React

תוכנת קוד פתוח שפורסמה על ידי פייסבוק (מטא) שמטרתה בניית אתרי אינטרנט מורכבים.

ריאקט היא פריימוורק (סביבת פיתוח) מבוססת קומפוננטות.

קומפוננטה היא רכיב ויזואלי.

ה-ui מחולק לחלקים קטנים, כל אחד מהם הוא קומפוננטה, שגם יכולה להכיל בתוכה קומפוננטות אחרות.

ריאקט היא SPA – Single Page Application

התקנות

* Npm I –g create-react-app
* בדיקה שעובד: create-react-app –version
* יצירת אפליקציית ריאקט חדשה:

create-react-app kitten-website --template typescript

* יצירת קומפוננטות אוטומטית: npm i -g react-cli-snippets
  + ואז: create fc home/Home

הרצת האפפ: npm start

Tsx – typescript extended

טייפסקריפט שאפשר לכתוב בתוכו קוד html

סוגי קומפוננטות:

* פעם נהגו לבנות Class components, הסתבר שזו עבודה מאד מורכבת וסזיפית
* כיום נהוג לעבוד עם Function components

ניתן לרנדר רשימות בריאקט באמצעות ה-HOF map וריאקט חוסכת לנו את הצורך ב-reduce. עם זאת, ריאקט מכריחה אותנו לתייג כל תגית מרשימה באמצעות מאפיין הנקרא key. בדרך כלל נרנדר לתוך ה-key את שדה ה-id של הרשומה.

על button ניתן להוסיף onClick ולעשות אינטרפולציה לפונקציה. למשל:

            <button onClick={displaySale}>Display Sale</button>

State

מצב. ספציפית בעולם התוכנה, מתאר את חתימת הדאטה הנוכחית שיש לאובייקט מסויים.

במקרה שלנו, האובייקט הוא קומפוננטה. בריאקט, הקומפוננטה מנהלת את הדאטה וכדי לשנות, ליצור, למחוק, כל דבר שקשור לדאטה, עלינו להשתמש בכלים של ריאקט ולא לנסות למשל לשנות דאטה על ידי שינוי ערך של משתנים. פשוט כי זה לא יעבוד.

Side Effect

לפעמים, קומפוננטה צריכה לבצע פעולות מחוץ לתחום גזרתה: ללכת להביא json משרת מרוחק. למשל לשאוב מידע מקומפוננטה אחרת. למשל להשתמש במשתנים גלובליים (כמו window).

כדי לבצע פעולות כאלה, אסור בתכלית האיסור לבצע אותן בצורה ישירה, מכיוון שאם הן יסתיימו ברינדור (שיבלול) מחדש של הקומפוננטה, מה שיגרום לקריאה מחדש לפעולה החיצונית. וזה דבר לא רצוי.

פעולה חיצונית כזו נקראת side effect והדרך החוקית היחידה לבצע אותה, היא דרך פונקציה של ריאקט שנקראת useEffect

הפונקציית callback שאנחנו מספקים ל-useEffect מתבצעת אך ורק פעם אחת בעת הרינדור הראשון של הקומפונטטה ולא באף רינדור אחר.

אלא אם כן, אחד המשתנים שהעברנו במערך שהוא הפרמטר השני, משתנה.

למשל:

useEffect(() => {

        setInterval(() => {

            console.log('displaying time')

            displayTime()

        }, 1000)

    }, [time])

האפקט בדוגמה הזו, יתרחש פעם אחת בעת רינדור ראשוני של הקומפוננטה, ואז רק אם משתנה ה-time יעודכן. (במקרה זה שוב ניצור סוג של לולאה אינסופית של רינדור – להיזהר ממצבים כאלה)

React strict mode

מוד עבודה דיפולטי של ריאקט, נוצר במהלך create-react-app כתגית הבסיס לאפליקציה ב-index.tsx. ניתן להיפטר ממנו לעת עתה.

Routing

זהו ניתוב פנימי באתר, ונועד להציג את הדף הנכון לפי ה-url שהמשתמש הקליד (או לחץ) וגם, להציג בשורת ה-url את הנתיב הנוכחי.

<https://mysite.com/products> in this case, /products is the route

https://mysite.com/sports/basketball in this case, /sports/basketball is the route

כדי להשתמש בראוטינג, עלינו להתקין npm חדש.

npm i react-router-dom @types/react-router-dom

התקנת שרת מקומי של northwind:

npm i –g northwind-back-end

npm i –g northwind-rest-api

הבאת נתונים משרת מרוחק:

1. לבנות קלאס עבור קונפיגורציה של המערכת. נהוג לקרוא לקלאס הזה AppConfig ולמקם אותו בתיקיית utils
2. לבנות קלאס מסוג Model. מודלים הם קלאסים שמתארים נתונים, כמו למשל נתונים שמגיעים משרת מרוחק.
3. בניית קלאס מסוג service. קלאסים מסוג זה מכילים קוד לוגי בלבד, למשל שמיישם גישה לשרת מרוחק, אין להם שום UI, והם משמשים כשכבה הלוגית שבאמצעותה הקומפוננטות מיישמות לוגיקה.
4. במילים אחרות: קוד לוגי שלא קשור לתצוגה, לא אמור לשבת בתוך קומפוננטה, אלא בתוך service, והקומפוננטה תשתמש ב-service

npm i axios

DOM

Document Object Model – מבנה האובייקטים שמחזיק הדפדפן בזכרון ומייצגים את דף ה-HTML. כל תגית ב-HTML מיוצגת על ידי אובייקט בזכרון.

כשאנחנו מתכנתים ב-javascript אנחנו יכולים לגשת לאלמנטים ב-DOM ולשנות אותם. למשל על ידי שימוש ב-getElementById או querySelector.

כשאנחנו משנים ישירות את ה-DOM, הדפדפן מתבקש לרנדר מחדש את הדף, לא תמיד את כולו, לפעמים רק מנקודת השינוי ומטה.

גישות מרובות ועוקבות ל-DOM גורמות לרינדור מחדש של הדף פעם אחת עבור כל גישה ל-DOM, ולכן, מאד לא יעילות.

**Virtual DOM**

זהו העתק של ה-DOM שריאקט מנהלת לצרכי יעילות. כשאנו עושים שינויים בדאטה (בסטייט מסויים) ריאקט לא נוגעת ב-DOM של הדפדפן. אלא, היא מעדכנת אובייקט DOM פנימי משלה, הנקרא Virtual DOM, ומתי שהיא מחליטה שהגיעה הזמן לרנדר את המסך מחדש, היא עושה זאת.

התוצאה, היא עבודה הרבה יותר יעילה, כי ריאקט מסוגלת לאגד מספר שינויים בדום הוירטואלי, ואז להחיל אותם על ה-DOM המוחשי (של הדפדפן) במכה אחת.

מה שזה אומר, זה שריאקט צריכה לסנכרן את ה-Virtual DOM עם ה-DOM של הדפדפן, ואם הסינכרון יישבר משום מה (בגללנו) אז ריאקט תאבד קשר עם ה-DOM ודברים לא צפויים יתחילו לקרות.

לכן, אסור בתכלית האיסור, בתוך ריאקט, לנסות לשנות ישירות את ה-DOM, כלומר, השימוש ב-document.getElementById אסור!

כמו כן, אסור לתת מאפייני id לתגיות, כי

1. ריאקט עשויה להצטרך לתת id לתגיות בעצמה לצורך תיפקודה
2. אם לא ניתן id, לא נתפתה להשתמש ב-getElementById
3. אם ניתן id בקומפוננטה מסויימת, אין שום מניעה לרנדר את הקומפוננטה מספר פעמים, אז אנחנו יוצאים כפילות בזמן ש-id צריך להיות ערך חד חד ערכי.

**Hooks**

כשאנו מפתחים באמצעות framework מסויים (ריאקט למשל היא פריימוורק, וורדפרס למשל היא פריימוורק) אנו יכולים בדרך כלל, להגדיר פונקציות מסויימות, בכוונה תחילה שהפריימוורק עצמה, תפעיל אותן בשבילנו.

גם ריאקט מאפשרת שימוש ב-hooks

הכללים של ריאקט ל-hooks

1. פונצקיה שהיא hook חייבת להתחיל במילה use. למשל useState, useEffect
2. ניתן להשתמש ב-hooks אך ורק בתוך functional components או בתוך hooks אחרים
3. הקריאה ל-hook חייבת להתבצע ישירות בתוך הקומפוננטה, ולא בתוך תנאי או לולאה
4. ניתן לבנות custom hooks משלנו.

Props

קיצור של properties. משמש לשליחת מידע מ-parent component ל-child component

לפרמטר ה-props נהוג לתת טיפוס בשם הקומפוננטה (קומפוננטת ה-child, לא ה-parent), בתוספת הסיומת Props

אם למשל שם הקומפוננטת child הוא ProductCard אזי שם הטיפוס עבור ה-props יהיה ProductCardProps.

נגדיר את הטיפוס הזה כ-interface

Route parameters

לפעמים נרצה להעביר מידע מקומפוננטה לקומפוננטה אחרת בזמן ניווט בין אחת לשנייה. כאן, לא נוכל להשתמש ב-props מכיוון שאין כאן קשר הורה-ילד החיוני לשימוש ב-props. במקרים כאלו נשתמש ב-route params.

נגיד היינו רוצים להעביר דאטה כלשהו בשורת url. היינו יכולים לעשות כך:

<http://mysite.com/products/details?product-id=8>

http: protocol (udp, http, https, ftp)

mysite.com: domain

/products/details: path

?... : query string

Param1=value1&param2=value2&param3=value3

בריאקט אנחנו משתמשים בשיטה אחרת:

לא כותבים כך:

<http://mysite.com/products/details?product-id=8>

אלא:

<http://mysite.com/products/details/8>

טפסים

ישנן כל מיני דרכים ושיטות להתמודד עם טפסים וסיבמוטים של טפסים בריאקט, אנחנו נשתמש בחבילה הפופולרית ביותר היום הנקראת react-hook-form

מתקינים: npm i react-hook-form

הודעות הצלחה ושגיאה (חיווי)

לצרכי פיתוח, נחמד מאד להשתמש ב-alert אך מבחינת שימושיות למשתמש הקצה מדובר בלא פחות מאסון.

כמו כן, במקרה של שגיאה, אין להציג למשתמש את הודכעת השגיאה של השרת, אלא הודעה מותאמת, התעזור לו להתגבר על השגיאה ולהצליח להשתמש במערכת. אם לא ננהג כך, האתר\אפליקציה שלנו לא יהיו ידידותיים למשתמש, ובסופו של דבר אף אחד לא ישתמש בהם.

אנחנו נעבוד עם ספרייה שנקראת notyf

npm i notyf

Redux

רידאקס היא ספרייה לניהול state ברמת האפליקציה. כלומר – לא ברמת הקומפוננטה. כש-state מנוהל ברמת האפליקציה, כל מיני קומפוננטות יכולות לעדכן אותו, ולהתעדכן ממנו. וכך, מובטח לי ששינוי בקומפוננטה מסויימת, יכול להשתקף בעוד קומפוננטות על המסך.

דוגמה קלאסית לצורך ב-state ברמת האפליקציה (נקרא גם state גלובלי) : עגלת קניות. (דוגמת אלי אקספרס)

קיימים שני "סגנונות" לעבודה עם רידאקס:

1. Classic redux
2. Redux toolbox

הטולבוקס אינו כתיבה מחדש של הרידאקס, אלא ספרייה המשתמשת ברידאקס הקלאסי ומאפשרת למפתחים ליצור אפליקציות רידאקס יותר פשוטות ובפחות שורות קוד.

אנחנו נלמד דווקא את הרידאקס הקלאסי (ואז מעבר לטולבוקס אמור להיות מאד פשוט)

התקנת רידאקס קלאסי : npm i redux

5 שלבים לבניית רידאקס:

1. Global State: מאגר מידע גלובלי, עבור יישות מסויימת למשל products, employees, categories וכיו"ב. אנחנו מיישמים אותו על ידי class
2. ActionType – אילו פעולות אנחנו רוצים לאפשר לבצע על הגלובל סטייט. קרי, על המידע עצמו. מדובר בסט סגור של פעולות, למשל: הוספת מוצר, מחיקת מוצר, טעינת מוצרים... לצורך כך נשתמש ב-enum
3. ActionObject – זהו אובייקט היתאר פעולה אחת ספציפית שאנו רוצים לעשות על המוצר, למשל מה המידע הדרוש עבור הוספת מוצר. את ה-ActionObject נתאר באמצעות interface
4. Reducer – זו פונקציה שלעולם לא תיקרא ישירות על ידינו, אלא אנחנו נכתוב את הקוד שלה, כדי שרידאקס תפעיל אותה בשבילנו. הפונקציה הזו מקבלת את הסטייט הנוכחי, ואת הפעולה שאנחנו רוצים לבצע על המידע, היא צריכה להחזיר לנו, את תצורת המידע לאחר הפעולה. למשל: יש לנו כרגע בסטייט מערך של 80 מוצרים. קיבלנו action מסוג deleteProduct ו-payload עם ה-productId למחיקה. תפקיד ה-reducer הוא למצוא את המוצר למחיקה בסטייט הנוכחי, ולהחזיר סטייט חדש, שלא כולל את המוצר למחיקה, קרי להחזיר מערך ובו 79 מוצרים ללא המוצר שיש למחוק. אסור לנו לגוע ישירות בסטייט, רק לרידאקס מותר לעשות שינויים בסטייט. אנחנו בזה שאנחנו מחזירים אובייקט סטייט חדש, אנחנו מאפשרים לרידאקס לעדכן את הסטייט, אך אם נעדכן אותו ישירות, רידאקס לא תוכל יותר לשלוט בסטייט, ויקרו תוצאות בלתי צפויות. בדיוק כשם שבריאקט אנחנו לא נוגעים ישירות ב-DOM.
5. Store – זהו אובייקט שייחשף לאפליקציה שלנו, ובאמצעותו נוכל לתקשר עם רידאקס. הוא בעצם מייצג את ה-GloblStore עבור האפפ שלנו.

אחת המטרות העיקריות שלשמן ריאקט ורידאקס נוצרו מלכתחילה היא לסנכרן מידע בין קומפוננטות שונות הנמצאות על אותו מסך.

לצורך כך, נשתמש בפעולת רידאקס הנקראת subscribe

כל קומפוננטה יכולה לעשות subscribe ל-store כלשהו ברידאקס ועל ידי כך להתעדכן כל הזמן כאשר קורים שינויי דאטה ב-store.

מרגע שעשינו subscribe ל-store כלשהו, חובה עלינו לדאוג לעשות לו unsubscribe בתזמון המתאים. מהו התזמון המתאים? ברגע שהקומפוננטה נהרסת. מתי קומפוננטה נהרסת? להזכירכם, קומפוננטה היא פיסת GUI, ולכן היא נהרסת מרגע שהיא אינה מוצגת יותר על המסך.