07/10/2022

AGUITREN

BY: IÑIGO BRUK, GORKA URIARTE, UNAI DÍEZ, MI NOMBRE

INDICE

RETO1

En este proyecto, nuestra empresa colaborará con un equipo asociado con la rama de robótica. Este proyecto se centrará en una página web para los trabajadores de EuskoTren, con el fin de poder obtener información en tiempo real sobre el tranvía de Vitoria-Gasteiz.

Para ello el equipo de robótica nos dirá las variables que necesitamos y nosotros utilizaremos esas variables con el fin de poder implementarlo en la página web.

Para la planificación utilizaremos trello para comenzar y clickup para la programación de tiempo del proyecto. Con trello estaremos utilizando tarjetas con el fin de decir que es lo que tenemos que hacer y el estado en el que esta. Con clickup, organizaremos los tiempos realizados con las tarjetas que creamos en trello.

Para que todos los integrantes del equipo puedan tener acceso a todos los documentos que utilicemos, utilizaremos dos aplicaciones: GitHub, que será el principal uso, y Google drive, que servirá como copia de seguridad.
En relación a la comunicación utilizaremos la aplicación WhatsApp.

El objetivo con este proyecto es que los de robótica puedan con un clic o escribiendo probar lo que ellos estén implementando, cómoda y fácilmente a través de la página web.

DESARROLLO

Todos los documentos que necesitamos estarán en un repositorio GitHub llamado aguitren, en él hemos creado 6 ramas (1 para la parte del desarrollo,1 para la parte de testeo/hotfix+ 1 para cada integrante de este proyecto) + la principal llamada definitivo.

- maestro -> última versión de cada fase a la hora de implementar (main)
  - staging -> última versión para verificar antes de estar en definitivo
  - Desarrollo -> cada uno lo que está haciendo en su correspondiente rama
    - Bruk
    - Gorka
    - ameer
    - Unai

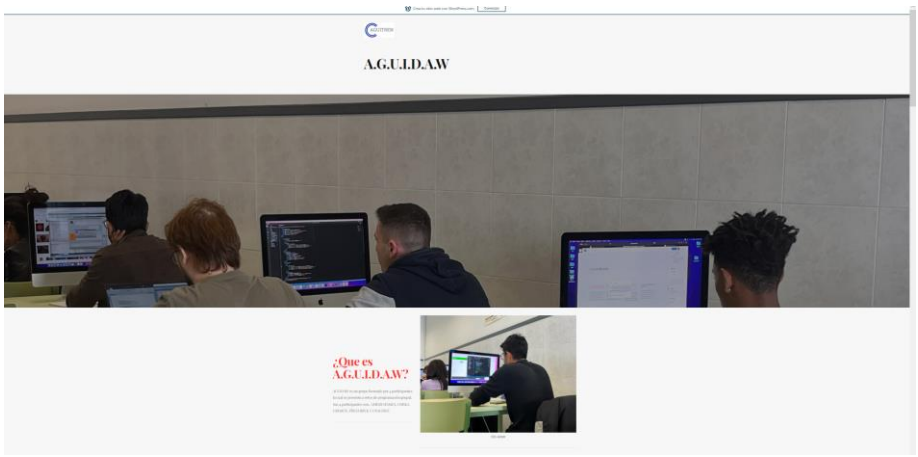
ESTRUCTURA DE CARPETAS

Lo primero que vemos al entrar en la carpeta es el index.html con varias carpetas. Esas carpetas están con el nombre donde guardaremos los correspondientes archivos.

- img: imagenes
- html: contenido de la página web
- css: hojas de estilos de la pagina
- js: programación javascript y JSON

WORDPRESS

Aparte de la estructura de carpetas, también hemos hecho un WordPress, con un pequeño inicio con un inicio y el logo. También tenemos implementado más contenido dentro del WordPress.



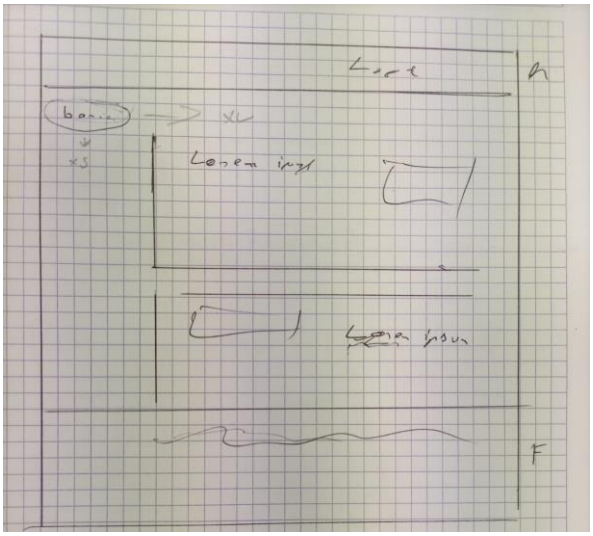
Para todo el contenido del nuestro WordPress se ha realizado un pdf con todos los detalles que incluye dicha página web a través del apartado del anexo.

PAGINA WEB

Para empezar con la página web, hemos hecho un pequeño esquema de cómo iba a ser las páginas web situada en varias pantallas diferentes.

ESTRUCTURA DE LA PAGINA WEB

Todas las páginas tienen un encabezamiento llamado “header” con el logo, título y una barra de navegación.



Abajo del header tiene el navegador y el main, el navegador siempre será el mismo mientras que el contenido del main dependerá de la página a la que se vaya.

Por último, está el footer tiene dos cosas sencillas:

- una lista básica con el logo de tres redes sociales
- un apartado para la suscripción a la newsletter

Para todo el contenido de las páginas de la aplicación del tranvía, se ha realizado un manual de usuario aparte con sus instrucciones insertadas dentro del anexo implementado al final del documento

TIPOGRAFIA Y PLANTILLAS DE ESTILOS

Para todo el contenido de la aplicación, dado a que solamente ellos pueden interactuar a través de un ordenador en la que se va a conectar, toda la página esta escalado e implementado a través de un grid, ocupando toda la pantalla:

```css
body{
    display: grid;
    height: 100vh;
    overflow-x: hidden;
    grid-template-columns: 100%;
    grid-template-rows: 1fr 10fr 1fr;
    grid-template-areas: 'nav' 'main' 'footer';
}
```

```css
*{
    margin: 0;
    padding:0;
    box-sizing: border-box;
}
```

JS: JAVA SCRIPT

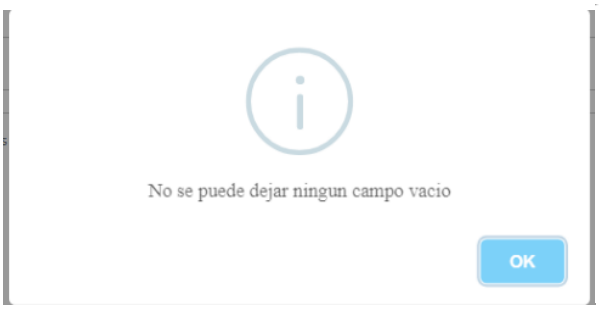
Para poder controlar de forma eficiente el entorno cliente de la página, hemos utilizado el lenguaje JavaScript para la posibilidad de controlar todos los eventos posibles dentro de la aplicación web.

Para el control total del js, hemos decidido realizar tres js diferentes: index.js, manual.js y recorrido.js.

En el index.js solamente recorre dos funciones dependiendo de la pagina web de donde este el usuario. Para la comprobación de ello, hemos puesto dos condiciones en las cuales, si detecta ciertos botones con una identificación exclusiva en la página, recoge esos botones e incluimos una función para dichos botones. En total hay dos funciones:

```js
if(document.getElementById("enviarcontacto")){
    document.getElementById("enviarcontacto"). addEventListener("click",enviarcontac);
}
if(document.getElementById("sesion")){
    document.getElementById("sesion"). addEventListener("click",iniciosession);
}
```

La primera función se llama “enviarcontac”, en la cual, a través de un formulario, se recogen todos los valores, se comprueban si existe contenido en cada una de ellas, e incluso si esta bien estructurado el correo, luego se avisa a través de una confirmación si se quiere enviar, y en caso de aceptar, se confirma el envío.




```
function enviarcontac(){
        let nombre=document.getElementById('nombre').value;
        let correo=document.getElementById('correo').value;
        let asunto=document.getElementById('asunto').value;
        let descripcion=document.getElementById('descripcion').value;
        let regNombre=new RegExp("^([\da-z_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$")
        console.log(nombre.length)
        if(nombre.length==0 || correo.length==0 || asunto.length==0 || descripcion.length==0 ){
            swal("No se puede dejar ningun campo vacio", {icon: "info",})
        }
        else{

            if (!regNombre.test(correo)){
                swal("El apartado de correo electronico no tiene la estructura de un correo", {icon: "info",}
            }
            else{
                swal({
                    title: "Estas seguro?",
                    text: "Una vez que le des a 'Si' se enviará",
                    icon: "warning",
                    buttons: true,
                    confirmButtonText: 'Si!',
                    cancelButtonText: 'No!',
                    dangerMode: true,
                })
                .then((willDelete) => {
                    if (willDelete) {
                    swal("Se ha enviado correctamente", {
                        icon: "success",
                    });
                    }
                });
            }
        }
}
```

La segunda función esta basado en un inicio de sesión normal, dado a que los de la robótica necesitan un inicio de sesión personal para entrar en sus aplicaciones, hemos insertado un control muy básico con una expresión regular al usuario y a la contraseña (dado a que solo tienen un usuario con contraseña), en la cual, si insertan correctamente, se les envía al index.html y se guarda como un "sessionStorage" el usuario y la contraseña. En caso opuesto se les saldrá una alerta personalizada de error en la cual desaparece al pasar cierto tiempo.

```
function iniciosession() {
    let nombre=document.getElementById ("user").value;
    regNombre=new RegExp("^admin$")
    let passwd=document.getElementById ('passwd').value;
    regPasswd=new RegExp("^12345$")
    if (regNombre.test(nombre) && regPasswd.test(passwd))
    {

       window.location="../index.html";
       window.sessionStorage.setItem("user", "admin")
    }
    else
        swal("Usuario o contraseña no validos", {icon: "error",timer: 5000,buttons: false})
}
```

En el apartado del recorrido.js principalmente insertamos una serie de variables para el control de las paradas y animaciones.

```
let abrirPuertas = document.getElementById("abrir-puertas");
let cerrarPuertas = document.getElementById("cerrar-puertas");

let puertaIzq = document.getElementById("puerta-izq");
let puertaDer = document.getElementById("puerta-der");

let parada1 = document.getElementsByClassName('parada1')[0];
let parada2 = document.getElementsByClassName('parada2')[0];
let parada3 = document.getElementsByClassName('parada3')[0];
let parada4 = document.getElementsByClassName('parada4')[0];
let parada5 = document.getElementsByClassName('parada5')[0];

let estado1 = document.getElementById("estado1");
let estado2 = document.getElementById("estado2");
let estado3 = document.getElementById("estado3");
let estado4 = document.getElementById("estado4");
let estado5 = document.getElementById("estado5");


let adelante = document.getElementById("adelante");
let emergencia = document.getElementById("emergencia");
```

```
puertaIzq.classList.add('abrir');
puertaDer.classList.add('abrir');

puertaIzq.style.animation = "abrirpuertas 1.5s";
puertaDer.style.animation = "abrirpuertas 1.5s";
let imgAnim;
```

```
function texto(response) {
    console.log(response);
  return response.text()
}

setInterval(function(){
    fetch('interfaz/leer_variable_base.htm', {
credentials: 'same-origin'
})
.then(texto)
.then(function(result) {
console.log(result)
let datos =result.split("|");
let variables = new Array();
for(i = 0; i < datos.length-1; i= i+2){
  variable = new Map()

  variable.set("nombre-variable", datos[i])
  variable.set("valor", datos[i+1])
  variables.push(variable)
}
console.log(variables)
})
.catch(function(error) {
console.log('request failed', error)
})  ;
},4000)
```

Al comienzo del js aparece una función que sirve para el control del recibo de datos y el control de respuesta en modo texto, dado a que lo que va a recibir una función es la respuesta de un fetch. Luego aparece un setinterval para la obtención de los datos de las variables del Tia portal, para poder implementarlo en la aplicación. También para ver todos los datos de la propia app hemos realizado un bucle en modo “Map” para comprobar el resultado.

PD: el lugar donde se realiza el fetch es un hmi cuya función es solamente para poder realizar la correcta sustitución de las variables del Tia portal con nuestra app a través de una línea.

En el botón de emergencia, hemos realizado una función anónima en la cual se para el tranvía, dando la oportunidad de que en caso de que se quiera reanudar el tranvía se pueda continuar desde el sitio donde se realizó la parada.

Al tocar el botón de emergencia también envía al servidor todas las acciones de cambio de la parada.

PD: la variable como esta en booleano, en el Tia portal solamente admite dos valores:

- 0=false
- 1=true

```
emergencia.addEventListener("click",function(){
    if(emergencia.innerText == "Emergencia"){
        imgAnim.pause();
        emergencia.innerHTML = "Reanudar";

        // actualizando la variable de emergencia
        fetch('automatic.html', {
            method: 'POST' ,
            body:"PE=" + 1
        })
        .then(res => {
            console.log(res)
        })
        .catch(err => {
            console.log(err)
        })

    }
    else
    {
        imgAnim.play();
        emergencia.innerHTML = "Emergencia";

        // actualizando la variable de emergencia
        fetch('automatic.html', {
            method: 'POST' ,
            body:"PE=" + 0
        })
        .then(res => {
            console.log(res)
        })
        .catch(err => {
            console.log(err)
        })
    }
}),
```

```
adelante.addEventListener("click",function(){
    let imgTren  = document.getElementsByClassName('img-train')[0];

    puertaIzq.classList.remove('abrir');
    puertaDer.classList.remove('abrir');

    puertaIzq.style.animation = "cerrarpuertas 1.5s";
    puertaDer.style.animation = "cerrarpuertas 1.5s";

    imgAnim = imgTren.animate([
        {right:'0.5vw'},
        {right:'100vw'}
    ],{
        duration:30000,
        delay:5000,
    });

    let imgStyles = window.getComputedStyle(imgTren);

    let p1Styles = window.getComputedStyle(parada1);
    let p2Styles = window.getComputedStyle(parada2);
    let p3Styles = window.getComputedStyle(parada3);
    let p4Styles = window.getComputedStyle(parada4);
```

También, se le envía la información de que esta arrancado el tranvía a través de un post.

En caso de tocar el botón de adelante, el tranvía se arranca y las puertas se cierran. Para ello, en el apartado de puertas, se elimina la clase 'abrir' y se implementa una animación para poder cerrar puertas e incluso se implementa una animación del tren para poder implementar simultáneamente con los de la robótica.

```
fetch('automatic.html', {
    method: 'POST' ,
    body:"ARRANQUE=" + 1
})
.then(res => {
    console.log(res)
})
.catch(err => {
    console.log(err)
})
```

También, dado a que se tiene que parar el tren en una parada, tenemos que indicar que en ciertos sitios se tiene que controlar la distancia del recorrido y si esta en un sitio especifico, parar el tranvía durante cierto tiempo y arrancar otra vez después del cierre. Esto se realiza llamando a varias funciones externas, y el intervalo es de 0.2 segundos.

```
setInterval(()=>{

    if( (parseInt(imgStyles.getPropertyValue('left')) < (parseInt(p1Styles.getPropertyValue('left')) +
    {
        actualizaDistancia(8000);
        pauseAnim(imgAnim,parada1);
        abrirP();
    }

    if( (parseInt(imgStyles.getPropertyValue('left')) < (parseInt(p2Styles.getPropertyValue('left')) +
    {
        actualizaDistancia(6000);
        pauseAnim(imgAnim,parada2);
        abrirP();
    }
    if( (parseInt(imgStyles.getPropertyValue('left')) < (parseInt(p3Styles.getPropertyValue('left')) +
    {
        actualizaDistancia(4000);
        pauseAnim(imgAnim,parada3);
        abrirP();
    }
    if( (parseInt(imgStyles.getPropertyValue('left')) < (parseInt(p4Styles.getPropertyValue('left')) +
    {
        actualizaDistancia(2000);
        pauseAnim(imgAnim,parada4);
        abrirP();
    }

},200);
```

Lo primero que pide es la actualización de la distancia que esta el tranvía. Por petición del equipo de robótica, dado a sus necesidades, necesitaban que el tranvía estuviese en ciertas distancias para la posibilidad de poder controlar de forma interna ciertas actividades, así que les enviamos a través de una variable con el método post dicho dato.

```
function actualizaDistancia(distancia)
{
    // actualizando la distanica
    fetch('automatic.html', {
        method: 'POST' ,
        body:"DISTANCIA=" + distancia
    })/*
    .then(res => {
       console.log(res)
    })*/
    .catch(err => {
        console.log(err)
    })
}
```

```
function pauseAnim(anim,parada)
{
    anim.pause();

    setTimeout(()=>{

        anim.play();
        cerrarP();

    },4000)
}
```

Luego pedimos que se pare el tranvía en la parada en la que está situada.

Además, tras pasar 4 segundos, se reanuda el tranvía y se cierran las puertas.

Para la realización de las puertas se han realizado dos funciones diferentes en las cuales se completan dos objetivos sencillos: abrir y cerrar las puertas.

Para ello, depende del objetivo de lo que se quiere realizar con las puertas, se añade o se elimina las clases ‘abrir’ y ‘cerrar’, teniendo una de las dos clases implementadas. Además, se inserta una animación llamado ‘abrir puertas’ o ‘cerrarpuertas’, que dura 1.5 segundos.

Después de ello, se le envía a través de un método post el estado de las puertas en valor booleano.

```
function abrirP()
{
    puertaIzq.classList.add('abrir');
    puertaDer.classList.add('abrir');

    puertaIzq.style.animation = "abrirpuertas 1.5s";
    puertaDer.style.animation = "abrirpuertas 1.5s";

    // abriendo puertas
    fetch('automatic.html', {
        method: 'POST' ,
        body:"PUERTAS=" + 1
    })
    .then(res => {
        console.log(res)
    })
    .catch(err => {
        console.log(err)
    })
}
```

```
function cerrarP()
{
    puertaIzq.classList.remove('abrir');
    puertaDer.classList.remove('abrir');

    puertaIzq.style.animation = "cerrarpuertas 1.5s";
    puertaDer.style.animation = "cerrarpuertas 1.5s";

    puertaIzq.classList.add('cerrar');
    puertaDer.classList.add('cerrar');

    // cerrando puertas
    fetch('automatic.html', {
        method: 'POST' ,
        body:"PUERTAS=" + 0
    })
    .then(res => {
        console.log(res)
    })
    .catch(err => {
        console.log(err)
    })
}
```

Para terminar, dado a que no todas las paradas estaran operativas en todo el momento, en la pagina web, al cambiar el estado de la parada, primero comprobamos si esta activo la parada en la cual se ha cambiado. Despues cojemos la parada y le enviamos el estado de la parava en valor de booleano.

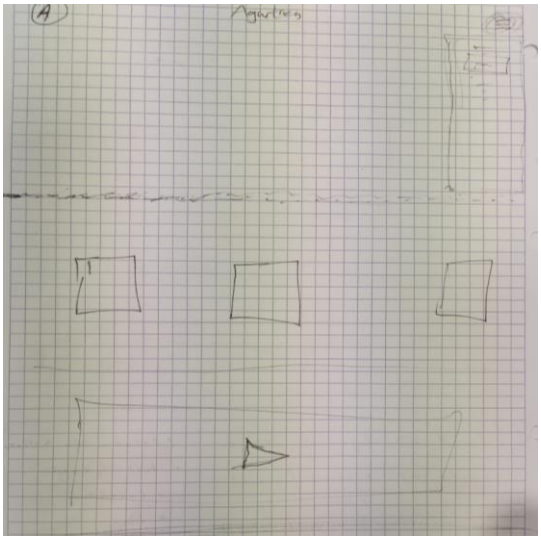
```
function cambiarEstado(event,parada)
{
    event.preventDefault();

    parada.classList.toggle("afuera-de-servicio");
    if(!parada.checked){ // enviar el pulso al servidor

        let nombreParada = "PARADA_" + parada.value;
        let resuelto=nombreParada+ "=" + 1;
 console.log(parada.value)
 console.log(nombreParada)
 console.log(resuelto)
        fetch('automatic.html', {
            method: 'POST' ,
            body: resuelto
        })
        .then(res => {
            console.log(res)
        })
        .catch(err => {
            console.log(err)
        })
    }
    else{
     let nombreParada = "PARADA_" + parada.value;
        let resuelto=nombreParada+ "=" + 0;
        fetch('automatic.html', {
            method: 'POST' ,
            body: resuelto
        })
        .then(res => {
            console.log(res)
        })
        .catch(err => {
            console.log(err)
        })
    }
}
```

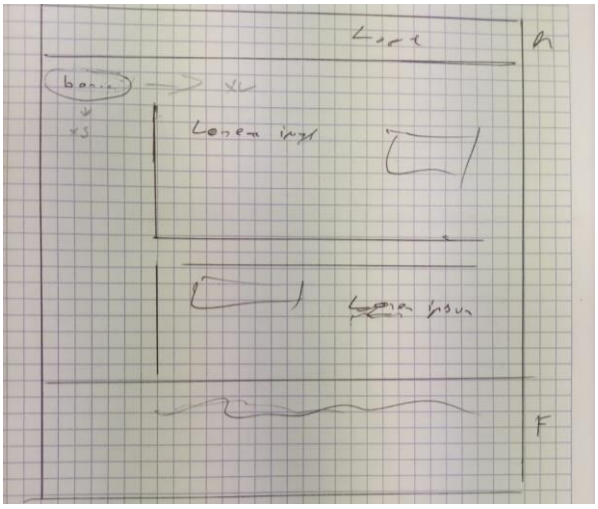
CONTROL DE OBJETIVOS

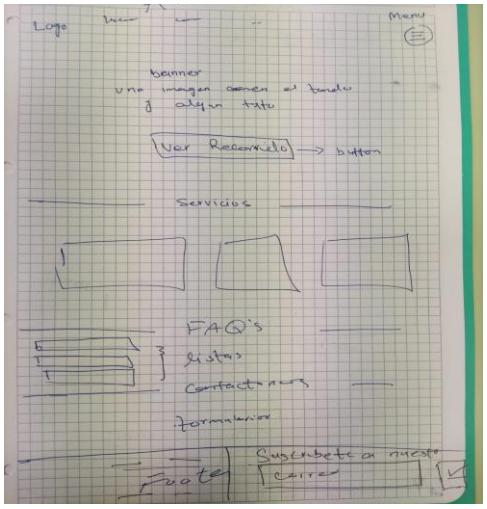
OBJETIVOS DE LOS PRIMEROS DIAS

Para este proyecto, cada uno realizo una plantilla básica para la aplicación web. El objetivo de las plantillas era realizar con unas ilustraciones básicas en papel para poder tener una visualización previa a nuestra página web.

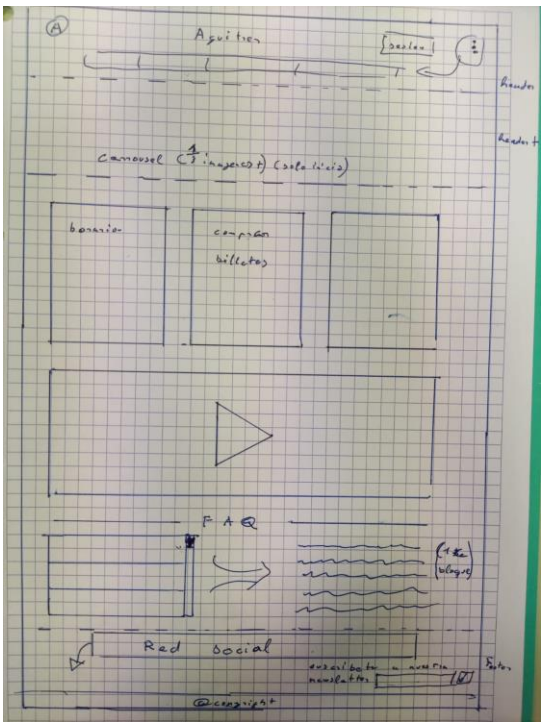
Estos son las imágenes:

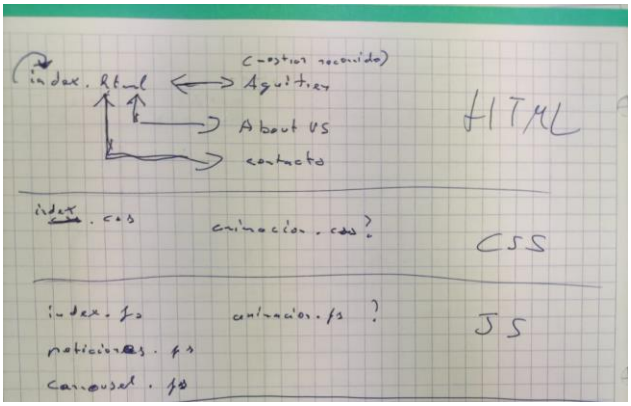




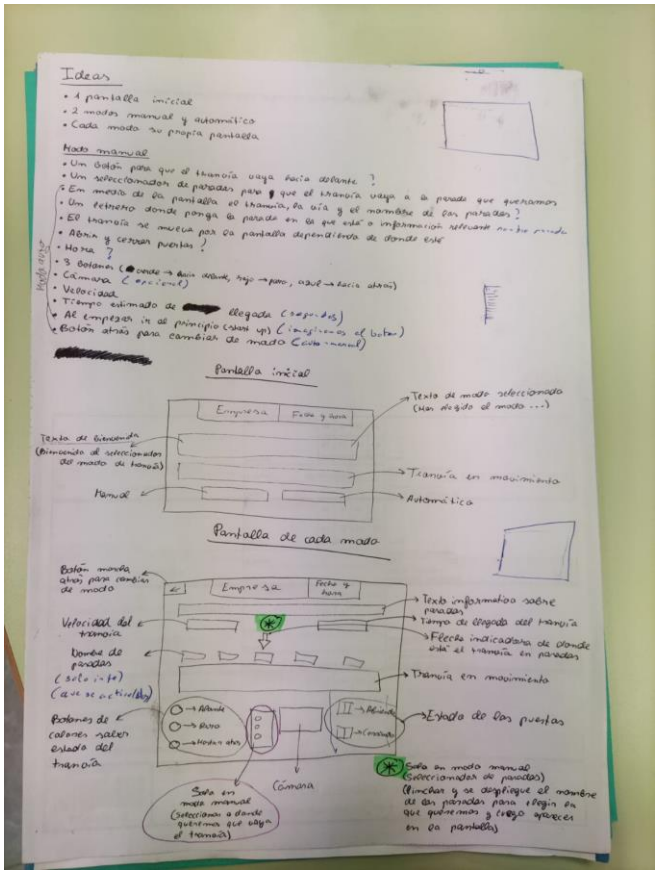


Despues hicimos uno más limpio entre todos con el objetivo de poder implementar todas las ideas y esquematizar todo el contenido, y realizar un árbol de todos los ficheros que tal vez necesitásemos para el correcto funcionamiento de la página:

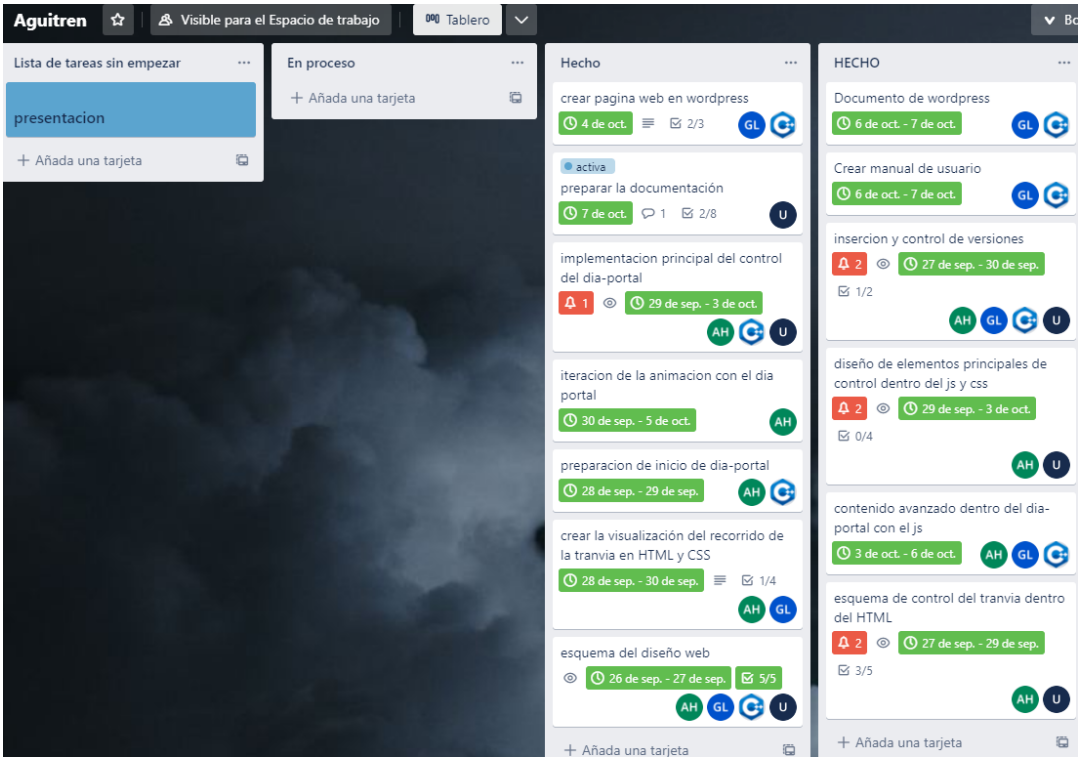




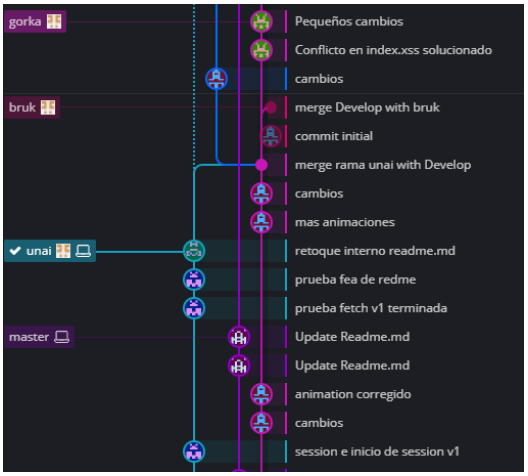
Aun así, hasta la primera reunión con los de robótica no hubo ningún inconveniente, pero reuniéndonos con ellos, tuvimos que retocar todos los apartados, pero los objetivos se quedaron muy claros.

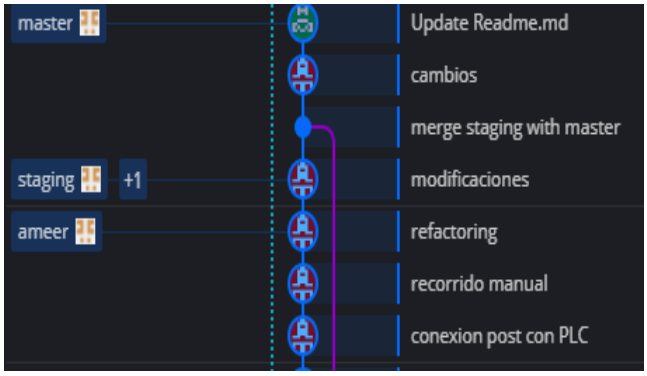


Con ello, pudimos planificar todo el contenido e implementar una planificación a través de trello. Así se quedó al terminar este proyecto:



Para poder tener un poco de control, todos los integrantes, con un plazo no mayor de 2 días, subíamos a nuestro repositorio todo lo que habíamos realizado,





CONCLUSION

Como en este reto hemos colaborado con el equipo de robótica, ahora comprendemos mejor como es trabajar con algún departamento externo a nosotros y poder comunicarnos en caso de configurar y comunicar en cualquiera de nuestros trabajos.

El control de las variables de siemens no es tan sencillo como parece y en el reto hemos tenido diversos problemas, uno de esos problemas fue a la hora de intentar controlar el contenido de una variable a través de un botón.

ANEXOS

WORDPRESS: https://fulloot.wordpress.com

DOCUMENTOS EXTERNOS:

- WordPress
- Manual de usuario

GitHub: https://github.com/dev-ameer-hamza/aguitren