Função auxiliar:

function typeId(id){

if (id.length >=24)

return new ObjectId(id)

else

return parseInt(id)

}

# BOOKS:

// Alínea 3

router.post('/', async (req, res) => {

try {

let result;

if (

req.body === undefined || // req.body is null or undefined

(Array.isArray(req.body) && req.body.length == 0) || // req.body is an empty array []

(!Array.isArray(req.body) && Object.keys(req.body).length == 0) || // req.body is an empty object {}

(Array.isArray(req.body) && req.body.some(item => Object.keys(item).length == 0)) // req.body contains [{}]

) {

return res.status(400).send({ error: "Request body is empty" });

}

if (!Array.isArray(req.body)) {

const insertFields = {};

insertFields.title = req.body.title;

insertFields.isbn = req.body.isbn;

insertFields.pageCount = req.body.pageCount;

insertFields.publishedDate = req.body.publishedDate;

insertFields.thumbnailUrl = req.body.thumbnailUrl;

insertFields.longDescription = req.body.longDescription;

insertFields.status = req.body.status;

insertFields.authors = req.body.authors;

insertFields.categories = req.body.categories;

result = await db.collection('books').insertOne(insertFields);

} else {

const insertFields = req.body.map(book => ({

title: book.title,

isbn: book.isbn,

pageCount: book.pageCount,

publishedDate: book.publishedDate,

thumbnailUrl: book.thumbnailUrl,

longDescription: book.longDescription,

status: book.status,

authors: book.authors,

categories: book.categories

}));

result = await db.collection('books').insertMany(insertFields);

}

if (result.insertedCount == 0) {

return res.status(404).send({ error: "Book(s) not inserted" });

}

res.send(result).status(200);

} catch (error) {

res.send(error).status(500);

}

});

// Alínea 5

router.get("/:id", async (req, res) => {

try {

const id = typeId(req.params.id); // Ensure `id` is parsed as an integer

let book = await db.collection('books')

.aggregate([

// Match the book document with the specified ID

{ $match: { \_id: id } },

// Lookup comments for the book and set to an empty array if none found

{ $lookup: {

from: "comments",

localField: "\_id",

foreignField: "book\_id",

as: "comments"

}},

// Lookup reviews to calculate average score

{ $lookup: {

from: "users",

let: { book\_id: "$\_id" },

pipeline: [

{ $unwind: "$reviews" },

{ $match: { $expr: { $eq: ["$reviews.book\_id", "$$book\_id"] } }},

{ $group: { \_id: null, average\_score: { $avg: "$reviews.score" }}}

],

as: "average\_score"

}},

// Flatten average\_score array, set to null if empty

{ $addFields: {

average\_score: { $ifNull: [{ $arrayElemAt: ["$average\_score.average\_score", 0] }, null] }

}},

// Project the necessary fields

{ $project: {

\_id: 1,

title: 1,

isbn: 1,

pageCount: 1,

publishedDate: 1,

thumbnailUrl: 1,

longDescription: 1,

status: 1,

authors: 1,

categories: 1,

average\_score: 1,

comments: { $ifNull: ["$comments", []] } // Set comments to empty array if none found

}}

])

.toArray();

if (book.length === 0) {

return res.status(404).send({ error: "Book not found" });

}

res.status(200).send(book[0]); // Send only the first result

} catch (error) {

res.status(500).send(error);

}

});

// Alínea 7

router.delete('/:id', async (req, res) => {

try {

const id = typeId(req.params.id);

const result = await db.collection('books').deleteOne({ \_id: id });

if (result.deletedCount == 0) {

return res.status(404).send({ error: "Book not found" });

}

res.status(200).send(result);

} catch (error) {

res.status(500).send(error);

}

});

// Alínea 9

router.put('/:id', async (req, res) => {

try {

const id = typeId(req.params.id);

const updateFields = {};

if (req.body.title) updateFields.title = req.body.title;

if (req.body.isbn) updateFields.isbn = req.body.isbn;

if (req.body.pageCount) updateFields.pageCount = req.body.pageCount;

if (req.body.publishedDate) updateFields.publishedDate = req.body.publishedDate;

if (req.body.thumbnailUrl) updateFields.thumbnailUrl = req.body.thumbnailUrl;

if (req.body.longDescription) updateFields.longDescription = req.body.longDescription;

if (req.body.status) updateFields.status = req.body.status;

if (req.body.authors) updateFields.authors = req.body.authors;

if (req.body.categories) updateFields.categories = req.body.categories;

const result = await db.collection('books')

.updateOne(

{ \_id: id },

{ $set: updateFields }

);

if (result.modifiedCount == 0) {

return res.status(400).send({ error: "Book not found or not updated" });

}

res.status(200).send(result);

} catch (error) {

res.status(500).send(error);

}

});

# USERS:

// Alínea 4

router.post('/', async (req, res) => {

try {

let result;

if (

req.body === undefined || // req.body is null or undefined

(Array.isArray(req.body) && req.body.length == 0) || // req.body is an empty array []

(!Array.isArray(req.body) && Object.keys(req.body).length == 0) || // req.body is an empty object {}

(Array.isArray(req.body) && req.body.some(item => Object.keys(item).length == 0)) // req.body contains [{}]

) {

return res.status(400).send({ error: "Request body is empty" });

}

if (!Array.isArray(req.body)) {

const insertFields = {};

insertFields.first\_name = req.body.first\_name;

insertFields.last\_name = req.body.last\_name;

insertFields.year\_of\_birth = req.body.year\_of\_birth;

insertFields.job = req.body.job;

insertFields.reviews = req.body.reviews;

result = await db.collection('users').insertOne(insertFields);

} else {

const insertFields = req.body.map(user => ({

first\_name: user.first\_name,

last\_name: user.last\_name,

year\_of\_birth: user.year\_of\_birth,

job: user.job,

reviews: user.reviews

}));

result = await db.collection('users').insertMany(insertFields);

}

if (result.insertedCount == 0) {

return res.status(404).send({ error: "User(s) not inserted" });

}

res.send(result).status(200);

} catch (error) {

res.send(error).status(500);

}

});

// Alínea 6

router.get("/:id", async (req, res) => {

try {

const id = typeId(req.params.id);

let result = await db.collection('users')

.aggregate([

{ $match: { \_id: id } },

// Set reviews to an empty array if it's null

{ $addFields: { reviews: { $ifNull: ["$reviews", []] } } },

// Unwind reviews with preserveNullAndEmptyArrays to handle cases where reviews is empty

{ $unwind: { path: "$reviews", preserveNullAndEmptyArrays: true } },

{ $sort: { "reviews.score": -1 } },

{ $limit: 3 },

{ $group: {

\_id: "$\_id",

first\_name: { $first: "$first\_name" },

last\_name: { $first: "$last\_name" },

year\_of\_birth: { $first: "$year\_of\_birth" },

job: { $first: "$job" },

reviews: { $push: "$reviews" }

}}

])

.toArray();

if (result.length === 0) {

return res.status(404).send({ error: "User not found" });

}

res.status(200).send(result);

} catch (error) {

res.status(500).send(error);

}

});

// Alínea 8

router.delete('/:id', async (req, res) => {

try {

const id = typeId(req.params.id);

const result = await db.collection('users').deleteOne({ \_id: id });

if (result.deletedCount == 0) {

return res.status(404).send({ error: "User not found" });

}

res.status(200).send(result);

} catch (error) {

res.status(500).send(error);

}

});

// Alínea 10

router.put('/:id', async (req, res) => {

try {

const id = typeId(req.params.id);

const updateFields = {};

if (req.body.first\_name) updateFields.first\_name = req.body.first\_name;

if (req.body.last\_name) updateFields.last\_name = req.body.last\_name;

if (req.body.year\_of\_birth) updateFields.year\_of\_birth = req.body.year\_of\_birth;

if (req.body.job) updateFields.job = req.body.job;

if (req.body.reviews) updateFields.reviews = req.body.reviews;

const result = await db.collection('users')

.updateOne(

{ \_id: id },

{ $set: updateFields }

);

if (result.modifiedCount == 0) {

return res.status(400).send({ error: "User not found or not updated" });

}

res.status(200).send(result);

} catch (error) {

res.status(500).send(error);

}

});