SmartHire: Generative AI for Streamlined Job Fit and Interview Evaluation

Overview

This project is an AI-Powered Interview Evaluation System that utilizes a multi-agent architecture to analyze resumes, predict job fit, generate interview questions, and evaluate interview answers. The system leverages Generative AI models and machine learning to streamline the hiring process by automating tasks like resume feature extraction, job fit prediction, interview question generation, and answer evaluation. The entire system is orchestrated via LangGraph workflow and presented through a Streamlit web interface.

Objective

The system implements an AI Interview Evaluation System with the following key functionalities:

Resume Analysis:

Extract features from uploaded resumes and match them against the provided job description. Use Generative AI models for analyzing the content of resumes, extracting key information, and preparing it for the next steps in the evaluation. Implemented as resume_parser agent in the multi-agent workflow.

Job Fit Prediction:

Based on the extracted resume features and the job description, predict the job fit using a generative AI model. Provide reasoning for why the candidate is either a “Fit” or “Not Fit” for the job role. Implemented as job_fit_analyzer agent in the multi-agent workflow.

Interview Question Generation:

Automatically generate interview questions tailored to the candidate’s profile, including: Concept-based questions. Code-related questions. Use Generative AI to create these questions based on the candidate’s experience level and the job description. Implemented as question_generator agent in the multi-agent workflow.

Interview Answer Evaluation:

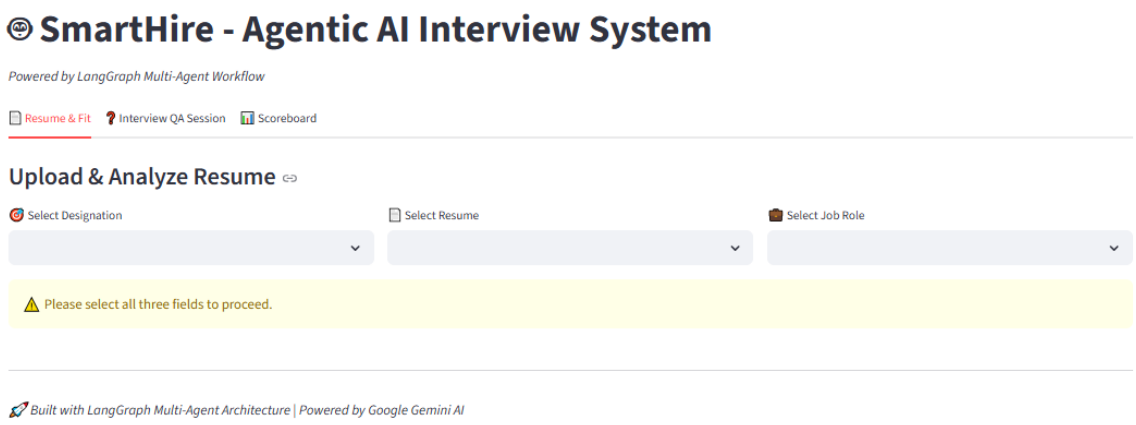
Evaluate answers provided by the candidate to interview questions. This evaluation should be done using a generative AI model to assess the completeness and quality of the answer based on predefined standards. Implemented as answer_evaluator agent in the multi-agent workflow.

Multi-Agent Architecture:

The system implements a stateful workflow using LangGraph to orchestrate multiple specialized AI agents. Each agent focuses on a specific task and maintains state through the CandidateState schema. The workflow includes proper error handling, fallback mechanisms, and state persistence.

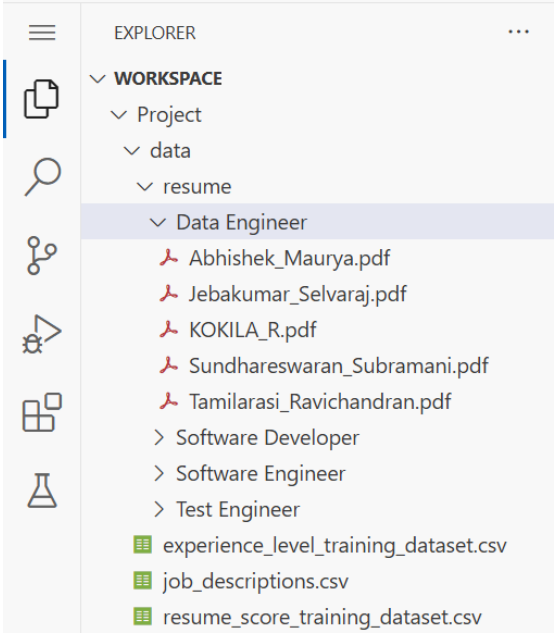
UI Interface:

Create a Streamlit-based user interface where users can upload resumes, input job descriptions, and view the generated interview questions and evaluations. The interface should also display the job fit analysis and allow for interactive exploration of the generated content.



File Structure:

Data Directory File Structure:



Given: data directory:

1) data/resume: organized by designation with corresponding resumes.

2) data/job_descriptions.csv: contains job descriptions.

3) data/experience_level_training_dataset.csv: Used for training ML model for experience prediction.

4) data/resume_score_training_dataset.csv: Used for training ML model for resume score prediction.

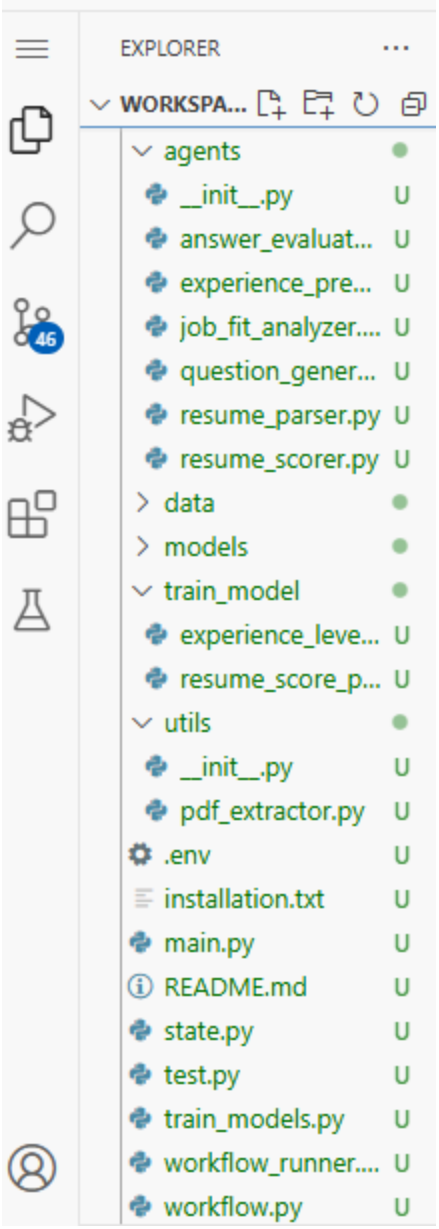
env: Contains API keys for Gemini models.

installation.txt: Packages to be installed.

Current Project Structure:

```
├── agents/ # Specialized AI agents
│   ├── resume_parser.py # Extract resume features using Gemini AI
│   ├── experience_predictor.py # Predict experience level using ML
│   ├── resume_scorer.py # Score resume quality using ML
│   ├── job_fit_analyzer.py # Analyze job fit using Gemini AI
│   ├── question_generator.py # Generate interview questions using Gemini AI
│   ├── answer_evaluator.py # Evaluate answers using SBERT and Gemini AI
│   └── __init__.py
├── utils/ # Utility functions
```

```
│   ├── pdf_extractor.py # Extract text from PDF resumes
│   └── __init__.py
├── train_model/ # Alternative training modules
│   ├── resume_score_predictor.py
│   └── experience_level_estimator.py
├── state.py # Defines CandidateState schema for workflow
├── workflow.py # Defines LangGraph workflow structure
├── workflow_runner.py # Runs workflow with progress tracking
├── main.py # Streamlit UI application
├── train_models.py # Training script for ML models
├── test.py # Unit test suite
├── models/ # Directory for trained ML models
└── data/ # Data directory with resumes and datasets
```



Modules to be created & implemented:

state.py: Defines the shared state for the LangGraph workflow Module Functionality: Defines the CandidateState TypedDict that maintains all data throughout the workflow lifecycle. The state includes resume text, job descriptions, extracted features, scores, questions, answers, and error tracking.

Data Structure: { “resume_text”: str, # Raw resume text “job_description”: str, # Job description text “job_title”: str, # Target job title “candidate_name”: str, # Name of candidate “resume_features”: dict, # Extracted resume features “experience_level”: str, # Predicted experience level “resume_score”: float, # Resume quality score “job_fit”: dict, #

Job fit analysis “questions”: list, # Generated interview questions “answers”: list, # Candidate answers “final_score”: float, # Final evaluation score “feedback”: str, # Overall feedback “current_step”: str, # Current workflow step “errors”: list # Error tracking }

agents/resume_parser.py: Parses the information from resume text using Gemini AI Function Signature: def parse_resume(state: CandidateState) -> CandidateState

Return Type: The function returns the updated state with parsed resume features: { “resume_features”: { “full_name”: str, “email”: str, “phone”: str, “education”: { “degree”: str, “major”: str, “university”: str }, “total_experience_years”: float, “skills”: list of str, “projects”: list of str, “certifications”: list of str, “leadership_experience”: int, “has_research_work”: int }, “candidate_name”: str, “current_step”: str }

agents/job_fit_analyzer.py: In this module, creates a function to predict whether a candidate is a fit for a job based on their resume features and the job description. The function uses a Generative AI model (Gemini) to analyze structured resume data and the job description, and then return a prediction indicating whether the candidate is a “Fit” or “Not Fit” for the job role.

Function to Implement: The function job_fit_analyze, takes the following parameters: state (CandidateState): The current workflow state containing resume features and job description.

Function Signature: def analyze_job_fit(state: CandidateState) -> CandidateState

Return Type: The function returns the updated state with job fit information: { “job_fit”: { “job_fit”: “Fit” or “Not Fit”, “fit_score”: float (0-10), “reason”: “Short and specific explanation” }, “current_step”: str }

job_fit: Indicates whether the candidate is a “Fit” or “Not Fit” for the role. fit_score: Numeric score from 0-10 indicating fit level reason: Provides a brief explanation of why the candidate is considered a fit or not.

agents/question_generator.py - Interview Question Generation In this module, creates a function that generates interview questions for a candidate based on their resume features, job description, job fit evaluation, and other relevant factors. The function utilizes a Generative AI model (Gemini) to create technical interview questions tailored to the candidate’s profile.

The function generate_questions creates interview questions based on the following inputs from state: Resume Features: A dictionary of structured resume features extracted from the candidate’s resume. Job Description: The job description for the target role. Experience Level: The candidate’s experience level (e.g., Junior, Mid-level, Senior). Job Title: The title of the job role the candidate is being considered for.

The AI will generate: 3 Interview Questions: Mix of Conceptual questions & Code questions. Each question will include a reference answer. The questions will be relevant to the job description and the candidate’s experience level.

Function Signature: def generate_questions(state: CandidateState) -> CandidateState

Return Type: The function returns the updated state with questions: [{ “id”: “Q1”, “type”: “concept”, # or “code” “question”: “What is the difference between a stack and a queue?”, “reference_answer”: “A stack is a LIFO structure (Last In First Out), while a queue is a FIFO structure…” }]

agents/answer_evaluator.py - Interview Answer Evaluation In this module, creates functions to evaluate the answers provided by a candidate in an interview. The evaluation process includes both conceptual questions (theoretical or explanatory) and coding questions (practical programming problems). The evaluation will be done using semantic similarity for conceptual answers and Generative AI (Gemini) for feedback and scoring.

Functions to Implement: Implement the primary function evaluate_answers in this module. Below are the details:

evaluate_answers - Master Evaluation Function Functionality: This function evaluates all submitted answers using different methods for different question types: For conceptual questions: Uses SBERT (Sentence Transformers) to calculate semantic similarity For coding questions: Uses Gemini AI to assess code quality, logic, and completeness The function consolidates scores and returns final evaluation.

Steps: The function checks each question type (concept or code). For concept questions, uses SBERT semantic similarity with reference answers. For code questions, uses Gemini AI for assessment. Calculates final score as average of all question scores. Returns comprehensive evaluation results.

Expected Output: The function returns the updated state with evaluation results: { “answers”: [# Updated with scores and feedback { “question_id”: “Q1”, “answer”: str, “score”: float, “feedback”: str }], “final_score”: float, # Average of all question scores “current_step”: str }

Function Signature: def evaluate_answers(state: CandidateState) -> CandidateState

agents/experience_predictor.py - Experience Level Prediction This module trains and uses a machine learning model that predicts a candidate’s experience level based on various features extracted from their resume. The model is trained on data/experience_level_training_dataset.csv and can be used to predict experience level (Junior, Mid-level, Senior) for new candidates.

Functions Implemented: 1. train_experience_model - Model Training Functionality: Train the ML model using resume features and experience level targets. Save the trained model & encoder in models/ directory. Function Signature: def train_experience_model() -> bool

2. predict_experience - Experience Level Prediction Functionality: This function loads the trained model and uses it to predict the experience level for new resumes based on their features. Function Signature: def predict_experience(state: CandidateState) -> CandidateState Expected Output: Updated state with experience level prediction.

agents/resume_scorer.py - Resume Quality Scoring This module trains and uses a machine learning model that predicts a resume quality score based on several features extracted

from a candidate's resume. The model is trained on data/resume_score_training_dataset.csv and can be used to predict resume quality scores (0-10) for new resumes.

Functions Implemented: 1. train_resume_score_model - Model Training Functionality: Train the ML model using resume features and score targets. Save the trained model, scaler, and features in models/ directory. Function Signature: def train_resume_score_model() -> bool

2. predict_resume_score - Resume Quality Score Prediction Functionality: This function loads the trained model and uses it to predict the resume quality score for new resumes based on their features. Function Signature: def predict_resume_score(state: CandidateState) -> CandidateState Expected Output: Updated state with resume score (0-10).

utils/pdf_extractor.py - Extracting Text from PDF Resumes In this module, creates a function to extract raw text from a PDF resume. The function uses the PyMuPDF library (via fitz) to extract the text content from each page of the PDF file. This text will then be used for further processing, such as feature extraction for resume analysis.

Function to Implement: The function extract_text_from_pdf extracts text from a given PDF file and returns it as a string.

Function Signature: def extract_text_from_pdf(pdf_path: str) -> str: Parameters: pdf_path: The path to the PDF file from which text needs to be extracted. Return Type: The function returns the extracted text from the PDF as a string.

workflow.py - LangGraph Workflow Definition This module defines the multi-agent workflow using LangGraph. It orchestrates the various agents in a sequential workflow: resume parsing → experience prediction → resume scoring → job fit analysis → question generation.

Components: - StateGraph: Defines the workflow with CandidateState schema - Nodes: Each agent is a node in the workflow - Edge connections: Defines the flow between agents - Entry point: Starting point of the workflow

workflow_runner.py - Workflow Execution with Progress Tracking This module runs the LangGraph workflow while providing real-time progress updates to the UI. It includes error handling, timeout management, and progress bar updates during execution.

main.py: Streamlit Application: SmartHire: Generative AI for Streamlined Job Fit and Interview Evaluation (Updated) This is a comprehensive AI-powered system for managing and evaluating interviews using multiple LangGraph agents for resume analysis, job fit prediction, experience level classification, interview question generation, and scoring. The application leverages Generative AI (Gemini) for job fit prediction and feedback generation, and machine learning models for predicting experience level and resume quality.

Key Components: Streamlit Setup: The app is built using Streamlit, which provides a clean user interface for interacting with the system. The application includes three main tabs:

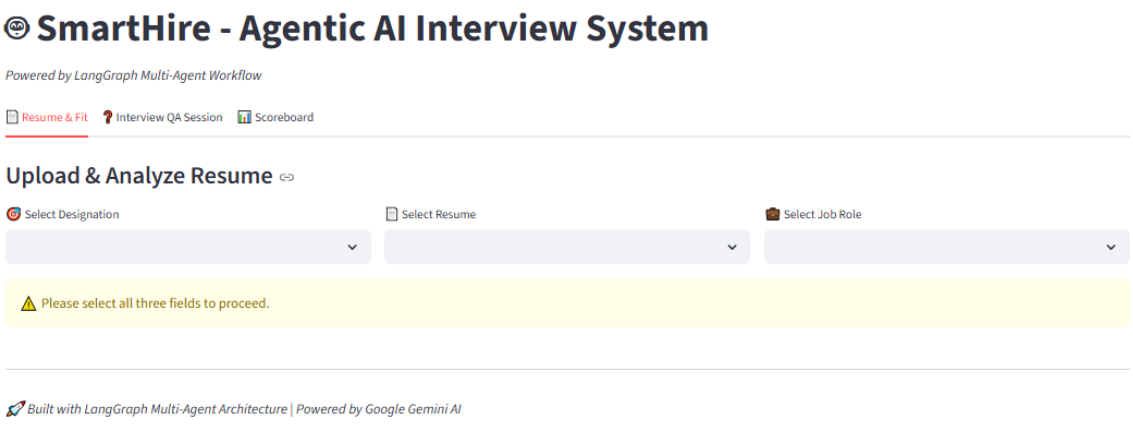
Resume & Fit: Upload and analyze resumes, and assess job fit. Interview QA Session: Conduct interview sessions, generate questions, and evaluate answers. Scoreboard: Track candidates' performance and evaluate overall scores.

1. TAB 1: Resume & Fit Functionality: Resume Upload: Candidates can upload their resumes (in PDF format) for analysis. Job Selection: Select the job role to match the candidate's resume with the requirements. Resume Analysis: The app extracts text from the uploaded resume using the extract_text_from_pdf function and processes it using the LangGraph workflow for comprehensive analysis.

Machine Learning Scoring: Experience Level: The system predicts the candidate's experience level (e.g., Junior, Mid-level, Senior) using a pre-trained machine learning model via the experience_predictor agent. Resume Score: A score is assigned to the resume based on key features using the resume_scorer agent. Job Fit: Generative AI predicts how well the candidate matches the job description via the job_fit_analyzer agent.

Outputs: Experience Level: Displayed in the app. Resume Score: Displays the calculated resume score. Job Fit: Shows if the candidate is a Fit or Not Fit for the job with reasoning.

UI Elements: Select Designation: Dropdown to select the designation from data/resume/ subdirectories. Select Resume: Dropdown to select the uploaded resume. Select Job Role: Dropdown to select the job title from job_descriptions.csv. Resume Features: Display the extracted features from the resume. Evaluation Results: Display the Experience Level, Resume Score, and Job Fit.

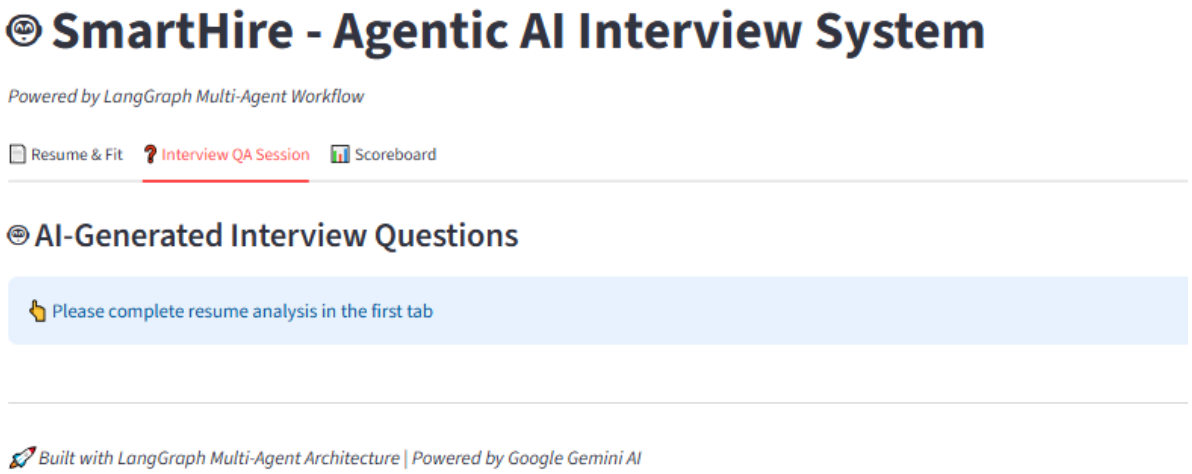


2. TAB 2: Interview QA Session Functionality: Interview Questions: The system displays questions generated via the question_generator agent based on the candidate's profile. Answer Submission: The student answers each question. The answers are evaluated using the answer_evaluator agent, which scores the answer and provides feedback. Real-time Evaluation: After submitting answers, the score and feedback for each question are displayed.

Outputs: Student's Answer: Display the student's response to each question. Score and Feedback: After evaluating the answer, display the score (0-10) and feedback. Final

Evaluation: Once all answers are evaluated, the average score is calculated and available in Tab 3.

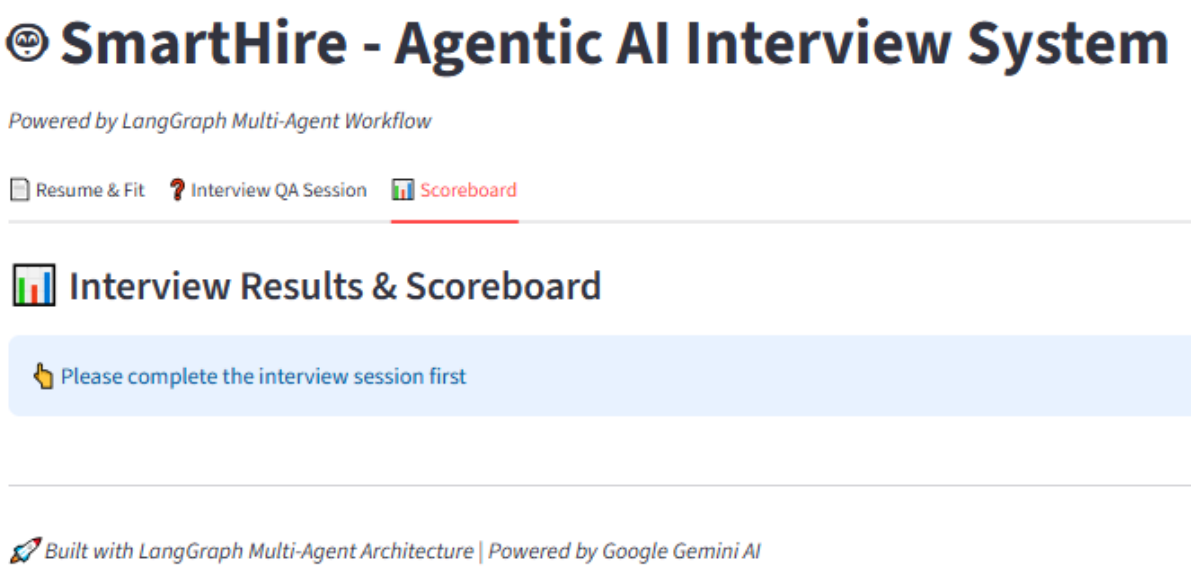
UI Elements: Interview Questions: Display each generated question with a space for the student to input answers. Submit Answer: Students can submit their answers, which are then evaluated. Progress Tracking: Shows evaluation progress for all questions.



3. TAB 3: Scoreboard Functionality: This tab displays the final evaluation results including scores from all components. The scoreboard shows the overall performance metrics including resume score, experience level, job fit score, and final evaluation score.

Outputs: Performance metrics for the candidate including Overall Score, Resume Score, Experience Level, Job Fit Score, and Question-by-question evaluation.

UI Elements: Performance Metrics: Displays key metrics in a dashboard format. Detailed Analysis: Expandable sections for question-by-question feedback. Export Results: Download detailed evaluation reports in JSON format.



Explanation of Key Functions and Models: extract_text_from_pdf: Extracts text from resumes (PDF format) using PyMuPDF (fitz).

parse_resume (resume_parser agent): Extracts structured data from the raw resume text using Generative AI to identify features such as skills, projects, certifications, and education.

analyze_job_fit (job_fit_analyzer agent): Uses Generative AI to predict whether the candidate is a Fit for the selected job based on their resume features and the job description.

predict_experience (experience_predictor agent): A machine learning model predicts the experience level of the candidate based on features like skills count, years of experience, and leadership experience.

predict_resume_score (resume_scorer agent): Predicts the resume quality score based on the resume's features (skills, certifications, experience).

generate_questions (question_generator agent): Generates a set of interview questions based on the candidate's resume features, job description, and experience level.

evaluate_answers (answer_evaluator agent): Evaluates the student's answers to interview questions using semantic similarity for conceptual questions and Generative AI for coding answers. It returns scores and feedback.

LangGraph Multi-Agent Workflow: The system uses a stateful multi-agent architecture where each specialized agent focuses on one task: Resume Parser Agent → Experience Predictor Agent → Resume Scorer Agent → Job Fit Analyzer Agent → Question Generator Agent

UI Flow: Resume & Fit: Candidate uploads a resume. System performs resume analysis via LangGraph workflow and provides evaluations. Interview QA Session: System displays generated questions based on the candidate's profile. Candidate answers the questions.

System evaluates answers and displays scores and feedback. Scoreboard: View the final comprehensive evaluation results.

Technical Stack: - LangGraph: Multi-agent workflow orchestration - Streamlit: Interactive web interface - Google Gemini AI: Natural language processing and evaluation - Sentence Transformers: Semantic similarity for answer evaluation - PyMuPDF: PDF text extraction - Scikit-learn: Machine learning models - Pandas: Data manipulation - Python-dotenv: Environment variable management

Commands to Create a Google Gemini API Key Open your web browser.

Launch any browser (e.g., Chrome, Firefox) on your computer.

Go to Google AI Studio.

In the address bar, type aistudio.google.com and press Enter.

Sign in to your Google account. Click the "Sign In" button in the top-right corner.

Enter your Google email and password, then click "Next" to log in.

If you don't have an account, click "Create Account" and follow the prompts to make one. Navigate to the API Key section.

On the Google AI Studio homepage, look at the left sidebar. Click on "Get API Key" (usually near the top-left corner).

Create a new API key. In the API Key section, click the "Create API Key" button. A pop-up will appear—select "Create API Key in new project" (or choose an existing project if you have one).

Click "Create" to generate the key. Copy the generated API key. Once the key is created, it will appear on the screen. Click the "Copy" button next to the key (or highlight it and press Ctrl+C/Command+C).

Save the key in a secure place (e.g., a text file or password manager) because it won't be shown again.

Implementation Explanation: Before executing the main.py, enter the Gemini API keys in the .env file (GEMINI_API_KEY_1 through GEMINI_API_KEY_4).

Open the terminal in the Project directory. Check the path is in the Project directory, if not use cd command to navigate. To install required packages, run pip install -r installation.txt in terminal.

Implementation Explanation:

Before executing the main.py, enter the Gemini API key in the .env file.

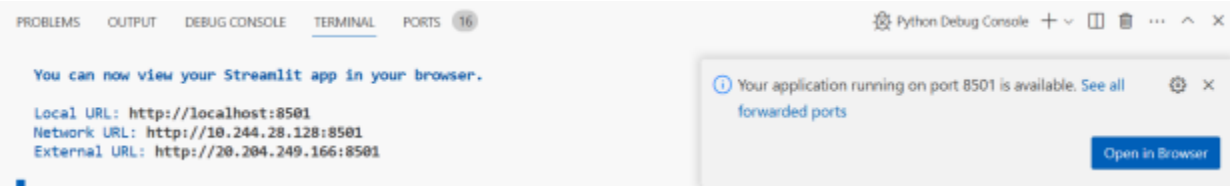
Open the main.py integrated terminal.

check the path it in the Project directory, if not use cd command to navigate.

To install required packages, run pip install -r installation.txt in terminal.

Use python3 -m streamlit run main.py to execute the application, then you will get the pop-up window below,

click the assigned port (Open in Browser) which will navigate to streamlit application window.



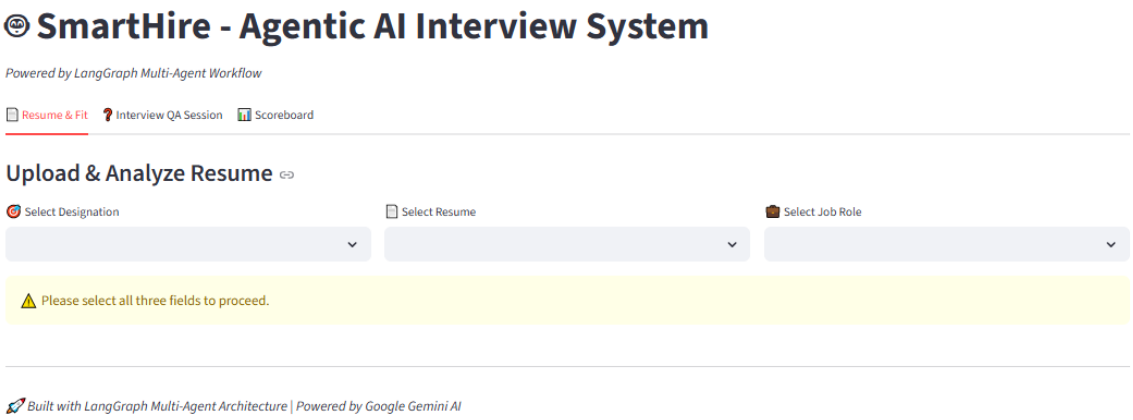
IMPORTANT: Before you are running testcases make sure your Gemini API free tier is not exhausted.

To check the testcases, you can use python3 -W ignore -m pytest tests.py -v (check the directory it should

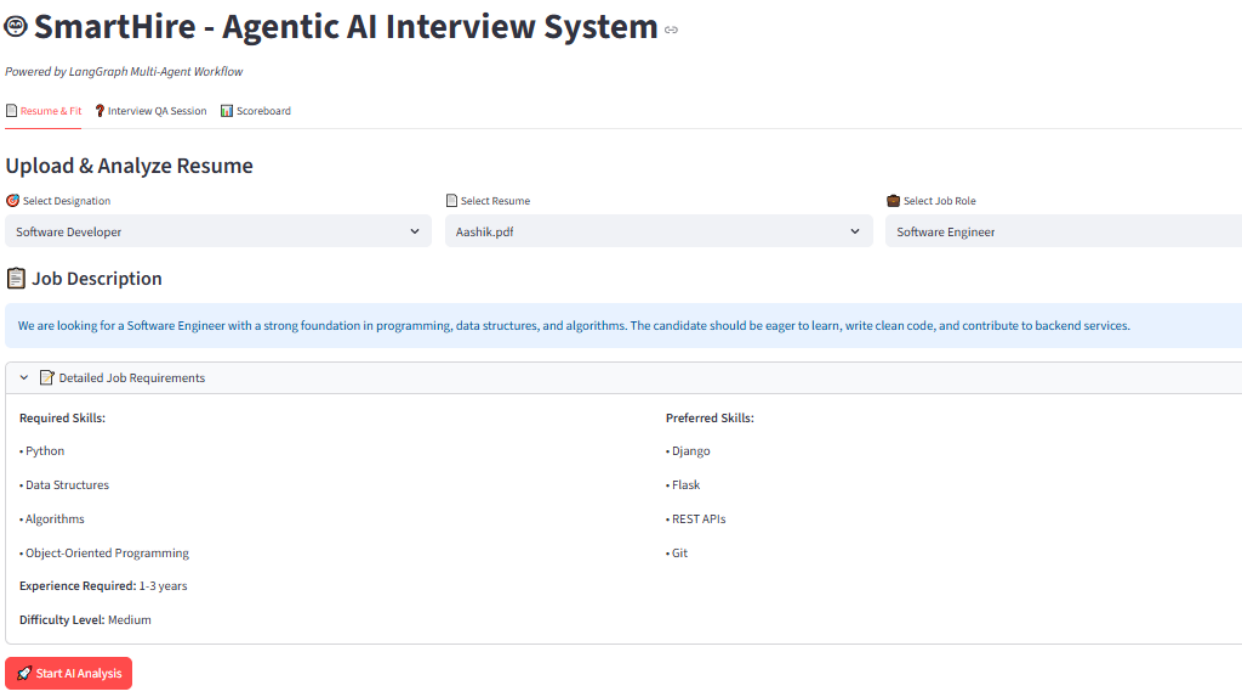
be Project directory)

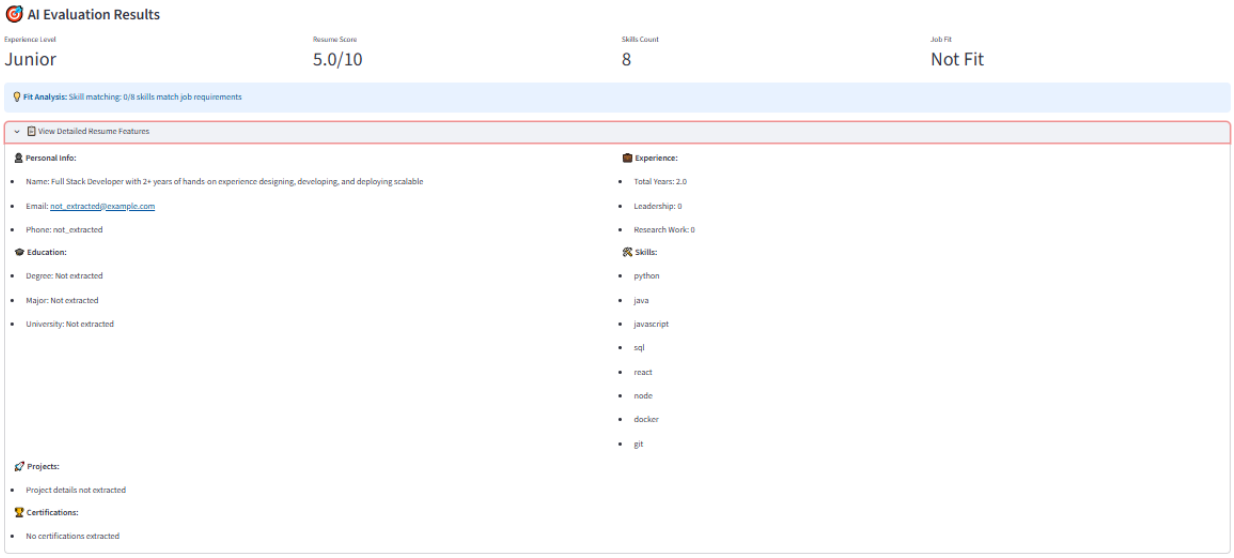
Sample Output:

This below shared UI image will be the first page when we start the streamlit



To train ML models, run python train_models.py first. Use streamlit run main.py to execute the application, then you will get the pop-up window below, click the assigned port (Open in Browser) which will navigate to streamlit application window.





So each QA session will be like a cards with questions and stored permanently. Once a response is done, same person can't try another QA session for same job role.

SmartHire - Agentic AI Interview System

Powered by LangGraph Multi-Agent Workflow

Resume & Fit | Interview QA Session | Scoreboard

AI-Generated Interview Questions

✅ Generated 3 personalized questions

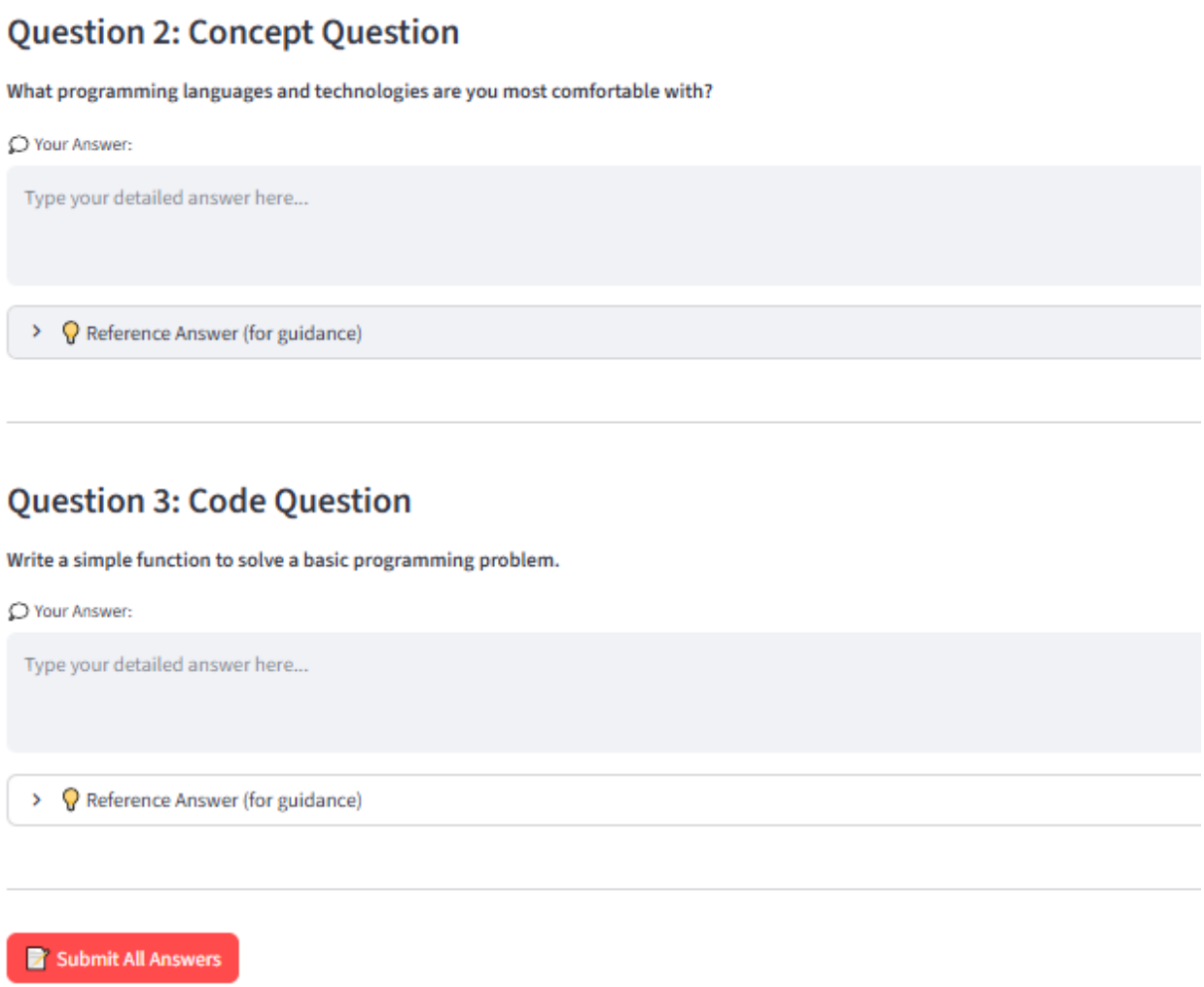
Question 1: Concept Question

Tell me about your experience with Software Engineer and what interests you about this position?

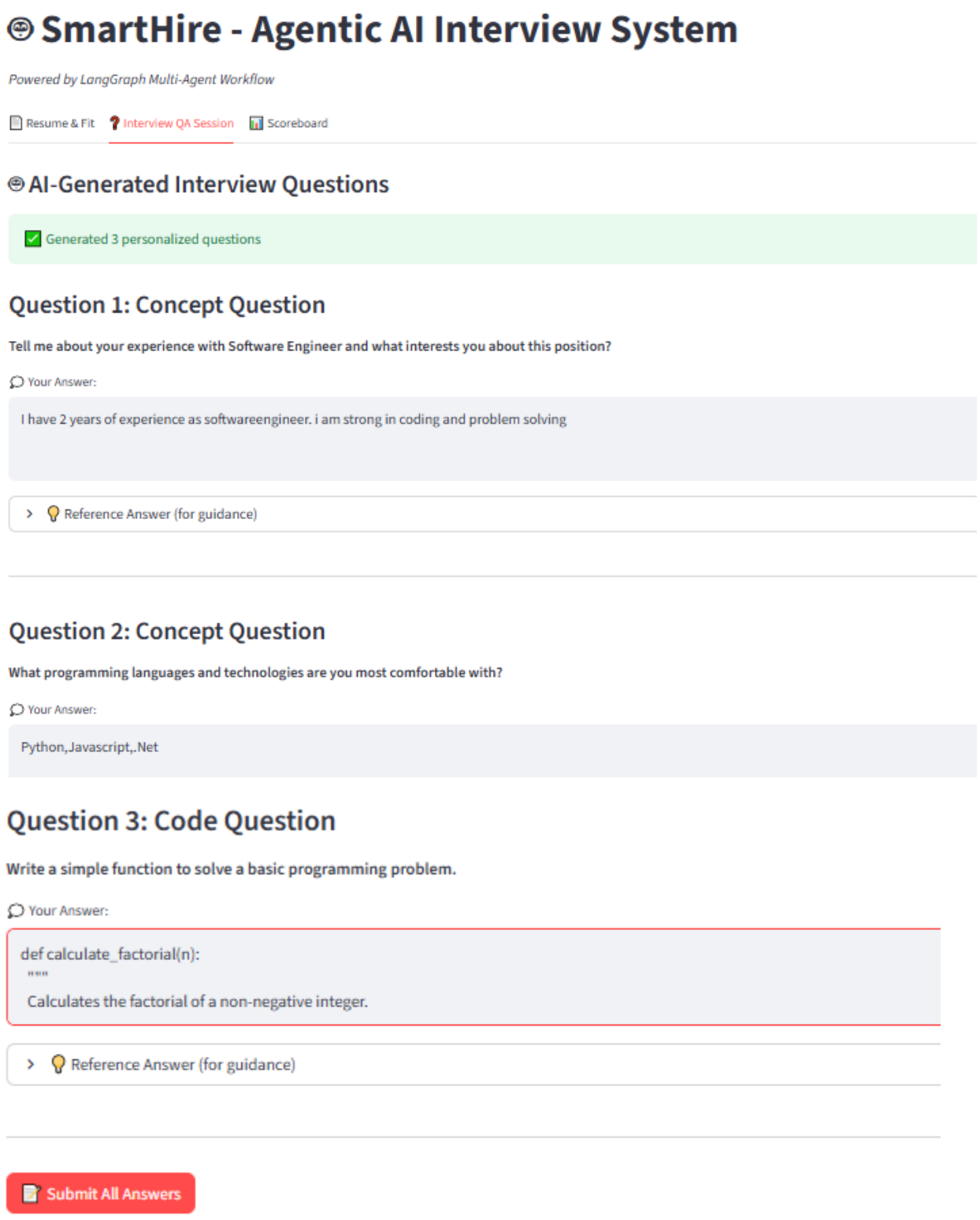
Your Answer:

Type your detailed answer here...

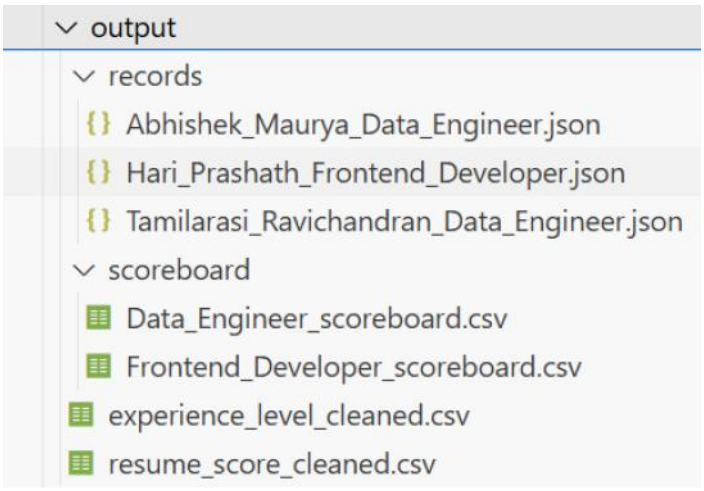
Reference Answer (for guidance)



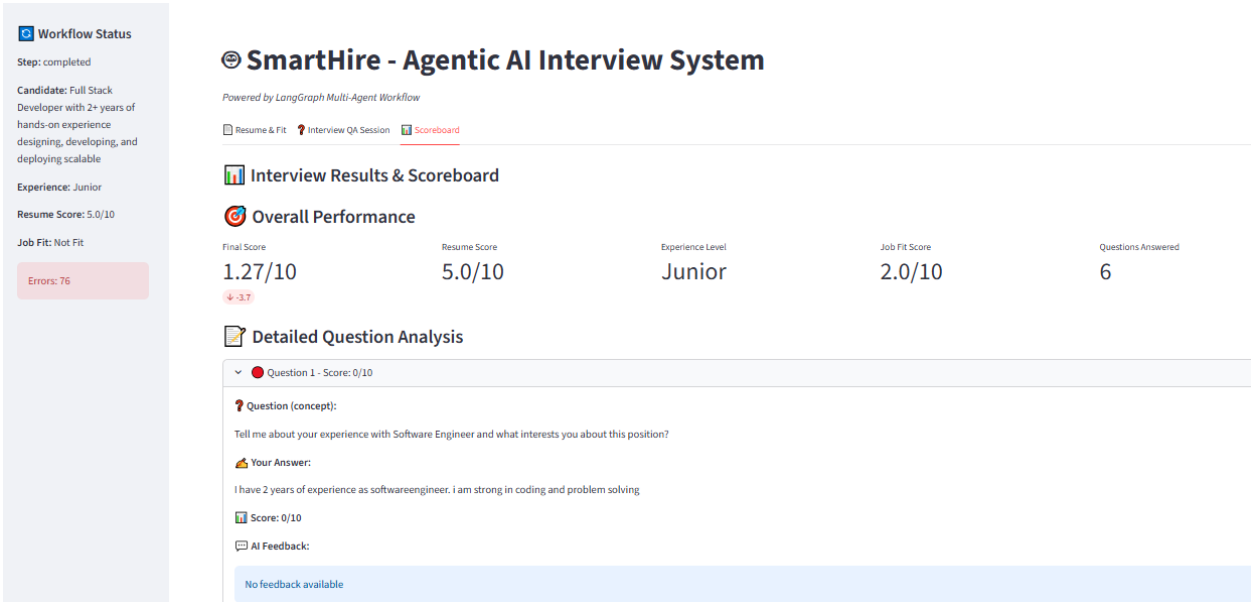
Type the appropriate answer and submit.



Once all Questions are completed it will be updated in scoreboard. The records and scoreboard are maintained in output directory.



Now, you can check the score board,



Question 2 - Score: 0/10

Question (concept):

What programming languages and technologies are you most comfortable with?

Your Answer:

Python,Javascript,.Net

Score: 0/10

AI Feedback:

No feedback available

Question 3 - Score: 0/10

Question (code):

Write a simple function to solve a basic programming problem.

Your Answer:

def calculate_factorial(n): """ Calculates the factorial of a non-negative integer.

Args: n: A non-negative integer.

Returns: The factorial of n, or an error message if n is negative. """ if not isinstance(n, int): return "Error: Input must be an integer." if n < 0: return "Error: Factorial is not defined for negative numbers." elif n == 0: return 1 else: factorial = 1 for i in range(1, n + 1): factorial *= i return factorial

Example usage:

print(f"Factorial of 5: {calculate_factorial(5)}") print(f"Factorial of 0: {calculate_factorial(0)}") print(f"Factorial of -3: {calculate_factorial(-3)}") print(f"Factorial of 3.5: {calculate_factorial(3.5)}")

Score: 0/10

AI Feedback:

No feedback available

Question 4 - Score: 2.84/10

Your Answer:

I have 2 years of experience as softwareengineer. i am strong in coding and problem solving

Score: 2.84/10

AI Feedback:

Semantic similarity: 0.28

Question 5 - Score: 0.98/10

Your Answer:

Python,Javascript,.Net

Score: 0.98/10

AI Feedback:

Semantic similarity: 0.10

Question 6 - Score: 0.0/10

Your Answer:

def calculate_factorial(n): """ Calculates the factorial of a non-negative integer.

Args: n: A non-negative integer.

Returns: The factorial of n, or an error message if n is negative. """ if not isinstance(n, int): return "Error: Input must be an integer." if n < 0: return "Error: Factorial is not defined for negative numbers." elif n == 0: return 1 else: factorial = 1 for i in range(1, n + 1): factorial *= i return factorial

Example usage:

print(f"Factorial of 5: {calculate_factorial(5)}") print(f"Factorial of 0: {calculate_factorial(0)}") print(f"Factorial of -3: {calculate_factorial(-3)}") print(f"Factorial of 3.5: {calculate_factorial(3.5)}")

Score: 0.0/10

AI Feedback:

Evaluation failed

Now you can find the summery report, also you can download this json format.