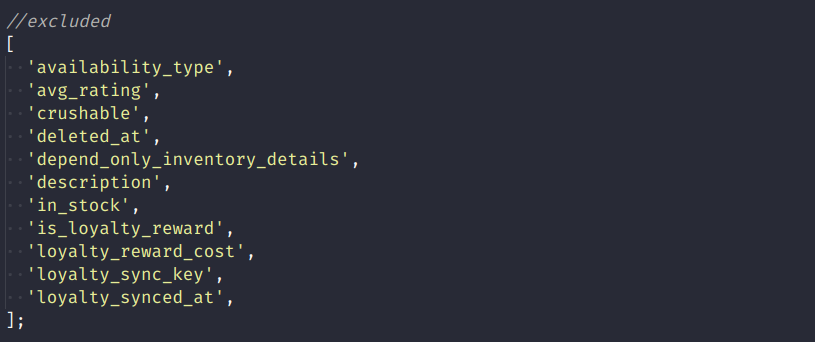
**Guide to Migrating Algolia Index with Different Schemas Using JavaScript**

**Problem Statement:**

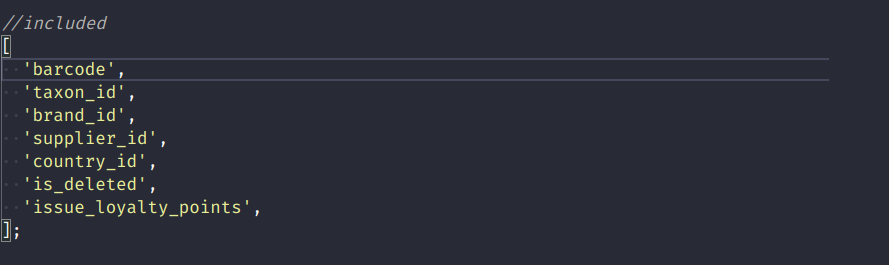
Since We have two indexes in Algolia, v1 of the product schema and v2 of the product schema. Some of the fields in the v1 schema might not exist in the v2 schema, and some new fields might be added to the v2 schema. Additionally, some fields might be renamed in the v2 schema.

This presents a challenge when migrating data from the v1 index to the v2 index since the schema has changed.

For Example: v1 Product schema as the following fields excluded in v2:



For Example: v2 Product schema as the following new fields included:



To overcome this challenge, we can adopt a strategy that involves transforming the records from the v1 schema to match the v2 schema before migrating them to the v2 index. This strategy ensures that the migrated data conforms to the v2 schema.

**Overview of the Approach:**

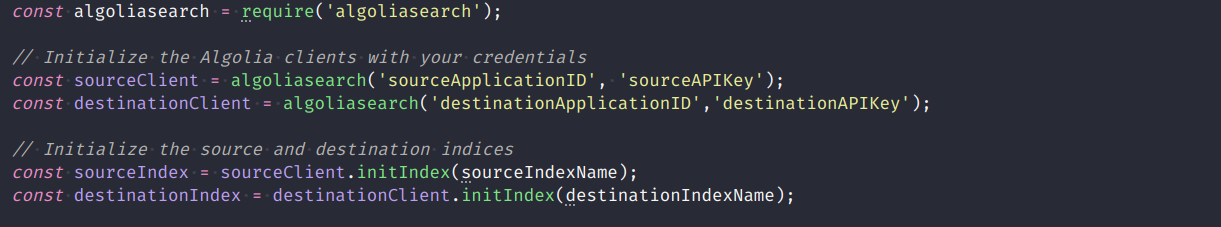
The approach to migrate data between two Algolia indices involves these steps:

1. Initialize the Algolia clients with your credentials.
2. Initialize the source and destination indices.
3. Define a function to transform the records in the source index to the format of the destination index.
4. Set up the search parameters for the source index.
5. Retrieve the records from the source index in batches.
6. Transform the retrieved records using the function defined in step 3.
7. Save the transformed records to the destination index.

The code for this approach is written in JavaScript and uses the algoliasearch library to communicate with the Algolia search engine.

**Setting up the Algolia clients:**

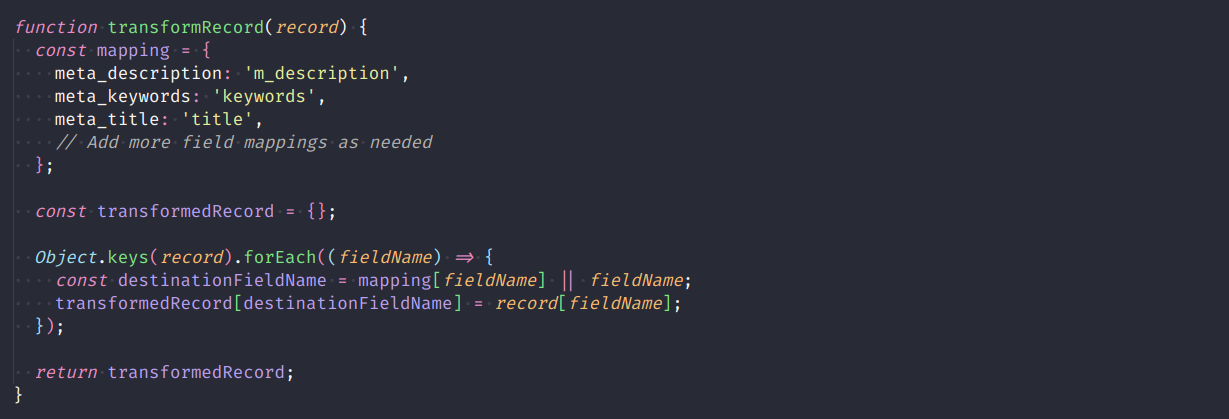
First, you need to initialize the Algolia clients for the source and destination indices using the Algolia API credentials, as well as specify the names of the source and destination indices:



**Transforming the data:**

Next, you need to define a function that will be responsible for dynamically mapping the fields of a record from the source index to match the fields of the destination index.

You can define this function to map fields as needed, based on your specific use case. Here's an example function that maps a few fields:



This function takes a single record from the source index as input and returns a transformed record that maps the fields of the source index to match the fields of the destination index.

In this example, the function maps the fields meta\_description, meta\_keywords, and meta\_title in the source index to m\_description, keywords, and title, respectively, in the destination index. You can add more field mappings as needed.

The mapping object defines the field mappings between the source and destination indexes, where the **keys represent** the fields of the **source index**, and the **values represent** the corresponding fields of the **destination index**.

The Object.keys method is used to iterate over the fields of each record, and the mapping object is used to map the fields to the corresponding fields of the destination index.

If a field is not found in the mapping object, the original field name is used as the destination field name.

**Excluding unused/deprecated fields:**

Some fields are excluded from the transformation because they are not needed in the new schema. These fields are specified in an array called **excludedFields**.  
  
Note - Below mentioned are fields are sample.



**Adding new fields:**

Some new fields are included in the transformation because they are required in the new schema. These fields are specified in an array called **newFields**. Initially, the values for these fields are set to null for testing purposes.

Note - Below mentioned are fields are sample.



After defining the **transformRecord** function, you need to define a function that will be responsible for migrating the records from the source index to the destination index. Here's an example function that does this:

