UNIVERSIDAD TECNOLÓGICA NACIONAL

Desarrollo de una Aplicación Web con React.js

PoC

Profesores:

Bressano, Mario

Otaduy, Andrés

Comisión: 302

Alumnos:

Andrada, Gastón

Cinel, Santiago

Fermanelli, Sebastián

Karlen Aguirre, Esteban

Indice

1. Introducción

1.0.1 React.js y su importancia en el desarrollo web

React.js es una biblioteca de JavaScript utilizada para construir interfaces de usuario interactivas y dinámicas para aplicaciones web. Fue desarrollada por Facebook y es ampliamente utilizada en la industria para crear aplicaciones web modernas y escalables. React.js se basa en la idea de componentes reutilizables, lo que permite dividir la interfaz de usuario en pequeños componentes independientes y gestionar su estado de manera eficiente.

1.0.2 Propósito del proyecto Proof of Concept

El propósito de este proyecto Proof of Concept (PoC) es explorar las capacidades y ventajas de React.js en el contexto de nuestro entorno de desarrollo específico. A través de la implementación de un prototipo funcional, buscamos evaluar cómo React.js puede mejorar la eficiencia del desarrollo, la escalabilidad y la experiencia del usuario en nuestras aplicaciones web.

1.0.3 Objetivos y metas que se buscan alcanzar con la PoC

1. Evaluar la productividad: Determinar la eficiencia y la rapidez con la que podemos desarrollar componentes y aplicaciones utilizando React.js.

2. Comprobar la reutilización de componentes: Analizar la capacidad de React.js para permitir la creación y reutilización efectiva de componentes, lo que puede mejorar la mantenibilidad y la escalabilidad del código.

3. Evaluar la optimización de rendimiento: Medir cómo React.js gestiona y optimiza la representación del DOM, lo que puede impactar positivamente en la velocidad y la fluidez de la aplicación.

4. Explorar la integración con otras tecnologías: Investigar la facilidad de integración de React.js con otras herramientas y tecnologías relevantes para nuestro stack de desarrollo.

Al alcanzar estos objetivos, esperamos obtener una comprensión clara de cómo React.js puede potenciar nuestro proceso de desarrollo y cómo puede adaptarse a nuestras necesidades específicas en la creación de aplicaciones web escalables y eficientes.

2. Informe

2.0.1 Descripción de las características y funcionalidades que se quieren demostrar

1. Componentización: Mostrar cómo React.js permite dividir la interfaz de usuario en componentes autónomos y reutilizables, facilitando así la construcción de aplicaciones escalables y mantenibles.

2. Estado y Props: Explorar la gestión del estado y las propiedades (Props) en React.js para permitir una comunicación efectiva entre componentes y una actualización dinámica de la interfaz de usuario en respuesta a cambios de datos.

3. Virtual DOM: Demostrar el concepto del Virtual DOM, explicando cómo React.js optimiza las actualizaciones del DOM al trabajar con una representación virtual del mismo, lo que mejora el rendimiento de la aplicación.

4. Ciclo de vida del componente: Examinar el ciclo de vida de un componente en React.js y cómo podemos aprovecharlo para ejecutar lógica específica en diferentes etapas del ciclo de vida.

5. Manejo de eventos: Ilustrar cómo React.js maneja y responde a eventos del usuario de manera eficiente, garantizando una experiencia interactiva y receptiva.

2.1. Tecnologías utilizadas

2.1.1 React.js: Por qué se eligió esta biblioteca

React.js es la biblioteca elegida para este proyecto por varias razones clave:

- Eficiencia y Rendimiento: React.js utiliza un modelo de representación virtual (Virtual DOM) que mejora la eficiencia al minimizar las actualizaciones del DOM real, lo que resulta en una aplicación más rápida y receptiva.

- Componentización y Reutilización: La capacidad de React.js para descomponer la interfaz de usuario en componentes reutilizables fomenta la modularidad y facilita la construcción y el mantenimiento de aplicaciones complejas.

- Gestión Efectiva del Estado: React.js ofrece un manejo eficiente del estado de la aplicación y una propagación de cambios controlada a través de los componentes, lo que simplifica la gestión de datos y su representación en la interfaz de usuario.

- Comunidad Activa y Soporte Sólido: React.js cuenta con una comunidad vibrante y activa, lo que garantiza una amplia documentación, tutoriales, librerías y actualizaciones constantes que mantienen la biblioteca actualizada y segura.

- Flexibilidad y Escalabilidad: React.js es altamente flexible y puede integrarse fácilmente con otras tecnologías, lo que lo convierte en una elección ideal para proyectos de diferentes escalas y requisitos.

2.1.2 Otras tecnologías y herramientas que se utilizarán en el proyecto

Además de React.js, utilizaremos las siguientes tecnologías y herramientas para desarrollar la PoC:

- Node.js: Utilizaremos Node.js para ejecutar el entorno de desarrollo y construir aplicaciones basadas en JavaScript en el lado del servidor.

- npm (Node Package Manager): Utilizaremos npm para gestionar las dependencias de nuestro proyecto, lo que facilita la instalación y actualización de bibliotecas y herramientas necesarias.

- Vite: Se usará Vite para ejecutar un comando de construcción de la aplicación, preconfigurado para generar activos estáticos altamente optimizados para producción.

- ESLint y Prettier: Estas herramientas nos ayudarán a mantener un código limpio, consistente y sin errores a lo largo del proyecto, siguiendo buenas prácticas y estándares de codificación.

Estas tecnologías y herramientas se complementan entre sí para garantizar un flujo de desarrollo eficiente y la implementación exitosa de la Proof of Concept (PoC) con React.js.

2.2. Arquitectura y diseño

2.2.1 Descripción de la arquitectura de la aplicación React.js

La arquitectura de la aplicación React.js seguirá un patrón de arquitectura de componentes. Este patrón se basa en la creación y gestión de componentes reutilizables, cada uno con su propia lógica y presentación. La estructura seguirá el principio de un solo punto de verdad (Single Source of Truth) para el estado de la aplicación, lo que significa que el estado se gestionará centralmente y se pasará a los componentes como propiedades.

2.2.2 Diseño de componentes principales y su relación con la estructura de la aplicación

Componente 1: App

Este será el componente principal que actúa como contenedor para toda la aplicación.

Contendrá otros componentes y se encargará de distribuir el estado y las propiedades a los componentes hijos según sea necesario.

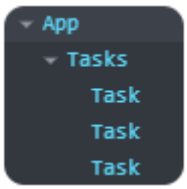
Componente 2: Tasks

Responsable de mostrar la lista de tareas, contendrá una lista con los elementos a mostrar (cada tarea).

Componente 3: Task

Es el componente mínimo de la aplicación el cual representa una tarea individual.

2.2.3 Esquema de la arquitectura y diseño

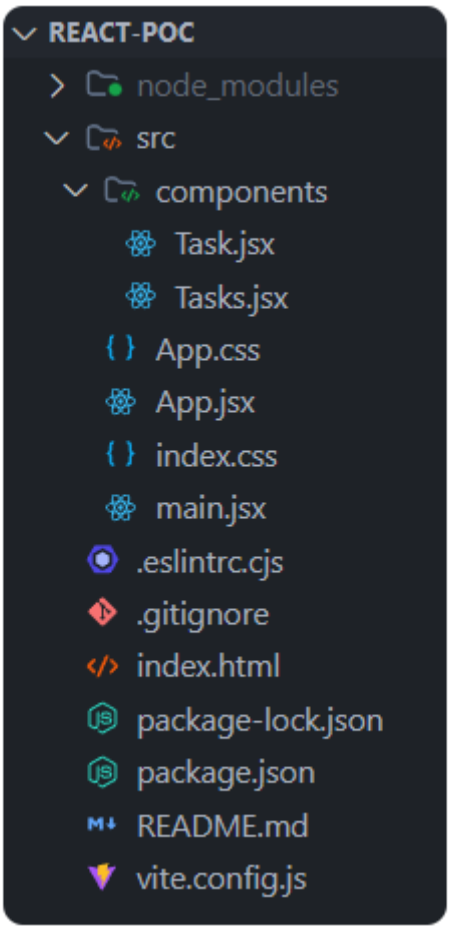


Este esquema muestra la relación jerárquica entre los componentes principales de la aplicación React.js y cómo están organizados en la estructura de la aplicación.

2.3. Implementación

2.3.1 Estructura de directorios y archivos del proyecto React.js

La estructura de directorios y archivos del proyecto React.js se organizará de la siguiente manera:



- components/: Directorio que contiene los componentes de la aplicación.

- App.jsx: Componente principal que renderiza otros componentes y maneja el estado de la aplicación.

- main.jsx: Punto de entrada de la aplicación donde se renderiza el componente App.

- package.json: Archivo que lista las dependencias y configuraciones del proyecto.

- .eslintrc.cjs: Configuración de ESLint para identificar problemas en el código JavaScript y mantener consistencia en el estilo de codificación en un proyecto.

2.3.2 Desarrollo de componentes y su funcionalidad

Cada componente tendrá su propia funcionalidad específica:

- Tasks.jsx: Componente que muestra la lista de tareas.

- Task.jsx: Componente que representa una tarea individual.

2.3.3 Integración de componentes y manejo de estados

El componente App actuará como el contenedor principal e integrará todos los componentes mencionados. Este componente también manejará el estado de la aplicación y pasará las propiedades y el estado requerido a los componentes hijos.

2.3.4 Explicación de cualquier lógica o algoritmo relevante utilizado

Dentro de los componentes, se pueden implementar lógicas específicas según las necesidades de la aplicación. Por ejemplo, la lógica para cambiar dinámicamente el contenido en App.jsx basado en la selección del usuario puede implicar el uso de estados y funciones de control de flujo.

3. Conclusiones

3.0.1 Logros alcanzados y metas cumplidas

Durante el desarrollo de esta Proof of Concept (PoC), logramos los siguientes hitos significativos:

- Implementación Exitosa: Se logró una implementación exitosa de la aplicación utilizando React.js, demostrando la capacidad de la tecnología para crear interfaces de usuario interactivas y eficientes.

- Componentización y Reutilización: Se demostró la eficacia de la componentización y la reutilización de código en React.js, facilitando la construcción y el mantenimiento de la aplicación.

- Optimización de Rendimiento: Se optimizaron componentes para mejorar el rendimiento, lo que resultó en una experiencia de usuario más fluida y receptiva.

3.0.2 Desafíos enfrentados durante el desarrollo

Durante el desarrollo de la PoC, nos encontramos con ciertos desafíos:

- Curva de Aprendizaje: La curva de aprendizaje inicial de React.js para algunos miembros del equipo fue un desafío, ya que implicaba un cambio en la forma de pensar sobre el desarrollo de aplicaciones web.

- Manejo del Estado: Aunque React.js facilita el manejo del estado, determinar la mejor manera de gestionar el estado global y local para una aplicación específica presentó ciertos desafíos.

- Optimización del Rendimiento: La optimización del rendimiento de la aplicación fue un desafío, especialmente en componentes que requerían una actualización frecuente.

3.0.3 Lecciones aprendidas y posibles mejoras para futuras iteraciones

- Planificación y Capacitación: Es crucial invertir tiempo en la planificación adecuada y en la capacitación del equipo antes de comenzar a trabajar con nuevas tecnologías como React.js.

- Optimización Continua: La optimización del rendimiento debe ser una preocupación constante y se deben realizar pruebas y ajustes continuos para garantizar una experiencia de usuario óptima.

- Feedback y Colaboración: Fomentar la comunicación efectiva y el feedback entre los miembros del equipo es esencial para abordar desafíos y tomar decisiones informadas.

3.0.4 Reflexión sobre la viabilidad de la idea y la utilidad de React.js en su implementación

Basándonos en esta PoC, concluimos que la adopción de React.js es altamente viable para la implementación de la idea propuesta. React.js demostró ser una herramienta potente y eficiente para desarrollar aplicaciones web interactivas y escalables. Su capacidad para organizar el código en componentes reutilizables y optimizar el rendimiento hace que sea una elección sólida para futuros proyectos.

4. Referencias

React.js Official Documentation.

Vite Official Documentation.