ENGINEERING EXERCISE

DATA PROCESSING

1. INTRODUCTION

If you have reached this stage, then you are ready for the challenge that lies ahead of you. As a software engineer, the choice of a programming language should not be your biggest constraint. In this exercise, we are going to test you on your ability to adapt to change. We are thrown curveballs in life and the choice of the programming language should not stop an engineer from realizing their true potential.

The programming language to be used to solve this exercise is **BUN.JS**. Let us take a moment to appreciate the reigning champion, Node.js. However, as with all great inventions, Node.js has its limitations. Emerging from the shadows of Node.js, Bun.js aims to address these limitations by offering a complete, all-in-one JavaScript runtime.

Although Bun.js is still in its early stages, it's clear that it has the potential to significantly impact the JavaScript landscape. By offering a comprehensive toolkit, impressive speed, and compatibility with existing Node.js projects, Bun.js is poised to challenge Node.js's dominance in the JavaScript runtime space.

2. CHALLENGE DESCRIPTION

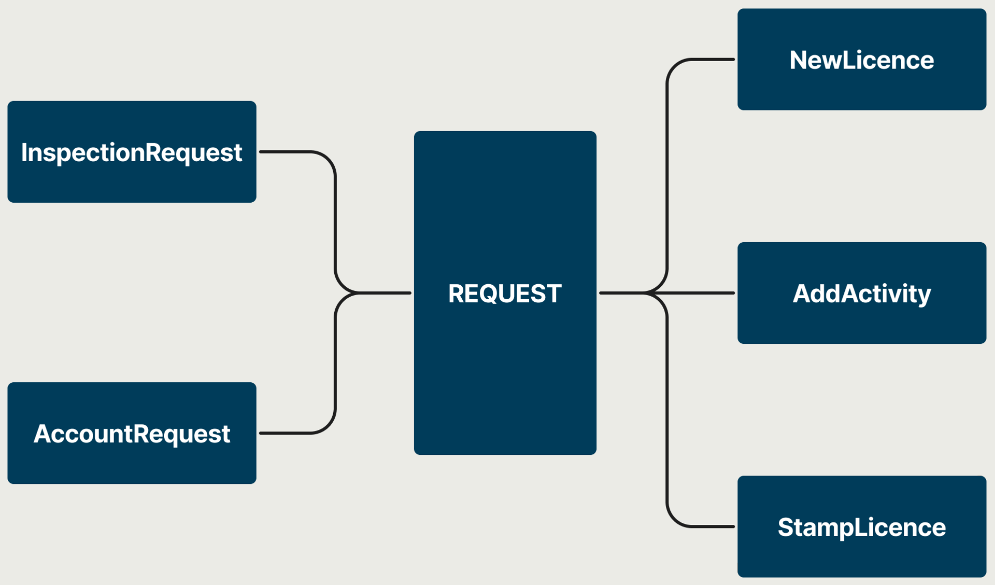
Bun.js provides the fastest SQLite in the JavaScript ecosystem. It natively implements a high-performance SQLite3 driver. To accomplish the exercise, you are expected to use the SQLite API in bun.js.

PROBLEM STATEMENT

A Request Management System processes multiple types of requests through the application. Each request has information that is different from another one. To accommodate this data management issue, the data architects designed the schema wherein a column in the table named "RequestData" stored all request-specific data as a JSON payload. The table for Requests in the current system looked like as shown in the figure below:

Column Name	Column Description	Data Type	Notes
RequestID	A unique ID for the each request in the system.	integer	Primary Key
RequestType	The type of the request.	Integer	Enumerator (1-5)
RequestStatus	The status of the request	Integer	Enumerator (1-3)
RequestData	The detailed information about the Request	Text	JSON Data

However, the product team wants to collect statistics that are based on the details inside the RequestData column which are currently not possible because of the structure of the table. The new schema is based on the types of the requests and is as shown below:



REQUEST DETAILS

The request types are explained below for reference to be used in the task

Request Type	Request Name	Request Description
1	New License	Request a new operating license for the company
2	Account Request	Request a new user account to be associated with the company
3	Inspection Request	Request an inspection for the facility
4	Add New Activity	Add a new activity to the license
5	Stamp License Letter	Request a stamped license certificate

DATA FILE

The CSV file to be used as a test file can be downloaded from the link: [LINK]

TASK DETAILS

The main requirements for this exercise to be accomplished is:

- Analyze the provided CSV file to figure out the Schema for the RequestData field for the different types of Requests
- Update the above entity model with details of each Request Type table
- Create an API that allows the user to upload the CSV file that will be received by the backend to do the processing and inserting the data in its rightful table in the database.
- Once all the records are added to the resultant tables, provide a summary of the import process to the front-end

- Display the number of records for each type that were imported and the total time it took to import the complete dataset into the database.

3. BONUS TASK

An important part of designing a systematic process is to ensure that it is resilient and can handle errors effectively. As you can imagine, the type enforcement cannot be mandated on the JSON key-value pairs, there is a possibility that a mismatch will occur which might stop the complete process. This becomes more concerning when you are dealing with records that are in millions.

This bonus task can be accomplished in two part and doing either will be considered:

1. Provide a plan of how you would handle this kind of situation.
2. Modify your code to handle this kind of issue in the best way you can.

4. SUBMISSION GUIDELINES

The final completed task must be uploaded to a GitHub repository and the link shared on submission so that it can be evaluated.

5. EVALUATION CRITERIA

Your performance will be assessed based on the following factors:

- Completeness of the functionality as per the requirements
- Code quality, readability, and organization
- Proper error handling and validation

A set of “validation” data files (csv) will be used to check the performance of your submission. These files will follow the same structure as the test file but may include test cases to evaluate certain functionalities specifically.