## **QuoteCheck Coding Challenge: Create a Matching API in Python**

### **Objective:**

Your task is to build a RESTful API in Python that can take user input and attempt to match it against a predefined list of items. The API should return the best match along with a similarity score.

### **Requirements:**

1. **API Framework**: Use Flask or FastAPI to create the API.
2. **Input**:
   * The API should accept a POST request to an endpoint.
   * The request body should contain a JSON object with the following structure:

Example input:  
json  
{

"trade": "painting",

"unit\_of\_measure": "m2",

}

1. **Data**:
   * The API should have a predefined list of strings to match against. You can hardcode this list from the items.json file provided.
   * You can find a list of example inputs from the inputs.json file provided.
2. **Matching Logic**:
   * Implement a simple matching algorithm to find the closest match from the items list.
   * Create a similarity score based on how many fields matched, and how close they matched. You can consider the weighting of each field, i.e. a match on “trade” field might hold more weight than a match on “unit of measure” field.
   * Return the best match and the similarity score as a JSON response.
   * Note: it is not necessary to optimise the matching algorithm. We’re less concerned about the accuracy and preciseness of the match/similarity score, and more concerned about how the code is structured and how the requirements are considered.

Example input:  
json  
{

"trade": "painting",

"unit\_of\_measure": "m2",

}

Example output for the above example input:  
json  
{

"best\_match": {

“trade”: “Painting”,

“unit\_of\_measure”: “M2”,

“rate”: 15.0

},

"similarity\_score": 0.99

}

Example input:  
json  
{

"trade": "plumbing",

"unit\_of\_measure": "item",

}

Example output for the above example input:  
json  
{

"best\_match": {

“trade”: “Plumbing”,

“unit\_of\_measure”: “each”,

“rate”: 150.0

},

"similarity\_score": 0.6

}

1. **Error Handling**:
   * Handle cases where the input string is empty or not provided, returning a relevant error message.
   * Handle cases where there is no match found in the list, returning an appropriate response.
2. **Testing**:
   * Include at least 3 unit tests to verify the functionality of your matching logic and API endpoints.
   * Example test cases:
     + Test for exact match.
     + Test for partial match.
     + Test for no match.

### **Submission:**

Provide your solution as a GitHub repository link containing:

* The Python code for the API.
* Instructions on how to run the API locally.
* The unit tests.