Implementation Standards and Development Approach

Development Philosophy

TechPrep AI follows a structured development approach prioritizing code quality, maintainability, and scalability. Our implementation standards ensure consistent development practices across the entire application, making it easier for team members to collaborate and maintain the codebase.

Code Organization and Architecture

Frontend Implementation

Our React application follows atomic design principles and TypeScript for type safety. All components are structured to be reusable and maintainable.

Component Structure Example:

```
import { useState, useCallback } from 'react'
import type { InterviewQuestion } from '@shared/types'
import { useQuery } from '@tanstack/react-query'
import { Button } from '@/components/atoms'

export const InterviewSession: React.FC = () => {
  // State declarations at the top for clarity
  const [currentQuestion, setCurrentQuestion] = useState<InterviewQuestion | null>
(null)
  const [userAnswer, setUserAnswer] = useState<string>('')

  // API calls using React Query for efficient data fetching
  const { data: questionBank, isLoading } = useQuery(['questions'],
fetchQuestions)

  // Event handlers using useCallback for performance optimization
  const handleAnswerSubmit = useCallback(async (answer: string) => {
    try {
      const result = await submitAnswer(answer)
      setUserAnswer('')
      return result
    } catch (error) {
      console.error('Error submitting answer:', error)
    }
  }, [])

  if (isLoading) return <LoadingSpinner />

  return (
    <div className="interview-container p-4">
      <h2 className="text-2xl font-bold mb-4">Technical Interview Session</h2>
```

```
      {currentQuestion && (
        <QuestionCard
          question={currentQuestion}
          onSubmit={handleAnswerSubmit}
        />
      )}
    </div>
  )
}
```

Backend Implementation
Our FastAPI backend implements a service-oriented architecture for better separation of concerns and maintainability.
Service Layer Example:

```
from fastapi import HTTPException
from app.models.interview import Interview
from app.schemas.interview import InterviewCreate
from app.services.ai import AIService

class InterviewService:
    def __init__(self, ai_service: AIService):
        self.ai_service = ai_service

    async def create_interview(self, data: InterviewCreate) -> Interview:
        try:
            # Generate initial questions using AI service
            questions = await self.ai_service.generate_questions(
                difficulty=data.difficulty,
                topic=data.topic
            )

            # Create interview session
            interview = await Interview.create(
                user_id=data.user_id,
                questions=questions,
                status="in_progress"
            )
            return interview
        except Exception as e:
            logger.error(f"Error creating interview: {str(e)}")
            raise HTTPException(status_code=500, detail="Internal server error")
```

Database Schema
Using Prisma for type-safe database operations:

```
// prisma/schema.prisma
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

```
model User {
  id            String     @id @default(cuid())
  email         String     @unique
  name          String?
  interviews    Interview[]
  createdAt     DateTime   @default(now())
  updatedAt     DateTime   @updatedAt
}

model Interview {
  id            String     @id @default(cuid())
  userId        String
  status        String
  questions     Json[]
  feedback      Json?
  user          User       @relation(fields: [userId], references: [id])
  createdAt     DateTime   @default(now())
  updatedAt     DateTime   @updatedAt
}


Development Workflow
Version Control Practices
Branch Naming Convention:

main: Production-ready code
develop: Integration branch
feature/[feature-name]: New features
fix/[bug-name]: Bug fixes
release/[version]: Release preparations

Commit Message Format:

type(scope): description

# Examples:
feat(interview): implement real-time feedback system
fix(auth): resolve token refresh issue
docs(api): update endpoint documentation
```