Step-by-Step Explanation of Epic Evaluation System Using Gemini API

Overview:  
This script evaluates Agile Epic descriptions using Google's Gemini 2.0 Flash model. It checks whether each part of the epic (like title, problem statement, user stories, etc.) meets the required quality standards. The script uses the Google Generative AI API to generate and evaluate content.  
  
1. Environment Setup:  
- The `.env` file stores the `GOOGLE\_API\_KEY` securely.  
- `load\_dotenv()` loads environment variables from the `.env` file into Python.  
- The key is required to authenticate and access the Gemini model API.  
  
2. Libraries Used:  
- `os`: Access environment variables and file paths.  
- `dotenv`: Load environment variables from the `.env` file.  
- `json`: Read and write JSON data.  
- `logging`: Log process information for debugging.  
- `google.generativeai`: Main library to interact with Google's Gemini model.  
- `datetime`: Generate timestamped filenames.  
- `typing.Dict`: Used for type hinting function outputs.  
  
3. Google API Configuration:  
- The script fetches the API key from environment variables and configures the Gemini API using `genai.configure(api\_key=...)`.  
  
4. QUALITY\_STANDARDS:  
- A large multiline string defines what makes a high-quality epic. Each element like Title, Problem Statement, Product Outcome, etc., has its criteria and examples.  
  
5. Class `EpicEvaluator`:  
- Manages all logic related to identifying and evaluating epic elements using the Gemini model.  
  
6. `identify\_elements()`:  
- Sends the epic text to the Gemini model and asks it to extract elements like Title, Problem Statement, User Stories, etc.  
- Expects a JSON format response.  
- Parses and validates the response to ensure it's usable.  
  
7. `evaluate\_element()`:  
- Each element extracted (e.g., Title, Problem Statement) is evaluated against its corresponding quality standards.  
- Again, the Gemini model is used.  
- The output includes: quality level (HIGH, MEDIUM, LOW), explanation, and improvement recommendations.  
  
8. `evaluate\_epic()`:  
- This function combines the above two:  
 a. Identifies all elements.  
 b. Evaluates each one.  
 c. Returns a complete evaluation report as a dictionary.  
  
9. `save\_to\_json()`:  
- Utility function that saves the final evaluation report to a `.json` file for record-keeping or further use.  
  
10. `main()`:  
- Sets up sample epics.  
- Creates an `EpicEvaluator` object.  
- Evaluates each sample epic.  
- Saves and prints results.  
  
Output:  
- For each epic, a JSON report is generated containing evaluations of all elements.  
- Each evaluation includes the quality rating, explanation, and suggestions for improvement.  
- Results are printed and also saved in the `evaluation\_results` directory.