AI Roleplay Trainer

A comprehensive role-playing application where users can engage with AI personas in various interactive situations and receive detailed performance feedback.

Features

🎭 Interactive Roleplay Sessions

- **Diverse Scenarios**: Job interviews, customer service, difficult conversations, networking events, and more
- **Real-time Chat Interface**: Natural conversation flow with AI personas
- **Context-Aware Responses**: AI adapts to your communication style and scenario requirements
- **Anonymous Sessions**: No registration required - uses UUID-based session tracking

📊 Performance Analysis & Feedback

- **Detailed Feedback**: Comprehensive analysis of communication patterns
- **Actionable Insights**: Specific improvement suggestions tailored to each scenario
- **Performance Scoring**: Objective metrics to track your progress
- **Strengths & Weaknesses**: Balanced assessment of your communication skills

📱 Session Management

- **Session History**: View all past roleplay sessions
- **Full Transcripts**: Review complete conversation histories
- **Progress Tracking**: Monitor improvement over time

- **Easy Navigation**: Intuitive interface for accessing past sessions

🎯 Scenario Categories

Career Development
- Job Interview Practice
- Salary Negotiation
- Performance Reviews

Customer Service
- Difficult Customer Situations
- Complaint Resolution
- Service Recovery

Social Skills
- First Date Conversations
- Networking Events
- Small Talk Mastery

Management
- Giving Constructive Feedback
- Team Conflict Resolution
- Performance Management

Technology Stack

Backend

- **FastAPI**: High-performance Python web framework
- **Supabase**: PostgreSQL database with real-time capabilities
- **Pydantic**: Data validation and serialization
- **Uvicorn**: ASGI server for production deployment

Frontend

- **Jinja2**: Server-side template rendering

- **TailwindCSS**: Utility-first CSS framework
- **Vanilla JavaScript**: Progressive enhancement for interactivity
- **Font Awesome**: Professional icon library

Database Schema

- **users**: Anonymous session tracking
- **situations**: Roleplay scenario definitions
- **roleplay_sessions**: Session management and timing
- **dialogue_messages**: Conversation history storage
- **session_summaries**: AI-generated feedback and analysis

Installation & Setup

Prerequisites

- Python 3.8+
- Supabase account and project
- OpenAI API key (for AI responses)

Quick Start

1. **Clone and Setup**
   `bash git clone <repository> cd ai-roleplay-trainer pip install -r requirements.txt`
2. **Configure Environment**
   Update `config.py` with your credentials:
   `python SUPABASE_URL = "your-supabase-url" SUPABASE_ANON_KEY = "your-anon-key" OPENAI_API_KEY = "your-openai-key" # To be added`

3. **Initialize Database**
   The database tables are already created in Supabase:
   - users
   - situations (pre-populated with 6 scenarios)
   - roleplay_sessions
   - dialogue_messages
   - session_summaries
4. **Run Application**
   `bash python run.py # or uvicorn main:app --host 0.0.0.0 --port 8000 --reload`
5. **Access Application**
   Open http://localhost:8000 in your browser

Usage Guide

Starting a Session

1. Visit the homepage
2. Browse available roleplay scenarios by category
3. Select a scenario that matches your learning goals
4. Click "Start Practice" to begin

During the Session

1. Read the scenario description and context
2. Engage with the AI persona through the chat interface
3. Type natural responses as you would in real life
4. Continue the conversation to practice your skills
5. Click "End Session" when ready for feedback

Reviewing Feedback

1. View your performance score and overall assessment
2. Read detailed analysis of your strengths
3. Review specific improvement suggestions
4. Access key insights for future development

Managing Sessions

1. Visit "History" to see all past sessions
2. Click "Review Conversation" for full transcripts
3. Access "View Feedback" for detailed analysis
4. Track your progress over time

API Endpoints

Core Routes

- `GET /` - Homepage with scenario selection
- `POST /start-session` - Initialize new roleplay session
- `GET /session/{session_id}` - Chat interface
- `POST /session/{session_id}/message` - Send message
- `POST /session/{session_id}/end` - End session
- `GET /session/{session_id}/feedback` - View feedback
- `GET /session/{session_id}/review` - Full transcript
- `GET /history` - Session history
- `GET /health` - Health check

Future Enhancements

OpenAI Integration

Currently uses mock AI responses. To integrate OpenAI:

1. **Add OpenAI API Key**
   `python # In config.py OPENAI_API_KEY = "your-openai-api-key"`
2. **Update AIPersonaService**
   Replace mock responses in `services.py` with OpenAI API calls:
   `python async def generate_response(self, situation, history): # Replace mock logic with OpenAI API calls response = await openai.ChatCompletion.acreate( model="gpt-3.5-turbo", messages=self._build_conversation_context(situation, history) ) return response.choices[0].message.content`

Additional Features

- Voice-to-text input for more natural interaction
- Advanced analytics and progress tracking
- Custom scenario creation
- Team/organization management
- Integration with learning management systems

Project Structure

```
ai-roleplay-trainer/
├── main.py                 # FastAPI application
├── config.py               # Configuration settings
├── database.py             # Supabase client setup
├── models.py               # Pydantic data models
├── services.py             # Business logic services
├── requirements.txt        # Python dependencies
├── run.py                  # Application entry point
├── templates/              # Jinja2 HTML templates
│   ├── base.html          # Base template
│   ├── home.html          # Homepage
│   ├── chat.html          # Chat interface
│   ├── feedback.html      # Feedback display
│   ├── history.html       # Session history
│   ├── review.html        # Session review
│   └── error.html         # Error page
└── static/                # Static assets
    ├── css/
    │   └── styles.css     # Custom styles
    └── js/
        └── main.js        # JavaScript functionality
```

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests if applicable
5. Submit a pull request

License

MIT License - see LICENSE file for details

Support

For questions or support, please contact the development team.

Built with ❤️ for improving communication skills through AI-powered practice.