



D-goals

Trabajo Final para la asignatura de Sistemas y
Tecnologías Web

Carolina Alvarez Martin: alu0100944723
Enrique Manuel Pedroza Castillo: alu0100886351

Trabajo Final para la asignatura de Sistemas y Tecnologías Web	1
Introducción	3
Objetivo	3
Tecnologías elegidas	3
Metodología	4
Prototipado	4
Paleta de colores	7
Implementación.	9
Servidor	9
Configuración Básica	9
Modelos	10
Peticiones	12
Login y Registro	12
update usuario y usuario	12
POST habit y GET habit	14
delete Habit	15
Base de datos	17
Servicios Angular	19
AuthService	19
Login	21
Singin	24
NewUser	25
New Habit	28
Home	29
Aplicaciones externas	35
Pivotal Tracker	35
Propuesta de Mejora	35

Introducción

Objetivo

El objetivo del proyecto es crear una aplicación que permita la gestión de los diferentes hábitos que un usuario pueda tener. Todo con el objetivo de una mejor gestión del tiempo disponible y poder ser más productivo

Tecnologías elegidas

Las tecnologías empleadas para poder llevar a cabo el proyecto son las siguientes:

1. **Angular**: para poder escribir tanto la base como el desarrollo del proyecto
2. **Nodejs**: para poder crear un servicio de escucha de las peticiones hechas por el cliente
3. **Nginx**: es un balanceador de carga que lo usamos como proxy inverso, de forma que al disponer de múltiples servidores este distribuirá la carga equitativamente entre estos
4. **Bootstrap**: es un CDN que usamos para diferentes componentes dentro del sitio web
5. **MongoDB**: es una base de datos no relacional que usaremos para cargar diferente información.

Metodología

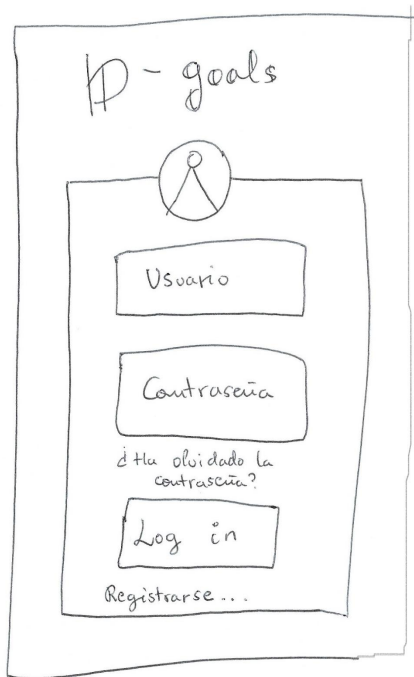
La metodología de trabajo que hemos usado durante el proyecto ha sido **SCRUM** de la siguiente manera:

- **Reunión de revisión de sprint** (Semanal)
 - Reunión de **retrospectiva** (¿Qué hemos hecho durante el sprint?): Hacemos un resumen del estado actual del proyecto y de los objetivos a realizar a continuación.
 - Reunión de **planificación** del siguiente sprint: Exponer y dividir trabajos planificados para el avance general del proyecto.
 - Las reuniones han sido realizadas a través de la plataforma **Google meet**.
- **Reunión diaria** (Temas más importantes del día) : Se ha usado la plataforma **Slack** para poder notificar al momento bien sea dudas o avances de cada integrante del grupo al resto de compañeros

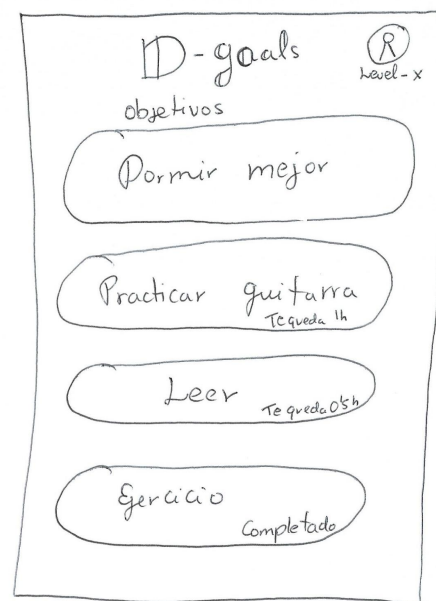
Prototipado

Una de las principales tareas que se decidió realizar, fueron los bocetos de nuestra aplicación, esto como orientación a lo que queremos que sea nuestra aplicación :

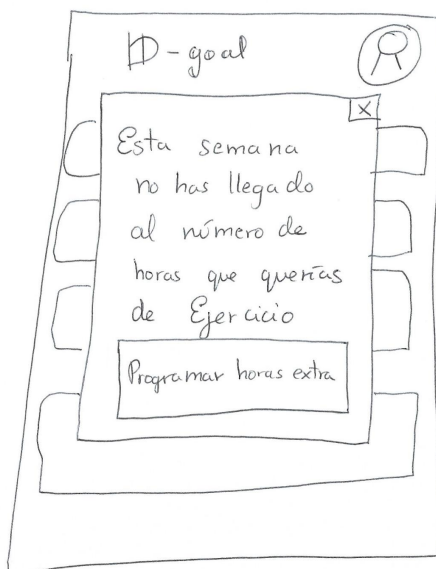
Vista de login



Vista de los hábitos a elegir



Vista de fallo en el hábito



Vista del resumen de los hábitos



Vista de calendario

Septiembre							sig. mes >
Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo	
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30	31					

Paleta de colores

A partir de este paso, la siguiente decision fue la eleccion de paletas de color, las candidatas trabajadas por el grupo fueron las siguientes:

Home About Services Gallery Contact

Iniciar Sesión

☒ Remember me

January 2017						
Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Level 1 > Level 2 > Level 3 > Level 4

Home About Services Gallery Contact

Iniciar Sesión

☒ Remember me

January 2017						
Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Level 1 > Level 2 > Level 3 > Level 4

nav 1

nav 2

nav 3

nav 4

nav 5

2020

JAN

Month

Year

< January 2017 >

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat

Link

Dropdown

...

Button

Elegimos utilizar la siguiente paleta de colores:

Home

About

Services

Gallery

Contact

Iniciar Sesión

Email

Password

☒ Remember me

Iniciar Sesión

Registrarse

< January 2017 >

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

12 May 2016

Hábito 1

Hábito 2

Hábito 3

Hábito 4

Level 1

>

Level 2

>

Level 3

>

Level 4

Implementación.

Servidor

Configuración Básica

En el lado del servidor se implementa un servicio escrito en **nodejs/javascript** que escuche por las peticiones a la base de datos y retorne los resultados. Como la base de datos está sobre un **Mongodb**, tendremos que incluir el módulo de **mongoose**, además de usar **express** para interpretar las peticiones.

```
const express = require('express');
const app = express();
app.use(bodyParser.json());
const bodyParser = require('body-parser');
```

En un fichero aparte que es donde estableceremos la configuración relativa a la base de datos llamado "**mongoose.js**", en el que tendremos que establecer como opciones el **UrlParser**, **UnifiedTopology** y las credenciales para poder acceder a la base de datos, en nuestro caso **user:usuario1**

```
const mongoose = require('mongoose');
const opciones = {useNewUrlParser:true,
  user:"user",
  pass:"usuario1",
  useUnifiedTopology: true};
```

Indicamos las sentencias para poder configurar globalmente la conexión a la base de datos, que sería a la ip **10.6.129.31** en la base de datos "**app_user**", en la que haremos una callback para determinar si se realiza la conexión con éxito o no:

```
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://10.6.129.31:8082/app_user', opciones).then(()=>{
  console.log("Conexión a la base de datos correcta");
}).catch((e)=>{
  console.log("Error al conectar con la base de datos");
  console.log(e);
});
```

Esta configuración se importará desde el fichero principal, además de establecer ciertas cabeceras para garantizar la compatibilidad con la plataforma en **Angular** :

```
const {mongoose} = require('./mongoose');

const {Usuario} = require('./models/index');
const {Habito} = require ('./models/index')

app.use(function (req, res, next) {

  // Website you wish to allow to connect
  res.setHeader('Access-Control-Allow-Origin', '*');

  // Request methods you wish to allow
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');

  // Request headers you wish to allow
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');

  // Set to true if you need the website to include cookies in the requests sent
  // to the API (e.g. in case you use sessions)
  res.setHeader('Access-Control-Allow-Credentials', true);

  // Pass to next layer of middleware
  next();
});
```

Modelos

Ahora tendremos que crear los modelos para las diferentes “tablas”, o mejor dicho, colecciones de la base de datos, para ello creamos un directorio llamado “**models**” en donde guardaremos todos los modelos que iremos creando:

```
server
├── models
│   ├── habit_model.js
│   ├── index.js
│   └── user_model.js
├── mongoose.js
├── petition.js
└── server.js
```

Como vemos de momento tenemos dos modelos, de hábitos y de usuarios, y un índice. Entraremos primero con los modelos, por ejemplo el de los hábitos:

```
const mongoose = require('mongoose');

const Schema_habito = new mongoose.Schema({
  nombre:{type: String,unique:true},
  descripcion:{type:String}
})

const Habito = mongoose.model('habitos',Schema_habito);
module.exports = {Habito}
```

Como vemos lo primero que hacemos es importar la librería de **mongoose**, posteriormente creamos el **Schema**, en este caso nos encontramos con un **nombre** que será de tipo **string** y será **único** (con lo cual no se podrá repetir). Insertamos el **Schema** dentro de la conexión a la base de datos y lo guardamos en una constante "**Hábito**", para finalmente exportarla.

La exportación irá hacia el **índice**, el **índice** es un fichero intermedio en el que tenemos todos los modelos centralizados, esto con el objetivo de facilitar la lectura del código y mantener un orden. Como veremos en el código, lo único que se hace es importar los modelos de los correspondientes ficheros y exportarlos:

```
const {Usuario} = require("./user_model");
const {Habito} = require ("./habit_model")
module.exports = {
  Usuario,
  Habito
}
```

Por último en el fichero principal del servidor importamos desde el índice:

```
const {Usuario} = require('./models/index');
const {Habito} = require ('./models/index');
```

Peticiones

Para resolver, el equipo ha decidido que utilizara la petición **POST**, para hacer las consultas. Para las peticiones tenemos las siguientes:

Login y Registro

Las peticiones a login o registro, son peticiones contrarias, en las que login simplemente hará una consulta sobre si un determinado usuario con una determinada contraseña existe dentro de la base de datos, sea cual sea el resultado retornarlo.

```
// Login de la aplicación
app.post("/login", (req, res) => {
  // guardamos los parametros de usuario y password
  let user = req.body.user;
  let pass = req.body.password;
  // hacemos la consulta segun el usuario y la pass, retornamos el perfil
  Usuario.find({nombre:user,password:pass}).then((lists) => {
    res.send(lists);
  }).catch((e) => {
    res.send(e);
  });
});
```

Por otro lado tenemos el Registro, en el que queremos ingresar un nuevo usuario, en el que se le ha pedido previamente una serie de información, para posteriormente insertarlo en la base de datos, de nuevo, sea cual sea el resultado se retorna.

```
app.post("/signin", (req, res) => {
  // Creamos el objeto de perfil nuevo, conforme a la informacion proporcionada por el formulario html
  let newUser = new Usuario({
    nombre : req.body.user,
    password : req.body.password,
    correo : req.body.email,
    schema_version:1.1
  });
  // Escribimos en la Base de datos el nuevo perfil
  newUser.save().then((listDoc) => {
    res.send(listDoc);
  })
});
```

update usuario y usuario

Tenemos peticiones para poder inicializar o bien cambiar las diferentes características del perfil de un usuario determinado. Empezaremos con usuario:

```
app.post("/usuario",(req,res)=>{  
  let user = req.body.user;  
  let nombre_habito = req.body.habit["nombre"];  
  let habit = { $set: {  
    habito:{  
      [nombre_habito]:{  
        dias:req.body.habit["dias"],  
        horario:req.body.habit["horario"],  
        horas:req.body.habit["horas"]  
      }  
    },  
    nombre:user  
  }  
};  
  
Usuario.find({nombre:user}).then((lists)=>{  
  res.send(lists);  
  let cuenta = lists[0];  
  Usuario.updateOne ({habitto:{vacio:true},nombre:user},habit,function(err,res){  
    if (err) throw err;  
    console.log("1 document updated");  
  });  
  res.send();  
}).catch((e)=>{  
  res.send(e);  
});  
});
```

Como podemos ver, es una petición que lo primero que hace es obtener las diferentes propiedades del cuerpo de la petición, en este caso, los días, el horario y las horas totales del hábito, posteriormente lo que hacemos es obtener la cuenta que tenga como username el usuario especificado en la petición y posteriormente establece el perfil como se espera.



```
app.post("/update_usuario",(req,res)=>{
  let user = req.body.user;
  let cambio = req.body.cambio;
  let valor = req.body.valor;
  let query = { $set:{
    [cambio]:valor,
  nombre:user
  }}
  Usuario.find({nombre:user}).then((lists)=>{
    res.send(lists);
    let cuenta = lists[0][cambio];
    Usuario.updateOne ({[cambio]:cuenta,nombre:user},query,function(err,res){
    if (err) throw err;
    console.log("1 document updated");
    });
    res.send();

  }).catch((e)=>{
    res.send(e);
  });
})
```

Posteriormente tenemos update usuario, que de forma similar al anterior, se sustituye la propiedad principal del perfil por una nueva, en vez de un hábito concreto.

POST habit y GET habit

De la misma forma que antes, creamos dos peticiones para poder o bien recoger todas las peticiones (**GET habit**) o bien para ingresar una nueva (**POST habit**)



```
app.post("/habit",(req,res)=>{
  let newHabit = new Habito({
    nombre : req.body.name,
    descripcion: req.body.description
  });

  newHabit.save().then((listDoc)=>{
    res.send(listDoc);
  })
})

// Petición GET para obtener los hábitos

app.get("/habit",(req,res)=>{
  Habito.find().then((lists)=>{
    console.log("entro")
    res.statusCode = '200';
    res.statusMessage = 'Solicitud realizada con éxito';
    res.send(lists);

    console.log(lists);
  }).catch((e)=>{
    res.statusCode = '500';
    res.send(e);
  });
})
```

delete Habit

Por último también hemos creado una petición en caso de que se quiera en futuros desarrollos eliminar un hábito concreto:



```
app.delete("/habit",(req,res)=>{  
  Habito.findOneAndRemove({  
    nombre: req.body.name  
  }).then((removed)=>{  
    res.send(removed);  
  })  
})
```


Base de datos

En la base de datos hemos decidido usar una base de datos llamada “App_user” que junta toda la información de los usuarios y de la aplicación. Dentro de esta base de datos tenemos dos colecciones “hábitos” y “usuarios”:

```
> show collections
habitos
usuarios
```

Si echamos un vistazo a la colección de usuarios, veremos que es donde están los perfiles de cada usuario:

```
{
  "_id" : ObjectId("60d3bf69f228e206036da91d"),
  "numero_de_horas_totales" : 0,
  "tareas_cumplidas" : 0,
  "racha" : 0,
  "nombre" : "tmp_",
  "password" : "Password22",
  "correo" : "asd@asdf.com",
  "schema_version" : 1.1,
  "habito" : {
    "dormir" : {
      "dias" : [
        "Lunes",
        "Martes",
        "Miercoles",
        "Jueves"
      ],
      "horario" : [
        "00:14",
        "13:00"
      ],
      "horas" : 0
    },
    "correr" : {
      "dias" : [
        "Lunes",
        "Martes",
        "Miercoles",
        "Jueves",
        "Viernes",
        "Sabado",
        "Domingo"
      ],
      "horario" : [
        "00:28",
        "13:00"
      ],
      "horas" : 0
    }
  },
  "__v" : 0
}
```

Dentro del perfil podemos ver que tal y como está dispuesto en el Modelo Usuario en el servidor de peticiones, está conformado básicamente por los siguientes elementos:

- `_id`: identificador único del objeto dentro de la colección
- `numero_de_horas_totales`: tal como su nombre indica es la suma de todas las horas invertidas en hábitos por parte del usuario
- `tareas_cumplidas`: es el computo total de hábitos y tareas cumplidas

- racha: es el conteo consecutivo de hábitos realizados, se pone a 0 si el usuario incumple alguno de sus hábitos programados
- nombre: nombre del usuario
- password: contraseña de la cuenta del usuario
- correo: Correo de contacto del usuario
- Schema_version: Es una etiqueta que indicará la versión del esquema de la base de datos para en caso de tener que cambiarlo poder identificar rápidamente aquellos que hayan sido creados con el esquema antiguo y poder actualizarlos.
- Hábito: es un objeto que guarda los diferentes hábitos a los que el usuario se ha suscrito

Dentro de Hábito nos encontramos que , a su vez, existe una serie de datos:

- nombre de hábito, será la clave hash con el que se podrá seleccionar el hábito en sí, un ejemplo sería “dormir”, o “estudiar”.
- días: incluye un vector con los diferentes días de la semana en las que está programada realizar el hábito.
- horario: un vector con las horas de inicio y de fin del hábito.
- horas: Total de horas del hábito concreto.

Por otro lado tenemos la colección “hábitos” que contiene, como su nombre indica, los hábitos que los usuarios pueden suscribirse:

```
> db.habitos.find().pretty()
{
  "_id" : ObjectId("6097c987a6af3c543a1dc514"),
  "nombre" : "dormir",
  "descripcion" : "vamos a dormir para descansar",
  "__v" : 0
}
{
  "_id" : ObjectId("6097c9f8a50fd255d6823ca3"),
  "nombre" : "correr",
  "descripcion" : "correr es una actividad sana que nos permite mejorar el metabolismo",
  "__v" : 0
}
{
  "_id" : ObjectId("60cafa8d88727736315483c8"),
  "nombre" : "estudiar",
  "descripcion" : "es importante para obtener mejores calificaciones",
  "__v" : 0
}
```

Como vemos está estructurado de la siguiente forma:

- _id: identificador único del objeto
- nombre del hábito.
- descripción breve de en qué consiste dicho hábito.

Servicios Angular

En nuestro proyecto angular tenemos diferentes componentes :

AuthService

Es un recurso de typescript que usaremos para la gestión del inicio de sesión mediante del localStorage del navegador y del uso de los modelos para poder gestionar la sesión de un perfil autenticado.

Lo primero que hacemos dentro del fichero es, de forma global crear tres variables, la primera sería una instancia de “User” definida en el modelo typescript dentro del directorio __models:

```
export class User {  
    public username: string;  
    public password: string;  
    public email: string;  
    public token?: string;  
    constructor(user:string,pass:string,mail:string,token:string){  
        this.username=user;  
        this.password=pass;  
        this.email=mail;  
        this.token=token;  
    }  
    public get_json(){  
        return JSON.stringify({  
            username: this.username,  
            password:this.password,  
            email:this.email,  
            token:this.token  
        });  
    }  
}
```

Luego creamos una variable observable pública del modelo “User”, y por último un BehaviorSubject de “User” también. Dentro del constructor lo que hacemos es iniciar todas las variables dentro del navegador

```
private currentUserSubject: BehaviorSubject<User>;
public currentUser: Observable<User>;
model = new User('', '', '', '');

// En el constructor de la clase creamos la cookie almacenada localmente
constructor(private http: HttpClient) {
  localStorage.setItem('currentUser', this.model.get_json());
  this.currentUserSubject = new BehaviorSubject<User>(JSON.parse(this.model.get_json()));
  this.currentUser = this.currentUserSubject.asObservable();
}
```

Lo siguiente que nos encontramos es un método login:

En el que con la información recogida en un formulario haremos una petición al servidor y este nos devolverá el resultado, en caso de que exista alguna cuenta con las credenciales nos permitirá continuar al home de la página y guardar el perfil dentro del localStorage.

```
login(username:string,password:string){
  // Retornamos el resultado del POST, y preparamos para escuchar en la ruta /login

  let data =JSON.stringify({
    user: username,
    password: password});

  const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })}

  return this.http.post<any>(`http://10.6.130.59:8081/login`,data,httpOptions).subscribe(data =>{
    // console.log(data[0]);
    if(data[0]){
      let datos = data[0];
      let resultado = new User(datos["nombre"],datos["password"],datos["correo"],datos["_id"]);
      localStorage.setItem('currentUser',JSON.stringify(resultado));
      this.currentUserSubject.next(resultado);
      return resultado;
    }
    else{
      $('#error').addClass("show")
      $('#error').removeClass("ocultar");
      return 0;
    }
  }, error => {
    console.log(JSON.stringify(error.json()));
  });
}
```

Posteriormente nos encontramos con un método de registro o “singin”, en el cual enviamos la información al servidor para que cree la cuenta con lo que hemos recogido

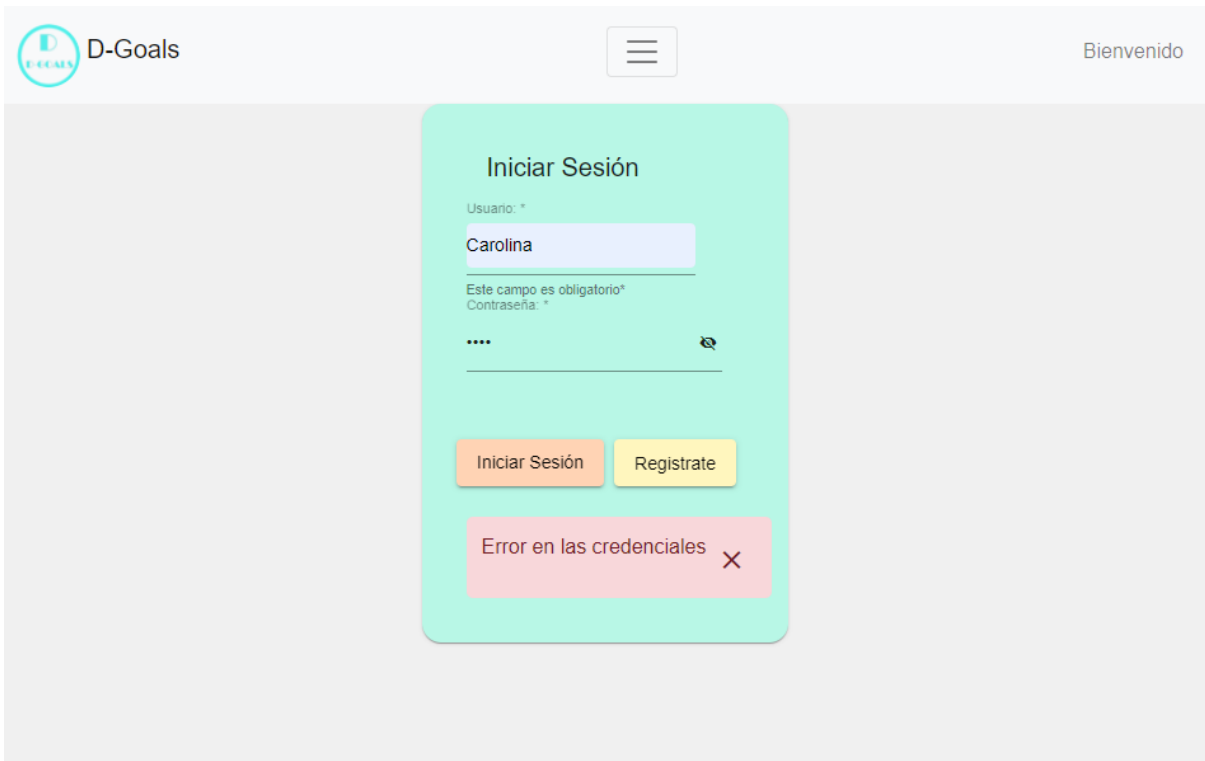
```
singin(username:string,password:string,correo:string){  
  // Retornamos el resultado del POST, y preparamos para escuchar en la ruta /login  
  
  let data =JSON.stringify({  
    user: username,  
    password: password,  
    email:correo});  
  
  const httpOptions = {  
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })}  
  
  return this.http.post<any>(`http://10.6.130.59:8081/singin`,data,httpOptions).subscribe(data =>{  
    console.log(data);  
  }, error => {  
    console.log(JSON.stringify(error.json()));  
  });  
}
```

Por último nos encontramos el método logout que sirve para cerrar la sesión y eliminar la cuenta del localStorage.

```
// Elimina la cuenta del usuario <aka cookie>  
logout(){  
  localStorage.removeItem('currentUser');  
  // this.currentUserSubject.next();  
}  
current_user(){  
  
  return this.currentUserSubject;  
}  
// Getter de usuario de la sesion  
public get currentUserValue(): User{  
  return this.currentUserSubject.value;  
}
```

Login

Posteriormente nos encontramos con el componente Login-form que incluye un formulario para el inicio de sesión mediante templates y elementos de angular como mat-form-field o mat-form-input. En este formulario nos encontramos con el botón Inicio de Sesión deshabilitado hasta que se haya escrito el nombre de usuario y la contraseña. Si las credenciales no son correctas se mostrará una alerta debajo de esta.



The screenshot shows the D-Goals login interface. At the top left is the D-Goals logo, and at the top right is the text "Bienvenido". In the center is a light blue rounded rectangle containing the "Iniciar Sesión" form. The form has two input fields: "Usuario: *" with the value "Carolina" and "Contraseña: *" with masked characters "....". Below the fields is a red error message "Error en las credenciales" with a close icon. Two buttons, "Iniciar Sesión" (orange) and "Registrate" (yellow), are at the bottom of the form.

D-Goals Bienvenido

Iniciar Sesión

Usuario: *

Carolina

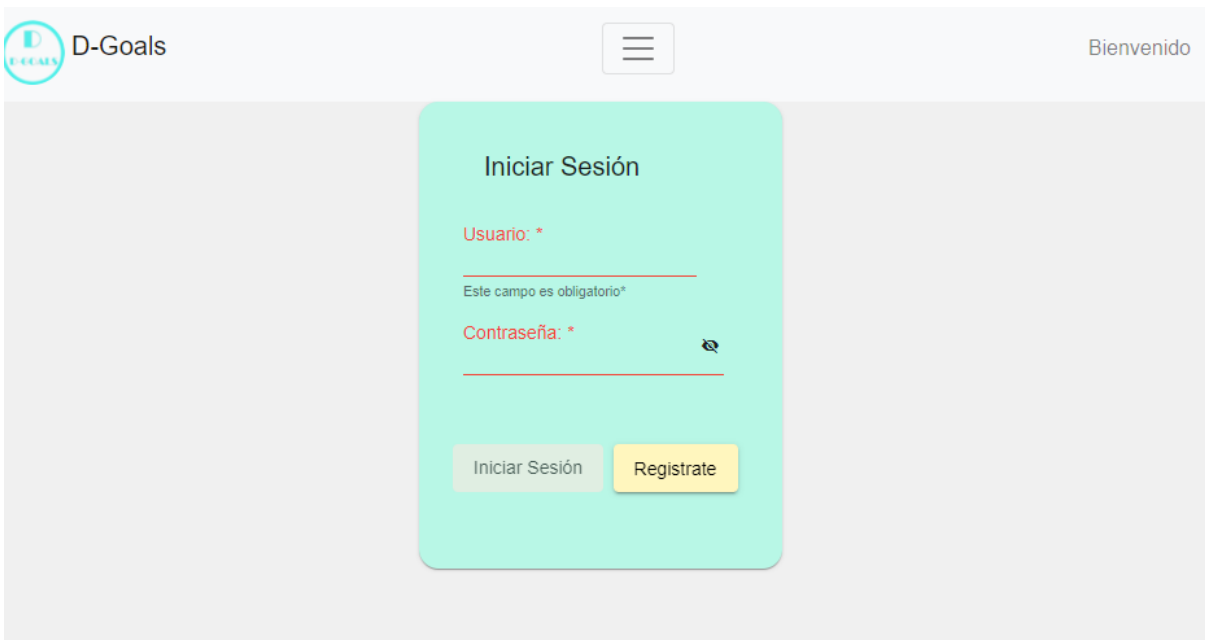
Este campo es obligatorio*

Contraseña: *

....

Iniciar Sesión Registrate

Error en las credenciales X



This screenshot shows the same D-Goals login interface, but with validation errors. The "Usuario: *" field has a red error message "Este campo es obligatorio*" below it. The "Contraseña: *" field also has a red error message "Este campo es obligatorio*" below it. The "Iniciar Sesión" button is now disabled and greyed out, while the "Registrate" button remains yellow and active.

D-Goals Bienvenido

Iniciar Sesión

Usuario: *

Este campo es obligatorio*

Contraseña: *

Este campo es obligatorio*

Iniciar Sesión Registrate

En el código componente login nos encontramos con lo siguiente :

```
constructor( private http: HttpClient,private router: Router ) { }

ngOnInit(): void {
  $('#error').addClass("ocultar")
}
```

De esta forma lo que hacemos es ocultar un mensaje de error nada más cargar la página, posteriormente nos encontramos con la declaración de diferentes variables y un método onSubmit, que sirve cuando el usuario quiere enviar la información de los formularios, recoge dicha información y los pasa al método login de auth_services, para poder iniciar la sesión local, por último vamos al método hacer petición

```
//model = new User("Carol","1234567123","", '');
model = new User('','', '', '');
autenticacion = new AuthService(this.http);
submitted = false;

onSubmit() {
  this.submitted = true;
  var username = (<HTMLInputElement>document.getElementById("name")).value;
  var password = (<HTMLInputElement>document.getElementById("password")).value;
  this.autenticacion.login(username,password);
  this.hacerPetición();
}
```

En el método hacerPetición, lo que hacemos es obtener el perfil cargado y en caso de que exista, ir a /home de la página. Por otro lado tenemos un eventListener que escuchar en caso de que el usuario haga click para poder ocultar o remover clases.

```
hacerPeticion(){
  var tmp = this.autenticacion.current_user();
  tmp.pipe(skip(1)).subscribe(
    (value)=>{
      this.router.navigate(["/home"]);
    }
  );
}

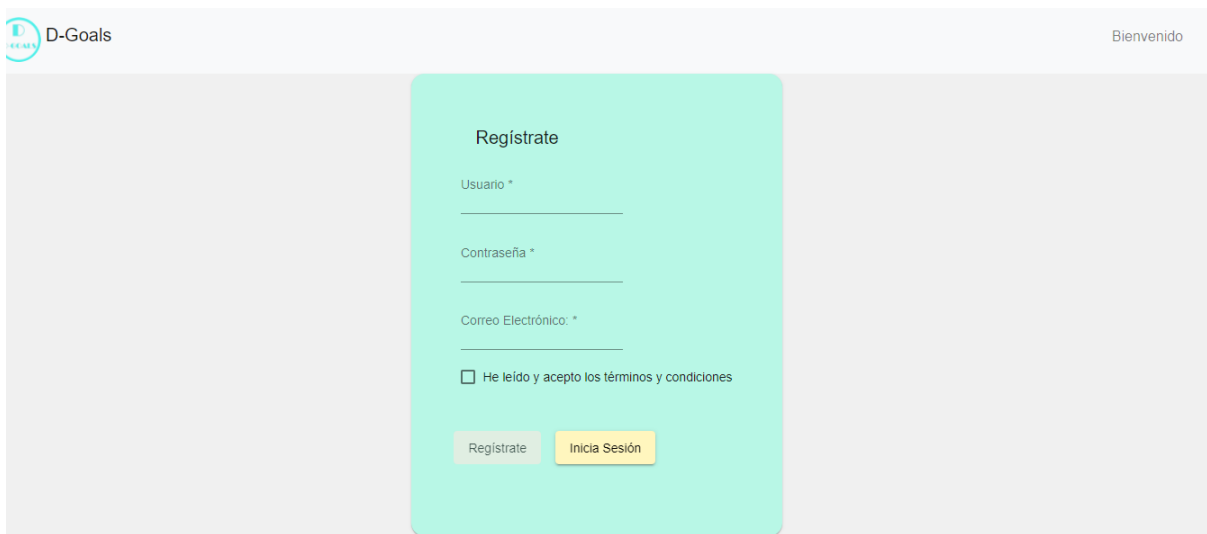
ocultarAlerta(event?: MouseEvent){
  console.log("hi")
  $('#error').removeClass("show");
  $('#error').addClass("ocultar");
}
```

Singin

Después nos encontramos con el componente Signin que incluye el formulario para el registro. En este formulario el botón de Registro está deshabilitado hasta que se cumplan los requisitos siguientes:

- Más de 4 caracteres en el campo del nombre de usuario.
- Mínimo una mayúscula, una minúscula y un número en el campo de contraseña.
- 8 caracteres mínimo para el campo contraseña.
- Estructura de email para el campo de correo electrónico. (ejemplo@ejemplo).
- Marcar el checkbox de los términos de uso.

También se mostrará una alerta si al hacer la petición de registro al servidor este devuelve que el usuario ya se encuentra registrado.



Dentro del componente de registro nos encontramos a algo parecido al de login, como vemos en la siguiente imagen, seguimos disponiendo del método onSubmit y del HacerPetición para poder confirmar que la sesión ha sido iniciada.

```
ngOnInit(): void {  
  $('#error').addClass("ocultar");  
}  
  
model = new User("", "", "", "");  
autenticacion = new AuthService(this.http);  
submitted = false;  
  
onSubmit() {  
  this.submitted = true;  
  console.log(this.model)  
  this.hacerPetición()  
}  
  
hacerPetición(){  
  let resultado = this.autenticacion.singin(this.model.username, this.model.password, this.model.email);  
  console.log(this.model);  
}
```

Por último nos encontramos con mensajes de alerta, que de igual forma, se muestran al hacer click sobre algún elemento html de la página

```
ocultarAlerta(event?: MouseEvent){  
  console.log("hi")  
  $('#error').removeClass("show");  
  $('#error').addClass("ocultar");  
}
```

NewUser

Después tenemos los componentes de Newuser y NewHabit en los que tenemos una serie de tarjetas con los hábitos disponibles y cuando hacemos click en uno de ellos nos muestra un cuadro de diálogo con un formulario para la inclusión de ese hábito en la base de datos:



Es la página que nos mostrará tan pronto el sitio web detecte que nuestra cuenta es nueva, en ese caso nos pedirá por un primer hábito y lo agregará a nuestro perfil. Para ello haremos lo siguiente:

Por razones de estilo el grupo ha decidido establecer una distribución de los elementos por columnas, esto lo hacemos gracias a Typescript ya que lo haremos en forma de dos columnas a través de un grid de Bootstrap, y separamos los hábitos en pares e impares según nos lo devuelva la base de datos mediante una petición al servidor node.

```
this.http.get<any>(`http://10.6.130.59:8081/habit`, httpOptions).subscribe(data =>{
  console.log(data);
  for(var x=0; x< data.length; x++){
    var container = document.createElement("div");
    container.classList.add("container");

    document.getElementById("").classList.add("")

    var div_imagen = document.createElement("div");
    div_imagen.classList.add("text-center");
    var imagen = document.createElement("img");
    imagen.src="../../assets/Square_200x200.png";
    imagen.classList.add("rounded");

    var div_titulo = document.createElement("div");
    var titulo = document.createElement("h1");
    let element = data[x];
    titulo.textContent = element["nombre"];
    titulo.classList.add("titulo");
```

```
var div_descripcion = document.createElement("div");
var descripcion = document.createElement("p");
descripcion.textContent=element["descripcion"];
descripcion.classList.add("descripcion");

div_descripcion.appendChild(descripcion);
div_titulo.appendChild(titulo);
div_imagen.appendChild(imagen);

container.appendChild(div_imagen);
container.appendChild(div_titulo);
container.appendChild(div_descripcion);

if(x%2==0){
    izquierdo?.appendChild(container);
}
if(x%2==1){
    derecho?.appendChild(container);
}
container.addEventListener("click",this.seleccion,false);
};
return data;
```

Lo siguiente que encontraremos será un método que funciona como un “eventlistener” de cuando el usuario haga clic en guardar el hábito programado, haciendo eso se registrará el nombre del hábito, los días de la semana que está programada y las horas en las que se pondrá en práctica el hábito, todo para posteriormente guardarlo en el perfil mediante una petición al servidor:

```
guardarHabit(event?: MouseEvent){
  var hora_inicio =(<HTMLInputElement>document.getElementById("start")).value;
  var hora_fin = (<HTMLInputElement>document.getElementById("end"))?.value;
  var dias = document.getElementsByClassName("form-check-input");
  var rango_dias = [];
  var horario = [];
  horario.push(hora_inicio);
  horario.push(hora_fin);
  for (let i = 0; i < dias.length; i++) {
    var dia=dias[i] as HTMLInputElement;
    if(dia.checked ==true){
      rango_dias.push(dias[i].id);
    }
  }
  let nombre_habito = document.getElementById("exampleModalLabel")?.textContent;
  let habit_elegido = {
    nombre: nombre_habito,
    dias: rango_dias,
    horario: horario,
    horas:0
  }
  let perfil =JSON.parse( localStorage.getItem("currentUser"));
  console.log(perfil["username"]);
```

```
let data =JSON.stringify({
  user: perfil["username"],
  habit: habit_elegido
});

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })}

return this.http.post<any>(`http://10.6.130.59:8081/usuario`,data,httpOptions).subscribe(data =>{
  this.router.navigate(["/home"]);
return data;
})
}
```

Y por último tenemos un método, selección, que se dispara al momento de que el usuario elige un método, mostrando una ventana emergente que está previamente declarado en HTML, esperando a que el usuario guarde para poder ejecutar el método anterior, guardarHabit.

New Habit

En new Habit lo que hacemos es algo parecido que en new-user en cuanto a cómo mostramos la página al usuario, por ello los métodos ngOnInit y selección son iguales. El contenido diferente que nos encontraríamos sería en guardar hábito, pues lo que hacemos es guardar dentro de nuestro perfil el habit nuevo que habíamos elegido o bien modificar **solo** los dias de la semana en los que hacemos el hábito o bien el rango de hora en el que lo hacemos:

```

////////// agregar nueva tarea a la cuenta , hay que modificarla
let perfil =JSON.parse( window.localStorage.getItem("currentUser"));
let data =JSON.stringify({
  user: perfil["username"],
  password: perfil["password"]
});
const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })}

this.http.post<any>(`http://10.6.130.59:8081/login`,data,httpOptions).subscribe(data =>{

  if(data[0]["habito"][nombre_habito]){
    var habito = data[0]["habito"][nombre_habito]
    habito["dias"]=rango_dias;
    habito["horario"]=horario;
    let datos =JSON.stringify({
      user: perfil["username"],
      cambio: "habito",
      valor:data[0]["habito"]
    });
    this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{
    }
  }
  else{
    var habito = data[0]["habito"];
    habito[nombre_habito] = {
      "dias":rango_dias,
      "horario":horario,
      "horas":0
    }
    let datos =JSON.stringify({
      user: perfil["username"],
      cambio: "habito",
      valor:data[0]["habito"]
    });
    this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{
    }
  }
}

```

Home

A continuación tenemos el componente Home, en el que se incluye el componente de calendario y una lista de las tareas a realizar para hoy:


D-Goals
¡Hola Carolina!

Jun 2021

Domingo	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Mis actividades de hoy

dormir
02:36-17:00
✓ x

correr
01:36-13:00
✓ x

Agregar nuevo habito

En el componente lo primero que hacemos es cargar el perfil del usuario, y en caso de que exista hacemos una petición para poder cargar el perfil y poder guardarlo:

```

ngOnInit(): void {
  var tmp = window.localStorage.getItem("currentUser")
  if(tmp){
    var perfil = JSON.parse(tmp);
    let username = perfil["username"].toString();
    let password = perfil["password"].toString();
    this.getPerfil(username,password);
  }
}

```

El siguiente método que nos encontramos es getPerfil, que recibe un username y una password por parámetro, para poder posteriormente crear la petición al servidor:

```

getPerfil(username:string,password:string){
  let data =JSON.stringify({
    user: username,
    password: password});

  const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })}

  return this.http.post<any>(`http://10.6.130.59:8081/login`,data,httpOptions).subscribe(data =>{

```

En caso de que exista un hábito y no este vacío, lo que haremos será cargarlos y crear un código html inyectado desde el propio typescript para que muestre al usuario los diferentes hábitos que tienen para el propio día en el que se encuentre mientras usa la aplicación:

```
this.Perfil= data[0]["habito"];
if(this.Perfil["vacio"]!=true){
  Object.keys(this.Perfil).forEach(key => {
    let dias = this.Perfil[key]["dias"];
    var today = new Date().getDay();
    if(dias.includes(this.number_to_day(today))) {
      var actividades = document.getElementById("actividades");
      var fila = document.createElement("div");
      var nombre = document.createElement("div");
      var inicio = document.createElement("div");
      var fin = document.createElement("div");
      var hecho = document.createElement("div");
      var no_hecho = document.createElement("div");

      fila.classList.add("row");
      nombre.classList.add("col");
      inicio.classList.add("col");
      fin.classList.add("col");
      hecho.classList.add("col-1");
      no_hecho.classList.add("col-1");
      hecho.classList.add("done");
      no_hecho.classList.add("fail");
      nombre.textContent = key;
      inicio.textContent = this.Perfil[key]["horario"][0];
      fin.textContent = this.Perfil[key]["horario"][1];
      hecho.textContent = "V";
      no_hecho.textContent = "X";
    }
  });
}
```

```
hecho.addEventListener('click',()=>{
  this.tarea_cumplida(nombre,inicio,fin);
});
no_hecho.addEventListener('click',()=>{
  this.tarea_no_cumplida();
});
fila.appendChild(nombre);
fila.appendChild(inicio);
fila.appendChild(fin);
fila.appendChild(hecho);
fila.appendChild(no_hecho);
actividades.appendChild(fila);
}
})
}
```

Otros métodos que tenemos dentro del documento typescript de home es `number_to_day`, esto es debido a que hacemos uso de “date”, que usa un formato numérico y en nuestro proyecto consideramos que necesitamos el valor en letras, con lo cual hicimos una correspondencia entre el número y un string con el nombre del día:

```
number_to_day(numero:Number){
  switch(numero){
    case 0:{return "Domingo";}
    case 1:{return "Lunes";}
    case 2:{return "Martes";}
    case 3:{return "Miercoles";}
    case 4:{return "Jueves";}
    case 5:{return "Viernes";}
    default:{return "Sabado";}
  }
}
```

Otro método que hemos implementado es “`tarea_cumplica`”, recibe por parámetro la siguiente información:

- Tarea: es la tarea que el usuario ha cumplido

- inicio: la hora de inicio programada para el hábito
- fin: la hora de fin programada del hábito
- elemento: es el elemento html padre , que apunta al hábito dentro de la lista

Con toda esta información lo que haremos es que, en caso de que el usuario cumpla con un hábito, nos interesa saber primero cual es el hábito que ha completado, luego cuanto tiempo corresponde a ser cumplido dicho hábito y por último eliminar el hábito de la lista de hoy.

```
tarea_cumplida(tarea:HTMLDivElement,inicio:HTMLDivElement,fin:HTMLDivElement,elemento:HTMLDivElement){
  tarea.parentNode.parentNode.removeChild(elemento)
  var tiempo_i = inicio.textContent;
  var tiempo_f = fin.textContent;
  // console.log(tiempo_i.substring(3,5));
  var tmp_i = Number(tiempo_i.substring(0,2));
  var tmp_f = Number(tiempo_f.substring(0,2));

  var tiempo_total = tmp_f-tmp_i;

  tmp_i =Number(tiempo_i.substring(3,5))
  tmp_f = Number(tiempo_f.substring(3,5))

  var decimas = Math.abs((tmp_f - tmp_i)/60);
  tiempo_total= tiempo_total+decimas;
  // console.log(tiempo_total);
}
```

Una vez disponemos de toda esta información, hacemos una petición al servidor, para que agregue los cambios en la base de datos. Para ello lo primero que hacemos es obtener nuestra cuenta de usuario, y cargaremos el estado actual del hábito en cuanto al conteo de horas totales:

```
return this.http.post<any>(`http://10.6.130.59:8081/login`,data,httpOptions).subscribe(data =>{
  var habito = data[0]["habito"][tarea.textContent]
  habito["horas"]+=tiempo_total;
  // console.log(data[0]["habito"]);

  let datos =JSON.stringify({
    user: cuenta["username"],
    cambio: "habito",
    valor:data[0]["habito"]
  });
  this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{});
});
```

Lo siguiente es actualizar el valor de las horas totales y el número de las tareas cumplidas:

```
data[0]["numero_de_horas_totales"]+=tiempo_total;
datos =JSON.stringify({
  user: cuenta["username"],
  cambio: "numero_de_horas_totales",
  valor:data[0]["numero_de_horas_totales"]
});
this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{});

data[0]["tareas_cumplidas"]+=1;
datos =JSON.stringify({
  user: cuenta["username"],
  cambio: "tareas_cumplidas",
  valor:data[0]["tareas_cumplidas"]
});
this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{});
```

y por último actualizar la racha de tareas seguidas cumplidas:

```
data[0]["racha"]+=1;
datos =JSON.stringify({
  user: cuenta["username"],
  cambio: "racha",
  valor:data[0]["racha"]
});
this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{});
```

Por otro lado, el usuario también puede no cumplir con el hábito, en dicho caso lo que haremos es modificar la racha a 0, puesto que no hay horas que agregar o conteo que sumar al total registrado.

```
tarea_no_cumplida(tarea:HTMLDivElement,elemento:HTMLDivElement){
  const cuenta = JSON.parse(window.localStorage.getItem("currentUser"));
  tarea.parentNode.parentNode.removeChild(elemento)
  let data =JSON.stringify({
    user: cuenta["username"],
    password: cuenta["password"]});

  const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })}

  return this.http.post<any>(`http://10.6.130.59:8081/login`,data,httpOptions).subscribe(data =>{
    var racha_broken = 0

    let datos =JSON.stringify({
      user: cuenta["username"],
      cambio: "racha",
      valor:racha_broken
    });
    this.http.post<any>(`http://10.6.130.59:8081/update_usuario`,datos,httpOptions).subscribe(data =>{});

  }, error => {
    console.log(JSON.stringify(error.json()));
  });
}
```

Por último tenemos el método que usamos para queremos crear un nuevo hábito, a través del evento onClick de un botón, este lo que hará es simplemente dirigirnos a la página donde agregamos los diferentes hábitos:

```
crearHabit(event?: MouseEvent){
  this.router.navigate(["/newhabit"]);
}
```

Aplicaciones externas

Pivotal Tracker

Hemos utilizado Pivotal Tracker para el seguimiento de las tareas a realizar. En un principio el grupo estaba formado por tres miembros. Dividiendo las tareas en tres bloques fundamentales:

- Estilos
- Diseño de página
- Scripts

Finalmente se dividieron las tareas de forma que había dos grupos de desarrollo, uno centrado en estilos y el otro en scripts, y a medida de que cada grupo avanza implementa parte del código de la estructura de la página a implementar.

De esta forma el avance en el proyecto era rápido dado a que la colaboración por parte de las distintas partes del equipo entre sí, estaba orientada a optimizar los recursos disponibles.

Propuesta de Mejora

Como propuesta de mejora, el equipo tiene pensado usar en versiones futuras el calendario para poder mostrar la carga de hábitos en los diferentes días del mes, así como poder desplazarnos en los diferentes meses anteriores para poder ver las estadísticas de hábitos cumplidos y no cumplidos, así como ver los de días venideros para poder visualizar los hábitos programados.

Dentro de cada casilla del calendario se tiene pensado agregar, un icono que funcione a modo de semáforo que nos indique la carga de hábitos en un día, siendo por ejemplo, el color verde teniendo pocos hábitos, el color amarillo algunos hábitos y en rojo muchos hábitos programados.

Otra parte a mejorar es que se puede sustraer estadísticas de los diferentes hábitos de forma individual. Tal y como se ve en los bocetos, viendo la evolución de cumplimiento y la disciplina por parte del usuario para mostrar de forma gráfica el avance del mismo.

Por último, agregar una vista en la que los propios usuarios puedan proponer hábitos nuevos que no estén listados por parte de la administración del sitio web. De esta forma el resto de usuarios se verían beneficiados al poder descubrir nuevos hábitos en su vida.