TalentScout Hiring Assistant Chatbot Intelligent Recruitment Screening System

AI-Powered Technical Interview Platform

Contents

1 Project Overview

1.1 Brief Description

TalentScout is an intelligent Hiring Assistant chatbot designed specifically for technology recruitment. The system automates the initial screening process by collecting essential candidate information and conducting tailored technical assessments based on each candidate's declared technology stack.

1.2 Key Capabilities

- **Intelligent Data Collection**: Automated gathering of candidate details with robust validation
- **Dynamic Technical Assessment**: AI-generated questions based on candidate's tech stack
- **Real-time Answer Evaluation**: Instant feedback and scoring of technical responses
- **Multilingual Support**: Conversations in multiple languages with auto-detection
- **Sentiment Analysis**: Real-time emotion tracking during interactions
- **Personalized Experience**: Preference learning and adaptation per candidate
- **Comprehensive Validation**: Email, phone, location, and technical stack verification

2 Installation Instructions

2.1 Prerequisites

- Python 3.8+
- OpenAI API key (for question generation and evaluation)
- Internet connection (for email validation and geocoding)

2.2 Setup Steps

2.2.1 1. Environment Setup

```
# Create virtual environment
python -m venv talentscout_env
source talentscout_env/bin/activate  # On Windows: talentscout_env\Scripts\

# Clone or download project files
# Navigate to project directory
cd talentscout-hiring-assistant
```

2.2.2 2. Install Dependencies

```
pip install -r requirements.txt
```

Required packages:

- streamlit
- openai
- pydantic
- email-validator
- phonenumbers
- geopy
- pycountry
- rapidfuzz
- python-dotenv
- langdetect
- vaderSentiment

2.2.3 3. Configuration

Create `.streamlit/secrets.toml` file:

```
[openai]
api_key = "your_openai_api_key_here"

[app]
EMAIL_DELIVERABILITY = "relaxed"
DEFAULT_REGION = "US"
LOG_LEVEL = "INFO"
```

2.2.4 4. Launch Application

```
streamlit run app.py
```

3 Usage Guide

3.1 User Interaction Flow

3.1.1 Initial Greeting

1. User accesses the application through web browser
2. System displays welcome message in detected/preferred language
3. User provides consent to begin screening process

3.1.2 Information Collection Phase

The system sequentially collects:

- **Full Name**: Validated for proper format (first + last name)
- **Email Address**: Normalized and validated with optional DNS/MX checking
- **Phone Number**: International format validation with E164 normalization
- **Years of Experience**: Numeric validation (0-60 years)
- **Desired Positions**: Technical role validation against keyword database
- **Current Location**: City-Country geocoding with ISO country validation
- **Technology Stack**: Case-insensitive parsing with fuzzy matching

3.1.3 Technical Assessment Phase

1. System analyzes declared technology stack
2. Generates 3-5 tailored technical questions per technology
3. Presents questions with difficulty indicators
4. Evaluates responses in real-time
5. Provides instant feedback and scoring

3.2 Advanced Features Usage

3.2.1 Language Support

- Auto-detection from user input
- Manual language override via ISO codes (en, hi, es, etc.)
- Localized prompts and system messages

3.2.2 Personalization

- Preference storage by email address
- Difficulty level adaptation
- Recent technology topic memory
- Language preference persistence

4 Technical Details

4.1 Architecture Overview

- **Frontend**: Streamlit web application with custom CSS styling
- **Backend**: Python-based processing with OpenAI LLM integration
- **Data Storage**: JSON file-based candidate and profile storage
- **Validation**: Multi-layered input validation with external API integration

4.2 Core Components

4.2.1 Application Structure

- `app.py`: Main Streamlit application with UI and conversation flow
- `schemas.py`: Pydantic models for data validation and structure
- `llm.py`: OpenAI integration for question generation and evaluation
- `utils.py`: Utility functions for sentiment analysis and language detection
- `storage.py`: File-based data persistence and retrieval
- `prompts.py`: LLM prompt templates and examples

4.2.2 Key Libraries and Their Roles

- **Streamlit**: Web application framework and chat interface
- **OpenAI**: Language model for question generation and evaluation
- **Pydantic**: Data validation and serialization
- **email-validator**: Email syntax and deliverability checking
- **phonenumbers**: International phone number validation
- **geopy/Nominatim**: Geocoding and location validation
- **RapidFuzz**: Fuzzy string matching for technology recognition
- **langdetect**: Automatic language detection
- **vaderSentiment**: Real-time sentiment analysis

4.3 Data Models

4.3.1 Candidate Schema

```
class Candidate(BaseModel):
    consent: bool = False
    full_name: str = ""
    email: str = ""
    phone: str = ""
    years_experience: float = 0.0
    desired_positions: List[str] = []
    current_location: str = ""
    tech_stack: TechStack = TechStack()
    language: str = "en"
```

4.3.2 Technology Stack Schema

```
class TechStack(BaseModel):
    languages: List[str] = []
    frameworks: List[str] = []
    databases: List[str] = []
    tools: List[str] = []
```

5 Prompt Design

5.1 Information Gathering Prompts

Prompts were designed with the following principles:

- **Clarity**: Simple, unambiguous language
- **Specificity**: Clear format expectations (e.g., "City, Country")
- **Multilingual**: Localized versions for key languages
- **Error Handling**: Graceful failure messages with retry instructions

5.2 Technical Question Generation

- **Context-Aware**: Questions tailored to declared technology stack
- **Difficulty Scaling**: Beginner to advanced levels based on experience
- **Comprehensive Coverage**: Multiple aspects of each technology
- **Practical Focus**: Real-world scenarios and problem-solving

5.3 Evaluation Criteria

- Technical accuracy and depth
- Practical application understanding

- Problem-solving approach
- Communication clarity
- Best practices awareness

6 Challenges & Solutions

6.1 Input Validation Challenges

6.1.1 Challenge: Email Validation Reliability

Problem: Standard email validation often failed due to DNS/MX lookup issues behind corporate firewalls.

Solution: Implemented layered validation approach:

- Syntax validation (always performed)
- Unicode normalization for international characters
- Domain fuzzy matching for common typos
- Optional DNS/MX checking with graceful degradation

6.1.2 Challenge: Location Validation Accuracy

Problem: Users often provided inconsistent city-country combinations.

Solution: Implemented strict geocoding validation:

- Country normalization using ISO standards
- Geocoding constrained by country boundaries
- Suggestion system for likely alternatives
- Fuzzy matching for common country name variations

6.2 Technical Stack Processing

6.2.1 Challenge: Case-Sensitive Technology Recognition

Problem: Users entered technology names in various cases, causing recognition failures.

Solution: Implemented comprehensive case-insensitive matching:

- Lowercase indexing of canonical technology names
- Alias mapping for common variations
- Fuzzy matching with high confidence thresholds
- Whitelist filtering to prevent false positives

6.3 Performance Optimization

6.3.1 Challenge: LLM API Rate Limits and Costs

Problem: Frequent OpenAI API calls could hit rate limits and increase costs.
Solution: Implemented smart caching and fallback strategies:

- Question template caching
- Exponential backoff for API retries
- Deterministic fallback questions
- Batch processing where possible

6.4 User Experience Enhancements

6.4.1 Challenge: Conversation State Management

Problem: Streamlit's rerun model required careful state management to maintain conversation flow.
Solution: Implemented robust session state handling:

- Input-first processing pattern
- Comprehensive state persistence
- Phase-based conversation tracking
- Graceful error recovery

7 Security and Privacy Considerations

7.1 Data Protection

- Local file storage for candidate data
- No external data transmission except for validation APIs
- Consent-based data collection
- Secure API key management through Streamlit secrets

7.2 Privacy Compliance

- GDPR-compatible consent mechanism
- Data minimization principles
- User control over data deletion
- Transparent data usage policies

8 Future Enhancements

8.1 Planned Features

- Video interview integration
- Advanced analytics and reporting
- Integration with ATS systems
- Machine learning-based candidate scoring
- Real-time collaboration features
- Mobile application development

8.2 Scalability Considerations

- Database migration from file-based storage
- Microservices architecture implementation
- Cloud deployment optimization
- Load balancing and caching strategies

9 Conclusion

The TalentScout Hiring Assistant represents a comprehensive solution for automated technical recruitment screening. By combining robust validation, intelligent question generation, and advanced user experience features, it provides an efficient and effective tool for identifying qualified technology candidates.

The system's modular architecture, extensive error handling, and focus on user experience make it suitable for both small-scale implementations and enterprise-level deployments. The documentation and codebase provide a solid foundation for future enhancements and customizations.