Engineering Interview Assignment (Golang)

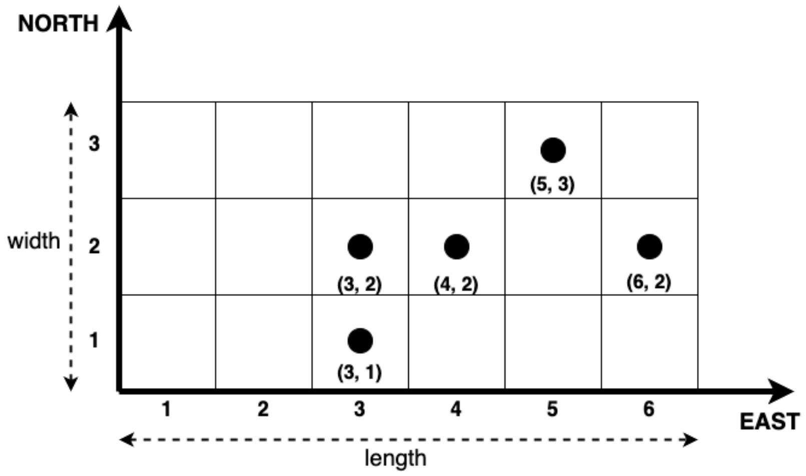
Dear Candidate,

We are pleased to tell you that you have been selected for our Technical Interview step. As an engineer in backend, you play a critical role in building the backbone of our application, ensuring scalability, security, and seamless functionality. We require someone that is able to write high quality, secure, scalable, and high performance code.

In this assignment, we aim to assess your fundamental computational problem-solving skills and your capacity to adapt to our technology and standards.

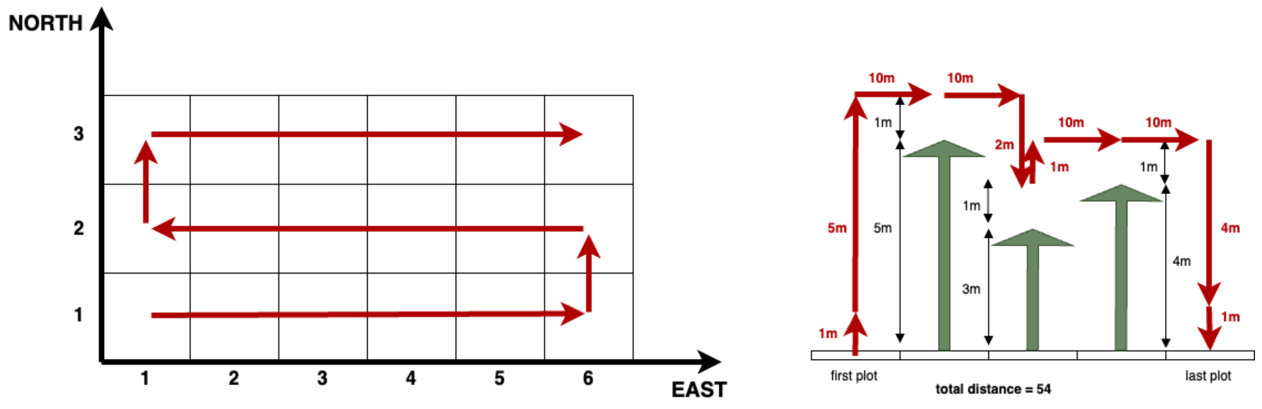
1. Background

Hypothetically, your task will be to develop a simple plantation management and drone patrol planning software for our customers. This software will enable our customers to record their plantation estate and palm trees in the system. Our hypothetical customers have unique requirements: their estates are always square or rectangular, made up of perfect 10x10 meter square plots. Interestingly, they consistently plant their palm trees right in the center of each plot. That way the closest distance between trees is always 10 meters.



The provided image above details our plantation modeling method. The `length` and `width` represent the estate's dimensions from West to East and South to North respectively, in 10-meter increments. The depicted estate consists of 18 plots, with a length and width of 60 and 30 meters respectively. It contains five trees, situated in the middle of plots (3,1), (3,2), (4, 2), (6,2), and (5,3). In addition to the plot location, our system allows customers to record the tree heights, which can range from 1 to 30 meters. The type of seed used is very unique, causing the tree heights to always be integers.

We need the drone to regularly monitor the plots for weeds or pests in our drone planning. Our customers possess a unique drone that can only fly along one of the three axes at a time: vertical, east-west, or north-south, in either direction.



The drone pilot will begin the routine at the southwestern-most plot (1,1), regardless of whether it is empty or has a tree. From there, the drone will move east to the easternmost plot before moving north one plot. Next, the drone will travel west until it reaches the western-most plot, then it will move north by one plot. This pattern is repeated until the drone reaches the final plot. The left part of the image above shows the horizontal traveling pattern.

To monitor a plot, the drone needs to be precisely 1 meter above the empty plot or tree to get data. For the first plot, the drone must fly from ground level to a position exactly 1 meter above the plot or tree to get the data. After monitoring the last plot, the drone should descend back to ground level. The drone should not crash into a tree in the paths and should follow the minimum number of axis pivoting.

The right part of the above image shows a sample traveling pattern for an estate with 5x1 size and 3 trees. The trees are located in plot (2,1), (3, 1), (4, 1) with height 5m, 3m, and 4m respectively. It shows the total traveling distance is 54 meters, 40 meters horizontally and 14 meters vertically.

Your task is simple, create an algorithm to calculate the total drone traveling distance.

2. Requirements

Below are the functional and non-functional requirements that you need to fulfill.

2.1. Functional Requirements

You are to build a 4 HTTP API endpoints:

2.1.1. POST /estate

This endpoint creates and stores new estate in the database. This will receive two integers `width` and `length`, representing a palm oil plantation estate. The integers will represent the size of 10x10m2 plots where a palm tree is planted. For example, the width of 5 and length of 10 represents 50 10x10m2 plots. The integers must be from 1 to 50000.

YouThis endpoint will return status `201` containing data of a string of UUIDv4 representing the estate. It will return HTTP status `400` if it receives invalid value or format (e.g. negative or zero value, string, etc).

Below is correct format input/output

```
curl -XPOST --data '{"length": 10, "width" 10}' http://localhost/estate
{
   "id": "aaaaaa-bbbbbb-cccccc-ddddd"
}
```

2.1.2. POST /estate/<id>/tree

This endpoint stores tree data in a given estate with the ID `id`. It receives three positive integers: `x`, `y`, and `height`. The `x` and `y` represent the plot location on the West-East and South-North axis respectively. The `height` indicates the tree's height, which must range from 1 to 30 meters, inclusive.

This endpoint will return status `201` containing data of a string of UUIDv4 representing the tree. It will return HTTP status `404` if the estate is not found. Otherwise it returns `400` if it receives invalid value or format such as coordinate out of bound, plot already has tree, negative number, zero.

Example of correct format input/output

```
curl -XPOST --data '{"x": 10, "y" 10, "height": 30}' http://localhost/estate
{
   "id": "aaaaaa-bbbbbb-cccccc-ddddd"
}
```

2.1.3. GET /estate/<id>/stats

This endpoint will simply return the stats of the tree in the estate with ID `<id>` The stats contains the `count` of the trees, `max` height of the trees if any, `min` height of the trees if any, `median` height of the trees in that estate if any. If the estate has no tree, return `0` for all values. On success it will return status `200` . Otherwise, it will return `404` if the estate is not found.

Example of correct format

```
curl -XGET http://localhost/estate/aaaaa-bbbbb-ccccc-dddd/stats
{
   "count": 0,
   "max": 0,
   "min": 0,
   "median": 0,
}
```

2.1.4. GET /estate/<id>/drone-plan

This endpoint will simply return the sum distance of the drone monitoring travel in the estate with ID `<id>`. On success it will return status `200` . Otherwise, it will return `404` if the estate is not found.

Example of correct format

```
curl -XGET http://localhost/estate/aaaaa-bbbbb-ccccc-dddd/drone-plan
{
   "distance": 200
}
```

2.1.5. BONUS: GET /estate/<id>/drone-plan?max_distance=<max_distance>

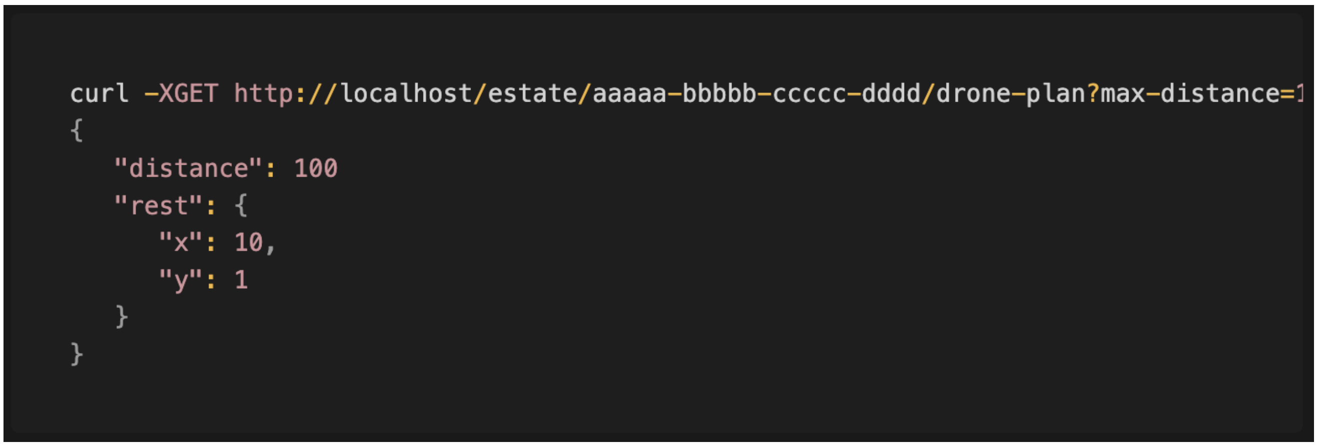
This is a bonus task. You don't have to work on this. But if you do, you must impress us with the correct answer.

In this bonus task, the endpoint in 2.1.4 will take an additional optional argument, `max_distance`. This represents a positive integer indicating the maximum sum of vertical and horizontal distance, in meters, that the drone can travel with its main battery. After reaching this maximum distance, the drone will do a forced landing with its backup battery and wait for the pilot to recharge its main battery. If the endpoint receives this argument, it will return `rest`, which contains the `x, y` coordinate of the location where the drone will first land. Here, `x` and `y` represent the positions along the West-East and South-North axes, respectively. If the max distance is further than the sum of distance, return the last point coordinates.

Should it receive an invalid value of the parameter, it will return HTTP Status Code `404` Bad Request.

It will return `404` if the estate is not found.

Example format of the output.



```
curl -XGET http://localhost/estate/aaaaa-bbbbb-ccccc-dddd/drone-plan?max-distance=1
{
   "distance": 100
   "rest": {
      "x": 10,
      "y": 1
   }
}
```

2.2. Non-Functional Requirements

Here are some non-functional requirements that your submission must meet.

2.2.1. Adaptability

You will be working with the same technologies and libraries that we currently use, as you will be involved in legacy projects. You'll begin with the Starter Project that we will provide. The technologies currently in use for this project are as follows:

1. Makefile
2. Docker & Docker Compose
3. Go v1.22
4. PostgreSQL
5. OpenAPI v3 & Generated Code
6. GoMock
7. Echo Go Web Framework

You may add additional libraries on top of those libraries but you may not change the intended library, e.g. from Echo to Gin web framework.

2.2.2. API Definition

You need to write the API Definition using OpenAPI v3 definition in the `api.yaml` file located in the project root directory. This file generates the stub code and type definitions, which should be utilized in your code. A sample Hello World API definition is also provided as a reference. The API definition has to follow the functional requirements above.

We are using this OpenAPI as a contract between our backend and apps system. The way it works, the OpenAPI will be used to generate server stubs and types in the backend and client and types in the frontend. Once you run the `make init` you can see the generated OpenAPI codein the `generated` directory. You **must** utilize the generated code since we want to assess how well you adapted with our existing standard.

2.2.3. Database Design

You need to design and write your database to store and retrieve the estate and tree data. You need to write the definition in the `database.sql` file. We will use PostgreSQL as the database.

2.2.4. Testing

We are building a culture of testing in our engineering team. We value candidates who understand the importance of writing tests. Please write unit tests for your functions and methods, as well as functional tests for your API endpoints. In the `Makefile` provided, there are commands to run the unit. The `make test_unit` command runs the unit tests. You should aim for at least 70% unit test code coverage.

2.2.5. Code Organization and Readability

Your code will also be evaluated based on the organization and readability. We value clean, organized code with clear comments and documentation. It's important that your code can be easily understood by other engineers. Please ensure your variable names are descriptive and your code follows the Go formatting and style guidelines.

3. Submission Guidelines

Please download the Starter Project in the link below and start your submission from the project.

https://storage.googleapis.com/648010c1-f244-4641-98f2-73ff6c1b4e99/BackendGolangV2024-20240429.zip

You should be able to finish the task within 5 to 10 hours. The problem set author finished this within 3 hours.

You must upload your code to a new GitHub repository. Please provide the HTTPS or SSH URL (for instance, `https://github.com/example/example.git` or `git@github.com:example/example.git`) to our recruitment team. You can choose to create a public or private repository. If you choose a private repository, please add our reviewer GitHub account `sawitpro-tech-recruitment` for code access.

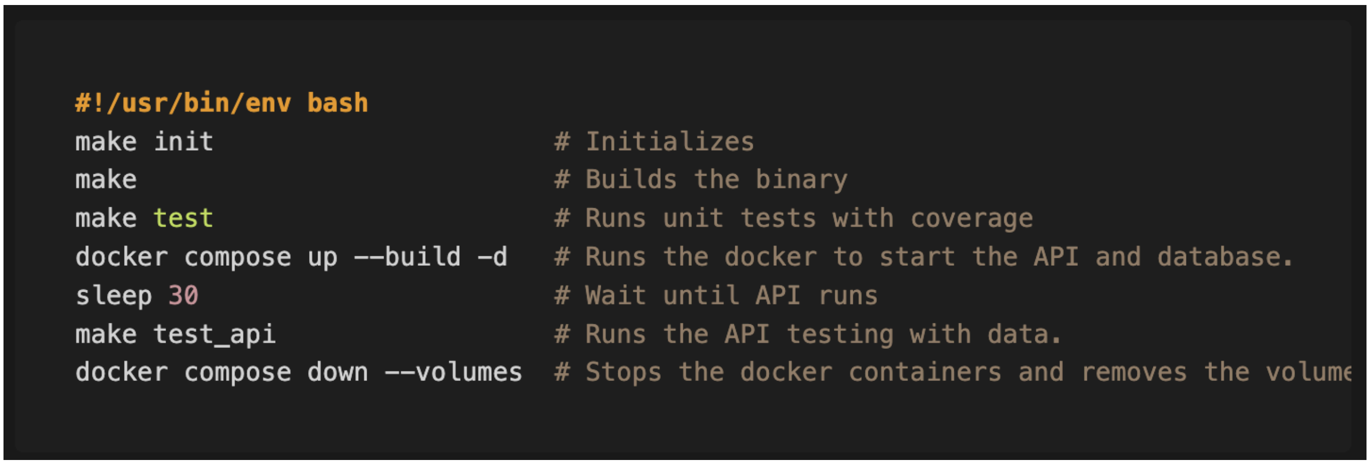
Your code will be reviewed using Ubuntu-based / Mac-based computers. The laptop will have the following installed

1. Go v1.21
2. Node v20.21

3. Docker v20.10

4. Makefile

Your code will be run using the already provided command below. The Docker should run the API in URL `http://localhost:8080/`



```bash
#!/usr/bin/env bash
make init                      # Initializes
make                           # Builds the binary
make test                      # Runs unit tests with coverage
docker compose up --build -d   # Runs the docker to start the API and database.
sleep 30                       # Wait until API runs
make test_api                  # Runs the API testing with data.
docker compose down --volumes  # Stops the docker containers and removes the volume
```

WARNING: We will be testing your code automatically. If it does not pass the above commands on your own computer, please do not even bother submitting it.

4. Evaluation Criteria

We will evaluate your code based on both the correctness of the functional requirements and the quality of the non-functional requirements.

Looking forward to receiving your code! Have fun!