Server Challenge

1 Purpose

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

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 7.10 (see http://node.green).

You are encouraged to use ES6/7 grammar.

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

You can use any public package in the NPM repository.

The use of libraries such as **lodash** or **underscore** is discouraged.

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

The server must use the Hapi.js framework (https://hapijs.com).

To emulate a database, you should use Hapi.js **catbox** plugin (https://github.com/hapijs/catbox) and the **catbox-memory** (https://github.com/hapijs/catbox-memory) adapter.

The input (URL parameters, query parameters, payload) and output (response) of all server requests must be validated using **joi** (https://github.com/hapijs/joi). 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 **lout** plugin (https://github.com/hapijs/lout) for this task.

2.2 REST API Routes

The following routes should be implemented:

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

This route should list the to-do items on the list taking into account 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*' or '*DATE_ADDED*'. If not specified, the default order is '*DATE_ADDED*'.

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.*);
- **dateAdded** – Creation date of the item (e.g. *2015-10-26T07:46:36.611Z*).

PUT /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.*);
- **dateAdded** – Creation date of the item (e.g. *2015-10-26T07:46:36.611Z*).

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 and its state is *INCOMPLETE* before making any edits. If the item does not exist it must reply with an HTTP 404 error, and if it is already in a *COMPLETE* state with 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.*);
- **dateAdded** – Creation date of the item (e.g. *2015-10-26T07:46:36.611Z*).

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 reply with an HTTP 404 error.

This route should return an empty response if it succeeds.

2.3 Bonus

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

Tip: Check Hapi.js **hapi-auth-cookie** (https://github.com/hapijs/hapi-auth-cookie) and **bell** (https://github.com/hapijs/bell) plugins.