# Introducción

La idea de este doc es que quede una guía de cómo trabajar con el proyecto de front y maomeno explicar las cosas básicas de las tecnologías y cómo las usamos y combinamos.

Primero veamos las tecnologías básicas que se usan:

## HTML:

Es el lenguaje que se usa para la maquetación de la app, o sea, el posicionamiento y orden de los componentes y cosas que queremos mostrar, las etiquetas que más van a estar viendo son:

* + span
  + p
  + h1,h2..
  + img
  + input
  + button
  + div. Los div son dios, usen divs, todo es divs, no se necesita nada más.

No recomiendo usar cosas como table, ul, ol, porque vienen con estilos horribles y al final joden más de lo que ayuda, usen divs, miren este video <https://www.youtube.com/watch?v=RjY_cFk-bjI>, no lo vi pero seguro explica algo bien

## CSS:

Permite hacer que el html quede lindo, en general van a estar creando clases y usando eso, les dejo este pdf con muchos ejemplos de cómo seleccionar los elementos que quieran sin tener que ponerle a todo clases y eso, ponele capaz tienen un:

<div class=’lista’><span>texto todo fachero</span></div>

Y para ponerle estilos al span pueden usar un selector::

.lista > span{} o lista span {}

[CSS selectors cheatsheet](https://appletree.or.kr/quick_reference_cards/CSS/CSS%20selectors%20cheatsheet.pdf)

También da cosas para organizar los elementos de html, con las props de display y position, les dejo unos videos cortitos para que se hagan una idea:

[CSS Flexbox in 100 Seconds](https://www.youtube.com/watch?v=K74l26pE4YA)

[CSS Grid in 100 Seconds](https://www.youtube.com/watch?v=uuOXPWCh-6o)

Y acá algunos estilos básicos que se usan mucho

[La guía de propiedades CSS - CSS en español - Lenguaje CSS](https://lenguajecss.com/css/introduccion/guia-css/) (ta medio fea la p[agina pero por lo que vi tiene cosas interesantes)

## Javascript:

Es donde vamos a tener la lógica, algunas cosas raras de javascript:

* + No hay tipos de datos estrictos, pod[es pasar cualquier cosa por par[ametros y no le importa ponele suma(a,b){return a+b}, a esto si quiero la llamo como suma(2,3) o suma(“pito”,”cola”) y va a funcar igual
  + Funciones flecha, se escriben as[i: (param1,param2,...) => {...codigo} mantienen el contexto, o sea que se puede usar el this. adentro, yo en general las uso para todos, pod[es declararlas y ponerlas en variables:

const suma = (a,b) => {return a+b};

Hay un par de variaciones de sintaxis como:

const potencia = n => n^2;

Ahí como es un param solo no son necesarios los par[entesis, y como solo quiero tener un return no es necesario meter {return n^2} y se pone => n^2

* + Se pueden pasar funciones por params:

const callback = ()=>{console.log(“esto es un callback :D”)};

const llamaCallback = (funcion) => {funcion()};

llamaCallback(callback);

* + Podemos definir objetos como se nos cante el orto:

let obj = {nombre:”pedro”};

obj = {apellido:”pedrosa”, edad:23};

Aunque cambien las props no importa, también podemos ponerle funciones como props:

let obj = {nombre:”pedro”, apellido:”jose”, funci[on: ()=>{//hace algo}}

NOTA: desde esas funciones no podemos acceder al this del objeto

* + Podemos negar cualquier cosa:

const obj = {nombre:”pedro”};

let niegoObj= !obj; //asigna false

let dobleNiegoObj = !!obj; //asigna true

const objNull = null;

niegoObj = !objNull; //asigna true, es como si tomara el null como false

dobleNiegoObj = !!objNull; //asigna false

* + Async y await, es todo un tema, les dejo este link [¿Cómo funcionan las Promises y Async/Await en JavaScript? [2022]](https://www.youtube.com/watch?v=rKK1q7nFt7M), de nuevo, no vi este video pero confío que explica algo, si no entienden preguntenme o pidanle al chatgepeto, este es un tema raro y puede ser bastante complejo, recomiendo siempre pensar en si algo es una promesa o no.

Javascript se puede usar con la etiqueta <script> de html, y hay 2 formas que vamos a usar principalmente:

<script type=”module”>

console.log(“codigo :D”);

//mas cosas

</script>

o

<script src=”/ruta/al/javascript/script.js” type=”module”>

**pongan el ‘type=”module”’ o no los va a dejar importar cosas, importante esto**

## DOM:

Es una estructura tipo árbol que tiene todas las etiquetas de nuestros .html, podemos acceder desde javascript para agarrar elementos particulares, agregar cosas, ponerle clases de forma dinámica y en general, hacer lo que se les cante el orto, recomiendo preguntarle a chatGPT cosas como: “como agregar una clase a un elemento con id con javascript” y les va a explicar todo

## JQuery:

es una librería que da una sintaxis más linda para el manejo del DOM desde javascript, la sintaxis es: $(“selector”) este selector es igual que los de css, así que pueden usar el pdf de antes para ver como agarrar elementos particulares

Ejemplo: quiero agarrar todos los elementos de una lista con forma: <div id=”lista”><p>el1</p><p>el2</p></div>

Quiero agarrar los p[arrafos entonces puedo hacer:

$(“#lista > p”).loquequierohacer();

Es más lindo que tener que hacer un

document.getElementById(“lista”).children.filter(e => e.tag == ‘p’);

Les dejo este link con varios ejemplos y cosas que pueden hacer:

[jQuery Syntax](https://www.w3schools.com/jquery/jquery_syntax.asp)

## LocalStorage:

El localStorage permite guardar info que se mantenga en el navegador aunque cerremos, o reiniciemos, o cambiemos la tab, esto es bastante útil para guardar info como el usuario logueado, y los parámetros que necesite la pantalla para funcionar (por ejemplo en la pantalla de ConsultaUsuario hay que pasarle que usuario queremos consultar como par[ametro)

Para guardar cosas en el localStorage podemos accederlo directamente desde javascript:

const misCosas = {cantidadDePatos:3, hora:”30:21”, edad:32}

localStorage.setItem(“misCosas”,JSON.stringify(misCosas));

Y para obtenerlo:

const misCosasString = localStorage.getItem(“misCosas”);

const misCosas = JSON.parse(misCosasString)

Si se fijan uso esta cosa rara de JSON.parse y JSON.stringify, esto lo que hace es pasar de un string a json y viceversa, es importante usar el JSON.stringify para guardar las cosas porque sino queda guardado un [Object object] y perdemos la info.

## 

# Conceptos base

## Router

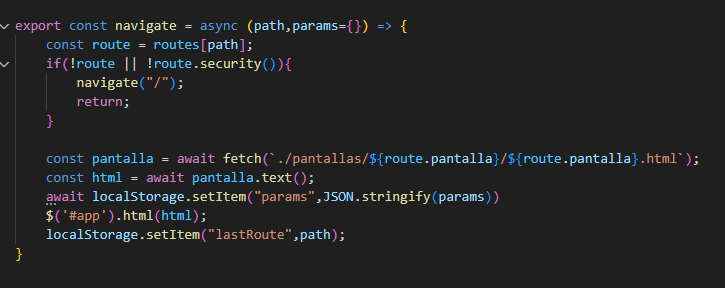
El router maneja en qué pantalla estamos, en general consiste de varias partes:

1. Lista de páginas que tiene el programa



Esto es un json, cada propiedad es una ruta del sistema, cada ruta es un objeto que a su vez tiene el nombre del html de la pantalla que corresponda, y una función security para ver si el usuario tiene permisos de entrar, en este caso las security o retornan true de base porque cualquiera puede entrar, o se fijan si un hay un usuario logueado, en javascript !! convierte los objetos a boolean je

1. Una funci[on de cambiar de pantalla, en nuestro caso es navigate:



Lo que hace es recibir la ruta, y los par[ametros que le queremos pasar a la pantalla, busca en las rutas la que le pasamos (si no la encuentra por ahora manda al home pero deber[ia mandarte a una pantalla de error, si se fijan tambi[en ah[i se llama la funci[on security(), para ver si tiene acceso), luego se pide el html de la pantalla, y por [ultimo usando jquery agarro el elemento #app, que es donde vamos a poner el content de las pantallas y le setea el html, si se fijan tambi[en deja guardado en el localStorage la ruta y los par[ametros, para que los reciba la pantalla.

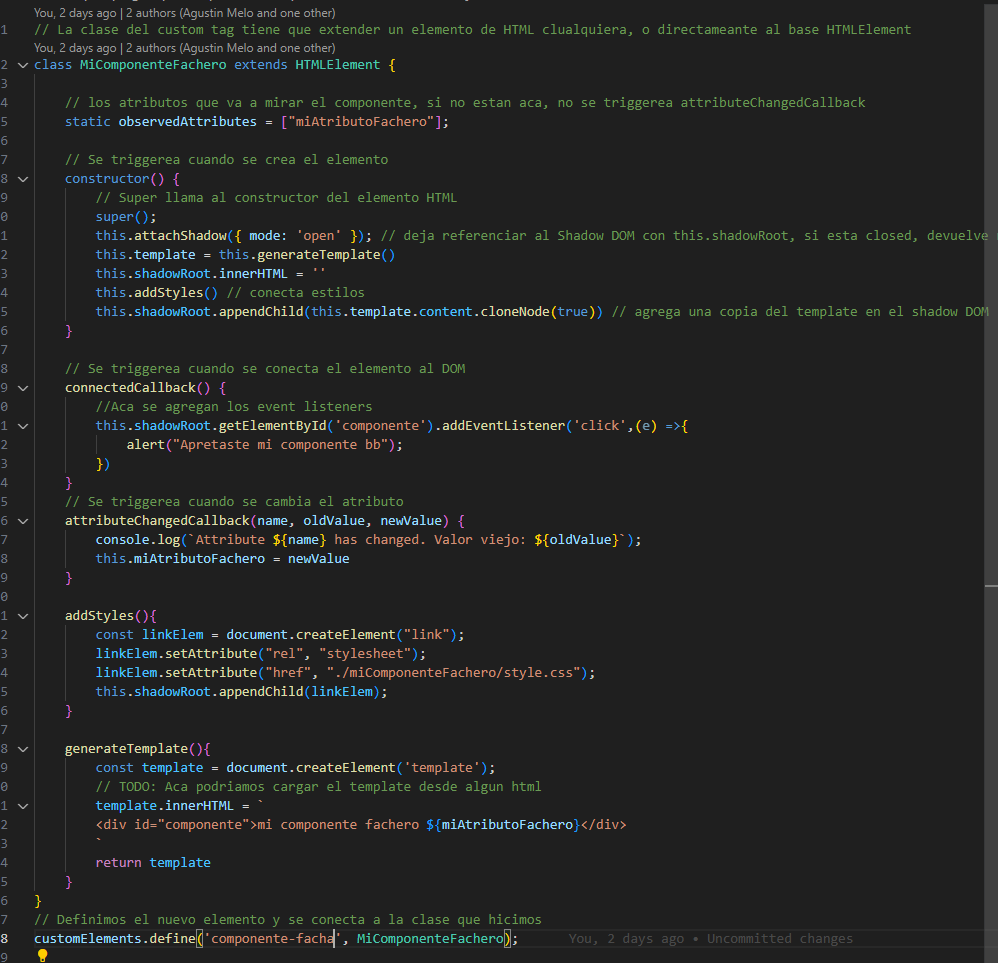
1. Alguna cosa m[as que todav[ia no se me ocurri[o, pero las importantes son esas primeras 2

## 

## Componentes

Nosotros vamos a usar componentes personalizados, que son b[asicamente etiquetas de html que creamos nosotros, esto est[a muy potente y es como se trabaja en cualquier framework de frontend (react,vue,angular 🤮), lo que vamos a usar viene con javascript vanilla y si bien es medio feo en c[odigo creo que viendo la estructura se va a entender un poco:

Para crear un componente arrancamos creando una clase que hereda de HTMLElement:



Ac[a dejo una estructura de ejemplo maomeno, para usar este componente en nuestro html luego tenemos que importar este javascript:

<script type=”module” src=”/ruta/a/miComponenteFachero.js”>

Y esto nos va a dejar usar:

<componente-facha miAtributoFachero=”pitos”></componente-facha>

Algunos consejos que les puedo dar para hacer componentes son:

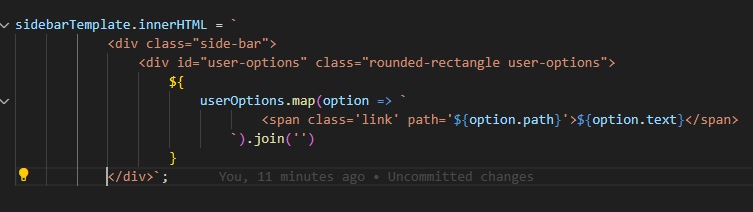
* Si quieren listar cosas o tener elementos din[amicos, en el m[etodo de getTemplate() pueden usar un arreglo con la info que quieran poner, y usar las funciones de .map().join(),

.map() lo que hace es recorrer el arreglo y aplicar una funci[on que recibe un elemento del arreglo y deber[ia retornar algo en funci[on del elemento, cuando termina de recorrer el arreglo retorna una copia con los returns de nuestra funci[on en cada elemento, le deber[ian pasar una funci[on flecha que retorne un string con html:

lista.map(l => l.nombre +” ”+ l.apellido); //Retorna un arreglo de strings con la forma “{nombre} {apellido}”

.join() concatena los elementos en un string

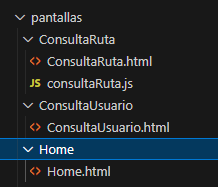
Les dejo un ejemplo en el que listo las opciones para el men[u del usuario:



## 

## Pantallas

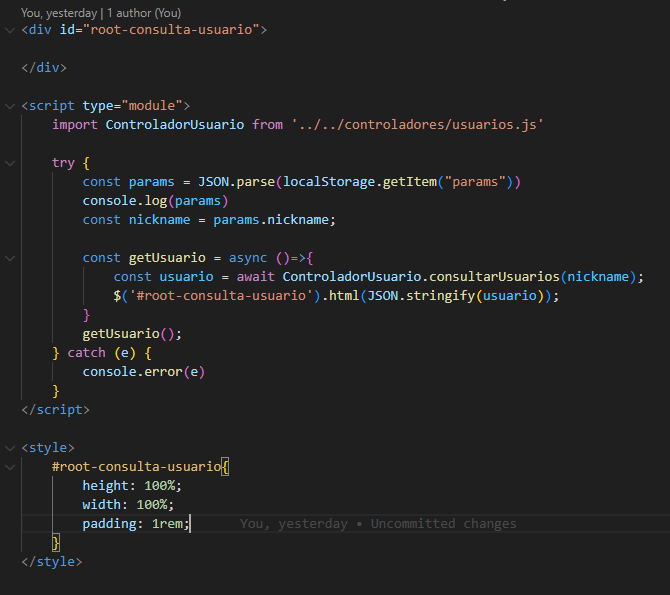
En nuestra app las pantallas van a ser archivos .html, estas est[an en la carpeta de pantallas y cada una tiene su carpeta propia:



**IMPORTANTE: pongan el mismo nombre a la carpeta que al .html que quieran cargar, esto es porque el router busca en esa ruta para cargar el .html**

Cada pantalla va a tener sus cosas distintas, pueden crear todos los archivos que se les cante en la carpeta de la pantalla, por ejemplo en la imagen se ve que ConsultaRuta tiene un .js creado, también pueden crear .css.

Sino la otra es tener todo en un archivo (creo que si no hay mucho c[odigo no hay problema con esto), la estructura que recomiendo ser[ia as[i:



En esta imagen tambi[en ven un ejemplo f[acil de c[omo cargar par[ametros, ese script se ejecuta cuando se carga el html, o sea que lo pueden usar para ir a buscar al localStorage o a los controladores los datos que quieran, y modificar el DOM para mostrar lo que se les cante

## 

## Controladores

Los controladores son archivos .js que tienen una clase controlador que solo tiene funciones est[aticas as[incronas, los controladores no tienen contexto o propiedades adem[as de estas.

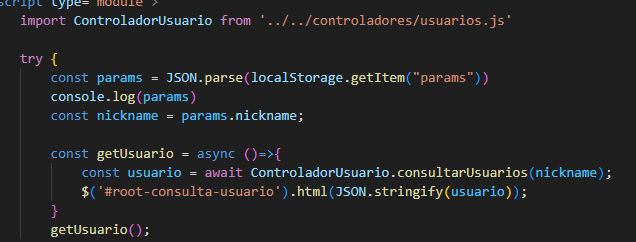
Estas clases son las que van a pegarle al proyecto de java para traer info y hacer los ABM, es por eso que todo es async y vamos a usar promesas para ver cu[ando java nos conteste.

Este es un ejemplo de controlador:



Como no tenemos la conexi[on con java por ahora solo va a retornar datos hardcodeados, pero est[a excelente esto, porque ya estamos simulando usar los controladores, y cuando conectemos a java solo vamos a tener que cambiar ac[a y nada del html o la l[ogica de los formularios.

Para usar este controlador podemos importarlo:



Si se fijan para llamar a la funci[on asincr[onicamente del controlador tuve que hacer una funci[on medio auxiliar y llamarla despu[es, otro ejemplo de esto ser[ia:

import ControladorRuta from ‘/ruta/de/controlador.js’

const getRuta = async (nombre) =>{

const res = await ControladorRuta.consultarRuta(nombre);

//hago algo con el res

}

getRuta(JSON.parse(localStorage.getItem(‘param’)).ruta);

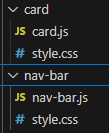
# Estructura del proyecto

Esta es nuestra hermosa estructura de proyecto, creo que las carpetas son bastante explicativas pero dejo una lista explicando maomeno cada una:

## assets:

donde vamos a poner im[agenes o [iconos que usemos en la app, por ejemplo yo ahora dej[e un logo provisorio

## componentes:

donde vamos a tener los componentes, cada componente a su vez tiene una carpeta con al menos 2 archivos que son el js con la clase y el archivo de css con los estilos

## 

## controladores:

tiene los 3 controladores, van a ser los mismos que los del back

## pantallas:

ya medio que se explic[o en la secci[on de pantallas anterior

## utils:

es medio la basura de la app, ac[a va todo lo que no sepan donde meter, por ahora dej[e la librer[ia de jquery, y el index.js es b[asicamente una especie de archivo con cosas [utiles para importar en varios lados, por ejemplo tiene una funci[on que te retorna el usuario parseado del localStorage, o tiene la funci[on navigate() para cambiar de pantalla, le vamos a ir agregando cosas a medida que sea necesario, cosas cl[asicas para meter son funciones de parseo de fechas, validaciones que se usen mucho, listas de cosas que se repite, yqs, vemos si merece la pena meterlo.

Ac[a dejo un ejemplo de algo que agregar[ia al index.js del utils y c[omo lo importar[ia en otro archivo:

index.js:

export const estados = [“aprobada”,”rechazada”,”pendiente”];

ConsultaRuta.html:

<script type=”module”> //Ac[a el type module es NECESARIO sino no les va a dejar importar

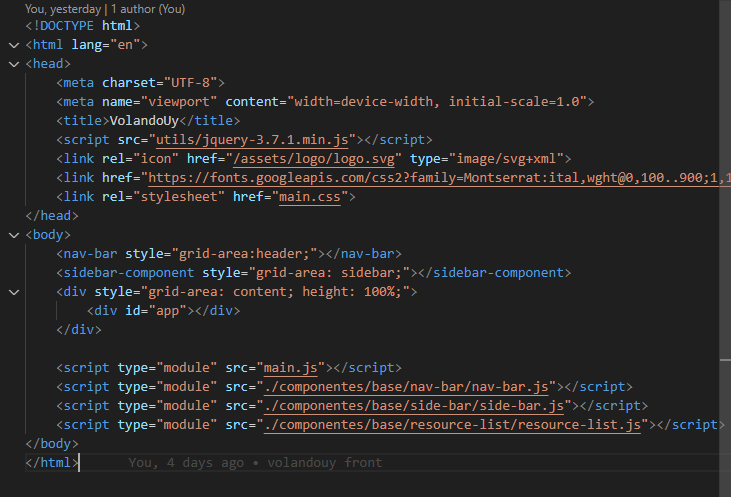
import { estados } from “./ruta/a/utils/index.js”

console.log(estados) //imprime [“aprobada”,”rechazada”,”pendiente”]

</script>

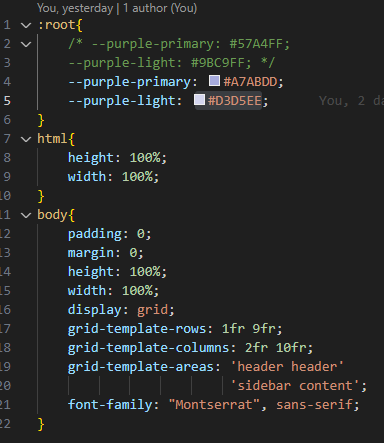
## index.html:

es el html que se carga base, si entran van a ver que carga la navbar (header), la sidebar, y deja el <div id=”app”> para que el router pueda cargar los datos:



## main.css:

ac[a dejamos los estilos generales de la app, y MUY IMPORTANTE definimos las vars con los colores que queramos, no me gusta que queden en muchos lados los mismos colores, si lo van a usar varias veces definan una var para que sea f[acil de cambiar:



## main.js:

este es el script que se ejecuta al entrar a la p[agina, por ahora lo [unico que hace es cargar la [ultima ruta y mandarte para ah[i:

# 

# Filtros

NO EST[AN TODAV[IA :C PERO CREO QUE VAN A FUNCAR PARECIDO A LOS PARAMS, Y VA A HACER UN NAVIGATE PARA RECARGAR LA P[AGINA O ALGO AS[I

# Formularios

NO HAY NING[UN EJEMPLO TODAV[IA, PERO TA, LO IMPORTANTE SER[IA USAR LOS MISMOS ESTILOS DE INPUT, CAPAZ ESOS PUEDEN IR EN EL MAIN.CSS Y QUE AL FINAL DE VALIDAR LOS DATOS Y ESO LO QUE TIENE QUE HACER EL FORM ES LLAMAR A UNA FUNCI[ON DEL CONTROLADOR QUE CORRESPONDA