

Node.js Backend Module

Technical Design

Billing Automated

Requisition Service

(BARS)



# Table of Contents

[Table of Contents 2](#_Toc98431335)

[Revision History 3](#_Toc98431336)

[1. Purpose 4](#_Toc98431337)

[2. Requirements 4](#_Toc98431338)

[2.1. Business Requirements 4](#_Toc98431339)

[2.2. Technical Requirements 4](#_Toc98431340)

[3. Design 5](#_Toc98431341)

[3.1. Application Flow 5](#_Toc98431342)

[3.2. Input – Output 6](#_Toc98431344)

[3.2.1. Input 6](#_Toc98431345)

[3.2.2. Output 7](#_Toc98431346)

[3.2.3. Validation Messages 8](#_Toc98431347)

[3.3. CSV and Text Reader Implementation 8](#_Toc98431350)

[3.4. Database Design 9](#_Toc98431351)

[4. Development Architecture 10](#_Toc98431352)

[5. Tools to be Used 10](#_Toc98431353)

[6. Testing 10](#_Toc98431354)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Summary** | **Updated By** |
| 3.0 | 3/22/2022 | Update documentation. | Kenneth Bolima |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Purpose

This document shall provide guidance to developers in developing Billing Automated Requisition Service (BARS).

The details on this document defines the requirements for the system such as procedures, technology stack, application architecture, etc. Also, it will guide the developers to any technical requirement of the system.

Developers should be able to create a working Billing Automated Requisition Service (BARS) backend application that utilizes NodeJS and MongoDB. The developers need to use different tools to test the application such as Postman, Browser, etc.

# Requirements

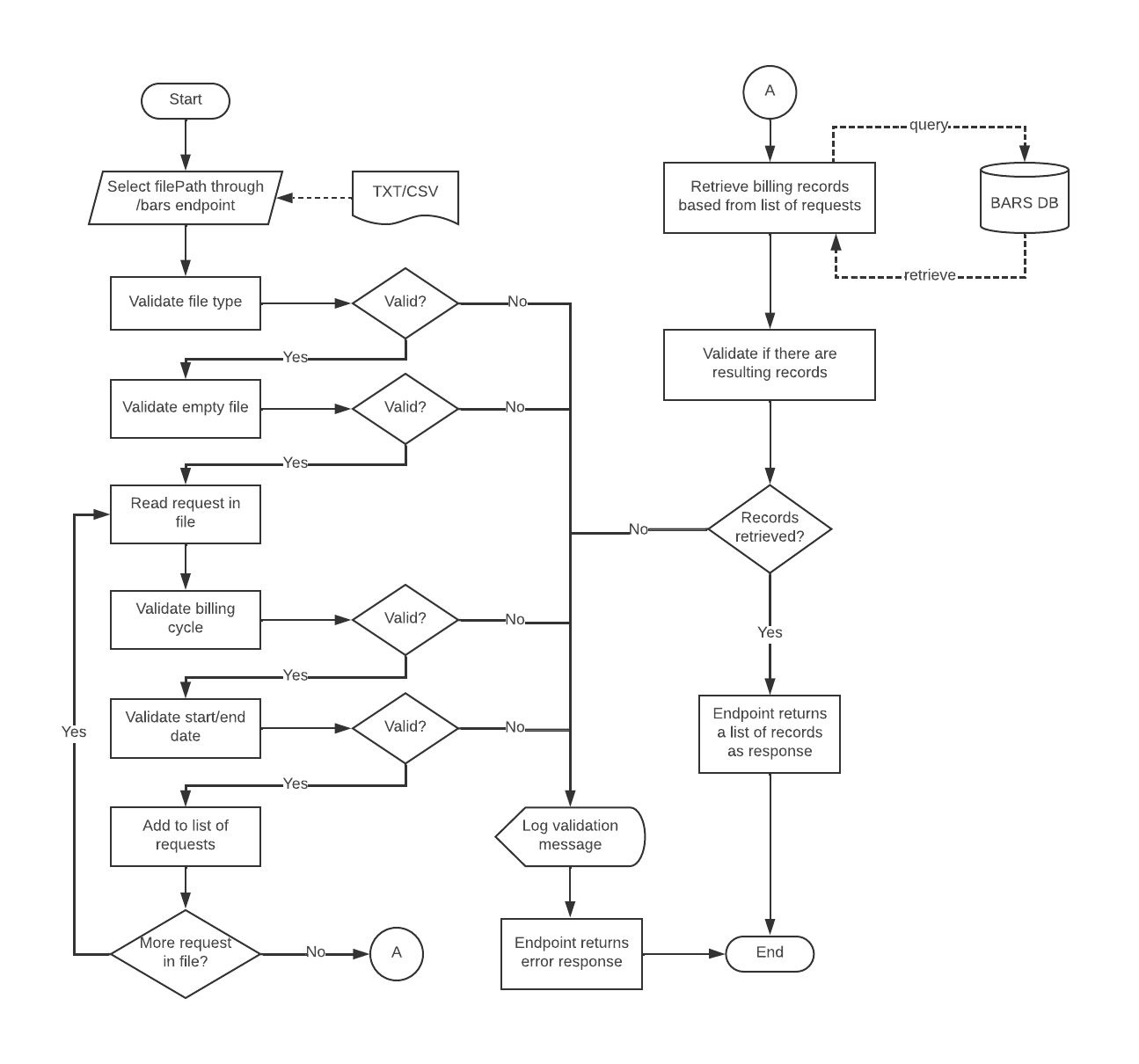
## Business Requirements

Refer to the **BARS Functional Design** document.

## Technical Requirements

* The final application should run on postman, arc, or browser.
* Developers should use mandated tools specified in Section 4: Development Architecture

# Design

* 1. **Application Flow**

## Input – Output

## Input

The request file should contain the following fields:

• Billing Cycle range is 1-12

• Start Date format depends on file type

• End Date format depends on file type

#### ***Text File***

If upload file is in TXT format, the alloted space for the following fields are:

* Billing Cycle 2 spaces Ex. 01
* Start Date 8 spaces Ex. 01012012 (MMDDYYYY)
* End Date 8 spaces Ex. 01012012 (MMDDYYYY)

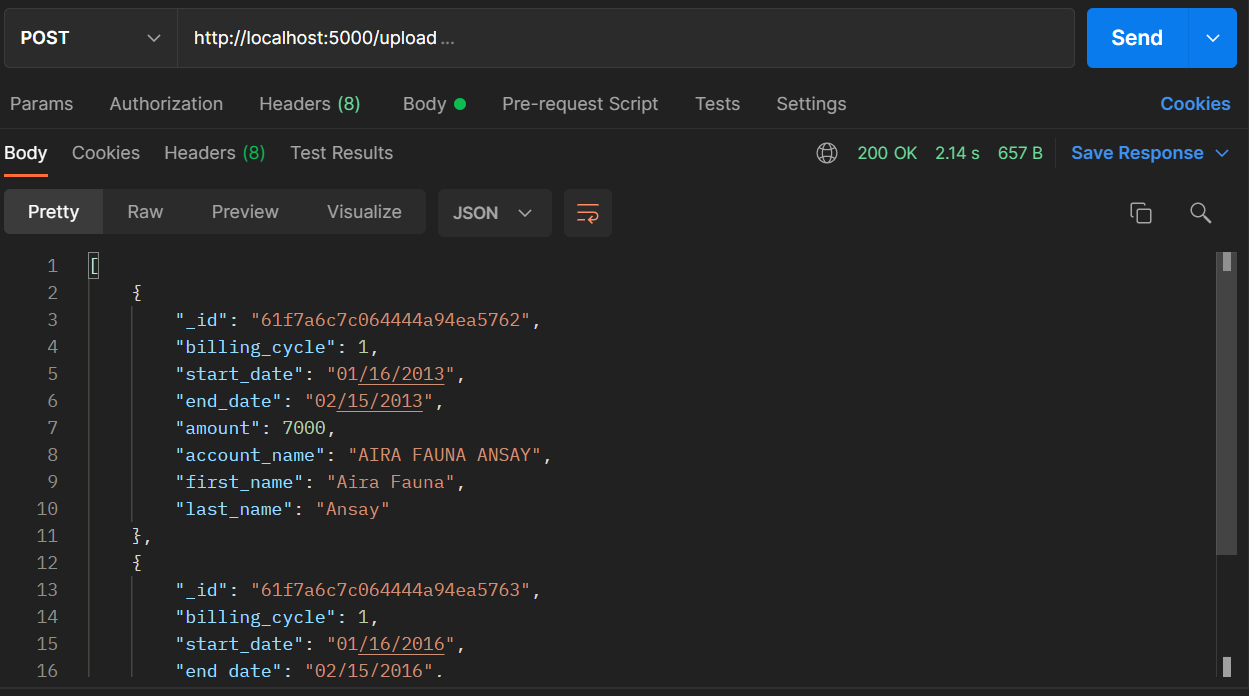
#### ***CSV File***

If uploaded file is in CSV format, the alloted space for the following fields are:

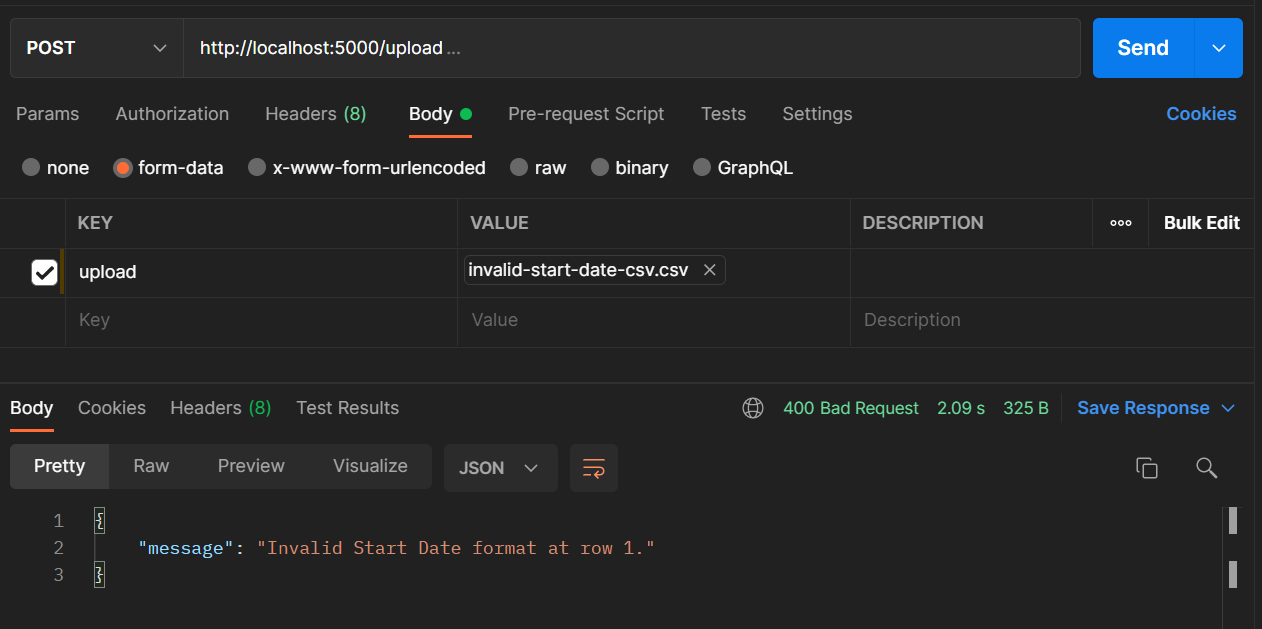
* Billing Cycle 1st column Ex. 1
* Start Date 2nd column Ex. MM/DD/YYYY
* End Date 3rd column Ex. MM/DD/YYYY

## Output

#### ***Successful Response***



#### ***Error Response***



## Validation Messages

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Message** | **Validation Condition** |
| Information | “Please input an existing request file path.” | Check if the file path entered exists. |
| Information | “No request(s) to read from the input file.” | If the current file being read does not contain any request (empty file). |
| Information | “No Record Found!” | If there are no resulting records from the database. |
| Information | “File is not supported for processing” | File extension is not supported to process |
| Error | “ERROR: Billing Cycle not on range at row *<RowNumber>*.” | Billing cycle should be from 1 – 12 only. |
| Error | “ERROR: Invalid Start Date format at row *<RowNumber>*.” | Depending on the file type being read, date should be on the correct date format. |
| Error | “ERROR: Invalid End Date format at row *<RowNumber>*.” | Depending on the file type being read, date should be on the correct date format. |

## CSV and Text Reader Implementation

Create a ***fileReader.js***which should contain the following functions:

* ***readCsv(filename)*** – function for reading CSV files and parsing request
* ***readTxt(filename)***– function for reading TXT files and parsing request

Both methods must return the requests as list of objects. Example:

requests = [

    {

        billing\_cycle: 1,

        start\_date: new Date(Date.UTC(2013, 0, 16, 0)),

        end\_date: new Date(Date.UTC(2013, 1, 15, 0))

    },

    {

        billing\_cycle: 1,

        start\_date: new Date(Date.UTC(2016, 0, 16, 0)),

        end\_date: new Date(Date.UTC(2016, 1, 15, 0))

    }

]

These methods must be unit tested using *Jest* testing framework. See TCER.

## Database Design

Database name is bars\_db. The database should have a billings collection which has the following schema:

{

    billing\_cycle: Number,

    billing\_month: String,

    amount: Number,

    start\_date: Date,

    end\_date: Date,

    last\_edited: String,

    account: {

        account\_name: String,

        date\_created: Date,

        is\_active: String,

        last\_edited: String,

        customer: {

            first\_name: String,

            last\_name: String,

            address: String,

            status: String,

            date\_created: Date,

            last\_edited: String

        }

    }

}

The content of the billings collection should be initialized with the bars\_db.js script on either MongoDB shell or MongoDB Compass.

# Development Architecture

**Technology Stack:**

* Programming Language
  + JavaScript Language
    - NodeJS
* NoSQL Database
  + MongoDB
* Application Build Configuration
  + Node

# Tools to be Used

Developers will use the following for developing and building their applications:

* Node v14.9.0
* IDE: Visual Studio Code, MongoDB Compass for MongoDB GUI
* MongoDB
* Code Quality Tool: SonarQube Server and SonarQube Runner
* Postman

# Testing

Please refer to the **BARS Function Test Plan** document.