**Code Review (Optional Parameters) - Null Safety**

In the previous section, we named 2 ways where null bugs may occur: null variables and optional parameters. We went through all cases of nullable variables, and found that they were mostly being handled safely.

The second common way that leads to null errors is where null values/variables are assigned to nullable/optional parameters, which occurs with functions and objects. Both functions and objects may allow optional parameters/keys, and if these parameters do not have a default value, we need to see how the function would handle them if they are not passed (i.e. when they are null).

In this section, we will mainly try to identify API endpoints that utilize user input, and then see if any of the endpoint parameters is optional (i.e. can be null). If they do allow optional parameters, and do not perform proper tests to ensure that null values are not used, then logic bugs may occur, which is what we will try to identify.

**Identifying Null Parameters**

JavaScript supports named parameters (e.g. ({param1, param2})) and positional parameters (e.g. (param1, param2)). The main benefit of using named parameters is that we don't need to place them in order correctly and can simply call each argument with the parameter's name. This also allows optional parameters, which accept null as their value, as mentioned above. However, suppose the function accepts an optional parameter, and it did not perform proper null checks and eventually did use the parameter while being null. In that case, it may lead to run-time errors and logic bugs.

The same thing applies to objects, which contain named parameters that we can then be assigned to a function through direct assignment (e.g. const param1 = obj.param1;) or object destructuring (e.g. const {param1, param2} = obj;). This is very commonly used, especially with JSON objects in POST requests (e.g. req.body;), which is much simpler and shorter than assigning each JSON parameter to a variable.

The question is, what if an endpoint expects us to send a certain parameter, and we don't? This is quite normal and is usually handled by verifying the JSON object through schemas (as we've seen before) or any other means. So, if we send an empty body, for example, it should be handled gracefully by throwing an error specifying what exactly was missing. Having any null safety bugs with optional parameters in end-points may lead to serious logic bugs, especially since they will be directly controllable by the end-user.

Checking for missing API/JSON parameters is quite similar to the null checks we mentioned in the previous section, and can suffer from the same pitfalls we discussed previously. So, when it comes to null safety logic bugs, this is the area that is most likely to cause issues.

**Required vs Optional Parameters**

Since we are mainly looking for issues with user input, we can go back to the endpoints that utilize user input, which we listed previously in the Unexpected Input - Code Review section. However, this time, we will be checking whether there are sufficient tests to validate that the input is not null.

We have also previously found out that all endpoints either validate the user input through a schema, unless it is an ID, in which case it would be validated by searching for it in the database and throwing an error if no matches are found. It is always useful to go through all user-controllable input and check whether any of them are missing any checks, but as we have already verified that in the unexpected input sections, we can safely assume that all input will be validated through the database or with a schema.

So, does this mean that the code is safe from null safety issues? Of course not, as schemas may still allow null parameters if not configured properly.

While it is different from one validation tool to another, almost all of them allow required and optional parameters, just like named functions, as we discussed earlier. In this case, the yup package allows us to specify mandatory fields by using the .required() option. If the endpoint allows certain optional parameters, then the .required() option will not be used, and the endpoint should act accordingly, depending on whether the parameter is used or not. This means that we need to identify all instances of optional parameters used in different schemas, and then study those for null safety issues, as the endpoint may not be coded with solid logic, which may lead to a logic bug.

Luckily, we have already identified endpoints that utilize schemas with user input in a previous section. Here they are again, along with the names of their schemas:

* validateCouponCode -> CouponCodeSchema
* validateCartItemDetails -> CartItemSchema
* resetPassword -> passwordResetSchema
* validateUserDetails -> UserSchema

**Exercise:** Check the above 4 schemas for optional parameters (missing the .required option), and then look for those optional parameters in the endpoints that utilize their schemas (also found in the Unexpected Input - Code Review section) to see if they properly check for null values being used those optional parameters.

**Identifying Optional Parameters**

We can now start identifying optional parameters that may hold null values, and then review their functions to see how they handle optional parameters. Let's start with the above schemas, and go through them one by one to see if they support any optional parameters.

The first one, CouponCodeSchema, is quite basic as it only supports one parameter coupon, which is denoted with .required(), meaning it is not optional, and the schema validation test will throw an error if we don't provide it:

Code: js

export const CouponCodeSchema = yup.object({

// coupon must be an md5 hash of the coupon code

coupon: yup

.string()

.matches(/^[a-f0-9]{32}$/i, "Invalid coupon.")

.required(),

});

Next, we have CartItemSchema (that we patched from unexpected input bugs), which has four parameters, but all of them have .required(), meaning none of them is optional. On top of that, the entire object is itself .required(), meaning that if we send an empty cart, it will also error out:

Code: js

export const CartItemSchema = yup

.object({

name: yup.string().required(),

category: yup.mixed().oneOf(["subscription", "exam", "cubes"]).required(),

price: yup.number().positive().min(1).required(),

// in usd

amount: yup.number().positive().min(1).required(), // item count

})

.required();

The passwordResetSchema comes after that, as follows:

Code: js

const passwordResetSchema = object({

// validate mongodb object id

id: mixed((value) => ObjectId.isValid(value)).typeError("Invalid id"),

// validate bcrypt hash

token: string().matches(/[0-9a-f]{32}/i, "Invalid token"),

// validate password

password: string().min(5),

}).required();

We see that while the entire object is .required(), none of the parameters has this option, which is a bit of an odd case. So, we have to provide at least one of them, but it does not matter which one we send, since all of them are considered optional. This is definitely interesting, and we should take a look at it later on.

Finally, we have one of the most used schemas in the whole application, which is UserSchema. Unlike the previous three schemas, all of which were used once each, this schema is used for most user endpoints, like createUser, login, updateUserDetails, and requestPasswordResetLink. All of these functions are sensitive and interesting, which makes this schema high on the list of priorities. So, let's look at its parameters:

Code: js

export const UserSchema = yup

.object({

id: yup.string(),

name: yup.string(),

username: yup.string(),

email: yup.string().email().required(),

password: yup.string().min(5).required(),

registrationDate: yup.date(),

})

.required();

As we can see, four out of the six parameters are optional, namely: id, name, username, and registrationDate. This is quite an odd choice, as we expect parameters like id or username to be required, but perhaps this is due to it being used with multiple endpoints, and not all endpoints require all of these fields. In any case, it is worth shortlisting for our local testing.