Web page for translation and education

Creating a web page like the one you've described involves a few steps, integrating both frontend and backend development. Given your requirements, let's break down the approach into manageable parts. I'll outline the tools and technologies you might use and provide a high-level overview of how to implement this project.

1. Preparation

- **Data File**: Ensure your CSV file is well-structured with the Korean sentences, Spanish sentences, and explanations.
- **Python Code**: Your Python code for sentence embedding and similarity calculation will serve as the backend logic.

2. Choose a Web Framework

For the backend, which handles sentence embedding and similarity calculations, Python frameworks like Flask or Django are suitable. Flask is lightweight and easy to get started with, which makes it a good choice for this project.

3. Backend Development

- **Flask App Setup**: Start by setting up a basic Flask app that can handle web requests.
- **API Endpoints**: Create API endpoints to receive sentences from the frontend, run them through your sentence-embedding and similarity calculation code, and return the results.
    - One endpoint could be for receiving a sentence and returning a random sentence from your data file (along with its translation and explanation).
    - Another endpoint could be for receiving both the original sentence and the user's translated sentence, calculating the similarity, and then returning the similarity score along with the necessary details from your data file.

4. Frontend Development

You can use HTML, CSS, and JavaScript to create the frontend. Frameworks like React or Vue can be beneficial but are not strictly necessary for a simple project.

- **Page 1**: Create the initial page with two buttons as described. Use JavaScript to handle the button clicks and display the appropriate blocks for input and random sentences.
- **Page 2**: This page will show the results after the user submits their

translation. It will display the original sentence, the correct translation from your file, the user's translation, the similarity score, and the explanation.

5. Integration

- **Fetching Random Sentences**: Use AJAX (Asynchronous JavaScript and XML) to communicate with your Flask backend to fetch a random sentence and its details when the first page loads or when a user selects a direction (Korean to Spanish or vice versa).
- **Submitting Translations and Calculating Similarity**: Similarly, use AJAX to submit the user's translation to the Flask backend, calculate the similarity, and then display the results on the second page.

6. Deployment

Once your application is ready, you can deploy it to a web server. Platforms like Heroku, PythonAnywhere, or DigitalOcean are good starting points for deploying Flask applications.

Step-by-Step Implementation

Backend (Flask)

1. **Setup Flask**: Install Flask and set up a basic application structure.
2. **Create Endpoints**:
    - An endpoint to serve the main page.
    - An endpoint to randomly select and return a sentence pair from your CSV.
    - An endpoint to calculate and return the similarity score.

Frontend

1. **HTML/CSS**: Design your pages.
2. **JavaScript**: Implement the logic for handling button clicks, displaying sentences, sending and receiving data to/from Flask.

Integration

- Use fetchin JavaScript to interact with your Flask API, sending and receiving JSON data.