Summary

The goal is to create a REST API that is able to manage a Starbucks coffee shop inventory. You do not need to implement any frontend, the API is the frontend.

- The inventory consists of categories of drinks.
- Each category has products that the customer can order.
- On any given product, the user can add any number of extras (each with a given price).
- Products have a name, price and stock availability.
- The total price to charge the customer is the cost of the product plus any extras.
- You can define the price values as you like.

As an example, one possible configuration of this inventory would be (but you can choose your own):

- Espresso Drinks *(category)*
    - Latte
    - Mocha
    - Macchiato
    - Cappuccino
    - Americano
    - Espresso
- Brewed Coffee *(category)*
    - Filter coffee
    - Caffe Misto
- Tea *(category)*
    - Mint
    - Chamomile Herbal
    - Earl Grey
- Extras (the client needs to pay more for each extra)
    - Cinnamon
    - Yellow sugar
    - Syrup
    - Whipped Cream

The operations required are:

- Create the inventory structure as mentioned above
- CRUD API for the above structure: products, categories and extras
- Be able to order a drink
    - To order a drink means that the API will compute the price, validate that there's enough stock, validate that the customer has provided enough money for it and return the exchange amount to give back to the customer.
    - Let's say you order a drink that costs 11€ and you only give 10€. That should return something like "insufficient funds".

Submission Criteria:

- Must contain a README with the instructions on how to run

- You can use a non-relational or relational database
- You can use the programming language that you feel more comfortable with
- Follow best development practices and test strategies
- The code must be in a git repository
- Do not spend more than 4 hours. If you spend more than that, please stop and send what you have. It is OK to do only 50% of the exercise. The most important is that 50% is correctly implemented.