Práctica 4 - React.js - IWEB

Noviembre 2021

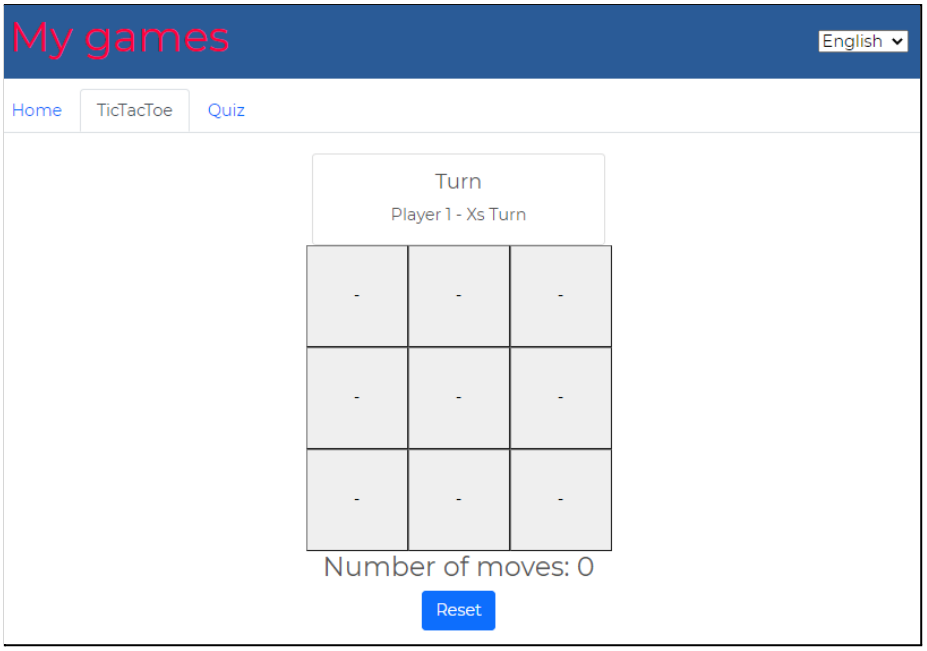
Introducción

Requisitos:

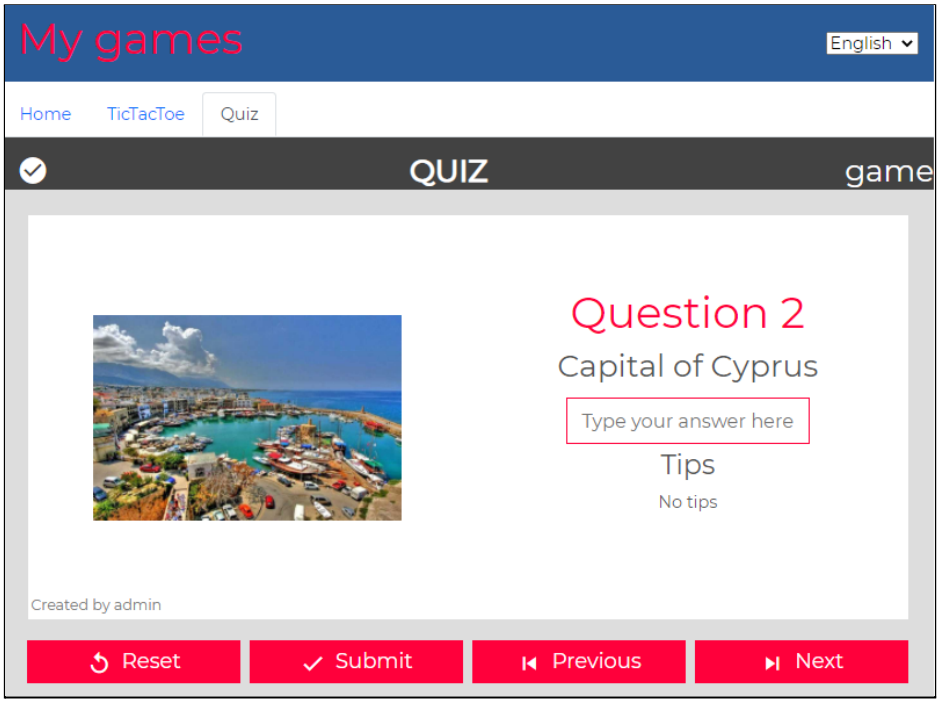
- Nociones básicas de las tecnologías explicadas en clase (HTML, CSS, JS, React)
- Editor de código
- Tener instalado node.js, git y yarn

El objetivo de esta práctica es realizar, utilizando las tecnologías vistas en clase, una Single Page Application (SPA) de minijuegos. Nuestra aplicación web consta de 3 pestañas al menos, la primera presenta el site, la segunda muestra el tres en raya realizado en clase (https://github.com/ging/iweb_tictactoe_2021) (el juego podéis ver que tiene varias ramas en el repositorio de github con diferentes alternativas, usar la rama step7 que tiene los componentes como clases o step7-hooks que tiene los componentes como funciones) y la tercera un juego de preguntas que permita jugar a un juego equivalente al Quiz creado en la asignatura de CORE. Estas pestañas se pueden ver en las siguientes tres capturas, siendo estas solo una propuesta posible de diseño:

Página de bienvenida



Segunda pestaña, tres en raya



Tercera pestaña, quiz

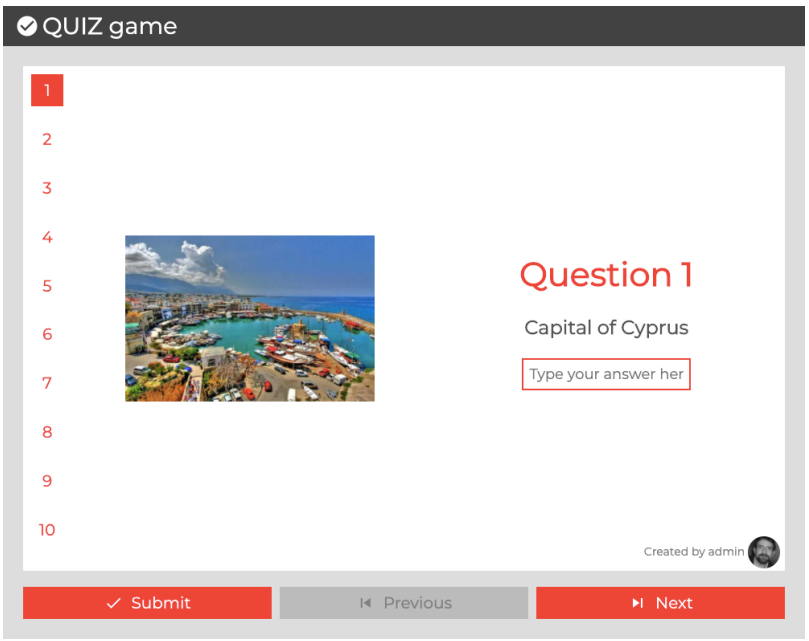
La página además debe estar internacionalizada, es decir tiene al menos dos idiomas que el usuario puede seleccionar con un selector situado en la cabecera de la página. Al seleccionar el idioma Español todos los textos cambian, tanto en la cabecera, como en los juegos. Nota: El tres en raya se provee sin internacionalizar, esta labor queda para el alumno. Por ejemplo la página principal se vería así (vemos que cambia incluso la imagen que se muestra en Español):

Página principal en Español

Para la navegación entre las pestañas implementadas, la SPA utilizará react router para mostrar diferentes rutas en el navegador según el usuario vaya navegando por la web. http://localhost:3000/ para la página principal, http://localhost:3000/tictactoe para el tres en raya, http://localhost:3000/quiz para el juego del quiz.

El juego "quiz" deberá desarrollarse desde cero. Este juego consistirá en contestar 10 preguntas aleatorias procedentes de la batería de preguntas del juego de Quiz. Para ello, es necesario descargarse dicha batería (las preguntas y sus respuestas) de un servidor web similar al realizado en CORE. Se mostrarán sucesivamente las distintas preguntas al usuario (cada vez que se cargue el juego las 10 preguntas pueden ser diferentes), que podrá ir contestando a cada una de ellas. Al terminar, el usuario pulsará el botón de "Submit" para evaluar sus respuestas y obtener su porcentaje de aciertos. La evaluación de las respuestas se realizará en la propia aplicación web, no en el servidor.

En la siguiente imagen se muestra otro posible ejemplo de esta aplicación y con navegación entre preguntas:



Primero se recomienda programar la aplicación con datos estáticos (mock data) y después modificar el código para bajarse los datos actualizados del servidor. Hay un ejemplo de datos de mock en Moodle (`mock-data.js`). Estos datos son un extracto de los datos provenientes de:

https://core.dit.upm.es/api/quizzes/random10wa?token=TU_TOKEN

Nota: Es necesario sustituir TU_TOKEN por el correspondiente a cada pareja de laboratorio. Puede consultarse el token en https://core.dit.upm.es, iniciando sesión y accediendo al perfil de usuario en la esquina superior derecha de la página.

Primeros pasos: Preparación del entorno de desarrollo

1. Instala node.js y git sino no lo has hecho ya
2. Instala yarn
3. Instala Create React App: `yarn global add create-react-app`
4. Comienza un nuevo proyecto: `yarn create-react-app my-app`
5. Entra en el directorio creado: `cd my-app`
6. Inicia el servidor de desarrollo: `yarn 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.

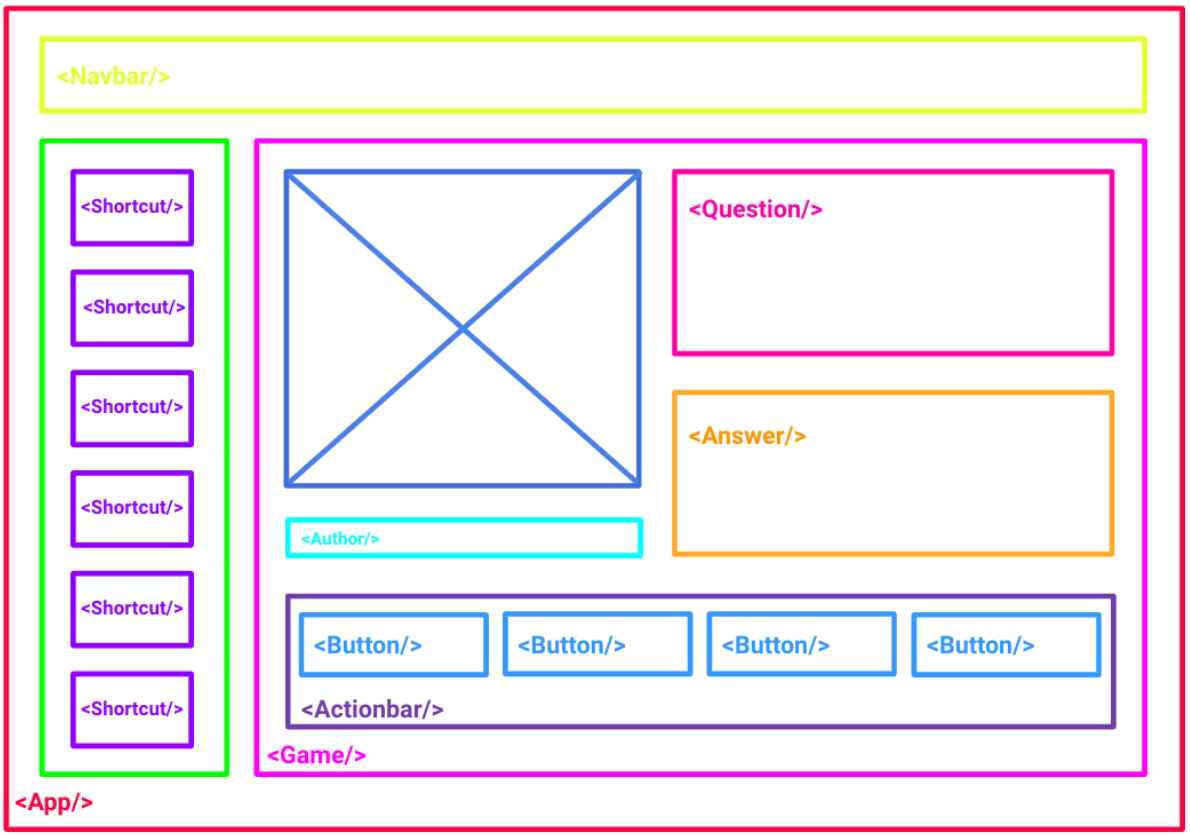
Al utilizar Create React App se crea una estructura de directorios en la carpeta `my-app`, que contiene todos los ficheros necesarios para comenzar el desarrollo. Para entender mejor cómo funciona el servidor de desarrollo, observa el contenido de `public/index.html`. Se trata

de la página web que renderizará nuestra aplicación de React. Como ves, apenas tiene contenido; sólo tiene un elemento `div` con id `root`.

En la carpeta `src` se encuentran todos los componentes de React del proyecto. El fichero `src/index.js` es el que sirve de puente entre React y el HTML de la página. En él, se renderiza el componente `App` (el componente principal de nuestra aplicación) en el elemento con `id="root"` de `index.html`, y dentro de él, todos los componentes que se definan.

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 para el juego del quiz es el siguiente (se valorará la originalidad del alumno al proponer el suyo propio):



No hace falta que el diseño sea el definitivo, se puede ir cambiando durante el desarrollo. El tres en raya se proporciona ya desarrollado (solo habrá que hacer ajustes a nuestro gusto e incluir la internacionalización) y la SPA completa tendréis que decidir vosotros el diseño y la separación en componentes que le dáis.

Estado de la aplicación

El primer paso es siempre definir el estado de nuestra aplicación. Para nuestra aplicación "App" que engloba todo (nuestra SPA que carga las 3 pestañas y dependiendo de la ruta

carga una u otra) sólo será necesario tener un estado que sea el idioma que se está mostrando (lang). Cada uno de los juegos mantendrá además su propio estado.

Rutas

Esta aplicación muestra por defecto la página principal que es una página estática (es decir cuando se solicita http://localhost:3000) que podemos definir en un componente <Home>

Si el usuario solicita (haciendo click en la pestaña "tictactoe") http://localhost:3000/tictactoe entonces la aplicación muestra en lugar el componente <Tictactoe> que tendrá el estado adecuado del tres en raya.

Si el usuario solicita (haciendo click en la pestaña "quiz") http://localhost:3000/quiz entonces la aplicación muestra el componente <Quiz> que tendrá el estado adecuado del cuestionario (explicado en detalle más adelante).

```
<nav className="nav nav-tabs" role="tablist">
  <NavLink to="/">Home</NavLink>
  <NavLink to="/tictactoe">TicTacToe</NavLink>
  <NavLink to="/quiz">Quiz</NavLink>
</nav>
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="tictactoe" element={<Tictactoe />} />
  <Route path="quiz" element={<Quiz />} />
</Routes>
```

Para esta funcionalidad utilizamos react router. El componente NavLink puede especificar una clase especial si el link está activo, ver documentación oficial en https://v5.reactrouter.com/web/api/NavLink) .

Internacionalización

La aplicación debe estar internacionalizada, es decir debe proveer un selector de idioma con las opciones Español/Inglés. La selección del usuario la guardamos en el componente App.js (haremos como hemos visto un método handleLanguageChange que permita editar dicho parámetro que está en el estado de App.js.

Con dicho parámetro y el diccionario en json con las traducciones nos creamos un React Context en un fichero llamado LangContext así:

```
import React from "react";
import es from './lang/es.json';

export const LangContext = React.createContext({userLang: 'es', dictionary: es});
```

Ese LangContext lo tendremos que importar en todos los componentes que queramos internacionalizar y además en App.js tendremos que poner un componente padre de todos los que hagamos y que haga `<LangContext.Provider value={`...

Juego Quiz

El juego que se muestra en la pestaña “quiz” hay que desarrollarlo desde cero. Crearemos para ello un componente Quiz.js que será el padre de dicho juego, y que como hemos visto arriba al explicar las rutas será el que se renderice en caso de que se pida la ruta http://localhost:3000/quiz. Esta sección explica el desarrollo de dicho juego.

Dadas las características del juego, la parte principal del estado será la lista de preguntas del Quiz (`quizzes`). Además, puesto que las preguntas se muestran de una en una, debemos llevar la cuenta del quiz que se está visualizando en cada momento (`currentQuiz`). También tenemos que saber cuando el usuario ha pulsado el botón de “Submit" y terminado el juego (`finished`) y recoger la puntuación final (`score`). Un posible estado con todos estos requisitos sería de la siguiente manera:

```
initialState = {
     score: 0,
     finished: false,
     currentQuiz: 0,
     quizzes: [

     ]
}
```

Se deja a elección del alumno desarrollar el componente Quiz.js como clase (con un constructor que haga this.initialstate= {,,,) o como función (usando entonces el hook useState).

El siguiente paso es importar los datos de las preguntas y respuestas del quiz a la aplicación. Como ya se ha mencionado, se comenzará con datos estáticos de ejemplo disponibles en Moodle, que almacenaremos en `src/assets/mock-data.js`. Para tener acceso a este fichero desde nuestra app, primero debemos importarlo. En la parte superior del fichero de `App.js` añadimos:

```
import { quizzes } from "../assets/mock-data";
```

Ahora ya tenemos acceso a `quizzes` desde nuestro componente. En el estado inicial debemos asignar el valor de `quizzes` a la parte del estado correspondiente:

```
quizzes: [ ...quizzes ]
```

O si usamos hooks habrá que asignarlo al estado inicial de quizzes:

```
const [quizzes, setQuizzes] = useState([]);
```

Primer componente

Una vez que ya están disponibles los datos, vamos a crear nuestro primer componente personalizado. En el directorio `src` creamos un componente nuevo llamado `Game.js`. Escribimos el código básico para un componente (importación y declaración):

```
import React from 'react';
export default class Game extends React.Component {
  render() {
    return (
      <div>Mi primer componente</div>
    );
  }
}
```

¿Cómo hacemos uso de este componente? Cómo se ve en el diseño preliminar de nuestra aplicación, el "padre" de este componente es <App/>, por lo que vamos a `App.js` e importamos el componente nuevo:

```
import Game from "./Game";
```

También tenemos que modificar la función del componente `App` para renderizar `Game` donde estaban los componentes que venían por defecto:

```
function App(props) {
 return (
    <div className="App">
        <Game/>
    </div>
  );
}
```

El componente `Game` debe renderizar la pregunta que está seleccionada. En nuestro estado, por defecto es la primera pregunta. Para poder tener acceso a ella desde `Game`, es necesario pasársela como `prop`.

```
function App(props) {
  return (
    <div className="App">
        <Game quiz={props.quizzes[props.currentQuiz]} />
    </div>
  );
}
```

En el método render de `Game` podemos mostrar, como prueba, el enunciado de la pregunta actual. Si observamos el fichero de datos de mock, comprobamos que en cada pregunta el

enunciado viene definido en la clave `quizzes` de cada objeto del array de preguntas. Para hacer uso de él desde `Game`, debemos hacerlo de la siguiente manera:

```
export default function Game(props) {

    return (
      <div>{props.quiz.question}</div>
    );

}
```

En el navegador deberíamos ver el enunciado de la primera pregunta “Capital of Cyprus”.

Siguientes pasos

- Mostrar para cada quiz su imagen correspondiente (`attachment`) si la tiene
- Mostrar para cada quiz el nombre del autor y su avatar (si lo tiene)
- Crear botones de “Anterior”, “Siguiente” y “Submit”.
- Lograr que los botones de “Anterior” y “Siguiente” cambien el quiz que se muestra al hacer click en ellos. Para ello es necesario modificar la parte del estado asociada a `currentQuiz`. Es importante prestar especial atención a los extremos: es necesario deshabilitar (usar el atributo `disabled`) el botón de “Siguiente” al llegar a la última pregunta y el de “Anterior” cuando se está en la primera. También hay que contemplar el caso de que el array de preguntas esté vacío.
- Crear un input para que el usuario pueda responder a las preguntas.
- Se puede crear índice de preguntas para acceder más rápidamente a cada pregunta sin necesidad de usar los botones de “Anterior” y “Siguiente”. (Pista: usar la función *map* para iterar sobre la lista de quizzes y mostrar un botón que lleve a cada quiz).
- Lograr que el botón de “Submit” evalúe si las respuestas son correctas. Ésta debe actualizar la parte del estado asociada a `score` (con la puntuación obtenida) y a `finished` (`true`).
- Mostrar la puntuación obtenida una vez el juego haya finalizado (en vez de las preguntas).
- Conseguir descargar los quizzes desde el servidor cada vez que se carga la página en vez de cargarlos desde el fichero de mock. Para ello se debe realizar una petición asíncrona a la URL que se menciona en la introducción y asignar el resultado de la petición al estado. La petición asíncrona se puede hacer en el componentDidMount del componente `App` o utilizando el hook useEffect. Para hacer esta petición se puede emplear fetch, AJAX (incluido en jQuery), axios, etc.

Para terminar

Una vez terminado el desarrollo, es necesario empaquetar el código en ficheros estáticos. En el terminal, nos situamos en el directorio del proyecto y hacemos `yarn build`. o `npm run build` A este proceso se le llama "**preparar para producción**" (*build for production*).
En el directorio `build` se crearán todos los archivos necesarios para ejecutar la webapp. Puedes comprobarlo abriendo el `index.html` de la carpeta `build` y viendo que todo funciona sin necesidad del servidor de desarrollo. Incluso si quisiéramos podríamos subirlo a un servidor de ficheros estáticos, como *neocities* y nuestra webapp seguiría funcionando. Para probar nuestra app de producción podemos ejecutar `serve -s build` en la consola y acceder con el navegador a http://localhost:5000/

Ficheros a entregar

El alumno debe entregar todo el código realizado en un archivo comprimido, incluida la aplicación lista para producción. **NO** incluir el directorio `node_modules`. Todas las dependencias necesarias deben estar reflejadas en el `package.json`.

Se debe incluir un fichero llamado `00datos.txt` en la raíz del fichero comprimido en el que figure el nombre de los integrantes de la pareja y **el listado de las mejoras realizadas** sobre la práctica básica. Si en este fichero no hay un listado de mejoras realizadas no las buscaremos en la aplicación y por lo tanto no serán evaluadas.

> ***IMPORTANTE: **No se corregirán las prácticas que no incluyan la aplicación lista para producción en el directorio `build`, tal y como se explica en el apartado anterior.**

Evaluación

La práctica se valorará sobre un máximo de 10 puntos, los cuales se podrán obtener realizando las siguientes tareas:

- **7 puntos (obligatorio)**: Correcto funcionamiento de la práctica con los requisitos que hemos visto en este enunciado:
  - La página es una SPA con 3 pestañas, que atienden usando las rutas a las siguientes URLs http://localhost:3000/ http://localhost:3000/tictactoe y http://localhost:3000/quiz
  - La primera pestaña es una página estática que da la bienvenida, la segunda el tres en raya realizado en clase y la tercera el juego del quiz
  - La página está internacionalizada en español/inglés
  - El quiz debe:
    - Bajarse los quizzes del servidor.
    - Contemplar el caso de que el array de quizzes esté vacío.
    - Mostrar los quizzes en la interfaz web de una en una.
    - Mostrar la imagen asociada al quiz, si la tiene.

    - Contemplar el caso de que el quiz no tenga imagen (no mostrar nada o mostrar una imagen por defecto)
    - Mostrar el nombre del usuario que ha creado el quiz y su foto (si la tiene)
    - Cambiar de quiz al hacer click en los botones de siguiente/atrás.
    - Permitir al usuario responder a los quizzes.
    - Permitir al usuario evaluar sus respuestas y calcular su puntuación.
    - Desarrollar la lógica de cambiar de quiz y evaluar las respuestas
    - Empaquetar la aplicación resultante para producción utilizando las instrucciones provistas.
- **Mejora Índice y Reset - 1 punto**:
    - Mostrar el índice de quizzes y cambiar de quiz al hacer click en los botones del índice
    - Botón de reset que pida un conjunto nuevo de quizzes al servidor y reinicie el juego.
- **Mejora CSS - 1 punto:**
    - Personalización de los estilos (CSS, animaciones, diseño original, spinner, etc.). Sólo se valorará si se estima que el alumno ha invertido una cantidad de esfuerzo significativa.
- **Mejora cuenta atrás - 1 punto:**
    - Cuenta atrás que limite el tiempo que tiene el usuario para responder las preguntas y termine el juego si se acaba el tiempo.
- **Mejora tecla intro - 1 punto:** Conseguir que al pulsar la tecla enter dentro del input de texto se pase al siguiente quiz o, si es la última, que termine el juego comprobando el nº de aciertos (submit).
- **Mejora tests - 1 punto:** Crear una batería de tests con Jest (ya incluído en create-react-app).

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.

Contacto (aunque insisto que mejor vía foro):
Enrique Barra Arias - enrique.barra@upm.es