

Task: Comprehensive User Management API with Global Error Handling and File-Based Data Storage

Scenario

You are building a multilingual API for managing users. Instead of a full database, user data will be stored in a file. Your task is to implement global error handling, FluentValidation, and middleware, while also managing user data using file-based storage.

Task Requirements

1. Global Error Handling Middleware

- Create middleware to handle unhandled exceptions and return standardized responses.
- Include detailed error information.

2. Input Validation with FluentValidation

Define validation rules for the `User` model:

User should have: Id, first name, last name, email, age and etc.

Add logical validation rules, example validation rules:

- `FirstName` and `LastName` are required.
- `Email` must follow a valid email format.
- `Age` must be between 18 and 120.

3. File-Based Data Storage

- Store user data in a JSON file (e.g., `users.json` or `users.txt`).
- Create utility methods for:
 - Reading users from the file.
 - Writing users to the file.

Example `users.json` structure:

```
[
  { "Id": 1, "FirstName": "John", "LastName": "Doe", "Email": "john.doe@example.com",
    "Age": 25 },
  { "Id": 2, "FirstName": "Jane", "LastName": "Smith", "Email": "jane.smith@example.com",
    "Age": 30 }
]
```

4. Localization

- Localize validation error messages for at least two languages (e.g., English and Georgian) using `.resx` files.
- Example:

- Georgian: "Email is required." → "ელფოსტა სავალდებულოა."
- English: "Email is required."

5. Culture Middleware

- Implement middleware to set the request culture based on the `Accept-Language` header.
- Ensure the selected culture is applied to validation and error messages.

6. API Endpoints

Implement the following endpoints:

- `POST /users`: Adds a new user to the file after validation.
- `GET /users/{id}`: Retrieves a user by their `id`. Return `404` if the user does not exist.
- `PUT /users/{id}`: Updates an existing user's details. Validate the data before updating.
- `DELETE /users/{id}`: Deletes a user by their `id`. Return `404` if the user is not found.
- `GET /users`: Returns all users from the file.
- `GET /error-demo`: Simulates an unhandled exception.

7. Error Response Format

Design a consistent error response model:

```
{
  "status": 400,
  "message": "Validation failed.",
  "errors": ["FirstName is required.", "Email must be a valid email."]
}
```

8. Error Logging

Implement logging for file read/write errors.

9. Swagger UI Integration

- Add Swagger UI to the project for API documentation and testing.
- Configure Swagger to support different request cultures (e.g., through `Accept-Language` headers).

Bonus Tasks

- Add search functionality to `GET /users`, allowing filtering by `FirstName` or `LastName`.

Topics to research:

- How to test localization using the `Accept-Language` header.
- How to test the API endpoints using SwaggerUI.