Test plan

Introduction

This test plan is used to test the taco food truck and take order online, deliver the orders to customers smoothly and as quickly as possible.
The objective is, our developers' code can interact well with

1. Customer orders(waits at most 20 orders)
2. Planning REST_API
3. Refueling
4. Buying groceries when supplies run low
5. Depositing money at the local bank since it is not safe to carry loose cash
6. cook inside vehicles ONLY when stationary
7. Exceptions and how to handle

Test Coverage

Normally we should have a system designment here(document or link), and the tests should cover all the conditions and processes of the designment.

Test resources

QA team members will be resolved: WuYi

Below need to be check with product owner and how many we need to test
Hardware and system: OS/website deployment
Software: our developers's software/website and planning REST_API
Test client: browsers/mobiles (What we have and how customer will order our taco food)

Test cases/scenarios

(Normally it's another document or excel, which contains the details test summary, steps, etc, to be simple we put them here as it's the core of the homework)

Please note:

1. Things would be complicated if there are N trucks, according to the assignment, using 'A taco food truck', so we only consider 1 truck.

2. Since we have one truck to do all the things, there are many race conditions, the developer might need to define priority for some tasks, e.g. refueling should be high priority obviously, E.g The truck is almost out of oil, now it still have order to deliver(place A), but it also need to refuel in place B, the oil may only enough to one of them, now even the truck is nearer place A, it still should go to place B to refuel. We might think many scenarios like this.

Test scenarios:

1. Verify that the user can access the taco food orders online, website is good.
2. Verify that the planning REST_API is accessible.
3. Verify that when there is new and first order, the truck is stationary, the cook should start right away.
4. Verify that it is allowed to cook inside vehicles only when stationary. Test that when there the truck is moving, the cook should not start anyway, even the order count is more than 20.
5. Verify that the REST_API works expected: calculates the best route. Simple case, the truck is in one street, 3 orders and need to deliver to 3 places, A, B, C in the same street from the near to the distant. The REST_API should visit A first to deliver order, then B and C.
   Complex case: need to deliver N(N<=20) orders in N different places, check that the route is actually the best(May need to get a verifying algorithm about this)
6. Verify that the REST_API works expected：for fastest travel and for shortest travel. Consider that we need to deliver 20 orders to one customer ASAP, two routes: the normal way distance is 10 miles, speed limit is 50 miles/hour, needs 12 minutes; the highway is 15 miles but speed limit is 100 miles/hour, needs 9 minutes. The REST_API should choose the second route if we selected the fastest travel option.
   Or, two routes, the first one is shorter, but the traffic condition is bad and actually will cost much more time, then REST_API should choose the longer but faster route.
7. Verify feed of incoming orders and how it interacts with the planning REST_API. Usually the developer's automation system wait a few orders(at most 20) and cook, and then call the planning REST_API to deliver, but since we should not allow the customer wait too long, e.g. 1 hour, the case is, if there is 1 order, wait even 1 hour but no new order, the system should need to call the REST_API to deliver right away.
8. Verify that the truck can waits for at most 20 orders before starting deliveries. Consider that the order comes every minutes, there are 20 orders now, even there is a new order comes and the deliver place is the same as the 20th, but since it need a few time to cook, the truck should start to move to deliver by calling the planning REST_API. The 21st order need to wait after the delivers are completed.
9. Verify the system works well when need refueling. There are N(N<=20) orders and N-1 orders have been delivered, the truck need to refuel, the oil is only enough to one place: the last order destination, or the gas station. The truck is expected to go to the gas station first as the priority is higher, or else if the truck deliver the order first, it will run out the oil and couldn't arrive the gas station.

Or, if a truck is running out of gas, it still can deliver all of the N orders(consider the order is usually around the truck, not too far away), the expected result, the system will pass N+1 places to the REST_API, and let the REST_API control the routes, in this case, the truck should deliver the order and refueling by the way.
For this scenario, it depends on when need to refuel, how long the truck can still drive(or on average how many orders can still deliver).

10. Verify that the truck will buy groceries when supplies run low. There are N orders to deliver and will pass N+1(the grocery place) to the REST_API, and the truck will buy groceries by the way. Since even after buying the groceries, if the truck has order to deliver and moving, cooking is not allowed, so priority should be similar.

11. Verify that the truck will deposit money at the local bank since it is not safe to carry loose cash. There are N orders to deliver and will pass N+1(the back place) to the REST_API, and the truck will buy deposit by the way.(It depends on how important the owner thinks depositing money at the bank, or how safe in the city, e.g. if the owner wants to depositing the money once the number reaches certain amount, then it’s a separate task and pass the place to the REST_API.

12. For senario #9-11, if the gas station/supply place/bank is not specially defined, and there are many of them can be choose, e.g the truck can refuel in any gas station in the city. Then first of all, the hired developer should call the REST_API, or do it himself, to choose one of the place which near one of the order destinations. Consider #9, there are N orders(N<=20), and the city has M gas station, the REST_API should be able to choose one gas station which the distance is nearest to one of the N order places, or choose one gas station which in the N order routes path through(best).

13. Mixed and extreme conditions. E.g Test the system works well when need refueling, buying groceries, depositing money at local bank and has a few orders to deliver at the same time. Consider the oil is enough to do all the work, and priority of the 3 tasks is same as delivering orders, pass all the places to REST_API.
If oil is not enough for all tasks but can do one or two tasks before go to gas station, it needs to define the priority, is depositing money more important? Or if one of the order has higher priority as the customer already waits long and complain

14. Exception Testing, what if business is very good, e.g 100 or even 1000 orders comes, need too much time to cook, and also since need time to deliver, shall we interrupt the order system UI to customer, something like a restaurant is full and cannot serve customer temporarily.
Also, 100 orders comes shortly, which first 20 should be delivered, group by locations, or by order time?

15. What if the REST_API service is down or network is failed, any other service backup? Shall the business closed until the service back to normal?

Schedule

Test stat time
Test end time
Details of test process to be updated here.

Test Results

Reference: Test result doc

Bug tracking/Debugging

Reference: Excel or bug docs put here.