[Bulk Upload service]

Functional Requirement

1. Seller able to upload CSV/Excel file containing list of brands, collection, products, store location.
2. Seller able to see progress of file upload and errors if any.

Non-functional

1. Scalable for large E comm website
2. Available, fault tolerant

Bulk Upload

1. Download the sample template file by clicking the download button below .

Download File

Profile

Download Template

2. Attach the updated file once it is filled with all the required fields.

Select a file

or drag and drop csv here

Accepted file types: .csv

Upload source

History

Refresh | **Bulk Upload**

Search here | All

1.csv 2 brands uploaded successfully Upload by Anurag Kishore | 9 Dec 2024	COMPLETED
1.csv 2 brands uploaded successfully Upload by Anurag Kishore | 5 Dec 2024	COMPLETED
1.csv 2 brands uploaded successfully Upload by Anurag Kishore | 28 Nov 2024	COMPLETED
1.csv 2 brands uploaded successfully Upload by Anurag Kishore | 25 Nov 2024	COMPLETED

Feedback

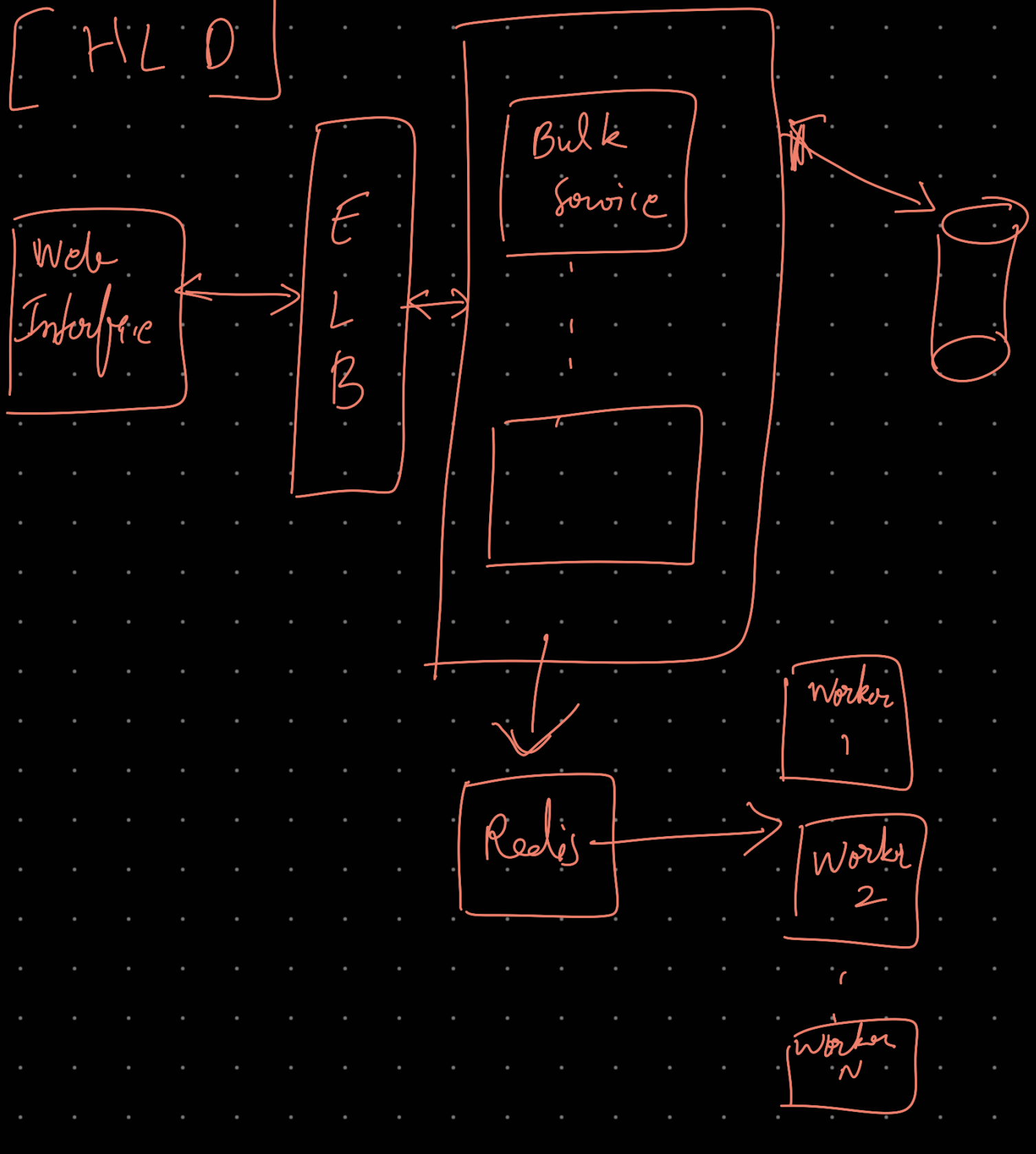
Upload progress

Capacity Estimation

File size ⇒ 20 MB → Large file

20,000 – 50000 records per file

[HLD]



1. **File Upload:**
    - User uploads the CSV file.
    - The backend validates the file and stores it in cloud storage.
    - The system triggers a background process to split and enqueue rows for processing.

2. **Splitting and Enqueuing:**
    - Split the file into smaller chunks (e.g., 1000 rows per chunk).
    - Push each chunk or row as a message to the queue.

3. **Row Processing:**
    - A pool of worker services consumes messages from the queue.
    - Each worker processes rows (e.g., parsing, validation, transformation).
    - The worker performs database updates in small batches for efficiency.

4. **Database Updates:**
    - Use batched `INSERT` or `UPDATE` statements with transactions.
    - Ensure idempotency to handle retries (e.g., using unique keys).

System Architecture with BullMQ

1. **Producer:**
    - Handles the file upload and splits the CSV into chunks.
    - Pushes each chunk (or chunk metadata) as a job to a Redis-backed queue.
2. **Worker:**
    - Listens to the queue.
    - Processes each chunk independently, ensuring scalability and fault tolerance.
3. **Queue (Redis):**
    - Acts as the intermediate storage for job messages, ensuring scalability and asynchronous processing.

↳ Use Node.js stream to split large file, can use Papa parse package:

STREAMING

"Did I mention the file is huge?"

That's what streaming is for. Specify a step callback to receive the results row-by-row. This way, you won't load the whole file into memory and crash the browser.

→ S3 url

```
Papa.parse("http://example.com/big.csv", {
    download: true,
    step: function(row) {
        console.log("Row:", row.data);
    },
    complete: function() {
        console.log("All done!");
    }
});
```

* When Multiple workers update DB count for total records parsed, use $inc in MongoDB

↓

[DB Design]

① No fixed schema

② No complex joins

③ Large writes and Data Size

④ No need of transaction

↓↓

NOSQL (Mongo DB)

Schema

① Upload schema

② Template schema

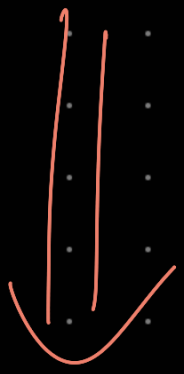
Upload schema

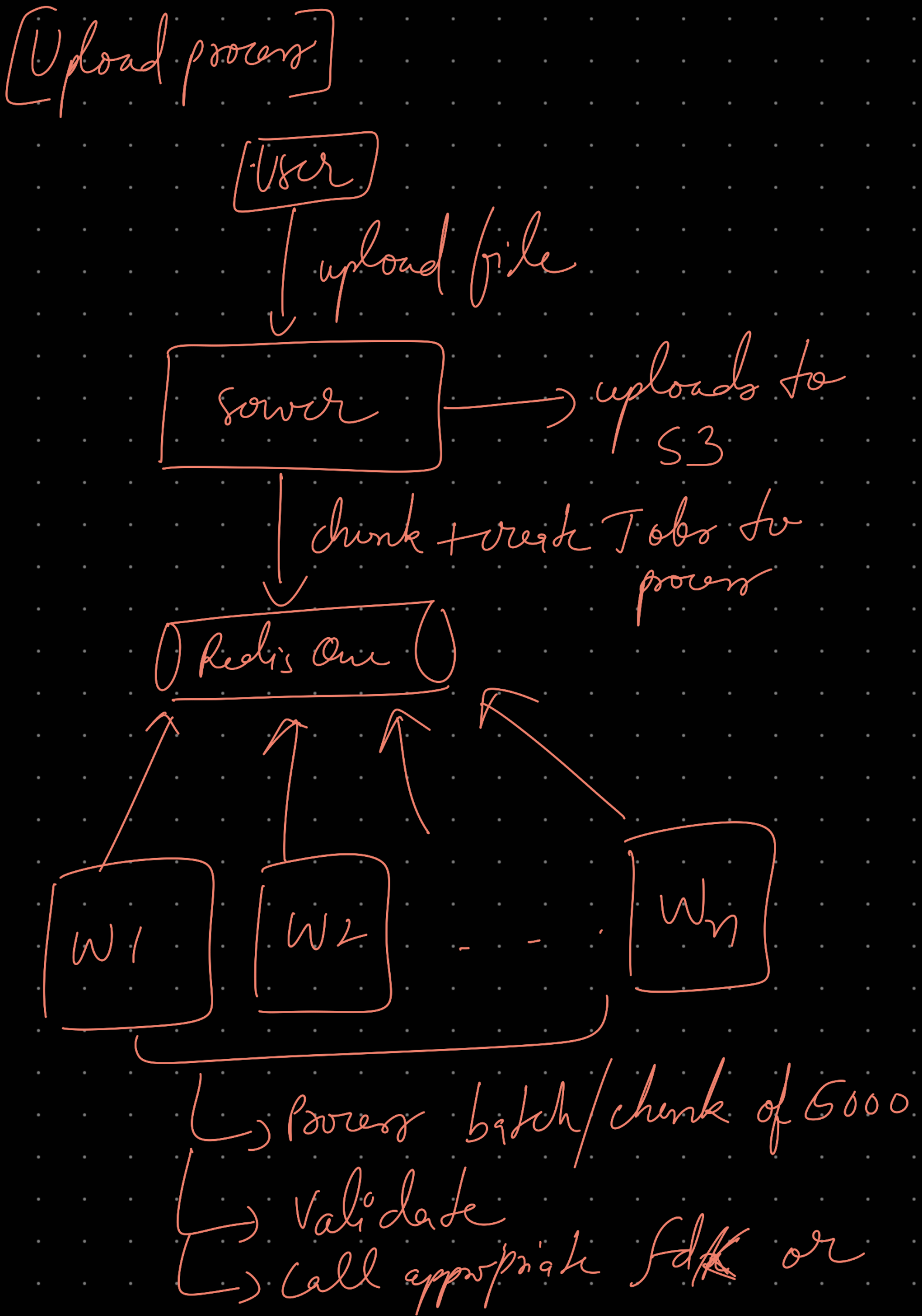
{
  _id : ___
  fileName : ___
  fileUrl : ___
  user : { }
  status : {
    value : InProcess / complete / failed
    details : {
      uploaded : ___
      error : ___
      duplicate : ___
    }
  }
}

Template Schema

```
{
  "_id": ObjectId("unique_template_id"),  // Unique identifier for the template
  "type": "brand",  // Template type, in this case "brand"
  "columns": [
    {
      "column_name": "Brand Name",  // Name of the expected column
      "type": "string",  // Expected data type of the column
      "format": "text",  // Format of the column, e.g., text, for string columns
      "required": true,  // Indicates that this column is required
      "max_length": 100,  // Maximum length for the column value
      "default_value": "",  // Default value for missing data (optional)
      "description": "The name of the brand"  // Description of the column
    },
    {
      "column_name": "Brand Code",  // Name of the expected column
      "type": "string",  // Expected data type of the column
      "format": "text",  // Format of the column
      "required": true,  // This column is required
      "max_length": 50,  // Maximum length for the column value
      "default_value": "",  // Default value for missing data
      "description": "Unique code assigned to the brand"  // Description of the column
    },
```

↳ Store these template in Redis Cache for validation




Upload process
User
upload file
server
uploads to S3
chunk + create Jobs to process
Redis Que
W1
W2
Wn
Process batch/chunk of 5000
Validate
call appropriate fdk or

└ service to bulk write
└ Store progress, error count in DB

[API]

POST /upload → upload file endpoint

GET /jobs → paginated list

GET /job/{id} → Single Job details

Challenges

① First design had problems, no chunking, so able to upload 500 - 800 rows

↓

Approach failed due to memory overflow in worker pods for large bulk upload having 20,000 - 30,000 rows

(2) Scaled with Distributed worker + Bull MQ

(3) Added client + Server side validations