Backend Developer Challenge: Bento API Integration

Objective:

Build a RESTful API that integrates with Bento's delivery fee endpoint, applies a margin to the fee, and returns the new fee. Additionally, the API should be well-documented using Swagger and should demonstrate robust error handling. For testing purposes, you must also provide two endpoints:

1. One to fetch the new delivery fee.
2. One to fetch the last 10 requests stored in the database.

Instructions:

1. **Start a User Session**:
    - Go to the **Bento** website and create a user account or log in if you already have one.
    - You need to figure out how to obtain a **Bearer token** (the user session token) for authentication to interact with the Bento API. This may involve inspecting the network requests in your browser after login, or other methods you find suitable.
2. **Retrieve Your User UUID**:
    - Once you have obtained the **user session token**, make a `GET` request to the following endpoint to retrieve your user profile and UUID:

```
curl --location 'https://api.bento.ky/api/v1/users/profile' \
--header 'authorization: Bearer YOUR_USER_SESSION_TOKEN' \
--header 'accept: application/json'
```

    - Look for the `uuid` field in the response. This will be your **`ANONYMOUS_USER_UUID`**.
3. **Fetch Delivery Fee**:
    - Using the **`YOUR_USER_SESSION_TOKEN`** and **`ANONYMOUS_USER_UUID`** you obtained, make a request to the Bento API to fetch the delivery fee.

```
curl --location 'https://api.bento.ky/api/v1/delivery/fee' \
--header 'accept: application/json' \
--header 'authorization: Bearer YOUR_USER_SESSION_TOKEN' \
--header 'content-type: application/json' \
--data '{
    "addressFrom": {
        "coordinates": {
            "lat": 19.3331008,
            "lng": -81.3801101
        }
    },
    "addressTo": {
        "coordinatesAdjustment": {
            "lat": 19.280354483797733,
            "lng": -81.37386862188578
        }
    },
    "merchant": {
        "id": "8JbEqL0RTgHfRREvtSuO"
    },
    "user": {
        "uuid": "ANONYMOUS_USER_UUID"
    }
}'
```

4. **Process the Response**:
    - The response will contain the fee in cents. Convert it to dollars and apply a **13% margin** to the original fee. Example: If the response contains a fee of `813` cents (which is `8.13` dollars), the new fee should be `8.13 * 1.13 = 9.19`.
5. **Create an API for Your Solution**:
    - Build **two distinct endpoints**:
        1. **/delivery/fee** (or similar) to fetch the new delivery fee after applying the margin.
        2. **/requests/last** (or similar) to fetch the last 10 requests stored in the database.

Example response for the **/delivery/fee** endpoint:

```
{
  "originalFee": 8.13,
  "newFee": 9.19,
  "deliveryTime": 0,
  "distanceMeters": 6651.61,
  "message": null
}
```

Example response for the **/requests/last** endpoint:

```
[
  {
    "originalFee": 8.13,
    "newFee": 9.19,
    "deliveryTime": 0,
    "distanceMeters": 6651.61,
    "message": null,
    "timestamp": "2025-03-13T10:00:00Z",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0
  },
  ...
]
```

6. **Document the API**:
   - Use **Swagger** to document your API. Your documentation should include:
     - **API endpoints** (e.g., `/delivery/fee`, `/requests/last`)
     - **Request format** (e.g., address parameters)
     - **Response format** (with examples)
     - **Authentication** (how to provide the Bearer token)
7. **Store the Data**:
   - Store the following information in a database (e.g., Firebase Firestore):
     - **Original fee** (in dollars)
     - **New fee** (after margin applied)
     - **Delivery time**
     - **Distance meters**
     - **Coordinates**
     - **User UUID**
     - **Merchant ID**
     - **Timestamp**: Store the timestamp of when the request was made (e.g., in ISO 8601 format).
     - **User Agent**: Store the user agent of the request, which can provide additional context about how the API is being accessed.
8. **Error Handling (Differential)**:
   - Proper error handling is a key part of this challenge. We will evaluate how you handle:
     - Invalid or expired tokens.
     - Failed API requests (e.g., network errors, timeouts).
     - Unexpected or malformed response formats.
     - Missing or inconsistent data.
   - Ensure your API returns meaningful error messages with appropriate HTTP status codes. Handle edge cases and provide clear, actionable error responses.
9. **Provide Access to the API and Documentation**:
   - **Host your code on GitHub** and share the repository link with us for code evaluation.
   - Provide a link to your Swagger documentation for review.
   - Ensure your API is accessible for testing purposes.
10. **GitHub and Commits**:
    - We will review your code by looking at the commits in your GitHub repository. It's **all about the journey**, so make sure your commit messages are clear, and the process is easy to follow.

Evaluation Criteria:

- **API Implementation**: Correct integration with the Bento API and proper application of the margin.
- **Documentation**: Clear and comprehensive Swagger documentation.
- **Error Handling**: Robust error handling, including token validation and unexpected responses.
- **Code Quality**: Clean, maintainable code.
- **Functionality**: Properly storing data, returning the expected response, and having two clear endpoints.

Timeline:

- You have **3 to 5 days** to complete this challenge.