Dolt App - Implementierungsleitfaden

Dieser Leitfaden bietet detaillierte Anweisungen zur Implementierung wichtiger Features für die Dolt App, um sie produktionsreif zu machen.

Inhaltsverzeichnis

---

1. Standort-Freigabe im Chat implementieren

1.1 Datenmodell erweitern

Öffne zuerst `lib/firebase.ts` und füge folgende Funktionen hinzu:

typescript

```typescript
/**
 * Anfrage zum Freigeben des Standorts senden
 * @param chatId Die Chat-ID
 * @param userId Die Benutzer-ID des Anfragenden
 * @returns ID der erstellten Nachricht
 */
export const requestLocationSharing = async (chatId: string, userId: string): Promise<
  try {
    // Chat überprüfen
    const chatRef = doc(db, "chats", chatId);
    const chatSnap = await getDoc(chatRef);

    if (!chatSnap.exists()) {
      throw new Error("Chat nicht gefunden");
    }

    const chatData = chatSnap.data();

    // Prüfen, ob der Benutzer ein Teilnehmer ist
    if (!chatData.participants.includes(userId)) {
      throw new Error("Unbefugter Zugriff auf diesen Chat");
    }

    // Nachricht über die Anfrage hinzufügen
    const messagesRef = collection(db, `chats/${chatId}/messages`);
    const messageDoc = await addDoc(messagesRef, {
      type: "location_request",
      senderId: userId,
      timestamp: serverTimestamp(),
      content: "Hat eine Anfrage zur Standortfreigabe gesendet"
    });

    // Chat aktualisieren
    await updateDoc(chatRef, {
      lastMessage: "Standortfreigabe angefragt",
      lastMessageAt: serverTimestamp()
    });

    return messageDoc.id;
  } catch (error) {
    console.error("Fehler beim Anfragen der Standortfreigabe:", error);
    throw error;
  }
};

/**
 * Auf Standortfreigabe-Anfrage antworten
 * @param chatId Die Chat-ID
 * @param userId Die Benutzer-ID des Antwortenden
 * @param approved Zustimmung (true) oder Ablehnung (false)
 * @param taskId Die Aufgaben-ID
 * @returns true wenn der Standort freigegeben wurde, false wenn nicht
 */
export const respondToLocationRequest = async (
  chatId: string,
  userId: string,
  approved: boolean,
  taskId: string
): Promise<boolean> => {
  try {
    // Chat und Benutzerrolle überprüfen
    const chatRef = doc(db, "chats", chatId);
    const chatSnap = await getDoc(chatRef);

    if (!chatSnap.exists()) {
      throw new Error("Chat nicht gefunden");
    }

    const chatData = chatSnap.data();
```

```
// Prüfen, ob der Benutzer ein Teilnehmer ist
if (!chatData.participants.includes(userId)) {
  throw new Error("Unbefugter Zugriff auf diesen Chat");
}

// Task abrufen
const taskRef = doc(db, "tasks", taskId);
const taskSnap = await getDoc(taskRef);

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

const taskData = taskSnap.data();

// Bestimmen, ob Benutzer der Ersteller oder Tasker ist
const isCreator = taskData.creatorId === userId;
const isTasker = taskData.taskerId === userId;

if (!isCreator && !isTasker) {
  throw new Error("Benutzer ist weder Ersteller noch Ausführender der Aufgabe");
}

// Antwort-Nachricht hinzufügen
const messagesRef = collection(db, `chats/${chatId}/messages`);
await addDoc(messagesRef, {
  type: "location_response",
  senderId: userId,
  timestamp: serverTimestamp(),
  approved: approved,
  content: approved ? "Hat der Standortfreigabe zugestimmt" : "Hat die Standortfre
});

// Lokalen chat.locationSharingStatus erstellen, falls nicht vorhanden
const locationSharingStatus = chatData.locationSharingStatus || {
  creatorApproved: false,
  taskerApproved: false,
  sharedAt: null
};

// Status aktualisieren
if (isCreator) {
  locationSharingStatus.creatorApproved = approved;
} else if (isTasker) {
  locationSharingStatus.taskerApproved = approved;
}

// Chat-Dokument aktualisieren
await updateDoc(chatRef, {
  locationSharingStatus: locationSharingStatus,
  lastMessage: approved ? "Standortfreigabe zugestimmt" : "Standortfreigabe abgele
  lastMessageAt: serverTimestamp()
});

// Wenn beide zugestimmt haben, standort freigeben
const bothApproved = locationSharingStatus.creatorApproved && locationSharingStatu

if (bothApproved && !chatData.locationSharingStatus?.sharedAt) {
  // Automatische Systembenachrichtigung erstellen
  await addDoc(messagesRef, {
    type: "location_shared",
    senderId: "system",
    timestamp: serverTimestamp(),
    taskId: taskId,
    location: taskData.location
  });

  // Standortfreigabe in Chat und Task aktualisieren
  await updateDoc(chatRef, {
    "locationSharingStatus.sharedAt": serverTimestamp(),
    lastMessage: "🗺️ Standort wurde freigegeben",
```

```
        lastMessageAt: serverTimestamp()
      });

      // Task aktualisieren
      await updateDoc(taskRef, {
        "location.locationShared": true
      });

      return true;
    }

    return false;
  } catch (error) {
    console.error("Fehler beim Beantworten der Standortfreigabe:", error);
    throw error;
  }
};

/**
 * Prüft, ob der genaue Standort für einen Chat freigegeben wurde
 */
export const isLocationSharedInChat = async (chatId: string): Promise<boolean> => {
  try {
    const chatRef = doc(db, "chats", chatId);
    const chatSnap = await getDoc(chatRef);

    if (!chatSnap.exists()) {
      return false;
    }

    const chatData = chatSnap.data();
    return !!(chatData.locationSharingStatus?.sharedAt);
  } catch (error) {
    console.error("Fehler beim Prüfen des Standortfreigabestatus:", error);
    return false;
  }
};
```

1.2 Chat-Komponente aktualisieren

Erstelle eine neue Komponente `components/chat/LocationSharingButton.tsx`:

tsx

```tsx
import React, { useState } from 'react';
import { useAuth } from '@/context/AuthContext';
import { requestLocationSharing } from '@/lib/firebase';
import { Button } from '@/components/ui/button';
import { MapPin } from 'lucide-react';
import { useToast } from '@/hooks/use-toast';

interface LocationSharingButtonProps {
  chatId: string;
  locationShared: boolean;
}

export default function LocationSharingButton({
  chatId,
  locationShared
}: LocationSharingButtonProps) {
  const { user } = useAuth();
  const { toast } = useToast();
  const [loading, setLoading] = useState(false);

  // Falls der Standort bereits freigegeben wurde, nichts anzeigen
  if (locationShared) {
    return null;
  }

  const handleRequestLocation = async () => {
    if (!user || !chatId) return;

    try {
      setLoading(true);
      await requestLocationSharing(chatId, user.uid);
      toast({
        title: "Anfrage gesendet",
        description: "Standortfreigabe wurde angefragt"
      });
    } catch (error) {
      console.error("Fehler bei der Standortanfrage:", error);
      toast({
        title: "Fehler",
        description: "Die Anfrage konnte nicht gesendet werden",
        variant: "destructive"
      });
    } finally {
      setLoading(false);
    }
  };

  return (
    <Button
      variant="outline"
      size="sm"
      onClick={handleRequestLocation}
      disabled={loading}
    >
      <MapPin className="h-4 w-4 mr-2" />
      Standort freigeben
    </Button>
  );
}
```

1.3 ChatMessage-Komponente aktualisieren

Erstelle oder aktualisiere `components/chat/ChatMessage.tsx`:

tsx

```tsx
import React, { useState } from 'react';
import { useAuth } from '@/context/AuthContext';
import { format } from 'date-fns';
import { de } from 'date-fns/locale';
import { MapPin, Check, X } from 'lucide-react';
import { Button } from '@/components/ui/button';
import { respondToLocationRequest } from '@/lib/firebase';
import { useToast } from '@/hooks/use-toast';

interface ChatMessageProps {
  message: any;
  chatId: string;
  taskId: string;
}

export default function ChatMessage({ message, chatId, taskId }: ChatMessageProps) {
  const { user } = useAuth();
  const { toast } = useToast();
  const [loading, setLoading] = useState(false);

  if (!user) return null;

  const isOwnMessage = message.senderId === user.uid;
  const isSystemMessage = message.senderId === "system";

  const handleRespondToLocation = async (approved: boolean) => {
    if (!user || !chatId || !taskId) return;

    try {
      setLoading(true);
      const shared = await respondToLocationRequest(chatId, user.uid, approved, taskId

      if (shared) {
        toast({
          title: "Standort freigegeben",
          description: "Der genaue Standort wurde für beide Parteien freigegeben"
        });
      } else {
        toast({
          title: approved ? "Zugestimmt" : "Abgelehnt",
          description: approved
            ? "Du hast der Standortfreigabe zugestimmt"
            : "Du hast die Standortfreigabe abgelehnt"
        });
      }
    } catch (error) {
      console.error("Fehler bei der Standortantwort:", error);
      toast({
        title: "Fehler",
        description: "Die Antwort konnte nicht gesendet werden",
        variant: "destructive"
      });
    } finally {
      setLoading(false);
    }
  };

  // Rendere Nachrichten basierend auf ihrem Typ
  const renderMessageContent = () => {
    switch (message.type) {
      case "location_request":
        return (
          <div className="bg-yellow-50 rounded-lg p-3 my-2 mx-12 text-center">
            <p className="text-sm text-yellow-700">
              {isOwnMessage
                ? "Du hast eine Standortfreigabe angefragt"
                : "Der andere Nutzer möchte den genauen Standort freigeben"}
            </p>

            {!isOwnMessage && (
```

```
          <div className="flex justify-center space-x-2 mt-2">
            <Button
              size="sm"
              variant="outline"
              className="bg-green-50 hover:bg-green-100 text-green-600"
              onClick={() => handleRespondToLocation(true)}
              disabled={loading}
            >
              <Check className="h-4 w-4 mr-1" />
              Zustimmen
            </Button>
            <Button
              size="sm"
              variant="outline"
              className="bg-red-50 hover:bg-red-100 text-red-600"
              onClick={() => handleRespondToLocation(false)}
              disabled={loading}
            >
              <X className="h-4 w-4 mr-1" />
              Ablehnen
            </Button>
          </div>
        )}
      </div>
    );

  case "location_response":
    return (
      <div className={`bg-gray-50 rounded-lg p-3 my-2 mx-12 text-center`}>
        <p className="text-sm text-gray-700">
          {isOwnMessage
            ? `Du hast ${message.approved ? 'zugestimmt' : 'abgelehnt'}`
            : `Der andere Nutzer hat ${message.approved ? 'zugestimmt' : 'abgelehn
        </p>
      </div>
    );

  case "location_shared":
    return (
      <div className="bg-green-50 rounded-lg p-3 my-2 text-center">
        <p className="text-sm text-green-700 font-medium mb-2">
          🎉 Der genaue Standort wurde freigegeben
        </p>

        <div className="border rounded-lg overflow-hidden">
          <div className="p-3 bg-white">
            <p className="font-medium text-gray-800">{message.location.address}</p
            <div className="mt-2">
              <a
                href={`https://maps.google.com/?q=${message.location.coordinates.l
                target="_blank"
                rel="noopener noreferrer"
                className="text-blue-600 hover:text-blue-800 text-sm flex items-ce
              >
                <MapPin className="h-4 w-4 mr-1" />
                In Google Maps öffnen
              </a>
            </div>
          </div>
        </div>
      </div>
    );

  default:
    // Normale Textnachricht
    return (
      <div className={`${
        isSystemMessage
          ? 'bg-gray-100 mx-12 text-center'
          : isOwnMessage
            ? 'bg-indigo-100 ml-12'
```

```
                : 'bg-gray-100 mr-12'
            } rounded-lg p-3 my-2`}>
              <p className="text-gray-800">{message.content}</p>
              <p className="text-xs text-gray-500 text-right mt-1">
                {message.timestamp ?
                  format(new Date(message.timestamp.seconds * 1000), 'HH:mm', { locale:
                  : ''}
              </p>
            </div>
          );
      }
    };

    return renderMessageContent();
}
```

1.4 ChatView aktualisieren

Aktualisiere deine `components/chat/ChatView.tsx` oder erstelle sie neu:

tsx

```tsx
import React, { useState, useEffect, useRef } from 'react';
import { useAuth } from '@/context/AuthContext';
import { getMessages, sendMessage, isLocationSharedInChat } from '@/lib/firebase';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Send, ArrowLeft } from 'lucide-react';
import { useToast } from '@/hooks/use-toast';
import ChatMessage from './ChatMessage';
import LocationSharingButton from './LocationSharingButton';

interface ChatViewProps {
  chatId: string;
  taskId: string;
  taskData: any;
  onBack?: () => void;
}

export default function ChatView({
  chatId,
  taskId,
  taskData,
  onBack
}: ChatViewProps) {
  const { user } = useAuth();
  const { toast } = useToast();
  const [messages, setMessages] = useState([]);
  const [newMessage, setNewMessage] = useState('');
  const [sending, setSending] = useState(false);
  const [locationShared, setLocationShared] = useState(false);
  const messagesEndRef = useRef(null);

  // Nachrichten laden
  useEffect(() => {
    if (!chatId) return;

    // Prüfen, ob der Standort bereits freigegeben wurde
    const checkLocationStatus = async () => {
      try {
        const shared = await isLocationSharedInChat(chatId);
        setLocationShared(shared);
      } catch (error) {
        console.error("Fehler beim Prüfen des Standortstatus:", error);
      }
    };

    checkLocationStatus();

    // Nachrichten abonnieren
    const unsubscribe = getMessages(chatId, (newMessages) => {
      setMessages(newMessages);
    });

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

  // Automatisches Scrollen
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  const handleSendMessage = async (e) => {
    e.preventDefault();

    if (!newMessage.trim() || !user || !chatId) return;

    try {
      setSending(true);
      await sendMessage(chatId, user.uid, newMessage.trim());
      setNewMessage('');
    } catch (error) {
```

```
    console.error("Fehler beim Senden der Nachricht:", error);
    toast({
      title: "Fehler",
      description: "Die Nachricht konnte nicht gesendet werden",
      variant: "destructive"
    });
  } finally {
    setSending(false);
  }
};

return (
  <div className="flex flex-col h-full">
    {/* Chat-Header */}
    <div className="bg-white border-b p-4 flex justify-between items-center">
      <div className="flex items-center">
        {onBack && (
          <Button variant="ghost" size="icon" onClick={onBack} className="mr-2">
            <ArrowLeft className="h-5 w-5" />
          </Button>
        )}
        <div>
          <h2 className="font-bold">{taskData?.title || 'Chat'}</h2>
          <p className="text-sm text-gray-500">
            {locationShared
              ? 'Standort: Freigegeben'
              : `Standort: ${taskData?.location?.city || 'Nicht verfügbar'}`}
          </p>
        </div>
      </div>

      <LocationSharingButton
        chatId={chatId}
        locationShared={locationShared}
      />
    </div>

    {/* Nachrichtenliste */}
    <div className="flex-1 overflow-y-auto p-4 bg-gray-50">
      {messages.length === 0 ? (
        <div className="text-center py-8 text-gray-500">
          Noch keine Nachrichten. Schreibe etwas, um die Unterhaltung zu beginnen.
        </div>
      ) : (
        messages.map((msg) => (
          <ChatMessage
            key={msg.id}
            message={msg}
            chatId={chatId}
            taskId={taskId}
          />
        ))
      )}
      <div ref={messagesEndRef} />
    </div>

    {/* Nachrichteneingabe */}
    <form onSubmit={handleSendMessage} className="p-4 bg-white border-t flex gap-2">
      <Input
        value={newMessage}
        onChange={(e) => setNewMessage(e.target.value)}
        placeholder="Nachricht schreiben..."
        className="flex-1"
        disabled={sending}
      />
      <Button type="submit" disabled={!newMessage.trim() || sending}>
        <Send className="h-4 w-4" />
      </Button>
    </form>
  </div>
```

```
  );
}
```

1.5 TaskDetail aktualisieren

Aktualisiere die Standortanzeige in `components/tasks/TaskDetail.tsx`:

```tsx
// Standortanzeige basierend auf dem Freigabestatus
const renderLocation = () => {
  const userIsParticipant =
    task.creatorId === user?.uid ||
    task.taskerId === user?.uid;

  if (task.location?.locationShared && userIsParticipant) {
    // Vollständiger Standort für Teilnehmer, wenn freigegeben
    return (
      <div className="border rounded-lg p-3 bg-white shadow-sm">
        <div className="flex items-start">
          <MapPin className="h-5 w-5 text-indigo-600 mt-0.5 mr-2 flex-shrink-0" />
          <div>
            <p className="font-medium">{task.location.address}</p>
            <a
              href={`https://maps.google.com/?q=${task.location.coordinates.lat},${tas
              target="_blank"
              rel="noopener noreferrer"
              className="text-blue-600 hover:text-blue-800 text-sm inline-flex items-c
            >
              In Google Maps öffnen
            </a>
          </div>
        </div>
      </div>
    );
  } else {
    // Nur Stadt und Entfernung für alle anderen
    return (
      <div className="flex items-center">
        <MapPin className="h-5 w-5 text-indigo-600 mr-2 flex-shrink-0" />
        <div>
          <p className="font-medium">{task.location?.city || 'Standort nicht verfügbar
          {calculateDistance && (
            <p className="text-sm text-gray-500">
              {calculateDistance(task.location?.coordinates) || 'Entfernung unbekannt'
            </p>
          )}
          {userIsParticipant && !task.location?.locationShared && (
            <p className="text-xs text-gray-500 mt-1">
              Der genaue Standort wird nach gegenseitiger Zustimmung im Chat freigegeb
            </p>
          )}
        </div>
      </div>
    );
  }
};
```

2. Bewertungssystem vervollständigen

2.1 Bewertungsdialog erstellen

Erstelle eine neue Komponente `components/reviews/ReviewDialog.tsx`:

tsx

```tsx
import React, { useState } from 'react';
import { useAuth } from '@/context/AuthContext';
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogFooter } from '@/compo
import { Button } from '@/components/ui/button';
import { Textarea } from '@/components/ui/textarea';
import { Star } from 'lucide-react';
import { useToast } from '@/hooks/use-toast';
import { createReview, completeTask } from '@/lib/firebase';

interface ReviewDialogProps {
  isOpen: boolean;
  onClose: () => void;
  taskId: string;
  userId: string;
  taskTitle: string;
}

export default function ReviewDialog({
  isOpen,
  onClose,
  taskId,
  userId,
  taskTitle
}: ReviewDialogProps) {
  const { user } = useAuth();
  const { toast } = useToast();
  const [rating, setRating] = useState(0);
  const [hoverRating, setHoverRating] = useState(0);
  const [review, setReview] = useState('');
  const [submitting, setSubmitting] = useState(false);

  const handleSubmit = async () => {
    if (!rating) {
      toast({
        title: "Bewertung erforderlich",
        description: "Bitte gib eine Sternebewertung ab",
        variant: "destructive"
      });
      return;
    }

    if (!user) return;

    try {
      setSubmitting(true);

      const reviewData = {
        userId: userId,
        authorId: user.uid,
        taskId: taskId,
        taskTitle: taskTitle,
        rating: rating,
        text: review.trim()
      };

      // Bewertung erstellen
      await createReview(reviewData);

      // Task als abgeschlossen markieren
      await completeTask(taskId, rating, review.trim());

      toast({
        title: "Bewertung abgesendet",
        description: "Vielen Dank für deine Bewertung!"
      });

      onClose();
    } catch (error) {
      console.error("Fehler beim Erstellen der Bewertung:", error);
      toast({
```

```
        title: "Fehler",
        description: "Deine Bewertung konnte nicht gespeichert werden",
        variant: "destructive"
      });
    } finally {
      setSubmitting(false);
    }
  };

  return (
    <Dialog open={isOpen} onOpenChange={onClose}>
      <DialogContent className="sm:max-w-[425px]">
        <DialogHeader>
          <DialogTitle>Aufgabe bewerten</DialogTitle>
        </DialogHeader>

        <div className="py-4 space-y-4">
          {/* Sternebewertung */}
          <div>
            <p className="text-sm font-medium mb-2">Wie zufrieden bist du?</p>
            <div className="flex justify-center">
              {[1, 2, 3, 4, 5].map((star) => (
                <button
                  key={star}
                  type="button"
                  className="p-1"
                  onClick={() => setRating(star)}
                  onMouseEnter={() => setHoverRating(star)}
                  onMouseLeave={() => setHoverRating(0)}
                >
                  <Star
                    className={`w-8 h-8 ${
                      (hoverRating || rating) >= star
                        ? 'text-yellow-400 fill-yellow-400'
                        : 'text-gray-300'
                    }`}
                  />
                </button>
              ))}
            </div>
          </div>

          {/* Bewertungstext */}
          <div>
            <p className="text-sm font-medium mb-2">Deine Erfahrung (optional)</p>
            <Textarea
              value={review}
              onChange={(e) => setReview(e.target.value)}
              placeholder="Beschreibe deine Erfahrung..."
              className="h-32"
            />
          </div>
        </div>

        <DialogFooter>
          <Button variant="outline" onClick={onClose} disabled={submitting}>
            Abbrechen
          </Button>
          <Button onClick={handleSubmit} disabled={!rating || submitting}>
            {submitting ? 'Wird gesendet...' : 'Bewertung senden'}
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  );
}
```

2.2 Funktion zum Abschließen einer Aufgabe aktualisieren

Aktualisiere die `completeTask`-Funktion in `lib/firebase.ts`:

```typescript
/**
 * Schließt eine Aufgabe ab und fügt eine Bewertung hinzu
 */
export const completeTask = async (taskId: string, rating: number, review: 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()
    });

    // Tasker-Status aktualisieren (completedTasks und Rating)
    const userRef = doc(db, "users", taskData.taskerId);
    await updateDoc(userRef, {
      completedTasks: increment(1)
    });

    // Benachrichtigung erstellen
    const notificationsRef = collection(db, "notifications");
    await addDoc(notificationsRef, {
      userId: taskData.taskerId,
      type: "task_completed",
      taskId: taskId,
      taskTitle: taskData.title,
      read: false,
      createdAt: serverTimestamp()
    });

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

2.3 TaskDetail aktualisieren für Aufgabenabschluss

Füge in `components/tasks/TaskDetail.tsx` einen Abschnitt für die Aufgabenabschluss-Funktionalität hinzu:

```tsx
// Zustand für den Bewertungsdialog
const [isReviewDialogOpen, setIsReviewDialogOpen] = useState(false);

// In deinem JSX, wenn der Benutzer der Aufgabenersteller ist und die Aufgabe zugewies
{user?.uid === task.creatorId && task.status === "matched" && (
  <div className="mt-4">
    <Button
      className="w-full"
      onClick={() => setIsReviewDialogOpen(true)}
    >
      <Check className="mr-2 h-4 w-4" />
      Aufgabe abschließen & bewerten
    </Button>
  </div>
)}

{/* Bewertungsdialog */}
<ReviewDialog
  isOpen={isReviewDialogOpen}
  onClose={() => setIsReviewDialogOpen(false)}
  taskId={task.id}
  userId={task.taskerId}
  taskTitle={task.title}
/>
```

2.4 Bewertungsanzeige in Profilen verbessern

Erstelle eine neue Komponente `components/reviews/ReviewsList.tsx`:

tsx

```tsx
import React, { useEffect, useState } from 'react';
import { getUserReviews } from '@/lib/firebase';
import { format } from 'date-fns';
import { de } from 'date-fns/locale';
import { Star } from 'lucide-react';
import { Skeleton } from '@/components/ui/skeleton';

interface ReviewsListProps {
  userId: string;
}

export default function ReviewsList({ userId }: ReviewsListProps) {
  const [reviews, setReviews] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchReviews = async () => {
      try {
        setLoading(true);
        const fetchedReviews = await getUserReviews(userId);
        setReviews(fetchedReviews);
      } catch (error) {
        console.error("Fehler beim Laden der Bewertungen:", error);
      } finally {
        setLoading(false);
      }
    };

    if (userId) {
      fetchReviews();
    }
  }, [userId]);

  if (loading) {
    return (
      <div className="space-y-4">
        {[1, 2, 3].map((index) => (
          <div key={index} className="p-4 border rounded-lg">
            <div className="flex items-center gap-2 mb-2">
              <Skeleton className="h-5 w-24" />
              <Skeleton className="h-4 w-32" />
            </div>
            <Skeleton className="h-4 w-full mb-1" />
            <Skeleton className="h-4 w-3/4" />
          </div>
        ))}
      </div>
    );
  }

  if (reviews.length === 0) {
    return (
      <div className="text-center py-6 text-gray-500">
        Noch keine Bewertungen vorhanden.
      </div>
    );
  }

  return (
    <div className="space-y-4">
      {reviews.map((review) => (
        <div key={review.id} className="p-4 border rounded-lg">
          <div className="flex items-center justify-between mb-2">
            <div className="flex">
              {[1, 2, 3, 4, 5].map((star) => (
                <Star
                  key={star}
                  className={`w-4 h-4 ${
                    star <= review.rating
                      ? 'text-yellow-400 fill-yellow-400'
```

```
                    : 'text-gray-300'
                }`}
              />
            ))}
          </div>
          <span className="text-sm text-gray-500">
            {review.createdAt ?
              format(
                new Date(review.createdAt.seconds * 1000),
                'dd. MMMM yyyy',
                { locale: de }
              ) : ''}
          </span>
        </div>

        {review.text && (
          <p className="text-gray-700">{review.text}</p>
        )}

        <div className="mt-2 text-sm text-gray-500">
          <span>Für: {review.taskTitle || 'Aufgabe'}</span>
        </div>
      </div>
    ))}
  </div>
 );
}
```

2.5 Profilseite aktualisieren

Aktualisiere die Profilseite, um die Bewertungsliste zu integrieren:

tsx

```tsx
// In components/profile/UserProfile.tsx oder pages/profile/[id].tsx
import ReviewsList from '@/components/reviews/ReviewsList';

// In deinem JSX, wo die Tabs sind:
<Tabs defaultValue="reviews" className="w-full">
  <TabsList className="grid grid-cols-2 bg-gray-100">
    <TabsTrigger value="reviews">
      <Star className="w-4 h-4 mr-2" />
      Bewertungen
    </TabsTrigger>
    <TabsTrigger value="tasks">
      <ClipboardList className="w-4 h-4 mr-2" />
      Aufgaben
    </TabsTrigger>
  </TabsList>

  <TabsContent value="reviews" className="p-4">
    <ReviewsList userId={userId} />
  </TabsContent>

  <TabsContent value="tasks" className="p-4">
    {/* Bestehender Code für Aufgabenliste */}
  </TabsContent>
</Tabs>
```

3. Aufgabenfilterung und -sortierung hinzufügen

3.1 Erstelle eine FilterBar-Komponente

Erstelle eine neue Komponente `components/tasks/TaskFilterBar.tsx`:

tsx

```tsx
import React, { useState } from 'react';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue
} from '@/components/ui/select';
import {
  Popover,
  PopoverContent,
  PopoverTrigger
} from '@/components/ui/popover';
import { Check, ChevronDown, Filter, MapPin, SortAsc, X } from 'lucide-react';
import { Slider } from '@/components/ui/slider';

export type TaskFilters = {
  category?: string;
  query?: string;
  maxDistance?: number;
  sortBy?: 'date' | 'distance' | 'price';
  sortDirection?: 'asc' | 'desc';
};

interface TaskFilterBarProps {
  filters: TaskFilters;
  onFiltersChange: (filters: TaskFilters) => void;
  userLocation?: { lat: number; lng: number } | null;
}

export default function TaskFilterBar({
  filters,
  onFiltersChange,
  userLocation
}: TaskFilterBarProps) {
  // Lokale Zustände für Filter-Popover
  const [isFilterOpen, setIsFilterOpen] = useState(false);
  const [tempFilters, setTempFilters] = useState<TaskFilters>(filters);

  // Kategorien
  const categories = [
    { id: 'all', label: 'Alle Kategorien' },
    { id: 'cleaning', label: 'Reinigung' },
    { id: 'moving', label: 'Umzug' },
    { id: 'delivery', label: 'Lieferung' },
    { id: 'assembly', label: 'Montage' },
    { id: 'gardening', label: 'Gartenarbeit' },
    { id: 'tutoring', label: 'Nachhilfe' },
    { id: 'other', label: 'Sonstiges' }
  ];

  // Sort-Optionen
  const sortOptions = [
    { value: 'date-desc', label: 'Neueste zuerst' },
    { value: 'date-asc', label: 'Älteste zuerst' },
    { value: 'price-desc', label: 'Höchster Preis' },
    { value: 'price-asc', label: 'Niedrigster Preis' },
    { value: 'distance-asc', label: 'Nächste zuerst', disabled: !userLocation }
  ];

  // Aktualisiere die temporären Filter
  const updateTempFilters = (key: string, value: any) => {
    setTempFilters(prev => ({ ...prev, [key]: value }));
  };

  // Filter anwenden
  const applyFilters = () => {
    onFiltersChange(tempFilters);
```

```
  setIsFilterOpen(false);
};

// Filter zurücksetzen
const resetFilters = () => {
  const newFilters = {
    query: '',
    category: 'all',
    maxDistance: 50,
    sortBy: 'date',
    sortDirection: 'desc'
  };
  setTempFilters(newFilters);
  onFiltersChange(newFilters);
  setIsFilterOpen(false);
};

// Sortierung ändern
const handleSortChange = (sortValue: string) => {
  const [sortBy, sortDirection] = sortValue.split('-') as [
    'date' | 'distance' | 'price',
    'asc' | 'desc'
  ];
  onFiltersChange({ ...filters, sortBy, sortDirection });
};

// Aktuelle Sortierung bestimmen
const currentSort = `${filters.sortBy || 'date'}-${filters.sortDirection || 'desc'}`

// Formatiert die Entfernung für die Anzeige
const formatDistance = (distance: number) => {
  if (distance >= 1) {
    return `${distance} km`;
  }
  return `${distance * 1000} m`;
};

return (
  <div className="bg-white p-4 rounded-lg shadow-sm mb-4 space-y-4">
    {/* Suchleiste */}
    <div className="flex gap-2">
      <Input
        placeholder="Aufgaben suchen..."
        value={filters.query || ''}
        onChange={(e) => onFiltersChange({ ...filters, query: e.target.value })}
        className="flex-1"
      />

      {/* Filter-Button */}
      <Popover open={isFilterOpen} onOpenChange={setIsFilterOpen}>
        <PopoverTrigger asChild>
          <Button variant="outline" size="icon">
            <Filter className="h-4 w-4" />
          </Button>
        </PopoverTrigger>
        <PopoverContent className="w-80">
          <div className="space-y-4">
            <h3 className="font-medium">Filter</h3>

            {/* Kategorie */}
            <div>
              <label className="text-sm font-medium">Kategorie</label>
              <Select
                value={tempFilters.category || 'all'}
                onValueChange={(value) => updateTempFilters('category', value)}
              >
                <SelectTrigger>
                  <SelectValue placeholder="Kategorie wählen" />
                </SelectTrigger>
                <SelectContent>
                  {categories.map((category) => (
```

```
                  <SelectItem key={category.id} value={category.id}>
                    {category.label}
                  </SelectItem>
                ))}
              </SelectContent>
            </Select>
          </div>

          {/* Entfernung (nur wenn Standort verfügbar) */}
          {userLocation && (
            <div>
              <div className="flex justify-between">
                <label className="text-sm font-medium">Maximale Entfernung</label>
                <span className="text-sm text-gray-500">
                  {formatDistance(tempFilters.maxDistance || 50)}
                </span>
              </div>
              <Slider
                value={[tempFilters.maxDistance || 50]}
                min={1}
                max={100}
                step={1}
                onValueChange={(value) => updateTempFilters('maxDistance', value[0
                className="mt-2"
              />
            </div>
          )}

          {/* Filter-Aktionen */}
          <div className="flex justify-between gap-2">
            <Button
              variant="outline"
              size="sm"
              onClick={resetFilters}
            >
              Zurücksetzen
            </Button>
            <Button
              size="sm"
              onClick={applyFilters}
            >
              Filter anwenden
            </Button>
          </div>
        </div>
      </PopoverContent>
    </Popover>

    {/* Sortierung */}
    <Select value={currentSort} onValueChange={handleSortChange}>
      <SelectTrigger className="w-[180px]">
        <SortAsc className="h-4 w-4 mr-2" />
        <SelectValue />
      </SelectTrigger>
      <SelectContent>
        {sortOptions.map((option) => (
          <SelectItem
            key={option.value}
            value={option.value}
            disabled={option.disabled}
          >
            {option.label}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
  </div>

  {/* Aktive Filter anzeigen */}
  {(filters.category && filters.category !== 'all') || filters.maxDistance ? (
    <div className="flex flex-wrap gap-2">
```

```
            {/* Kategorie-Badge */}
            {filters.category && filters.category !== 'all' && (
              <span className="inline-flex items-center px-2 py-1 rounded-full bg-indigo
                {categories.find(c => c.id === filters.category)?.label}
                <button
                  onClick={() => onFiltersChange({ ...filters, category: 'all' })}
                  className="ml-1 p-0.5 rounded-full bg-indigo-200 hover:bg-indigo-300"
                >
                  <X className="h-3 w-3" />
                </button>
              </span>
            )}

            {/* Entfernungs-Badge */}
            {filters.maxDistance && (
              <span className="inline-flex items-center px-2 py-1 rounded-full bg-indigo
                <MapPin className="h-3 w-3 mr-1" />
                Max. {formatDistance(filters.maxDistance)}
                <button
                  onClick={() => onFiltersChange({ ...filters, maxDistance: undefined })
                  className="ml-1 p-0.5 rounded-full bg-indigo-200 hover:bg-indigo-300"
                >
                  <X className="h-3 w-3" />
                </button>
              </span>
            )}
          </div>
        ) : null}
      </div>
    );
  }
```

3.2 Firebase-Funktionen für gefilterte und sortierte Abfragen aktualisieren

Aktualisiere die `getTasks`-Funktion in `lib/firebase.ts`:

typescript

```
/**
 * Aufgaben mit Filtern und Sortierung abrufen
 * @param filters Filter und Sortieroptionen
 * @param userLocation Standort des Benutzers für Entfernungsberechnung
 */
export const getTasks = async (
  filters: {
    creatorId?: string;
    status?: string;
    category?: string;
    query?: string;
    sortBy?: 'date' | 'distance' | 'price';
    sortDirection?: 'asc' | 'desc';
  } = {},
  userLocation?: { lat: number; lng: number } | null,
  limit = 50
) => {
  const tasksRef = collection(db, "tasks");

  // Basis-Query erstellen
  let taskQuery = query(tasksRef);

  // Filters anwenden
  // Status-Filter (falls kein creatorId, nur offene Tasks)
  if (filters.status) {
    taskQuery = query(taskQuery, where("status", "==", filters.status));
  } else if (!filters.creatorId) {
    taskQuery = query(taskQuery, where("status", "==", "open"));
  }

  // Wenn nach Ersteller gefiltert wird
  if (filters.creatorId) {
    taskQuery = query(taskQuery, where("creatorId", "==", filters.creatorId));
  }

  // Wenn nach Kategorie gefiltert wird (außer 'all')
  if (filters.category && filters.category !== 'all') {
    taskQuery = query(taskQuery, where("category", "==", filters.category));
  }

  // Sortierung anwenden
  if (filters.sortBy === 'date') {
    taskQuery = query(
      taskQuery,
      orderBy("createdAt", filters.sortDirection === 'asc' ? 'asc' : 'desc')
    );
  } else if (filters.sortBy === 'price') {
    taskQuery = query(
      taskQuery,
      orderBy("price", filters.sortDirection === 'asc' ? 'asc' : 'desc')
    );
  } else {
    // Standardsortierung nach Datum, falls keine Sortierung angegeben
    taskQuery = query(taskQuery, orderBy("createdAt", "desc"));
  }

  // Limit anwenden
  taskQuery = query(taskQuery, limit(limit));

  try {
    // Abfrage ausführen
    const taskSnapshot = await getDocs(taskQuery);

    // Aufgaben verarbeiten
    let tasks = taskSnapshot.docs.map(doc => {
      const data = doc.data();

      return {
        id: doc.id,
        ...data,
```

```
    // Sicherstellen, dass imageUrls immer ein Array ist
    imageUrls: Array.isArray(data.imageUrls) ? data.imageUrls : [],
    // Für Abwärtskompatibilität
    imageUrl: data.imageUrl || (Array.isArray(data.imageUrls) && data.imageUrls.le
  };
});

// Textsuche in-memory durchführen (falls query vorhanden)
if (filters.query) {
  const searchTerms = filters.query.toLowerCase().split(' ');
  tasks = tasks.filter(task => {
    const titleLower = (task.title || '').toLowerCase();
    const descriptionLower = (task.description || '').toLowerCase();
    const addressLower = (task.location?.address || '').toLowerCase();

    return searchTerms.every(term =>
      titleLower.includes(term) ||
      descriptionLower.includes(term) ||
      addressLower.includes(term)
    );
  });
}

// Nach Entfernung sortieren (falls gewünscht und Standort verfügbar)
if (filters.sortBy === 'distance' && userLocation) {
  // Entfernung für jede Aufgabe berechnen
  tasks.forEach(task => {
    if (task.location?.coordinates) {
      task.distance = calculateDistance(
        userLocation.lat,
        userLocation.lng,
        task.location.coordinates.lat,
        task.location.coordinates.lng
      );
    } else {
      task.distance = Number.MAX_SAFE_INTEGER; // Aufgaben ohne Standort am Ende
    }
  });

  // Nach Entfernung sortieren
  tasks.sort((a, b) => {
    if (filters.sortDirection === 'desc') {
      return (b.distance || 0) - (a.distance || 0);
    }
    return (a.distance || 0) - (b.distance || 0);
  });
}

// Benutzerprofile für Ersteller in Batch abrufen
const uniqueCreatorIds = Array.from(new Set(tasks.map(task => task.creatorId)));
const creatorProfiles = {};

if (uniqueCreatorIds.length > 0) {
  const profilePromises = uniqueCreatorIds.map(id => getDoc(doc(db, "users", id)))
  const profileDocs = await Promise.all(profilePromises);

  profileDocs.forEach(doc => {
    if (doc.exists()) {
      creatorProfiles[doc.id] = doc.data();
    }
  });
}

// Ersteller-Infos zu den Aufgaben hinzufügen
return tasks.map(task => {
  const creatorProfile = creatorProfiles[task.creatorId];

  return {
    ...task,
    creatorName: creatorProfile?.displayName || 'Unbekannter Benutzer',
    creatorPhotoURL: creatorProfile?.photoURL || '',
```

```
        creatorRating: creatorProfile?.rating || 0
      };
    });
  } catch (error) {
    console.error("Fehler beim Abrufen der Aufgaben:", error);
    return [];
  }
};

/**
 * Entfernung zwischen zwei Koordinaten berechnen (Haversine-Formel)
 */
export const calculateDistance = (lat1: number, lng1: number, lat2: number, lng2: numb
  const R = 6371; // Erdradius in Kilometern
  const dLat = (lat2 - lat1) * Math.PI / 180;
  const dLng = (lng2 - lng1) * Math.PI / 180;

  const a =
    Math.sin(dLat/2) * Math.sin(dLat/2) +
    Math.cos(lat1 * Math.PI / 180) * Math.cos(lat2 * Math.PI / 180) *
    Math.sin(dLng/2) * Math.sin(dLng/2);

  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  const distance = R * c; // Entfernung in Kilometern

  return Math.round(distance * 10) / 10; // Auf eine Nachkommastelle runden
};
```

3.3 TasksPage aktualisieren

Aktualisiere deine `pages/TasksPage.tsx` oder erstelle sie, falls sie noch nicht existiert:

tsx

```tsx
import React, { useState, useEffect } from 'react';
import { useAuth } from '@/context/AuthContext';
import { getTasks } from '@/lib/firebase';
import TaskCard from '@/components/tasks/TaskCard';
import TaskFilterBar, { TaskFilters } from '@/components/tasks/TaskFilterBar';
import { Button } from '@/components/ui/button';
import { PlusCircle } from 'lucide-react';
import { useNavigate } from 'react-router-dom';
import { Skeleton } from '@/components/ui/skeleton';

export default function TasksPage() {
  const { user } = useAuth();
  const navigate = useNavigate();
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [userLocation, setUserLocation] = useState(null);
  const [filters, setFilters] = useState<TaskFilters>({
    category: 'all',
    query: '',
    sortBy: 'date',
    sortDirection: 'desc'
  });

  // Benutzerlocation abrufen
  useEffect(() => {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(
        (position) => {
          setUserLocation({
            lat: position.coords.latitude,
            lng: position.coords.longitude
          });
        },
        (error) => {
          console.error("Fehler beim Abrufen des Standorts:", error);
        }
      );
    }
  }, []);

  // Aufgaben laden
  useEffect(() => {
    const loadTasks = async () => {
      try {
        setLoading(true);
        const loadedTasks = await getTasks(filters, userLocation);
        setTasks(loadedTasks);
      } catch (error) {
        console.error("Fehler beim Laden der Aufgaben:", error);
      } finally {
        setLoading(false);
      }
    };

    loadTasks();
  }, [filters, userLocation]);

  // Filter aktualisieren
  const handleFiltersChange = (newFilters: TaskFilters) => {
    setFilters(newFilters);
  };

  return (
    <div className="container mx-auto p-4 max-w-3xl">
      <div className="flex justify-between items-center mb-4">
        <h1 className="text-2xl font-bold">Aufgaben entdecken</h1>

        {user && (
          <Button onClick={() => navigate('/tasks/create')}>
            <PlusCircle className="h-4 w-4 mr-2" />
```

```
            Aufgabe erstellen
          </Button>
        )}
      </div>

      {/* Filter-Leiste */}
      <TaskFilterBar
        filters={filters}
        onFiltersChange={handleFiltersChange}
        userLocation={userLocation}
      />

      {/* Aufgabenliste */}
      <div className="space-y-4">
        {loading ? (
          // Lade-Animation
          Array.from({ length: 5 }).map((_, index) => (
            <div key={index} className="bg-white rounded-lg p-4 shadow-sm">
              <div className="flex gap-4">
                <Skeleton className="h-24 w-24 rounded-md" />
                <div className="flex-1 space-y-2">
                  <Skeleton className="h-6 w-3/4" />
                  <Skeleton className="h-4 w-1/2" />
                  <Skeleton className="h-4 w-1/4" />
                </div>
              </div>
            </div>
          ))
        ) : tasks.length > 0 ? (
          tasks.map(task => (
            <TaskCard
              key={task.id}
              task={task}
              userLocation={userLocation}
              onClick={() => navigate(`/tasks/${task.id}`)}
            />
          ))
        ) : (
          <div className="bg-white rounded-lg p-8 text-center shadow-sm">
            <p className="text-gray-500">Keine Aufgaben gefunden.</p>
            {Object.keys(filters).some(key => filters[key] && key !== 'sortBy' && key
              <p className="mt-2 text-sm text-gray-500">
                Versuche es mit anderen Filtereinstellungen.
              </p>
            )}
          </div>
        )}
      </div>
    </div>
  );
}
```

4. Performance-Optimierung durch Code-Splitting

4.1 Router mit Code-Splitting einrichten

Erstelle oder aktualisiere deine `App.tsx` oder Hauptrouterdatei:

tsx

```tsx
import React, { Suspense, lazy } from 'react';
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider } from '@/context/AuthContext';
import { Toaster } from '@/components/ui/toaster';
import Layout from '@/components/layout/Layout';
import LoadingScreen from '@/components/ui/LoadingScreen';

// Lazy-loaded Komponenten
const HomePage = lazy(() => import('@/pages/HomePage'));
const TasksPage = lazy(() => import('@/pages/TasksPage'));
const TaskDetailPage = lazy(() => import('@/pages/TaskDetailPage'));
const CreateTaskPage = lazy(() => import('@/pages/CreateTaskPage'));
const ProfilePage = lazy(() => import('@/pages/ProfilePage'));
const ChatPage = lazy(() => import('@/pages/ChatPage'));
const LoginPage = lazy(() => import('@/pages/LoginPage'));
const RegisterPage = lazy(() => import('@/pages/RegisterPage'));
const MyTasksPage = lazy(() => import('@/pages/MyTasksPage'));
const NotificationsPage = lazy(() => import('@/pages/NotificationsPage'));

// Protected Route Komponente
const ProtectedRoute = ({ children }) => {
  const { user, loading } = useAuth();

  if (loading) return <LoadingScreen />;
  if (!user) return <Navigate to="/login" />;

  return children;
};

export default function App() {
  return (
    <BrowserRouter>
      <AuthProvider>
        <Suspense fallback={<LoadingScreen />}>
          <Layout>
            <Routes>
              <Route path="/" element={<HomePage />} />
              <Route path="/tasks" element={<TasksPage />} />
              <Route path="/tasks/:id" element={<TaskDetailPage />} />
              <Route path="/login" element={<LoginPage />} />
              <Route path="/register" element={<RegisterPage />} />

              {/* Geschützte Routen */}
              <Route path="/tasks/create" element={
                <ProtectedRoute>
                  <CreateTaskPage />
                </ProtectedRoute>
              } />
              <Route path="/profile/:id" element={<ProfilePage />} />
              <Route path="/profile" element={
                <ProtectedRoute>
                  <ProfilePage />
                </ProtectedRoute>
              } />
              <Route path="/my-tasks" element={
                <ProtectedRoute>
                  <MyTasksPage />
                </ProtectedRoute>
              } />
              <Route path="/chats/:id" element={
                <ProtectedRoute>
                  <ChatPage />
                </ProtectedRoute>
              } />
              <Route path="/notifications" element={
                <ProtectedRoute>
                  <NotificationsPage />
                </ProtectedRoute>
              } />
```

```
              {/* Fallback für nicht existierende Routen */}
              <Route path="*" element={<Navigate to="/" replace />} />
            </Routes>
          </Layout>
        </Suspense>
        <Toaster />
      </AuthProvider>
    </BrowserRouter>
  );
}
```

4.2 LoadingScreen-Komponente erstellen

Erstelle eine LoadingScreen-Komponente unter `components/ui/LoadingScreen.tsx`:

```tsx
import React from 'react';

export default function LoadingScreen() {
  return (
    <div className="flex flex-col items-center justify-center min-h-[50vh] p-4">
      <div className="w-12 h-12 rounded-full border-4 border-indigo-200 border-t-indig
      <p className="text-gray-500">Wird geladen...</p>
    </div>
  );
}
```

4.3 Lazy-Loading für Bilder implementieren

Erstelle eine LazyImage-Komponente unter `components/ui/LazyImage.tsx`:

tsx

```tsx
import React, { useState, useEffect } from 'react';
import { Skeleton } from '@/components/ui/skeleton';

interface LazyImageProps {
  src: string;
  alt: string;
  className?: string;
  fallbackSrc?: string;
  width?: number | string;
  height?: number | string;
}

export default function LazyImage({
  src,
  alt,
  className = '',
  fallbackSrc = '/placeholder-image.jpg',
  width,
  height
}: LazyImageProps) {
  const [isLoaded, setIsLoaded] = useState(false);
  const [error, setError] = useState(false);
  const [imageSrc, setImageSrc] = useState(src);

  useEffect(() => {
    // Wenn sich die Quell-URL ändert, Status zurücksetzen
    setIsLoaded(false);
    setError(false);
    setImageSrc(src);
  }, [src]);

  const handleLoad = () => {
    setIsLoaded(true);
  };

  const handleError = () => {
    setError(true);
    if (fallbackSrc && fallbackSrc !== src) {
      setImageSrc(fallbackSrc);
    }
  };

  return (
    <div className={`relative overflow-hidden ${className}`} style={{ width, height }}>
      {!isLoaded && !error && (
        <Skeleton className="absolute inset-0 w-full h-full" />
      )}

      <img
        src={imageSrc}
        alt={alt}
        className={`w-full h-full object-cover transition-opacity duration-300 ${
          isLoaded ? 'opacity-100' : 'opacity-0'
        }`}
        onLoad={handleLoad}
        onError={handleError}
      />
    </div>
  );
}
```

4.4 Optimiere TaskCard durch LazyImage

Aktualisiere `components/tasks/TaskCard.tsx`, um LazyImage zu verwenden:

```tsx
// In der TaskCard-Komponente

// Importiere LazyImage
import LazyImage from '@/components/ui/LazyImage';

// Ändere den Bildbereich
{task.imageUrl || (task.imageUrls && task.imageUrls.length > 0) ? (
  <LazyImage
    src={task.imageUrl || task.imageUrls[0]}
    alt={task.title}
    className="rounded-md h-24 w-24 flex-shrink-0"
    fallbackSrc="/task-placeholder.jpg"
  />
) : (
  <div className="bg-gray-200 rounded-md h-24 w-24 flex-shrink-0 flex items-center jus
    <ImageIcon className="h-8 w-8 text-gray-400" />
  </div>
)}
```

4.5 TaskList-Komponente mit virtualisierter Liste

Erstelle eine virtualisierte TaskList-Komponente unter `components/tasks/TaskList.tsx`:

tsx

```tsx
import React, { useRef, useEffect } from 'react';
import TaskCard from '@/components/tasks/TaskCard';
import { useVirtualizer } from '@tanstack/react-virtual';

interface TaskListProps {
  tasks: any[];
  userLocation?: { lat: number; lng: number } | null;
  onTaskClick: (taskId: string) => void;
}

export default function TaskList({ tasks, userLocation, onTaskClick }: TaskListProps) {
  const parentRef = useRef<HTMLDivElement>(null);

  // Virtualisierung für die Liste
  const virtualizer = useVirtualizer({
    count: tasks.length,
    getScrollElement: () => parentRef.current,
    estimateSize: () => 120, // Geschätzte Höhe einer Task-Karte
    overscan: 5, // Anzahl der zusätzlich gerenderten Elemente
  });

  // Berechne die Gesamthöhe der Liste
  const totalHeight = virtualizer.getTotalSize();

  // Virtualisierte Elemente
  const virtualItems = virtualizer.getVirtualItems();

  return (
    <div
      ref={parentRef}
      className="overflow-auto max-h-[calc(100vh-200px)]"
    >
      {/* Container mit berechneter Gesamthöhe */}
      <div
        className="relative w-full"
        style={{ height: `${totalHeight}px` }}
      >
        {virtualItems.map(virtualItem => (
          <div
            key={virtualItem.key}
            className="absolute top-0 left-0 w-full"
            style={{
              height: `${virtualItem.size}px`,
              transform: `translateY(${virtualItem.start}px)`,
            }}
          >
            <div className="p-1">
              <TaskCard
                task={tasks[virtualItem.index]}
                userLocation={userLocation}
                onClick={() => onTaskClick(tasks[virtualItem.index].id)}
              />
            </div>
          </div>
        ))}
      </div>
    </div>
  );
}
```

Hinweis: Für diese Komponente benötigst du die React Virtual-Bibliothek:

bash

```bash
npm install @tanstack/react-virtual
```

4.6 Optimiere die TasksPage

Aktualisiere deine TasksPage unter `pages/TasksPage.tsx`, um die virtualisierte Liste zu verwenden:

```tsx
// In deiner TasksPage.tsx

// Importiere die TaskList-Komponente
import TaskList from '@/components/tasks/TaskList';

// Ersetze den bestehenden Tasks-Rendering-Code
{loading ? (
  // Lade-Animation
  Array.from({ length: 5 }).map((_, index) => (
    <div key={index} className="bg-white rounded-lg p-4 shadow-sm">
      <div className="flex gap-4">
        <Skeleton className="h-24 w-24 rounded-md" />
        <div className="flex-1 space-y-2">
          <Skeleton className="h-6 w-3/4" />
          <Skeleton className="h-4 w-1/2" />
          <Skeleton className="h-4 w-1/4" />
        </div>
      </div>
    </div>
  ))
) : tasks.length > 0 ? (
  <TaskList
    tasks={tasks}
    userLocation={userLocation}
    onTaskClick={(taskId) => navigate(`/tasks/${taskId}`)}
  />
) : (
  <div className="bg-white rounded-lg p-8 text-center shadow-sm">
    <p className="text-gray-500">Keine Aufgaben gefunden.</p>
    {Object.keys(filters).some(key => filters[key] && key !== 'sortBy' && key !== 'sor
      <p className="mt-2 text-sm text-gray-500">
        Versuche es mit anderen Filtereinstellungen.
      </p>
    )}
  </div>
)}
```

5. "Passwort vergessen"-Funktion implementieren

5.1 Firebase-Funktion für Passwort-Reset hinzufügen

Füge folgende Funktion zu `lib/firebase.ts` hinzu:

typescript

```typescript
/**
 * Sendet eine E-Mail zum Zurücksetzen des Passworts
 * @param email Die E-Mail-Adresse des Benutzers
 */
export const sendPasswordResetEmail = async (email: string): Promise<void> => {
  try {
    await auth.sendPasswordResetEmail(email);
    console.log("Passwort-Reset-E-Mail gesendet an:", email);
    return Promise.resolve();
  } catch (error) {
    console.error("Fehler beim Senden der Passwort-Reset-E-Mail:", error);
    // Benutzerfreundliche Fehlermeldungen
    if (error.code === 'auth/user-not-found') {
      throw new Error('Es existiert kein Konto mit dieser E-Mail-Adresse.');
    } else if (error.code === 'auth/invalid-email') {
      throw new Error('Ungültige E-Mail-Adresse.');
    } else {
      throw new Error('Fehler beim Senden der E-Mail. Bitte versuche es später erneut.
    }
  }
};
```

5.2 ForgotPassword-Komponente erstellen

Erstelle eine neue Komponente unter `components/auth/ForgotPasswordForm.tsx`:

tsx

```tsx
import React, { useState } from 'react';
import { sendPasswordResetEmail } from '@/lib/firebase';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';
import { useToast } from '@/hooks/use-toast';
import { ArrowLeft } from 'lucide-react';
import { z } from 'zod';

// Email-Validierung mit Zod
const emailSchema = z.string().email('Bitte gib eine gültige E-Mail-Adresse ein');

interface ForgotPasswordFormProps {
  onBack: () => void;
}

export default function ForgotPasswordForm({ onBack }: ForgotPasswordFormProps) {
  const [email, setEmail] = useState('');
  const [emailError, setEmailError] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [isSuccess, setIsSuccess] = useState(false);
  const { toast } = useToast();

  const validateEmail = (email: string): boolean => {
    try {
      emailSchema.parse(email);
      setEmailError('');
      return true;
    } catch (error) {
      setEmailError('Bitte gib eine gültige E-Mail-Adresse ein');
      return false;
    }
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    // Validiere die E-Mail
    if (!validateEmail(email)) {
      return;
    }

    setIsSubmitting(true);

    try {
      await sendPasswordResetEmail(email);
      setIsSuccess(true);
      toast({
        title: "E-Mail gesendet",
        description: "Die Anweisungen zum Zurücksetzen deines Passworts wurden an dein
      });
    } catch (error) {
      setIsSuccess(false);
      toast({
        title: "Fehler",
        description: error instanceof Error ? error.message : "Ein unbekannter Fehler
        variant: "destructive",
      });
    } finally {
      setIsSubmitting(false);
    }
  };

  return (
    <div className="max-w-md w-full space-y-6 p-6 bg-white rounded-lg shadow-sm">
      <div className="flex items-center">
        <button
          type="button"
          onClick={onBack}
          className="p-1 rounded-full hover:bg-gray-100 mr-2"
```

```
          >
            <ArrowLeft className="h-5 w-5 text-gray-500" />
          </button>
          <h2 className="text-2xl font-bold">Passwort zurücksetzen</h2>
        </div>

        {isSuccess ? (
          <div className="text-center space-y-4">
            <div className="bg-green-100 text-green-800 p-4 rounded-lg">
              <p>
                Wenn ein Konto mit der E-Mail <strong>{email}</strong> existiert, haben
              </p>
            </div>
            <p className="text-gray-600">
              Überprüfe deinen Posteingang und folge den Anweisungen in der E-Mail.
            </p>
            <Button onClick={onBack} className="mt-4">
              Zurück zur Anmeldung
            </Button>
          </div>
        ) : (
          <form onSubmit={handleSubmit} className="space-y-4">
            <div className="space-y-2">
              <p className="text-gray-600">
                Gib deine E-Mail-Adresse ein, und wir senden dir einen Link zum Zurückse
              </p>

              <div className="space-y-1">
                <Label htmlFor="email">E-Mail-Adresse</Label>
                <Input
                  id="email"
                  type="email"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  placeholder="deine@email.de"
                  disabled={isSubmitting}
                  required
                />
                {emailError && (
                  <p className="text-sm text-red-500">{emailError}</p>
                )}
              </div>
            </div>

            <Button
              type="submit"
              className="w-full"
              disabled={isSubmitting}
            >
              {isSubmitting ? 'Wird gesendet...' : 'Passwort zurücksetzen'}
            </Button>
          </form>
        )}
      </div>
    );
  }
```

5.3 LoginPage aktualisieren

Aktualisiere deine `pages/LoginPage.tsx`, um einen Link zur Passwort-Reset-Funktion hinzuzufügen:

tsx

```tsx
// In pages/LoginPage.tsx

// Importiere die neue Komponente
import ForgotPasswordForm from '@/components/auth/ForgotPasswordForm';

// Füge einen neuen State hinzu
const [showForgotPassword, setShowForgotPassword] = useState(false);

// Füge einen "Passwort vergessen"-Link hinzu
<div className="flex justify-between items-center mt-2">
  <div className="flex items-center">
    <input
      type="checkbox"
      id="remember"
      className="h-4 w-4 text-indigo-600"
    />
    <label htmlFor="remember" className="ml-2 text-sm text-gray-600">
      Angemeldet bleiben
    </label>
  </div>
  <button
    type="button"
    onClick={() => setShowForgotPassword(true)}
    className="text-sm text-indigo-600 hover:text-indigo-700"
  >
    Passwort vergessen?
  </button>
</div>

// Ändere den Rendering-Code, um zwischen Login und ForgotPassword zu wechseln
return (
  <div className="flex justify-center items-center min-h-screen bg-gray-50 p-4">
    {showForgotPassword ? (
      <ForgotPasswordForm onBack={() => setShowForgotPassword(false)} />
    ) : (
      <div className="max-w-md w-full space-y-6 p-6 bg-white rounded-lg shadow-sm">
        {/* Bestehender Login-Code */}
      </div>
    )}
  </div>
);
```