda utilizar el ejecutor de paquetes de npx para evitar cualquier conflicto entre versiones de módulos. Los archivos finales optimizados de la aplicación podrán encontrarse en la carpeta dist del directorio de la aplicación, estos archivos son los empleados para el despliegue en cualquier servidor.

```
rc@DESKTOP-888I391 MINGW64 ~/Desktop/APORTE USAC/my-notes-app-fe
$ npx ng build --configuration=production
✓ Browser application bundle generation complete.
✓ Copying assets complete.
⠦ Generating index html...1 rules skipped due to selector errors:
  legend+* -> Cannot read properties of undefined (reading 'type')
✓ Index html generation complete.

Initial Chunk Files           | Names                         |  Raw Size | Estimated Transfer Size
main.9e507edc955d603d.js      | main                          | 360.86 kB |                94.68 kB
styles.83516f0505acb5a6.css   | styles                        | 224.37 kB |                25.29 kB
scripts.c9c6a43cee3e1491.js   | scripts                       | 165.20 kB |                48.74 kB
```

- Levantado de aplicación con Docker

Para un fácil despliegue se incluye en el directorio de la aplicación archivos de configuración de Docker para reducir la complejidad en el despliegue de la aplicación en un entorno de producción.

Continuación del apéndice 2.

Este archivo de configuración se encarga de la construcción de la aplicación y del levantado de la aplicación en un servidor Nginx en el puerto 80 del servidor, es importante recordar que si bien Docker facilita el despliegue en producción es importante configurar las variables de ambiente como se indica en los pasos iniciales para evitar cualquier tipo de conflicto.

Fuente: elaboración propia.

Apéndice 3. **Manual de levantado de aplicación backend**

A continuación, se muestra el manual de levantado de la aplicación backend desarrollada para ejemplificar de mejor manera los ejemplos desarrollados en la aplicación frontend para el trabajo de graduación.

Para un fácil levantado de la aplicación backend, se incluye una configuración desarrollada con Docker-Compose. Esta configuración facilita el levantado de la aplicación backend. Si bien el despliegue es facilitado haciendo uso de Docker-Compose, es necesario configurar variables de ambiente para poder realizar el levantado de la aplicación. El repositorio de la aplicación se encuentra en el siguiente repositorio: https://github.com/uhcoronaa/my-notes-app-be.

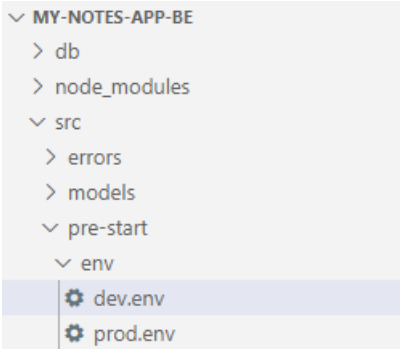
Adicional al manual de levantado de la aplicación, se incluye un videotutorial en donde se ejemplifica el levantado de la aplicación backend con Docker-Compose y el levantado de la aplicación frontend del trabajo de graduación. El video puede ser encontrado en el siguiente enlace: https://drive.google.com/file/d/15A1S6e5-vGihDNPdR2eQZTK86FMyD6F7/view?usp=sharing.

También puede ser encontrado un videotutorial con una explicación mucho más detallada del levantado de la aplicación backend haciendo uso de NodeJS sin necesidad de hacer uso de Docker-Compose, el video puede ser encontrado en el siguiente enlace: https://drive.google.com/file/d/1YpB_gyVqpx2YCU9Gmf6X7AMz0Td3hwXA/view?usp=sharing.

Continuación del apéndice 3.

- Configuración de variables de ambiente

Al igual que en la aplicación frontend, esta aplicación permite el manejo varios ambientes de ejecución a través del manejo de varios archivos con variables de ambiente.



El manejo de múltiples ambientes es realizado a través del nombre del archivo de las variables de ambiente, este nombre funciona como diferenciador para el ambiente a ejecutar, por lo que al desplegar la aplicación se hará uso del nombre del archivo como diferenciador para definir el ambiente a utilizar. En este caso la configuración de los archivos Docker-Compose está hecha para el manejo del ambiente de producción, por lo que el nombre del archivo por defecto es prod.env.

Dentro del repositorio puede encontrarse varios archivos con la estructura necesaria para el manejo de las variables de ambiente a configurar, basta con modificar los valores de las variables de ambiente con los valores deseados o crear nuevos archivos para crear nuevos ambientes de ejecución.

Continuación del apéndice 3.

- Variables de ambiente a configurar

    - NODE_ENV

Variable utilizada para especificar el tipo de ambiente en el que se ejecutara la aplicación. Por defecto se usa el valor dev para un ambiente de tipo desarrollo y production para un ambiente de tipo producción.

    - PORT

Variable utilizada para especificar el puerto en el que se realizara el despliegue de la aplicación, esta variable es de utilidad para saber el puerto a configurar en las variables de ambiente para proyectos frontend o realizar el mapeado de puertos en caso de implementar servidores proxy.

    - TOKEN_SECRET

Variable utilizada para especificar la llave con la que se creara los tokens de autorización de la aplicación. La longitud de esta llave debe de ser compatible con el algoritmo de encriptación HS256.

    - TOKEN_REFRESH_SECRET

Variable utilizada para especificar la llave con la que se creara los tokens de actualización de la aplicación. La longitud de esta llave debe de ser compatible con el algoritmo de encriptación HS256.

Continuación del apéndice 3.

- MONGO_DB_URL

Variable con la URL de conexión a base de datos de mongo, para esta aplicación la base de datos ya está incluida en el archivo de configuración de Docker-Compose por lo que la conexión debe realizarse a la dirección db o localhost como dirección de conexión a la base de datos, se deja un ejemplo de la cadena de conexión.

```
mongodb://usuarios-de-base-de-datos:password-base-de-datos@db:27017/nombre-de-base-de-datos
```

- SALT_ROUNDS

Variable de entorno que indica el número de iteraciones a realizar en la librería bcrypt para crear una cadena de caracteres aleatoria que permita la generación de cadenas de caracteres encriptadas.

- Configuración inicial de base de datos

La base de datos requiere una configuración de variables de ambiente. Esta configuración es necesaria para que Docker ejecute la configuración inicial de la base de datos a utilizar en la aplicación backend, dentro del repositorio se encuentra una carpeta con el nombre db con la estructura esperada de las variables de ambiente a utilizar, basta con cambiar los valores de muestra con nuevos valores para realizar el levantado de la base de datos.

Continuación del apéndice 3.

- Variables de ambiente para configuración de base de datos

    - MONGO_INITDB_ROOT_USERNAME

Variable dentro de archivo de variables de entorno con el nombre de usuario root para el manejo de la base de datos.

    - MONGO_INITDB_ROOT_PASSWORD

Variable dentro de archivo de variables de entorno con la contraseña de usuario root para el manejo de la base de datos.

    - MONGO_INITDB_DATABASE

Variable dentro de archivo de variables de entorno con el nombre de la base de datos a utilizar en la aplicación.

    - User

Variable dentro de archivo JavaScript de inicialización que contiene las credenciales del usuario administrador de la base de datos. Las credenciales generadas en este objeto son las mismas a utilizar en la URL de conexión de la base de datos.

Continuación del apéndice 3.

- Dbname

Variable dentro de archivo JavaScript de inicialización que contiene el nombre de la base de datos a crear.

- Rootuser

Variable dentro de archivo JavaScript de inicialización que contiene las credenciales del usuario root de la base de datos.

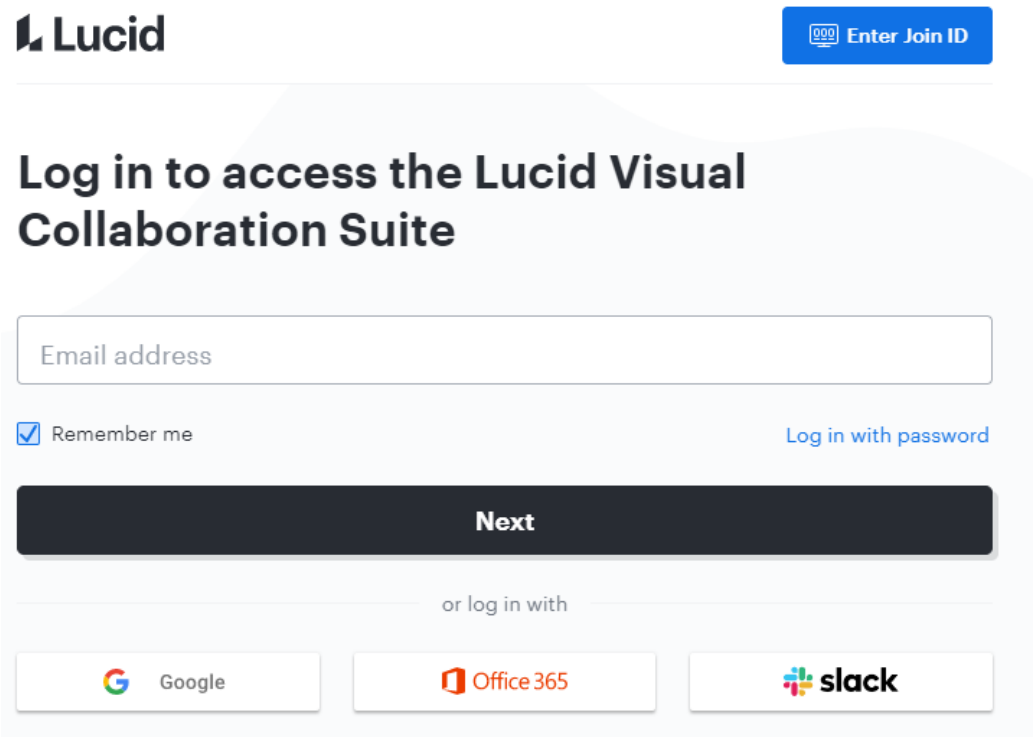
- Levantado con Docker-Compose

Una vez configurada las variables de ambiente, basta con modificar el mapeo de puertos en el archivo Docker-Compose para que coincidan con el puerto utilizado para la aplicación y hacer uso del comando de inicio de Docker-Compose para realizar el despliegue de la aplicación en el puerto especificado. El puerto utilizado por defecto en la aplicación es el puerto 27017.

Fuente: elaboración propia.

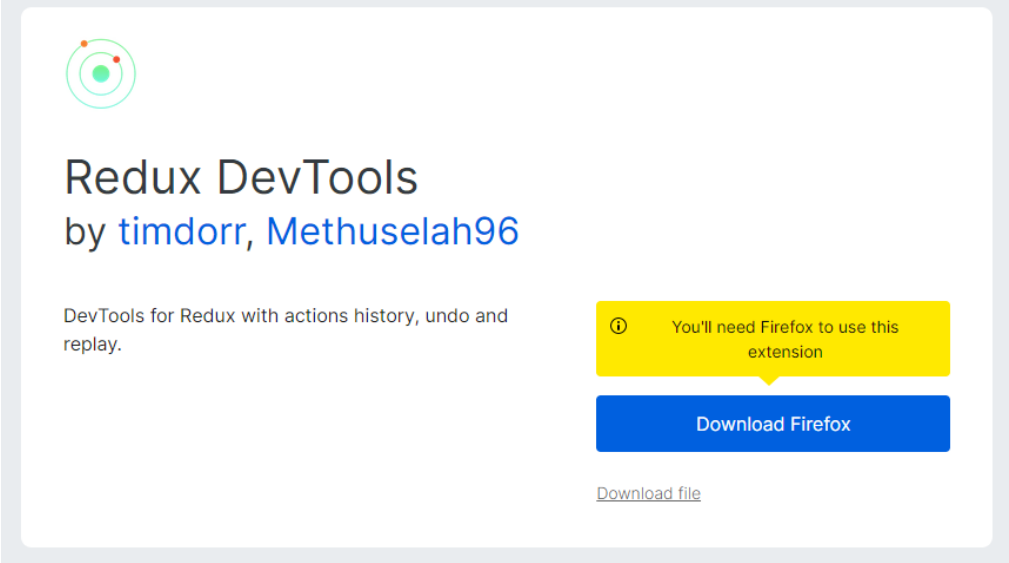
ANEXOS

Anexo 1. **Herramienta Lucid**



Fuente: Lucid. *Herramienta.* Recuperado de https://lucid.app/users/login#/login. Consultado: 21 de mayo de 2022.

Anexo 2. **Extensión Redux DevTools para visualización de estado desde herramientas de desarrollo del navegador**



Fuente: Firefox Browser Add-Ons. *Redux DevTools*. Recuperado de https://addons.mozilla.org/en-US/firefox/addon/reduxdevtools/.
Consultado: 21 de mayo de 2022.

Anexo 3. **Configuración de Redux DevTools**

app.module.ts

```
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { environment } from '../environments/environment'; // Angular CLI environment

@NgModule({
  imports: [
    StoreModule.forRoot(reducers),
    // Instrumentation must be imported after importing StoreModule (config is optional)
    StoreDevtoolsModule.instrument({
      maxAge: 25, // Retains last 25 states
      logOnly: environment.production, // Restrict extension to log-only mode
      autoPause: true, // Pauses recording actions and state changes when the extension window is not open
    }),
  ],
})
export class AppModule {}
```

Fuente: NgRx Devtools. *NgRx*. Recuperado de https://ngrx.io/guide/store-devtools. Consultado: 21 de mayo de 2022.