

# Práctica 1 - React.js - IWEB

## Introducción y Objetivos

### Requisitos:


- Nociones básicas de las tecnologías explicadas en clase (HTML, CSS, JS, **React**)
- Editor de código Visual Studio Code (o similar)
- Tener instalado [node.js](https://nodejs.org/) versión 16 (LTS) y [git](https://git-scm.com/)

### Motivación:

Intenta mirar esta práctica como un primer contacto con React y las tecnologías relacionadas, pero en la que se acabará haciendo una aplicación real y completamente funcional. Piensa en este enunciado como en una definición de una captura de requisitos resultados de varias reuniones con un cliente que es “el que paga”. Nos han pedido hacer esta aplicación con estas características mínimas y estos requisitos. Y para que nos paguen (y por lo tanto obtener el 10/10) debe cumplir con estas especificaciones y para ello nos pasan una batería de tests que debe pasar.

### Objetivo:

El objetivo de esta práctica es realizar una Single Page Application (SPA) de el tiempo. Esta aplicación tendrá dos campos en los que el usuario introduce latitud y longitud y un botón que al ser pulsado realizará una query a un servidor (API) y mostrará los resultados.

 Bienvenido a la página de Enrique Barra

### El tiempo





Latitud:

Longitud:

Buscar

Timezone: Europe/Madrid

El tiempo en los próximos días será:

30/8/2022	31/8/2022	1/9/2022	2/9/2022
			
Temp: 29.27°C	Temp: 29.09°C	Temp: 28.93°C	Temp: 26.83°C
Humedad: 32%	Humedad: 18%	Humedad: 15%	Humedad: 23%
Viento: 6.58m/s	Viento: 6.04m/s	Viento: 5.72m/s	Viento: 6.46m/s

## Primeros pasos: Preparación del entorno de desarrollo

Para poder hacer esta aplicación se proporciona un esqueleto o scaffold en github con el resultado de ejecutar el comando “npx create-react-app P1\_el\_tiempo” y añadir la herramienta de autocorrección y los tests a pasar para superar esta práctica.

Se recomienda al alumno a experimentar a crear sus propias aplicaciones fuera del repositorio de esta práctica desde cero con el comando “npx create-react-app mi\_nueva\_aplicacion” que creará una aplicación esqueleto de React “limpia”.

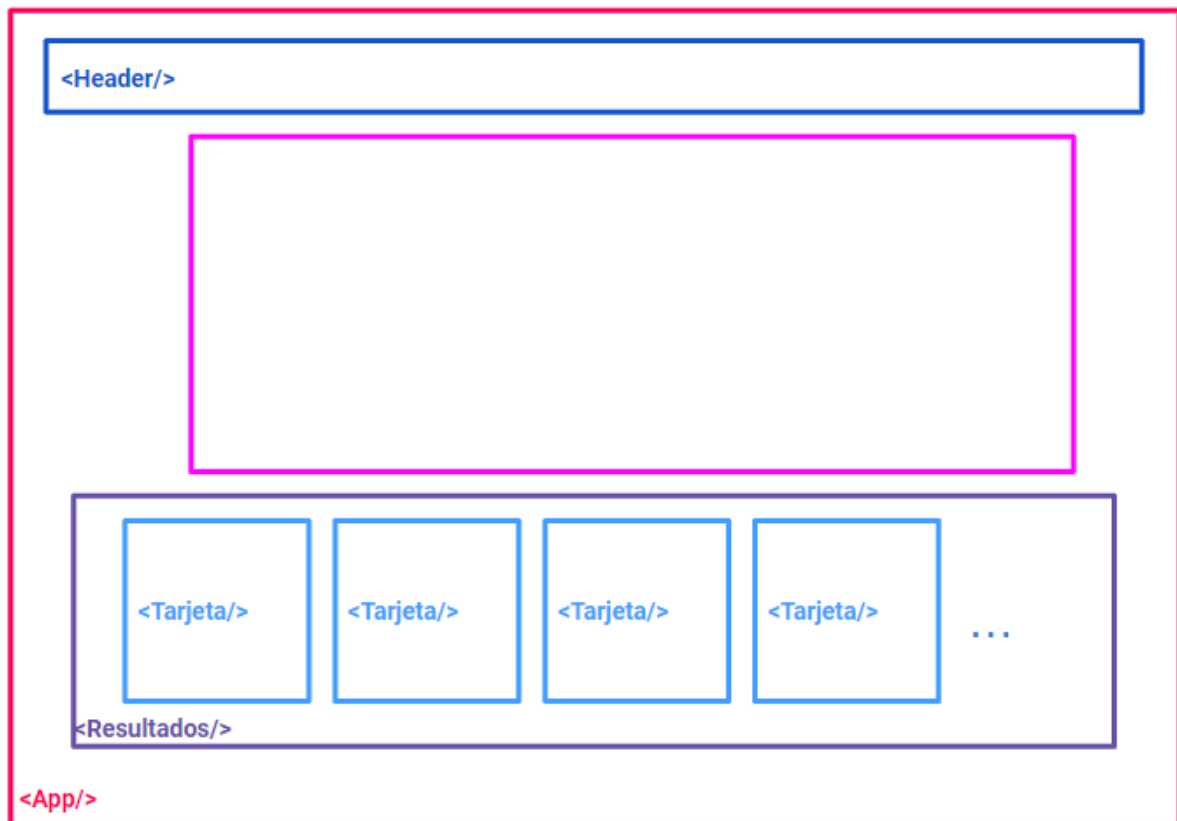
1. Instala [node.js](#) y [git](#) sino no lo has hecho ya
2. Clona el esqueleto de la aplicación a tu ordenador: `git clone https://github.com/IWEB-UPM/P1_el_tiempo.git`
3. Entra en el directorio creado: `cd P1_el_tiempo`
4. Instala los paquetes necesarios que indica el package.json: `npm install`
5. Inicia el servidor de desarrollo: `npm start`

A partir de ahora, si abrimos el navegador en la URL <http://localhost:3000>, podemos visualizar la aplicación en tiempo real mientras desarrollamos. Cuando queramos parar el desarrollo basta con hacer `Control + C` en el terminal.

Ahora hay que abrir la carpeta P1\_el\_tiempo en el editor Visual Studio Code y empezar a desarrollar nuestra aplicación.

## Comenzando a desarrollar

Antes de empezar a programar, uno debe pensar cómo estructurar la aplicación, y decidir *grosso modo* qué componentes va a necesitar y la jerarquía de los mismos. Un posible diseño es el siguiente:



En esta práctica los componentes App, Header y Resultados se tienen que llamar así por definición. Lo que está en el cuadro rosa puede renderizarlo App directamente o hacer un componente “Buscador” con ese contenido y el componente Tarjeta se puede llamar como se quiera.

También se recomienda definir el estado de nuestra aplicación (aunque sea una primera aproximación que luego irá mutando según añadamos funcionalidades). ¿Qué **información** necesito para pintar esta aplicación?

## API de el tiempo

En esta aplicación utilizaremos un API real disponible en <https://openweathermap.org/api>. Pero a la hora de desarrollar aplicaciones que “atacan” APIs hay que utilizar unos datos fake para ir haciendo pruebas con datos controlados y luego cuando tengamos todo desarrollado implementamos la llamada al API y refinamos lo que falte. Para este propósito se provee en el proyecto de github en la carpeta `src/constants` un fichero js que exporta un mock de los datos. Examine dicho fichero para ver el formato de los datos, por ejemplo tendrá que obtener de él la predicción del tiempo para los próximos días (campo “daily”), la zona horaria (“timezone”), etc.

La documentación del API está en <https://openweathermap.org/api/one-call-api> donde como vemos nos explica la sintaxis de las llamadas al API, los valores que pueden tener los parámetros y la respuesta.

Ejemplo de llamada al API:

<https://api.openweathermap.org/data/2.5/onecall?lat=40.416775&lon=-3.703790&appid=813560a5cd9fd811b4564db03ee31671> (podéis probar a poner esa URL

en un navegador o a usarla como primera llamada al API “a pincho”, posteriormente esta URL habrá que componerla con la latitud, longitud y el appid adecuado).

## Configuración de la aplicación (fichero config.js)

Las aplicaciones suelen tener una configuración que le indica los parámetros por defecto con los que carga, datos de servidores, urls, número de vidas inicial y power-ups en el caso de juegos, etc. Muy importante que esta configuración si contiene las KEYS o claves privadas, estas no se suban a los repositorios porque si no cualquiera podría verlas. En ese caso se pone un placeholder y se sustituye en local.

Para esta aplicación se provee un fichero en el proyecto de github en la carpeta *src/config* llamado *config\_default.js* que exporta una configuración inicial propuesta. El primer paso que tendremos que hacer es copiarlo en nuestra copia local donde pondremos el API key oficial. Esto se puede hacer con el editor de código Visual Studio Code o con un comando:

```
cp src/config/config_default.js src/config/config.js
```

A continuación abrir dicho fichero config.js y sustituir <YOUR\_API\_KEY> por la KEY provista:

```
813560a5cd9fd811b4564db03ee31671
```

Esta key la he solicitado yo para el developer plan siguiendo el plan educativo que proponen. Os podéis registrar en OpenWeatherMap con vuestro email de alumno @alumnos.upm.es y solicitar una key vuestra propia en la url:

<https://openweathermap.org/our-initiatives/student-initiative>

En el fichero config\_default.js (y por lo tanto en el config.js) se exporta una constante CONFIG que habrá que importar y que tiene un json con varios valores, el server\_url para componer la llamada al API, default\_lat y default\_lon para usarlos como placeholder en los input, un atributo use\_server que se debe usar primero a false para probar con mock data y posteriormente cambiar a true para empezar a usar el API real y num\_items que son el número de días de predicción que debe mostrar la aplicación.

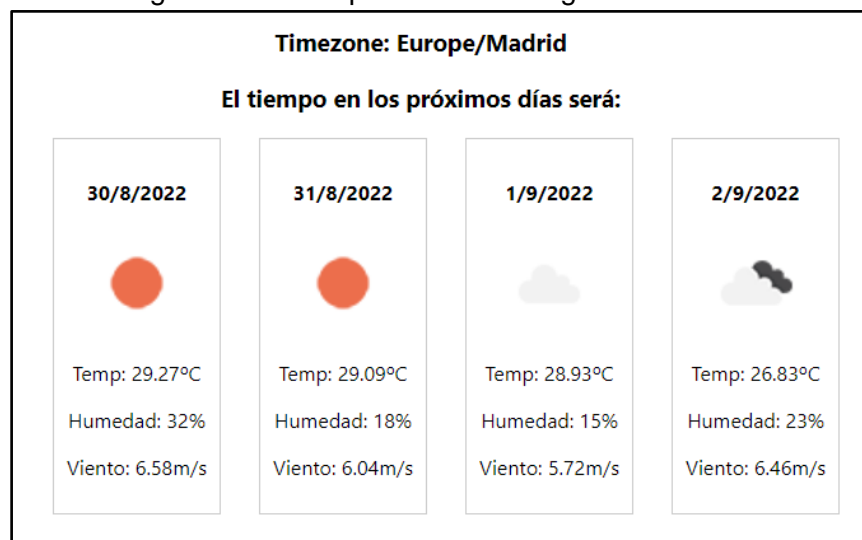
## Requisitos que debe cumplir nuestra aplicación

La práctica se valorará sobre un máximo de 10 puntos. La mayoría de los requisitos son de funcionalidad, para la interfaz de usuario puede maquetar como desee, se recomienda usar flexbox para posicionar los elementos <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> y css3 para dar estilos variados.

Requisitos:

- **1 punto:** La aplicación tiene un componente Header con el logo y el mensaje de bienvenida con tu nombre (el nombre debe coincidir con el que le hemos dado al autocorrector la primera vez que lo ejecutamos).
  - El componente Header debe renderizar un elemento div con el id “cabecera”

- Dentro de dicho div debe tener una imagen con la clase "logo" y un h3 con la clase "mensaje"
- El h3 debe contener el texto que desee pero al menos debe poner tu nombre. Por ejemplo como se ve en la captura inicial de esta práctica "Bienvenido a la página de Enrique Barra"
- **0.5 puntos:** La aplicación tiene un h2 con id 'titulo' y el texto 'El tiempo'
- **1 punto:** La aplicación tiene los input para que el usuario introduzca la latitud y la longitud y el botón buscar. Los input latitud y longitud inicializan con los valores por defecto indicados en el fichero de configuración
  - El input para la latitud tiene id "latitud" y el valor por defecto "default\_lat" del fichero de config
  - El input para la longitud tiene el id "longitud" y el valor por defecto "default\_lon" del fichero de config
  - El button tiene el id "buscar"
- **1 punto:** Los input latitud y longitud cambian cuando el usuario escribe en ellos (y son componentes controlados de React, usan el estado de React para guardar su contenido)
- **1 punto:** El componente 'Resultados' recibe dos atributos (props) 'numitems' que indica cuantas tarjetas debe mostrar e 'items' con los datos que debe renderizar
  - Renderiza algo similar a lo que se ve en la figura:



- Tiene que indicar la timezone obtenida de la prop "items"
- Cada tarjeta contiene la fecha como indica la figura. En los datos, en el campo "daily" viene la predicción a varios días (de los que tendremos que mostrar los "numitems" primeros), y dentro del tiempo de cada día están sus datos para mostrar y crear la tarjeta.
- Todos los tiempos están mostrados en formato timestamp de unix y para procesarlos se recomienda hacer lo siguiente:
  - `new Date(TIMESTAMP * 1000).toLocaleDateString()`
  - puedes probar en una consola js que `new Date(1656590400 * 1000).toLocaleDateString()` da como resultado '30/6/2022'
  - consultar: <https://stackoverflow.com/questions/847185/convert-a-unix-timestamp-to-time-in-javascript/847196#847196>

- El icono o la imagen a mostrar debe tener la clase “tiempoimg” tiene una URL que también viene en los datos. Su nombre está en el campo “weather[0].icon” y puedes encontrar las imágenes en la carpeta public del repositorio de github o formarlas con la URL:  
“http://openweathermap.org/img/wn/\${dia.weather[0].icon}.png”
- **1 punto:** La aplicación al hacer click en 'buscar' carga los datos de constants/mock.js si en la configuración se indica que no se use el servidor
  - parámetro “use\_server” a false en el fichero de config
  - cuando se hace click en “buscar”
- **1 punto:** La aplicación llama al servidor si se indica así en la configuración y funciona bien con un 200 OK
  - para probar que la respuesta es correcta se puede hacer la comprobación `if(response.status === 200)`
  - La aplicación debe renderizar en este caso de 200 OK los datos que obtiene del servidor igual que hacía con los datos fake o mock.
- **1 punto:** La query formada para llamar al servidor es correcta
  - contiene el parámetro `server_url` que se indica en config
  - contiene `appid=API_KEY` siendo `API_KEY` el parámetro `api_key` indicado en config
  - contiene `lat=XXX` y `lon=YYY`. Donde XXX e YYY son los parámetros que ha introducido el usuario
- **1 punto:** La aplicación llama al servidor si se indica así en la configuración y funciona para códigos de error (`response.status` distinto de 200)
  - en este caso `response.json()` será un json con un campo “cod” con el código del error y un campo “message” que debe mostrar en un div de id “error” (puede hacer esta funcionalidad en un componente nuevo o directamente que pinte dicho div App o el componente Resultado o como prefiera). Un ejemplo funcionando sería la siguiente figura, donde si introducimos la latitud mal da un error:



**Bienvenido a la página de Enrique Barra**

**El tiempo**

Latitud:

Longitud:

**Error**

**Se ha producido un error.**

Descripción: Obtenido error al llamar al API. Código 400

Mensaje del servidor: wrong latitude


## Pruebas manuales y capturas de pantalla

Como se ha comentado se provee una batería de tests que la aplicación debe pasar y que serán los que produzcan la nota final. Pero es importante desarrollar la aplicación viendo poco a poco el resultado y visualizando cada cambio introducido hasta que funcione. Considerando los test provistos por el autocorrector como una serie de requisitos “estrictos” o “estáticos” como por ejemplo que tenga determinado id o clase alguna etiqueta o que el componente se llame de determinada manera.

No se recomienda por lo tanto ir desarrollando sin visualizar lo que hacemos y pasar el autocorrector a cada paso “a ver si se consiguen los puntos”, porque los errores que dan las baterías de tests son más crípticos que lo que veremos en el navegador o en la consola por nosotros mismos al visitar <http://localhost:3000>

Adicionalmente a pasar la batería de tests y obtener un 10/10 hay que hacer dos capturas de pantalla con la aplicación completa, es decir en la versión final antes de entregar. Dichas capturas se tienen que colocar en formato png, jpg o pdf en el directorio “miscapturas”. Y el autocorrector las subirá junto con el código de la práctica y el resto de evidencias a Moodle. Estas capturas son obligatorias y deben ser personales, en ellas se debe ver que la cabecera de la práctica pone el nombre del alumno y que el estilo es el entregado en el código. Estas capturas serán revisadas por el profesor para comprobar que se ha realizado la funcionalidad correctamente.

En esta práctica hay que subir dos capturas similares a las siguientes, es decir una con un funcionamiento correcto de la petición al API y una con un error.



Bienvenido a la página de Enrique Barra





**El tiempo**

Latitud:

Longitud:

Timezone: Europe/Madrid

El tiempo en los próximos días será:

<p>30/8/2022</p>  <p>Temp: 29.27°C Humedad: 32% Viento: 6.58m/s</p>	<p>31/8/2022</p>  <p>Temp: 29.09°C Humedad: 18% Viento: 6.04m/s</p>	<p>1/9/2022</p>  <p>Temp: 28.93°C Humedad: 15% Viento: 5.72m/s</p>	<p>2/9/2022</p>  <p>Temp: 26.83°C Humedad: 23% Viento: 6.46m/s</p>
--	--	---	---



Bienvenido a la página de Enrique Barra

**El tiempo**

Latitud:

Longitud:

**Error**

**Se ha producido un error.**

Descripción: Obtenido error al llamar al API. Código 400

Mensaje del servidor: wrong latitude

## Pruebas con el autocorrector

El autocorrector es la herramienta que permite pasar la batería de tests a la práctica y producir una nota. También subirla a Moodle junto con el código desarrollado, las capturas y otras evidencias de evaluación.

Ejecute el autocorrector tantas veces como desee en la práctica y suba la nota a Moodle también tantas veces como desee hasta que se cierre la entrega, **nos quedaremos con la última nota que hayamos subido.**

## Dudas y tutorías

Para dudas sobre el enunciado y sobre la práctica en general vamos a utilizar el foro de la asignatura. Lo único que está prohibido es subir vuestra solución completa al foro en un zip o enlace o similar. Si se pueden compartir trozos de código para pedir ayuda o para ilustrar lo que os está ocurriendo.

Como insistimos en la guía de la asignatura y comentamos el día de la presentación la copia de la práctica es un suspenso automático de toda la asignatura (y posibilidad de tomar medidas disciplinarias a nivel UPM dependiendo de la gravedad del caso) tanto para el que copia como para el que se deja copiar. Así que por favor se prudente y no compartas tu código como un zip con tus compañeros, una vez que sale de tu ordenador no sabes quién puede entregar tu código en su nombre.

Contacto (aunque insisto que para las dudas es mejor vía foro):

Enrique Barra Arias - [enrique.barra@upm.es](mailto:enrique.barra@upm.es)