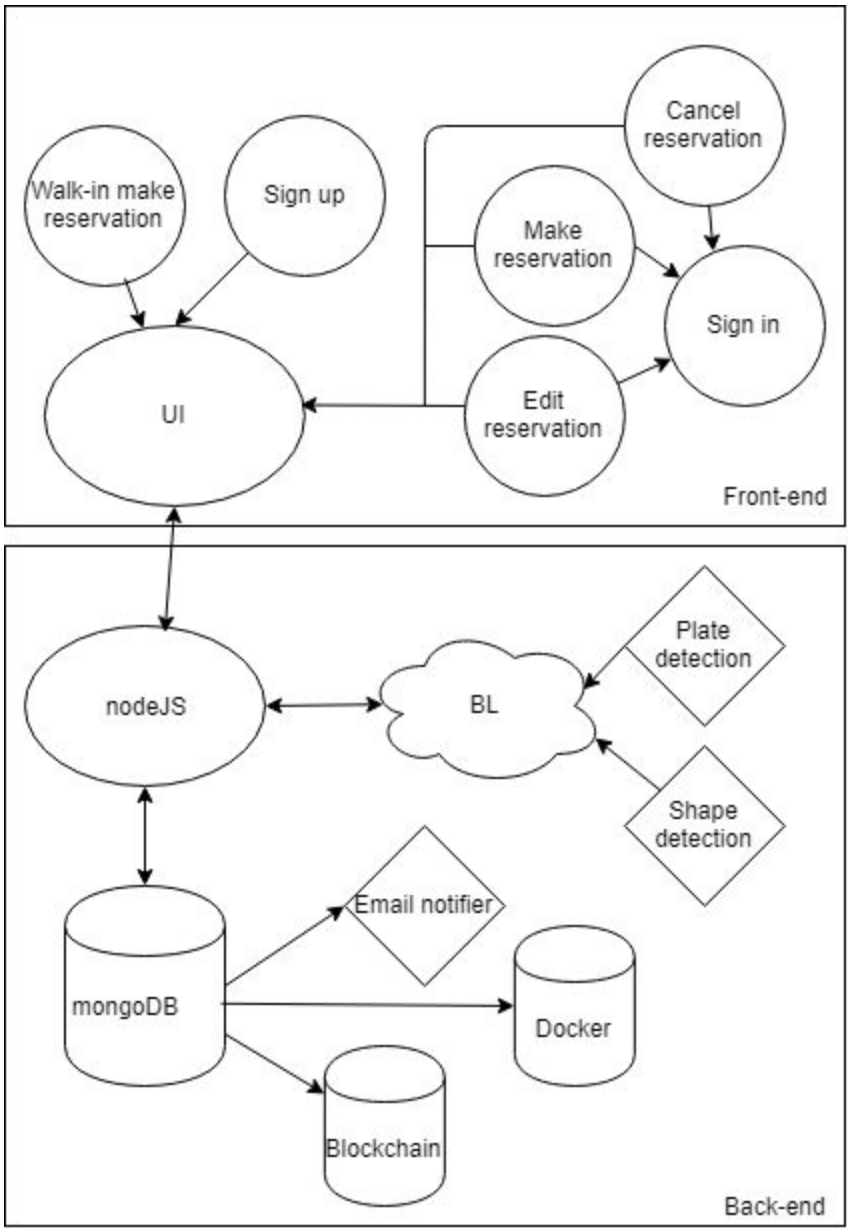
# Integration testing




**UI** – User Interface module, which is visible to the end user, where all the inputs are given.
**BL** – Is the Business Logic module, which has all the calculations for the price and assigning the slot for vehicles based on their shape( big or small).
**Blockchain** - Use for storing database to prevent unusual attack from hackers.
**mongoDB** - Is NoSQL database, to store the customers' data.
**nodeJS** - node js is used for interacting between database and subsystems.
**Email notifier**- Use to notify the customers about their reservation’s information.

In this Integration Testing, we use the Top-down approach to test the application. This approach starts from the top-level layer, which is front-end, to the lower layer, which is the back-end layer.

The diagram is divided into two significant layers: front-end and back-end. The Top-down approach will test the application in the following flow: IU( main page, login page, etc.) to the server running by nodeJS. After the data from users is sent from front-end to the server, the server (nodeJS) manages data and stores it to MongoDB. MongoDB is considered a center core in the back-end layer.

NodeJs also responses to getting the data from MongoDB and sends it to BL modules. In combination with the data from the plate and shape detection subsystem, the BL calculates the price based on the given data. After that, BL sends the result back to nodeJS to update database and email notifier module to send email to customers about the order.

To secure the payment data, the blockchain is coming to play. Blockchain module will access the MongoDB to take the payment data and secure them by adding more blocks to the blockchain. The goal of the blockchain is to prevent any changes in MongoDB. For example, if hackers distort the database, the payment details stored in the blockchain must be the same as the original data.

# Blacklisted License Plate

The goal is to integrate the applications of garage which are the customer data and the blacklisted license plate with the my NoSQL MongoDB database in the end, our group has managed to send the email when the customer reserves and check if the current customer license plate is blacklisted or not in the final demo. However, the first problem that we encounter is which database should we use or what should be the middle-end between the database with the user interface. At first, we intend to use django with MySQL and use the python as our backend main code ; however, we have to use the MongoDB with the online database to deal with and node js is our main code but we encounter the second hard part is our team hasn't had the experience to deal with the database and nodeJS before so we have to take time to learn.

Once the issue was solved, we were able to retrieve the customer data when the customers register or reserve and upload the blacklisted data to update the blacklisted database every time. Hence, after all these things we were able to connect all the things through the database due to the blockchain which needs to complete the payment and record that,the blacklisted to determine the current car is a blacklisted car or not and the docker to share information through the garages that used our services.

# Docker

Initialling while integration the start error was because liblxc can’t traverse to the container’s path. It was fixed chmod +x /root /root/go

Integration tests sometimes rely on external resources. In this case it was Mongo DB, an external service, an SSH server. When I ran a test that use such a service, I ensured I start with a clean state. This is where Docker is useful. Docker allows to quickly spawn a new instance of a service. I ran new docker instance at the beginning of the test and stop it at the end. This way the tests always run in the same context.

# Blockchain

The intended use for the blockchain originally was to secure all customer data. I quickly realized that pure, raw data on the blockchain is very messy and expensive to handle. Another issue I found was that once the data was on the blockchain, I could not change it later on. This instantly ruled out placing data such as credit card numbers, license plate numbers and email addresses. This is because users sometimes change this data. I thought perhaps I could just add another block and just take the last addition, but I ruled this out due to the huge amount of waste this would create. In order to secure the mutable data, I chose to use encryption and decryption. So that at least, there would be some security. The only data that was chosen to be stored on the blockchain itself was payment information: information such as the transaction ID of the customer and the user ID of the customer. I thought perhaps this would make sure no reservations would be able to be changed once they are made. I quickly realized that there was a larger overall use of this. I could turn this all into a massive payment system using this payment information. I integrated my code to the database a few group members created, and I fetched information such as the transaction ID, user ID, and the new variable, the cost. Using the cost, I was able to do the accounting for how much money each customer owed at any given time. The only trouble I had at this point was to make sure that the database was not skipping any transaction ID’s. I quickly realized that a skipped transaction ID would mean a missed block. And a missed block in the blockchain would throw errors.