VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING (CO3001)

Assignment - Urban waste collection aid - UWC 2.0

Task 2

Advisor:	Phan Trung Hieu		
Group:	One		
Students:	Nguyen Tien Phat	- ID: 2011797	- L02
	Pham Thi Hong Hieu	- ID: 2020025	- L01
	Nguyen Hoang Quang Khanh	- ID: 2013458	- L02
	Nguyen Tuan Kiet	- ID: 2013577	- L02
	Nguyen Trung Nghia	- ID: 2010448	- L01
	Duong Thai Thinh	- ID: 2014589	- L02

HO CHI MINH CITY, FEBRUARY 2023

Member list & Workload

Task 2

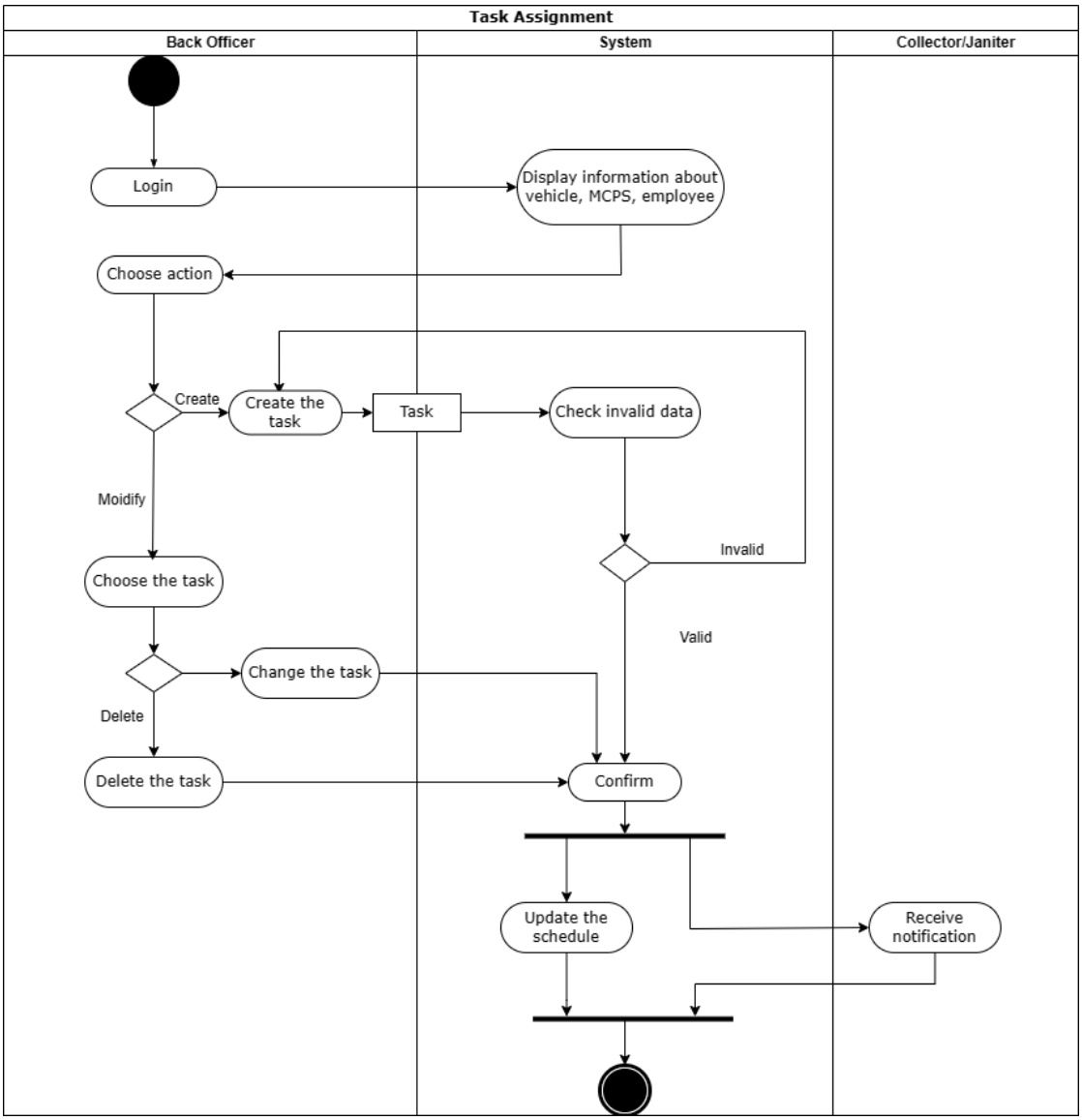
No.	Fullname	Student ID	Tasks
1	Nguyen Tien Phat	2011797	Class diagram & UI
2	Pham Thi Hong Hieu	2020025	Sequence diagram
3	Nguyen Hoang Quang Khanh	2013458	UI
4	Nguyen Tuan Kiet	2013577	Sequence diagram
5	Nguyen Trung Nghia	2010448	Activity diagram
6	Duong Thai Thinh	2014589	Class diagram

Contents

1 TASK 2

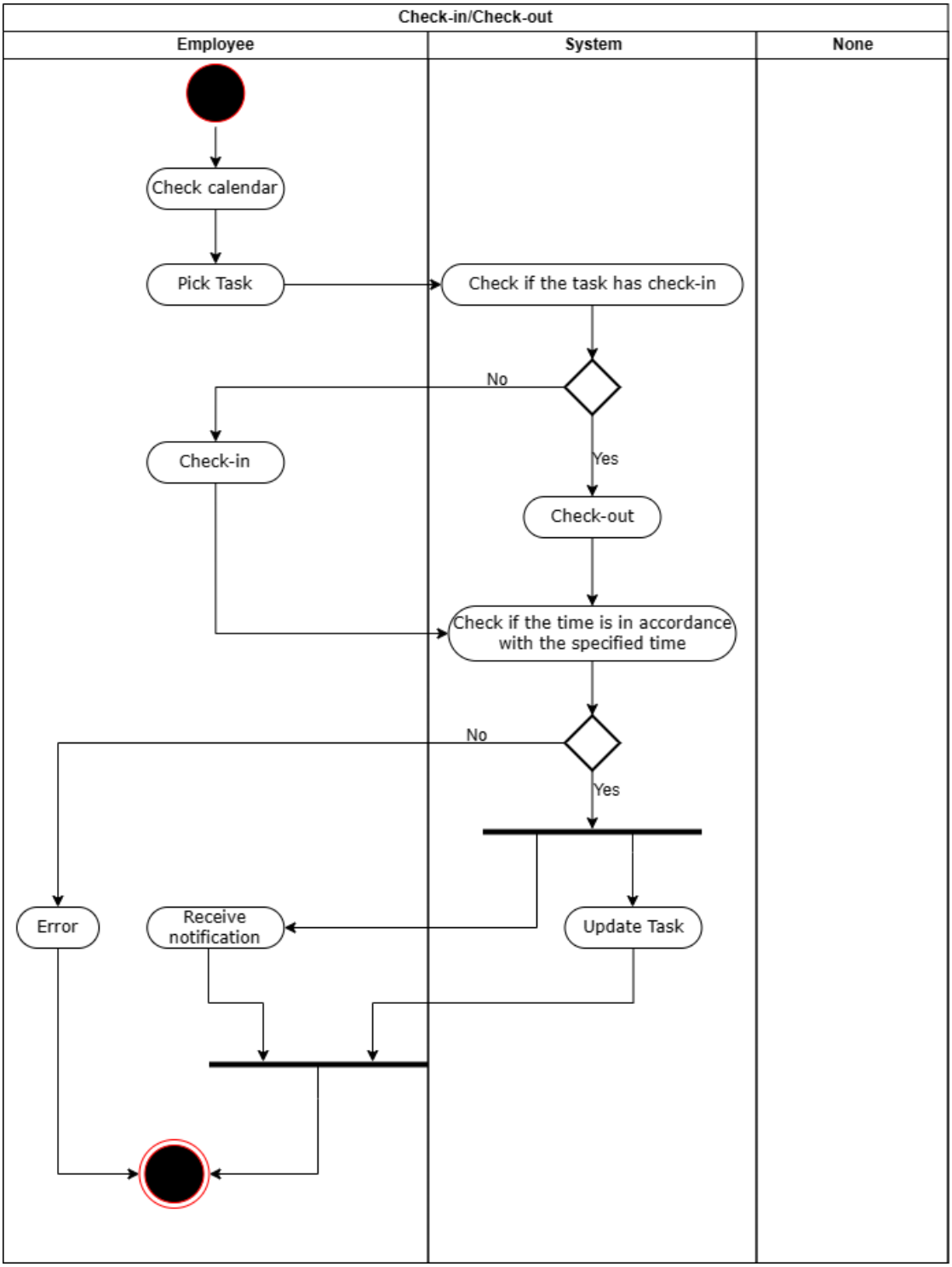
1.1 Activity diagram to capture the business process between systems and the stakeholders in Task Assignment module

Activity diagram



Description

When the Back Officer login to the system, the system will display information about vehicles, collectors, janitors, routes, and MCPs. The Back Officer can choose to create new tasks or modify existing tasks. With creating a new task, the system will check whether the fields entered by Back Officer are valid. If the information is valid, the system will confirm and send notifications to the Collectors/Janitors and update the schedule, otherwise, the Back Officer must re-enter the information. The Back Officer can edit/delete existing tasks on the system, then the system will confirm and send notifications to Collectors/Janitors and update the schedule.


Check-in/Check-out
Employee
System
None
Check calendar
Pick Task
Check if the task has check-in
No
Yes
Check-in
Check-out
Check if the time is in accordance with the specified time
No
Yes
Error
Receive notification
Update Task

Description
Employees can check their calendars, select a task, and the system will check if the task has been checked-in or not. If the system detects that the task has not been checked in, the employee will check it in. If it has already been checked in, the employee will check it out, and the system will check if the time is suitable according to the specified schedule. If it is not suitable, an error notification will be displayed. If it is suitable, the system will update the task's time and send a notification to the employee.

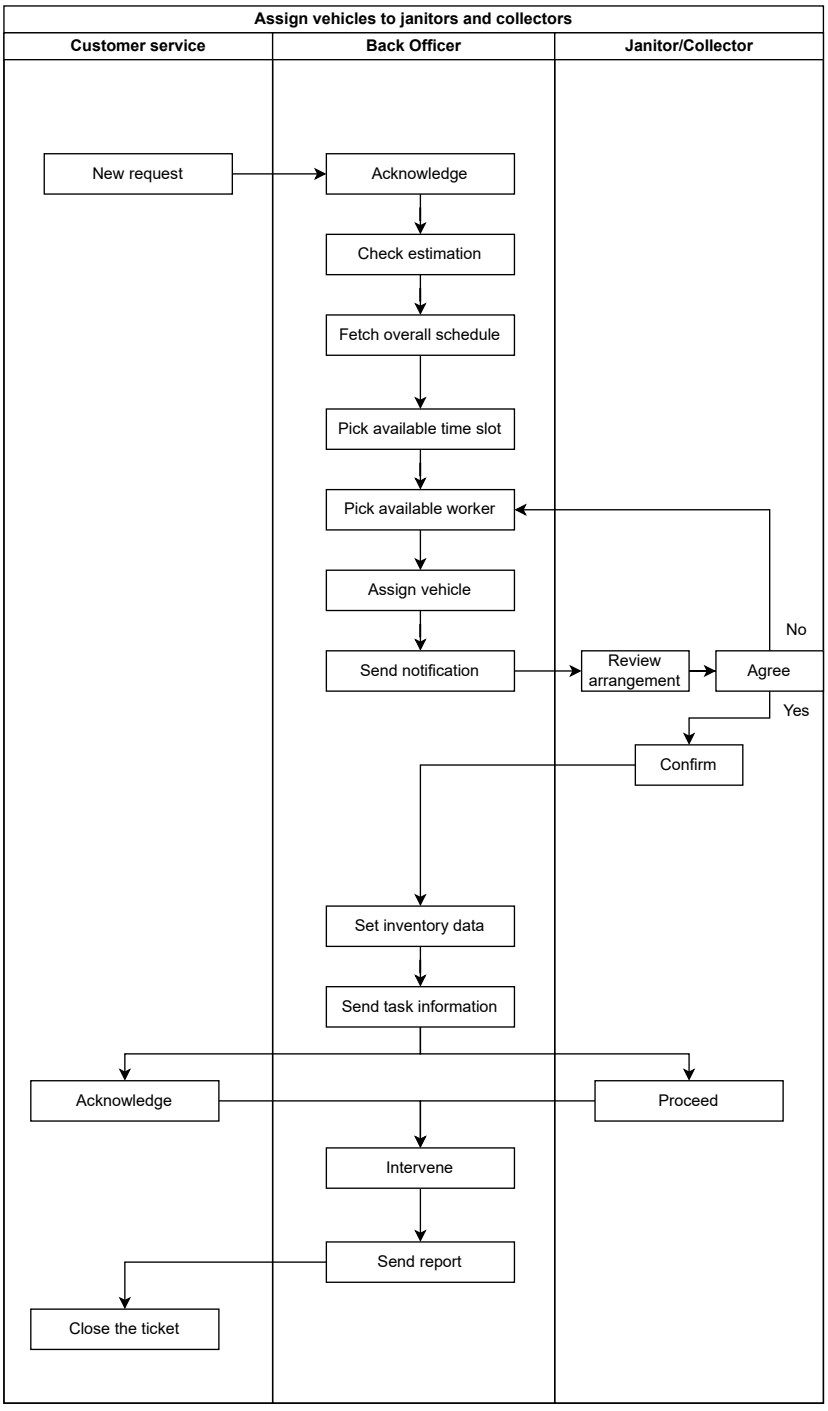
1.2 Swimlane diagram and Sequence diagram

1.2.1 Swimlane diagram for the general process

The stakeholders in the management role will create a new ticket to the UWC2.0 (New request). A ticket must contain these parameters:

- Job description: determine whether janitors, collectors, or both are required.
- Workload: How many janitors/collectors are requested? The estimated weight of waste will be collected.
- Desired date and time

The back office receives a notification when a ticket is created. They change the ticket's status from "New" to "In preparation". The ticket owner also receives notifications when the ticket is modified (Acknowledged). The back office reviews the requirements and clarifies them to determine the final workload and desired date and time (Check estimation). Next, the back office loads the overall calendar in UWC2.0 (Fetches the overall schedule) and selects the required time slot (Picks available time slot). A list of available workers in this time slot is displayed, sorted by total working hours per week in increasing order. The back office can choose the worker manually or the system will assign one automatically (Pick an available worker). Assigning essential vehicles follows a similar process. Once the back office confirms these decisions, the janitors/collectors receive a notification of their scheduled assignment (Send notification). They confirm whether they agree with the schedule or not. If they don't, the back office must assign the task to other workers. Once the workers confirm, the back office updates the usage status of vehicles (Set inventory data). The final task details are sent to the management/customer service team and the workers (Send task information). The workers perform the task and update its status according to their progress. The back office tracks the progress, handles incidents/issues, and checks the status (Intervene). Finally, the back office changes the ticket's status to "Done". The customer service reviews it once more and then closes it.
Swimlane diagram

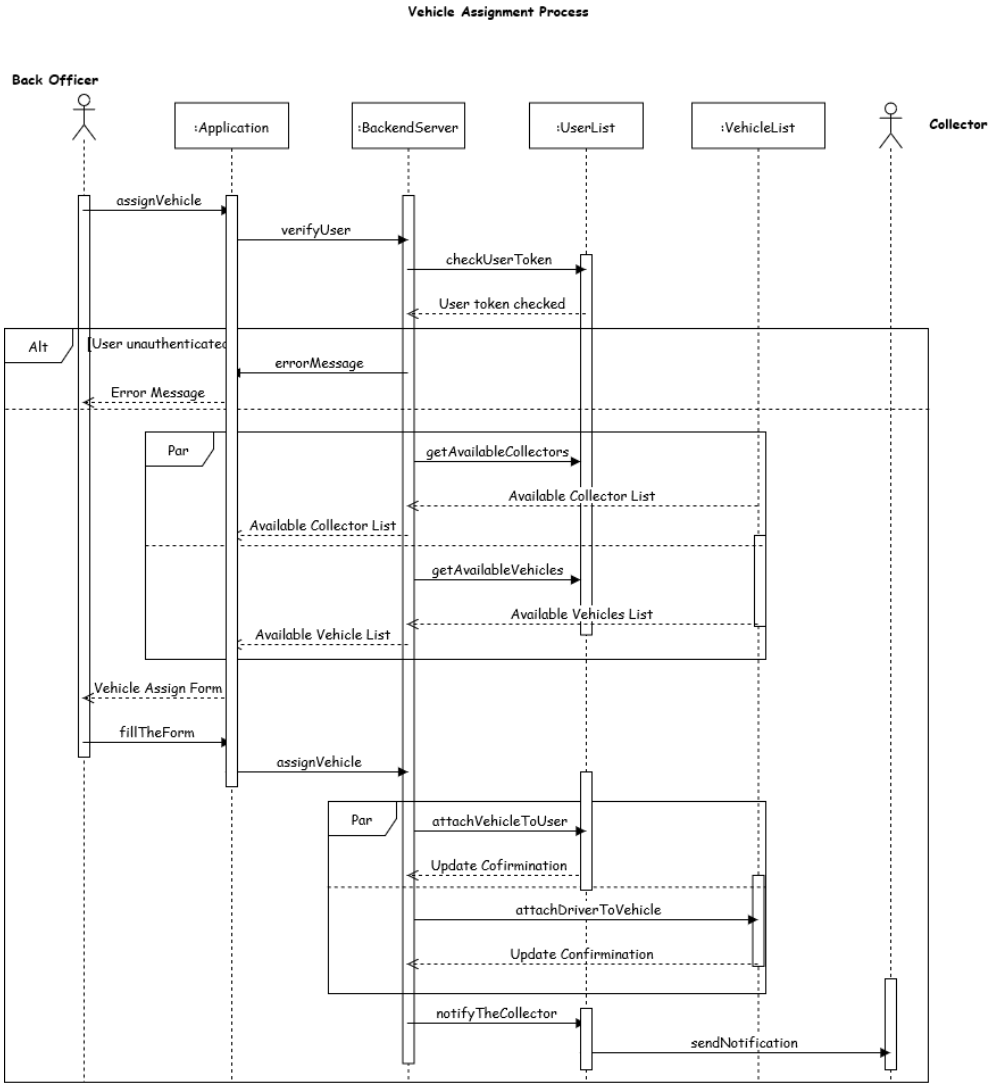

Assign vehicles to janitors and collectors
Customer service
Back Officer
Janitor/Collector
New request
Acknowledge
Check estimation
Fetch overall schedule
Pick available time slot
Pick available worker
Assign vehicle
Send notification
Review arrangement
Agree
No
Yes
Confirm
Set inventory data
Send task information
Acknowledge
Proceed
Intervene
Send report
Close the ticket

1.2.2 Sequence diagram with functions and programming logics

a) Conceptual solution for vehicle assignment task

1. The back officer request the system to assign a vehicle
2. The system checks back officer's authentication by verifying their login token
3. If the token is invalid, the system throws an error message and ends the process, else fetches parallelly the available employees and vehicles back to the user
4. The user chooses suitable employee-vehicle pair and submits it to the system
5. System modified the data accordingly and notify the assigned employee

b) Sequence Diagram



1.3 Class diagram of Task Assignment module

Description

A. Entity class (blue)

1. User
   (a) Attitude
      - ID: string: String of some characters, It's unique.
      - name: string: Full name of this user.
      - age: int: Age of this user.
      - gender: character: Gender of this user, chose among F(female), M(male) and O(other).
      - address: string: String of this user address.
      - emailAddress: string: String of this user email, it could be a regex to determine the right format.
      - phone: string: String of 10 or 11 numbers.
      - role: string: Role of his/ her at that time, It could be leader, training guy, manager team,...
      - userName: string: String of some character, it's unique
      - password: string: String of some character, it includes at least one special character.

   (b) Method
      - logIn(): void: User logins into system.
      - logOut(): void: User log out from system.
      - getID(): string: Get ID of this user.
      - getName(): string: Get full name of this user.
      - chat(): void: Send message to other users.

2. Back officer
   (a) Method
      - viewTask(): void: Move to overView page.
      - viewTool(): void: Move to all tool page.
      - viewEmployee(): void: Move to all employee page.
      - viewDashBoard(): void: Move to dashboard page.

3. IT staff
   (a) Method
      - createAccount(): Create a new account.
      - deleteAccount(): Delete a exist account.

4. Employee (abstract class)
   (a) Method
      - confirmTask(): bool: This method allows the employee to confirm that they have completed a task assigned to them. It returns a boolean value indicating whether the task was successfully confirmed or not.

- CheckIn(): string: This method allows the employee to check in to work, typically by logging their arrival time. It returns a string value indicating that the employee has successfully checked in.
- CheckOut(): string: This method allows the employee to check out of work, typically by logging their departure time. It returns a string value indicating that the employee has successfully checked out.
- requestChange(): void: This method allows the employee to request a change in their assigned tasks or work schedule. It does not return any value, but rather triggers a notification or request for management to review and respond to.
- viewTask(): void: This method allows the employee to view their assigned tasks for the day or week. It does not return any value, but rather displays the tasks on screen or in a report.

5. Collector
   (a) Method
   - viewTaskOfCollector(): void: This method allows the collector to view their assigned tasks, such as date, area, MCP. It does not return any value, but rather displays the tasks on screen or in a report.
   - viewVehicle(): void: This method allows the collector to view information about the vehicle they are using for their work, such as its current status or maintenance needs. It does not return any value, but rather displays the vehicle information on screen or in a report.

6. Janitor
   (a) Method
   - viewTaskOfJanitor(): void: This method allows the Janitor view their assigned tasks, such as date, route. It does not return any value, but rather displays the tasks on screen or in a report.
   - viewTroller(): void: This method allows the Janitor to view information about the troller they are using for their work, such as its current status or maintenance needs. It does not return any value, but rather displays the troller information on screen or in a report.

7. Tool (abstract class)
   (a) Attitude
   - ID: string: This property represents the unique identifier of the tool.
   - name: string: This property represents the name of the tool. It is “troller” or “vehicle”.
   - status: bool: This property represents the current status of the tool, such as whether it is available or in use.
   - glocation: string: This property represents the current location of the tool, such as a warehouse or job site.

   (b) Method
   - getID(): string: This method returns the unique identifier of the tool as a string.
   - getName(): string: This method returns the name of the tool as a string.

- setName(name: string): bool: This method sets the name of the tool to the given string and returns a boolean indicating whether the operation was successful.
- setID(id: string): bool: This method sets the unique identifier of the tool to the given string and returns a boolean indicating whether the operation was successful.

8. Vehicle
   (a) Attitude
   - weight: int: This property represents the weight of the vehicle.
   - capacityOfWaste: int: This property represents the capacity of waste that the vehicle can hold.
   - fuelStatus: int: This property represents the current level of fuel in the vehicle. It is a number from 0 to 10. (with 0 is empty and 10 is full)
   - capacityOfFuel: int: This property represents the total capacity of fuel that the vehicle can hold. It is a number. (lits)

   (b) Method
   - getCapacityOfWaste(): int: This method returns the capacity of waste that the vehicle can hold as an integer value (kg).
   - setCapacityOfWaste(capacity: int): void: This method sets the capacity of waste that the vehicle can hold to the given integer value (kg).
   - getWeight(): int: This method returns the weight of the vehicle as an integer value (kg).
   - setWeight(weight: int): void: This method sets the weight of the vehicle to the given integer value (kg).
   - getCapacityOfFuel(): int: This method returns the total capacity of fuel that the vehicle can hold as an integer value (liter).
   - setCapacityOfFuel(capacity: int): void: This method sets the total capacity of fuel that the vehicle can hold to the given integer value (liter)
   - getFuelStatus(): int: This method returns the current level of fuel in the vehicle as an integer value (liter).
   - setFuelStatus(status: int): void: This method sets the current level of fuel in the vehicle to the given integer value (liter).

9. Troller
   (a) Attitude
   - capacity: int: This property represents the capacity of waste that the troller can hold (kg).

   (b) Method
   - getCapacity(): int: This method returns the capacity of waste that the troller can hold as an integer value (kg).
   - setCapacityOfWaste(capacity: int): void: This method sets the capacity of waste that the troller can hold to the given integer value (kg).

10. Task (interface class)
    (a) Attitude

- ID: string: This property represents the unique identifier of the task.
- backOfficerCreate: BackOfficer: This property represents the back officer who created the task.
- dateCreate: string: This property represents the date when the task was created.
- timeTask: string: This property represents the time when the task is scheduled to be completed.

(b) Method

- getID(): string: This method returns the unique identifier of the task as a string.
- getPersonCreate(): BackOfficer: This method returns the back officer who created the task.
- getDateCreate(): string: This method returns the date when the task was created as a string.
- getTimeTask(): string: This method returns the time when the task is scheduled to be completed as a string.
- setTimeTask(time: string): void: This method sets the time when the task is scheduled to be completed to the given string value.

11. TaskOfJanitor

(a) Attitude

- name: Collector: This property represents the name of the collector assigned to the task.

(b) Method

- assignRoute(): void: This method assigns one or more route to the collector for the task.
- getPersonCreate(): BackOfficer: This method returns the back officer who created the task.
- assignVehicle(): void: This method assigns a vehicle to the collector for the task.
- assignMCPs(): void: This method assigns MPCs to the collector for the task.

12. TaskOfCollector

(a) Attitude

- name: Collector: This property represents the name of the collector assigned to the task.

(b) Method

- assignRoute(): void: This method assigns a route to the collector for the task.
- assignVehicle(): void: This method assigns a vehicle to the collector for the task.

13. MCP

(a) Attitude

- ID: string: This property represents the unique identifier of the MCP.
- address: string: This property represents the address of the MCP.
- status: int: This property represents the current status of the MCP, which is an integer value between 0 and 10 (0: empty and 10: full).

- capacity: float: This property represents the capacity of the MCP (kg).

(b) Method

- updateStatus(): void: This method updates the status of the MCP.
- getCapacity(): float: This method returns the capacity of the MCP as a floating point number.
- getStatus(): int: This method returns the current status of the MCP as an integer value between 0 and 10.
- getAddress(): string: This method returns the address of the MCP as a string (x and y).

14. Route

(a) Attitude

- IDRoute: string: This property represents the unique identifier of the route.
- numberMCP: int: This property represents the number of MCPs on the route.
- location: string: This property represents the location of the route.

(b) Method

- getId(): string: This method returns the unique identifier of the route as a string.
- getNumberMCP(): int: This method returns the number of MCPs on the route as an integer.
- setNumberMCP(): void: This method sets the number of MCPs on the route.
- setLocation(): void: This method sets the location of the route.
- getLocation(): string: This method returns the location of the route as a string.

15. Area

(a) Attitude

- IDArea: string: This property represents the unique identifier of the area.
- NumberJanitor: int: This property represents the number of janitors assigned to the area.
- location: string: This property represents the location of the area.

(b) Method

- getId(): string: This method returns the unique identifier of the area as a string.
- getNumberJanitor(): int: This method returns the number of janitors assigned to the area as an integer.
- setNumberJanitor(): void: This method sets the number of janitors assigned to the area.
- setLocation(): void: This method sets the location of the area.
- getLocation(): string: This method returns the location of the area as a string.

B. View class (yellow)

1. OverView

(a) Method

- viewAllTask(): void: This method displays an overview of all the tasks in the system. It can be implemented to show a list of all the tasks, including their IDs, creation dates, assigned personnel, and statuses
- ilterTask(): void: This method allows the user to filter the displayed tasks based on certain criteria. It can be implemented to provide filtering options, such as filtering by task status, creation date, assigned personnel, or location, among others. The method should display the filtered list of tasks after the user selects their desired filter criteria.

2. MapView
   (a) Method
   - viewAllMap(): void: This method displays the map view of all areas, routes, and MCPs. It can be implemented to show a graphical representation of the system's layout and components, highlighting the different areas and their corresponding routes, as well as the location of each MCP on the map.

3. Dashboard
   (a) Attitude
   - totalTask: int: This property represents the total number of tasks in the system.
   - janitorTask: int: This property represents the number of tasks assigned to janitors.
   - collectorTask: int: This property represents the number of tasks assigned to collectors.

   (b) Method
   - viewTotalTask(): void: This method displays the total number of tasks in the system.
   - viewCollectorTask(): void: This method displays the number of tasks assigned to collectors.
   - viewJanitorTask(): void: This method displays the number of tasks assigned to janitors.
   - viewTaskMonthly: void: This method displays a monthly overview of tasks, such as the total number of tasks completed, the number of tasks assigned to each collector and janitor, and any other relevant metrics. The method can be implemented to display a graphical representation of the data, such as a chart or a graph.

C. Controller class (purple)

1. MapController
   (a) Method
   - viewMcps(): void: This method displays a map view of all MCPs in the system. It can be implemented to show the location of each MCP on the map and any relevant information about each MCP, such as its ID, address, and current status.
   - searchMCP(): void: This method allows the user to search for a specific MCP in the system. It can be implemented to provide a search bar or other input field where the user can enter the MCP ID or other relevant search criteria. The method should display the MCP's location and any other relevant information after the user submits their search query.

- searchVehicle(): void: This method allows the user to search for a specific vehicle in the system. It can be implemented to provide a search bar or other input field where the user can enter the vehicle ID or other relevant search criteria. The method should display the vehicle's location and any other relevant information after the user submits their search query.
- viewRoute(): void: This method displays a map view of all routes in the system. It can be implemented to show the location of each route on the map and any relevant information about each route, such as its ID, number of MCPs, and assigned janitors and collectors.
- createShortestRoute(): void: This method calculates and creates the shortest route for a specific area or task. It can be implemented to take into account various factors, such as the distance between MCPs, traffic, and the location of tasks, among others. The method should display the shortest route on the map after it has been created.

2. TaskController

(a) Method

- addTask(): void: This method allows the user to add a new task to the system. It can be implemented to provide a form or input fields where the user can enter the relevant details about the task, such as its ID, assigned janitor or collector, date and time, location, and other details. The method should validate the user's input and add the new task to the system.
- updateTask(): void: This method allows the user to update an existing task in the system. It can be implemented to provide a form or input fields where the user can edit the relevant details about the task, such as its assigned janitor or collector, date and time, location, and other details. The method should validate the user's input and update the task in the system.
- deleteTask(): void: This method allows the user to delete an existing task from the system. It can be implemented to display a list of all tasks in the system and allow the user to select the task they want to delete (it is a button like a garbage icon at each line of overview task). The method should prompt the user to confirm the deletion and then remove the task from the system.

D. Database class (red)

1. TaskDb

(a) Attitude

- number: int: This property stores the number of tasks currently stored in the task database.
- listTask: Task[]: This property stores the actual task objects that are stored in the database. It can be implemented as an array or list of task objects.

(b) Method

- getNumber(): int: This method returns the number of tasks currently stored in the task database.
- getTask(): Task: This method allows the user to retrieve a specific task from the database. It can be implemented to take an input parameter such as the task ID or some other identifier, and then search the database for the task with that ID. If the task is found, the method should return the task object.

- getAllTask(): Task[]: This method returns an array of all tasks currently stored in the task database. It can be implemented to simply return the entire list of tasks stored in the database.

2. MapDb

   (a) Attitude

   - numberRoute: int: This property stores the number of routes currently stored in the map database.
   - numberArea: int: This property stores the number of areas currently stored in the map database.
   - numberMCP: int: This property stores the number of MCPs currently stored in the map database.
   - listMCP: MCP[]: This property stores the actual MCP objects that are stored in the database. It can be implemented as an array or list of MCP objects.
   - listRoute: Route[]: This property stores the actual Route objects that are stored in the database. It can be implemented as an array or list of Route objects.
   - listArea: Area[]: This property stores the actual Area objects that are stored in the database. It can be implemented as an array or list of Area objects.

   (b) Method

   - getNumberRoute(): int: This method returns the number of routes currently stored in the map database.
   - getNumberArea(): int: This method returns the number of areas currently stored in the map database.
   - getNumberMCP(): int: This method returns the number of MCPs currently stored in the map database.
   - getAllMCP(): MCP[]: This method returns an array of all MCPs currently stored in the map database. It can be implemented to simply return the entire list of MCP objects stored in the database.
   - getAllMap(): void: This method allows the user to view the entire map. It can be implemented to display a visual representation of the map, including all MCPs, areas, and routes.

3. ToolDb

   (a) Attitude

   - number: an integer representing the total number of tools in the database.
   - listTruck: an array of Truck objects representing all the trucks in the database.
   - listTroller: an array of Troller objects representing all the trollies in the database.

   (b) Method

   - getNumber(): a method that returns the total number of tools in the database.
   - getAllTruck(): a method that returns an array of all the Truck objects in the database.
   - getAllTroller(): a method that returns an array of all the Troller objects in the database.
   - getTool(): a method that retrieves a specific tool from the database.

- getAllTool(): a method that retrieves all the tools in the database.

4. EmployeeDb

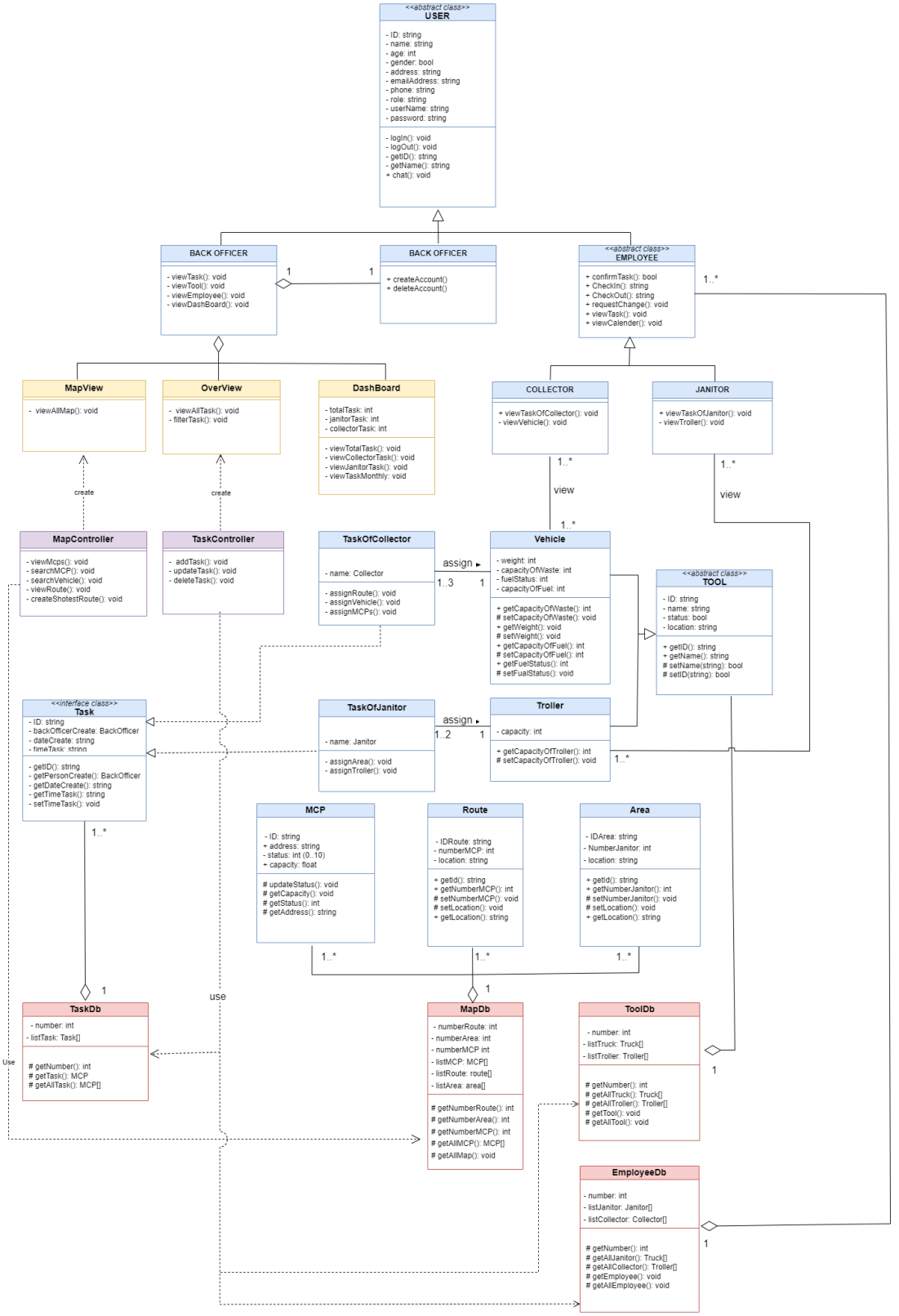
   (a) Attitude

   - number: int: an integer that represents the total number of employees in the database.
   - listJanitor: Janitor[]: a list of Janitor objects that represent the janitor employees in the database.
   - listCollector: Collector[]: a list of Collector objects that represent the collector employees in the database.

   (b) Method

   - getNumber(): int: a method that returns the total number of employees in the database.
   - getAllJanitor(): Truck[]: a method that returns a list of all janitor employees in the database.
   - getAllCollector(): Troller[]: a method that returns a list of all collector employees in the database.
   - getEmployee(): void: a method that gets an employee from the database.
   - getAllEmployee(): void: a method that gets all employees from the database.

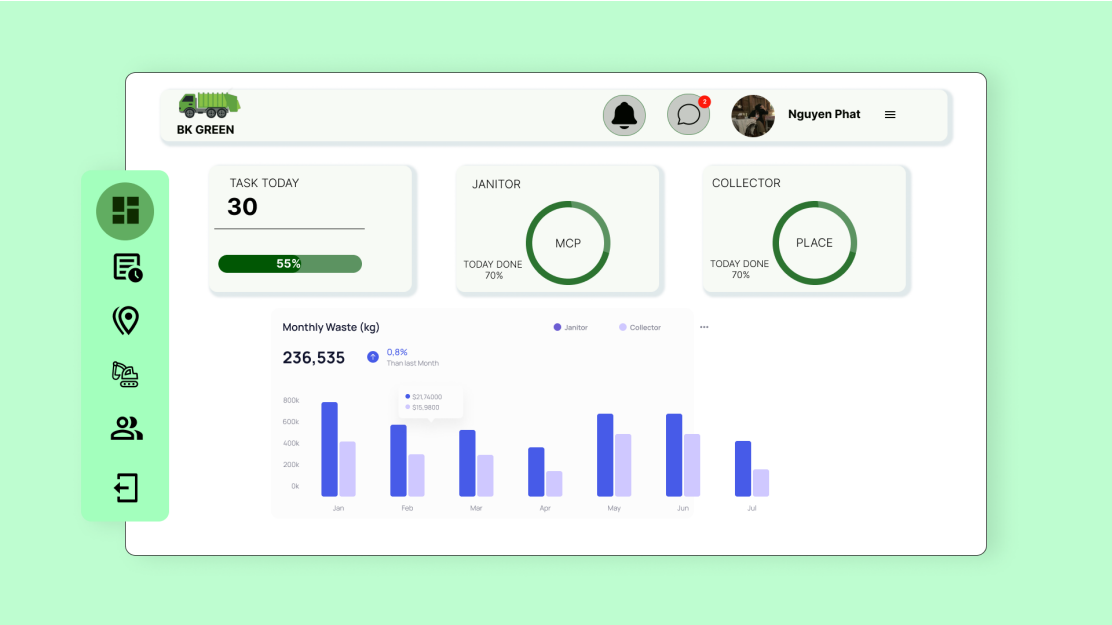
Class diagram


<<abstract class>>
USER
- ID: string
- name: string
- age: int
- gender: bool
- address: string
- emailAddress: string
- phone: string
- role: string
- userName: string
- password: string
- logIn(): void
- logOut(): void
- getID(): string
- getName(): string
+ chat(): void
BACK OFFICER
- viewTask(): void
- viewTool(): void
- viewEmployee(): void
- viewDashBoard(): void
BACK OFFICER
+ createAccount()
+ deleteAccount()
<<abstract class>>
EMPLOYEE
+ confirmTask(): bool
+ CheckIn(): string
+ CheckOut(): string
+ requestChange(): void
+ viewTask(): void
+ viewCalender(): void
1
1
1..*
MapView
- viewAllMap(): void
OverView
- viewAllTask(): void
- filterTask(): void
DashBoard
- totalTask: int
- janitorTask: int
- collectorTask: int
- viewTotalTask(): void
- viewCollectorTask(): void
- viewJanitorTask(): void
- viewTaskMonthly: void
COLLECTOR
+ viewTaskOfCollector(): void
- viewVehicle(): void
JANITOR
+ viewTaskOfJanitor(): void
- viewTroller(): void
create
create
1..*
view
1..*
view
MapController
- viewMcps(): void
- searchMCP(): void
- searchVehicle(): void
- viewRoute(): void
- createShotestRoute(): void
TaskController
- addTask(): void
- updateTask(): void
- deleteTask(): void
TaskOfCollector
- name: Collector
- assignRoute(): void
- assignVehicle(): void
- assignMCPs(): void
assign
1..3
1
Vehicle
- weight: int
- capacityOfWaste: int
- fuelStatus: int
- capacityOfFuel: int
+ getCapacityOfWaste(): int
# setCapacityOfWaste(): void
+ getWeight(): void
# setWeight(): void
+ getCapacityOfFuel(): int
# setCapacityOfFuel(): int
+ getFuelStatus(): int
# setFualStatus(): void
1..*
<<abstract class>>
TOOL
- ID: string
- name: string
- status: bool
- location: string
+ getID(): string
+ getName(): string
# setName(string): bool
# setID(string): bool
<<interface class>>
Task
- ID: string
- backOfficerCreate: BackOfficer
- dateCreate: string
- timeTask: string
- getID(): string
- getPersonCreate(): BackOfficer
- getDateCreate(): string
- getTimeTask(): string
- setTimeTask(): void
TaskOfJanitor
- name: Janitor
- assignArea(): void
- assignTroller(): void
assign
1..2
1
Troller
- capacity: int
+ getCapacityOfTroller(): int
# setCapacityOfTroller(): void
1..*
1..*
MCP
- ID: string
+ address: string
- status: int (0..10)
+ capacity: float
# updateStatus(): void
# getCapacity(): void
# getStatus(): int
# getAddress(): string
Route
- IDRoute: string
- numberMCP: int
- location: string
+ getId(): string
+ getNumberMCP(): int
# setNumberMCP(): void
# setLocation(): void
+ getLocation(): string
Area
- IDArea: string
- NumberJanitor: int
- location: string
+ getId(): string
+ getNumberJanitor(): int
# setNumberJanitor(): void
# setLocation(): void
+ getLocation(): string
1..*
1..*
1..*
1
1
use
TaskDb
- number: int
- listTask: Task[]
getNumber(): int
getTask(): MCP
getAllTask(): MCP[]
MapDb
- numberRoute: int
- numberArea: int
- numberMCP int
- listMCP: MCP[]
- listRoute: route[]
- listArea: area[]
getNumberRoute(): int
getNumberArea(): int
getNumberMCP(): int
getAllMCP(): MCP[]
getAllMap(): void
ToolDb
- number: int
- listTruck: Truck[]
- listTroller: Troller[]
getNumber(): int
getAllTruck(): Truck[]
getAllTroller(): Troller[]
getTool(): void
getAllTool(): void
1
EmployeeDb
- number: int
- listJanitor: Janitor[]
- listCollector: Collector[]
getNumber(): int
getAllJanitor(): Truck[]
getAllCollector(): Troller[]
getEmployee(): void
getAllEmployee(): void
1
Use

1.4 User Interface

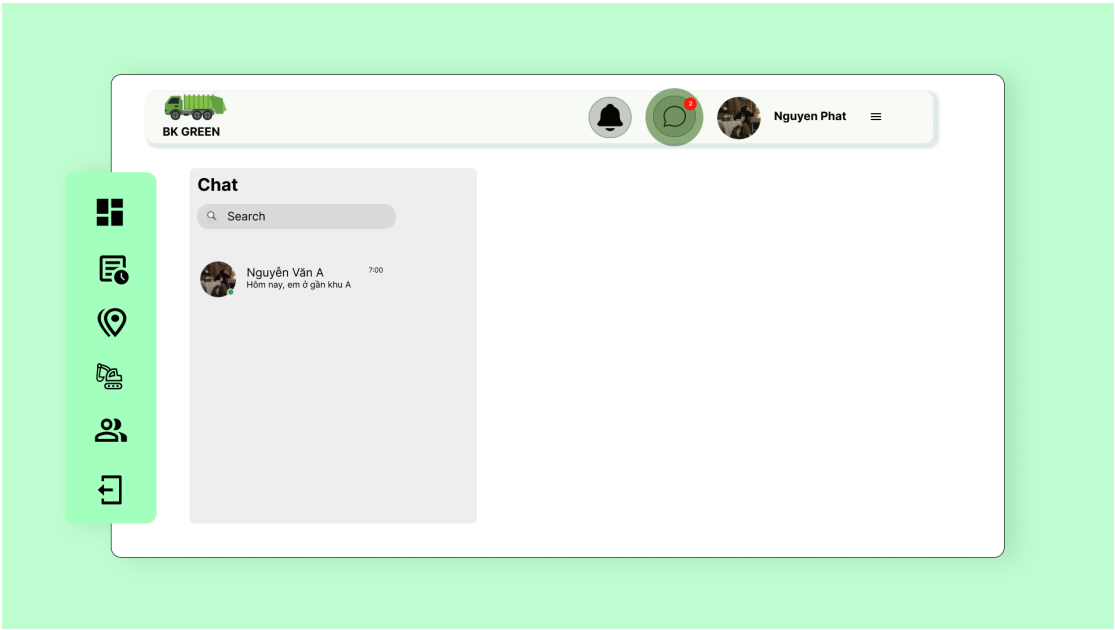
1.4.1 Home_page

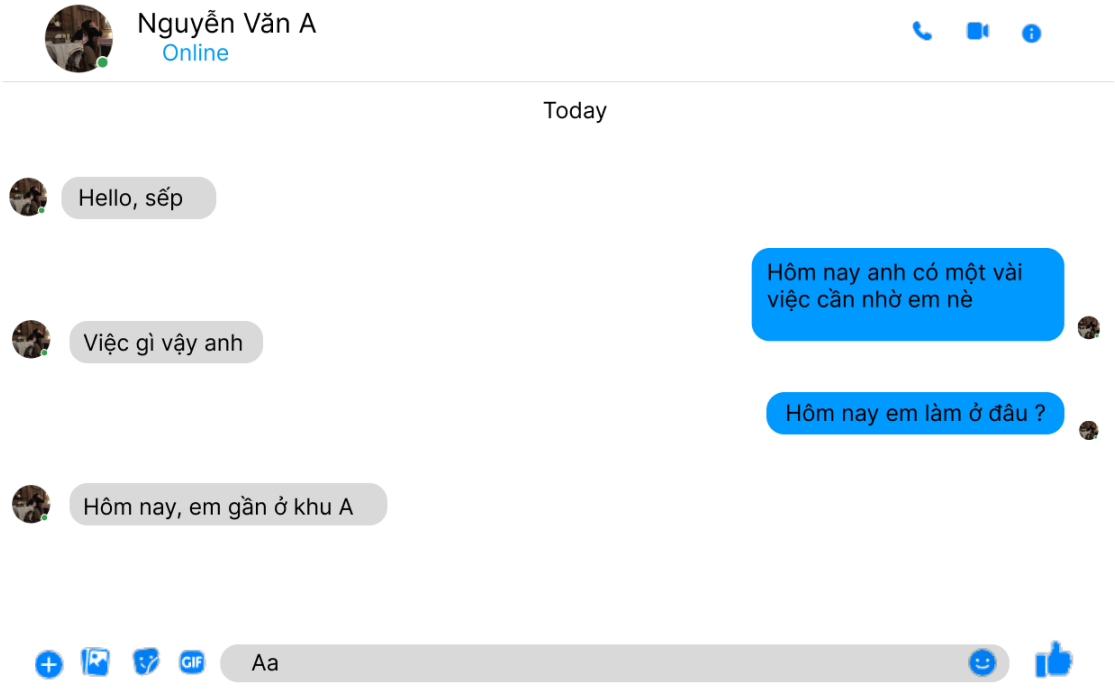
After logging in as a manager, the user will be taken to the homepage for managers. At the top of the page, there is the logo, notifications, messages, and user information. The navigation bar includes the following buttons: home, task, map, tools, worker, and log out. Clicking on any of these buttons will take the user to the corresponding page. The home page is the default page for managers and includes basic information about the number of tasks, worker progress, and the amount of waste generated in a day.



1.4.2 Page_Message

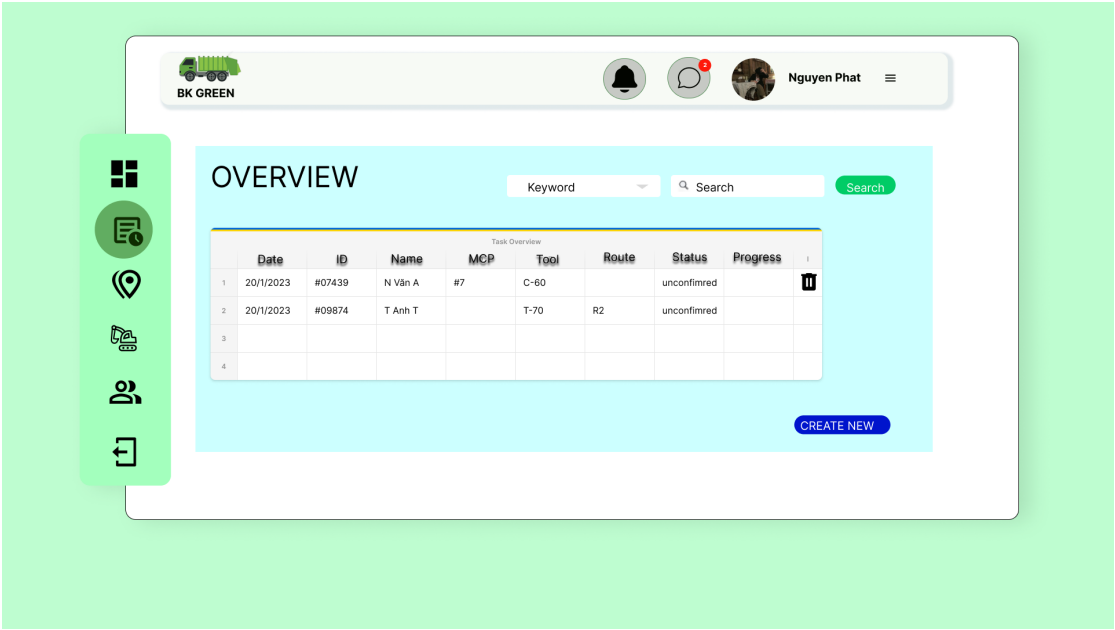
When a manager wants to send a message to a worker, they click the "message" button to go to the message page. The UI of this page includes information about the people who have messaged, and when they want to message someone, they simply click on the message to that person. The UI will display all the messages that the user has sent to the other person and vice versa. Additionally, if the user wants to search for someone, they can simply enter the information into the "search" bar and find the person they want to message.


BK GREEN
Nguyen Phat
Chat
Search
Nguyễn Văn A
Hôm nay, em ở gần khu A
7:00


Nguyễn Văn A
Online
Today
Hello, sếp
Hôm nay anh có một vài việc cần nhờ em nè
Việc gì vậy anh
Hôm nay em làm ở đâu ?
Hôm nay, em gần ở khu A
Aa

1.4.3 Page_Task

When a manager wants to see the assigned tasks, they simply click the "task" button on the navigation bar, and the UI will display detailed information about the assigned tasks in a table format with fields such as "Date, ID, Name, MCP, Tool, Route, Status, Progress." Additionally, the user can quickly search for information by selecting a keyword in the "Keyword" field and searching for the selected field in the "Search" bar next to it. At the bottom of the UI page, there is a "Create New" button for users to create a new task.



On the "New Task" page, the user will see three tables on the left for workers, tools, and routes that have not been assigned work. The user can select the worker, tool, and route they want by clicking on the data in each table. What is selected will appear in the table on the right according to the corresponding field. If the user wants to delete information that is not appropriate, they simply click on the data they want to delete. At this point, a confirmation UI will appear with "Yes" to confirm and "No" to cancel. In addition, on the left side of the worker table, there is a "Janitor" button (this is the default for the person at the top of the table being a janitor), and if the user wants to switch to a collector, they simply click here, and a UI with two choices "Janitor" and "Collector" will appear.

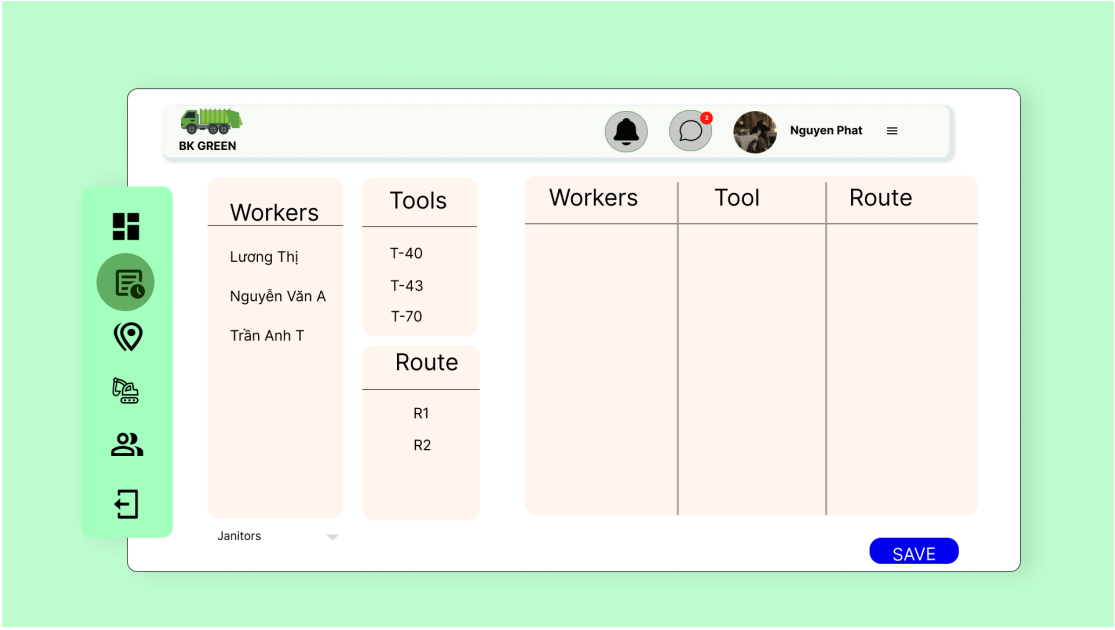
Date

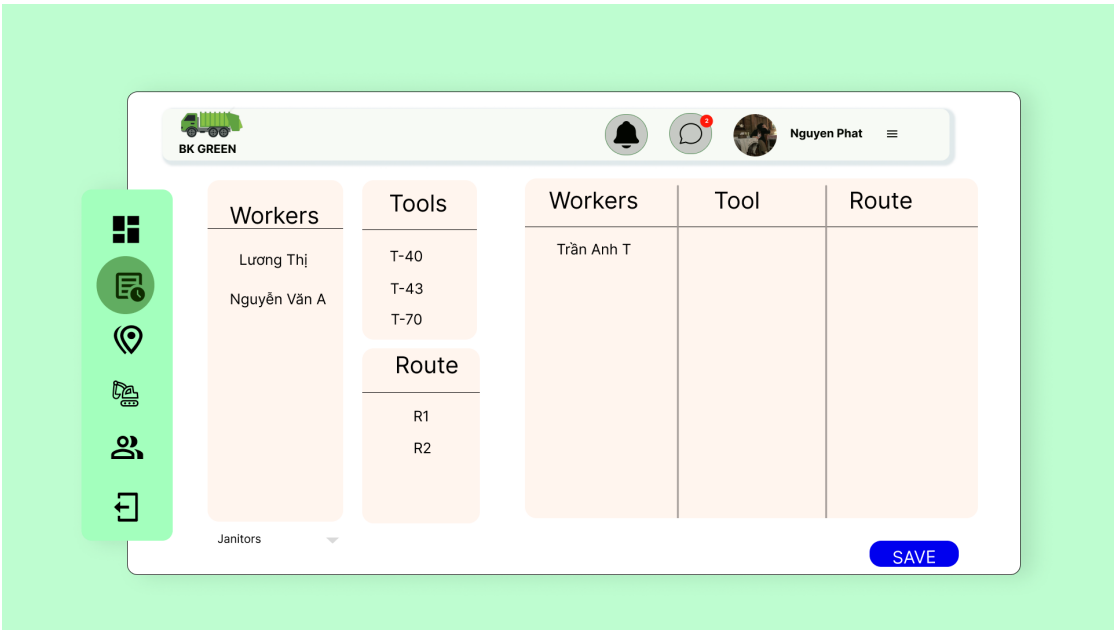
ID

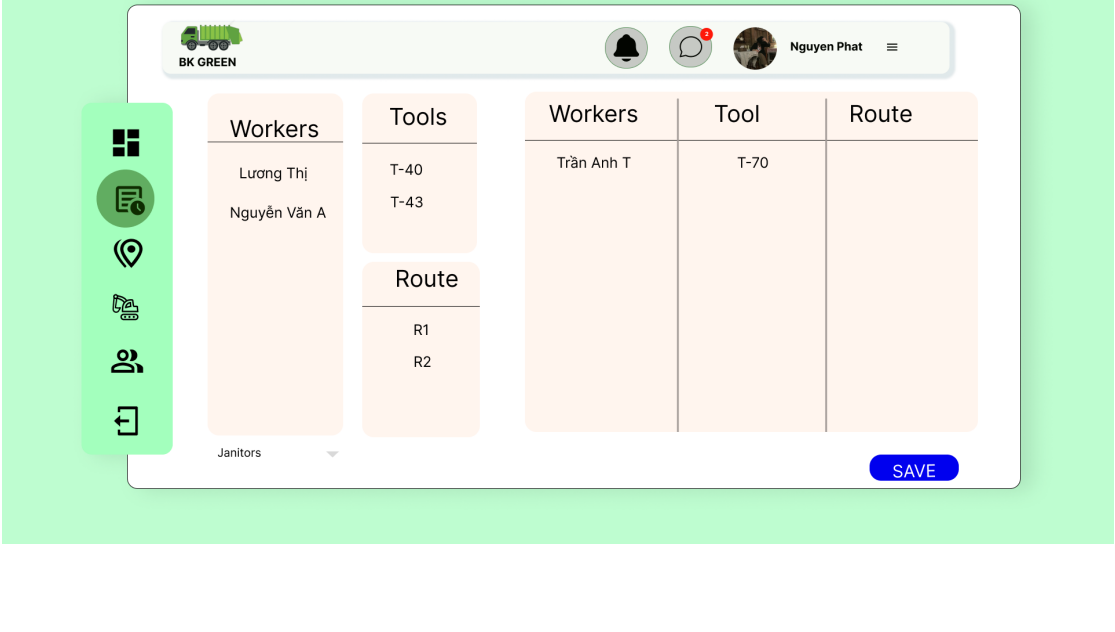
Name

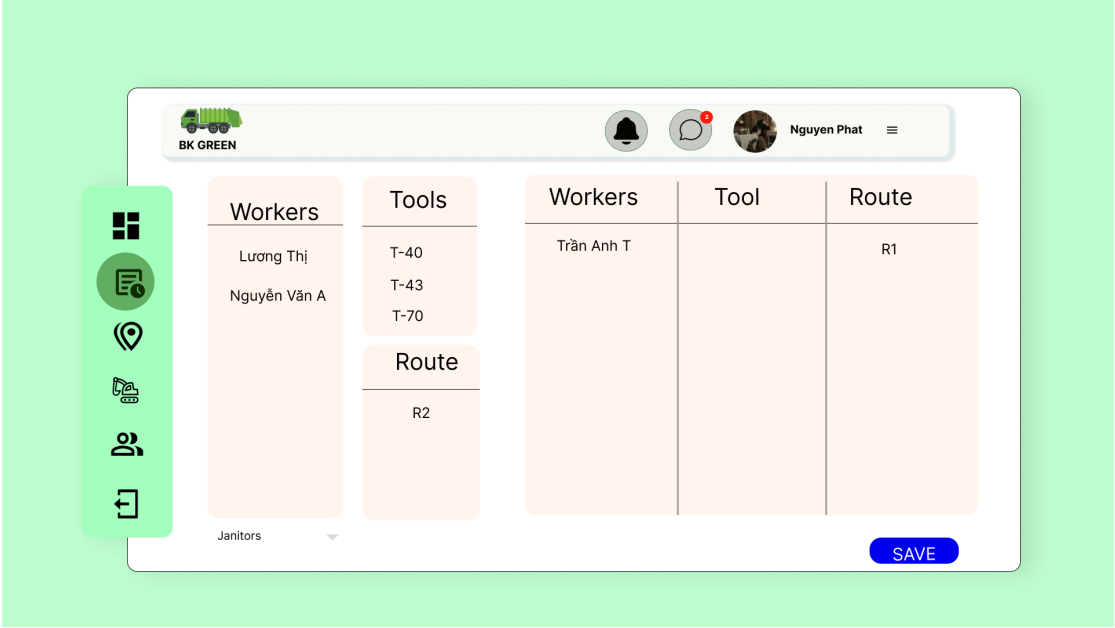
Tool

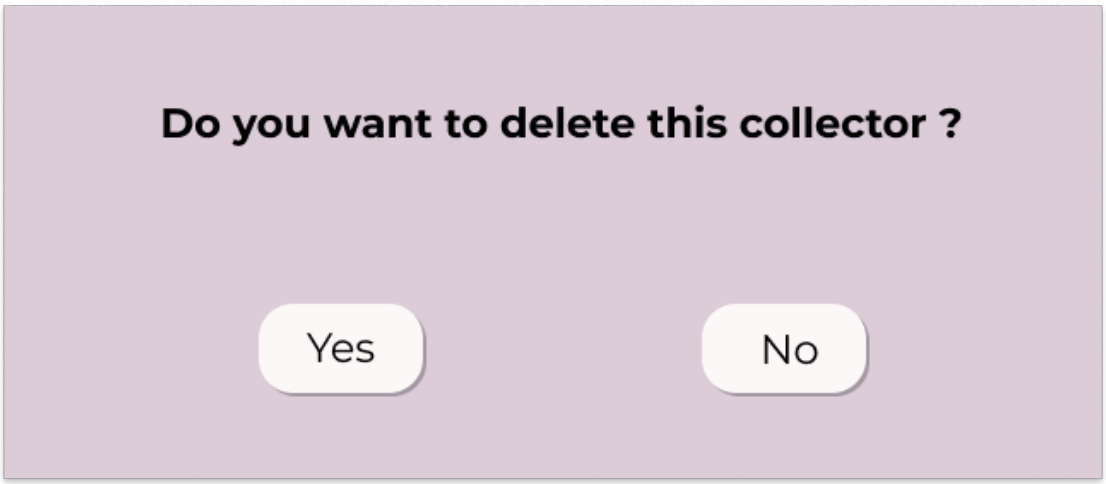
Route

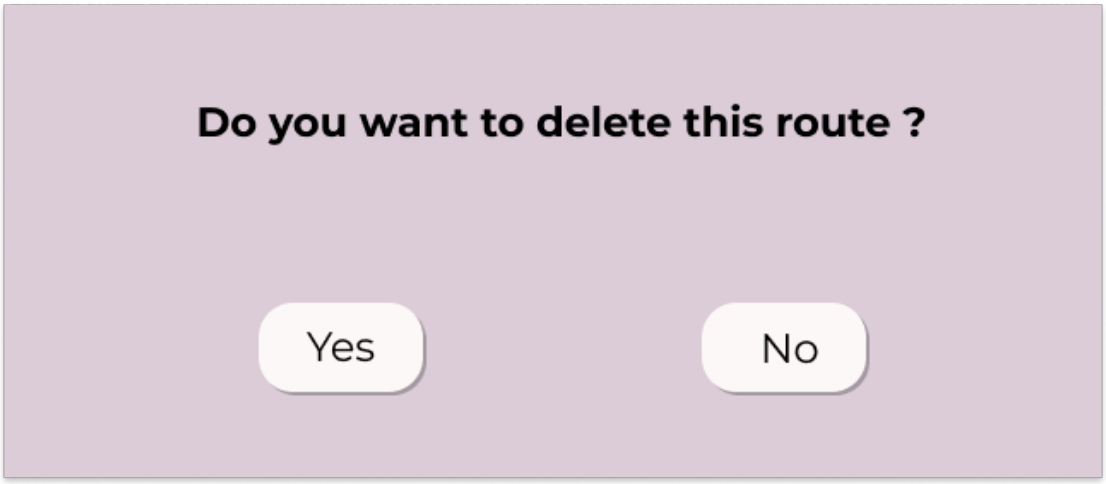


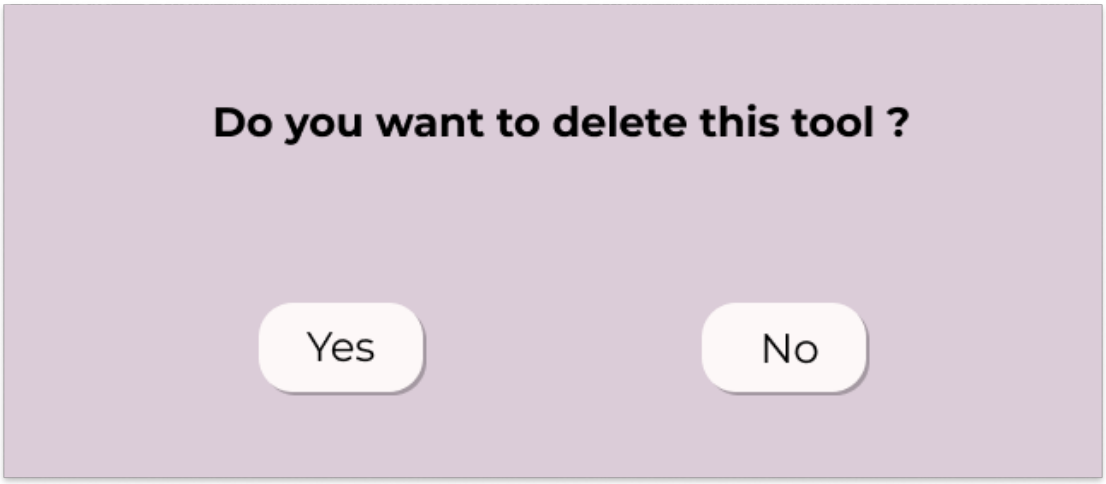

BK GREEN
Nguyen Phat
Workers
Lương Thị
Nguyễn Văn A
Tools
T-40
T-43
T-70
Route
R1
R2
Workers
Tool
Route
Trần Anh T
Janitors
SAVE

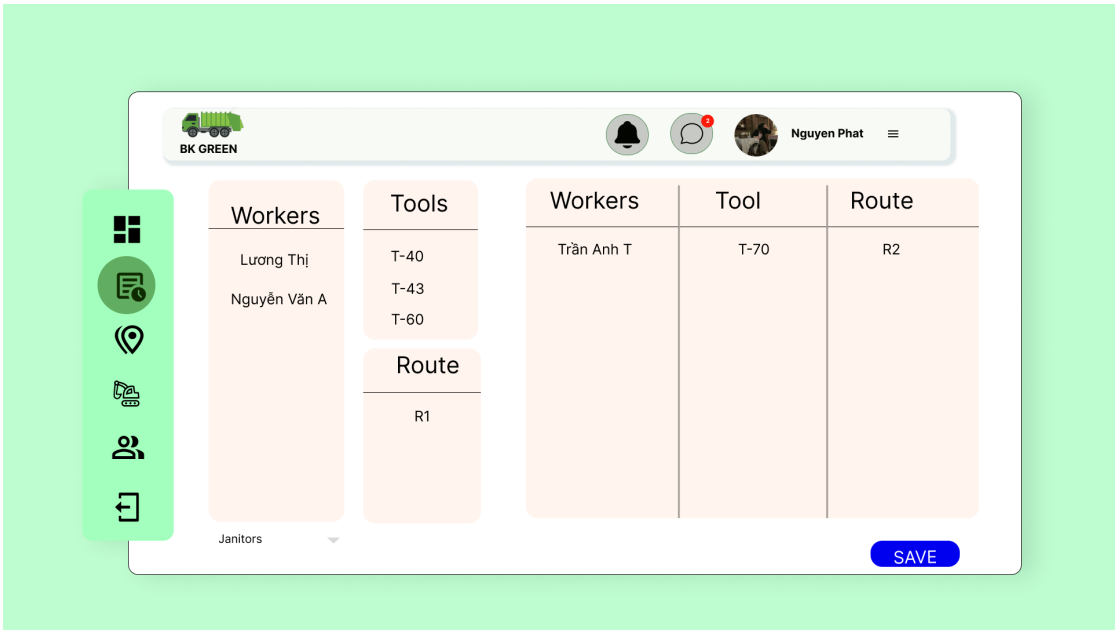

BK GREEN
Nguyen Phat
Workers
Lương Thị
Nguyễn Văn A
Tools
T-40
T-43
Route
R1
R2
Workers
Tool
Route
Trần Anh T
T-70
Janitors
SAVE

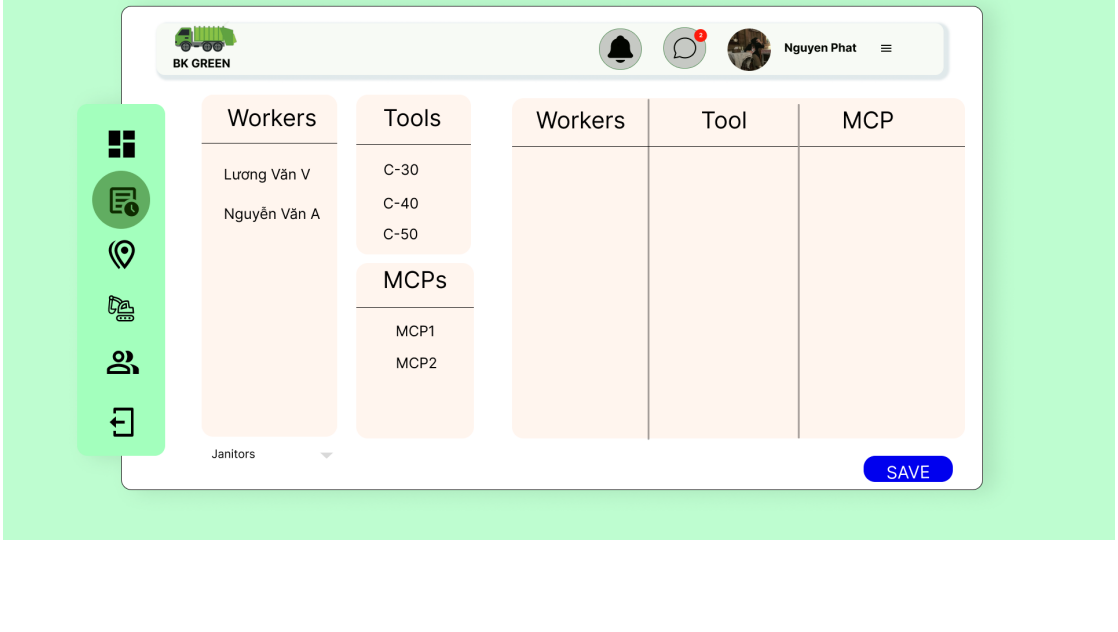

BK GREEN
Nguyen Phat
Workers
Lương Thị
Nguyễn Văn A
Tools
T-40
T-43
T-70
Route
R2
Workers
Tool
Route
Trần Anh T
R1
Janitors
SAVE

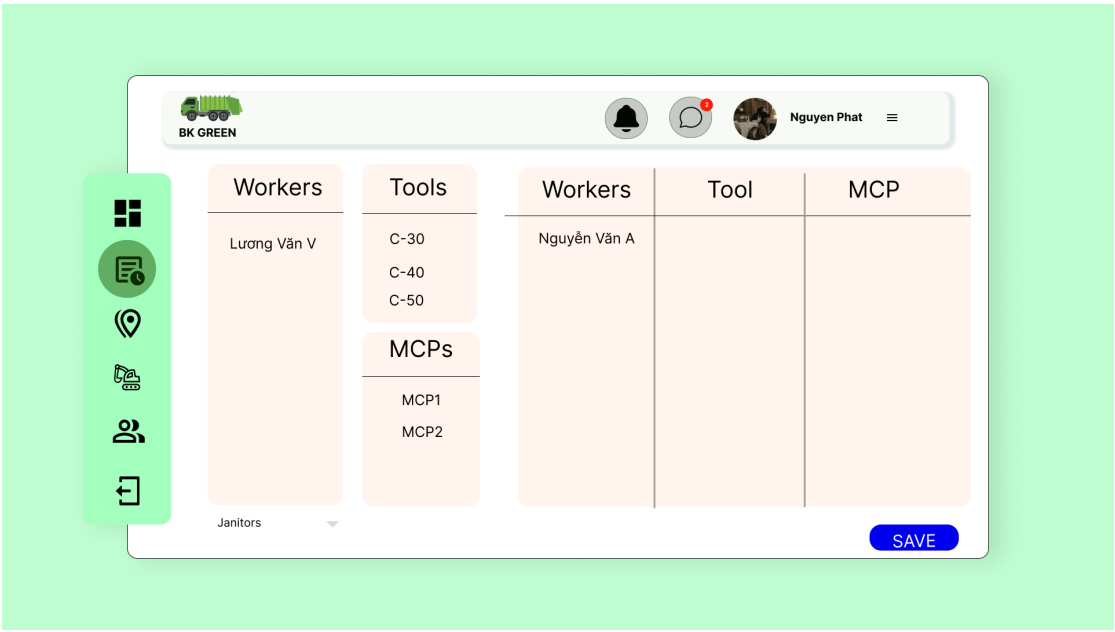

Do you want to delete this collector ?
Yes
No

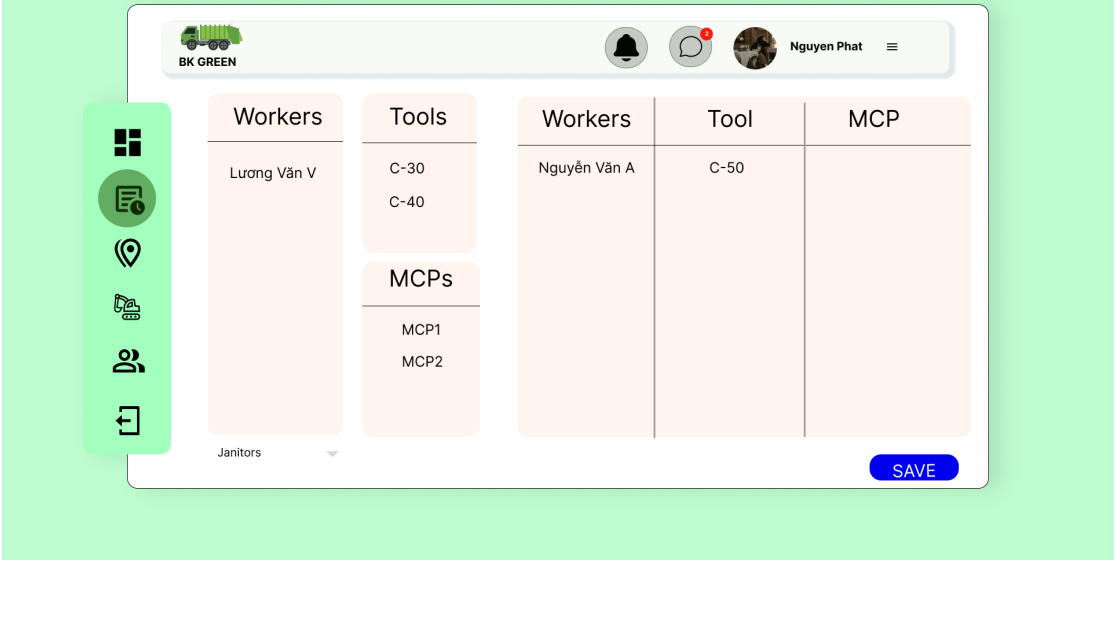

Do you want to delete this route ?
Yes
No

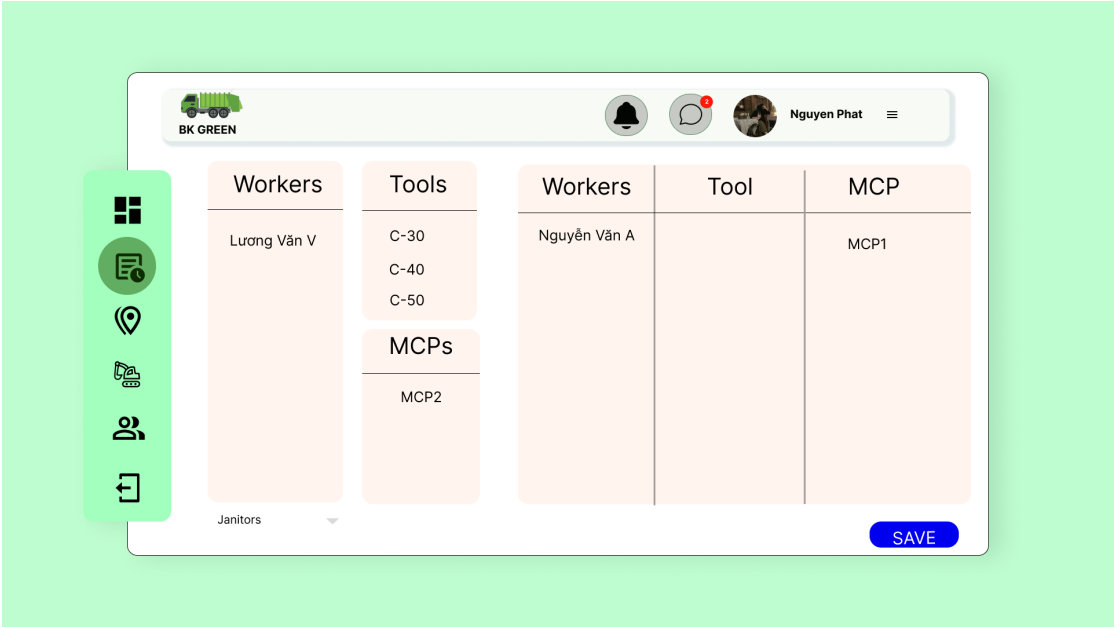

Do you want to delete this tool ?
Yes
No

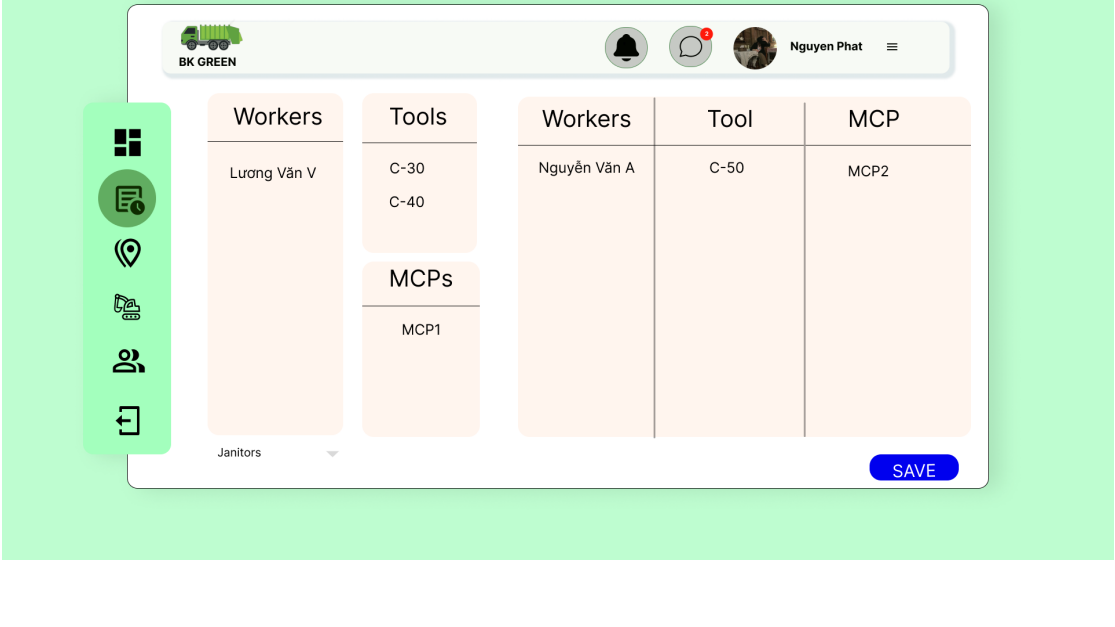

BK GREEN
Nguyen Phat
Workers
Lương Thị
Nguyễn Văn A
Tools
T-40
T-43
T-60
Route
R1
Workers
Tool
Route
Trần Anh T
T-70
R2
Janitors
SAVE


BK GREEN
Nguyen Phat
Workers
Lương Văn V
Nguyễn Văn A
Tools
C-30
C-40
C-50
MCPs
MCP1
MCP2
Workers
Tool
MCP
Janitors
SAVE


BK GREEN
Nguyen Phat
Workers
Lương Văn V
Tools
C-30
C-40
C-50
MCPs
MCP1
MCP2
Workers
Tool
MCP
Nguyễn Văn A
Janitors
SAVE

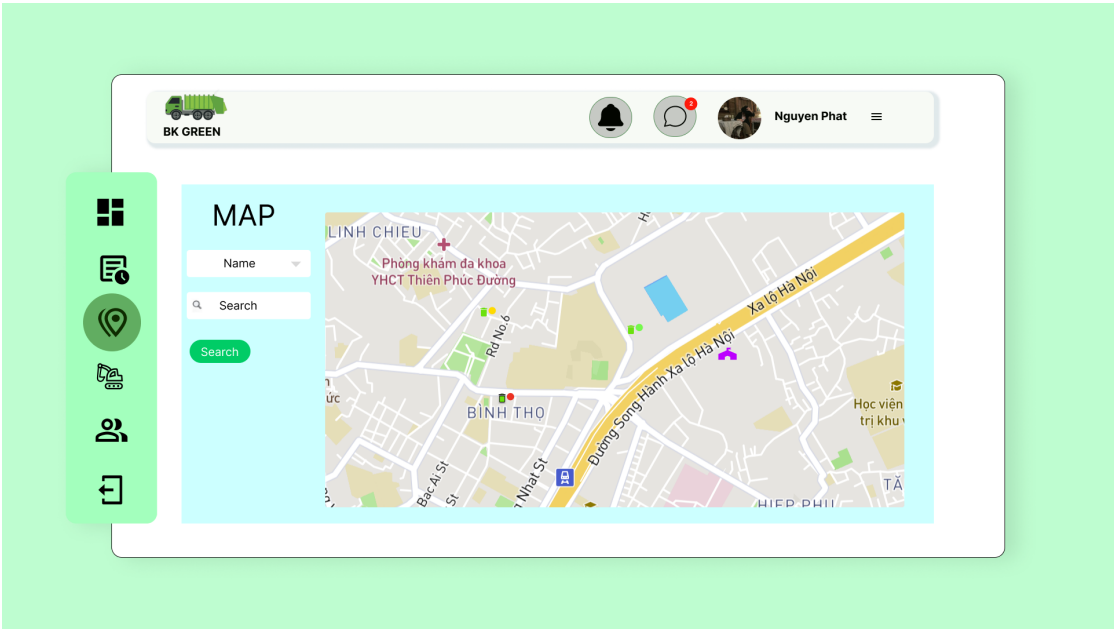

BK GREEN
Nguyen Phat
Workers
Lương Văn V
Tools
C-30
C-40
MCPs
MCP1
MCP2
Workers
Tool
MCP
Nguyễn Văn A
C-50
Janitors
SAVE

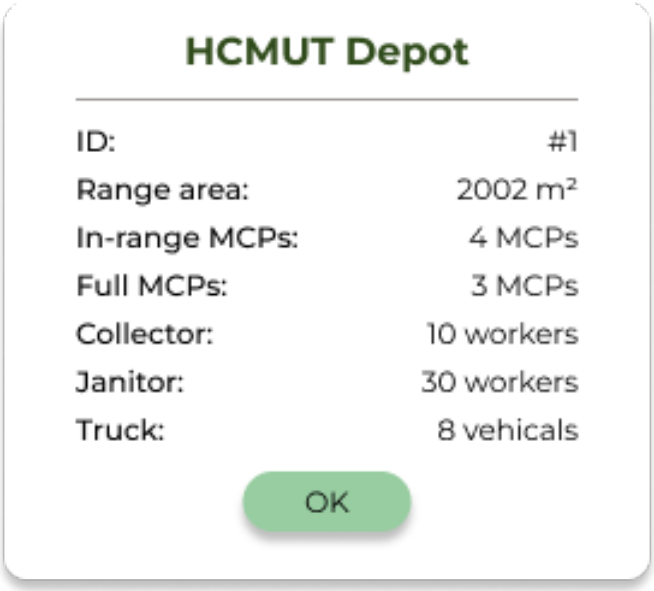

BK GREEN
Nguyen Phat
Workers
Lương Văn V
Tools
C-30
C-40
C-50
MCPs
MCP2
Workers
Tool
MCP
Nguyễn Văn A
MCP1
Janitors
SAVE

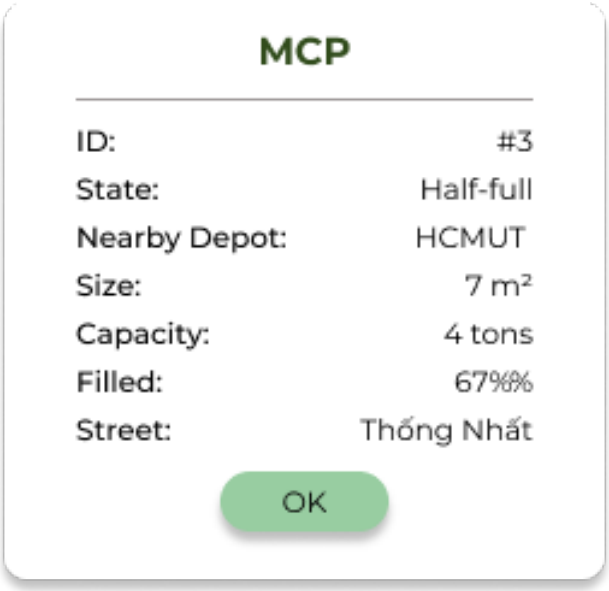

BK GREEN
Nguyen Phat
Workers
Lương Văn V
Tools
C-30
C-40
MCPs
MCP1
Workers
Tool
MCP
Nguyễn Văn A
C-50
MCP2
Janitors
SAVE

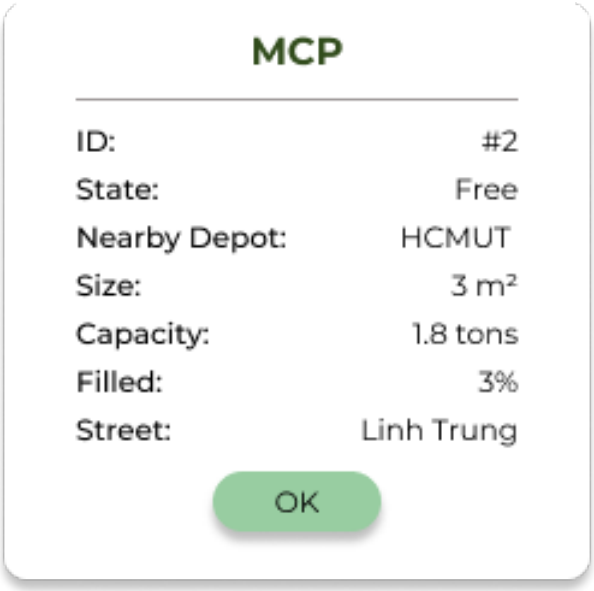
1.4.4 Page_Map

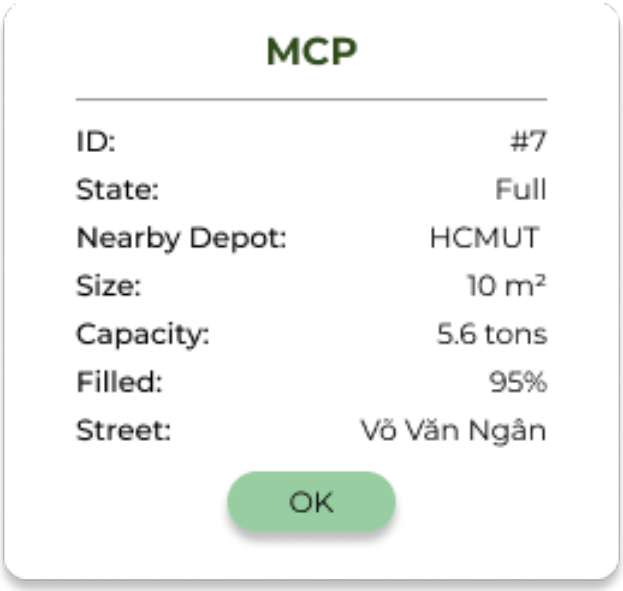
When the manager wants to view the work map, they click the "Address" button, and the map page will appear with a UI that includes a map on the right and search tools on the left. On the map, there are MCPs, depots, and when the user clicks on them, detailed information about each selected item will appear. On the left, the user can search for MCPs by selecting the information field they want in the "Name" bar, inputting data into the "Search" field, and then clicking the "Search" button. The UI map will then show the desired MCP.




HCMUT Depot
ID: #1
Range area: 2002 m²
In-range MCPs: 4 MCPs
Full MCPs: 3 MCPs
Collector: 10 workers
Janitor: 30 workers
Truck: 8 vehicals
OK

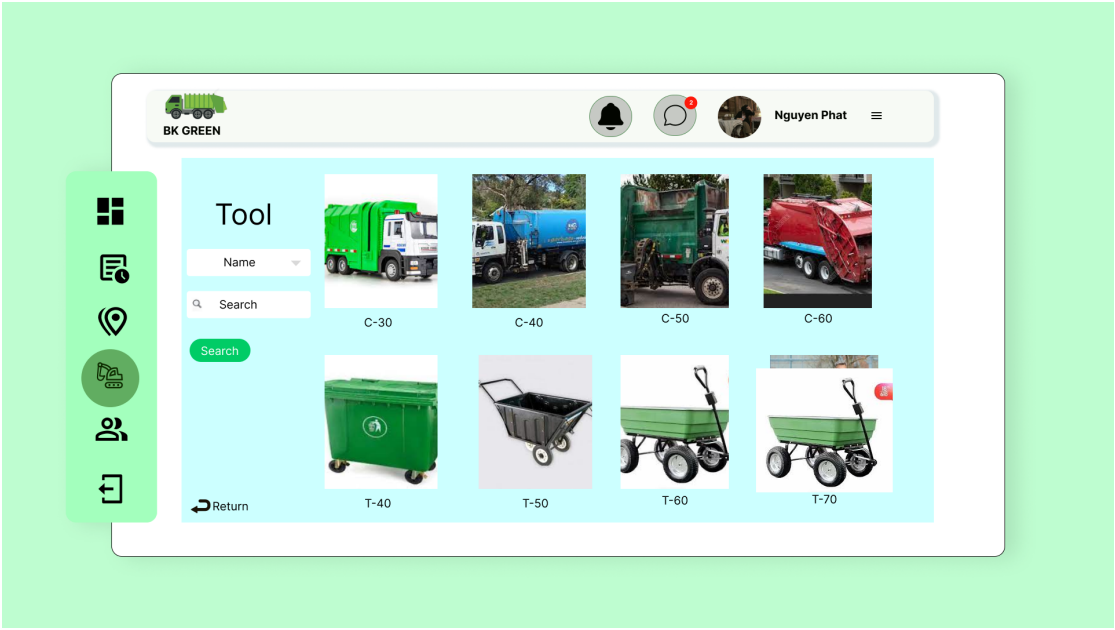

MCP
ID: #3
State: Half-full
Nearby Depot: HCMUT
Size: 7 m²
Capacity: 4 tons
Filled: 67%%
Street: Thống Nhất
OK

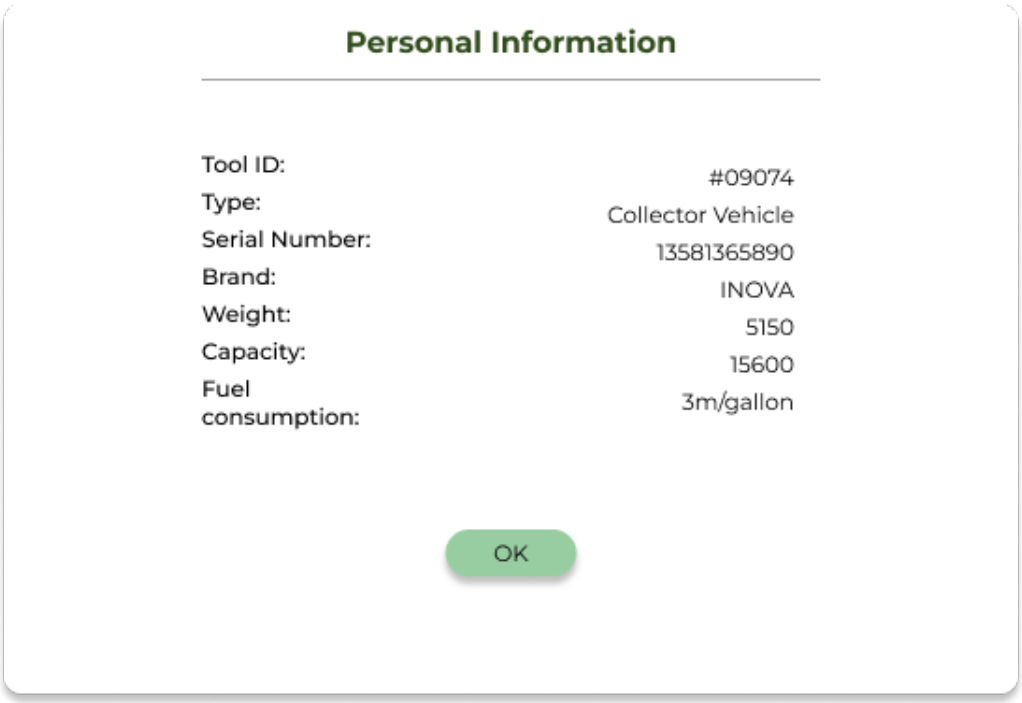

MCP
ID: #2
State: Free
Nearby Depot: HCMUT
Size: 3 m²
Capacity: 1.8 tons
Filled: 3%
Street: Linh Trung
OK


MCP
ID: #7
State: Full
Nearby Depot: HCMUT
Size: 10 m²
Capacity: 5.6 tons
Filled: 95%
Street: Võ Văn Ngân
OK

1.4.5 Page_Tool

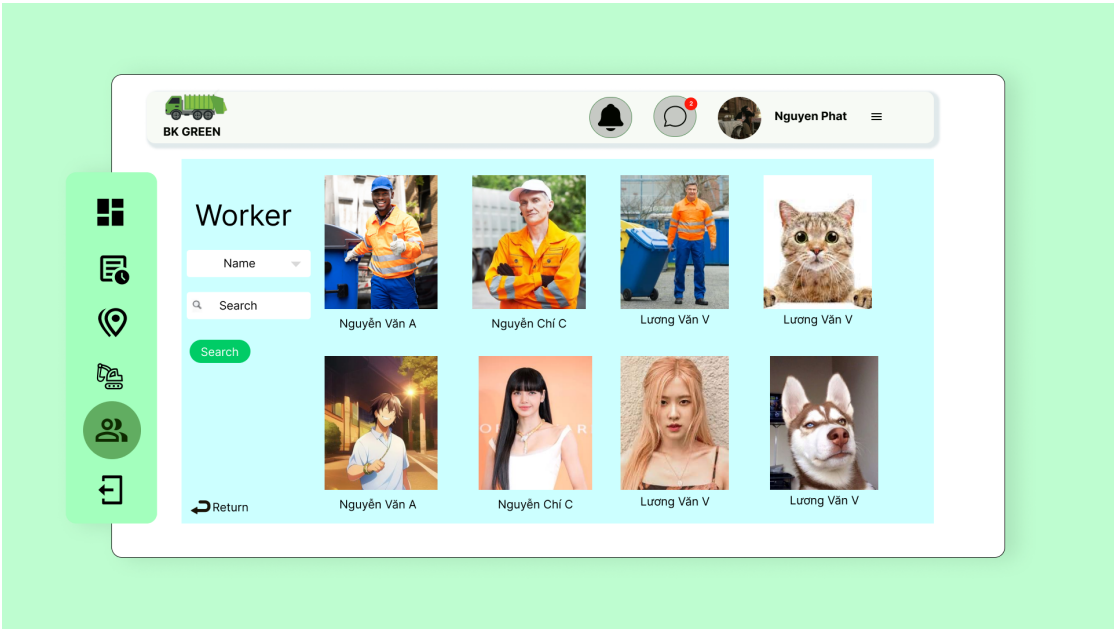
When the user clicks on the "Tool" button, they will see a UI that includes search tools on the left and a list of tools on the right. When the user clicks on the picture of any tools, information about that tool will appear. Similarly, the toolbars also have the same operations as the Map' search tools.

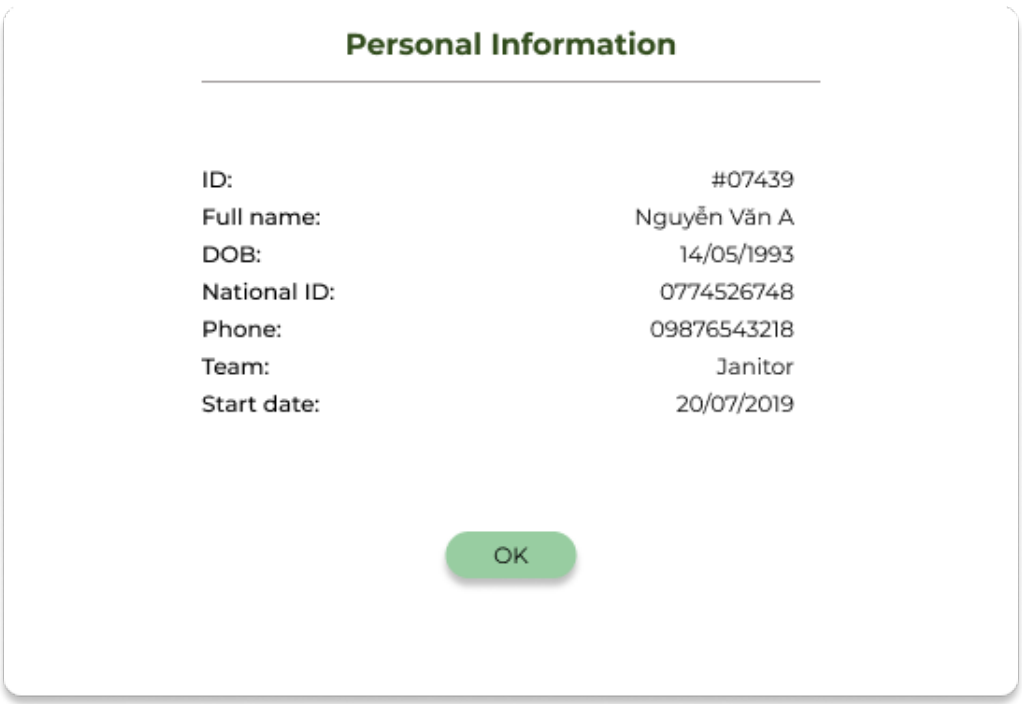


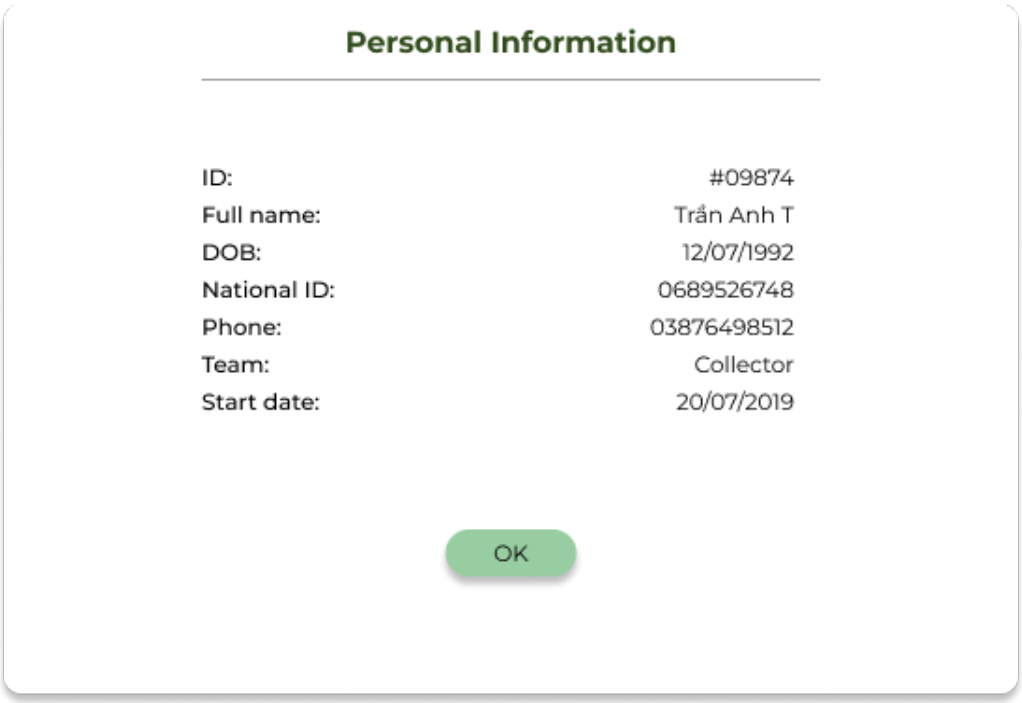


1.4.6 Page_Worker

When the user clicks on the "Worker" button, they will see a UI that includes search tools on the left and a list of workers on the right. When the user clicks on the picture of any worker, information about that person will appear. Similarly, the toolbars also have the same operations as the Map' search tools.


BK GREEN
Nguyen Phat
Worker
Name
Search
Search
Return
Nguyễn Văn A
Nguyễn Chí C
Lương Văn V
Lương Văn V
Nguyễn Văn A
Nguyễn Chí C
Lương Văn V
Lương Văn V


Personal Information
ID: #07439
Full name: Nguyễn Văn A
DOB: 14/05/1993
National ID: 0774526748
Phone: 09876543218
Team: Janitor
Start date: 20/07/2019
OK


Personal Information
ID:
#09874
Full name:
Trần Anh T
DOB:
12/07/1992
National ID:
0689526748
Phone:
03876498512
Team:
Collector
Start date:
20/07/2019
OK

2 Bibliography