Instituto Politécnico Formosa

Segundo Parcial: Galeria de Superhéroes

Nota importante sobre comentarios: Para facilitar la identificación y lectura de los comentarios en el código, es necesario que instalen la extensión "Better Comments" en Visual Studio Code.

Pasos para instalar la extensión:

1. Abrir Visual Studio Code.
2. Ir a la pestaña de extensiones: a. Hagan clic en el ícono de extensiones en la barra lateral izquierda, o presione Ctrl+Shift+X (Windows/Linux).
3. Buscar la extensión "Better Comments": a. Escriban "Better Comments" en la barra de búsqueda.
4. Instalar la extensión: a. Seleccionen la extensión desarrollada por Aaron Bond y hagan clic en "Install".
5. Reiniciar Visual Studio Code si es necesario.

Esta extensión les permitirá visualizar los comentarios con distintos colores y formatos, facilitando la comprensión de las tareas que deben realizar.

Instituto Politécnico Formosa

Instrucciones Generales

El presente examen consiste en completar la implementación de una aplicación frontend en React que consume una API REST existente. Se les proporcionará una carpeta base con:

- **Páginas completas** (Login, Register, Home) con diseño y estilos ya implementados
- **Componentes visuales** (Navbar, Footer, Loading) con diseño ya implementado
- **Custom Hook useForm**.
- **Rutas ya configuradas**.
- **Estructura de carpetas** base del proyecto
- **Backend funcionando** que debe ser configurado y ejecutado

Importante: Los archivos proporcionados NO deben ser modificados en su estructura visual. El trabajo se enfoca en:

- Configurar el proyecto backend y frontend
- Ejecutar el seed de la base de datos
- Integrar la lógica de autenticación y consumo de la API en los componentes proporcionados
- Verificar que el usuario esté autenticado consultando el endpoint /api/profile

Una vez finalizado, se debe subir el trabajo a GitHub y entregar el enlace en Google Classroom antes del horario límite establecido.

Criterios de Evaluación

1. Presentación del código

- Código limpio, ordenado y bien indentado.
- Uso obligatorio de try-catch en todas las peticiones asíncronas.
- Respeto de la estructura de carpetas proporcionada.
- Uso exclusivo de ESModules (import/export).
- Código funcional, modularizado y sin errores de ejecución.
- No modificación de los componentes visuales proporcionados (excepto para integrar la lógica necesaria).

2. Hooks y Estado

- **useState:** Manejo correcto del estado local en componentes.
- **useEffect:** Implementación adecuada para efectos secundarios (fetch de datos, verificación de autenticación).
- **Gestión de dependencias:** Arrays de dependencias correctos en useEffect.

3. Integración con API Backend

- Configuración correcta de **credentials: 'include'** en todas las peticiones.
- Implementación correcta de todos los endpoints:
    - POST /api/register
    - POST /api/login
    - GET /api/profile
    - POST /api/logout
    - GET /api/superheroes
- Verificación de autenticación usando /api/profile en las rutas protegidas.
- Manejo apropiado de respuestas y códigos HTTP.
- Implementación de estados de carga (loading).
- Mensajes claros de éxito o error.

4. Autonomía y originalidad del trabajo

Está estrictamente prohibido:

- Usar herramientas automáticas como IA generativa o autocompletado avanzado.
- Compartir archivos o código con otros compañeros durante la evaluación.
- Modificar los componentes visuales proporcionados más allá de la integración de lógica necesaria.

Cualquier incumplimiento será motivo de desaprobación directa.

Entrega mediante Git y GitHub – Uso de ramas y commits

Requisitos obligatorios para el control de versiones

1. El proyecto debe ser versionado con Git desde el inicio.
2. Se deben crear y utilizar dos ramas:
    - **main:** rama principal para la entrega final.
    - **develop:** rama de desarrollo donde se realiza el examen.
3. Todo el desarrollo debe hacerse en la rama develop.
4. Al finalizar, se debe hacer un merge limpio de develop hacia main, sin conflictos.
5. Se deben realizar **al menos 5 commits** con mensajes claros y descriptivos.

Requisitos del repositorio remoto

1. El nombre del repositorio en GitHub debe seguir la siguiente convención: **segundo-parcial-tlp2-apellido-nombre** Ejemplo: segundo-parcial-tlp2-perez-juan
2. El repositorio debe contener:
    - Todos los archivos del proyecto (package.json, src/ completo, vite.config.js, etc.)
    - El archivo .env.example con los nombres de las variables utilizadas
    - El archivo .env y la carpeta node_modules NO deben subirse
3. El enlace al repositorio debe entregarse en Google Classroom antes del horario límite establecido.

Consignas

1. Configuración inicial del proyecto

Nota: Se proporcionan dos carpetas: backend (con la API completa) y frontend (con el proyecto base de React).

1.1. Configuración del Backend

Tareas a realizar:

1. **Ingresar a la carpeta backend:**
2. **Instalar dependencias:**
3. **Configurar variables de entorno:**
    - Crear archivo .env en la raíz de la carpeta backend
    - Configurar las variables según el archivo .env.example proporcionado
4. **Ejecutar el seed de la base de datos:**

```
npm run db:seed
```

- Este comando insertará los datos de prueba necesarios en la base de datos
- Incluye usuarios de ejemplo y superhéroes

5. **Iniciar el servidor backend:**

```
npm start
```

El servidor debe estar corriendo en `http://localhost:3000`

1.2. Configuración del Frontend

Estructura proporcionada:

```
frontend/
└── src/
    ├── components/
    │   ├── Navbar.jsx       (proporcionado - solo integrar lógica)
    │   ├── Footer.jsx       (proporcionado - sin modificar)
    │   └── Loading.jsx      (proporcionado - sin modificar)
    ├── pages/
    │   ├── LoginPage.jsx    (proporcionado - solo integrar lógica)
    │   ├── RegisterPage.jsx (proporcionado - solo integrar lógica)
    │   └── HomePage.jsx     (proporcionado - solo integrar lógica)
    ├── hooks/
    │   └── useForm.js       (proporcionado - sin modificar)
    └── router/
        ├── AppRouter.jsx    (proporcionado - sin modificar)
        ├── PrivateRoute.jsx (proporcionado - sin modificar)
        └── PublicRoute.jsx  (proporcionado - sin modificar)
    ├── App.jsx              (proporcionado - sin modificar)
    ├── main.jsx             (proporcionado - sin modificar)
    └── index.css            (proporcionado - sin modificar)
```

Tareas a realizar:

1. **Ingresar a la carpeta frontend:**
2. **Instalar dependencias:**
3. **Verificar configuración de .gitignore:**
    a. Debe incluir: node_modules, dist, .env, .env.local
4. **Iniciar el servidor de desarrollo:**

---

2. Integración de Lógica en Componente Navbar

Ubicación: src/components/Navbar.jsx (proporcionado)

Nota: El diseño y estructura visual ya están completos. Solo debe integrar la lógica.

Integraciones a realizar:

1. **Implementar función Logout:**
    - Al hacer click en el botón de Logout, realizar petición POST a /api/logout
    - Usar credentials: 'include'
    - Después del logout exitoso, redireccionar a /login
    - Manejar errores apropiadamente
2. **Mostrar nombre del usuario:**
    - Si hay usuario autenticado, mostrar su nombre obtenido de /api/profile

---

3. Integración de Lógica en Páginas

Nota: Las páginas ya están proporcionadas con su diseño completo. Su tarea es integrar la lógica necesaria para que funcionen correctamente.

5.1. Página Login

Ubicación: src/pages/LoginPage.jsx (proporcionado)

Integraciones a realizar:

- Usar custom hook useForm para manejar el estado del formulario (campos: username, password)
- Implementar la función de submit que:
    - Valide que los campos no estén vacíos
    - Realice petición POST a /api/login con los datos del formulario
    - Use credentials: 'include' en la petición
    - Al login exitoso, redirija a /home usando useNavigate
    - Muestre mensajes de error si falla la autenticación
- Implementar estado de carga (loading) durante la petición
- Deshabilitar botón de submit mientras se procesa

3.2. Página Register

Ubicación: src/pages/RegisterPage.jsx (proporcionado)

Integraciones a realizar:

- Usar custom hook useForm para manejar el estado (campos: username, email, password, name, lastname)
- Implementar la función de submit que:
    - Valide que todos los campos obligatorios estén completos
    - Realice petición POST a /api/register con los datos del formulario
    - Use credentials: 'include' en la petición
    - Al registro exitoso, redirija a /home usando useNavigate
    - Muestre mensajes de error si falla el registro
- Implementar estado de carga durante la petición
- Deshabilitar botón de submit mientras se procesa

3.3. Página Home

Ubicación: src/pages/HomePage.jsx (proporcionado)

Integraciones a realizar:

1. **Obtener datos del usuario:**

- Al montar el componente, hacer petición GET a /api/profile para obtener datos del usuario
- Usar credentials: 'include'
- Mostrar mensaje de bienvenida personalizado: "¡Bienvenido/a, {name}!"

2. **Obtener lista de superhéroes:**
   - Al montar el componente, hacer petición GET a /api/superheroes
   - Usar credentials: 'include'
   - El endpoint devuelve un array de superhéroes en orden aleatorio
   - Guardar superhéroes en el estado
3. **Renderizar lista de superhéroes:**
   - Usar .map() para renderizar cada superhéroe en las cards del diseño proporcionado
   - Cada card debe mostrar:
     - Imagen del superhéroe
     - Nombre del superhéroe
   - Usar key única para cada card
4. **Implementar botón "Recargar":**
   - El botón ya está en el diseño proporcionado
   - Al hacer click, volver a hacer petición GET a /api/superheroes
   - Como el endpoint devuelve superhéroes en orden aleatorio, cada recarga mostrará un orden diferente
   - Mostrar estado de loading durante la recarga
   - Deshabilitar botón mientras se recarga
5. **Estados de la página:**
   - Mostrar componente Loading (proporcionado) mientras se cargan los datos iniciales
   - Mostrar mensaje apropiado si no hay superhéroes
   - Manejar errores en las peticiones

---

4. Configuración de Peticiones HTTP

Todas las peticiones a la API deben cumplir con:

1. **Usar credentials: 'include'** para enviar y recibir cookies de sesión
2. **La URL base de la API es:** http://localhost:3000
3. **Manejo de errores:**
   - Usar bloques try-catch
   - Verificar response.ok
   - Parsear errores del servidor
   - Mostrar mensajes claros al usuario
4. **Estados de carga:**
   - Implementar estado loading en cada petición
   - Mostrar componente Loading o deshabilitar botones

Endpoints de la API

Autenticación

1. **POST /api/register**
   - Body: { username, email, password, name, lastname }
   - Respuesta exitosa: { message, user }
2. **POST /api/login**
   - Body: { username, password }
   - Respuesta exitosa: { message, user }
3. **GET /api/profile <u>CRÍTICO PARA VERIFICACIÓN DE AUTENTICACIÓN</u>**
   - Requiere autenticación (cookie)
   - Respuesta exitosa: { id, username, email, name, lastname }
   - Respuesta sin autenticación: 401 Unauthorized
   - **Este endpoint se usa para verificar si el usuario está autenticado**
4. **POST /api/logout**
   - Requiere autenticación (cookie)
   - Respuesta exitosa: { message }

Superhéroes

5. **GET /api/superheroes**
   - Requiere autenticación (cookie)
   - Respuesta: Array de superhéroes en orden aleatorio
   - Cada superhéroe incluye: id, nombre, imagen, y otros atributos

---

Especificaciones Técnicas

Verificación de Autenticación

Importante: La verificación de si el usuario está autenticado se debe realizar consultando el endpoint /api/profile:

- En PrivateRoute: Si /api/profile falla → redireccionar a /login
- En PublicRoute: Si /api/profile tiene éxito → redireccionar a /home
- En Navbar: Si /api/profile tiene éxito → mostrar opciones de usuario autenticado

Manejo de Estado

- Usar useState para estados locales (formularios, loading, errores, datos)
- Usar useEffect para efectos secundarios (peticiones al montar, verificación de auth)
- NO usar bibliotecas de estado global (Context API, Redux, Zustand)

Validaciones

Validaciones mínimas requeridas:

- Login: username y password no vacíos
- Register: todos los campos obligatorios completos
- Mostrar mensajes de error claros al usuario

Estructura del README.md

El archivo README.md debe incluir:

1. **Título del proyecto:** "Segundo Parcial - Galeria de Superhéroes"
2. **Descripción:** Breve explicación de la aplicación

Aclaraciones Finales

- El backend ya está desarrollado y funcionando. Solo debe configurar e iniciar el servidor.
- **Debe ejecutar el comando npm run db:seed para insertar los datos de prueba en la base de datos.**
- Todas las páginas y componentes visuales ya están diseñados y estilizados completamente.
- Ya está implementado el customHook useForm.
- Ya está implementado la configuración de rutas.
- **NO modifique el diseño visual de los componentes proporcionados.**

- Enfóquese en implementar la lógica de integración con el backend.
- **La verificación de autenticación debe hacerse consultando /api/profile.**
- La aplicación debe ser completamente funcional sin errores en consola.
- El código debe ser original del estudiante.
- Cualquier duda debe consultarse al docente durante el examen.