We would like to create a car parking management system. A car parking lot has many slots. The management system stays at the door and controls the entry of a car based on the available parking information. Every slot is numbered from 1 - N. The lot in our discussion has a total of 150 slots. You can consider the lowest numbered slot is nearest to the entry.

The system will have several APIs-

1. For parking- takes in car number, color and type, finds a nearby slot that is empty and park the car there - returns the occupied slot number as response
2. For unparking - takes in a slot number, checks if it is already occupied and empty the slot- returns the car info that comes out
3. For finding the parking stat - shows how many slots are available
4. For finding cars of same type - returns a list of cars of same type e.g. sedan, suv
5. For finding a car with a number - returns the car with matched number if it's parked in the lot

Requirements

- The service has to be written in nodejs using express(or koajs) as framework
- To be able to call parking API some sort of authentication has to be provided, a basic auth implementation with JWT token should suffice
- Use any database you like. Possibly include the architecture of the service like sequence diagrams(to show API flows) and an erd/relationship between entities of your database design. If you want to simply do it using some json as db, that's fine too eg. https://github.com/typicode/lowdb If you use firestore, we will be super happy! GCP provides some free credits btw.
- Do necessary validations and error handling(please!)
- Make sure you provide a ReadMe with running instructions as well as your API spec- request format and response format for each of the APIs
- Include integration tests as well(you are expected to write at least 1 test case for each of the APIs to confirm your API works- we may just run your tests and consider your system is working!)
- Share the code with us in a github repo
- You should but not limited to-
  - write modular and extensible code(here's a sample structure)
  - use any libs of your choice
  - follow single responsibility principle
  - avoid code duplications
  - use software design patterns wherever appropriate
  - separate persistence and application logics

Bonus Points

- Deploy the service in either heroku or any free tier from gcp or aws using docker
  or,
- Deploy the service using kubernetes in https://kubesail.com/ or any free kubernetes cluster provider or even using minikube in local and have proper scripts for us to be able to run the same in local (double bonus points!)
- Create a frontend using React to test out the entire system