Forms

מה הדרך הכי טובה לטפל בטפסים עם ריאקט?

מטרות השיעור

- גישות שונות לטיפול בטפסים ומתי להשתמש בכל אחת?
- קוד חוזר? בואו נכיר דפוסים שיעזרו לנו לפתור קוד שחוזר ב- React components
- ספרייה שתעזור לנו לטפל בטפסים - Formik

תרגיל:
בואו נכין טופס התחברות

שאלה:
במרבית הטפסים מה הפעולות שחוזרות על עצמן ?

פעולות בטפסים

- אימות הקלט
- הצגת הודעות שגיאה
- תפיסת הקלט מהמשתמש
- לוגיקה של הטופס

שאלה:
אילו דרכים React מספקת
להתמודדות עם טפסים?

דרך 1:
ללא שליטה ב- state

ללא שליטה

- בשליטת האלמנט ב- DOM
- ניתן להביא ערך התחלתי עם defaultValue
- פחות קפדני ונותן למשתמש לערוך טקסט עם פחות התערבות של ה- component ולרוב אנחנו ניקח את הערך רק בנקודה שעושים submit
- בשביל להשיג זאת אנחנו צריכים לתפוס את ה- DOM element של הפקד בטופס

שאלה:

איך נתפוס את ה- DOM element

אחרי ש- React מניח אותו ?

Ref

- מאפשר לתפוס את האלמנט dom כאשר הוא מונח
- זהו attribute שאנחנו שמים על האלמנט שאנחנו רוצים לתפוס
- את ה- ref אנחנו יוצרים עם:
  - this.myRef = React.createRef()
- את האלמנט עליו הנחנו את ה- ref אנחנו תופסים עם: this.myRef.current
- במידה והנחנו את ה- ref על React Component אז אנחנו מקבלים את ה- instance של ה- component class

דרך 2 לטיפול בדפסים:
טופס בשליטה ב- state

controlled form

- בשיטה זו פקדי הטופס שלנו ישמרו ב- state של ה- React Component
- לכל פקד ה- value יהיה מחובר למשתנה ב- state
- כל פקד יממש פונקצית עדכון שתעדכן את המשתנה ב- state

תרגיל:
ממש את ה- form הקודם על ידי
שליטת הפקדים ב- state

בעיה:
קוד משותף בין טפסים שונים

פתרון שגוי:
ירושה

למה לא נעשה ירושה אף פעם ב- React

- מונע מהילד לעבוד בנפרד מהאבא
- קשה יותר להוסיף לוגיקה ל - Render
- יוצר תלותיות מיותרת לפעמים
- עלול להיווצר מצב של צורך בריבוי ירושות

פתרון נכון:
הרכבה - Composition

Composition

- בניגוד לירושה שאנחנו יורשים מהאבא , במקרה זה הילד קיים באבא והאבא משתמש בילד במקום המתאים
- האבא יכול להוסיף לילד כוחות נוספים
- באמצעות הדפוס הזה ניתן ליצור באבא את ההתנהגות המשותפת ובכך לפתור את בעיית שכפול הקוד שלנו

שאלה:

איך נעשה composition עם React כדי לפתור את בעית שכפול הקוד הנוכחית ?

HOC - Higher Order Component

- פונקציה שמחזירה את האבא
- צורות שימוש:
  - createParent(Son)
  - createParent(someConfigurations)(SON)
- משתמשים כדי ליצור שימוש חוזר של אלמנטים שחוזרים על עצמם ב- Class Component

תרגיל:
בואו ניצור HOC כדי לפתור את הקוד המשוכפל ב- Register ו- Login

דרך נוספת ליצור Composition עם React

render prop

- מיועד גם כדי לשים את הקוד המשותף של ה- class component ב- component נפרד לצורך reuse
- הקוד המשותף נקרא לו האבא הוא component שמקבל prop של פונקציה שמחזירה jsx של הבן.
- אל הפונקציה האבא יכול לשלוח ארגומנטים שאיתם הוא יעביר מידע חזרה לבן
- הבן מעביר מידה לאבא דרך ה props שלו

תרגיל:

בואו נוציא את הלוגיקה המשותפת של Login ו-Register וניצור Composition על ידי render props

שאלה:
איך אני משתף לוגיקה משותפת שמשלבת אירועים
ו- state ב- Component מבוססת פונקציה

custom Hooks

- שימוש ב- hooks ניתן לשים בפונקציה משותפת
- על ידי זה ניתן להוציא החוצה לוגיקה שחוזרת על עצמה
- הפונקציות האלה נקראות Custom hooks
- הם צריכות להתחיל במילה use

תרגיל:

נוציא את הלוגיקה שחוזרת על עצמה ב- Login מבוסס הפונקציה על ידי שימוש ב- custom hooks

שאלה:

מה זה Formik ואיך הוא יכול לעזור לי ביצירת הטפסים שלי

Formik

- ספריית עזר לבניית טפסים ב- React
- הטפסים נשמרים ב- state
- הספרייה משלבים עם render props או HOC
- הספרייה עוזרת לנו לטפל גם בולידציות והצגת שגיאות

תרגיל:
בנה את טופס ההתחברות עם Formik

סיכום

- למדנו על שתי דרכים שריאקט נותנת לנו לטפל בטפסים
- למדנו דפוסים שמאפשרים לנו להוציא קוד משותף מ - component מבוסס class
- למדנו להשתמש ב- custom hooks להוציא קוד משותף מ- component מבוסס פונקציה
- למדנו על formik