Vend-O-Matic: Follow-up Questions

How did you think about associating/generating an inventory sequence?
I mirrored an average vending machine's IDs such as A1, A2, A3… And from that point I used said ID to create a dictionary of associated values of the item such as price, amount(remaining), and name.
Support your decision to store the Inventory data as you did; what influenced your design/structure? What other storage options did you consider? Were dependencies brought in (why or why not)?
Trying to minimize the amount of dependencies I used an in memory JSON file to mimic a JSON response from an actual database. This structure allowed my project to stay smaller in dependencies while mimicking a DB.
What other HTTP verbs would you consider for inspecting whether or not an inventory item existed?
Not related to inventory but, POST makes more sense for inserting a coin as PUT is idempotent, calling it once should have the same effect as multiple requests. POST would be more applicable in this case as we are adding coins and not replacing the values with the new total of coins. https://developer.mozilla.org/en-US/docs/Glossary/Idempotent Instead of GET, HEAD could be implemented mirroring the 'X-Inventory-Remaining' header from the /inventory/<id> PUT and removing the body of the response. But this isn't the best use case as the current GET uses a body and not a header.
If we removed the requirement to accept quarters only; what changes would you make to support money management in the machine? Support for penny-accuracy to provide correct change. Support correct change. Some kind of bucketing for coin inventory, challenges with returning correct change.

To support multiple denominations, I'd track a coin/bill inventory in the machine (e.g., a map of denomination-in-cents to quantity). I'd represent all prices and balances in an integer representing cents to avoid floating-point issues (e.g., $5.50 = 550). To return change, I'd iterate from the largest denomination to smallest and compute
how many to return using floor division, updating the remainder with modulo. If coin inventory is limited, I'd cap the returned count by what's available (e.g., min(remaining // denom, available[denom])). For example, with 550 cents remaining and a max return denomination of $1 (100 cents), I'd return 550 // 100 = 5 dollars, then use the remainder 550 % 100 = 50 and return 50 // 25 = 2 quarters.

What improvements would you make if we wanted to have a single host serve more than a single physical machine?

I would separate out the machines by IDs as I currently do and tie inventories to the machine IDs so if we were using a SQL DB I would make a machines table that would have columns for change, and an inventory id referring to an inventory table containing the cost of drinks, their IDs and the amount remaining

What would you do if asked to synchronize inventories across linked machines?

If the vending machines shared a linked inventory that fulfilled items across machines, I would centralize inventory ownership so each machine references a shared inventory by inventory_id. To prevent two machines from ordering the last item at the same time, I would use a reservation system where selecting an item attempts to reserve inventory only if it is available. Reservations would be released when the transaction ends or after a timeout if the transaction is abandoned. If a reservation cannot be created, the machine would return an out-of-stock response to the user.