# Documentación

## Requisitos.

Para la realización del proyecto se necesitó: Flutter, AndroidStudio SDK y Java 8 o superior.

## Creación del proyecto.

Para crear el proyecto se selecciono la carpeta de origen donde se mantendrá el proyecto. Una vez en la carpeta abrimos la terminal en la dirección deseada y corremos el comando:

*Flutter create nass\_app --org com.upb*

Este comando se encargará de crearnos el proyecto con la configuración necesaria para compilar en Android, IOS y Web.

## Diseño

Para el diseño se realizó una revisión de diferentes propuestas de diseño subidas a Dribble y Pinterest, y se procedió a seleccionar los componentes que se usaran dentro del diseño final de la aplicación.

### Tipografía

Para la tipografía de la aplicación se uso la aplicación web “Fontjoy” la cual, por medio de un sistema de Deep Learning, se encarga de generar combinaciones de diferentes tipos de familias. La combinación que decidimos utilizar esta compuesta por: Titan One para los títulos, Taviraj para los subtítulos y Roboto para los párrafos.



Fig1: Tipografía utilizada en la aplicación.

### Colores

Para los colores de la aplicación se seleccionaron 5 generados desde la aplicación web “coolors” la cual nos dio diferentes combinaciones de colores que se podían ir bloqueando hasta obtener una paleta de colores acorde a los gustos de los desarrolladores en este caso. A continuación, se muestra la paleta final con sus valores en hexadecimal, que será utilizada en el desarrollo de la aplicación.

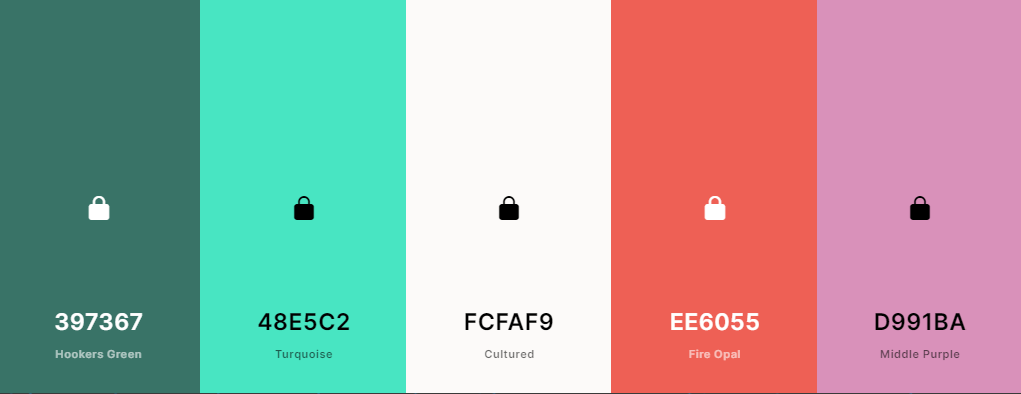


Fig2: Colores principales de la aplicación.

## Manejador de estados de la aplicación.

En Flutter tiene sentido mantener el estado por encima de los widgets que lo utilizan, ya que en frameworks declarativos para realizar un cambio en la interfaz del usuario, es necesario reconstruirla. Es por esto por lo que se tomó la decisión de usar [Riverpod](https://riverpod.dev/) como gestor de estado.



Fig3: Paquetes necesarios para la gestión de estados

## Arquitectura de software

Para la aplicación se hará uso de la arquitectura modelo vista controlador (MVC). Sin embargo, para el correcto funcionamiento de la aplicación es necesario agregar a la arquitectura los archivos de manejo de rutas, y de manejo de configuración relacionados con la estética.

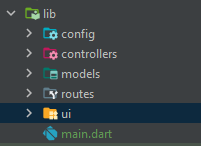


Fig4: Arquitectura de archivos a utilizar.

Es importante resaltar que la arquitectura MVC solo es posible gracias al uso de manejadores de estados como Riverpod, Provider, getX, entre otros, ya que estos permiten separar por completo la lógica, de la interfaz de usuario.

## Rutas

Para obtener una experiencia de usuario por medio del manejo de urls en la web, es necesario usar el paquete de navegación opcional de Flutter llamado Navigator 2.0, el cual exige un llamado declarativo del sistema de rutas. Para la aplicación se hará uso de la estructura de navegación presentada a continuación, en la Figura 5.

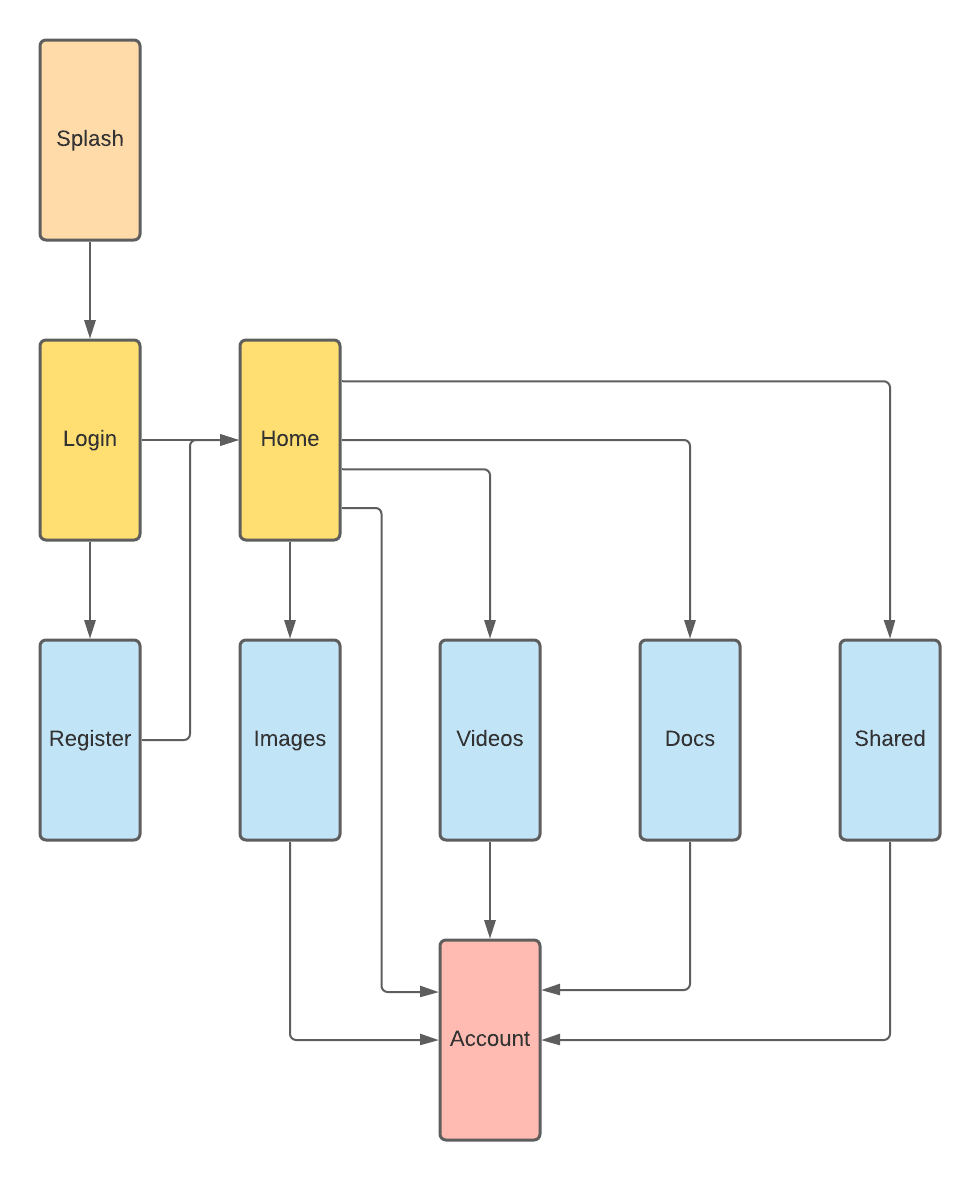


Fig5: Estructura de navegación.

### Desarrollo de la navegación

Para la navegación se realizaron pruebas con tres librerías diferentes ya que, debido a lo nuevo del lanzamiento del Navigator 2.0 y su complejidad, aun tienen algunos problemas de compatibilidad con otros paquetes. Se utilizo primero el paquete VRouter, sin embargo, al manejar FutureBuilders, encontramos un error al realizar retrocesos de la página, por lo que tuvimos que cambiar de librería. Luego utilizamos la librería Beam, sin embargo, la documentación se encontraba muy desactualizada respecto a los cambios que se le han realizado en los últimos meses, por lo que se decidió tampoco hacer uso de esta librería. Por último, utilizamos la librería Yeet, la cual, gracias a su simplicidad y documentación, supero con éxito las primeras pruebas de navegación. Por el momento seguiremos haciendo uso de esta librería para el proyecto, sin embargo, si se llega a encontrar un nuevo problema que no podamos superar, se podría replantear esta decisión.

final yeet = Yeet*(* children: *[* Yeet*(* path: '/',  
 builder: *(*\_, \_\_*)* => HomePage*()*,  
 *)*,  
 Yeet*(* path: '/login',  
 builder: *(*params, \_*)* => LoginPage*()*,  
 *)*,  
 Yeet*(* path: '/register',  
 builder: *(*params, \_*)* => RegisterPage*()*,  
 *)*,  
 Yeet*(* path: '/images',  
 builder: *(*params, \_*)* => ImagesPage*()*,  
 *)*,  
 Yeet*(* path: '/videos',  
 builder: *(*params, \_*)* => VideosPage*()*,  
 *)*,  
 Yeet*(* path: '/docs',  
 builder: *(*params, \_*)* => DocsPage*()*,  
 *)*,  
 Yeet*(* path: '/shared',  
 builder: *(*params, \_*)* => SharedPage*()*,  
 children: *[* Yeet*(* path: '/:sharedID',  
 builder: *(*params, \_*)* => SharedDataPage*(* sharedID: params*[*'sharedID'*]*,  
 *)*,  
 *)  
 ]*,  
 *)*,  
 Yeet*(* path: ':\_(.\*)',  
 builder: *(*\_, \_\_*)* => Page404*()*,  
 *)*,  
 *]*,  
*)*;

Gist1: Variable con la configuración de las rutas que se usaran.

## HomePage

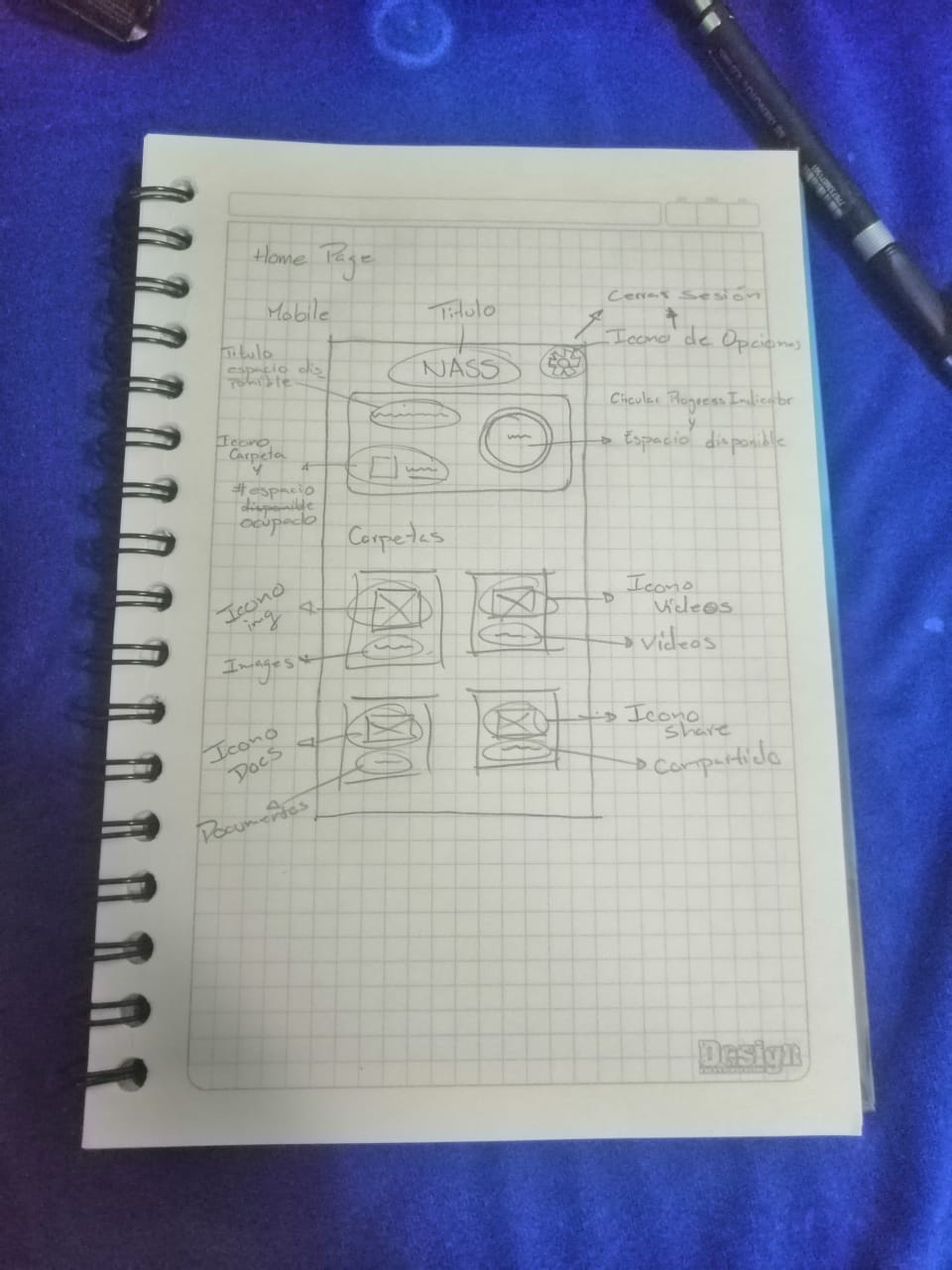
Para el diseño del HomePage, se tuvo en cuenta el diseño de Vagh Ritika publicado en dribble que pueden encontrar en: [Cloud Storage App by Vagh Ritika for MindInventory on Dribbble](https://dribbble.com/shots/15272009-Cloud-Storage-App), y el diseño de Farrel Putra que pueden encontrar en: [Cloud Storage Apps by Farrel Putra 🦊 for Pixelz Studio on Dribbble](https://dribbble.com/shots/9236389-Cloud-Storage-Apps). Teniendo en cuentra estos dos trabajos se diseño este primer esquema: 

Fig6: Bosquejo Home Page

Ya con el bosquejo definido, procedemos a realizar la construcción del diseño en adobe XD. El resultado final se muestra a continuación.

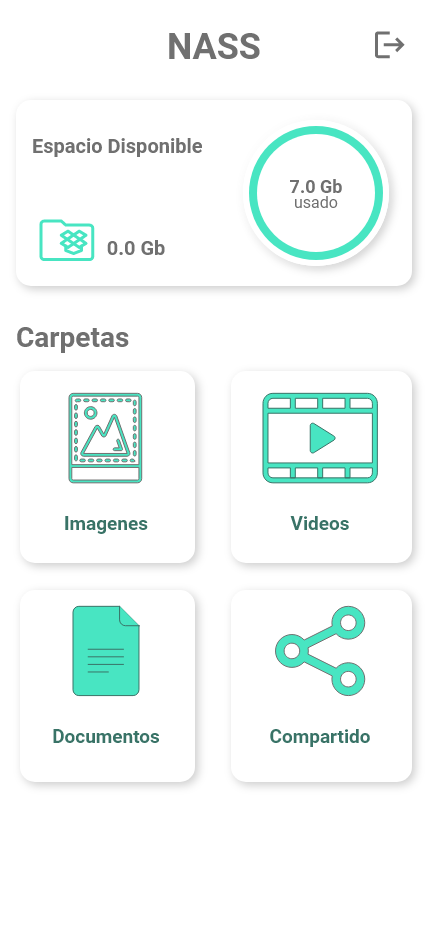


Fig 7: Diseño con Adobe XD del HomePage