تطوير الواجهات الامامية بواسطة react

REACT FRONT-END DEVELOPMENT

اعداد وتقديم: م. محمد أسامة بيطار
الهاتف: 0932735606 | تاريخ:2024

المحتويات

محتويات الكورس:

الجلسة 1: مقدمة عامة عن الويب والفرونت إند، ومكاتب الفرونت إند (3 ساعات)

- **المحتوى:**
  - **مقدمة عن الويب (45 دقيقة):**
    - كيف يعمل الويب؟ مقدمة عنHTML ، CSS، JavaScript.
    - دور الفرونت إند في تطوير الويب.
  - **مقدمة عن الفرونت إند (45 دقيقة):**
    - نظرة عامة على مكاتب الفرونت إند الشائعةReact) ، Angular، Vue).
    - لماذا نختارReact ؟ مقارنة مختصرة بين المكتبات.
  - **استراحة (30 دقيقة)**
  - **بدء مشروع React (60 دقيقة):**
    - تثبيت React.
    - إنشاء مشروع React بسيط باستخدام create-react-app.
    - مقدمة سريعة عن بنية مشروعReact.

الجلسة 2: مقدمة إلى React والكومبوننت (3 ساعات)

- **المحتوى:**
  - **مقدمة إلى React (45 دقيقة):**
    - ما هيReact ؟ ولماذا تستخدم؟
    - المفاهيم الأساسية في) React الكومبوننت، الJSX، التصيير.(
  - **مفهوم الكومبوننت (45 دقيقة):**
    - ما هو الكومبوننت وكيفية بنائه.
    - أهمية تقسيم الصفحة إلى كومبوننتات.
    - أمثلة على بناء كومبوننتات بسيطة.
  - **استراحة (30 دقيقة)**
  - **العمل مع Props (60 دقيقة):**
    - ما هي الProps؟
    - كيفية تمرير البيانات بين الكومبوننتات باستخدام الProps.

- أمثلة عملية وتمرين قصير على الـProps.

الجلسة 3: تمرين عملي على الكومبوننت، الProps، ومقدمة في إدارة الحالة (3 ساعات)

- **المحتوى:**
  - **تمرين عملي على الكومبوننت والـ Props (90 دقيقة):**
    - تنفيذ مشروع صغير يستخدم الكومبوننتات والـ Propsبشكل مكثف.
    - حل المشاكل التي قد تظهر أثناء التمرين.
  - **استراحة (30 دقيقة)**
  - **مقدمة في إدارة الحالة (State Management) (60 دقيقة):**
    - تعريف الـ Stateوأهميته في React.
    - استخدام الـuseState hook.
    - تحضير الطلاب لتمارين قادمة على إدارة الحالة.

الجلسة 4: إدارة الحالة باستخدام useState (3 ساعات)

- **المحتوى:**
  - **استخدام الـ useState (60 دقيقة):**
    - شرح مفصل لكيفية استخدام الـ useStateلإدارة الحالة.
    - أمثلة على إنشاء وتحديث الـState.
  - **استراحة (30 دقيقة)**
  - **تمرين عملي على useState (90 دقيقة):**
    - تنفيذ تطبيق بسيط يتضمن إدارة حالة متعددة باستخدام useState.
    - العمل على التعامل مع التحديات المتعلقة بإعادة التصيير (re-rendering).

الجلسة 5: التعامل مع الآثار الجانبية باستخدام useEffect (3 ساعات)

- **المحتوى:**
  - **مقدمة إلى useEffect (45 دقيقة):**
    - ما هو الـuseEffect؟ ولماذا نحتاجه؟
    - كيفية التعامل مع الآثار الجانبية (side effects) في React.
  - **أمثلة على استخدام useEffect (45 دقيقة):**
    - تطبيقات عملية مثل جلب البيانات (data fetching) والتفاعل مع DOM.

- **استراحة (30 دقيقة)**
- **تمرين عملي على useEffect (60 دقيقة):**
  - إنشاء تطبيق يستخدم useEffect لتنفيذ آثار جانبية متعددة.
  - حل مشاكل شائعة مثل التعامل مع الدورات اللانهائية.(infinite loops)

الجلسة 6: إدارة الحالة المتقدمة باستخدام useContext (3 ساعات)

- **المحتوى:**
  - **مقدمة إلى useContext (45 دقيقة):**
    - التعرف على useContext وأهميته في مشاركة البيانات عبر الكومبونتات.
    - كيفية إعداد context واستخدامه في التطبيق.
  - **أمثلة عملية على useContext (45 دقيقة):**
    - إنشاء تطبيق صغير يستخدم context لمشاركة الحالة بين عدة كومبونتات.
  - **استراحة (30 دقيقة)**
  - **تمرين عملي على useContext (60 دقيقة):**
    - بناء تطبيق معقد يتضمن إدارة حالة عامة باستخدام.useContext
    - استكشاف استخدام context في سيناريوهات مختلفة.

الجلسة 7: أساسيات Routing في React (3 ساعات)

- **المحتوى:**
  - **مقدمة إلى التوجيه (Routing) (45 دقيقة):**
    - شرح مفهوم التوجيه وأهميته في تطبيقات الويب.
    - مقدمة إلى مكتبة.React Router
  - **إنشاء مسارات بسيطة (45 دقيقة):**
    - كيفية إنشاء مسارات (routes) بسيطة.
    - التعامل مع الروابط (links) والتنقل بين الصفحات.
  - **استراحة (30 دقيقة)**
  - **تمرين عملي على Routing (60 دقيقة):**
    - بناء تطبيق متعدد الصفحات باستخدام.React Router
    - التعامل مع الروابط الديناميكية والتوجيه الشرطي.

الجلسة 8: تطبيق متقدم على Routing وإدارة الحالة (3 ساعات)

- **المحتوى:**
  - **توجيه متقدم (45 دقيقة):**
    - إنشاء مسارات متداخلة (nested routes) ومسارات ديناميكية.
    - التعامل مع الحالات الخاصة مثل 404.Not Found
  - **ربط Routing مع إدارة الحالة (45 دقيقة):**
    - كيفية مشاركة الحالة عبر المسارات المختلفة.
    - أمثلة على دمج useContext مع.Routing
  - **استراحة (30 دقيقة)**
  - **تمرين عملي متقدم (60 دقيقة):**
    - بناء تطبيق معقد يدمج بين التوجيه وإدارة الحالة العامة.

الجلسة 9: التعامل مع النموذج (Forms) في React (3 ساعات)

- **المحتوى:**
  - **إدارة النماذج (45 دقيقة):**
    - كيفية إنشاء ومعالجة النماذج في.React
    - التعامل مع بيانات النماذج والتحقق من الإدخال.(validation)
  - **استراتيجية إدارة حالة النماذج (45 دقيقة):**
    - استخدام useState و useEffectلإدارة حالة النماذج.
    - أمثلة عملية على بناء نماذج معقدة.
  - **استراحة (30 دقيقة)**
  - **تمرين عملي على النماذج (60 دقيقة):**
    - بناء نموذج تسجيل مستخدم وإدارة حالته.
    - التعامل مع التحقق من المدخلات (validation) بشكل ديناميكي.

الجلسة 10: التفاعل مع API (جلب البيانات) (3 ساعات)

- **المحتوى:**
  - **مقدمة إلى جلب البيانات (45 دقيقة):**
    - كيفية جلب البيانات من API باستخدام fetch أو.Axios
    - التعامل مع الطلبات غير المتزامنة.(asynchronous requests)

- **استخدام useEffect لجلب البيانات (45 دقيقة):**
  - أمثلة عملية على جلب البيانات وعرضها في التطبيق.
  - التعامل مع حالات التحميل (loading) والأخطاء.(error handling)
- **استراحة (30 دقيقة)**
- **تمرين عملي على جلب البيانات (60 دقيقة):**
  - بناء تطبيق يعرض بيانات من API خارجي.
  - تحسين تجربة المستخدم من خلال إدارة حالات التحميل والأخطاء.

الجلسة 11: تحسين الأداء في React (3 ساعات)

- **المحتوى:**
  - **مفاهيم أساسية لتحسين الأداء (45 دقيقة):**
    - التعرف على أسباب مشاكل الأداء في تطبيقات.React
    - استخدام React.memo و useCallback وuseMemo لتحسين الأداء.
  - **أدوات قياس الأداء (45 دقيقة):**
    - مقدمة إلى أدوات قياس الأداء مثل.React DevTools
    - كيفية التعرف على الزوايا الميتة (bottlenecks) وتحسينها.
  - **استراحة (30 دقيقة)**
  - **تمرين عملي على تحسين الأداء (60 دقيقة):**
    - تحسين تطبيق موجود باستخدام الأدوات والمفاهيم التي تم تعلمها.
    - مراجعة نتائج التحسينات.

الجلسة 12: إدارة المشروع النهائي - التخطيط والبدء (3 ساعات)

- **المحتوى:**
  - **التخطيط للمشروع النهائي (60 دقيقة):**
    - تحديد أهداف المشروع ومتطلباته.
    - تقسيم المهام وتحديد الأدوار.
  - **استراحة (30 دقيقة)**
  - **البدء في تطوير المشروع (90 دقيقة):**
    - إنشاء بنية المشروع وتوزيع المهام بين الفريق.
    - بدء العمل على العناصر الأساسية للمشروع.

الجلسة 13: تنفيذ المشروع النهائي - الجزء الأول (3 ساعات)

- **المحتوى:**
  - **تنفيذ الجزء الأول من المشروع (120 دقيقة):**
    - بناء الكومبونتات الأساسية وتطبيق المفاهيم المتعلمة سابقًا.
    - متابعة تقدم العمل والتأكد من توافق العناصر.
  - **استراحة (30 دقيقة)**

الجلسة 14: تنفيذ المشروع النهائي - الجزء الثاني (3 ساعات)

- **المحتوى:**
  - **استمرار تنفيذ المشروع (120 دقيقة):**
    - إكمال العمل على الأجزاء المتبقية.
    - دمج المكونات واختبار التطبيق بشكل مستمر.
  - **استراحة (30 دقيقة)**

الجلسة 15: مراجعة واختبار المشروع النهائي (3 ساعات)

- **المحتوى:**
  - **اختبار المشروع (90 دقيقة):**
    - إجراء اختبارات شاملة للتطبيق.
    - حل المشاكل المحتملة وتحسين الأداء.
  - **استراحة (30 دقيقة)**
  - **مراجعة الكود وتوثيق المشروع (60 دقيقة):**
    - مراجعة الكود لضمان الجودة والالتزام بالمعايير.
    - توثيق المراحل النهائية للمشروع.

الجلسة 16: عرض المشروع النهائي ومناقشة (3 ساعات)

- **المحتوى:**
  - **عرض المشروع النهائي (90 دقيقة):**
    - تقديم المشروع من قبل الطلاب.
    - مناقشة القرارات التقنية والنهج المتبع.
  - **استراحة (30 دقيقة)**
  - **تقديم الملاحظات والتقييم الختامي (60 دقيقة):**

- تقييم الأداء والعمل الجماعي.
- مناقشة التجربة العامة والتعلم من الدورة.

الفصل الأول:

مقدمة إلى تطوير الواجهات الامامية في الويب

مقدمة عامة:

دليلكم الشامل لتعلم واحدة من أقوى مكتبات JavaScript لبناء واجهات المستخدم التفاعلية. في عالم تطوير الويب الذي يتطور بسرعة، تبرز React كمكتبة متميزة تقدم حلولاً متقدمة وفعالة لبناء تطبيقات ويب عالية الجودة. هذا الكتاب مصمم ليكون مرجعًا شاملًا لكم، يوفر لكم الأدوات والمعرفة التي تحتاجونها لتصبحوا مطورين ويب بارعين.

أساسيات تطوير الويب:

- **كيف يعمل الويب؟**
  - عندما تقوم بكتابة عنوان موقع في متصفحك، يرسل المتصفح طلبًا إلى خادم الويب (Server) الذي يحتوي على ملفات الموقع. يقوم الخادم بإرسال هذه الملفات إلى المتصفح، الذي يعرضها كصفحة ويب.
- **لماذا يوجد Frontend وBackend؟**
  - **(Frontendالواجهة الأمامية)** يتعامل مع تصميم وعرض المعلومات التي يرى ويتفاعل معها المستخدم.
  - **(Backendالواجهة الخلفية)** يتعامل مع تخزين البيانات وإدارتها وتنفيذ المنطق المعقد خلف الكواليس.
- **لماذا يوجد العديد من الطرق واللغات البرمجية التي يمكن استخدامها؟**
  - كل لغة أو إطار عمل له خصائصه الخاصة التي تجعله مناسبًا لأنواع مختلفة من التطبيقات. هذا التنوع يوفر خيارات تناسب احتياجات محددة ويزيد من كفاءة التطوير.
- **ما الفرق بين تطوير الويب وتصميمه؟**
  - **تصميم الويب :**يركز على الشكل والمظهر العام للصفحات، مثل الألوان والخطوط.
  - **تطوير الويب :**يترجم التصميم إلى كود يمكن أن يعمل على المتصفح ويضيف التفاعل والوظائف.

التعلم وتطوير المهارات:

- **من أين أبدأ مساري في تعلم تطوير الويب؟**
  - ابدأ بتعلم الأساسيات مثلHTML ، CSS، و .JavaScriptبعد ذلك، يمكنك الانتقال إلى تعلم مكتبات وأطر عمل مثل.React
- **هل هذا الكورس هو الأفضل لمساري المهني؟**
  - إذا كان هدفك تعلم كيفية بناء واجهات المستخدم التفاعلية وتطوير مهاراتك فيReact ، فهذا الكورس مصمم خصيصًا لك.

- **لماذاReact ؟**
  - Reactتوفر طريقة مرنة وسهلة لبناء تطبيقات الويب التفاعلية من خلال مكونات قابلة لإعادة الاستخدام وأداء عالٍ.
- **ما الذي يميز React عن غيرها من بيئات العمل؟**
  - Reactتميز نفسها بمرونتها وأداءها العالي من خلال استخدام Virtual DOM الذي يعزز سرعة تحديث واجهة المستخدم.

الاستفادة من الكورس

- **ما هي ميزات React في سوق العمل؟**
  - Reactمهارة مطلوبة بشدة في سوق العمل، لأنها تُستخدم من قبل العديد من الشركات الكبرى في تطوير تطبيقات الويب.
- **كيف أستطيع أن أستفيد كامل الفائدة من الكورس؟**
  - لتستفيد بالكامل، قم بممارسة الأكواد، شارك في المشاريع العملية، واطرح الأسئلة عند الحاجة.
- **في نهاية الكورس، ما الذي أستطيع عمله؟ هل أنا قادر على الدخول إلى سوق العمل فورًا؟**
  - بنهاية الكورس، ستكون قادرًا على بناء تطبيقات ويب تفاعلية باستخدام .React قد تحتاج إلى بعض الخبرة العملية لتكون جاهزًا تمامًا لدخول سوق العمل.
- **ماذا أفعل بعد نهاية الكورس؟**
  - يمكنك العمل على مشاريع إضافية، تعلم أدوات ومكتبات أخرى مثلRedux ، واستكشاف مجالات جديدة في تطوير الويب.

أسئلة إضافية:

- **ما هي المهارات الأساسية التي سأتعلمها خلال هذا الكورس؟**
  - تعلم كيفية بناء مكوناتReact ، إدارة الحالة، التعامل معAPI ، وتطبيق الأساسيات لبناء واجهات مستخدم تفاعلية.
- **كيف يساعدني هذا الكورس في بناء مشاريع حقيقية؟**
  - سيوفر لك الكورس المهارات الأساسية لبناء تطبيقات حقيقية، من التعلم الأساسي إلى مشاريع تطبيقية متقدمة.
- **هل سأتمكن من العمل على مشاريع خاصة بي أثناء الكورس؟**
  - نعم، ستتمكن من العمل على مشاريع صغيرة ضمن الكورس وكذلك مشاريع أكبر في نهاية الدورة.

- **كيف يساهم تعلم React في تحسين فرصي الوظيفية؟**
  - Reactمهارة مطلوبة في سوق العمل، تعلمها يفتح لك أبوابًا واسعة للعمل في شركات تكنولوجيا ومشاريع تطوير الويب.
- **هل هذا الكورس مناسب للمبتدئين أم يتطلب خبرة سابقة في البرمجة؟**
  - الكورس مصمم للمبتدئين، لذا فهو يبدأ من الأساسيات ويأخذك خطوة بخطوة في تعلم.React

نظرة عامة على الويب وطريقة عمله:

كيف يعمل الويب:

عندما نتحدث عن كيفية عمل الويب، نحن نشير إلى الطريقة التي تتفاعل بها مختلف المكونات لتقديم تجربة مستخدم فعالة. الإنترنت عبارة عن شبكة عالمية تتكون من مجموعة من الخوادم (Servers)التي تستضيف مواقع الويب وتخزين البيانات، وأجهزة الكمبيوتر والمعدات التي يمكن الوصول إلى هذه المواقع عبر متصفحات الويب.

الـ Frontend والـ Backend وقواعد البيانات:

1. **الـ Frontend الواجهة الأمامية:**
   - الـ Frontend هو الجزء الذي يتفاعل معه المستخدم مباشرة. يتضمن تصميم وتطوير واجهات المستخدم التي تشمل النصوص، الصور، الأزرار، والروابط. يُبنى الـ Frontend باستخدام لغات البرمجة مثل HTML و CSSو.JavaScript
   - **HTML**يُستخدم لبناء هيكل الصفحة.
   - **CSS**يُستخدم لتنسيق وتجميل الصفحة.
   - **JavaScript**يُستخدم لإضافة التفاعل والديناميكية.
2. **الـ Backend الواجهة الخلفية:**
   - الـ Backend هو الجزء الذي يحدث خلف الكواليس ويتعامل مع البيانات والخوادم. يتولى الـ Backend معالجة الطلبات من الـFrontend ، إدارة قواعد البيانات، وتنفيذ العمليات الحسابية واللوجستية. تُبنى الـ Backend باستخدام لغات برمجة مثل Node.js، Python، Ruby، PHP، أو.Java
   - الـ Backend يستقبل الطلبات من الـ Frontend ويقوم بإرسال الاستجابات بعد معالجة البيانات اللازمة.
3. **قواعد البيانات:(Databases)**
   - قواعد البيانات هي المكان الذي تُخزّن فيه البيانات بشكل منظم. يمكن أن تكون قواعد البيانات من نوع SQL مثلMySQL ، (أوPostgreSQL) NoSQL مثل MongoDB).

- عندما يقوم المستخدم بإدخال بيانات عبر واجهة المستخدم، يتم إرسال هذه البيانات إلى الBackend ، الذي يقوم بدوره بتخزينها في قاعدة البيانات. وعندما يحتاج التطبيق إلى استرجاع البيانات، يقوم الـ Backend بالوصول إلى قاعدة البيانات وجلب المعلومات المطلوبة.

4. **كيف يتفاعل الـ Frontend و الـ Backend وقواعد البيانات؟**

عندما يقوم المستخدم بتفاعل مع واجهة المستخدم(Frontend) ، يتم إرسال طلب إلى الـ Backend عبر HTTP أو .HTTPS على سبيل المثال، عندما يضغط المستخدم على زر لتسجيل الدخول، يتم إرسال بيانات تسجيل الدخول إلى الـ .Backend يقوم الـ Backend بمعالجة هذه البيانات، مثل التحقق من صحة البيانات أو إجراء عمليات على قاعدة البيانات. بعد المعالجة، يقوم الـ Backend بإرسال الاستجابة إلى الFrontend ، الذي يقوم بتحديث واجهة المستخدم بناءً على النتائج.

مقدمة عن React

ما هو Node.js ؟

Node.jsهو بيئة تشغيل لـ JavaScript تُستخدم لبناء تطبيقات خادم (Server) على جانب الخادم بدلاً من المتصفح. يُمكنك التفكير في Node.js كأداة تتيح لك تشغيل كود JavaScript على الخادم، وليس فقط في المتصفح. هذا يعني أنك يمكن أن تستخدم JavaScript لإنشاء تطبيقات ويب كاملة، بما في ذلك الخوادم وقواعد البيانات.

كيف يعمل Node.js ؟

- **تشغيل JavaScript على الخادم :**بدلاً من تشغيل JavaScript فقط على المتصفح، يمكن لـ Node.js تشغيله على الخادم، مما يسمح بكتابة الكود الذي يتعامل مع الطلبات من المتصفحات، ويدير قواعد البيانات، ويتفاعل مع.APIs

- **غير متزامن وقائم على الأحداث :**يستخدم Node.js نموذج البرمجة غير المتزامن (Asynchronous)وقائم على الأحداث .(Event-driven) هذا يعني أن Node.js يمكنه التعامل مع العديد من الطلبات في وقت واحد بكفاءة عالية دون أن يتوقف عن العمل.

كيف يتفاعل Node.js مع React ؟

- **بناء :API** في تطبيقات الويب الحديثة، يستخدم Node.js عادةً لبناء واجهات برمجة التطبيقات (APIs) التي يتواصل معها تطبيق .React على سبيل المثال، إذا كان لديك تطبيق React على الواجهة الأمامية، فيمكنه إرسال طلبات إلى API مبنية باستخدام Node.jsللحصول على أو إرسال بيانات.

- **خادم التطوير :**يمكن استخدام Node.js كخادم لتشغيل تطبيقات React أثناء عملية التطوير. توفر أدوات مثل Create React App أدوات تطوير تعمل على خادم Node.js محلي لتسهيل عملية بناء وتطوير التطبيقات.

متى وكيف نشأتReact ؟

Reactتم اختراعه بواسطة فريق فيسبوك في عام 2011 وتم إصداره للعامة في عام 2013. كان الهدف من تطوير React هو حل بعض المشاكل التي كان يواجهها فريق فيسبوك أثناء تطوير واجهات المستخدم لتطبيقات الويب الكبيرة والمعقدة.

لماذا تم اختراعReact ؟

- **مشاكل في الأداء :**في ذلك الوقت، كانت التحديثات على واجهات المستخدم تتطلب إعادة تحميل الصفحة بالكامل، مما أدى إلى بطء الأداء وتجربة مستخدم غير سلسة.
- **التعقيد في إدارة المكونات :**كانت إدارة حالة المكونات والتفاعل بينها أمرًا معقدًا، خاصة في التطبيقات الكبيرة.

كيف حلت React هذه المشاكل؟

- **مفهوم المكونات :(Components)** قدمت React مفهوم المكونات القابلة لإعادة الاستخدام، مما سمح بتقسيم الواجهة إلى أجزاء أصغر وأكثر تنظيمًا، مما يسهل إدارتها.
- **Virtual DOM:**قدمت React تقنية الVirtual DOM ، التي تعزز الأداء من خلال تحديث أجزاء فقط من الصفحة بدلاً من إعادة تحميل الصفحة بالكامل.

كيف تعمل :React نظرة تفصيلية

Reactهي مكتبة JavaScript تُستخدم لبناء واجهات المستخدم التفاعلية، وتم تطويرها بواسطة فيسبوك لتسهيل عملية تطوير تطبيقات الويب. تعمل React بناءً على مفهوم المكونات (Components)، التي هي أجزاء مستقلة وقابلة لإعادة الاستخدام من الواجهة.

مفاهيم عامة حولها:

1. **المكونات:(Components)**
   - فيReact ، يُنظر إلى واجهة المستخدم على أنها مجموعة من المكونات المستقلة. كل مكون يمكن أن يحتوي على بياناته الخاصة (state) وتلقي البيانات من المكونات الأخرى عبر الخصائص.(props)
   - تُبنى المكونات باستخدام JavaScript وJSX، وهي لغة تُدمج JavaScript معHTML ، مما يسمح لك بكتابة كود يشبه HTML داخل ملفات.JavaScript
2. **الVirtual DOM:**
   - واحدة من الميزات الرائدة في React هي استخدام الـ .Virtual DOM بدلاً من تحديث DOM مباشرة في المتصفح (والذي يمكن أن يكون بطيئًا)، تقوم React بإنشاء نسخة افتراضية من .DOM عندما يحدث تغيير في البيانات، تقوم React بتحديث الـ Virtual DOM أولاً ثم تقارن التغييرات مع الـ DOM الأصلي، وتطبق التحديثات فقط على العناصر التي تغيرت. هذا يُحسن الأداء ويجعل التحديثات أسرع.

3. **إدارة الحالة:(State Management)**

   - تُعتبر إدارة الحالة جزءًا أساسيًا من React. يمكن لكل مكون الاحتفاظ بحالته الخاصة، والتي تمثل البيانات التي يمكن أن تتغير بمرور الوقت. عندما تتغير الحالة، تقوم React بإعادة عرض المكون لتحديث واجهة المستخدم. يمكن استخدام Hooks مثل useStateلإدارة الحالة في المكونات الوظيفية.

4. **التفاعل مع الـ:Backend**

   - يمكن لمكونات React إرسال واستقبال البيانات من الـ Backend عبر الطلبات الشبكية (API calls). عادةً ما تُستخدم مكتبات مثل Axios أو Fetch لتنفيذ هذه الطلبات، مما يسمح بتحديث المكونات بناءً على البيانات التي يتم جلبها.

5. **التنقل بين الصفحات:(Routing)**

   - توفر React أدوات لإدارة التنقل بين صفحات متعددة داخل تطبيق واحد باستخدام مكتبات مثل React Router. هذا يتيح لك إنشاء تطبيقات ذات صفحات متعددة دون الحاجة إلى إعادة تحميل الصفحة بالكامل.

اعداد بيئة العمل:

تثبيت Node.js و npm

Node.jsو **npm** هما الأدوات الأساسية التي نحتاجها للعمل مع React. إليك كيفية تثبيتهما:

- **Node.js:**هو بيئة تشغيل JavaScript على الخادم، ويتيح لك تشغيل تطبيقات JavaScript خارج المتصفح. يمكن تحميل Node.js من الموقع الرسمي nodejs.orgأثناء تثبيت Node.js، سيتم أيضًا تثبيت **npm** (Node Package Manager)تلقائيًا. npm هو أداة تُستخدم لإدارة المكتبات والبرامج الإضافية (packages) التي تحتاجها في مشروعك.

كيفية التثبيت:

1. انتقل إلى موقع Node.js.
2. اختر النسخة المناسبة
3. اتبع التعليمات على الشاشة لتثبيت Node.js و npm.

بعد التثبيت، يمكنك التحقق من أن Node.js و npm تم تثبيتهما بنجاح من خلال فتح نافذة الأوامر (Command Prompt)أو الطرفية (Terminal) واستخدام الأوامر التالية:

- node -v
- npm -v

إنشاء أول مشروع React باستخدامCreate React App

Create React App هو أداة تتيح لك بدء مشروع React بسرعة وسهولة، دون الحاجة للقلق بشأن إعداد بيئة العمل المعقدة.

كيفية إنشاء مشروع:React

1. افتح نافذة الأوامر أو الطرفية.
2. انتقل إلى المجلد الذي ترغب في إنشاء مشروعك فيه باستخدام الأمر) cdمثل cd : Documents/Projects).
3. استخدم الأمر التالي لإنشاء مشروع جديد:

```
npx create-react-app my-first-app

cd my-first-app

npm start
```

سيبدأ هذا الأمر الخادم المحلي، وستتمكن من رؤية تطبيق React في متصفحك عبر الذهاب إلى .http://localhost:3000

هيكلية المشروع ومكوناته الأساسية

عندما تنشئ مشروعًا باستخدامCreate React App ، سيتم إنشاء هيكل مشروع قياسي يحتوي على عدة ملفات ومجلدات. فيما يلي نظرة عامة على هيكل المشروع الأساسي:

- **node_modules/**: يحتوي على جميع الحزم والمكتبات التي تم تثبيتها من خلال.npm
- **public/**:
  - **index.html**: ملف HTML الرئيسي الذي يتم تحميله في المتصفح. يحتوي على عنصر يحمل معرف"root" ، وهو المكان الذي سيتم فيه عرض تطبيق React.
  - **favicon.ico**: أيقونة الموقع التي تظهر في شريط العنوان للمتصفح.
- **src/**:
  - **index.js**: الملف الرئيسي الذي يبدأ تشغيل تطبيق .React هنا يتم استدعاء ReactDOM.renderلعرض المكون الرئيسي في .
  - **App.js**: المكون الرئيسي لتطبيقك. يحتوي عادةً على المكون الأساسي الذي ستقوم بتطويره وتخصيصه.
  - **App.css**: ملف CSS الذي يحتوي على أنماط للمكون الرئيسي .App.js
- **.gitignore**: يحتوي على قائمة بالملفات والمجلدات التي يجب تجاهلها عند استخدام نظام التحكم في الإصدارات.Git

- **package.json**:يحتوي على معلومات حول المشروع، مثل اسم المشروع، النسخة، والحزم المستخدمة. كما يحتوي على الأوامر التي يمكن تشغيلها باستخدامnpm ، مثل start، build، و test.
- **package-lock.json**:يُستخدم لتثبيت الإصدارات المحددة من الحزم لضمان استقرار المشروع.

إعداد بيئة العمل باستخدام Visual Studio Code (VS Code)

1. **تنزيل وتثبيت VS Code:**
   - قم بزيارة موقع Visual Studio Code.
   - اختر النسخة المناسبة
   - اتبع التعليمات على الشاشة لتثبيت VS Code.
2. **فتح مشروعك في VS Code:**
   - بعد تثبيت VS Code ، افتح التطبيق.
   - استخدم خيار "فتح مجلد (Open Folder) "لفتح مجلد مشروع React الذي أنشأته باستخدام Create React App.
3. **تثبيت الإضافات الأساسية لـ React:**
   - **ESLint:**أداة لتحليل الكود والكشف عن الأخطاء البرمجية وتقديم تحسينات. قم بتثبيت إضافة ESLint من متجر VS Code عبر البحث عن "ESLint" وتثبيتها.
   - **Prettier:**أداة لتنسيق الكود بشكل آلي. قم بتثبيت إضافة Prettier عبر البحث عن "Prettier" وتثبيتها.
   - **Reactjs Code Snippets:**مجموعة من الاختصارات لكتابة كود React بسرعة. قم بتثبيت إضافة Reactjs Code Snippets عبر البحث عن "Reactjs Code Snippets" وتثبيتها.
   - **Bracket Pair Colorizer:**أداة لتلوين الأقواس المتطابقة، مما يساعد في قراءة الكود بشكل أفضل. قم بتثبيت إضافة Bracket Pair Colorizer عبر البحث عن "Bracket Pair Colorizer"وتثبيتها.

مقدمة إلى JSX ولماذا نستخدمه

JSX (JavaScript XML)هو امتداد لكتابة الكود في React يتيح لك دمج JavaScript مع HTML بشكل سلس. يشبه JSX كتابة HTML داخلJavaScript ، ويجعل عملية إنشاء الواجهات أكثر بديهية وسهلة.

- **لماذا نستخدمJSX ؟**
  - **سهولة القراءة والكتابة :**يجعل JSX كتابة مكونات React أكثر سهولة، لأنك تستطيع كتابة هيكل الواجهة بشكل مشابه لـ.HTML
  - **دمج JavaScript و HTML:** يسمح لك بإدماج كود JavaScript مباشرة داخلHTML ، مما يوفر تجربة أكثر تفاعلية وسرعة.

ما هي الPropsـ ؟

الـ Props (الخصائص) هي اختصار لـ "properties"فيReact ، وهي وسيلة لتمرير البيانات من مكون إلى آخر. تعتبر الـ Props بمثابة وسيلة للتواصل بين المكونات في تطبيق .React يتم تعريف الـ Propsفي المكون الأب ويتم تمريرها إلى المكونات الأبناء، حيث يمكن استخدامها لعرض البيانات أو تنفيذ وظائف محددة.

أهمية الPropsـ

- **تبادل البيانات :**يمكن للمكونات الأب أن تمرر البيانات إلى المكونات الأبناء، مما يتيح لها عرض البيانات أو استخدامها.
- **إعادة الاستخدام :**من خلال استخدام الPropsـ ، يمكن جعل المكونات أكثر مرونة وقابلة لإعادة الاستخدام مع بيانات مختلفة.
- **فصل المسؤوليات :**تساعد الـ Props في فصل المسؤوليات بين المكونات، مما يجعل الكود أكثر تنظيمًا وسهولة في الصيانة.

كيفية استخدام الPropsـ

3.1تمرير Props إلى مكون

عند استخدام مكون فيReact ، يمكنك تمرير خصائص إلى المكون باستخدام سمات .(attributes) إليك كيفية تمرير props لمكون:

مثال 1: تمرير نص إلى مكون

1. **إنشاء مكون فرعي يستقبل:Props**
   - قم بإنشاء ملف جديد باسم Greeting.jsفي مجلد src، وأضف الكود التالي:

```
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

```
export default Greeting;
```

استخدام المكون وتمرير Props من App.js:

- افتح src/App.jsوأضف استخدام مكون Greetingمع تمرير قيمة لخاصية name:

```
import logo from './logo.svg';
import './App.css';
import Greeting from './session one/components/props';
function App() {
  return (
    <div className="App">
      <Greeting name="osama bittar"></Greeting>
    </div>
  );
}

export default App;
```

استخدام الـ Props لتخصيص مكون

يمكنك استخدام الـ Props لتخصيص مكون بطرق مختلفة، مثل تغيير الأنماط أو المحتوى المعروض.

مثال 2: تخصيص نمط المكون

1. **إنشاء مكون يستقبل Props لتغيير النمط:**
   - قم بإنشاء ملف جديد باسم StyledComponent.jsفي مجلد src، وأضف الكود التالي:

```
2. import React from 'react';

function Greeting(props) {
  return <h1 className={props.class}>Hello, {props.name}!</h1>;
}

export default Greeting;
```

استخدام المكون وتمرير Props لتغيير النمط في App.js:

```
import logo from './logo.svg';
import './App.css';
import Greeting from './session one/components/props';
function App() {
  return (
    <div className="App">
      <Greeting name="osama bittar" class="red"></Greeting>
    </div>
  );
}
```

```
export default App;
```

استخدام الـ Props كـFunction

يمكنك تمرير دوال (functions) كمكونات أو خصائص إلى المكونات الأبناء لتنفيذ وظائف محددة.

مثال 3: تمرير دالة كمكون

1. **إنشاء مكون يستقبل دالة كـProp:**
   - قم بإنشاء ملف جديد باسم Button.js في مجلد src، وأضف الكود التالي:

```
import React from 'react';

function Button(props) {
  return <button onClick={props.onClick}>{props.label}</button>;
}

export default Button;
```

استخدام المكون وتمرير دالة كمكون في App.js:

```
import logo from './logo.svg';
import './App.css';
import Button from './session one/components/propsAsFunction';
import Greeting from './session one/components/props';

function App() {
  let x = 10;

  const increment = () => {
    for (let i = 0; i < 10; i++) {
      x += 10;
    }
    console.log(x)
  }


  return (
    <div className="App">
      <Greeting name="osama bittar" className="red" />
      <Button onClick={increment} label="click here" ></Button>
    </div>
  );
}

export default App;
```

تمرين: بناء تطبيق قائمة المنتجات باستخدام الProps

في هذا التمرين، ستقوم بإنشاء تطبيق React يعرض قائمة من المنتجات. ستستخدم الـ **props** لتمرير بيانات المنتجات من مكون رئيسي إلى مكونات فرعية

المتطلبات

1. **مكون رئيسي App:**
   - يحتوي على قائمة من المنتجات (كل منتج يحتوي على gid, name, price).
   - يعرض كل منتج باستخدام مكون فرعي يُدعى ProductCard.
2. **مكون فرعي ProductCard:**
   - يستقبل بيانات المنتج كـ **props** ويعرض تفاصيل المنتج.
   - يجب أن يتضمن المكون عرض اسم المنتج وسعره داخل بطاقة (Card) ذات تنسيق بسيط.

```
import React from 'react';
import ProductCard from './ProductCard';

function App() {
  const products = [
    { id: 1, name: 'Laptop', price: '$999' },
    { id: 2, name: 'Smartphone', price: '$499' },
    { id: 3, name: 'Headphones', price: '$199' },
  ];

  return (
    <div>
      <h1>Product List</h1>
      {products.map(product => (
        <ProductCard
          key={product.id}
          name={product.name}
          price={product.price}
        />
      ))}
    </div>
  );
}

export default App;
import React from 'react';
```

```jsx
function ProductCard({ name, price }) {
  return (
    <div style={{ border: '1px solid #ddd', padding: '16px', margin:
'8px', borderRadius: '4px' }}>
      <h2>{name}</h2>
      <p>Price: {price}</p>
    </div>
  );
}

export default ProductCard;
```

الفصل الثاني:

إدارة الحالة

مقدمة عن إدارة الحالة:

تعريف الحالة:(State)

فيReact ، تعتبر "الحالة" من أهم المفاهيم الأساسية التي يعتمد عليها إطار العمل لبناء تطبيقات تفاعلية وقابلة للتحديث. الحالة هي نوع من البيانات يُستخدم في تخزين معلومات ديناميكية داخل المكونات (components) التي قد تتغير بمرور الوقت استجابةً لتفاعلات المستخدم أو لعمليات النظام.

أهمية الحالة:

الحالة هي المسؤولة عن جعل تطبيقات React ديناميكية وتفاعلية. بدون الحالة، ستكون التطبيقات ثابتة وغير قادرة على التفاعل مع المستخدمين أو التكيف مع الأحداث الداخلية.
فعندما تتغير حالة أحد المكونات، يتولى React تحديث واجهة المستخدم تلقائيًا، مما يعني أن كل تغيير في الحالة يؤدي إلى إعادة عرض (re-render) الأجزاء المتعلقة بهذا التغيير من التطبيق.
على سبيل المثال، عند بناء واجهة تسجيل دخول، يتم تخزين المعلومات التي يدخلها المستخدم في الحقول (مثل البريد الإلكتروني وكلمة المرور) كجزء من الحالة، وعندما يتغير المحتوى، يتم تحديث الواجهة لإظهار هذه التغييرات.

كيفية التعامل مع الحالة في تطبيقاتReact

إدارة الحالة (State) في React هي عملية ضرورية لتطوير تطبيقات تفاعلية ديناميكية. تعني الحالة ببساطة البيانات التي يمكن أن تتغير بمرور الوقت، والتي يعتمد عليها التطبيق في عرض وتحديث واجهة المستخدم استجابة لتفاعلات المستخدم أو أحداث معينة. التعامل مع الحالة بشكل صحيح يضمن أن التطبيق يستجيب للتغيرات بسرعة وسلاسة.

الأساسيات النظرية:

فيReact ، كل مكون يمكن أن يكون له حالة خاصة به تُسمى **الحالة المحلية (Local State)**. تتم إدارة هذه الحالة باستخدام **Hooks** مثلuseState أو أدوات أخرى مثل useReducer. وفي بعض الأحيان، عندما يحتاج تطبيق كبير إلى مشاركة نفس البيانات بين عدة مكونات، نستخدم تقنيات مثل **Context API** أو مكتبات مثل **Redux** لإدارة **الحالة العامة (Global State)**.

كيف يتم التعامل مع الحالة؟

1. **تعريف الحالة:**
   يتم تعريف الحالة داخل مكون React باستخدام useState في المكونات الوظيفية (Functional Components). هذه الحالة هي متغير يُستخدم لتخزين البيانات التي ستتغير بمرور الوقت. يتم إعطاء هذا المتغير قيمة ابتدائية، ويتم توفير دالة لتحديث هذه القيمة لاحقًا عند الحاجة.

2. **تحديث الحالة:**
   يتم تحديث الحالة عن طريق استدعاء الدالة التي يتم إرجاعها من useState. كلما تغيرت الحالة، يقوم React تلقائيًا بإعادة عرض المكون لتحديث واجهة المستخدم بناءً على الحالة الجديدة. المهم هنا هو أن React يعتمد على مفهوم "التفاعلية(Reactivity) "، حيث يتم

تحديث العناصر على الشاشة تلقائيًا عند تغير الحالة، دون الحاجة إلى إجراء تحديثات يدوية.

3. **إعادة العرض:(Re-rendering)**
عندما يتم تحديث الحالة، يُعاد عرض المكون الذي يحتوي على هذه الحالة تلقائيًا، مما يعني أن أي عنصر في واجهة المستخدم يعتمد على الحالة سيتم تحديثه ليعكس القيم الجديدة. هذا الأمر هو ما يجعل React سريعًا وفعالًا في إدارة واجهات المستخدم الديناميكية.

مثال نظري:

فلنأخذ مثالاً بسيطاً على تطبيق صغير يحتوي على زر يزيد من قيمة عداد عند النقر عليه.

تخيل أنك تبني مكونًا بسيطًا لعرض عدد مرات النقر على زر:

- لديك مكون يسمى CounterComponent، وهو يعرض زرًا بالإضافة إلى عدد (count) يمثل عدد المرات التي تم فيها النقر على الزر.
- countهو جزء من الحالة داخل هذا المكون.
- كلما نقر المستخدم على الزر، يتم تحديث حالة countلزيادة الرقم بمقدار واحد.

```
function CounterComponent() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>لقد ضغطت على الزر {count} مرات</p>
            <button onClick={() => setCount(count + 1)}>
                اضغط لزيادة العدد
            </button>
        </div>
    );
}
```

شرح المثال:

1. **تعريف الحالة:**
في السطر الأول داخل المكون، نستخدم useStateلتعريف الحالة count، والتي تبدأ بقيمة 0. المتغير countيخزن قيمة العداد الحالية، ودالة setCountتُستخدم لتحديث هذه القيمة.

2. **عرض الحالة:**
في جزء JSX الذي يُستخدم لبناء واجهة المستخدم في(React ، نعرض عدد النقرات الحالية داخل عنصر <p>باستخدام {count}، وهو القيمة الحالية للحالة.

3. **تحديث الحالة:**
عندما ينقر المستخدم على الزر، يتم استدعاء .setCount(count + 1)هذه الدالة تقوم

بزيادة قيمة countبمقدار 1، وعندما يتغير count، يقوم React بإعادة عرض المكون تلقائيًا لتحديث واجهة المستخدم، بحيث يظهر العدد المحدث في الفقرة .<p>

الفائدة:

- **التفاعلية وسرعة الاستجابة** :بفضل إدارة الحالة، يصبح التطبيق قادرًا على التفاعل بسرعة مع المستخدم. عندما ينقر المستخدم على الزر، يتم تحديث العداد تلقائيًا دون الحاجة إلى إعادة تحميل الصفحة أو القيام بأي تحديثات يدوية.
- **فصل المنطق عن العرض** :يسمح React بفصل كيفية إدارة البيانات (الحالة) عن كيفية عرضها. يمكن أن يكون لديك العديد من العمليات التي تحدث في الخلفية، مثل جلب البيانات من خادم أو التعامل مع تفاعلات المستخدم، وكل ذلك يتم بفضل آلية إدارة الحالة.
- **إعادة العرض التلقائي** :أهم فوائد React هو ميكانيكية إعادة العرض التلقائي للمكونات عند تغير الحالة. بمجرد أن تتغير الحالة، يتم تحديث واجهة المستخدم بشكل تلقائي لتتماشى مع هذه التغييرات، مما يوفر الكثير من الجهد على المطور.

الفرق بين الحالة المحلية (Local State) والعامة(Global State)

فيReact ، تعتبر **الحالة (State)** واحدة من الركائز الأساسية لبناء تطبيقات تفاعلية ديناميكية. ومع توسع التطبيقات وكثرة مكوناتها، تظهر الحاجة إلى التعامل مع أنواع مختلفة من الحالات. تنقسم الحالة إلى نوعين رئيسيين :**الحالة المحلية (Local State)** و**الحالة العامة** .**(Global State)** لكل منهما استخداماته الخاصة، وطرق إدارة مختلفة، وذلك اعتمادًا على احتياجات التطبيق.

الحالة المحلية(Local State)

الحالة المحلية هي البيانات التي يتم تخزينها واستخدامها داخل مكون واحد فقط. هذه الحالة تُعرف داخل المكون ولا يتم مشاركتها مع أي مكونات أخرى. غالبًا ما تُستخدم الحالة المحلية في المكونات الصغيرة التي تتطلب تفاعلاً داخليًا بسيطًا، مثل التحكم في مدخلات نموذج (Form) أو إظهار/إخفاء مكون بناءً على حدث ما.

خصائص الحالة المحلية:

1. **محصورة داخل المكون** :الحالة المحلية تكون خاصة بالمكون ولا يمكن الوصول إليها من مكونات أخرى. لا يمكن لمكون آخر تعديل أو الوصول إلى الحالة المحلية لمكون معين.
2. **سهولة الإدارة** :بما أن الحالة تقتصر على المكون، يمكن إدارتها بسهولة باستخدام useStateأو useReducer.
3. **غير معقدة** :تُستخدم الحالة المحلية عادةً لإدارة بيانات بسيطة، مثل القيم المدخلة في الحقول، أو التحكم في الأزرار أو واجهات صغيرة.
4. **عمرها الافتراضي قصير** :الحالة المحلية غالبًا ما تكون مؤقتة وتُستخدم لتلبية تفاعلات المستخدم اللحظية، وتنتهي بانتهاء المكون أو تحديثه.

مثال على الحالة المحلية:

فلنفترض أن لدينا نموذج إدخال (Form) بسيط يطلب من المستخدم إدخال اسمه. نريد تخزين الاسم المدخل داخل المكون نفسه فقط، لذلك نستخدم الحالة المحلية.

```
// NameForm.js
import React, { useState } from 'react';

export function NameForm() {

    const [name, setName] = useState('');

    return (
        <div>
            <input
                type="text"
                value={name}
                onChange={(e) => setName(e.target.value)}
            />
            <p>اسمك هو: {name}</p>
        </div>
    );
}
```

شرح المثال:

- **تعريف الحالة**: استخدمنا useStateلتعريف حالة محلية name، التي تبدأ بقيمة فارغة.
- **تحديث الحالة**: كلما أدخل المستخدم قيمة جديدة في حقل النص، يتم تحديث nameعبر setName.
- **عرض الحالة**: يعرض <p>الاسم المدخل مباشرةً باستخدام {name}.

الحالة العامة(Global State)

الحالة العامة هي البيانات التي تُشرك بين عدة مكونات ضمن التطبيق. تُستخدم الحالة العامة عندما تكون هناك حاجة للوصول إلى نفس البيانات في أكثر من مكون، مثل معلومات المستخدم المسجل دخوله.

مثال على الحالة العامة:

سنستخدم **Context API**لتوفير الحالة العامة عبر التطبيق.

شرح المثال:

- **تعريف الحالة العامة**: في UserContext.js، نستخدم createContextلإنشاء Context يتيح لنا توفير حالة عامة عبر التطبيق UserProvider. يقوم بتوفير الحالة العامة للمكونات الفرعية.
- **استخدام الحالة العامة**: في Profile.jsو Navigation.js, نستخدم useContextللوصول إلى الحالة العامة UserContextوعرض بيانات المستخدم.

- **توفير الحالة** :في App.js, نستخدم UserProviderلتغليف المكونات التي تحتاج إلى الوصول إلى الحالة العامة.

الفائدة من كل نوع من الحالة:

- **الحالة المحلية:(Local State)**
تُستخدم لإدارة البيانات التي تهم مكونًا معينًا فقط، مثل إدخال نموذج أو حالة زر. توفر طريقة بسيطة وفعالة لإدارة البيانات المؤقتة وتحديث واجهة المستخدم استجابةً لتفاعلات المستخدم.

- **الحالة العامة:(Global State)**
تُستخدم عندما تحتاج إلى مشاركة البيانات بين عدة مكونات، مما يسهل الحفاظ على التناسق وتنسيق البيانات في جميع أنحاء التطبيق. تعتبر ضرورية لتطبيقات كبيرة ومعقدة تحتاج إلى مشاركة بيانات أساسية مثل حالة المستخدم أو إعدادات التطبيق بين مكونات متعددة.

تحديات إدارة الحالة في تطبيقات الويب:

إدارة الحالة في تطبيقات الويب تعد أحد الجوانب الأساسية والحرجة في تطوير البرمجيات، خصوصاً عندما يتزايد حجم وتعقيد التطبيقات. تتناول هذه الفقرة أبرز التحديات التي يواجهها المطورون في إدارة الحالة، وكيف يمكن أن تؤثر هذه التحديات على الأداء وتجربة المستخدم.

التعقيدات المتعلقة بإدارة الحالة في التطبيقات الكبيرة

في التطبيقات الكبيرة والمعقدة، تصبح إدارة الحالة أكثر صعوبة بسبب عدة عوامل:

- **تعدد المكونات والتفاعلات**:
في التطبيقات الكبيرة، هناك العديد من المكونات التي تحتاج إلى التفاعل مع بعضها البعض وتبادل البيانات. قد يؤدي تعدد المكونات إلى تعقيد عملية إدارة الحالة، حيث قد تحتاج البيانات إلى التمرير عبر عدة مستويات من المكونات، مما يخلق صعوبة في تتبع كيفية وصول البيانات وتحديثها.

- **تغيير الحالة من أماكن متعددة**:
عندما تتطلب الحالة أن تُعدل من عدة مكونات أو مصادر، يصبح من الصعب ضمان أن جميع التغييرات تتماشى بشكل صحيح. قد تؤدي التعديلات غير المنسقة إلى أخطاء في الواجهة، مثل تحديثات غير متوقعة أو تعارضات بين حالة المكونات المختلفة.

- **إدارة الحالات المعقدة**:
في حالة وجود حالات معقدة تعتمد على شروط متعددة أو تتطلب معالجة بيانات متداخلة، يصبح من الصعب تتبع التغيرات وحالة التطبيق بالكامل. إدارة مثل هذه الحالات تتطلب أدوات متقدمة واستراتيجيات أفضل لضمان أن جميع التغيرات تتم بشكل صحيح ومنسق.

كيف يمكن أن يؤدي سوء إدارة الحالة إلى مشاكل في الأداء وتجربة المستخدم

سوء إدارة الحالة يمكن أن يؤثر بشكل كبير على أداء التطبيق وتجربة المستخدم:

- **إعادة العرض الزائد:(Excessive Re-renders)**
عندما لا تتم إدارة الحالة بشكل صحيح، قد يؤدي ذلك إلى إعادة عرض غير ضرورية للمكونات. إعادة العرض الزائد تستنزف موارد النظام وتؤدي إلى تجربة مستخدم غير سلسة، حيث تصبح التفاعلات بطيئة وغير استجابة.

- **بطء في التفاعل:(Interaction Lag)**
سوء إدارة الحالة يمكن أن يؤدي إلى تأخير في استجابة التطبيق لتفاعلات المستخدم. على سبيل المثال، إذا كان هناك تأخير في تحديث البيانات بسبب إدارة غير فعالة للحالة، قد يشعر المستخدم بأن التطبيق غير مستجيب أو بطيء.

- **زيادة تعقيد الكود:(Code Complexity)**
إدارة الحالة بطريقة غير منظمة يمكن أن تؤدي إلى تعقيد الكود. الكود المعقد يصعب صيانته وتعديله، مما يؤدي إلى زيادة احتمالية الأخطاء وصعوبة اكتشافها وإصلاحها.

- **مشاكل في التزامن:(Concurrency Issues)**
في التطبيقات التي تتطلب معالجة متعددة أو تفاعلات متزامنة، قد يؤدي سوء إدارة الحالة إلى مشاكل في التزامن، حيث يمكن أن تحدث تعارضات بين عمليات مختلفة تؤدي إلى نتائج غير متوقعة.

الحاجة إلى أدوات وإستراتيجيات فعالة لإدارة الحالة

لتجاوز التحديات المذكورة وتحقيق إدارة فعالة للحالة، تحتاج التطبيقات إلى استخدام أدوات وإستراتيجيات فعالة:

- **استخدام مكتبات إدارة الحالة:**
الأدوات مثل **Redux**، **MobX**، و **Zustand**توفر آليات قوية لإدارة الحالة العامة والتعامل مع التعقيدات المرتبطة بها. توفر هذه المكتبات طرقًا لتنظيم البيانات بطريقة تجعل من السهل التعامل مع التحديثات ومزامنة الحالة بين مكونات متعددة.

- **تقنيات التحديث الفعال:**
تقنيات مثل (**Memoization**استخدام React.memo وuseMemo وuseCallback) يمكن أن تساعد في تحسين الأداء عن طريق تقليل عمليات إعادة العرض غير الضرورية. هذه التقنيات تضمن أن المكونات تُعاد عرضها فقط عندما تتغير بياناتها بشكل حقيقي.

- **استخدام Context API بحذر:**
Context APIمفيدة لمشاركة الحالة عبر مكونات متعددة، ولكن يجب استخدامها بحذر لتجنب إعادة العرض الزائد. من الأفضل استخدامها للحالة التي تكون ثابتة نسبياً ولا تتغير بشكل متكرر.

- **استراتيجيات تقسيم الحالة:**
تقسيم الحالة إلى أجزاء أصغر، وإدارة كل جزء بشكل مستقل، يمكن أن يساعد في تقليل التعقيد وتسهيل صيانة الكود. يمكنك استخدام **Custom Hooks**لتنظيم الحالة بشكل منطقي وتحسين قراءة الكود.

الفصل الثالث: إدارة الحالة من خلال useState

مفهوم useStateوكيفية استخدامه

useStateهو واحد من أهم الـ **Hooks**فيReact ، ويستخدم لإدارة الحالة داخل **المكونات الوظيفية** (Functional Components). الحالة (State) هي البيانات التي تتحكم في كيفية عرض المكون وتفاعله مع المستخدم. على سبيل المثال، إذا كنت تبني تطبيق عداد، فإن قيمة العداد التي تظهر على الشاشة تمثل حالة متغيرة بناءً على تفاعل المستخدم.

عند استخدام useState، يمكنك إنشاء حالة خاصة بالمكون وتحديثها لاحقًا عند الحاجة. في الماضي، كان يمكن إدارة الحالة فقط في **المكونات القائمة على الفئات** (Class Components) باستخدام this.setState، ولكن مع ظهور **Hooks**في React 16.8 ، أصبح بالإمكان استخدام الحالة في المكونات الوظيفية أيضًا.

كيفية استخدام useState

يتم استخدام useStateداخل المكون الوظيفي لتعريف حالة محلية. يستقبل useStateقيمة ابتدائية كمعامل ويعيد مصفوفة تحتوي على العنصرين التاليين:

1. **الحالة الحالية** :تمثل القيمة الحالية للحالة.
2. **دالة لتحديث الحالة** :تستخدم لتحديث الحالة عند الحاجة.

أهمية useState:

useState يجعل من السهل التعامل مع الحالة في المكونات الوظيفية، ويوفر وسيلة فعالة لإدارة البيانات التي تتغير مع التفاعل. يمكن استخدامه لإنشاء حالة لأي جزء من واجهة المستخدم، مثل إدخال النصوص، عدادات، أو حتى التحكم في حالة العرض/الإخفاء.

كيفية استخدام useStateلإنشاء الحالة المحلية

useStateهو Hook في React يُستخدم لإنشاء **الحالة المحلية** في المكونات الوظيفية. هذا الـ Hookيسمح لك بإدارة البيانات التي تتغير مع التفاعل داخل المكون، وهو بديل بسيط وفعال للطريقة القديمة التي كانت تعتمد على المكونات القائمة على الفئات.(Class Components)

لإنشاء الحالة المحلية باستخدام useState، تقوم بتعريف الحالة والدالة التي ستقوم بتحديثها. يتم استدعاء useStateداخل المكون الوظيفي، ويُعيد مصفوفة تحتوي على عنصرين:

1. **الحالة الحالية** :(Current State)تمثل القيمة الفعلية التي ترغب في تتبعها داخل المكون.
2. **دالة لتحديث الحالة** :(Setter Function)هذه الدالة يتم استخدامها لتغيير قيمة الحالة وإعادة تحديث (re-render) المكون.

ملاحظات:

- عندما تقوم بتعريف حالة محلية باستخدام useState، يتم تمرير **القيمة الأولية** كمعامل. هذه القيمة تمثل الحالة الأولى عند تحميل المكون لأول مرة. القيمة الأولية يمكن أن تكون ثابتة، أو يمكن أن تكون نتيجة عملية حسابية أو دالة، حسب احتياجات التطبيق.
- عندما تقوم بتحديث الحالة في React باستخدام دالة تحديث الحالة التي تم تعريفها مع useState(مثل setCount، يتسبب ذلك في **إعادة تحديث المكون** .عملية إعادة التحديث

تحدث تلقائيًا عند تغيير الحالة لأن React يعتمد على مفهوم "التزامن الافتراضي (Virtual DOM)" لتحديث واجهة المستخدم بشكل فعال وديناميكي.

مثال تطبيقي: إنشاء تطبيق إدارة المهام

في هذه المسألة، ستقوم بإنشاء تطبيق بسيط لإدارة المهام (To-Do List) باستخدام React و Hook useState. يهدف هذا التمرين إلى تعزيز فهمك لكيفية استخدام useState لإدارة الحالة داخل مكونات React ، وكيفية بناء واجهات تفاعلية.

متطلبات المسألة:

1. **إعداد بيئة العمل:**
   - تأكد من أنك قد قمت بإعداد بيئة تطوير React. يمكنك استخدام Create React App لإنشاء مشروع جديد.
   - قم بإنشاء الملفات والمكونات التالية: App.js, Task.js, TaskInput.js.
2. **متطلبات التطبيق:**
   - **مكون App**: يجب أن يكون المكون الرئيسي للتطبيق ويحتوي على الحالة العامة لتخزين قائمة المهام. يجب أن يكون قادرًا على إضافة مهام جديدة وإزالة المهام من القائمة.
   - **مكون Task**: يجب أن يعرض كل مهمة في قائمة ويحتوي على زر لإزالة المهمة.
   - **مكون TaskInput**: يجب أن يوفر حقل نصي لإدخال مهام جديدة ويقوم بإرسال البيانات إلى App لإضافة المهمة إلى القائمة.

خطوات تنفيذ المسألة:

1. **إنشاء مكون App:**
   - قم بإنشاء مكون App الذي يستخدم useState لإدارة حالة قائمة المهام.(tasks)
   - أضف دالة addTask لإضافة مهام جديدة إلى القائمة.
   - أضف دالة removeTask لإزالة مهام من القائمة.
   - قم بتمرير دالة addTask إلى مكون TaskInput ومرر قائمة المهام إلى مكون Task.

App.js:

```
import logo from './logo.svg';
import './App.css';
import Button from './session one/components/propsAsFunction';
import Greeting from './session one/components/props';
import CounterComponent from "./sessionTow/CounterComponent"
import NameForm from "./sessionTow/NameForm"
import Task from './sessionTow/Task';
import TaskInput from "./sessionTow/TaskInput"
import React, { useState } from 'react';
```

```
function App() {
  // تعريف الحالة لتخزين قائمة المهام
  const [tasks, setTasks] = useState([]);

  // دالة لإضافة مهمة جديدة
  const addTask = (task) => {
    setTasks([...tasks, task]);
  };

  // دالة لإزالة مهمة من القائمة
  const removeTask = (index) => {
    const newTasks = tasks.filter((_, i) => i !== index);
    setTasks(newTasks);
  };

  return (
    <div className="App">
      <h1>تطبيق إدارة المهام</h1>
      <TaskInput addTask={addTask} />
      <ul>
        {tasks.map((task, index) => (
          <Task key={index} task={task} onRemove={() =>
removeTask(index)} />
        ))}
      </ul>

    </div>
  );
}

export default App;
```

TaskInput.js:

```
import React, { useState } from 'react';

export default function TaskInput({ addTask }) {


    const [inputValue, setInputValue] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        if (inputValue.trim()) {
            addTask(inputValue);
            setInputValue('');
        }
    };
```

```
    return (
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                value={inputValue}
                onChange={(e) => setInputValue(e.target.value)}
                placeholder="أدخل مهمة جديدة"
            />
            <button type="submit">أضف مهمة</button>
        </form>
    );
}
```

Task.js:

```
import React from 'react';

export default function Task({ task, onRemove }) {
    return (
        <li>
            {task}
            <button onClick={onRemove}>إزالة</button>
        </li>
    );
}
```

شرح المثال:

1. **تعريف الحالة في App.js:**
   - نستخدم useStateلتعريف الحالة tasksلتخزين قائمة المهام.
   - نقوم بإنشاء دالة addTaskلإضافة مهام جديدة إلى القائمة، ودالة removeTaskلإزالة مهام من القائمة.
2. **مكون Task:**
   - يعرض مهمة واحدة ويحتوي على زر لإزالة المهمة من القائمة. عند النقر على الزر، يتم استدعاء دالة onRemoveالمرسلة من المكون الرئيسي.
3. **مكون TaskInput:**
   - يوفر حقل نصي لإدخال المهام وزر لإضافتها. يتم تحديث الحالة المحلية inputValueبناءً على إدخال المستخدم، وعند تقديم النموذج، يتم إرسال المهمة إلى المكون الرئيسي باستخدام دالة addTask.

فوائد استخدام useState في هذا المثال:

- **إدارة الحالة البسيطة**:
  باستخدام useState، نتمكن من إدارة حالة قائمة المهام وتحديثها بسهولة، مما يجعل واجهة المستخدم تتفاعل بشكل تلقائي مع التغييرات.

- **فصل المنطق**:
  المثال يوضح كيفية فصل منطق إدخال المهام (في TaskInput) وعرض المهام (في Task)، مما يجعل الكود أكثر تنظيماً وقابلاً للصيانة.

- **التفاعل الفوري**:
  عندما يضيف المستخدم مهمة جديدة أو يزيل واحدة، يتم تحديث واجهة المستخدم مباشرةً بفضل التحديثات التلقائية التي يوفرها useState.

الفصل الرابع: تنسيق المشروع واعداداته الأساسية

تخصيص إعدادات المشروع الأساسية:

تغيير أيقونة المشروع(favicon)

الأيقونة (favicon) هي تلك الصورة الصغيرة التي تظهر بجانب عنوان صفحة الويب في المتصفح. فيReact ، يمكن تعديل هذه الأيقونة بسهولة لاستبدالها بأيقونة مخصصة تناسب مشروعك.

كيفية تحديث الأيقونة في:React

1. **إضافة الأيقونة الجديدة :**قم بإعداد أيقونة جديدة بصيغة .ico أو .png وتأكد أن حجمها مناسب) عادة 32 32x بكسل أو 16 16x بكسل.(
2. **استبدال الأيقونة الافتراضية :**في مشروع React الذي تم إنشاؤه بواسطة Create React App، ستجد الأيقونة الافتراضية في مجلد public تحت اسم favicon.ico. قم باستبدال هذا الملف بالأيقونة الجديدة مع الحفاظ على الاسم كما هو.
3. **تحديث الأيقونة في المتصفح :**بمجرد استبدال الملف، تأكد من إعادة تشغيل المشروع وتنظيف الكاش (Cache) في المتصفح لرؤية الأيقونة الجديدة.

تحديث اسم المشروع

يعد اسم المشروع الذي يظهر في علامة تبويب المتصفح (title) جزءًا مهمًا من تجربة المستخدم ويجب أن يعكس بشكل دقيق محتوى المشروع.

كيفية تعديل اسم المشروع في المتصفح:

1. **فتح ملف :public/index.html**فيReact ، يتم التحكم في عنوان الصفحة الافتراضي عبر ملف index.html الموجود في مجلد public.
2. **تعديل الوسم :<title>**ابحث عن الوسم <title> داخل ملف index.html، وقم بتحديث النص الموجود بداخله ليكون اسم المشروع الخاص بك.

استيراد الصور والخطوط المخصصة

كيفية استيراد الصور في مشروعReact

في مشاريعReact ، يمكنك استيراد الصور بسهولة وإدراجها في مكونات الـ .JSX يعمل React بشكل متوافق مع حزم البناء) مثل (Webpack التي تدعم تضمين الصور في المشروع وجعلها جزءًا من الـ bundle النهائي.

كيفية استيراد الصور وتضمينها في:React

1. **إضافة الصور إلى مجلد المشروع:**
   - قم بحفظ الصور التي ترغب في استخدامها في مجلد المشروع، ويفضل في مجلد /src/assets/images أو /public للحفاظ على التنظيم.
2. **استيراد الصور في ملف:JSX**
   - يمكنك استيراد الصور باستخدام الطريقة التقليدية لاستيراد الملفات في React عبر import. مثلًا، إذا كان لديك صورة في مجلد src، يمكنك استيرادها كالآتي:

إعداد Tailwind CSS في مشروع React

ما هو Tailwind CSS ؟

تعريف Tailwind CSS وكيف يعمل ك CSS Framework: Tailwind CSS هو إطار عمل CSS يساعد المطورين على تصميم واجهات المستخدم بشكل أسرع وأكثر فعالية. يميز Tailwind عن غيره من أطر عمل CSS هو أنه يقدم مجموعة من الفئات (classes) التي يمكن استخدامها مباشرة في HTMLأو JSX لتطبيق أنماط معينة بدلاً من كتابة CSS مخصص لكل عنصر.

- **فئات جاهزة Tailwind :**يتيح لك استخدام فئات مثل bg-blue-500، text-center، و p-4 لتحديد الألوان، المحاذاة، والحشو، دون الحاجة لكتابة قواعد CSS مخصصة.
- **مرونة عالية :**بفضل نظام الفئات القابل للتخصيص، يمكنك تعديل تصميماتك بسرعة دون الحاجة للغوص في ملفات CSS الكبيرة.

الفرق بين Tailwind و CSS التقليدي:

- **CSS التقليدي :**يتطلب كتابة قواعد CSS مخصصة لكل عنصر أو مكون. يمكن أن يؤدي ذلك إلى تكرار الكود وعدم الكفاءة.
- **Tailwind CSS:**يعتمد على استخدام فئات جاهزة، مما يسهل التعديلات السريعة ويقلل من تكرار الكود. يوفر أيضًا نظامًا مرنًا للتخصيص بحيث يمكنك تغيير إعدادات الألوان، الخطوط، والمسافات عبر ملف التكوين.

تثبيت Tailwind في مشروع React

كيفية إعداد Tailwind في مشروع React جديد باستخدام:Create React App

1. **إنشاء مشروع React جديد:**
   - قم بإنشاء مشروع جديد باستخدام Create React Appعبر تنفيذ الأمر التالي في terminal:

```
npx create-react-app my-app
```

2. **تثبيت:Tailwind CSS**

- انتقل إلى مجلد المشروع:
- قم بتثبيت Tailwind CSS باستخدام npm أو:yarn

```
npm install tailwindcss postcss autoprefixer
```

3. **إعداد ملفات التكوين:**

- قم بإنشاء ملفات التكوين لـ Tailwind و PostCSSعبر تنفيذ الأمر:

```
npx tailwindcss init -p
```

.1

   - سيؤدي ذلك إلى إنشاء ملف tailwind.config.jsوملف postcss.config.jsفي مجلد المشروع.

تكوين Tailwind في مشروعReact

تخصيص الألوان، الخطوط، والمسافات داخل ملف :tailwind.config.js

1. **تخصيص الألوان:**

- يمكنك إضافة ألوان مخصصة ضمن قسم extendفي ملف :tailwind.config.js

```
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        customBlue: '#1DA1F2',
        customGray: '#F5F5F5',
      },
    },
  },
  plugins: [],
}
```

تخصيص الخطوط:

- يمكنك إضافة خطوط جديدة من خلال توسيع إعدادات الخطوط:

```
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        customBlue: '#1DA1F2',
        customGray: '#F5F5F5',
      },
      fontFamily: {
        sans: ['Inter', 'Arial', 'sans-serif'],
        serif: ['Merriweather', 'serif'],
      },
    },
  },
  plugins: [],
}
```

ضبط النُسق المخصصة (Themes) وتوسيعها:

- يمكن استخدام ملف tailwind.config.jsلضبط نسق مخصص يتناسب مع تصميم مشروعك. يمكنك تحديد الألوان والخطوط والأساليب التي تناسب تصميمك بشكل خاص.

تمرين React شامل باستخدام Tailwind CSSوprops, useState

إنشاء مكون بطاقة (Card) تفاعلية

في هذا التمرين، ستقوم بإنشاء تطبيق React يعرض بطاقة معلومات تفاعلية، تحتوي على:

1. **صورة** شخصية.
2. **اسم** الشخص.
3. **وصف** قصير.
4. زر **"إظهار المزيد "**الذي يعرض معلومات إضافية عند النقر عليه.

المتطلبات:

1. استخدم **props**لتمرير البيانات الخاصة بالاسم والوصف والصورة.
2. استخدم **useState**لإدارة حالة زر "إظهار المزيد" (عرض الوصف الإضافي).
3. استخدم **Tailwind CSS**لتنسيق البطاقة وجعلها جميلة.

الخطوات:

1. قم بإنشاء مكون React يُدعى ProfileCard.
2. استخدم propsلتمرير **اسم**، **وصف قصير**، **وصف إضافي**، و**صورة**.
3. استخدم useStateللتحكم في عرض الوصف الإضافي عند الضغط على زر "إظهار المزيد."
4. استخدم Tailwind CSS لتنسيق المكون بالشكل المناسب، بحيث يظهر الاسم والوصف القصير بشكل مميز، ويظهر الوصف الإضافي عند النقر على الزر.

الهيكل المطلوب:

- الصورة في أعلى البطاقة.
- اسم الشخص تحت الصورة.
- الوصف القصير أسفل الاسم.
- زر أسفل البطاقة يعرض "إظهار المزيد" وعند الضغط عليه يظهر الوصف الإضافي.

ProfileCard.js:

```
import React, { useState } from "react";
const ProfileCard = ({ image, name, shortDescription, longDescription
}) => {
```

```
  const [showMore, setShowMore] = useState(false);

  const toggleShowMore = () => {
    setShowMore(!showMore);
  };

  return (
    <div className="max-w-sm rounded-xl overflow-hidden shadow-lg bg-
white p-6 transform hover:scale-105 transition-transform duration-500
ease-in-out">
      {/* صورة الشخصية */}
      <img
        className="w-full h-56 object-cover rounded-t-lg"
        src={image}
        alt={`${name}'s avatar`}
      />

      {/* اسم الشخص */}
      <div className="font-bold text-2xl mt-4 text-gray-
800">{name}</div>

      {/* الوصف القصير */}
      <p className="text-gray-600 text-lg mt-2">{shortDescription}</p>

      {/* زر إظهار المزيد */}
      <button
        onClick={toggleShowMore}
        className="bg-gradient-to-r from-purple-400 to-indigo-500
hover:from-purple-600 hover:to-indigo-700 text-white font-bold py-2 px-
4 rounded mt-6 focus:outline-none focus:ring-2 focus:ring-purple-300"
      >
        {showMore ? "إخفاء" : "إظهار المزيد"}
      </button>

      {/* عرض الوصف الإضافي بناءً على حالة showMore */}
      {showMore && (
        <p className="text-gray-700 text-lg mt-4 transition-all
duration-300 ease-in-out">
          {longDescription}
        </p>
      )}
    </div>
  );
};

export default ProfileCard;
```

الفصل الخامس:
إدارة الحالات الجانبية بواسطة useEffect

مفهوم useEffect وكيفية استخدامه

استخدام useEffect للتعامل مع الآثار الجانبية في React

عندما نعمل في React ، لا تقتصر مهمة المكونات على التصيير (render) فقط، بل نحتاج أحيانًا إلى القيام ببعض العمليات خارجية أو التعامل مع بيانات من مصادر خارجية. هذه العمليات تُسمى **الآثار الجانبية (Side Effects)**. على سبيل المثال، تخيل أنك تحتاج إلى جلب بيانات من API ، أو الاشتراك في خدمة ما، أو حتى تعديل عنوان الصفحة بناءً على حالة معينة في التطبيق. هنا يأتي دور useEffect، وهو Hook في React مصمم للتعامل مع هذه العمليات.

ما هي الآثار الجانبية؟

الآثار الجانبية في React هي كل شيء يحدث **خارج** وظيفة التصيير الرئيسية للمكون. فبدلًا من الاكتفاء برسم الـ UI على الشاشة، قد نحتاج إلى إجراء بعض التغييرات أو تنفيذ عمليات إضافية. بعض الأمثلة الشائعة تشمل:

- **جلب البيانات**: مثل إجراء طلب HTTP للحصول على بيانات من خادم.
- **التعامل مع الـ DOM**: كإضافة أو إزالة مستمعين للأحداث.
- **تعديل العنوان أو الوصف**: تغيير عنوان الصفحة بناءً على محتويات المكون.

تركيب useEffect وكيفية عمله

```
useEffect(() => {
  // الوظيفة التي تحتوي على الأثر الجانبي
  return () => {
    // تنظيف الأثر الجانبي إذا كان ذلك مطلوبًا
  };
}, [التبعيات]);
```

الوظيفة الأولى: يتم تنفيذها عند كل تحديث للمكون.

الوظيفة الثانية (التنظيف): يتم استدعاؤها لإزالة أو تنظيف الأثر الجانبي السابق، إذا كان مطلوبًا (مثل إزالة المؤقتات أو الاشتراكات).

التبعيات (Dependencies): تحدد متى يجب استدعاء useEffect بناءً على التغييرات في القيم المحددة.

استخدام useEffect للتعامل مع الآثار الجانبية

useEffect هو Hook يستخدم في React لإخبار المكون بأن هناك تأثيرًا جانبيًا يجب تنفيذه بعد أن يتم تصيير المكون، أو في كل مرة يتغير شيء معين داخل المكون. الوظيفة الأساسية لـ useEffect تشبه الوظائف **المعتمدة على الأحداث**؛ أي أنها تُنفّذ عند حدوث شيء ما.

مثال بسيط : فلنبدأ بمثال يوضح كيفية استخدام useEffect لجلب بيانات من API عند تحميل المكون لأول مرة.

```
import React, { useState, useEffect } from 'react';

const UserList = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    // جلب بيانات المستخدمين من API
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setUsers(data))
      .catch(error => console.error(error));
  }, []); // تمرير مصفوفة فارغة لتشغيل useEffect مرة واحدة فقط بعد تحميل المكون

  return (
    <div>
      <h1>List of Users</h1>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
};

export default UserList;
```

في هذا المثال:

- نستخدم useEffect لجلب البيانات من API بعد تحميل المكون.
- fetch هو طريقة جلب البيانات.
- المصفوفة الفارغة [] في نهاية useEffect تعني أن الكود داخل useEffect سيعمل مرة واحدة فقط بعد أول تصيير للمكون.

كيف يعمل useEffect؟

useEffectيأخذ دالتين أساسيتين:

1. **دالة التنفيذ**:تحتوي على الكود الذي ترغب بتنفيذه (مثل جلب البيانات).
2. **مصفوفة التبعيات (Dependencies Array)**: وهي تحدد متى يجب استدعاء useEffect.إذا كانت فارغة، سيتم استدعاء useEffectمرة واحدة عند أول تصيير للمكون. إذا كانت تحتوي على متغيرات، سيتم استدعاء useEffectكلما تغيّرت تلك المتغيرات.

```
useEffect(() => {
  // هذا الكود يتم تنفيذه عند التصيير الأول
  // أو عند تغيير أحد المتغيرات في مصفوفة التبعيات
}, [dependency1, dependency2]);
```

متى يتم استخدام useEffect؟

تستخدم useEffectعندما تحتاج إلى:

- **جلب بيانات**:إذا كنت تريد جلب بيانات من API بعد تحميل الصفحة أو المكون.
- **التفاعل مع الـ DOM**: مثل إضافة مستمعين للأحداث (event listeners) أو تعديل عنصر في الصفحة.
- **تنفيذ أكواد خارجية**:مثل استخدام مكتبات خارجية أو أدوات تقوم بالتفاعل مع المكون.

التحكم في التبعيات (Dependencies) في useEffect

useEffectيعتمد على **مصفوفة التبعيات** (Dependencies Array)لتحديد متى يتم تنفيذ الأثر الجانبي. هناك ثلاثة سيناريوهات أساسية:

1. **بدون تبعيات**:يتم استدعاء useEffectبعد كل مرة يتم فيها إعادة تصيير المكون، مما قد يؤدي إلى تكرار تنفيذ الأثر الجانبي في كل تحديث، وهو ما يمكن أن يكون غير فعال.

```
useEffect(() => {
  // يتم تنفيذه بعد كل تصيير
});
```

2. **بالتبعيات**:يمكنك تمرير قيم معينة (متغيرات) في مصفوفة التبعيات. في هذه الحالة، سيتم تنفيذ useEffectفقط عندما تتغير أي من هذه القيم.

```
useEffect(() => {
  // يتم تنفيذه فقط عندما تتغير التبعيات
}, [dependency]);
```

3. **مصفوفة فارغة**:إذا قمت بتمرير مصفوفة فارغة []، سيتم استدعاء useEffectمرة واحدة فقط بعد أول تصيير للمكون، ولن يتم استدعاؤه مرة أخرى حتى يتم إزالة المكون من الصفحة.

```
4. useEffect(() => {
```

```
  // يتم تنفيذه مرة واحدة فقط
}, []);
```

الآثار الجانبية والتنظيف(Cleanup)

في بعض الأحيان، تحتاج إلى إزالة أو "تنظيف" الآثار الجانبية التي قمت بإنشائها، مثل إزالة مستمع للأحداث أو إيقاف مؤقت .(timer) يمكن استخدام دالة التنظيف داخل useEffectلهذا الغرض.

```
useEffect(() => {
  const timer = setInterval(() => {
    console.log('This runs every second!');
  }, 1000);

  // دالة التنظيف
  return () => {
    clearInterval(timer); // إيقاف المؤقت عند إزالة المكون
  };
}, []);
```

تجنب الدورات اللانهائية(Infinite Loops)

إذا كان هناك استخدام خاطئ لـ useEffectيمكن أن يؤدي إلى **دورات لانهائية** .يحدث هذا عادةً عندما يتم تعديل أحد المتغيرات في كل مرة يتم فيها تصيير المكون، مما يؤدي إلى إعادة تنفيذ useEffectمرارًا وتكرارًا.

```
useEffect(() => {
  setCount(count + 1); // يؤدي إلى تغيير count في كل تصيير
}, [count]);
```

تمرين: تطبيق قائمة المهام(To-Do List)

المتطلبات:

1. السماح للمستخدم بإضافة مهام جديدة.
2. جلب المهام المبدئية عند تحميل الصفحة من API وهمي.
3. عرض قائمة المهام في مكون منفصل.
4. تصميم عصري باستخدام.Tailwind CSS

App.js:

```
import React, { useState, useEffect } from "react";
import TaskList from "./TaskList";
import TaskForm from "./TaskForm";

const App = () => {
  const [tasks, setTasks] = useState([]);

  // جلب المهام من API وهمي
```

```
  useEffect(() => {
    const fetchTasks = async () => {
      const response = await fetch(
        "https://jsonplaceholder.typicode.com/todos?_limit=5"
      );
      const data = await response.json();
      setTasks(data);
    };
    fetchTasks();
  }, []);

  // إضافة مهمة جديدة
  const addTask = (newTask) => {
    const newTaskObj = {
      id: tasks.length + 1,
      title: newTask,
      completed: false,
    };
    setTasks([...tasks, newTaskObj]);
  };

  return (
    <div className="min-h-screen bg-gradient-to-r from-blue-500 to-
purple-600 p-8 flex flex-col items-center">
      <h1 className="text-3xl font-extrabold text-white mb-6">قائمة
المهام</h1>
      <TaskForm addTask={addTask} />
      <TaskList tasks={tasks} />
    </div>
  );
};

export default App;
```

TaskList.js:

```
import React from "react";

const TaskList = ({ tasks }) => {
  return (
    <div className="bg-white p-6 rounded-lg shadow-lg max-w-lg w-full">
      <h2 className="text-xl font-bold text-gray-700 mb-4">المهام الحالية</h2>
      <ul className="space-y-3">
        {tasks.map((task) => (
          <li
            key={task.id}
            className="p-3 bg-gray-100 rounded-md shadow-sm text-lg
font-medium"
```

```
          >
            {task.title}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TaskList;
```

TaskForm.js:

```
import React, { useState } from "react";

const TaskForm = ({ addTask }) => {
  const [newTask, setNewTask] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (newTask.trim() !== "") {
      addTask(newTask);
      setNewTask("");
    }
  };

  return (
    <form
      onSubmit={handleSubmit}
      className="bg-white p-6 rounded-lg shadow-md mb-6 max-w-lg w-
full"
    >
      <input
        type="text"
        className="border p-3 w-full mb-4 rounded-md"
        placeholder="أضف مهمة جديدة"
        value={newTask}
        onChange={(e) => setNewTask(e.target.value)}
      />
      <button
        type="submit"
        className="bg-blue-500 text-white p-3 w-full rounded-md
hover:bg-blue-600"
      >
        أضف المهمة
      </button>
    </form>
  );
```

```
};

export default TaskForm;
```

التنسيق:

- استخدمنا خلفية جذابة بتدرجات ألوانTailwind CSS ، مع تصميم بسيط وحديث لعناصر القائمة والنماذج.
- الزر مصمم ليكون متجاوبًا مع تدرج لوني جذاب وحواف مستديرة لتوفير تجربة مستخدم مريحة.

تمرين: تطبيق تتبع الطقس(Weather Tracker)

في هذا التمرين، سنقوم بإنشاء تطبيق بسيط لتتبع الطقس يعرض بيانات الطقس الحالية بناءً على المدينة التي يدخلها المستخدم. سنستخدم useStateلتخزين البيانات، و useEffectلجلب البيانات منAPI ، وpropsلتمرير البيانات بين المكونات. سنقوم بتصميم واجهة مستخدم أنيقة باستخدامTailwind CSS.

المتطلبات:

1. إدخال اسم مدينة لعرض بيانات الطقس.
2. عرض بيانات الطقس مثل درجة الحرارة وحالة الطقس.
3. تصميم عصري وجذاب باستخدامTailwind CSS.

App.js:

```
import React, { useState } from 'react';
import WeatherForm from './WeatherForm';
import WeatherInfo from './WeatherInfo';

const App = () => {
    const [weather, setWeather] = useState(null);

    // التعامل مع البيانات الواردة من WeatherForm
    const fetchWeather = async (city) => {
        const apiKey = 'YOUR_API_KEY'; // استبدل بمفتاح API الخاص بك
        const response = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=
${apiKey}&units=metric`);
        const data = await response.json();
        setWeather(data);
    };

    return (
        <div className="min-h-screen bg-gradient-to-r from-teal-400 to-
blue-500 p-8 flex flex-col items-center">
```

```
            <h1 className="text-4xl font-extrabold text-white mb-6">تتبع
الطقس</h1>
            <WeatherForm fetchWeather={fetchWeather} />
            {weather && <WeatherInfo weather={weather} />}
        </div>
    );
};

export default App;
```

WeatherForm.js:

```
import React, { useState } from 'react';

const WeatherForm = ({ fetchWeather }) => {
    const [city, setCity] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        if (city.trim() !== '') {
            fetchWeather(city);
            setCity('');
        }
    };

    return (
        <form onSubmit={handleSubmit} className="bg-white p-6 rounded-
lg shadow-md mb-6 max-w-lg w-full">
            <input
                type="text"
                className="border p-3 w-full mb-4 rounded-md"
                placeholder="أدخل اسم المدينة"
                value={city}
                onChange={(e) => setCity(e.target.value)}
            />
            <button type="submit" className="bg-teal-500 text-white p-3
w-full rounded-md hover:bg-teal-600">
                ابحث عن الطقس
            </button>
        </form>
    );
};

export default WeatherForm;
```

WeatherInfo.js:

```
import React from 'react';

const WeatherInfo = ({ weather }) => {
    const { main, weather: weatherData, name } = weather;
    const temperature = main.temp.toFixed(1);
    const description = weatherData[0].description;

    return (
        <div className="bg-white p-6 rounded-lg shadow-lg max-w-lg w-
full text-center">
            <h2 className="text-2xl font-bold text-gray-700 mb-
4">{name}</h2>
            <p className="text-4xl font-semibold text-gray-800 mb-
2">{temperature}°C</p>
            <p className="text-lg font-medium text-gray-600
capitalize">{description}</p>
        </div>
    );
};

export default WeatherInfo;
```

الفصل السادس: إدارة الحالة المعقدة باستخدام useReducer

فهم useReducer

ما هو useReducer وكيفية عمله؟

useReducer هو Hook في React يستخدم لإدارة الحالة المعقدة عندما تكون هناك حاجة للتعامل مع تحديثات متعددة بناءً على أنواع مختلفة من الإجراءات. يعمل useReducer بشكل مشابه لـ useState، ولكنه يوفر هيكلًا أفضل عندما يكون لديك منطق حالة معقد.

كيف يعمل useReducer:

- **الحالة الأولية (Initial State):** تحدد الحالة الأولية للتطبيق.
- **دالة التخفيض (Reducer Function):** تأخذ الحالة الحالية والإجراء، وتعيد الحالة الجديدة بناءً على نوع الإجراء.
- **إرسال الإجراءات (Dispatch Actions):** نستخدم dispatch لإرسال إجراءات إلى دالة التخفيض لتحديث الحالة.

مثال نظري:

تخيل أنك تدير مخزون متجر. لديك حالة تتضمن كمية المخزون ومعلومات أخرى. عندما يتلقى المتجر طلبًا، تحتاج إلى تحديث الكمية المتاحة في المخزون. بدلاً من تحديث الحالة مباشرة، يمكنك استخدام useReducer لإدارة التحديثات المختلفة التي قد تحدث عند تلقي الطلبات أو إرجاع البضائع.

الفرق بين useReducer و useState

- **useState:** يستخدم لإدارة حالات بسيطة حيث تحتاج إلى تخزين قيمة واحدة أو بضع قيم فقط. على سبيل المثال، عداد عدد الزوار على موقع ويب.
- **useReducer:** يستخدم عندما يكون لديك منطق حالة معقد يتطلب تحديثات متعددة بناءً على أنواع مختلفة من الإجراءات. يمكن أن يكون مفيدًا إذا كانت الحالة تتضمن كائنات أو مصفوفات ويجب تحديث أجزاء مختلفة منها.

مثال نظري:

إذا كان لديك نموذج يتطلب إدارة معلومات المستخدم (مثل الاسم، البريد الإلكتروني، وتفاصيل أخرى)، يمكن أن يصبح استخدام useState صعبًا عند التعامل مع التحديثات المتعددة. هنا يأتي دور useReducer حيث يمكنه تسهيل إدارة الحالة المعقدة بتحديد إجراءات مختلفة لتحديث أجزاء مختلفة من الحالة.

متى نستخدم useReducer؟

- **الحالات المعقدة:** عندما تكون الحالة عبارة عن كائن أو مصفوفة ويجب تحديث أجزاء متعددة منها.
- **أنواع مختلفة من التحديثات:** عندما تحتاج إلى تنفيذ أنواع متعددة من التحديثات على الحالة بناءً على إجراءات مختلفة.

- **تنظيم الكود :**عندما تريد فصل منطق الحالة عن مكوناتReact ، مما يجعل الكود أكثر وضوحًا وقابلية للصيانة.

بناء دوال التخفيض (Reducer Functions)

كيفية إنشاء دوال التخفيض

دالة التخفيض هي دالة تحدد كيف يتم تحديث الحالة بناءً على نوع الإجراء المرسل إليها. تأخذ دالة التخفيض الحالة الحالية والإجراء، وتعيد الحالة الجديدة.

مثال نظري:

إذا كنت تدير عدادًا لموقع ويب، فدالة التخفيض ستقرر كيفية زيادة أو تقليل العدد بناءً على الإجراء المرسل (زيادة أو تقليل).

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error('Unknown action type');
  }
}
```

تنظيم الكود باستخدام useReducer

استخدام useReducerيساعد في تنظيم الكود بشكل أفضل عندما تتعامل مع منطق حالة معقد. يمكنك فصل منطق التحديث عن مكوناتReact ، مما يجعل الكود أكثر وضوحًا وسهولة في الصيانة.

مثال متقدم:

عندما يكون لديك حالة معقدة تشمل عدادًا، رسالة، وحالة تحميل، يمكنك استخدام useReducer لتحديث كل جزء من الحالة بناءً على نوع الإجراء.

مثال على استخدام useReducerبشكل متقدم:

```
export default App;
import React, { useReducer } from 'react';

// الحالة الأولية
const initialState = {
  count: 0,
  message: '',
  loading: false,
};
```

```
// دالة التخفيض
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { ...state, count: state.count + 1 };
    case 'decrement':
      return { ...state, count: state.count - 1 };
    case 'setMessage':
      return { ...state, message: action.payload };
    case 'setLoading':
      return { ...state, loading: action.payload };
    default:
      throw new Error('Unknown action type');
  }
}

function AdvancedCounter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <p>Message: {state.message}</p>
      <p>Loading: {state.loading ? 'Yes' : 'No'}</p>
      <button onClick={() => dispatch({ type: 'increment'
})}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement'
})}>Decrement</button>
      <button onClick={() => dispatch({ type: 'setMessage', payload:
'Hello!' })}>Set Message</button>
      <button onClick={() => dispatch({ type: 'setLoading', payload:
true })}>Start Loading</button>
      <button onClick={() => dispatch({ type: 'setLoading', payload:
false })}>Stop Loading</button>
    </div>
  );
}
```

تمرين شامل: تطبيق إدارة قائمة التذكيرات مع تقويم

وصف التمرين:

في هذا التمرين، سنقوم بإنشاء تطبيق لإدارة قائمة التذكيرات، حيث يمكن للمستخدمين إضافة تذكيرات جديدة، تحديد تواريخ للتذكيرات، وعرض التذكيرات على تقويم شهري. سيكون لدينا واجهة تعرض التذكيرات اليومية وتسمح بإضافة تذكيرات جديدة مع تواريخها.

هيكل التطبيق:

1. **App:** المكون الرئيسي الذي يتضمن مكونات قائمة التذكيرات والتقويم.
2. **ReminderList:** مكون يعرض قائمة التذكيرات اليومية.
3. **ReminderItem:** مكون يعرض كل تذكير فردي.
4. **AddReminder:** مكون يتيح للمستخدمين إضافة تذكيرات جديدة مع تحديد تاريخ.
5. **CalendarView:** مكون يعرض التذكيرات على تقويم شهري.

متطلبات:

npm install date-fns

app.js:

```
import React, { useReducer, useEffect } from 'react';
import ReminderList from './ReminderList';
import AddReminder from './AddReminder';
import CalendarView from './CalendarView';

// الحالة الأولية
const initialState = {
    reminders: []
};

// دالة التخفيض
function reducer(state, action) {
    switch (action.type) {
        case 'ADD_REMINDER':
            return { ...state, reminders: [...state.reminders,
action.payload] };
        case 'REMOVE_REMINDER':
            return { ...state, reminders:
state.reminders.filter(reminder => reminder.id !== action.payload) };
        case 'LOAD_REMINDERS':
            return { ...state, reminders: action.payload };
        default:
            throw new Error('Unknown action type');
    }
}
```

```
function App() {
    const [state, dispatch] = useReducer(reducer, initialState);

    useEffect(() => {
        // محاكاة تحميل التذكيرات من API
        const loadReminders = async () => {
            const reminders = await fetch('/api/reminders').then(res =>
res.json());
            dispatch({ type: 'LOAD_REMINDERS', payload: reminders });
        };

        loadReminders();
    }, []);

    return (
        <div className="min-h-screen bg-gray-100 flex flex-col items-
center p-4">
            <h1 className="text-4xl font-bold mb-4">إدارة التذكيرات</h1>
            <AddReminder dispatch={dispatch} />
            <div className="flex w-full max-w-4xl">
                <ReminderList reminders={state.reminders}
dispatch={dispatch} />
                <CalendarView reminders={state.reminders} />
            </div>
        </div>
    );
}

export default App;
```

AddReminder.js:

```
import React, { useState } from 'react';
import { v4 as uuidv4 } from 'uuid';

function AddReminder({ dispatch }) {
    const [reminderText, setReminderText] = useState('');
    const [reminderDate, setReminderDate] = useState('');

    const handleAddReminder = () => {
        if (reminderText.trim() && reminderDate) {
            dispatch({
                type: 'ADD_REMINDER',
                payload: { id: uuidv4(), text: reminderText, date:
reminderDate }
            });
            setReminderText('');
```

```
            setReminderDate('');
        }
    };

    return (
        <div className="w-full max-w-md bg-white p-4 shadow-md rounded
mb-4">
            <h2 className="text-xl font-semibold mb-2">إضافة تذكير</h2>
            <input
                type="text"
                value={reminderText}
                onChange={(e) => setReminderText(e.target.value)}
                placeholder="أدخل نص التذكير"
                className="w-full p-2 border border-gray-300 rounded
mb-2"
            />
            <input
                type="date"
                value={reminderDate}
                onChange={(e) => setReminderDate(e.target.value)}
                className="w-full p-2 border border-gray-300 rounded
mb-2"
            />
            <button
                onClick={handleAddReminder}
                className="bg-blue-500 text-white px-4 py-2 rounded
hover:bg-blue-600"
            >
                إضافة تذكير
            </button>
        </div>
    );
}

export default AddReminder;
```

CalendarView.js:

```
import React from 'react';
import { format, startOfMonth, endOfMonth, eachDayOfInterval } from
'date-fns';

function CalendarView({ reminders }) {
    const today = new Date();
    const start = startOfMonth(today);
    const end = endOfMonth(today);
    const days = eachDayOfInterval({ start, end });
```

```
    const getRemindersForDay = (date) => {
        return reminders.filter(reminder => new
Date(reminder.date).toDateString() === date.toDateString());
    };

    return (
        <div className="w-1/2 p-4 bg-white shadow-md rounded">
            <h2 className="text-2xl font-semibold mb-4">التقويم</h2>
            <div className="grid grid-cols-7 gap-2">
                {days.map(day => (
                    <div key={day} className="border p-2 rounded">
                        <div className="font-semibold text-
center">{format(day, 'd')}</div>
                        <ul>
                            {getRemindersForDay(day).map(reminder => (
                                <li key={reminder.id} className="text-
sm">{reminder.text}</li>
                            ))}
                        </ul>
                    </div>
                ))}
            </div>
        </div>
    );
}
export default CalendarView;
```

ReminderItem.js:

```
import React from 'react';

function ReminderItem({ reminder, dispatch }) {
    return (
        <li className="bg-gray-100 p-4 mb-2 shadow-md rounded flex
justify-between items-center">
            <span>{reminder.text} - {new
Date(reminder.date).toLocaleDateString()}</span>
            <button
                onClick={() => dispatch({ type: 'REMOVE_REMINDER',
payload: reminder.id })}
                className="bg-red-500 text-white px-4 py-2 rounded
hover:bg-red-600"
            >
                حذف
            </button>
        </li>
    );
}
```

```
export default ReminderItem;
```

ReminderList.js:

```
import React from 'react';
import ReminderItem from './ReminderItem';

function ReminderList({ reminders, dispatch }) {
    return (
        <div className="w-1/2 p-4 bg-white shadow-md rounded">
            <h2 className="text-2xl font-semibold mb-4">قائمة التذكيرات</h2>
            <ul>
                {reminders.map(reminder => (
                    <ReminderItem key={reminder.id} reminder={reminder}
dispatch={dispatch} />
                ))}
            </ul>
        </div>
    );
}

export default ReminderList;
```

شرح التمرين:

1. **إدارة الحالة باستخدام useReducer:**
   - نستخدم useReducer لإدارة قائمة التذكيرات، حيث يمكننا إضافة وإزالة التذكيرات بطريقة منظمة.
2. **تحميل البيانات باستخدام useEffect:**
   - نستخدم useEffect لمحاكاة تحميل التذكيرات من واجهة برمجة التطبيقات عند بدء التشغيل.
3. **إرسال البيانات بين المكونات باستخدام props:**
   - نرسل الحالة والدالة dispatch بين المكونات المختلفة (AddReminder, ReminderList, وReminderItem) لتحديث حالة التذكيرات وعرضها.
4. **تصميم عصري باستخدام Tailwind CSS:**
   - استخدمنا Tailwind CSS لتصميم واجهة المستخدم بشكل عصري وجذاب، مع ألوان رائعة وتنسيق ممتاز.

الفصل السابع: دارة الحالة العامة باستخدام useContext

المراجع العامة:

رابط الملفات والمشاريع ضمن الكورس هذا على GitHub:

المراجع العامة المساعدة في كتابة هذا الكتاب: