**System for Asynchronous Image Processing**

This system is built to handle the asynchronous processing of images sourced from a CSV file. It compresses the images and stores the processed data in a database while providing APIs for users to monitor the processing status.

**Components:**

- Main Application (`server.js`): Acts as the core of the application by starting the Express server, establishing the database connection, and defining the routes.

- API Handlers (`api/imageApis.js`): Manages the endpoints for CSV file upload and status inquiries.

- Business Logic (`businessLogic/imageQuality.js`): Implements the core logic for image processing and webhook handling.

- imageProcessor(`imageProcessor/imageProcessor.js`): Operates asynchronously to download, compress images, and update the database with the new URLs.

- Helper Functions (`helper/csvHandler.js`): Responsible for parsing and validating the data extracted from CSV files.

- Database Models (`tables/Images.js`, `models/RequestStatus.js`)\*\*: Mongoose schemas and models used to structure and store product information and processing requests.

- Database Connection (`connection/mongo.js`)\*\*: Manages the setup and connection to the database using Mongoose.

- Testing Route (`userRoute.js`)\*\*: Exclusively for testing functionalities, not central to the assignment.

**FLOW:**

1. Uploading the CSV File:

- The user initiates an upload of a CSV file containing details about images and their corresponding image URLs.

- The file is processed by the `uploadCSVFile` middleware through Multer.

- The `uploadCSV` function validates the CSV content, parses the data, and saves the product and request details to the database.

- A unique request ID is generated, which triggers the asynchronous image processing workflow.

2. Handling Image Processing:

- The `processImagesAsync` function within `imageProcessor.js` retrieves the images from the URLs provided, compresses them with Sharp, and saves the compressed files to a specified location.

- The new URLs are then updated in the `Image` model, and the status of the request is modified in the `RequestStatus` model.

3. Checking Processing Status:

- Users can check the progress of their image processing requests by calling the `checkStatus` function in `imageApis.js` using their unique request ID.

­­