Das ist eine sehr gute Idee! Ein Benachrichtigungssystem mit Pflichtbewertungen würde die Nutzerinteraktion und Qualitätssicherung deutlich verbessern. Hier ist eine konkrete Implementierung dafür:

Benachrichtigungssystem mit Pflichtbewertungen

1. Datenmodell für Benachrichtigungen

Füge folgende Struktur zu deiner Firestore-Datenbank hinzu:

```
// In lib/firebase.ts - Benachrichtigungstypen definieren
export const NotificationTypes = {
  NEW_MESSAGE: 'new_message',
  TASK_MATCHED: 'task_matched',
  TASK_COMPLETED: 'task_completed',
  REVIEW_REQUIRED: 'review_required',
  NEW_TASK_NEARBY: 'new_task_nearby',
  APPLICATION_RECEIVED: 'application_received',
  REVIEW_RECEIVED: 'review_received'
};

/**
 * Benachrichtigung erstellen
 * @param userId Empfänger-ID
 * @param type Benachrichtigungstyp
 * @param data Zusätzliche Daten
 */
export const createNotification = async (
  userId: string,
  type: string,
  data: Record<string, any>
) => {
  try {
    const notificationsRef = collection(db, "notifications");

    await addDoc(notificationsRef, {
      userId,
      type,
      data,
      read: false,
      acted: false, // Wurde auf die Benachrichtigung reagiert (z.B. Bewertung abgegeben)
      createdAt: serverTimestamp(),
      priority: type === 'review_required' ? 'high' : 'normal' // Priorisierung
```

```
    });

    // Ungelesene Benachrichtigungen zählen
    const userRef = doc(db, "users", userId);
    await updateDoc(userRef, {
      unreadNotifications: increment(1)
    });

  } catch (error) {
    console.error("Fehler beim Erstellen der Benachrichtigung:", error);
  }
};

/**
 * Benachrichtigungen für einen Benutzer abrufen
 */
export const getUserNotifications = async (userId: string) => {
  try {
    const notificationsRef = collection(db, "notifications");
    const q = query(
      notificationsRef,
      where("userId", "==", userId),
      orderBy("createdAt", "desc"),
      limit(50)
    );

    const snapshot = await getDocs(q);

    return snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
  } catch (error) {
    console.error("Fehler beim Abrufen der Benachrichtigungen:", error);
    return [];
  }
};

/**
 * Benachrichtigung als gelesen markieren
 */
export const markNotificationAsRead = async (notificationId: string, userId: string) => {
  try {
    const notificationRef = doc(db, "notifications", notificationId);
    await updateDoc(notificationRef, {
      read: true
    });
```

```
    // Ungelesene Benachrichtigungen zählen
    const userRef = doc(db, "users", userId);
    await updateDoc(userRef, {
      unreadNotifications: increment(-1)
    });

    return true;
  } catch (error) {
    console.error("Fehler beim Markieren der Benachrichtigung:", error);
    return false;
  }
};

/**
 * Benachrichtigung als bearbeitet markieren
 */
export const markNotificationAsActed = async (notificationId: string) => {
  try {
    const notificationRef = doc(db, "notifications", notificationId);
    await updateDoc(notificationRef, {
      acted: true,
      actedAt: serverTimestamp()
    });

    return true;
  } catch (error) {
    console.error("Fehler beim Markieren der Benachrichtigung:", error);
    return false;
  }
};
```

2. Pflichtbewertungen bei Aufgabenabschluss

Aktualisiere die `completeTask`-Funktion, um eine Pflichtbewertung zu erstellen:

```
/**
 * Schließt eine Aufgabe ab und fordert eine Bewertung an
 */
export const completeTask = async (taskId: string) => {
  try {
    const taskRef = doc(db, "tasks", taskId);
    const taskSnapshot = await getDoc(taskRef);

    if (!taskSnapshot.exists()) {
      throw new Error("Aufgabe nicht gefunden");
    }
```

```
    const taskData = taskSnapshot.data();

    // Prüfen, ob die Aufgabe bereits abgeschlossen ist
    if (taskData.status === "completed") {
      throw new Error("Diese Aufgabe wurde bereits abgeschlossen");
    }

    // Prüfen, ob ein Tasker zugewiesen wurde
    if (!taskData.taskerId) {
      throw new Error("Diese Aufgabe hat keinen zugewiesenen Tasker");
    }

    // Aufgabenstatus aktualisieren
    await updateDoc(taskRef, {
      status: "completed",
      completedAt: serverTimestamp(),
      reviewStatus: "pending" // Neue Status-Property
    });

    // Bewertungsanforderung als Benachrichtigung für den Ersteller erstellen
    await createNotification(taskData.creatorId, NotificationTypes.REVIEW_REQUIRED, {
      taskId,
      taskTitle: taskData.title,
      taskerId: taskData.taskerId,
      requiresAction: true, // Erfordert eine Aktion vom Benutzer
      dueDate: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000) // 7 Tage Zeit
    });

    // Benachrichtigung für den Tasker erstellen
    await createNotification(taskData.taskerId, NotificationTypes.TASK_COMPLETED, {
      taskId,
      taskTitle: taskData.title
    });

    return true;
  } catch (error) {
    console.error("Fehler beim Abschließen der Aufgabe:", error);
    throw error;
  }
};
```

3. Benachrichtigungsseite erstellen

Erstelle eine neue Komponente `pages/NotificationsPage.tsx`:

```
import React, { useState, useEffect } from 'react';
import { useAuth } from '@/context/AuthContext';
import { getUserNotifications, markNotificationAsRead } from '@/lib/firebase';
import { NotificationCard } from '@/components/notifications/NotificationCard';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
import { Skeleton } from '@/components/ui/skeleton';
import { useTranslation } from 'react-i18next';
import { Bell } from 'lucide-react';

export default function NotificationsPage() {
  const { user } = useAuth();
  const { t } = useTranslation();
  const [notifications, setNotifications] = useState([]);
  const [loading, setLoading] = useState(true);
  const [activeTab, setActiveTab] = useState("all");

  useEffect(() => {
    const fetchNotifications = async () => {
      if (!user) return;

      try {
        setLoading(true);
        const fetchedNotifications = await getUserNotifications(user.uid);
        setNotifications(fetchedNotifications);
      } catch (error) {
        console.error("Fehler beim Laden der Benachrichtigungen:", error);
      } finally {
        setLoading(false);
      }
    };

    fetchNotifications();
  }, [user]);

  // Filterung nach Typ
  const filteredNotifications = activeTab === "all"
    ? notifications
    : notifications.filter(notif => {
        if (activeTab === "action_required") {
          return notif.data?.requiresAction && !notif.acted;
        }
        if (activeTab === "unread") {
          return !notif.read;
        }
        return true;
      });

  // Benachrichtigung als gelesen markieren
```

```
const handleMarkAsRead = async (notificationId) => {
  if (!user) return;

  const updated = await markNotificationAsRead(notificationId, user.uid);
  if (updated) {
    setNotifications(prev =>
      prev.map(notif =>
        notif.id === notificationId
          ? { ...notif, read: true }
          : notif
      )
    );
  }
};

return (
  <div className="container mx-auto p-4 max-w-3xl">
    <h1 className="text-2xl font-bold mb-4">{t('notifications.title')}</h1>

    <Tabs value={activeTab} onValueChange={setActiveTab} className="w-full mb-4">
      <TabsList className="grid grid-cols-3 w-full">
        <TabsTrigger value="all">
          {t('notifications.all')}
        </TabsTrigger>
        <TabsTrigger value="action_required">
          {t('notifications.actionRequired')}
        </TabsTrigger>
        <TabsTrigger value="unread">
          {t('notifications.unread')}
        </TabsTrigger>
      </TabsList>
    </Tabs>

    {loading ? (
      <div className="space-y-4">
        {[1, 2, 3].map((index) => (
          <div key={index} className="bg-white rounded-lg p-4 shadow-sm">
            <div className="flex items-center gap-3">
              <Skeleton className="h-10 w-10 rounded-full" />
              <div className="space-y-2 flex-1">
                <Skeleton className="h-4 w-3/4" />
                <Skeleton className="h-3 w-1/2" />
              </div>
            </div>
          </div>
        ))}
      </div>
    ) : filteredNotifications.length > 0 ? (
```

```
        <div className="space-y-4">
          {filteredNotifications.map((notification) => (
            <NotificationCard
              key={notification.id}
              notification={notification}
              onMarkAsRead={handleMarkAsRead}
            />
          ))}
        </div>
      ) : (
        <div className="bg-white rounded-lg p-8 text-center shadow-sm">
          <Bell className="h-12 w-12 text-gray-300 mx-auto mb-3" />
          <p className="text-gray-500">{t('notifications.empty')}</p>
        </div>
      )}
    </div>
  );
}
```

4. NotificationCard-Komponente

Erstelle eine neue Komponente `components/notifications/NotificationCard.tsx`:

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { Button } from '@/components/ui/button';
import { formatDistanceToNow } from 'date-fns';
import { de } from 'date-fns/locale';
import { useTranslation } from 'react-i18next';
import { markNotificationAsActed } from '@/lib/firebase';
import {
  Bell,
  MessageSquare,
  CheckCircle2,
  Star,
  AlertCircle,
  MapPin
} from 'lucide-react';

const NotificationTypes = {
  NEW_MESSAGE: 'new_message',
  TASK_MATCHED: 'task_matched',
  TASK_COMPLETED: 'task_completed',
  REVIEW_REQUIRED: 'review_required',
  NEW_TASK_NEARBY: 'new_task_nearby',
```

```
  APPLICATION_RECEIVED: 'application_received',
  REVIEW_RECEIVED: 'review_received'
};

interface NotificationCardProps {
  notification: any;
  onMarkAsRead: (id: string) => void;
}

export function NotificationCard({ notification, onMarkAsRead }: NotificationCardProps) {
  const navigate = useNavigate();
  const { t } = useTranslation();

  // Benachrichtigung als gelesen markieren, wenn darauf geklickt wird
  const handleClick = () => {
    if (!notification.read) {
      onMarkAsRead(notification.id);
    }
  };

  // Icon basierend auf Benachrichtigungstyp
  const getIcon = () => {
    switch (notification.type) {
      case NotificationTypes.NEW_MESSAGE:
        return <MessageSquare className="h-5 w-5 text-blue-500" />;
      case NotificationTypes.TASK_MATCHED:
      case NotificationTypes.TASK_COMPLETED:
        return <CheckCircle2 className="h-5 w-5 text-green-500" />;
      case NotificationTypes.REVIEW_REQUIRED:
        return <Star className="h-5 w-5 text-yellow-500" />;
      case NotificationTypes.NEW_TASK_NEARBY:
        return <MapPin className="h-5 w-5 text-indigo-500" />;
      case NotificationTypes.APPLICATION_RECEIVED:
        return <AlertCircle className="h-5 w-5 text-purple-500" />;
      default:
        return <Bell className="h-5 w-5 text-gray-500" />;
    }
  };

  // Titel basierend auf Benachrichtigungstyp
  const getTitle = () => {
    switch (notification.type) {
      case NotificationTypes.NEW_MESSAGE:
        return t('notifications.newMessage');
      case NotificationTypes.TASK_MATCHED:
        return t('notifications.taskMatched');
      case NotificationTypes.TASK_COMPLETED:
        return t('notifications.taskCompleted');
```

```
      case NotificationTypes.REVIEW_REQUIRED:
        return t('notifications.reviewRequired');
      case NotificationTypes.NEW_TASK_NEARBY:
        return t('notifications.newTaskNearby');
      case NotificationTypes.APPLICATION_RECEIVED:
        return t('notifications.applicationReceived');
      case NotificationTypes.REVIEW_RECEIVED:
        return t('notifications.reviewReceived');
      default:
        return t('notifications.newNotification');
    }
  };

  // Zeitpunkt formatieren
  const getTimeAgo = () => {
    if (notification.createdAt?.seconds) {
      const date = new Date(notification.createdAt.seconds * 1000);
      return formatDistanceToNow(date, { addSuffix: true, locale: de });
    }
    return '';
  };

  // Aktion für Benachrichtigungen, die eine Aktion erfordern
  const getAction = () => {
    if (notification.type === NotificationTypes.REVIEW_REQUIRED && !notification.acted) {
      return (
        <Button
          size="sm"
          onClick={(e) => {
            e.stopPropagation(); // Verhindert das Auslösen des Klick-Events für die gesamte
Karte

            // Navigiere zur Aufgabenseite mit geöffnetem Bewertungsdialog
            navigate(`/tasks/${notification.data.taskId}?showReview=true`);

            // Markiere die Benachrichtigung als bearbeitet
            markNotificationAsActed(notification.id);
          }}
        >
          {t('notifications.leaveReview')}
        </Button>
      );
    }

    if (notification.type === NotificationTypes.NEW_MESSAGE) {
      return (
        <Button
          size="sm"
```

```
          variant="outline"
          onClick={(e) => {
            e.stopPropagation();
            navigate(`/chats/${notification.data.chatId}`);
          }}
        >
          {t('notifications.viewMessage')}
        </Button>
      );
    }

    return null;
  };

  return (
    <div
      className={`p-4 rounded-lg shadow-sm cursor-pointer ${
        notification.read ? 'bg-white' : 'bg-blue-50'
      } ${notification.data?.requiresAction && !notification.acted ? 'border-l-4
border-yellow-500' : ''}`}
      onClick={() => {
        handleClick();

        // Standard-Navigation basierend auf Typ
        switch (notification.type) {
          case NotificationTypes.TASK_MATCHED:
          case NotificationTypes.TASK_COMPLETED:
          case NotificationTypes.REVIEW_REQUIRED:
            navigate(`/tasks/${notification.data.taskId}`);
            break;
          case NotificationTypes.NEW_MESSAGE:
            navigate(`/chats/${notification.data.chatId}`);
            break;
          case NotificationTypes.NEW_TASK_NEARBY:
            navigate(`/tasks/${notification.data.taskId}`);
            break;
          case NotificationTypes.APPLICATION_RECEIVED:
            navigate(`/tasks/${notification.data.taskId}`);
            break;
          default:
            // Keine Navigation
            break;
        }
      }}
    >
      <div className="flex items-center gap-3">
        <div className="flex-shrink-0">
          {getIcon()}
```

```
        </div>

        <div className="flex-1">
          <h3 className="font-medium">{getTitle()}</h3>
          <p className="text-sm text-gray-600 line-clamp-2">
            {notification.data?.taskTitle || notification.data?.message || ''}
          </p>
          <p className="text-xs text-gray-500 mt-1">{getTimeAgo()}</p>
        </div>

        {getAction()}
      </div>
    </div>
  );
}
```

5. Benachrichtigungsindikator für die Navbar

Füge einen Benachrichtigungsindikator zur Navbar hinzu:

```
// In components/layout/Navbar.tsx oder BottomNav.tsx

import { useAuth } from '@/context/AuthContext';
import { Bell } from 'lucide-react';
import { useState, useEffect } from 'react';
import { onSnapshot, query, collection, where, orderBy } from 'firebase/firestore';
import { db } from '@/lib/firebase';

// Innerhalb deiner Navbar-/BottomNav-Komponente
const [unreadCount, setUnreadCount] = useState(0);
const { user } = useAuth();

useEffect(() => {
  if (!user) return;

  // Ungelesene Benachrichtigungen in Echtzeit abonnieren
  const notificationsRef = collection(db, "notifications");
  const q = query(
    notificationsRef,
    where("userId", "==", user.uid),
    where("read", "==", false),
    orderBy("createdAt", "desc")
  );

  const unsubscribe = onSnapshot(q, (snapshot) => {
    setUnreadCount(snapshot.docs.length);
```

```
  });

  return () => unsubscribe();
}, [user]);

// In deinem JSX für den Benachrichtigungsbutton:
<Link to="/notifications" className="relative">
  <Bell className="h-6 w-6" />
  {unreadCount > 0 && (
    <span className="absolute -top-1 -right-1 bg-red-500 text-white rounded-full text-xs w-5
h-5 flex items-center justify-center">
      {unreadCount > 9 ? '9+' : unreadCount}
    </span>
  )}
</Link>
```

6. Integration der Pflichtbewertung in die TaskDetailPage

Modifiziere deine `pages/TaskDetailPage.tsx`:

```
import { useSearchParams } from 'react-router-dom';
import ReviewDialog from '@/components/reviews/ReviewDialog';

// In der TaskDetailPage-Komponente
const [searchParams] = useSearchParams();
const [showReviewDialog, setShowReviewDialog] = useState(
  searchParams.get('showReview') === 'true'
);

// Im useEffect, das die Aufgabe lädt
useEffect(() => {
  // Rest des vorhandenen Codes

  // Wenn die URL den Parameter showReview=true hat,
  // und der Benutzer der Ersteller ist, und der Status "completed" ist,
  // dann öffne den Bewertungsdialog automatisch
  if (
    searchParams.get('showReview') === 'true' &&
    user?.uid === task?.creatorId &&
    task?.status === 'completed' &&
    task?.reviewStatus === 'pending'
  ) {
    setShowReviewDialog(true);
  }
```

```
}, [task, user, searchParams]);

// In deinem JSX
<ReviewDialog
  isOpen={showReviewDialog}
  onClose={() => {
    setShowReviewDialog(false);
    // Wenn wir von einer Benachrichtigung kommen,
    // entferne den Query-Parameter aus der URL
    if (searchParams.get('showReview') === 'true') {
      navigate(`/tasks/${taskId}`, { replace: true });
    }
  }}
  // Verhindere das Schließen, wenn die Bewertung von einer Benachrichtigung kommt
  preventClose={searchParams.get('showReview') === 'true'}
  taskId={task?.id}
  userId={task?.taskerId}
  taskTitle={task?.title}
/>
```

7. Reviews-Status im Task-Modell berücksichtigen

Die `Task`-Dokumente benötigen ein neues Feld `reviewStatus` mit den möglichen Werten:

- `'pending'`: Bewertung ausstehend
- `'completed'`: Bewertung abgegeben
- `null`: Keine Bewertung erforderlich (Standard)

Aktualisiere die `createReview`-Funktion:

```
export const createReview = async (reviewData: {
  userId: string;
  authorId: string;
  taskId: string;
  taskTitle: string;
  rating: number;
  text: string;
}) => {
  try {
    const reviewsRef = collection(db, "reviews");

    // Füge die Bewertung hinzu
    const reviewDoc = await addDoc(reviewsRef, {
      ...reviewData,
      createdAt: serverTimestamp()
    });
```

```
    // Aktualisiere den Task-Status
    const taskRef = doc(db, "tasks", reviewData.taskId);
    await updateDoc(taskRef, {
      reviewStatus: 'completed',
      reviewDate: serverTimestamp()
    });

    // Aktualisiere die Benutzer-Bewertung
    const userRef = doc(db, "users", reviewData.userId);
    const userSnapshot = await getDoc(userRef);

    if (userSnapshot.exists()) {
      const userData = userSnapshot.data();
      const currentRating = userData.rating || 0;
      const currentCount = userData.ratingCount || 0;

      // Neue durchschnittliche Bewertung berechnen
      const newCount = currentCount + 1;
      const newRating = ((currentRating * currentCount) + reviewData.rating) / newCount;

      // Benutzerprofil aktualisieren
      await updateDoc(userRef, {
        rating: newRating,
        ratingCount: newCount
      });
    }

    // Benachrichtigung für den Tasker erstellen
    await createNotification(reviewData.userId, NotificationTypes.REVIEW_RECEIVED, {
      taskId: reviewData.taskId,
      taskTitle: reviewData.taskTitle,
      rating: reviewData.rating,
      text: reviewData.text.substring(0, 100) // Nur ein Ausschnitt
    });

    return reviewDoc.id;
  } catch (error) {
    console.error("Fehler beim Erstellen der Bewertung:", error);
    throw error;
  }
};
```

8. Überarbeitete ReviewDialog-Komponente

Aktualisiere die `ReviewDialog`-Komponente, um das Schließen zu verhindern:

```
// Erweitere die Props
interface ReviewDialogProps {
  // ...vorhandene Props
  preventClose?: boolean; // Verhindert das Schließen des Dialogs
}

// In der Komponente
// Wenn preventClose true ist, ignoriere den onClose-Handler
const handleDialogChange = (open: boolean) => {
  if (!open && !preventClose) {
    onClose();
  }
};

return (
  <Dialog
    open={isOpen}
    onOpenChange={handleDialogChange}
  >
    {/* ...vorhandener Dialog-Inhalt */}
  </Dialog>
);
```

9. Erweitere die Sprachdateien

Füge folgende Übersetzungen zu deinen Sprachdateien hinzu:

```
// In de.json
"notifications": {
  "title": "Benachrichtigungen",
  "all": "Alle",
  "actionRequired": "Aktionen",
  "unread": "Ungelesen",
  "empty": "Keine Benachrichtigungen vorhanden",
  "newMessage": "Neue Nachricht",
  "taskMatched": "Aufgabe zugewiesen",
  "taskCompleted": "Aufgabe abgeschlossen",
  "reviewRequired": "Bewertung erforderlich",
  "newTaskNearby": "Neue Aufgabe in deiner Nähe",
  "applicationReceived": "Neue Bewerbung",
  "reviewReceived": "Neue Bewertung",
  "newNotification": "Neue Benachrichtigung",
  "leaveReview": "Bewerten",
  "viewMessage": "Nachricht ansehen",
  "viewTask": "Aufgabe ansehen",
  "markAsRead": "Als gelesen markieren"
```

```
}

// In en.json
"notifications": {
  "title": "Notifications",
  "all": "All",
  "actionRequired": "Actions",
  "unread": "Unread",
  "empty": "No notifications",
  "newMessage": "New message",
  "taskMatched": "Task assigned",
  "taskCompleted": "Task completed",
  "reviewRequired": "Review required",
  "newTaskNearby": "New task nearby",
  "applicationReceived": "New application",
  "reviewReceived": "New review",
  "newNotification": "New notification",
  "leaveReview": "Leave a review",
  "viewMessage": "View message",
  "viewTask": "View task",
  "markAsRead": "Mark as read"
}
```

Diese umfassende Implementierung bietet dir:

1. Ein vollständiges Benachrichtigungssystem für verschiedene App-Ereignisse
2. Pflichtbewertungen, die Benutzer nicht ignorieren können
3. Priorisierung von Benachrichtigungen, die Aktionen erfordern
4. Echtzeitaktualisierungen für neue Benachrichtigungen
5. Eine vollständige UI für das Benachrichtigungszentrum

All dies wird die Benutzerengagement-Metriken deiner App deutlich verbessern und die Anzahl der abgegebenen Bewertungen maximieren.