

[BBVA - JAVASCRIPT+REACT | NIVEL 2]

# Trabajo Integrador Final



## Premisa

Este **Trabajo Final Integrador** tiene como **objetivo** evaluar de manera global todos los conocimientos y habilidades que desarrollaste a lo largo del curso de React. Deberás **crear una aplicación web completa con React que permita visualizar, agregar y gestionar una lista de usuarios y productos**, con navegación entre vistas, consumo de datos desde una API externa, y manejo adecuado de estado, props y comunicación entre componentes.

Esta temática simula funcionalidades reales que cualquier aplicación corporativa necesita: carga dinámica de datos, visualización organizada, navegación por vistas, interacciones controladas, y estructura mantenible.

Esta es tu oportunidad para poner a prueba todo lo aprendido, desarrollar buenas prácticas y demostrar autonomía técnica.



## Requerimientos

Tu aplicación deberá cumplir con los siguientes puntos:

- ✓ Utilizar Create React App o equivalente para inicializar el proyecto.
- ✓ Crear al menos 3 vistas: Inicio, Usuarios y Productos.
- ✓ Implementar rutas navegables con React Router (mínimo 3 rutas, al menos una dinámica).
- ✓ Consumir datos desde una API externa (por ejemplo: JSONPlaceholder, FakeStore API, etc.).
- ✓ Renderizar los datos en forma de listas o tarjetas reutilizando componentes.
- ✓ Usar useState para manejar información local (como filtros, inputs, etc.).
- ✓ Usar useEffect para realizar llamadas a la API.
- ✓ Mostrar estados de carga y mensajes de error ante fallos en la API.



## Acciones:


Te recomendamos seguir estos pasos para abordar el trabajo de manera ordenada:



### Inicializar el proyecto con CRA.







1. Crear componentes genéricos (CardUsuario, CardProducto, Layout, Navbar).
2. Organizar el proyecto con una estructura clara (/components, /pages, /services, etc.).

3. Realizar la conexión a la API utilizando Axios desde useEffect.
4. Usar useState para manejar interacciones (ej. mostrar/ocultar detalles).
5. Implementar lógica de carga y errores (ej. spinner, mensajes de alerta).
6. Crear rutas con React Router para navegar entre vistas (ej. /productos, /usuarios, /detalle/:id).
7. Probar que todos los elementos funcionen de forma integrada.
8. Aplicar buenas prácticas de limpieza de código y modularidad.

 Se recomienda documentar decisiones técnicas en un archivo README.md para que el mentor pueda evaluar el proceso.

## Bonus

Si querés ir un paso más allá, podés incorporar alguna de estas funcionalidades adicionales:

-  Crear un Dashboard con estadísticas básicas (cantidad de usuarios/productos, promedios, etc.).
-  Incluir un sistema de notificaciones temporales al agregar o eliminar productos.
-  Implementar una barra de búsqueda o filtrado dinámico.
-  Utilizar GitHub Copilot para ayudarte a crear pruebas con Jest.
-  Implementar un ejemplo simple de Context API para evitar props drilling innecesario.
-  Agregar un toggle de tema claro/oscuro utilizando Context.

## Aclaraciones:

- El proyecto debe estar hecho completamente en React.
- Se debe utilizar React Router para la navegación.
- Se espera al menos una llamada a una API real con datos dinámicos.
- El código debe ser claro, comentado y organizado en carpetas.
- Se recomienda usar Visual Studio Code y GitHub.
- Es obligatorio el uso de componentes funcionales, hooks y React Router.
- El diseño visual puede ser simple, pero debe ser funcional, limpio y ordenado.
- La app debe correr sin errores y mostrar mensajes claros si hay fallos.

## Entregables

- Repositorio en GitHub con el proyecto completo.
- Archivo README.md que incluya:
  - ◆ Breve descripción del proyecto
  - ◆ Instrucciones para correrlo localmente
  - ◆ Listado de funcionalidades implementadas
  - ◆ Bonus (si corresponde)

BBVA *by* M

