Solution Documentation

Problem Statement:

The task involved creating a REST API using C# ASP.NET Core to manage entities with CRUD operations and additional features like searching and filtering. The provided Entity model consists of various properties such as addresses, dates, names, etc.

Solution Overview:

1. Controller Setup:

- Created an `EntitiesController` to handle API endpoints related to entities.

- Defined endpoints for listing entities, fetching a single entity by ID, searching entities, and filtering entities based on gender, dates, and countries.

2. Data Mocking:

- Used a simple list of entities as mock data for demonstration purposes.

- This data is stored in-memory and is used to simulate database interactions.

3. Endpoint Implementation:

- Listing Entities Endpoint: Implemented a GET endpoint to fetch a list of all entities.

- Single Entity Endpoint: Implemented a GET endpoint to fetch a single entity by its ID.

- Searching Entities Endpoint: Implemented a GET endpoint to search for entities based on specific fields such as names, addresses, etc. This endpoint supports a query parameter to specify the search string.

- Advanced Filtering Endpoint: Implemented a GET endpoint to filter entities based on optional query parameters such as gender, start date, end date, and countries.

4. Handling Null Values:

- Added null checks to ensure robustness when accessing properties of entities, names, addresses, dates, etc.

- This helps prevent `NullReferenceException` and `ArgumentNullException` by ensuring that the source collections are not null before performing LINQ operations.

Reasoning Behind the Approach:

1. Simplicity:

- The solution prioritizes simplicity and clarity to ensure easy understanding and maintainability.

- Each endpoint is implemented using straightforward logic and follows RESTful conventions.

2. Error Handling:

- Implemented error handling to return appropriate status codes and error messages when necessary (e.g., returning 404 Not Found when a requested entity is not found).

3. Code Readability:

- Code readability is emphasized by using meaningful variable names, commenting where necessary, and breaking down complex logic into smaller, manageable parts.

- Null checks and error handling are explicitly included to enhance code readability and prevent unexpected runtime errors.

4. Mock Data Approach:

- Due to the absence of a real database, mock data is used for demonstration purposes.

- This approach allows for quick setup and testing without the need for external dependencies.

5. Flexibility:

- The solution provides flexibility by allowing optional query parameters for filtering and searching.

- Users can filter entities based on specific criteria, enhancing the usability of the API.

Conclusion:

The provided solution offers a functional implementation of a REST API for managing entities in a C# ASP.NET Core environment. It addresses the requirements outlined in the problem statement while prioritizing simplicity, error handling, and code readability. Additionally, the documentation provides insights into the rationale behind the approach taken, ensuring clarity and understanding for future reference and collaboration.