Atlan_Backend_Task

Problem Satement:

1. One of our clients wanted to search for slangs (in local language) for an answer to a text question on the basis of cities (which was the answer to a different MCQ question)

   Approach 1: We can create an API or a Database(SQL) to map the word(or Answer) to slangs in other languages

   Assumption: - English as primary language
   Lets assume we have only 1 word slangs

   Thought: - if we want some slangs we cannot directly translate the word. Because slangs are frequently used words while speaking rather then writing. While translation will not keep this fact in action. On the other side we want a question to get converted to slang(local language) so we can use some translator library.

   Example: -

   Nice -> acha -> Hindi

   Nice -> saras -> Gujarati

lang_id	**language**
“hin”	“hindi”
“guj”	“gujarati”

ID	**lang_id**	**word**	**slang**
1	“hin”	“nice”	“acha”
2	“guj”	“nice”	“saras”

Note: To create such type of Database the number of records could be very high and hence throughput will not be as required. we also need to consider the performance and latency in response.

Approach 2: Translation API approach

```
// Find slang in local/any language
        function findSlang(req, res, next) {
            console.log(req.query);
            try{
                const text = await translate(req.query.word, req.query.lang);
                  req.query.lang = 'hi' //(Hindi)
                  req.query.word = "Awesome" //(English - auto detection)
                res.send(text); // Jhakaas
            }
            catch(err){
                res.send(err.message);
            }
            next();
        };
```

2. A market research agency wanted to validate responses coming in against a set of business rules (eg. monthly savings cannot be more than monthly income) and send the response back to the data collector to fix it when the rules generate a flag.

Created a sample Postgres based relational database

```
CREATE DATABASE atlan;

  CREATE TABLE client_income_data(
      client_id SERIAL PRIMARY KEY,
      client_email VARCHAR(255),
      client_name VARCHAR(255),
      income_per_annum FLOAT,
      savings_per_annum FLOAT,
      mobile_number VARCHAR(15)
  );
```

Route to validate while insertion

```
// Validate data middleware sample
       function validateData(req, res, next) {
          const { income_per_annum, savings_per_annum, mobile_number} = req.body;
           if (income_per_annum < savings_per_annum) {
              res.send("Invalid Data Savings cannot be more than Income");
          }
          else if (isNaN(mobile_number)) {
              res.send("Invalid mobile number, only digits are acceptable");
          }
          else if (mobile_number.length !== 10) {
              res.send("Invalid mobile number, should be of 10 digits");
          }
          next();
      };
```

```
// Validate while insertion of a new client details

  app.post('/validateNew', validateData, async (req, res) => {
           try {
               const { client_email, client_name, income_per_annum, savings_per_annum,
mobile_number } = req.body;
               const newClient = await pool.query("INSERT INTO
client_income_data(client_email,client_name,income_per_annum,savings_per_annum,mobile_number)
               VALUES($1,$2,$3,$4,$5) RETURNING *", [client_email, client_name,
income_per_annum, savings_per_annum, mobile_number]);
               res.json(newClient.rows[0]);
           } catch (err) {
               res.send(err.message);
           }
       });
```

Route to validate All the records/responses if missed to validate

```
// Validate all and send invalid data to data collector
app.get('/validateAll', async (req, res) => {
    try {
        let inValidRows = await pool.query("SELECT * FROM
client_income_data WHERE savings_per_annum > income_per_annum");
        inValidRows = inValidRows.rows;
        if(inValidRows.length === 0)
        {
            res.send("All records are Valid");
        }
        else {
            res.send(inValidRows);
        }
    } catch (err) {
        console.log(err.message);
    }
});
```

3. A very common need for organizations is wanting all their data onto Google Sheets, wherein they could connect their CRM, and also generate graphs and charts offered by Sheets out of the box. In such cases, each response to the form becomes a row in the sheet, and questions in the form become columns.

```
async function exportCSV(req, res) {
         try {
             let data = await pool.query("SELECT * FROM client_income_data");
             data = data.rows;
             var file = fileSystem.createWriteStream("public/data.csv");
             fastcsv
                 .write(data, { headers: true })
                 .on("finish", function() {
                      res.send("<a href='/public/data.csv' download='data.csv'
id='download-link'></a>
                      <script>document.getElementById('download-link').click();</script>");
                 })
                 .pipe(file);
         } catch (err) {
             console.log(err.message);
         }
      }
```

4. A recent client partner wanted us to send an SMS to the customer whose details are collected in the response as soon as the ingestion was complete reliably. The content of the SMS consists of details of the customer, which were a part of the answers in the response. This customer was supposed to use this as a “receipt” for them having participated in the exercise.

```
async function SMS(req, res) {
          try {
              const {client_email, client_name, income_per_annum,
savings_per_annum, mobile_number} = req.body;
              var options = {
                  authorization: process.env.API_KEY,
                  message: ` Your Details :\n
                              Email ID :${client_email}\n
                              Name : ${client_name}\n
                              Income Per Annum: ${income_per_annum}\n
                              Savings Per Annum: ${savings_per_annum}\n
                              Contact : ${mobile_number}\n
                              Thankyou for your response`
                              ,
                  numbers: [mobile_number]
                  };
              const response = await fast2sms.sendMessage(options); //Asynchronous
Function.
              res.send(response.message);
          } catch (err) {
              res.send("Failed to send SMS to the Client");
          }
      }
```

.