Lab 03-02 94/100 pts

horizontal line

# Issue Tracker - Phase 3

This lab is a continuation of the Issue Tracker project that we started previously. In this assignment we will use [**joi**](https://joi.dev/) to improve the validation logic we wrote earlier.

*This lab assignment is a continuation of* ***Lab 03-01.***

# Step 1. Use your existing GitHub Repository

Use your **awd1111-issue-tracker** repository that you created earlier.

**Do not create a new repository!**

# Step 2. Install new dependencies

Install the following additional dependencies, using the **"npm install"** command:

1. joi

# Step 3. Start working through the exercises

That's all you need to get started on this assignment. You can now start working through the exercises.

# 

# Best Practices

JavaScript has evolved significantly in the past 5 years. In these exercises I expect you to follow some more modern practices than you may have learned previously.

* Indent code using **2 spaces** per level, as per [industry standards.](https://google.github.io/styleguide/jsguide.html#formatting-block-indentation)
* Name all variables and functions using **camelCase.**
* Don't use **var.** Period. **(You will lose 5 points for every use of the var keyword!)**
* Use **const** to define variables whenever possible. For example:

|  |
| --- |
| const pi = 3.14; const radius = window.prompt('radius'); const circumference = pi \* radius; |

* Use **let** to define variables only when **const** isn't possible. For example:

|  |
| --- |
| let x = 3; x += 5; |

* Prefer **arrow functions (=>)** for anonymous functions, over the traditional **function declaration** syntax. For example:

|  |
| --- |
| (x, y) => x + y |

* The **function declaration** syntax is permitted for creating **named functions.** For example:

|  |
| --- |
| function add(x, y) { return x + y; } |

* Use [**template strings**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) instead of string concatenation. For example:

|  |
| --- |
| const fullName = `${firstName} ${lastName}`; |

* Use [**async-await**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function) to support asynchronous database operations.

|  |
| --- |
| async function findAllPets() {  const db = await connect();  const pets = await db.collection('pets').find({}).toArray();  return pets; } |

* Use [**try-catch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch) to handle exceptions and promise rejections.

|  |
| --- |
| router.get('/list', async (req, res, next) => {  try {  const pets = await dbModule.findAllPets();  res.json(pets);  } catch (err) {  next(err);  } }); |

# Exercise 1. Update Users API (40 pts)

The previous lab assignment asked you to update the Users API to use a database. Use [**joi**](https://joi.dev/) to refactor that validation logic as directed below.

* Implement this API as a route module, located at **/routes/api/user.js**
* Test this API with [**Postman.**](https://www.postman.com/downloads/)

## GET /api/user/list

* *No changes needed at this time.*

## GET /api/user/:userId (5pts)

* The User ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **userId** is provided, then send the following response:
  + status = 404
  + json = { error: `userId ${userId} is not a valid ObjectId.` }

## POST /api/user/register (10pts)

* The following data **must** be provided as the body of the request, and read from **req.body:**
  + email (must be a valid email address)
  + password
  + givenName
  + familyName
  + role (must be one the previously defined roles)
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is missing or invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## POST /api/user/login (10pts)

* The following data **must** be provided as the body of the request, and read from **req.body:**
  + email
  + password
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is missing or invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## PUT /api/user/:userId (10pts) -2pts Fails to update without password. Password is not required.

* The User ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **userId** is provided, then send the following response:
  + status = 404
  + json = { error: `userId ${userId} is not a valid ObjectId.` }
* The following data **may** be provided as the body of the request, and read from **req.body:**
  + password
  + fullName
  + givenName
  + familyName
  + role (must be one the previously defined roles)
* All inputs are optional. Only update the fields that were provided.
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## DELETE /api/user/:userId (5pts)

* The User ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **userId** is provided, then send the following response:
  + status = 404
  + json = { error: `userId ${userId} is not a valid ObjectId.` }

# Exercise 2. Update Bugs API (60 pts)

The previous lab assignment asked you to update the Users API to use a database. Use [**joi**](https://joi.dev/) to refactor that validation logic as directed below.

* Implement this API as a route module, located at **/routes/api/bug.js**
* Test this API with [**Postman.**](https://www.postman.com/downloads/)

## GET /api/bug/list

* *No changes needed at this time.*

## GET /api/bug/:bugId (10pts)

* The Bug ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **bugId** is provided, then send the following response:
  + status = 404
  + json = { error: `bugId ${bugId} is not a valid ObjectId.` }

## PUT /api/bug/new (10pts)

* The following data **must** be provided as the body of the request, and read from **req.body:**
  + title
  + description
  + stepsToReproduce
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is missing or invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## 

## PUT /api/bug/:bugId (10pts)

* The Bug ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **bugId** is provided, then send the following response:
  + status = 404
  + json = { error: `bugId ${bugId} is not a valid ObjectId.` }
* The following data **may** be provided as the body of the request, and read from **req.body:**
  + title
  + description
  + stepsToReproduce
* All inputs are optional. Only update the fields that were provided.
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## PUT /api/bug/:bugId/classify (10pts) -2pts It allows a blank or anything in the field. classification (must be one the previously defined classifications) as stated below.

* The Bug ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **bugId** is provided, then send the following response:
  + status = 404
  + json = { error: `bugId ${bugId} is not a valid ObjectId.` }
* The following data **must** be provided as the body of the request, and read from **req.body:**
  + classification (must be one the previously defined classifications)
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## 

## PUT /api/bug/:bugId/assign (10pts)

* The Bug ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **bugId** is provided, then send the following response:
  + status = 404
  + json = { error: `bugId ${bugId} is not a valid ObjectId.` }
* The following data **must** be provided as the body of the request, and read from **req.body:**
  + assignedToUserId
  + ~~assignedToUserName~~ (query the database for the user's info)
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }

## PUT /api/bug/:bugId/close (10pts) -2 pts sending a boolean true causes call to timeout trying to update

* The Bug ID must be accepted as a **path parameter**, and read from **req.params**
* If an invalid **bugId** is provided, then send the following response:
  + status = 404
  + json = { error: `bugId ${bugId} is not a valid ObjectId.` }
* The following data **must** be provided as the body of the request, and read from **req.body:**
  + closed (must be a boolean true/false)
* Create a schema with **Joi,** and use it to validate all of the request data.
* If any data is invalid, then send the following response:
  + status = 400
  + json = { error: validateResult.error }