MEMORIA EXPLICATIVA

Buenas Gaizka, no sé muy bien cómo se hace una memoria de este tipo, pero iré explicando cómo fue todo el proceso en plan síntesis, intentando facilitar la valoración del perfil técnico.

ANTECEDENTES

- Antes que nada comentar que perdí 1 punto de cordura cuando llego el martes a Gijón del pueblo de segar, me llega la cobertura y veo que hay que hacer un sprint 3
- También resaltar que me he notado cierta "visión de túnel" tras hacer la red social del bootcamp, a la hora de resolver el blog sencillo. Un ejemplo: al pedir implementación de react, directamente recurro a create React App, con lo que no entendía muy bien lo de la estructura html del home pues automáticamente ya veía componentes tras un return en la page de home.
- Como esto era un Sprint 3 individual, se me hizo raro no tener las conversaciones diarias con el grupo de trabajo. Creo que cosas que se me atascaron o que me llevaron más tiempo de lo buscado, las podría haber hecho más eficientemente con los compañeros.

ORDEN DE TRABAJO

- Cuando me puse el míercoles hice lo que nunca pude hacer durante el bootcamp: no empezar a codear nada... y simplemente hacer un checklist en onedrive de lo que necesitaría para el backend y el frontend.
- Primero empecé, por el "respeto" a React, por la "zona de confort": el backend.
- Revisando el proyecto del bootcamp fui anotando un listado de tareas con una pequeña descripción de lo que iba a necesitar, y usando un código de colores (verde:hecho, amarillo:en ello,rojo:me quedó pendiente)
- Como el checklist está más abajo en la memoria, por agilizar me voy a ***centrar en lo que quedó pendiente***
    - **<u>Imágenes</u>**
        - No me cargaron las imágenes de la bd que tenía en la carpeta public en el backend. No sé si aquí influyó tanto la visión de túnel de usar la public en el frontend con la red social, y que hasta última hora no llegamos en el proyecto a andar cargando imágenes tras hacer el fetch.
        - Para los inputs sé que si quiero adjuntar imágenes son tipo "file", pero me daba problemas. Y aquí noté también cómo nunca llegamos a adjuntar archivos en el proyecto.
        - De la suma de lo anterior también deriva el no borrar las imágenes del servidor al borrar el post
    - **<u>Fechas</u>**
        - Tras hacer pruebas no conseguí que me la guardaran al crear la entrada, y en consecuencia al editar la entrada. En el proyecto bootcamp no llegué a tratar con el tema registro de fechas "actuales" para los posts, toqué los formularios de los usuarios que era más genérico el "birthday"

AUTOEVALUACIÓN

- Me ha gustado poder dejar “documentado” el checklist de lo que necesitaba, pero me queda una sensación parecida a cuando hablamos ese poco: me gustaría con calma poder mirar aspectos como node, y lo anterior también, para tener una base más sólida. Sé que suena un poco a síndrome del impostor, pero en mi me ayuda poder revisar proyectos guiados para tener más claros los “checklists”.

CHECKLIST

- Añadir a la carpeta raíz el archivo .gitignore con el texto node_modules/
    - Ignorará la carpeta node_modules/ (muy, muy pesada) a la hora de subir contenido al repositorio

- ***BACKEND***
    - Instalar node (entorno de ejecución del lado del servidor para permitir trabajar en el backend)
    - Instalar npm (manejador de paquetes de node)
        - npm install
    - Inicializar el proyecto en node
        - npm init
    - Instalación de librerías
        - Instalar express (framework de node)
            - Npm install express
            - Facilitar la creación de aplicaciones web y APIs:
                - Gestión solicitudes y respuestas HTTP
                - Enrutamiento para manejar métodos HTTP (GET, POST, PUT, DELETE...
                - Manejo funciones middleware para autenticación (verificar la identidad de un usuario o entidad para garantizar que sea quien dice ser), validación de datos antes de llegar al controlador final (***(req, res) => { ... }***) o ruta de la solicitud
        - Instalar cors (librería de node)
            - para manejar y permitir las solicitudes de recursos entre diferentes dominios u orígenes (dominio, protocolo y puerto) en un navegador.
        - Instalar moment(librería de node)
            - Para manejo de fechas y tiempo
        - Instalar Sequelize
            - Librería para interactuar, usando JS, con bases de datos relacionales como MySQL, PostgreSQL, SQLite y otros
        - Instalar mysql2
            - interfaz para conectarse a una base de datos MySQL desde una aplicación Node.js y ejecutar consultas SQL directamente
            - Necesario para usar Sequelize
    - Crear base de datos MySQL

- Crear archivo de conexión a la base de datos MySQL
  - Se requiere Sequelize y se exporta como módulo *de conexión a nuestra base de datos a index.js*

- Crear index.js
  - Se requiere Express
    - Donde se arma e inicializa el servidor
    - **Middleware** para analizar el cuerpo de la solicitud como JSON
      - app.use(express.json());
    - Middleware para analizar el cuerpo de la solicitud como datos de formulario codificados en URL
      - app.use(express.urlencoded({ extended: true }));
    - **Uso de Cors** para que el servidor responda a solicitudes de diferentes orígenes
      - app.use(cors());
    - Se configura la ruta estática para servir los archivos (imágenes, por ejemplo) desde la **carpeta "public"** en el backend
      - app.use(express.static('public'));
    - Se crean los endpoints (la buena práctica era importarlos de una carpeta específica para rutas y sus endpoints)
      - obtener todos los posts:
        - app.get('/allposts',…), consulta SELECT
      - crear un nuevo post:
        - app.post('/newpost',…), consulta INSERT
      - actualizar un post:
        - app.put('/posts/:id_entrada'…), consulta UPDATE
      - borrar un post (no borra imagen del servidor):
        - app.delete('/posts/:id_entrada',consulta DELETE
      - obtener un post específico
        - app.get('/post/:id_entrada', ...)
      -
  - Se requiere Cors
  - Se requiere Moment si queremos
  - Se importa el módulo de conexión a nuestra base de datos

- ***FRONTEND***
  - Instalar React
  - Instalar Librerías:

- Instalar react-router-dom
  - biblioteca que ofrece una variedad de componentes, como BrowserRouter, Route, Switch, Link, entre otros, que facilitan el manejo del enrutamiento en aplicaciones de React.
- Instalar bootstrap
- Instalar @fortawesome/react-fontawesome
  - componente de React que proporciona la integración con los íconos de Font Awesome.
- Instalar @fortawesome/free-solid-svg-icons
  - íconos de Font Awesome

- Rutas a tener en cuenta:
  - /
    - Para el feed de posts más header, nav y footer
  - /newpost
    - Para crear entradas
  - /editpost/:id_entrada
    - Para editar entrada
  - /post/:id_entrada
    - Para ver entrada completa
- Crear carpeta Router con RouterPrincipal.js
  - Donde se definen, tras importar aquí los componentes y páginas jsx necesarias, las diferentes rutas y se mapean a componentes específicos en la app al utilizar BrowserRouter.
  - Cuando el usuario navega a una determinada URL, BrowserRouter se encarga de manejar la correspondencia entre la URL y los componentes que deben mostrarse.
  - Estructura simple del "enrutamiento":
    - <Route path="/" element={<WebHome/>} />
    - <Route path="/newpost" element={<WebNuevaEntrada/>} />
    - <Route path="/post/:id_entrada" element={<WebEntrada/>} />
    - <Route path="/editpost/:id_entrada" element={<WebEditarEntrada/>} />

- Creación de páginas jsx
  - Son componentes que representan páginas o vistas individuales en una aplicación , y a su vez contienen los componentes creados
  - Listado con sus correspondientes componentes:
    - **WebHome**
      - Header
      - Navbar
      - EntradaHome
      - Footer
    - **WebNuevaEntrada**
      - Header
      - Navbar
      - EntradaNueva

        - Footer
    - **WebEditarEntrada**
        - Header
        - Navbar
        - EntradaEditable
        - Footer
    - **WebEntrada**
        - Header
        - Navbar
        - EntradaCompleta
        - Footer
- Creación de componentes
    - **Header**
        - Para el nombre del blog con un h1
    - **Navbar**
        - Con un botón para añadir entradas
    - **EntradaHome**
        - Vista preliminar de las entradas ya añadidas con lo siguiente:
            - Imagen de la entrada (con click a la entrada completa).
            - Encabezado en h2 con el título de la entrada ((con click a la entrada completa).
            - Fecha de publicación de la entrada
            - Extracto de 50 caracteres de la entrada
            - Botón tipo icono para borrar
    - **Footer**
        - Con el nombre del blog, copyright y año

    - **EntradaNueva**
        - Se accede al formulario de la entrada tras pinchar en el botón de añadir entradas en el NavBar
        - Su estructura es la siguiente:
            - Adjuntar imagen
            - Título de la entrada
            - Contenido de la entrada
            - Fecha será automática cuando se inserte la entrada
            - Botón de guardar
    - **EntradaEditable (mismo problema derivado de Entrada Nueva)**
        - Se accede a ella al darle al botón de editar en EntradaCompleta
        - Sigue la misma estructura que EntradaCompleta, teniendo que tener disponible el botón tipo icono de Guardar
        - Al pulsar en guardar se guardan en la base de datos los cambios realizados en la entrada
    - **EntradaCompleta**
        - Se accede a ella pinchando en la imagen o encabezado de la vista preliminar de la entrada en el home
        - Su estructura es la siguiente:

- Imagen de la entrada
- Encabezado en h1 con el título de la entrada
- Fecha de publicación de la entrada
- Contenido completo de la entrada
- 3 botones tipo icono:
  - Editar
  - Guardar (solo sale si está editando la entrada)
  - Borrar

- Editar App.js
  - Es el archivo principal de la aplicación, que normalmente se encuentra en la raíz del proyecto.
  - Es el componente principal que actúa como el punto de entrada para la interfaz de usuario de la aplicación.
  - Desde aquí se retorna RouterPrincipal.js