# Full Stack JavaScript Technical Challenge

## Task 2: Analysis Challenge

**What do you think is wrong with the code, if anything?**

**Can you see any potential problems that could lead to exceptions**

Require statement are missing

const superagent = require('superagent');

const User = require('./models/User');

const Shop = require('./models/Shop');

It is better to declare the variables as “const”, to reduce the scope of the variables and make sure that are only read values that cannot be modified.

  const invitationBody = req.body;

  const shopId = req.params.shopId;

  const authUrl = "https://url.to.auth.system.com/invitation";

The commented line is not required, because “upsert” option will merge filter and update to create a new entry.

          User.findOneAndUpdate({

            authId: invitationResponse.body.authId

          }, {

            //authId: invitationResponse.body.authId,

            email: invitationBody.email

          }, {

            upsert: true,

            new: true

          }

findOneAndUpdate() function is atomic except if we use the upsert option. Then, if the document does not exist and multiple clients launch the same operation before the new document of User is created, we may end with several Users with the same authId created. In order to prevent that, we need to define authId as an unique index. Doing this, only one operation will be successful.

The following code is wrong, because indexOf() returns -1 in case of failure. If we want to add the value when it does not exist, we need to compare the result of the function with -1 like we have in the second if-condition.

if (shop.invitations.indexOf(invitationResponse.body.invitationId)) {

  shop.invitations.push(invitationResponse.body.invitationId);

}

if (shop.users.indexOf(createdUser.\_id) === -1) {

  shop.users.push(createdUser);

}

shop.save() operation is asynchronous. So, all the changes done in the function may not be ready in the next query.

IF-condition is not handling properly other status values different than 201 and 200. If the invitationResponse.status is different than those values, the result is going to be returned as good.

In order to keep consistency, it will be better to return the final response with its status, as we are doing in another scenarios.

        res.status(201).json(invitationResponse);

**How would you refactor this code to:**

* + **Make it easier to read**
  + **Increase code reusability**
  + **Improve the stability of the system**
  + **Improve the testability of the code**

**How might you use the latest JavaScript features to refactor the code?**

A summary of all the changes to refactor the code is at the beginning of the ‘refactor.js’ file where you can find a refactored version of the provided code.