Testlio coding assignment (backend)

Description

All projects in Testlio contain issues. Issues describe problems and bugs, found by testers in client applications. Your job is to create a simple REST API to manage issue entities.

An initial project with some example code has been set up for you, but you're not limited to these tools & structure, feel free to make changes if you need to. The API must be implemented in Node.js.

Development environment

The repo contains a simplistic Docker Compose setup to make your life easier, but you are free to modify it or use a different toolset based on your preferences.

1. Install Docker and Docker Compose on your machine.
2. Build and run the Docker containers using the following command:

   ```
   docker-compose up --build -d
   ```

3. The API will be available at http://localhost:8080.
4. You can connect to the database from any MySQL client. It's available at localhost:3307 with the root user & the password you find in `.env`.

Preparations and workflow

- Create your own Git branch in the format `solution/{firstname}-{lastname}`, e.g. `solution/peter-griffin`.
- We encourage you to commit often, but feel free to use whatever approach works best for you.
- Currently the project repository has no remote set up, please keep it that way, don't publish or share it.

- Keep this README file for the instructions, and create a new `SUBMISSION.md` file to document any important decisions, trade-offs or technical details that will help us understand you solution.
- At the end, compress the repository to a .zip file and upload it to the link we provided in the email. Make sure the .zip file contains hidden files & directories (e.g. `.git`, `.env`) but doesn't contain the `node_modules` folder.

Currently implemented

- Basics of the REST API using Node.js and koa
- MySQL database with a table for issues
- Sequelize model for issues

A sample issue to illustrate the issue item structure:

```
{
    "id": 123,
    "title": "Bug in issue-service",
    "description": "Ah snap :("
}
```

Your tasks

Task 1: Implement an endpoint that creates a new issue

Task 2: Implement an endpoint that lists all stored issues

Task 3: Implement an endpoint that modifies an issue

Task 4: Implement issue revisions

Issues being one of the central models of Testlio, it is important to track the changes made to them. Every time a change is made to an issue, we want to track what exactly was changed, by whom and when.

Each change is a **revision**, containing the issue current state and the changes made.

For example:

```
{
    "issue": {
        "title": "Bug in issue-service",
        "description": "It does not generate revisions"
    },
    "changes": {
        "description": "It does not generate revisions"
    },
    "updatedAt": "2024-03-29T15:40:42.000Z"
}
```

Requirements:

- Store issue revision when creating a new issue
- Store issue revision when updating an issue
- Implement a new endpoint that returns all revisions of a particular issue

Task 5: Implement authentication

- Require a valid JWT token to be present for all requests (except for the discovery and health endpoints)
- Clients must include an X-Client-ID header in every request.
- Every time a change is made to the DB, store the change author's email address (i.e. `created_by` , `updated_by`).

Task 6: Before and after comparison

Create an endpoint that takes two revisions of a particular issue and returns the difference between the two.

The response object should contain the following data pieces:

- `before` : the issue's content at revision A
- `after` : the issue's content at revision B
- `changes` : summary of the differences (i.e. listing all properties that have changed and their values)
- `revisions` : the full trail of revisions between version A and B

By default the comparison can work from older to newer revisions, but for bonus points you can implement comparisons from newer to older revisions as well.

FAQ

Should I take the tasks in order?

Yes, we expect you to start from the first task and progress from there.

How much time should I spend on the task?

In our experience 4-6 hours is usually enough for a sufficient solution, but it depends on your availability and previous experience with the tools.

What do you look for in the solution?

Ideally we would like to see a working solution. The solution does not have to be perfect, so first make it work, then make it pretty.

We will look for REST API development best practices, like route naming, validations, error handling, pagination, ensuring data consistency and so on.

During the technical interview you can elaborate on your further design & architecture ideas, and we will also discuss additional aspects needed to take the service to production and operate it with high reliability and availability.

Bonus points

If you have extra time and energy after meeting our core expectations, here are some ideas to impress us:

- we would really love to see your code covered by tests
- you can generate documentation for the API
- you can define migrations for the data schema changes
- or you can even convert the codebase to TypeScript :)