




Backend Code challenge

✎ Writer	 Gus Gambarini
🕒 Last edited time	@February 17, 2025 4:53 PM

Have you shopped online? Let's imagine that you need to build the checkout backend service that will support a hand full of products and promotions within a given inventory.

Bellow are numbered requirements for the checkout service; each adding features and complexity to it. **The number 1. requirement is required for the test** and you can pick any or all of the following requirements to add to your solution.

The more requirements you solve the better. Let us know what are the chosen requirements, but watch your time availability.

Requirements

1. Build a checkout service that allows the purchase of these products:

SKU	Name	Price	Inventory Qty
120P90	Google TV	\$49.99	10
43N23P	MacBook Pro	\$5,399.99	5
A304SD	Alexa Speaker	\$109.50	10
234234	Raspberry Pi B	\$30.00	2

- The service must calculate the total of a purchase and the items in it. **It does not process payments.**

2. The service should manage the inventory:

- By adding an item to a purchase the inventory is reduced accordingly
- When Items are not in inventory they cannot be purchased
- When a purchase is not completed, items are returned to the inventory

3. The service should support the following promotions:

- Each sale of a MacBook Pro comes with a free Raspberry Pi B
- Buy 3 Google TV for the price of 2
- Buying more than 3 Alexa Speakers will have a 10% discount on all Alexa speakers

Example Scenarios:

Scanned Items: MacBook Pro, Raspberry Pi B

Total: \$5,399.99

Scanned Items: Google TV, Google TV, Google TV

Total: \$99.98

Scanned Items: Alexa Speaker, Alexa Speaker, Alexa Speaker

Total: \$295.65

4. The service will Integrate with other services:

- The service will communicate/integrate with other service/systems.
- It must consider communication with a order processing service/systems, that will provide the packaging and delivery of orders.
- It must consider communication with a payment service/systems, that will process payments or failure of payments.
- Consider that the other services/systems are going to adapt to any interface, payload or protocol of choice.

Important

Develop it in Golang and consider the following:

- Unit Tests - we are not expecting test coverage (1% is fine), but choose any part of the solution to show your approach to unit tests
- API design - the API could be used by a client application to process the checkout of the given products.
- Persistence - Any persistence/DB of choice is accepted, **including in memory storage**. But it is important to show how you approach ACID or BASE concepts in your solution.
- Concurrency - Identify and handle possible concurrency scenarios within the system solution.
- Provide a github (or other git repo SaaS) link with a README documentation.

Thank you for your time and we look forward to reviewing your solution. If you have any questions, please feel free to contact us. Please send us a link to your git repo.