**Image Processing System**

The image-processing system is designed to process images asynchronously from a CSV file, compress them, and store the output in a database. It also provides APIs to check the processing status.

**Components/Files we have used in this Assignment:**

* **API Server (server.js):** The main entry point of the application. It initializes the Express server, connects to the database, and sets up routes.
* **Controllers (controllers/imageController.js):** Manages the API endpoints for uploading the CSV file and checking the processing status.
* **Controllers (controllers/userController.js):** Used for testing the functionality used not a part of the Assignment.
* **Services (services/imageService.js, services/webhookService.js):** Handles the business logic for processing images and managing webhooks.
* **Workers (workers/imageWorker.js):** Processes images asynchronously, compresses them, and updates the database with output URLs.
* **Utilities (utils/csvHandler.js):** Provides helper functions to parse and validate CSV data.
* **Models (models/Product.js, models/Request.js):** Defines Mongoose schemas and models for storing product and request data.
* **Database** **Configuration (config/db.js):** Handles the database connection setup using Mongoose.
* **Route (userRoute.**js): Used for testing the functionality used not a part of the Assignment.

**Workflow of the Assignment:**

1. **CSV Upload:**

* A user uploads a CSV file containing product details and image URLs.
* The uploadCSVFile middleware processes the CSV file using multer.
* The uploadCSV function validates the file, parses the data, and stores the product and request details in the database.
* The system generates a unique request ID and initiates asynchronous image processing.

1. **Image Processing:**

* The processImagesAsync function in imageWorker.js fetches images from URLs, compresses them using sharp, and saves them to a specified output path.
* The output URLs are stored in the Product model, and the request status is updated in the Request model.

1. **Status Check:**

* The checkStatus function in imageController.js retrieves the status of a specific request using the request ID.

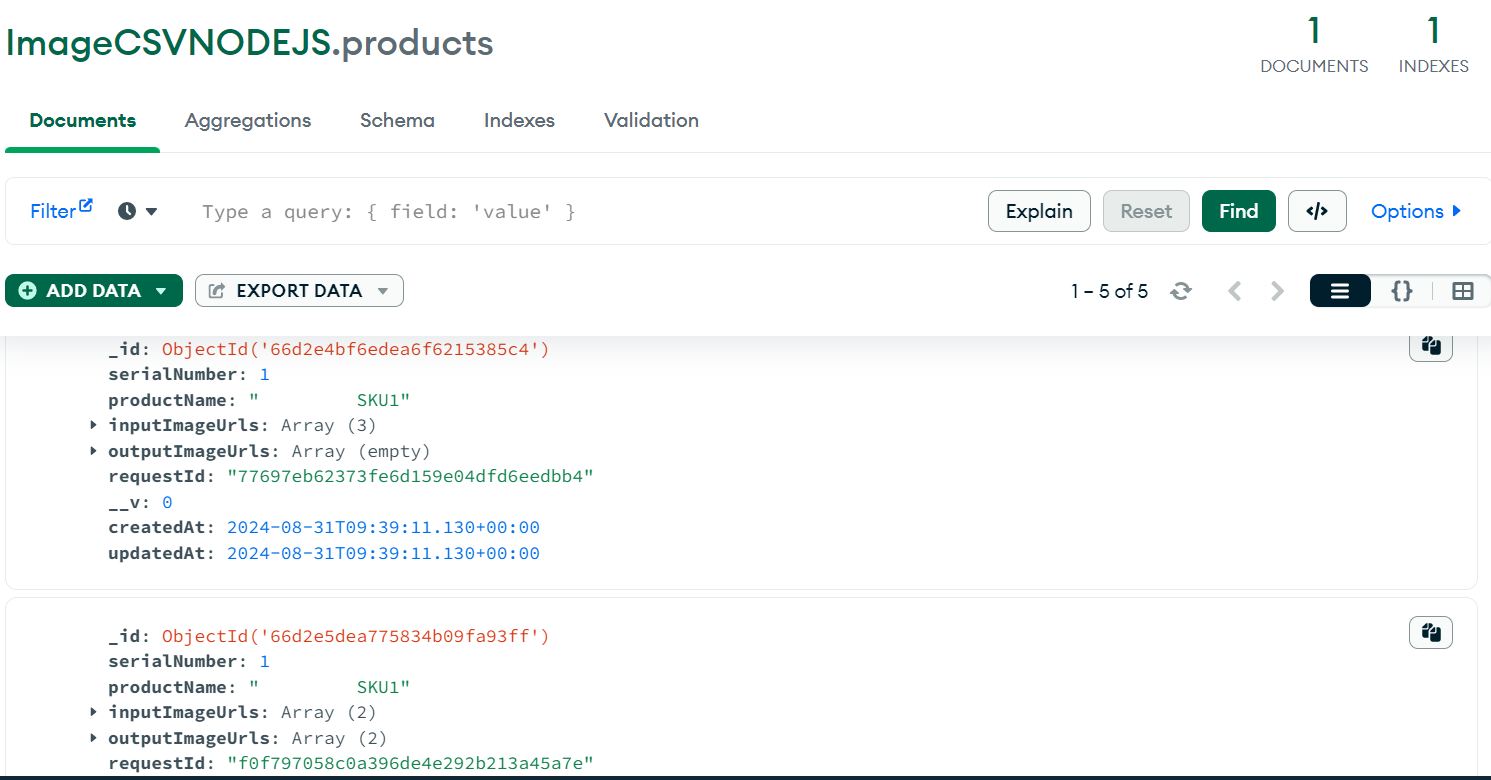
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Example of data in the database (MongoDB):**

****

