

File: client/src/pages/Reading.tsx

Type: tsx | Size: 6202 bytes

```
import React, { useState, useEffect } from 'react';
import { motion } from 'framer-motion';
import { useForm } from 'react-hook-form';
import { z } from 'zod';
import { zodResolver } from '@hookform/resolvers/zod';
import { useToast } from '@hooks/use-toast';
import { useAds } from '@hooks/use-ads';
import IntentionSelector from '@components/IntentionSelector';
import CeremonyAnimation from '@components/CeremonyAnimation';
import ReadingResultCard from '@components/ReadingResultCard';
import HeartLoader from '@components/HeartLoader';
import { generateReading, shareReading } from '@lib/openai';
import { getRandomLoaderMessage } from '@lib/loaderMessages';
import type { ReadingResult } from '@shared/schema';

const formSchema = z.object({
  question: z.string().min(3, "Please enter a question").max(300, "Question is too long"),
  intention: z.string().optional(),
});

type FormData = z.infer<typeof formSchema>;

export default function Reading() {
  const { toast } = useToast();
  const { trackAction } = useAds();
  const [isLoading, setIsLoading] = useState(false);
  const [readingResult, setReadingResult] = useState<ReadingResult | null>(null);
  const [selectedIntention, setSelectedIntention] = useState("clarity");

  const { register, handleSubmit, formState: { errors } } = useForm<FormData>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      question: "",
      intention: "clarity",
    }
  });

  const intentions = [
    { id: "clarity", name: "Clarity", icon: "fa-candle-holder" },
    { id: "truth", name: "Truth", icon: "fa-crystal-ball" },
    { id: "passion", name: "Passion", icon: "fa-fire-flame-curved" },
    { id: "peace", name: "Peace", icon: "fa-dove" },
  ];

  const onSubmit = async (data: FormData) => {
    try {
      setIsLoading(true);

      // Update intention from selected value
      data.intention = selectedIntention;

      // Get intention name for display
      const intentionName = intentions.find(i => i.id === selectedIntention)?.name || "Clarity";

      const result = await generateReading(data.question, intentionName);
      setReadingResult(result);

      // Track action to potentially show interstitial ad
      // This will increment the action counter and show an ad
      // after every N actions according to configuration
      trackAction();
    } catch (error) {
      toast({
        title: "Error",
        description: "Failed to generate reading. Please try again.",
        variant: "destructive",
      });
      console.error(error);
    } finally {
      setIsLoading(false);
    }
  };
}
```

```

    }
  };

const handleIntentionSelect = (id: string) => {
  setSelectedIntention(id);
};

const handleSaveReading = () => {
  // In a full implementation, this would save to user's history
  toast({
    title: "Reading Saved",
    description: "This reading has been saved to your history.",
  });
};

const handleShareReading = async () => {
  if (!readingResult) return;

  const success = await shareReading(readingResult);

  if (success) {
    toast({
      title: "Reading Copied",
      description: "Reading copied to clipboard. Ready to share!",
    });
  } else {
    toast({
      title: "Share Failed",
      description: "Could not copy to clipboard. Please try again.",
      variant: "destructive",
    });
  }
};

return (
  <section className="px-4 py-4">
    <div className="text-center mb-8">
      <h2 className="font-serif text-3xl text-mystical-pink">Oracle Reading</h2>
      <p className="text-mystical-lavender">Ask your heart's deepest questions</p>
    </div>

    {!readingResult ? (
      <div className="bg-mystical-gradient backdrop-blur-md rounded-xl p-5 mb-8">
        <form onSubmit={handleSubmit(onSubmit)}>
          <label className="block text-mystical-light mb-2 font-medium">What would you like to know?</label>
          <textarea
            className={`w-full bg-mystical-dark border ${errors.question ? 'border-red-400' : 'border-mystic...`}
            placeholder="e.g., 'How does he feel about me?' or 'Should I text him first?'"
            {...register('question')}
          ></textarea>

          {errors.question && (
            <p className="text-xs text-red-400 mb-4">{errors.question.message}</p>
          )}

          { /* Ceremony Element */ }
          <IntentionSelector
            intentions={intentions}
            selectedIntention={selectedIntention}
            onSelectIntention={handleIntentionSelect}
          />

          <motion.button
            type="submit"
            className="w-full bg-mystical-purple hover:bg-mystical-lavender text-white font-medium py-3 roun...
            whileHover={{ scale: 1.02 }}
            whileTap={{ scale: 0.98 }}
            disabled={isLoading}
          >
            {isLoading ? 'Processing...' : 'Begin Oracle Reading'}
          </motion.button>
        </form>
      </div>
    ) : (
      <div>
        <h2>Oracle Reading</h2>
        <p>Ask your heart's deepest questions</p>
      </div>
    )}
  </section>
);

```

```

) : (
  <div>
    <ReadingResultCard
      reading={readingResult}
      onSave={handleSaveReading}
      onShare={handleShareReading}
    />
    <div className="mt-6 text-center">
      <motion.button
        className="bg-mystical-dark border border-mystical-lavender text-mystical-lavender py-2 px-6 rou...
        whileHover={{ scale: 1.03 }}
        whileTap={{ scale: 0.97 }}
        onClick={() => setReadingResult(null)}
      >
        <i className="fas fa-wand-magic-sparkles mr-2"></i>
        New Reading
      </motion.button>
    </div>
  </div>
)}

{/* Heart-shaped Loader */}
<HeartLoader
  isVisible={isLoading}
  message={getRandomLoaderMessage('reading')}
  size="large"
/>
</section>
);
}

```