## **Project Overview**

This project is a microservices-based system built using NestJS for the backend and React.js for the frontend. The goal is to handle file uploads (CSV and Excel), process the data from those files, save it to a MySQL database, and allow the user to search, update, delete, and export that data. The system also has real-time notifications for batch processes like file exports.

### Tech Stack

* Backend: NestJS, TypeORM, BullJS (for background job handling), Redis (for queues), GraphQL (for communication between services and frontend).
* Frontend:React.Js & Tailwind CSS(for styling).
* Database: MySQL.
* Notifications: WebSockets (using Socket.IO).
* Docker: To deploy each service in separate containers.

### **Services I Implemented**

#### **1. Upload Service:**

* What it does: This service handles file uploads (CSV or Excel) from the frontend. Once a file is uploaded, it is saved to a local directory, and a BullJS job is created to process it.
* Tech used: NestJS, BullJS for queue management, Redis for handling job queues, GraphQL for communication.
* What’s done: The service successfully uploads files, saves them locally, and pushes the file path to the Bull queue.
* What’s missing: Nothing here. This service is complete.

#### **2. Process Service:**

* What it does: This service reads the file path from the Bull queue, processes the file (checks if it’s CSV or Excel), and extracts the data. The data is then saved to the MySQL database with the calculated ageOfVehicle.
* Tech used: NestJS, BullJS, Redis, TypeORM (for database interaction), GraphQL.
* What’s done: The service reads from the queue, processes the file, and saves the data to the database with the calculated vehicle age.
* What’s missing: Nothing major.

#### **3. Vehicle Service:**

* What it does: This service provides CRUD operations for vehicle records. It also handles pagination, sorting, and search functionality for vehicle data.
* Tech used: NestJS, TypeORM (for MySQL database), GraphQL (for data queries).
* What’s done: Basic CRUD operations are implemented. Data is saved correctly to the database.
* What’s missing: I still need to implement pagination, search, update, and delete features in the frontend to fully connect with this service.

#### **4. Export Service:**

* What it does: This service allows users to export vehicle data based on filters like car make, model, or age.
* Tech used: NestJS, GraphQL.
* What’s done: Service is working. Users can set criteria (like cars over 5 years old) and the batch job will generate the CSV file.
* What’s missing: I need to finish connecting this service to the frontend so users can trigger exports, download files, and receive notifications when the export completes.

#### **5. Notification Service:**

* What’s missing: This service sends real-time notifications to the frontend whenever a batch job (like an export) is completed. Notifications will stack on the frontend and disappear after 10 seconds.
* Tech used: WebSockets (Socket.IO for real-time communication), GraphQL.
* What’s done: The basic WebSocket setup is in place. It can send notifications when batch jobs are done.
* What’s missing: I still need to integrate this with the frontend so notifications show up in real-time and disappear after 10 seconds.

#### **6. API Gateway:**

* What it does: This is the entry point for all frontend requests. It connects the frontend with the various backend services (Upload, Process, Vehicle, Export, and Notification) via GraphQL.
* Tech used: NestJS, GraphQL, Microservices architecture.
* What’s done: The API Gateway successfully routes requests between the frontend and backend services.
* What’s missing: There’s currently a blocker in making a request to the file over the GraphQL API gateway to the backend.

### **Frontend (React.js)**

* UI Components: I’ve used React.js for the frontend and styled it using Tailwind CSS. The UI includes buttons for uploading files, exporting data, and a table to display vehicle records.
* Data Display: The table on the frontend displays vehicle data fetched from the Vehicle Service. It’s styled and functional but needs more features like pagination, search, and actions to buttons for updating and deleting records.
* File Upload: The upload button works, allowing users to select a file and send it to the backend for processing.
* File Export: The Export button works, you can choose how old cars should be listed and it will saved in local.

### **Approach I Took to Create This Project**

1. Research: I first researched NestJS, BullJS, Redis, and other technologies to get a good understanding of how they work. I also looked into how to handle file uploads and process data from CSV and Excel files.
2. Initial Project Setup: I created a single project to try out basic CRUD operations, file uploads, and background processing with BullJS. This helped me get hands-on experience with the tools.
3. Splitting the Project: After getting comfortable with the setup, I divided the project into separate services: Upload Service, Process Service, Vehicle Service, Export Service. Each service handles a specific task and communicates with the others through GraphQL.
4. Backend Development: I started by building each service one by one, making sure they all work independently. I connected them to Redis and BullJS to handle background jobs and TypeORM for database interaction.
5. Frontend Development: After the backend services were ready, I moved on to creating the frontend with React.js. I built the table UI, upload button..
6. Integrating Services: Finally, I connected the frontend with the backend services through the API Gateway and GraphQL, ensuring the data flows properly between the components.

### **What’s Left to Do**

* Micro-service Issue: I created api-gateway and should connect with other services.
* Notification: I implemented frontend ui Implementation In backend I got some errors. It need to be fixed
* Export: I exported file from database as .cvs file and it save in local storage. I should Implement download function.

### **Conclusion**

The project is a microservices-based system with file uploads, background job processing, and real-time notifications. I’ve implemented the core services and the basic functionality but still need to finish a few key features on the frontend, like pagination, search.