Server Challenge

1 Purpose

The purpose of this challenge is to assess your skill in JavaScript, Node.js, and the core technology used on Elecctro's application servers – Hapi.js (https://hapi.dev/).

2 Challenge

The challenge is to create a simple REST API server for managing a "to-do list". The server should be able to perform the following actions:

- List the items on the to-do list, optionally filtered by state and sorted.
- Add an item to the list.
- Edit an item on the list (i.e., change its description or mark it as complete).
- Remove an item from the list.

2.1 Requirements and Recommendations

The server should be programmed in JavaScript (ES6/7) and must be compatible with Node 12.0.0 and above (see http://node.green).

You are encouraged to use ES6/7 grammar.

The use of the '*var*' keyword is forbidden.

Use 'async/await' to handle asynchronous operations (e.g., retrieving data from a database).

You can use any public package in the NPM repository.

The use of utility libraries, such as **lodash** or **underscore** is forbidden.

The use of promise libraries, such as **bluebird** is forbidden.

You are encouraged to use ESLint and the Hapi.js plugin (https://hapi.dev/module/eslint-plugin/).

The server must use the Hapi.js framework (https://hapi.dev/).

The database must be SQLite or PostgreSQL and you should use Knex.js (http://knexjs.org/) query builder.

The input (URL parameters, query parameters, payload) and output (response) of all server requests must be validated using **joi** (https://joi.dev/). Check the Hapi.js documentation for how to do this.

The server must expose a route (GET /docs) with the auto-generated server documentation. You should use the **hapi-swagger** plugin (https://github.com/glennjones/hapi-swagger) for this task.

The source code must be published on a git repository. Make sensible commits and detail the changes on the commit messages.

2.2 REST API Routes

The following routes should be implemented:

POST /todos

This route should add an item to the to-do list.

The expected request body should contain a single JSON object with the following field:

- **description** – Description of the item (e.g., *Buy milk at the store.*).

The server should attribute a unique identifier to the created item (you can use sequential numbering or a UUID), set the state to '*INCOMPLETE*' and store the creation date.

The response should be a JSON object, representing the created item, having the following fields:

- **id** – Unique identifier of the list item (e.g., 56).
- **state** – State of the item (*INCOMPLETE*).
- **description** – Description of the item (e.g., *Buy milk at the store.*).
- **createdAt** – Creation date of the item (e.g., *2021-05-12T07:23:45.678Z*)
- **completedAt** – Completion date of the item (*null*).

GET /todos?filter=<STATE>&orderBy=<FIELD>

This route should list the to-do items considering the conditions imposed on the query parameters:

- The **filter** query parameter is optional and can be '*ALL*', '*COMPLETE*', or '*INCOMPLETE*'. If not specified, the default filter is '*ALL*'.
- The **orderBy** query parameter is also optional and can be '*DESCRIPTION*', '*CREATED_AT*', or 'COMPLETED_AT'. If not specified, the default order is '*CREATED_AT*'.

The response should be a JSON array of objects, where each object should have the following fields:

- **id** – Unique identifier of the list item (e.g., 56).
- **state** – State of the item (i.e., *COMPLETE* or *INCOMPLETE*).
- **description** – Description of the item (e.g., *Buy milk at the store.*).
- **createdAt** – Creation date of the item (e.g., *2021-05-12T07:23:45.678Z*).
- **completedAt** – Completion date of the item (e.g., *2021-05-13T11:23:45.678Z*).

PATCH /todo/{id}

This route should edit an item on the to-do list. The edited item will be referenced by id using the URL parameter **id** (see above).

The expected request body should contain a single JSON object with a combination of the following fields:

- **state** – State of the item (i.e., *COMPLETE*).
- **description** – Description of the item (e.g., *Buy milk at the store.*).

Both fields are optional, but at least one must be present.

The server should verify the referenced item exists. If the item does not exist it must return an HTTP 404 error.

The server should verify if the item state is *INCOMPLETE* before making any changes to the description. If the description is being modified and the item state is *COMPLETE*, the server should return an HTTP 400 error.

The response should be a JSON object, representing the edited item, having the following fields:

- **id** – Unique identifier of the list item (e.g., *56*).
- **state** – State of the item (i.e., *COMPLETE* or *INCOMPLETE*).
- **description** – Description of the item (e.g., *Buy milk at the store.*).
- **createdAt** – Creation date of the item (e.g., *2021-05-12T07:23:45.678Z*)
- **completedAt** – Completion date of the item (e.g., *2021-05-13T11:23:45.678Z*)

DELETE /todo/{id}

This route removes an item from the to-do list. The item will be referenced by id using the URL parameter **id** (see above).

The server should verify the referenced item exists. If the item does not exist it must return an HTTP 404 error.

This route should return an empty response if it succeeds.

2.3 Bonus – Authentication

Add a form of authentication, social or local, to be able to manage to-do lists for multiple users.

Tip: Check Hapi.js **jwt** (https://hapi.dev/module/jwt/) and **bell** (https://hapi.dev/module/bell/) plugins.

Suggested REST API routes:

POST /login

This route should receive a username and password, authenticate a user, and return a credential (e.g., a JWT token).

POST /logout

This route should invalidate the credentials of the authenticated user.

POST /users

This route should receive the user details (e.g., e-mail address, password, name, …) and create a new account, optionally returning a credential (e.g., a JWT token).

GET /me

This route should return the details of the authenticated user.

PATCH /me

This route should edit the details of the authenticated user.