סקירת קוד מעמיקה עבור home.tsx
בדיקת הקוד נעשתה לסטודנט יוני ברוך

הקדמה
מסמך זה מספק סקירת קוד מעמיקה עבור קובץ home.tsx, תוך התמקדות באיכות הקוד, מבנה נכון, אופטימיזציה, אבטחה, תחזוקה ושיפור הביצועים. הבדיקה כוללת הסברים מפורטים, הצגת הבעיות המרכזיות בקוד, ודוגמאות לשיפורים משמעותיים שניתן לבצע כדי להפוך את הקוד לנקי, יעיל, ותחזוקתי יותר.

הקוד כפי שהוא כיום
1קריאות קוד וניהול מצבים
הקובץ משתמש במשתנים רבים עם useState, מה שמקשה על ניהול המצב ויוצר בלבול בתחזוקת הקוד. כמו כן, אין הגדרה ברורה לניהול טעינה ושגיאות, מה שעלול לגרום לבאגים כאשר מתבצעת קריאה לשרת.
.2ביצועי רשימה(FlatList)
הרשימה נטענת עם כל הנתונים בו-זמנית, דבר שעלול לגרום לזליגות זיכרון ולביצועים חלשים כאשר הרשימה גדולה. אין שימוש בפרמטרים אופטימליים של FlatList, כמו initialNumToRender-ו getItemLayout.
.3אבטחת מידע בטוקן המשתמש
נכון להיום, הטוקן של המשתמש נשלח לכל הבקשות ישירות מהקונטקסט, ללא מנגנון שמוודא שהטוקן עדכני וללא טיפול בשגיאות כאשר הוא אינו תקין.

בעיות מרכזיות בקוד
בעיה: ניהול מצב מפוצל ולא מסודר
הסבר:
- useState משמש לניהול מספר מצבים שונים במקום להשתמש בפתרון מודולרי יותר כמו useReducer.
- כאשר יש הרבה useState, כל עדכון קטן עלול לגרום לרינדור מחדש של הקומפוננטה כולה.

פתרון מוצע: שימוש ב useReducer-לניהול מצבים מרוכז וברור.

---

2 בעיה: טעינת כל הנתונים ברשימה גורמת לזליגת זיכרון
הסבר:
- FlatList לא מוגדר נכון, מה שגורם לו לטעון את כל הנתונים מראש, גם כאשר אין צורך בכך.
- הדבר עלול לגרום לקריסות כאשר מספר הפריטים גדול מאוד.

פתרון מוצע: שימוש בפרמטרים כגון initialNumToRender, windowSize, getItemLayout.-

---

3 בעיה: ניהול אבטחה חלש בטוקן המשתמש
הסבר:
- הטוקן נשלח ישירות מהקונטקסט, ללא בדיקה אם הוא תקף.
- אין מנגנון שמוודא שהמשתמש עדיין מחובר לפני ביצוע קריאה לשרת.

פתרון מוצע: יצירת פונקציה מרכזית לניהול הטוקן עם בדיקות תקינות.

סיכום והמלצות
1. **שיפור ניהול מצבים (useReducer)** כדי להפוך את ניהול הסטייט למובנה ומסודר יותר.

2. **אופטימיזציה של ביצועי רשימות גדולות (FlatList)** כדי למנוע זליגת זיכרון ולשפר את חוויית המשתמש.
3. **הקשחת אבטחה בניהול הטוקן** כדי למנוע בעיות אבטחה ולשפר את יציבות האפליקציה.

הקוד לאחר השינויים המוצעים:

.1

```
/*
before:
// ניהול מצבים בעזרת useState
const [lists, setLists] = useState<AbstractList[]>([]);
const [loading, setLoading] = useState<boolean>(true);
const [error, setError] = useState<string | null>(null);


// after
// ניהול מצבים משופר באמצעות useReducer
const initialState = {
    lists: [],
    loading: true,
    error: null,
  };

  function reducer(state, action) {
    switch (action.type) {
      case "FETCH_SUCCESS":
        return { ...state, lists: action.payload, loading: false };
      case "FETCH_ERROR":
        return { ...state, error: action.payload, loading: false };
      default:
        return state;
    }
  }

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

.2

```
  // שימוש ב-FlatList ללא אופטימיזציה
<FlatList
  data={lists}
  keyExtractor={(item) => item.id}
  renderItem={({ item }) => <ListComponent list={item} />}
/>




// שיפור ביצועי FlatList עם windowSize ו-initialNumToRender
<FlatList
  data={lists}
  keyExtractor={(item) => item.id}
  renderItem={({ item }) => <ListComponent list={item} />}
  windowSize={5} // מגביל את כמות הפריטים שנשמרים בזיכרון
  initialNumToRender={10} // מגדיר כמה פריטים לטעון בהתחלה
  getItemLayout={(data, index) => (
    { length: 50, offset: 50 * index, index }
  )} // מונע רינדור מיותר על ידי חישוב גובה קבוע
/>
```

.3

```
// שליחת טוקן באופן ישיר ללא בדיקה
const { userToken } = useAuth();
fetch("https://api.example.com/data", {
  headers: { Authorization: `Bearer ${userToken}` },
});



// שיפור ניהול טוקן עם בדיקה תקפה
const { getToken } = useAuth();

const fetchDataSecurely = async () => {
  try {
    const token = await getToken();
    if (!token) throw new Error("משתמש לא מחובר");

    const response = await fetch("https://api.example.com/data", {
      headers: { Authorization: `Bearer ${token}` },
    });
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("שגיאה באחזור נתונים:", error);
  }
};
*/
```