Merging archive files server application using ASP.NET Core

Github repository link
https://github.com/AyaBebawe/MergingArchiveFiles.git

OVERVIEW

The application is using ASP.NET Core as a runtime It offers a RESTful API which Receives an arbitrary number of .ZIP-files in one request using **Form-Data** format.

.ZIP-files are received in **Form-Data** format which is useful when transmitting and dealing with large files unlike **'x-www-form-urlencoded'** which is suitable only for simple form submissions where the data is URL-encoded and included in the body of the HTTP request. The reason why **BSON** was not used is that it's not designed for transmitting files over HTTP, as **BSON** is tailored for representing structured data, not preferred when transmitting raw binary files.

So when a client uploads multiple ZIP-files, the content type of the request is received to our server as multipart/form-data. Then IFormFile Interface is used as it represents a file sent with the HTTP request and it is used to work with this encoding and simplifies the process of handling file uploads.

Then the API combines the files of these archives in an efficient way in one .ZIP-file and returns it as a downloadable Zip-File.

SPECIFICATIONS

The project also supports some non-functional specs as specified below:

1. **Security**
   - 1.1. Limit File Types: Allow only specific file types to be uploaded. This can prevent users from uploading potentially harmful files.
     - By using Implementing function to check input files' type **(ValidateZipFiles())**
   - 1.2. Limit request sizes which are crucial for preventing denial-of-service(DoS) attacks and ensuring the stability of the application.
     - By using [**RequestSizeLimit(3L * 1024 * 1024 * 1024)**] which is 3GB.
2. **Performance Enhancement**
   - 2.1. Logging Errors in LogFile instead of writing it on console
     - By configure logging to use Serilog and use **Ilogger** interface
   - 2.2. Utilize asynchronous I/O operations to improve responsiveness, especially when dealing with file I/O.

- One example by using **await entryStreamCopyToAsync(newEntryStream)** within the loop to potentially improve i/o performance when copying files to the target archived files.

2.3. Resources Utilization Limit:
- by the use of "**using**" statements to ensure the proper disposal of resources and to help manage the lifecycle of objects that implement the IDisposable interface. This can have positive effects on memory management and the overall performance.

3. Reliability

3.1. Error handling
- Error handling is implemented to handle various scenarios as:
  - Invalid file types
  - Single file entry
  - No file uploaded

3.2. Logging Errors in LogFile to be easy in debugging and trace any error.

4. Maintainability

4.1. Code Quality:
- Follow clean code standards and best practices.
- Use inline comments to explain different sections of code.

4.2. Code Separation:
- Divide Code to two layers: **Controller layer** and **Service layer** (business layer) to facilitate future updates and maintenance.

Expected Input:

The Frontend/Client needs to provide the following input:

- **HTTP Method and Endpoint**:
  - The frontend needs to send an **[HttpPost]** request to the endpoint defined in your controller, which is **/api/MergeArchiveFiles**

- **Request Headers**:
  - The request should have the **Content-Type** header set to **multipart/form-data.**

- **Request Body**:
  - The request body should be formatted as **multipart/form-data**, and it should contain a list of .zip files in a **key** with a unique name, which is **"zipFiles"** as specified by the [FromForm] attribute in the controller method parameter.

Expected Responses :

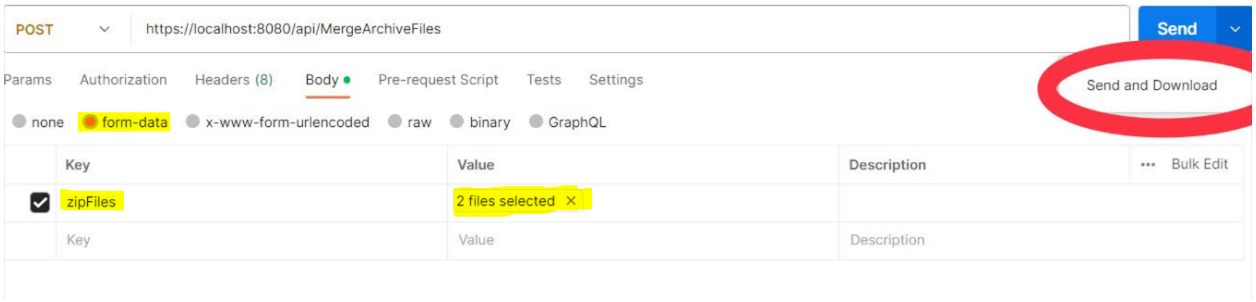
1. **Successful Response returns (Status: 200 OK)**
    - If the merging operation is successful, the API should return a **FileContentResult**.This result represents a file that can be **downloaded** by the client. The response will have a **Content-Type of "application/zip"** and include the merged ZIP file as the response body.

2. **Bad Request Response (400 Bad Request):**
    - If the input provided by the client is invalid (e.g., incorrect file format, not enough files,no files uploaded), the API should return a **BadRequestObjectResult** with an error message indicating the reason for the failure.
3. **Internal Server Error Response (500 Internal Server Error):**
    - If an unexpected exception occurs during the merging process, the API should return a StatusCodeResult with a status code of 500 and an error message"**This should not have happened. We are already working on a solution."**

Postman Sample :

(**localhost** is used here with URL: https://localhost:8080/api/MergeArchiveFiles)

Request:



Response : zip File can be downloaded by the client