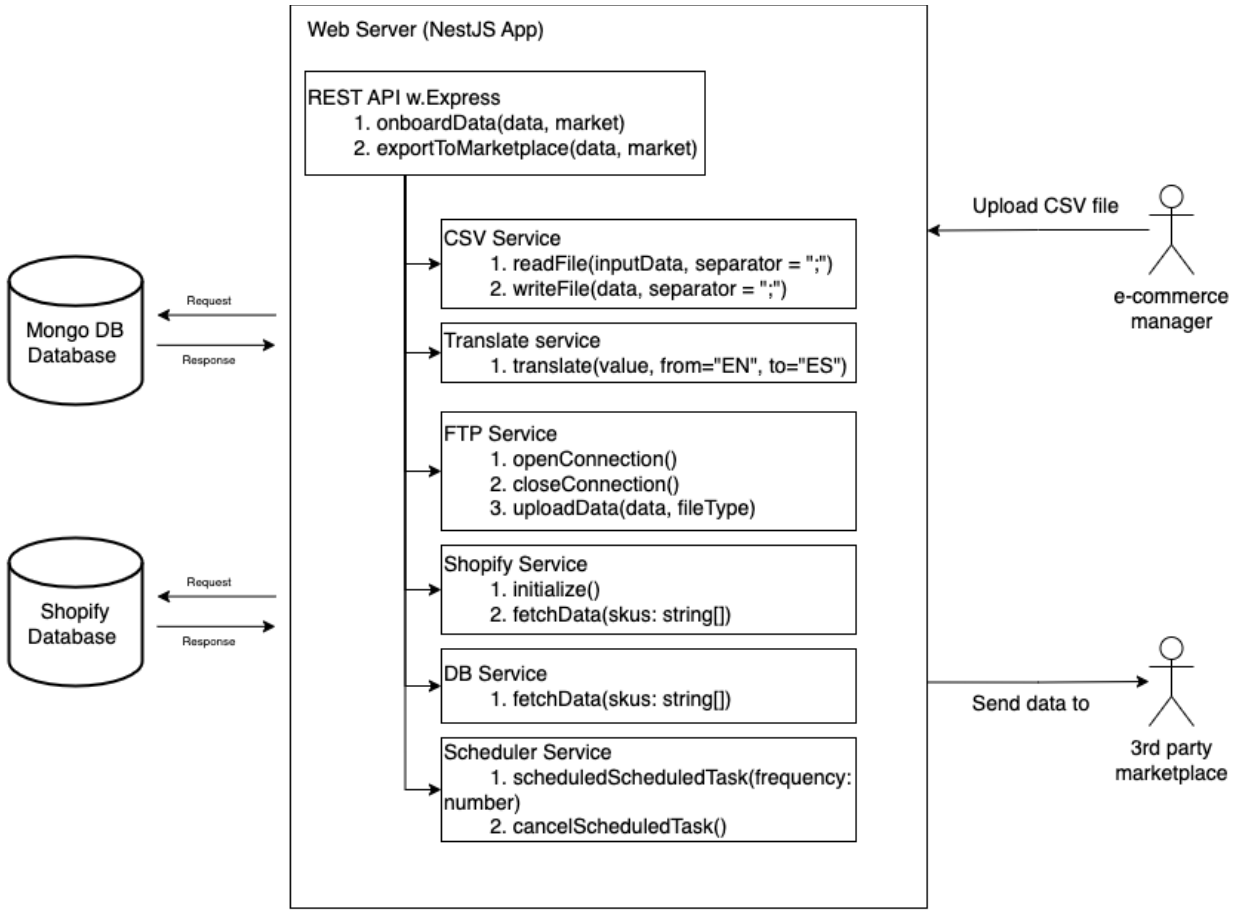
Project Description

Automate the process of exporting product data to a marketplace.

Application Design

Figure 1 provides a high-level detailing for the application architecture. The application consists of 3 main parts:

1. Databases
2. Web Server
3. Actors (e-com manager, marketplace, etc.)



Assumptions:
1. Shopify is assumed as a database entity in this use case as the primary function is to fetch data
2. The marketplace is considered as a single actor for simplicity, but could be multiple marketplaces

Figure 1: Application Architecture (High level)

The application is structured into **Controllers** and **Services**. **Controllers** are responsible for handling incoming HTTP requests and returning responses to the client. **Services** will be responsible for providing some data, which can be reused across the application.

In the case of our application, we have 6 main services:

1. **CSV Service**
   The CSV service as the name suggests is designed to deal with CSV files. In this case, the 2 primary use-cases are: reading and creating CSV files.

2. **Translate Service**
   The translate service makes use of the *"translate"* package with the DeepL engine to translate product information from the base language (English) to the language required by the marketplace i.e., Spanish.

3. **FTP Service**
   The FTP service is defined to connect and upload data to a remote server.

4. **Shopify Service**
   The Shopify Service as the name suggests exists to fetch data from the Shopify database.

5. **DB Service**
   The DB Service contains functions to query data from the MongoDB instance. All database queries are defined and executed over here.

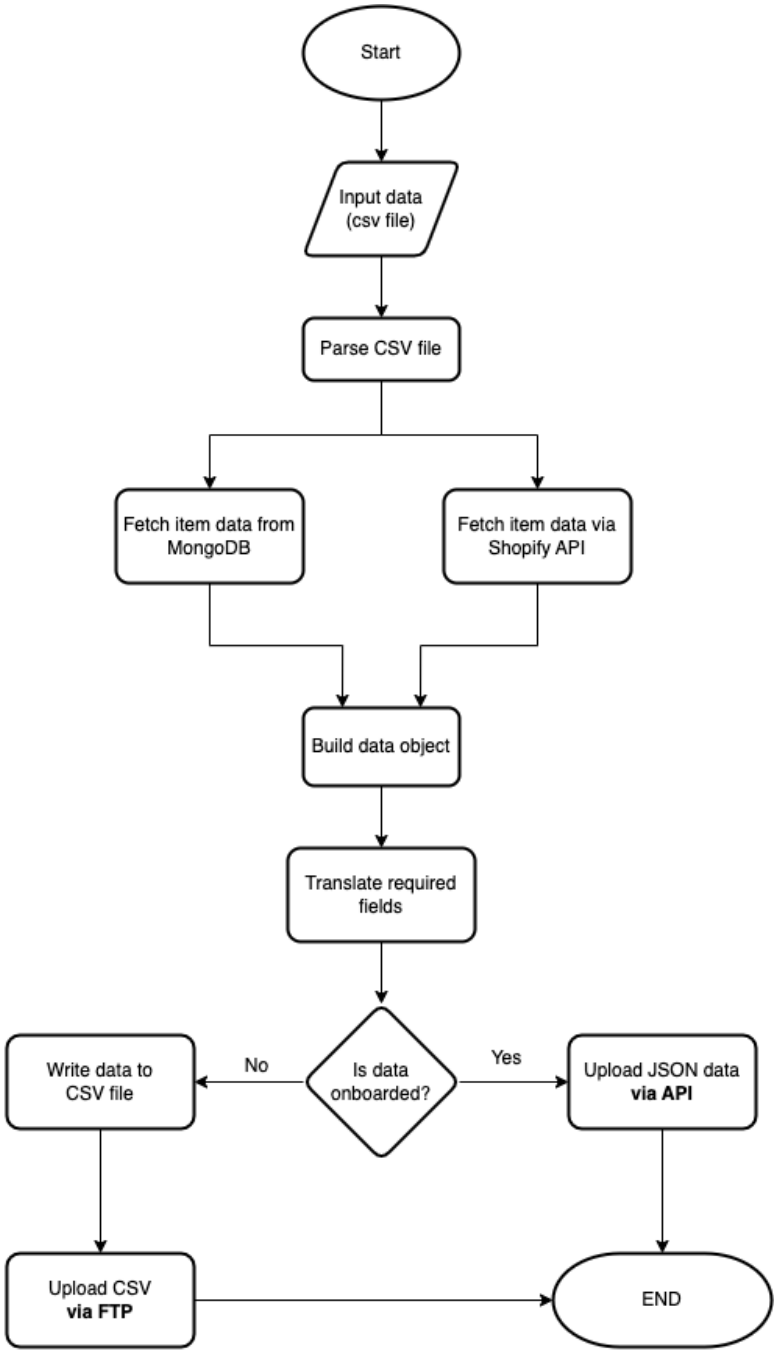
6. **Scheduler Service**
   The scheduler service is to run a task at a fixed interval or cancel a pre-existing task.

Process Documentation

Figure 2 provides us a basic workflow for the server-side script. The input CSV file is parsed into a valid JSON object to retrieve the SKU IDs provided. The SKU IDs are used to fetch data from the internal database (MongoDB instance) and Shopify APIs.

This data is aggregated to build the output data using the predefined data model. Localization actions are performed to translate the product data into the language required for the marketplace.

If data is being onboarded for the first time, the data is written to a CSV file and uploaded to the marketplace via FTP. If data is previously onboarded, then the updated output is uploaded via the marketplace API.



Note: The flowchart provides an illustration for a **single pass**. This process will run periodically based on the update frequency.

Figure 2: Flowchart to visualize process workflow

Project Setup

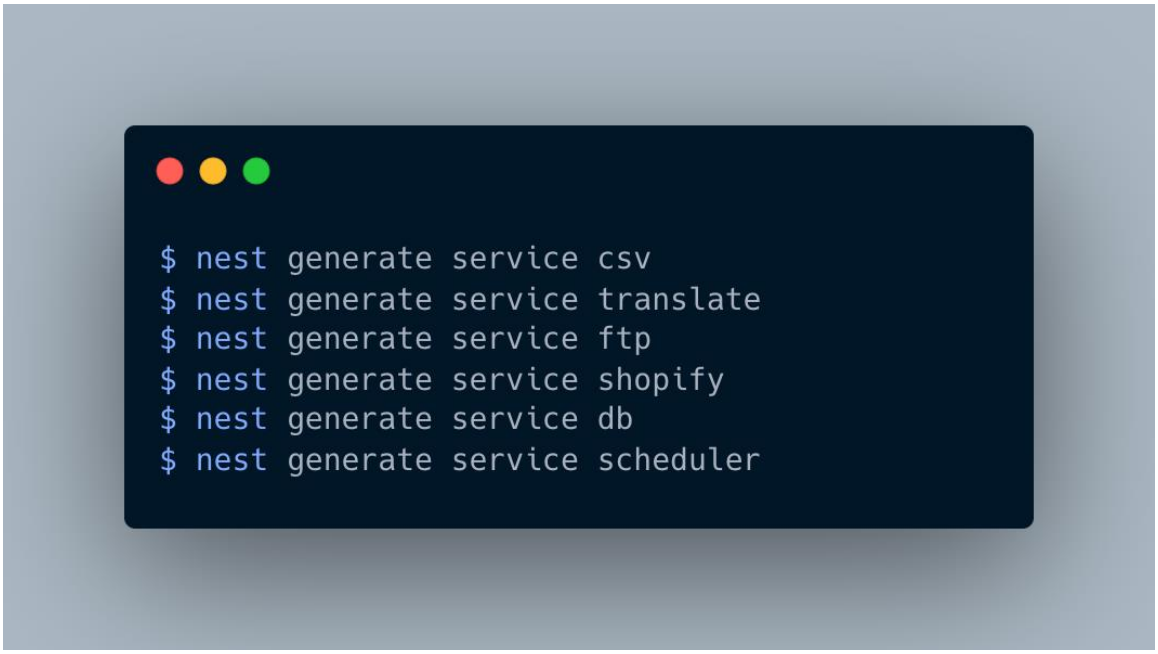
1. **Create a NestJS application**

```
$ npm i -g @nestjs/cli
$ nest new project-name
```

2. **Install the required packages**

Package	Purpose	Source
translate	Used for translating text using the Deepl API	link
ftp	Add support for FTP connectivity to upload files to a remote server	link
csv	Parse & generate CSV files	link
shopify-api-node	Access the Shopify inventory databases	link
Mongoose	Add connectivity to MongoDB to read/write data from the database	link

3. **Create Services**



4. **Steps to launch the app**
   npm install && npm run start

The entry point for the app is **app.module.ts**. The AppModule is configured to handle a buffer file input with a middleware (refer filebuffer.middleware.ts).

The endpoint for uploading the CSV file is **/api/upload/:filename.** The process runs asynchronously, so the user does not have to wait for the data export for the request to complete.

Assumptions made:

1. The id returned by the MongoDB instance is the **same** ID used by Shopify.
2. Data refresh is assumed to run only for the product data initially uploaded and not the entire product catalog.
3. Product schema is based on data provided and examples provided in Shopify documentation. This will vary in a real-world implementation.

Blockers:

1. Not sure where data attributes like FIT, WIDTH, LENGTH are coming from.

2. Unsure if the update is only for just the SKU IDs in the initial file or from the entire data set.

3.
4. API upload is intentionally left blank as I did not want to assume the request structure (body and headers). But it's a simple HTTP request.

```
    let result: Boolean = await this.csvService.writeToCSV( ${filePath}/${fileName} , exportProd

    if (result && isFirstUpload) {
        // Upload CSV file via FTP
        this.ftpService.uploadFile(fileName, filePath);
    } else {
        // Update `exportProducts` via API
    }
} catch (error) {
    this.logger.error(error, null, 'processCSVUpload');
```

5. Resuming the update process on server restart is not implemented

Resources

1. https://docs.nestjs.com
2. https://docs.nestjs.com/techniques/database
3. https://dev.to/this-is-learning/introduction-to-nestjs-services-2l5f
4. https://docs.nestjs.com/controllers
5. https://medium.com/globant/crud-application-using-nestjs-and-mongodb-99a0756adb76
6. https://gist.github.com/jonilsonds9/efc228e34a298fa461d378f48ef67836
7. https://docs.nestjs.com/techniques/task-scheduling

8. https://blog.logrocket.com/build-shopify-app-with-node-js/