**README FOR INTERVIEW TEST**

1. **Project’s tools and their version**

* Front end:
* IDE: VS Code version 1.88.1
* Frameworks: ReactJS, NextJS(app router), Ant design, SASS, typescript
* Additional dependencies and versions: check package.json for detailed information
* Port: localhost:3000
* Note: I will not send the node\_module folder as I deemed it to be unnecessary, if you need it however, feel free to contact me
* Back end:
* IDE: Pycharm 2024.1 Community Edition
* Language: Python 3.12
* Frameworks, dependencies and libraries: Flask 3.0.3 and its distributions, psycopg2
* Database: Postgresql version 16.2, with pgAdmin 4 version 8.2 that comes with the installation (I do not know how to send my database to you, so you need to create a new one)
* Port: http://127.0.0.1:5000

1. **Required data**

* Database:
* Host: localhost
* Database name: interview
* User: postgres
* Password: admin
* Port: 5001
* Database’s schema:
  + Tables: levels, parking\_slots
  + Table levels’ columns:
    - level\_id: int (Primary key)
    - total\_car\_slot: int
    - total\_motorbike\_slot: int
* Table parking\_slots’ columns:
* slot\_id: varchar(15) (Primary key)
* level\_id: int (Foreign key reference level\_id FROM levels)
* plate: varchar(20)
* is\_car: boolean
* park\_date: timestamp without time zone
* Commands to run front end and back end:
  + Front end: npm run dev
  + Back end: flask --app main run
  + **Note: database needs to be connected first before running back end**

1. **Project’s assumptions**

* As the garage’s number of levels as well as the layout for each level is still undecided, the slot id for each slot will be saved under the format <level\_id-incremental\_id-M> for motorbike slots and <level\_id-incremental\_id-C> for car slots, with incremental\_id being a continuous integer starting from 1 and end at the total slots for each type of vehicle (For example: if level 1 there are 2 slots for car and 3 slots for motorbike, the slot\_ids would look like [1-1-C, 1-2-C, 1-1-M, 1-2-M, 1-3-M]
* It is assumed that the level\_id and slot\_id is arbitrary and it can mean anything. Such as the ground floor having the level\_id of 2, the 2nd floor with the id of 1 and so on. The same for slot\_id as slot\_id of 1 can be the nearest slot to the exit, the furthest slot to the exit or in the middle of the level. It is up to the owner to decide the rules for the level ids and the slot ids, as well as the layout of each level to assign the slot ids of that level to
* The simulation will first find the nearest free slot that matches with the vehicle’s type to assign it to. It will first prioritize the level with lowest id, then the slot with lowest id.
* The simulation assumes that there is only one machine running the program/website, which means there is only one entrance and one exit and they both share the same computer and operator. As such the simulation doesn’t take into account of concurrent queries

1. **Potential improvements**

* Implements a card system that users have to pay a fee at the start of a period to use it for the remaining time of the period and users can continue to use the card as long as they pay before the payment deadline. This system can also gives users an overdue period, which can be a couple of days, after the card expires to renew it. Users with the card will have the assigned slot kept to them even after they exit the garage then re-enter after some time. As such, the fee, in the long run, will come out slightly more expensive compared to the amount regular users have to pay to park, but the regular users will lose their slot after leaving the garage.
* Database structure revision. As there are many variables at the moment, the database is very crude and rudimentary. After the garage has gone into operation and is stable, we can revise the database and come up with a more efficient one, both for storage as well as querying for data.
* Handling concurrent queries. Most queries in the simulation will not be affected when there are more than one transaction to the database. However, the assigning slots to vehicle may encounter issue where two transactions find the same empty slot, so they will proceed to assign the vehicle to the same slot id with unpredictable consequences. The solution may be to assign the slot in the first place, by turning the finding the nearest empty slot into a sub query, with its result being used for the main query of assigning a vehicle to a slot. Example: **UPDATE parking\_slots SET plate =%s, park\_date=%s WHERE slot\_id = (sub query to find the nearest suitable slot)**