NodeJS Exam - Cohort 46

Introduction

You are asked to write a REST API for a second hand market application. In this application, users will be able to buy and sell their used items. This application will allow users to register to the system, search for items, add items for sale, update existing items and remove items from sale.

Guidelines

- Your code should be clean, well structured and readable.
- You can use the provided starter project or create your own from scratch.
- **The API should be implemented exactly according to the specification below.** Use the same endpoint names, HTTP methods, request and response formats.
- Use the HTTP REST API conventions:
    - Appropriate HTTP status codes in the response
    - Appropriate HTTP methods for the endpoints
- For session management, you may use either an opaque session token or a JWT.
- You are expected to handle all edge cases according to the specification.
- You may use Google, Stackoverflow, Github and Generative AI such as ChatGPT. However, you are not allowed to copy and paste entire functions as-is.
- You are **not allowed to discuss** with other trainees or mentors during the exam, or share code during the exam.
- Submit the exam **on time**. Even if you haven't finished everything.
- Please do not share the exam with anyone else.

Submission instructions

- Submit the exam **on time**. Even if you haven't finished everything.
- Zip all of the files and folders with your application.
- Do not include the *node_modules* folder in your zip file.
- Send the zip file to the Education Director.

Specifications

1. The application should use HTTP on port **7890**.
2. User passwords must be stored securely using Bcrypt.
3. Session tokens (or JWT) will be sent in the Authorization HTTP header with Bearer scheme (*Authorization: Bearer <Token>)*.
4. You can store all the information in memory. You are not required to persist the data on disk or external database.
5. All error responses should have the following response format:

```javascript
{
   "error": "string",
}
```

Endpoints

The following is a list of all the endpoints required to implement. The detailed information for every endpoint can be found below this table.

Method	Endpoint	Description	Points
GET	/search	Search for items	2
POST	/users/register	Register a new user	3
POST	/users/login	Login with a user	2
???	/items	Get all items for sale by the logged in user	2
???	/items	Create a new item for sale	3
???	/items/{id}	Update existing item	5
???	/item/{id}	Delete an item from sale	3

A user object will have the following structure:

```javascript
{
  "id": "string",
  "email": "string",
  "password": "string",
}
```

An item will have the following structure:

```javascript
{
  "id": "string",
  "title": "string",
  "sellerEmail": "string",
  "price": 12.34,
}
```

Tests

You are requested to add the two following unit tests:

Endpoint	What to test	Points
POST /users/register	**Valid request returns HTTP success.** Steps: 1. Send *POST /users* request with a valid request object. 2. Check HTTP response status code. Expected: HTTP success status code.	2
POST /users/register	**Invalid request returns HTTP error.** Steps: 1. Send *POST /users* request with an empty object in the request body. 2. Check HTTP response status code. Expected: HTTP error status code.	2

Detailed endpoint specification

/search - Search items

Return a list of items for sale. This endpoint allows users to filter which items to search for according to a keyword. The keyword is optional. If no keyword is given, return all items.

For example: if **query**=car, it should match items with the following titles:

- A new car for sale
- Used red carpet
- USED CAR! BEST PRICE!

Please note that this endpoint does not require authentication.

Endpoint	/search?**query**=*{keyword}*
Method	GET
Requires authentication	No
Request URL parameters	• **query** - show only items with titles that contain the provided keyword anywhere in the string.
Request body	`empty`
Response body	`[` `  {`

	<pre>"id": "string",
	"title": "string",
	"sellerEmail": "string",
	"price": 12.34,
	}
	]</pre>

Checks	• **query** is case insensitive. For example, a query “car” should match items with “CAR” or “Car” somewhere in the title.

/users/register - Registration

Add a new user to the database. Generate a random user ID in a string format.

Endpoint	/user/register
Method	POST
Requires authentication	No

Request body	<pre>{
	"email": "string",
	"password": "string",
	}</pre>

Response body	<pre>{
	"id": "string",
	"email": "string",
	}</pre>

Checks	1. The request must contain email and password fields. 2. Email should be at least 3 characters long and must have ‘@’ sign somewhere in the string. 3. Password should be at least 8 characters long 4. There is no other user with the same email.

/users/login

Authenticate to the application. Verify the email and password combination. On successful login, create and return a session token.

Endpoint	/users/login
Method	POST
Requires authentication	No

Request body	<pre>{
	  "email": "string",
	  "password": "string",
	}</pre>

Response body	<pre>{
	  "token": "string",
	}</pre>

Checks	1. The request must contain email and password fields. 2. User email exists in the database 3. User password is correct.

/items - Get my items

Return a list of items that were created by the logged in user. Do not return items that were created by other users.

Endpoint	/item
Method	GET
Requires authentication	**YES**
Request body	`empty`

Response body	<pre>[
	  {
	  "id": "string",
	  "title": "string",
	  "sellerEmail": "string",
	  "price": 12.34,
	  }
	]</pre>

Checks	None

/items - Create new item for sale

Create a new item listing for sale. Create a new item object, generate a random string ID and auto fill the ‘senderEmail’ field with logged in user email.

Endpoint	/items
Method	???

Requires authentication	**YES**

Request body	<pre>{
	  "title": "string",
	  "price": 12.34,
	}</pre>

Response body	<pre>{
	  "id": "string",
	  "title": "string",
	  "sellerEmail": "string",
	  "price": 12.34,
	}</pre>

Checks	1. Title must be at least 3 characters long. 2. Price must be a number. 3. Price must be bigger than 0 and smaller than 10000.

/items - Update a existing item listing

Update item listing. This endpoint updates the content for a specific item. Item ID and Sender Email will not be updated. A user cannot update an item that was not created by him.

Endpoint	/items/:id
Method	???
Requires authentication	**YES**

Request body	<pre>{
	  "title": "string",
	  "price": 12.34,
	}</pre>

Response body	<pre>{
	  "id": "string",
	  "title": "string",
	  "sellerEmail": "string",
	  "price": 12.34,
	}</pre>

Checks	1. Item ID exists 2. Request contains title and price. 3. Title must be at least 3 characters long. 4. Price must be a number. 5. Price must be bigger than 0 and smaller than 10000. 6. The item was created by the logged in user.

/items - Delete an item

This endpoint deletes the specified item listing. A user cannot delete an item listing that was not created by him.

Endpoint	/items/:id
Method	???
Requires authentication	**YES**
Request body	`empty`
Response body	`empty`
Checks	1. Item ID exists 2. The item was created by the logged in user.

Hints

- You can use '`crypto.randomUUID()`' to generate a random uuid string.
- You can use Number.isNaN to check if a string contains a number. And Number.parseFloat to convert a string to a number.

Bonus [Optional]

+5 points

Make the database persistent between runs. Relaunching the application will keep all the previously added users and items. Save the data in a file(s).