ELANCO BACKEND PLACEMENT TASK

Documentation and Thought Process

OVERVIEW

1. Architecture Decisions

Modular Express Structure

I chose to use a modular file structure during development of the task. The code became convoluted as the number of routes increased. Separating the routes and associated JSDocs into their own modules allowed me to improve development flow and kept the application clean and maintainable.

Routes, utilities, and data-processing responsibilities were separated into their own directories:

- /routes – API endpoints grouped by feature (sightings, reports, summary insights)
- /utils – Reusable helpers responsible for filtering, error handling, and pagination of requests
- /data – Responsible for the reading, cleaning, and deduplicating data on server startup.

This approach ensures:

- Future scalability of existing endpoints
- Reusable modules for repeated tasks
- Easy readability for additional developers
- Reduced file size per module

Node.js

I decided to use Node.js for my RESTful API. There was a variety of reasons that helped me come to this conclusion:

- Node is lightweight, easy to deploy, well suited for REST API's
- There are massive amounts of documentation and examples available online to reference
- Node integrates directly with JSON data, which made API responses much easier to handle
- Lots of options for auto-generating documentation pages e.g. Swagger UI
- Suitable for more relevant languages like typescript

2. Data-Driven Design

XLSX Library

The brief provided a dataset inside of an Excel spreadsheet, therefore, I used an external xlsx library to parse the contents into memory during startup for the requirements of this task.

The reasons for this choice:

- Avoid making unnecessary database for simple tasks
- Faster response to queries due to in-memory reads
- Dataset was small enough to fit in memory

Dataset cleaning included:

- Removing rows with incomplete data

- Deduplicating rows that share the same primary ID
- Normalising the date values with ISO standard and slicing for readability
- Informing via console for rows removed during startup clean

3. How the system consumes and presents data

Data Input

On startup

- Excel file is read using xlsx
- Sheets are converted to JSON
- Cleaning and deduplication is applied to data
- Final data is stored in memory

Data processing

There are a variety of routes that have some shared utilities applied during processing

Each route uses one or more of the following methods to facilitate this:

- filterSightings() – Applies user queries for location, start date, and end date to query responses.
- paginatedResults() – Applies a limit and page count to queries without altering original data, user can query this using their own choice of limit and pages.
- invalidDateCheck() – Checks for dates not being in format of a number – defaults to no date chosen

Data Presentation

Once the data has been processed, the responses are returned as clean JSON objects

Below is some snippets showing partial examples of responses to queries:

http://localhost:3000/sightings


```
{
  "page": 1,
  "limit": 100,
  "total": 1000,
  "totalPages": 10,
  "next": {
    "page": 2,
    "limit": 100
  },
  "data": [
    {
      "id": "02WNholuSg6ndCk4c1dA",
      "date": "2022-08-01T06:40:31",
      "location": "Manchester",
      "species": "Marsh tick",
      "latinName": "Ixodes apronophorus"
    },
    {
      "id": "02rFwLCaAwZSVxdTDicK",
      "date": "2014-09-12T23:33:03",
      "location": "London",
      "species": "Southern rodent tick",
      "latinName": "Ixodes acuminatus"
    },
```

http://localhost:3000/reports/locations


```
{
  "Manchester": 72,
  "London": 76,
  "Glasgow": 78,
  "Birmingham": 82,
  "Southampton": 57,
  "Nottingham": 95,
  "Sheffield": 69,
  "Liverpool": 73,
  "Edinburgh": 73,
  "Newcastle": 57,
  "Leicester": 61,
  "Bristol": 77,
  "Cardiff": 71,
  "Leeds": 59
}
```


http://localhost:3000/sightings?location=London&startDate=2018-01-01


```
{
  "page": 1,
  "limit": 100,
  "total": 44,
  "totalPages": 1,
  "data": [
    {
      "id": "0mgV5wo4mQ9i0t8Hpfdw",
      "date": "2020-10-19T21:35:03",
      "location": "London",
      "species": "Tree-hole tick",
      "latinName": "Ixodes arboricola"
    },
    {
      "id": "3vPvL7YSxqalOZ0PRNAF",
      "date": "2018-02-25T00:06:31",
      "location": "London",
      "species": "Southern rodent tick",
      "latinName": "Ixodes acuminatus"
    },
    {
      "id": "4yzqbYbSPWIW6smAVQzQ",
      "date": "2019-09-22T09:11:24",
      "location": "London",
      "species": "Passerine tick",
      "latinName": "Dermacentor frontalis"
```

4. How I would improve with more time

The improvements I considered:

Database Integration

Currently, the API loads the data directly from an Excel spreadsheet using xlsx library. A future improvement would be integrating the data into a proper database such as:

- MySQL
- MongoDB

This would allow:

- More complex analysis
- persistence of user submitted sightings
- higher scalability for storage of larger datasets
- faster queries

Improved Machine learning insights

The current insights use hard-coded logic rather than machine learning or AI models. A future version could include finding a suitable library capable of additional functionality such as:

- anomaly detection e.g. spike of data in particular weeks across locations
- classification of zones that are lower or higher risk
- forecasting for future tick sightings based on previous results

Improved error handling

The API handles most normal user errors with a graceful message informing them of the mistake on request format. Additional improvements could be:

- More detailed error handling e.g. checking for numbers in location filter (Currently parsed as if they are strings)
- Logs of previous errors for developer analytics
- Middleware dedicated towards most error scenarios for more modular code base

Additional endpoints

More endpoints could be made available, especially if the datasheet changes to include more column headers. For example:

- Most active days/weeks
- year by year comparison trends report
- Graphical displays of data using additional libraries