

Proyecto final

El proyecto final consistirá en la realización de una Web-App o SPA basada en JavaScript, HTML y CSS. No se permite el uso de ningún framework JavaScript, aunque se recomienda hacer uso de algún framework CSS que facilite realizar la estructura de las páginas (Bootstrap, Foundation, Skeleton, Materialize...).

En caso de usar alguna librería, se recomienda hacer uso de algún framework para gestionar dependencias (Bower, NPM...).

La SPA deberá contar al menos 5 páginas, de las cuales una de ellas deberá ser un login, que deberá permitir recordar contraseña. Si no hemos iniciado sesión no podremos ir a ningún parte dentro de la App.

Todas las páginas (exceptuando el login) deberán hacer uso de servicios REST para cargar sus datos.

Parte 1

Como se ha comentado en la introducción, no se permite el uso de ningún framework JavaScript, así que vamos a tratar de hacer nuestro propio framework:

Analiza las necesidades del proyecto y piensa una estructura de proyecto lo más flexible posible. Deberás generar:

- 1) Un sistema de “navegación”: puedes cambiar la URL de la página en la que te encuentras sin necesidad de recargar la página con:

`window.history.pushState("Datos a enviar", "Nuevo título", "/nueva-url");`
- 2) Un sistema de páginas: puedes gestionar que el cambio de URL esté asociado al cambio de “página”. Realmente no tendremos páginas, pero simularemos el comportamiento cambiando el contenido de nuestra página. Haz que la página sea una clase y que tenga un método de pintado. Puedes usar herencia dentro de las páginas y hacer por ejemplo un tipo de páginas con Cabecera y Footer y otras que no tengan.
- 3) Una clase App o Main que se encargue de orquestar toda la APP.
- 4) Una clase APIClient que se encargue de realizar todas las peticiones HTTP necesarias, recuerda que debe ser transparente y no debe “conocer” la estructura ni las rutas de la API que vayas a usar.

- 5) Una clase `xClient` que sea el cliente de tu API (el nombre dependerá de la API que sea accedida). Esta clase será la responsable de conocer las rutas de tu API y modelará los objetos de respuesta.
- 6) Una pantalla o capa de Loading que pueda ser activada o desactivada. De manera que cuando realicemos alguna petición HTTP nos impida hacer click cualquier botón de la pantalla.

Parte 2

Nuestras páginas necesitarán acceder a APIs o realizar una navegación por la App. Por lo que se puede decir que tienen dependencias externas.

Haz que estas dos dependencias sean inyectadas en la construcción de nuestras páginas.

Tendremos una clase/objeto `NavigationController` dentro de nuestra App que será el encargado de realizar la navegación. Debería tener los métodos:

- `NavigateToUrl(string)`
- `NavigateToHome()`

Por otro lado tendremos las clases `ComidaClient` y `BebidaClient` encargadas de acceder a nuestras apis de Comida y Bebida.

Por otro lado, para hacer uso de las rutas mencionadas en el apartado será necesario que tengamos un servidor de ficheros (Apache o Express). En la última página de este documento se explica cómo crear un servidor Express.

Parte 3

Crea las siguientes páginas:

1. Login: Nos debe permitir hacer login en nuestra APP. Tendrá dos campos: usuario y contraseña, además de un campo para recordar los datos del usuario. Si el usuario introduce un usuario y una contraseña correctos pasará a la HOME.
2. Crear Cuenta de Usuario: En esta página podremos crear una cuenta de usuario. Deberá haber un formulario con todos los datos necesarios para la creación de un usuario.
3. Home: Realiza un dashboard con gráficas y un resumen de dato de la API. Por ejemplo, un gráfico con las calorías de las comidas o un gráfico que indique el porcentaje de bebidas alcohólicas.
4. Gestión de Comidas: Mostrará una tabla con todas las comidas de la API. Deberá tener una columna de opciones: Ver, Editar y Borrar que nos permita

operar con cada una de las comidas. Además, tendremos un botón añadir para crear nuevas comidas.

5. Gestión de Bebidas: Mismo funcionamiento que Gestión de Comidas pero con Bebidas.
6. Perfil de usuario: En esta página se mostrarán los datos del usuario y se permitirá modificarlos o incluso borrar el usuario.

Información de la API

Comidas

Listado de comidas:

GET a <http://formacion-indra-franlindebl.com/api/comidas>

Creación de una nueva comida:

POST a <http://formacion-indra-franlindebl.com/api/comidas>

Con parámetros:

```
{  
  "tipo": "Entrante",  
  "precio": 40,  
  "calorias": 50,  
  "existencias": 3,  
  "nombre": "Empanada de Carne"  
}
```

Edición de una comida:

PUT a http://formacion-indra-franlindebl.com/api/comidas/ID_ELEMENTO

Con parámetros:

```
{  
  "tipo": "Entrante",  
  "precio": 40,  
  "calorias": 50,  
  "existencias": 3,  
  "nombre": "Empanada de Carne"  
}
```

Borrado de una comida:

DELETE a http://formacion-indra-franlindebl.com/api/comidas/ID_ELEMENTO

Campos de Objeto Comida:

nombre

Nombre: nombre

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

Otras restricciones: Debe ser único

existencias

Nombre: existencias

Tipo: Number

Valor máximo: 1000

Valor mínimo: 0

Requerido: Si

calorías

Nombre: calorías

Tipo: Number

Valor máximo: 5000

Valor mínimo: 0

Requerido: Si

precio

Nombre: precio

Tipo: Number

Valor máximo: 10000

Valor mínimo: 0

Requerido: Si

tipo

Nombre: tipo

Tipo: String

Longitud máxima: no aplica

Longitud mínima: no aplica

Requerido: Si

Otras restricciones: Debe ser uno de los siguientes valores: ['Entrante', 'Principal', 'Postre'];

Bebidas:

Listado de bebidas:

GET a <http://formacion-indra-franlindebl.com/api/bebidas>

Creación de una nueva bebida:

POST a <http://formacion-indra-franlindebl.com/api/bebidas>

Con parámetros:

```
{  
  "grados": 0,  
  "esAlcoholica": false,  
  "precio": 150,  
  "calorias": 10,  
  "existencias": 10,  
  "nombre": "CocaCola Zero Sin Azúcar",  
}
```

Edición de una bebida:

PUT a http://formacion-indra-franlindebl.com/api/bebidas/ID_ELEMENTO

Con parámetros:

```
{  
  "grados": 0,  
  "esAlcoholica": false,  
  "precio": 150,  
  "calorias": 10,  
  "existencias": 10,  
  "nombre": "CocaCola Zero Sin Azúcar",  
}
```

Borrado de una comida:

DELETE a http://formacion-indra-franlindebl.com/api/bebidas/ID_ELEMENTO

Campos de Objeto Bebida:

nombre

Nombre: nombre

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

Otras restricciones: Debe ser único

existencias

Nombre: existencias

Tipo: Number

Valor máximo: 1000

Valor mínimo: 0

Requerido: Si

calorías

Nombre: calorias

Tipo: Number

Valor máximo: 5000

Valor mínimo: 0

Requerido: Si

precio

Nombre: precio

Tipo: Number

Valor máximo: 10000

Valor mínimo: 0

Requerido: Si

esAlcoholica

Nombre: esAlcoholica

Tipo: Boolean

Longitud máxima: no aplica

Longitud mínima: no aplica

Requerido: Si

grados

Nombre: grados

Tipo: Number

Valor máximo: 100

Valor mínimo: 0

Requerido: Si

Usuarios:

Listado de usuarios:

GET a: <http://formacion-indra-franlindebl.com/api/users>

Creación de un nuevo usuario:

POST a: <http://formacion-indra-franlindebl.com/api/users>

Con parámetros:

```
{  
  "email": "franlindebl@gmail.com",  
  "apellidos": "Linde Blázquez",  
  "nombre": "Fran",  
  "username": "franlindebl",  
  "password": "12345678"  
}
```

Edición de un usuario:

PUT a http://formacion-indra-franlindebl.com/api/users/ID_ELEMENTO

Con parámetros:

```
{  
  "email": "franlindebl@gmail.com",  
  "apellidos": "Linde Blázquez",  
  "nombre": "Fran",  
  "username": "franlindebl",  
  "password": "12345678"  
}
```

NOTA: La password será utilizada para comprobar que el usuario puede modificar ese usuario. No es un campo editable en esta operación.

Borrado de un usuario:

DELETE a: http://formacion-indra-franlindebl.com/api/users/ID_ELEMENTO

Con parámetros:

```
{  
  "password": "12345678"  
}
```

Login con un usuario:

POST a: <http://formacion-indra-franlindebl.com/api/users/login>

Con parámetros:

```
{  
  "username": "franlindebl",  
  "password": "12345678"  
}
```

En caso de login OK devolverá un 200 y la información del usuario.

En caso de fallo devolverá un error 401.

Campos de Objeto Usuario:

username

Nombre: username

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

Otras restricciones: Debe ser único

password

Nombre: password

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

nombre

Nombre: nombre

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

apellidos

Nombre: apellidos

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

email

Nombre: email

Tipo: String

Longitud máxima: 100

Longitud mínima: 4

Requerido: Si

Pasos para crear un servidor con express:

1) Si no tenemos uno, debemos generar el fichero package.json de nuestro proyecto:

```
npm init
```

2) Instalamos dependencia express

```
npm install express --save
```

3) Configuramos una tarea "start" en nuestro package json que levate node con nuestro serve.js, también debemos setear el puerto como parámetro:

```
{
  "name": "test-express",
  "version": "1.0.0",
  "description": "Prueba servidor con Express",
  "main": "index.js",
  "config": {
    "port": "8080"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node serve.js"
  },
  "keywords": [],
  "author": "Fran Linde",
  "license": "UNLICENSED",
  "dependencies": {
    "express": "^4.15.3"
  }
}
```

4) Creamos nuestro fichero serve.js en la raíz del proyecto:

```
var express = require('express');
var app = express();
app.use("/", express.static(__dirname + '/'));
app.listen(process.env.npm_package_config_port);
```

5) Para lanzar el servidor debemos ejecutar en la consola:

```
npm run start
```