# Full Stack JavaScript Technical Challenge

## Task 2: Analysis Challenge

## Prerequisites

Please note that this will require JavaScript, Express.js and Mongoose knowledge, as well as an understanding of REST APIs and best Node.js development practices.

## Overview

Below is a Node.js function that a developer has written. It is an express middleware that processes users' invitations to use private shops.

* req and res are the express request and response objects
* superagent is a module that makes http requests and is on npm
* "User" and "Shop" are mongoose models

## Step 1

Analyse the function below and provide answers to the following questions:

* What do you think is wrong with the code, if anything?
  + The superagent is asynchronous, so the *inviteUser* method might return before the superagent’s end() method is processed. Hence, the response might get lost. Using async/wait to solve.
  + The same is true for the moongoose exec() function.

Hence: Using async/await to make sure we have a response that we can return in Express.

* Can you see any potential problems that could lead to exceptions
  + No check and validation on the input values. ‘shopId’, and ‘email’ in the body.
  + Use of chained property access without checking if they are defined, or use of ‘?’, like req.params.shopId. Instead of req?.params?.shopId. (Although, here Express might make sure that those exist.)
* How would you refactor this code to:
  + Make it easier to read
    - Use switch/case instead of if, else if
    - Use temporary variables to shorten property access  
      body.authId and body.invitation.id instead of invitationResponse.body.authId …
    - Use constants instead of return codes for switch/case  
      case HTTP\_BAD\_REQUEST: instead of case 400:
    - Be consistent, for example in the way indexOf is used  
      if (indexOf(invitationID)) versus  
      if (indexOf(ceatedUser.\_id) === -1)
  + Increase code reusability
    - Extract access to persistence layer (Mongoose) in separate module
    - Extract callbacks
  + Improve the stability of the system
    - Input data validation
  + Improve the testability of the code
    - Extract inner callbacks as functions
    - Separate persistence layer
* How might you use the latest JavaScript features to refactor the code?
  + I don’t see any place in the code where ECMAScript 2021 features might really help. We don’t use classes or multiple promises.

## Step 2

Provide a sample refactor with changes and improvements you might make. The refactored code does not have to be executable; it will only be used for discussion.

See git hub superagent\_refactor.js

## Once both tasks are complete...

Commit and push your code from Task 1 and your analysis from Task 2 to your new repository. Then send us a link, we will review and get back to you.

## Good luck!