Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Database Systems
Assignment 2 Report

Waste Management System Database Implementation

Group 1 - CCO2

Lecturer: Prof. Nguyen Thi Ai Thao

Team members: Bui Ngoc Nam Anh – 2052833
Le Khanh Duy – 2052003
Dang Tien Dat – 2052436
Tran Pham Minh Hung – 2053067

Ho Chi Minh City, April 26, 2023

Contents

1 Database Physical Implementation **2**
1.1 Schema 2
1.2 Stored Procedures and Functions 9
1.3 Triggers 14
1.4 Data Insertion 25

2 Application Building **34**
2.1 User Creation 34
2.1.1 Create new user 34
2.1.2 Granting user permissions 34
2.2 App Features 35
2.2.1 Authentication 35
2.2.2 Employee Management 36
2.2.3 Route Map 37

1 Database Physical Implementation

1.1 Schema

For this section, we review the schema being used to create the database corresponded to the relational map in our Assignment 1.

First ,we created a new database along with some drop queries to ensure the uniqueness of the database and its tables. These queries also improve the performance of the creation part.

```
DROP DATABASE IF EXISTS `uwc_2.0`;
CREATE DATABASE `uwc_2.0`;
USE `uwc_2.0`;
--
-- Drop exist tables to load new ones.
--

DROP TABLE IF EXISTS worktime;
DROP TABLE IF EXISTS janitor;
DROP TABLE IF EXISTS collector;
DROP TABLE IF EXISTS employee;
DROP TABLE IF EXISTS vehicle;
DROP TABLE IF EXISTS contains_mcp;
DROP TABLE IF EXISTS mcp;
DROP TABLE IF EXISTS asset_supervisors;
DROP TABLE IF EXISTS asset;
DROP TABLE IF EXISTS route;
DROP TABLE IF EXISTS back_officer;
DROP TABLE IF EXISTS `user`;
```

The tables creating queries were also included in this schema sql file.

Queries to create **asset** table with *id* as its primary key:

```
--
-- Create model Asset


CREATE TABLE `asset` (
  `id` bigint AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `is_vehicle` bool NOT NULL,
  `longtitude` numeric(10, 7) NOT NULL,
  `latitude` numeric(10, 7) NOT NULL,
  `load` numeric(9, 3) NOT NULL,
  `capacity` numeric(9, 3) NOT NULL
);
```

These are the queries to create **vehicle** table and **mcp** table with *asset_id* as their primary key. These tables also have a foreign key refer to *id* of **asset** table, which is also its primary key - *asset_id*:

```
--
-- Create model Vehicle

CREATE TABLE `vehicle` (
  `asset_id` bigint NOT NULL PRIMARY KEY,
  `type` enum('truck','trolley') NOT NULL,
  CONSTRAINT `vehicle_asset_id_b86671db_fk_asset_id`
    FOREIGN KEY (`asset_id`) REFERENCES `asset` (`id`)
    ON DELETE CASCADE
);
--
-- Create model MCP

CREATE TABLE `mcp` (
  `asset_id` bigint NOT NULL PRIMARY KEY,
  `pop_density` numeric(9, 3) NOT NULL,
  `janitor_count` bigint NOT NULL DEFAULT 0,
  CONSTRAINT `mcp_asset_id_b4092464_fk_asset_id`
    FOREIGN KEY (`asset_id`) REFERENCES `asset` (`id`)
    ON DELETE CASCADE
);
```

Queries to create **user** table with *id* as its primary key:

```
--
-- Create model MyUser

CREATE TABLE `user` (
  `id` bigint AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `username` varchar(150) NOT NULL UNIQUE,
  `password` varchar(128) NOT NULL,
  `name` varchar(150) NOT NULL,
  `is_backofficer` bool NOT NULL,
  `is_active` bool NOT NULL,
  `date_joined` datetime DEFAULT CURRENT_TIMESTAMP,
  `address` varchar(100) NULL,
  `birth` date NULL,
  `gender` enum('male','female') NULL,
  `phone` varchar(15) NULL,
  `email` varchar(254) NOT NULL UNIQUE,
  `last_login` datetime NULL
);
```

Below are the queries to create **back_officer** table and **employee** table. Both having *user_id* as their primary key as well as foreign key refer to the **user** table. The **employee** table also have two more foreign keys as *vehicle_id* refers to *asset_id* of **vehicle** and *manager_id* refers to *user_id* of **back_officer**.

```
--
-- Create model BackOfficer

CREATE TABLE `back_officer` (
  `user_id` bigint NOT NULL PRIMARY KEY,
  `employee_count` bigint NOT NULL DEFAULT 0,
  `route_count` bigint NOT NULL DEFAULT 0,
  `vehicle_count` bigint NOT NULL DEFAULT 0,
  `MCP_count` bigint NOT NULL DEFAULT 0,
  CONSTRAINT `back_officer_user_id_813c1bfc_fk_user_id`
    FOREIGN KEY (`user_id`) REFERENCES `user` (`id`) ON DELETE CASCADE
);
```

```
--
-- Create model Employee

CREATE TABLE `employee` (
  `user_id` bigint AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `manager_id` bigint NULL,
  `vehicle_id` bigint NULL,
  `is_working` bool NOT NULL,
  `is_collector` bool NOT NULL,
  `start_date` date DEFAULT (CURRENT_DATE),
  `salary` bigint NOT NULL DEFAULT 6000000,
  CONSTRAINT `employee_vehicle_id_d04fc52a_fk_vehicle_id`
    FOREIGN KEY (`vehicle_id`) REFERENCES `vehicle`(`asset_id`)
    ON DELETE SET NULL,
  CONSTRAINT `employee_manager_id_54b357b6_fk_back_officer_id`
    FOREIGN KEY (`manager_id`) REFERENCES `back_officer` (`user_id`)
    ON DELETE SET NULL,
  CONSTRAINT `employee_id_cc4f5a1c_fk_user_id`
    FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
    ON DELETE CASCADE
);
```

Queries to create **worktime** table with *employee_id* refers to *user_id* of **employee** as its foreign key and *(start,end, weekday,employee_id)* as its primary key:

```
-- Create model WorkTime

CREATE TABLE `worktime` (
  `start` time NOT NULL,
  `end` time NOT NULL,
  `weekday` enum('Mon','Tue','Wed','Thur','Fri','Sat','Sun') NOT NULL,
  `employee_id` bigint NOT NULL,
  `id` bigint AUTO_INCREMENT NOT NULL UNIQUE,
  CONSTRAINT `schedule_employee_prime`
    PRIMARY KEY (`start`,`end`,`weekday`,`employee_id`),
  CONSTRAINT `worktime_employee_id_b2364680_fk_employee_id`
    FOREIGN KEY (`employee_id`) REFERENCES `employee` (`user_id`)
    ON DELETE CASCADE
);
```

Queries to create **route** table with *id* as its primary key and *manager_id* as its foreign key, refers to *user_id* of **back_officer**:

```
--
-- Create model Route

CREATE TABLE `route` (
  `id` bigint AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `last_update` datetime DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP,
  `created_at` datetime DEFAULT CURRENT_TIMESTAMP,
  `distance` numeric(9, 3) NULL,
  `manager_id` bigint NULL,
  CONSTRAINT `route_manager_id_77cebb21_fk_back_officer_id`
    FOREIGN KEY (`manager_id`) REFERENCES `back_officer` (`user_id`)
    ON DELETE SET NULL
);
```

The **janitor**table has *employee_id* both as its primary key and foreign key which refers to *user_id* of the **employee** table. *mcp_id*, which refers to *asset_id* of the **mcp** table is also its foreign key.

```
--
-- Create model Janitor

CREATE TABLE `janitor` (
  `mcp_start_date` date DEFAULT (CURRENT_DATE),
  `work_radius` numeric(9, 3) NULL,
  `employee_id` bigint PRIMARY KEY,
  `mcp_id` bigint NULL,
  CONSTRAINT `janitor_employee_id_12a5ab16_fk_employee_id`
    FOREIGN KEY (`employee_id`) REFERENCES `employee` (`user_id`)
    ON DELETE CASCADE,
  CONSTRAINT `janitor_mcp_id_c035f717_fk_mcp_id`
    FOREIGN KEY (`mcp_id`) REFERENCES `mcp` (`asset_id`)
    ON DELETE SET NULL
);
```

The **collector** table has *employee_id* both as its primary key and foreign key which refers to *user_id* of the **employee** table. *route_id*, which refers to *id* of the **route** table is also its foreign key.

```
--
-- Create model Collector

CREATE TABLE `collector` (
  `employee_id` bigint PRIMARY KEY,
  `route_id` bigint NULL,
  CONSTRAINT `collector_employee_id_34075942_fk_employee_id`
    FOREIGN KEY (`employee_id`) REFERENCES `employee` (`user_id`)
    ON DELETE CASCADE,
  CONSTRAINT `collector_route_id_65a37aa4_fk_route_id`
    FOREIGN KEY (`route_id`) REFERENCES `route` (`id`)
    ON DELETE SET NULL
);
```

The **asset_supervisors** table takes *asset_id* and *backofficer_id* as its foreign key. These key refer to *id* from **asset** table and *user_id* from **back_officer** table, respectively. The primary key of this table then contains these two key only.

```
--
-- Add field supervisors to asset

CREATE TABLE `asset_supervisors` (
  `asset_id` bigint NOT NULL,
  `backofficer_id` bigint NOT NULL,
  CONSTRAINT `asset_supervisors_asset_id_backofficer_id
    _6107fdf4_uniq`
    PRIMARY KEY (`asset_id`, `backofficer_id`),
  CONSTRAINT `asset_supervisors_asset_id_308d9954_fk_asset_id`
    FOREIGN KEY (`asset_id`) REFERENCES `asset` (`id`)
    ON DELETE CASCADE,
  CONSTRAINT `asset_supervisors_backofficer_id_6e97945b_
    fk_back_officer_id`
    FOREIGN KEY (`backofficer_id`) REFERENCES `back_officer`
    (`user_id`)
    ON DELETE CASCADE
);
```

The **contains_mcp** has *mcp_id* and *route_id* as its foreign key. These key refer to *asset_id* from **mcp** table and *id* from **route** table, respectively. The primary key of this table then contains these two key only.

```
--
-- Create model Contain

CREATE TABLE `contains_mcp` (
  `order` smallint NOT NULL,
  `mcp_id` bigint NOT NULL,
  `route_id` bigint NOT NULL,
  CONSTRAINT `contain_route_id_mcp_id_order_9f9f8a58_prime`
    PRIMARY KEY (`route_id`, `mcp_id`),
  CONSTRAINT `contain_mcp_id_d8a5b2b5_fk_mcp_id`
    FOREIGN KEY (`mcp_id`) REFERENCES `mcp` (`asset_id`)
    ON DELETE CASCADE,
  CONSTRAINT `contain_route_id_f497993c_fk_route_id`
    FOREIGN KEY (`route_id`) REFERENCES `route` (`id`)
    ON DELETE CASCADE,
  CONSTRAINT `contain_route_id_order_uniq_23jh23l`
  UNIQUE (`route_id`, `order`)
);
```

1.2 Stored Procedures and Functions

In this section and its following, we review some queries made in Assignment 1 and create stored procedures and functions for them. Beyond that, we will introduce additional stored procedures and functions that are possibly used in our application in Section 2.

Queries

Num	Description
1	Query all on-site collectors.
2	Query the locations of all on-site vehicles.
3	Query the route layout of a collector whose ID equals k.
4	For every weekday, count the number of shifts, the employee whose ID = k has to work.
5	Query all on-site janitors around an MCP whose ID = k.
6	Query the total load of a route whose ID = k.
7	Query all trucks that perform tasks on a route whose ID = k.

Below we specify the above queries in SQL procedural syntax.

```
-- Query all on-site collectors
    DELIMITER |
    DROP PROCEDURE IF EXISTS `GetWorkingCollector`|
    CREATE PROCEDURE `GetWorkingCollector` ()
    BEGIN
            SELECT *
            FROM collector, employee
            WHERE user_id = employee_id AND
                      `is_working` = 1;
    END |
```

```
-- Query the locations of all on-site vehicles
    DELIMITER |
    DROP PROCEDURE IF EXISTS `LocateWorkingVehicle`|
    CREATE PROCEDURE `LocateWorkingVehicle` ()
    BEGIN
            SELECT *
            FROM employee em, vehicle ve
            WHERE vehicle_id = ve.asset_id AND
                      em.is_working = 1;
    END |
```

```
-- Query the route layout of a collector whose ID equals k
    DELIMITER |
    DROP PROCEDURE IF EXISTS `GetRouteOnCollector`|
    CREATE PROCEDURE `GetRouteOnCollector` (col_id BIGINT)
    BEGIN
            SELECT id
            FROM contains_mcp, collector
            WHERE contains_mcp.route_id = collector.route_id AND
                      collector.employee_id = col_id;
    END |
```

```
-- For every weekday, count the number of shifts, the employee
   whose ID = k has to work
   DELIMITER |
   DROP PROCEDURE IF EXISTS `CountDailyShift`|
   CREATE PROCEDURE `CountDailyShift` (emp_id BIGINT)
   BEGIN
           SELECT COUNT(*) AS Num_of_shifts, weekday
           FROM worktime, employee
           WHERE user_id = employee_id AND
                     user_id = emp_id AND
           GROUP BY weekday;
   END |
```

```
-- Query all on-site janitors around an MCP whose ID = k
   DELIMITER |
   DROP PROCEDURE IF EXISTS `GetWorkingJanitorOfMCP`|
   CREATE PROCEDURE `GetWorkingJanitorOfMCP` (mcp_id BIGINT)
   BEGIN
           SELECT *
           FROM janitor, employee
           WHERE user_id = employee_id AND
                     `is_working` = 1 AND
                     janitor.mcp_id = mcp_id;
   END |
```

```
-- Query the total load of a route whose ID = k
    DELIMITER |
    DROP PROCEDURE IF EXISTS `GetRouteLoad`|
    CREATE PROCEDURE `GetRouteLoad`(
            route_id BIGINT,
            OUT `load` NUMERIC(9,3)
    )
    BEGIN
            SELECT SUM(asset.`load`) INTO `load`
            FROM route, contains_mcp, asset
            WHERE route.id = route_id AND
                    contains_mcp.route_id = route.id AND
                    contains_mcp.mcp_id = asset.id;
    END |
```

```
-- Query all trucks that perform tasks on a route whose ID = k.
    DELIMITER |
    DROP PROCEDURE IF EXISTS `GetTruckOnRoute`|
    CREATE PROCEDURE `GetTruckOnRoute` (route_id BIGINT)
    BEGIN
            SELECT *
            FROM collector, employee, vehicle
            WHERE user_id = employee_id AND
                    vehicle_id = asset_id AND
                    is_working = 1 AND
                    `type` = 'truck' AND
                    collector.route_id = route_id;
    END |
```

That sums up our queries demonstrated in Assignment 1. Now comes the procedures that are *practically* used in this application. We shall tabulate their descriptions without going further into detail since some of the implementations are quite long.

User Management

Procedure	Description
`InsertEmployee()`	Receive employee information to insert into the database
`UpdateEmployee()`	Perform update on employee info
`DeleteEmployee()`	Delete an employee from the database
`GetUserFromLogin()`	Perform authentication on a user upon login
`GetUser()`	Retrieve profile of an user given ID
`GetEmployees()`	Retrieve a list of all employees working under a backofficer

Map Management

Procedure	Description
`InsertRoute()`	Create a new route
`DeleteRoute()`	Delete a route and its layout from the database
`InsertMCPToRoute()`	Insert an MCP to a route
`RetrieveMCPsFromRoute()`	Acquire all MCPs from a route
`DeleteMCPsFromRoute()`	Delete all MCPs from a route without deleting itself
`RetrieveMap()`	Acquire all MCPs info from the map managed by the backofficer
`RetrieveRoutes()`	Retrieve a list of routes without their layout under the backofficer
`GetDistance()`	Retrieve the total distance of a route
`UpdateDistance()`	Update the total distance of a route
`GetMaxCapacity()`	Find the max capacity between all the trucks on a route.

Task Management

Procedure	Description
`AssignAreaToJanitor()`	Assign area to a janitor
`AssignRouteToCollector()`	Assign route to a collector
`InsertShift()`	Insert a shift to an employee's calendar
`RetrieveSchedule()`	Acquire all the weekly shift of an employee
`DeleteShift()`	Delete a shift from an employee's calendar

1.3 Triggers

For this section, we review the queries to create triggers. Firstly, we will show the summary for triggers in the table below, before show in code in detail.

User Management

Trigger	Input (Condition to Trigger)	Output (Trigger Action)
UserPhone_trigger_insert	Insert user phone number with wrong format	Prevent Insertion, Throw error.
UserPhone_trigger_update	Update user phone number with wrong format	Prevent Update, Throw error.
Birth_trigger_insert	Insert user birthday who is under 18 years old	Prevent Insertion, Throw error.
Birth_trigger_update	Update user birthday who is under 18 years old	Prevent Update, Throw error.
work_radius_trigger_insert	Insert work_radius on janitor exceed 500m	Set work_radius to 500m.
work_radius_trigger_update	Update work_radius on janitor exceed 500m	Set work_radius to 500m.

Back Officer Management Count

Trigger	Input (Condition to Trigger)	Output (Trigger Action)
count_employee_trigger	Insert new tuple on employee	Increase employee_count on back_officer.
delete_count_employee	Delete tuple on employee	Decrease employee_count on back_officer.
update_count_employee	Update manager_id on employee	Increase employee_count on NEW back_officer, Decrease employee_count on OLD back_officer.
count_asset_trigger	Insert new tuple on asset_supervisors	If asset is MCP: Increase MCP_count on back_officer. Otherwise: Increase vehicle_count on back_officer.

Trigger	Input (Condition to Trigger)	Output (Trigger Action)
update_count_asset_trigger	Update backofficer_id on asset_supervisors	If asset is MCP: - Increase MCP_count on NEW back_officer, - Decrease MCP_count on OLD back_officer. Otherwise: - Increase vehicle_count on NEW back_officer. - Decrease vehicle_count on OLD back_officer.
delete_count_asset_trigger	Delete tuple on asset_supervisors	If asset is MCP: Decrease MCP_count on back_officer. Otherwise: Decrease vehicle_count on back_officer.
count_route_trigger	Insert new tuple on route	Increase route_count on back_officer.
update_count_route_trigger	Update manager_id on route	Increase route_count on NEW back_officer. Decrease route_count on OLD back_officer.
delete_count_route_trigger	Delete tuple on route	Decrease route_count on back_officer.
count_janitor_trigger	Update mcp_id on janitor	If OLD mcp_id is NULL or update another mcp_id on janitor: - Increase janitor_count on NEW mcp. If exist OLD mcp_id and it is not the same with NEW mcp_id: - Decrease janitor_count on OLD mcp.
delete_count_janitor_trigger	delete tuple on janitor	If exist mcp_id: Decrease janitor_count on mcp

Map Management

Trigger	Input (Condition to Trigger)	Output (Trigger Action)
asset_load_trigger_insert	Insert load on asset exceed the capacity	Set load equal to capacity.
asset_load_trigger_update	Update load on asset exceed the capacity	Set load equal to capacity.
overloaded_mcp_route_insert	Insert mcp on contains_mcp, total load of MCP on route exceeds max capacity of the vehicles on that route.	Prevent Insertion, Throw error.

Task Management

Trigger	Input (Condition to Trigger)	Output (Trigger Action)
end_time_trigger_insert	Insert new tuple on worktime, end_time is not after start_time.	Prevent Insertion, Throw error.
Shift_Insert	Insert new tuple on worktime, collision with other exist shift of the same employee.	Prevent Insertion, Throw error.

Now we review detail in some important triggers. In this section, we use Delimeter to seperate each trigger, make the code clearer and prevent error with semicolon.

```
DELIMITER $$
```

Phone number format trigger:

```
-- Create phone number contraint
DROP TRIGGER IF EXISTS $$
CREATE TRIGGER UserPhone_trigger_insert
        BEFORE INSERT
    ON user FOR EACH ROW
BEGIN
        DECLARE phone_mess varchar(128);
        IF (length(NEW.phone) > 10
            OR (SELECT left(NEW.phone, 1)) <> '0') THEN
                SET phone_mess = concat('INSERT_Error:
                Phone number should have at max 10 digit,
                start at '0': ', new.phone);
        SIGNAL sqlstate '45000' SET message_text = phone_mess;
    END IF;
END$$
```

Birthday check trigger:

```
-- Create 18+ constraint
DROP TRIGGER IF EXISTS Birth_trigger_insert$$
CREATE TRIGGER Birth_trigger_insert
        BEFORE INSERT
    ON user FOR EACH ROW
BEGIN
        DECLARE bdate_mess varchar(128);
        IF timestampdiff(YEAR, NEW.birth, CURDATE()) < 18 THEN
                SET bdate_mess = concat('INSERT_Error:
                User should be over 18 years old: ',
                cast(new.birth as char));
                SIGNAL sqlstate '45000' SET message_text = bdate_mess;
    END IF;
END$$
```

Trigger incase total MCP load exceeds capacity of vehicle on route:

```
-- Create trigger to react if insert mcp overloading the truck on route.
DROP TRIGGER IF EXISTS overloaded_mcp_route_insert$$
CREATE TRIGGER overloaded_mcp_route_insert
        BEFORE INSERT
    ON contains_mcp FOR EACH ROW
BEGIN
        DECLARE mcp_overload_mess VARCHAR(128);
        IF ((GetRouteLoad(NEW.route_id)
        + (SELECT `load` FROM asset
            WHERE asset.id = NEW.mcp_id))
        > GetMaxCapacity(NEW.route_id)) THEN
                SET mcp_overload_mess = CONCAT('Collector Truck
                will be overloaded on this route if MCP: ',
                cast(NEW.mcp_id as char),' is added into route: ',
                cast(NEW.route_id as char),'.');
        SIGNAL sqlstate '45000' SET message_text = mcp_overload_mess;
    END IF;
END$$
```

In this trigger, we use a function $GetMaxCapacity()$ to find the capacity of the vehicle on the route which has the max capacity.

```
-- Create Function to find the max capacity of all trucks on the same route.
DROP FUNCTION IF EXISTS GetMaxCapacity$$
CREATE FUNCTION GetMaxCapacity (route_mcp_id numeric(9, 3))
RETURNS numeric(9, 3)
DETERMINISTIC
BEGIN
        DECLARE max_cap numeric(9, 3);
    SELECT MAX(capacity) INTO max_cap
                FROM collector, employee, asset
        WHERE
                        route_mcp_id = collector.route_id AND
            asset.id = employee.vehicle_id;
        RETURN max_cap;
END$$
```

Trigger to decrease employee count on backofficer if an employee tuple is deleted:

```
DROP TRIGGER IF EXISTS delete_count_employee$$
CREATE TRIGGER delete_count_employee
        AFTER DELETE
        ON employee for each row
BEGIN
        UPDATE back_officer
                SET employee_count = employee_count - 1
        WHERE back_officer.user_id = OLD.manager_id;
END $$
```

Trigger to alter MCP or vehicle count on backofficer if update on asset-supervisors:

```
-- Trigger count MCP and vehicle supervised by back_officer
DROP TRIGGER IF EXISTS update_count_asset_trigger $$
CREATE TRIGGER update_count_asset_trigger
        AFTER UPDATE
    ON asset_supervisors FOR EACH ROW
BEGIN
        IF (SELECT is_vehicle FROM asset
                WHERE asset.id = OLD.asset_id) = 0 THEN
                -- Update MCP
        BEGIN
                        UPDATE back_officer
                                SET MCP_count = MCP_count - 1
                                WHERE
                                 user_id = OLD.backofficer_id;
                        UPDATE back_officer
                                SET MCP_count = MCP_count + 1
                                WHERE
                                 user_id = NEW.backofficer_id;
                END;
```

```
        ELSE
                        -- Update Vehicle
                BEGIN
                        UPDATE back_officer
                                SET vehicle_count = vehicle_count - 1
                                WHERE
                                 user_id = OLD.backofficer_id;
                        UPDATE back_officer
                                SET vehicle_count = vehicle_count + 1
                                WHERE
                                 user_id = NEW.backofficer_id;
                END;
        END IF;
END$$
```

Trigger to alter janitor count works at mcp:

```
-- Trigger count janitors work at MCP
DROP TRIGGER IF EXISTS count_janitor_trigger$$
CREATE TRIGGER count_janitor_trigger
        AFTER UPDATE
    ON janitor FOR EACH ROW
BEGIN
        IF (OLD.mcp_id IS NULL OR (OLD.mcp_id IS NOT NULL
            AND OLD.mcp_id <> NEW.mcp_id)) THEN
                UPDATE mcp
                        SET janitor_count = janitor_count + 1
                        WHERE mcp.asset_id = NEW.mcp_id;
        END IF;
        IF (OLD.mcp_id IS NOT NULL AND OLD.mcp_id <> NEW.mcp_id)
        THEN
                UPDATE mcp
                        SET janitor_count = janitor_count - 1
                        WHERE mcp.asset_id = OLD.mcp_id;
    END IF;
END$$
```

Trigger to verify if insert end-time after start-time in worktime:

```
-- Crete end time - start time contraint
DROP TRIGGER IF EXISTS end_time_trigger_insert$$
CREATE TRIGGER end_time_trigger_insert
        BEFORE INSERT
    ON worktime FOR EACH ROW
BEGIN
        DECLARE endtime_mess varchar(128);
    IF timestampdiff(SECOND, NEW.start, NEW.end) <= 0 THEN
                SET endtime_mess = concat('INSERT_Error: End time
                should after start time: ', cast(new.start as char),
                ' - ', cast(new.end as char));
                SIGNAL sqlstate '45000'
                SET message_text = endtime_mess;
    END IF;
END$$
```

Trigger to verify if insert new shift without no collision:

```
-- Shift Inset trigger
DROP TRIGGER IF EXISTS Shift_Insert $$
CREATE TRIGGER Shift_Insert
        BEFORE INSERT
    ON worktime FOR EACH ROW
BEGIN
        DECLARE overload_shift_mess VARCHAR(128);
    DECLARE is_overloaded bool DEFAULT 0;
        DECLARE exit_shift bool DEFAULT 0;
    DECLARE start_time time;
        DECLARE end_time time;
        DECLARE curShift CURSOR
                FOR SELECT `start`, `end` FROM worktime
                WHERE
                        worktime.employee_id = NEW.employee_id AND
                                worktime.weekday = NEW.weekday;
    DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET exit_shift = 1;
```

```
    OPEN curShift;
    GetShift: LOOP
                FETCH curShift INTO `start_time`, `end_time`;
        IF exit_shift = 1 THEN LEAVE GetShift;
        END IF;
        -- New start time before old end time
        IF timestampdiff(second, NEW.start, end_time) > 0 THEN
                        -- New end time after old start time
            IF timestampdiff(second, NEW.end, start_time) < 0 THEN
                                SET is_overloaded = 1;
                        END IF;
        END IF;
        IF (is_overloaded) THEN
           SET overload_shift_mess = CONCAT('Shift
                    inserted must not overloading exist shifts: ',
                                    cast(NEW.start as char),' - ',
                    cast(NEW.end as char), ' on: ',
                    cast(NEW.weekday as char));
           SIGNAL sqlstate '45000'
                    SET message_text = overload_shift_mess;
        END IF;
    END LOOP GetShift;
    CLOSE curShift;
END $$
```

And finally, at the end of triggers, we end the Delimiter by:

```
DELIMITER ;
```

Command Showcase

We have insert to database the following mcp id = 1

id	is_vehicle	longtitude	latitude	load	capacity
1	0	106.6575010	10.7730450	75.990	395.141

Figure 1: Asset Before

Now, update load of mcp:

```
UPDATE asset
    SET `load` = 900
    WHERE id = 1;
```

Because of the triggger, the load of MCP is set to its capacity.

id	is_vehicle	longtitude	latitude	load	capacity
1	0	106.6575010	10.7730450	395.141	395.141

Figure 2: Asset After

Next, update the capacity and load of MCP 1, and we assign vehicle to collector and assign him to route:

```
UPDATE asset
    SET `capacity` = 900
    WHERE id = 1;
UPDATE asset
    SET `load` = 900
    WHERE id = 1;
UPDATE employee
    SET vehicle_id = 13
    WHERE user_id = 4;
UPDATE collector
    SET route_id = 1
    WHERE employee_id = 4;
```

id	is_vehicle	longtitude	latitude	load	capacity
1	0	106.6575010	10.7730450	900.000	900.000

Figure 3: MCP 1 Done

id	is_vehicle	longtitude	latitude	load	capacity
11	0	106.6579610	10.7832080	0.000	319.947
12	0	106.6533470	10.7708700	99.736	338.839
13	1	106.6257860	10.8106760	0.000	994.985

Figure 4: Vehicle 13

order	mcp_id	route_id
1	1	1
3	3	1
4	4	1
8	8	1
2	2	2
6	6	2
7	7	2
9	9	2
NULL	NULL	NULL

Figure 5: Route before

To find the total load of MCP on route:

```
SELECT GetRouteLoad(1);
```

GetRouteLoad(1)
951.994

Figure 6: Total load

Now, we will add MCP 9 to route, Mysql will state error.

```
SELECT `load` FROM asset WHERE id = 9;
CALL InsertMCPToRoute(9,1,9);
```

load
48.210

Figure 7: MCP 9 load

1142 21:26:26 CALL InsertMCPToRoute(9,1,9) Error Code: 1644. Collector Truck will be overloaded on this route if MCP: 9 is added into route: 1. 0.000 sec

Figure 8: Error Thrown

order	mcp_id	route_id
1	1	1
3	3	1
4	4	1
8	8	1
2	2	2
6	6	2
7	7	2
NULL	NULL	NULL

Figure 9: The route after

1.4 Data Insertion

For this section, we review the queries to insert data to our database. The database tables were inserted through many ways such as using straight **INSERT TO** command, using **CALL** to insert through stored procedure or insert on the modification of other tables.

Insert using **INSERT INTO** command:

```
-- Insert back officer
INSERT INTO `user`
VALUES (NULL,'VitNguNgok21',sha2('lulucc1122',0),'Le Van Luyen',1,0,
SYSDATE(),NULL,NULL,'male',NULL,'duyvt763@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets
```

As we can see in the queries above, the table **user** has been inserted by using the default **INSERT TO** function with the given parameters.

Insert by calling stored procedure:

```
-- Insert MCP
CALL InsertMCP(106.657501, 10.773045, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp1 = last_insert_id();
CALL InsertMCP(106.653745, 10.786654, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp2 = last_insert_id();
CALL InsertMCP(106.658821, 10.780086, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp3 = last_insert_id();
CALL InsertMCP(106.663421, 10.775828, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.659960, 10.764715, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.662589, 10.778742, RAND()*100, RAND()*100+300,
RAND()*60, 0);
```

These queries insert rows into the MCP table using **CALL** with parameters to call the stored procedure.

Insert on the modification of other tables:

```
-- procedure to insert MCP
DELIMITER |
DROP PROCEDURE IF EXISTS `InsertMCP`|
CREATE PROCEDURE `InsertMCP` (
        longtitude NUMERIC(10,7),
        latitude NUMERIC(10,7),
        `load` NUMERIC(9,3),
        capacity NUMERIC(9,3),
        pop_density NUMERIC(9,3),
        janitor_count BIGINT
)
BEGIN
        DECLARE temp_id BIGINT;
        INSERT INTO asset
        VALUES (NULL, 0, longtitude, latitude, `load`, capacity);
        SET temp_id=last_insert_id();
        INSERT INTO mcp
        VALUES (temp_id, pop_density, janitor_count);
END |
```

In the stored procedure above to insert the data to the MCP, during the insertion we also insert to the table asset. This must be done since the table MCP has a foreign key from table asset and we need to create an asset first before create the corresponded MCP.

Below are our entire queries for inserting tables with the minimum number of rows ensured.

Insert to the **MCP** table and the corresponded **asset** table:

```
-- Insert MCP
CALL InsertMCP(106.657501, 10.773045, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp1 = last_insert_id();
CALL InsertMCP(106.653745, 10.786654, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp2 = last_insert_id();
CALL InsertMCP(106.658821, 10.780086, RAND()*100, RAND()*100+300,
RAND()*60, 0);
SET @mcp3 = last_insert_id();
CALL InsertMCP(106.663421, 10.775828, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.659960, 10.764715, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.662589, 10.778742, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.666257, 10.776935, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.663459, 10.782816, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.669609, 10.784208, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.664690, 10.786800, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.657961, 10.783208, RAND()*100, RAND()*100+300,
RAND()*60, 0);
CALL InsertMCP(106.653347, 10.770870, RAND()*100, RAND()*100+300,
RAND()*60, 0);
```

Insert to the table **Vehicle** and the corresponded **asset** table:

```
-- Insert vehicle
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+900,'truck');
SET @truck1 = last_insert_id();
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+900,'truck');
SET @truck2 = last_insert_id();
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+900,'truck');
SET @truck3 = last_insert_id();
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+100,'trolley');
SET @troll1 = last_insert_id();
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+100,'trolley');
SET @troll2 = last_insert_id();
CALL InsertVehicle(106.625786,10.810676,0,RAND()*100+100,'trolley');
SET @troll3 = last_insert_id();
```

Insert to the table **backofficer** and the corresponded **user** table:

```
-- Insert back officer
INSERT INTO `user`
VALUES (NULL,'VitNguNgok21',sha2('lulucc1122',0),'Le Van Luyen',
1,0,SYSDATE(),NULL,NULL,'male',NULL,'duyvt763@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets

INSERT INTO `user`
VALUES (NULL,'Toinuoimeo24',sha2('eheheh123',0),'Pham Minh Chinh',
1,0,SYSDATE(),NULL,NULL,'male',NULL,'minhchinh24@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets
```

```
INSERT INTO `user`
VALUES (NULL,'phucnho11',sha2('nngp1102',0),'Nguyen Phuc Nho',
,0,SYSDATE(),NULL,NULL,'male',NULL,'phucnho11@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets


INSERT INTO `user`
VALUES (NULL,'TienMinh13',sha2('sussycoder123',0),'Dang Tien Minh',
1,0,SYSDATE(),NULL,NULL,'male',NULL,'tienminhsus123@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets


INSERT INTO `user`
VALUES (NULL,'ChienHugo33',sha2('BukBuky320',0),'Ho Thanh Than',
1,0,SYSDATE(),NULL,NULL,'male',NULL,'BukBuk321@gmail.com',NULL);
SET @temp_id = last_insert_id();
INSERT INTO back_officer
VALUES (@temp_id,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INSERT INTO asset_supervisors SELECT id, @temp_id FROM asset;
-- let this back officers supervise all assets
SET @latest_bo_id = @temp_id;
```

Insert to the **route** table and the corresponded **contains_mcp** table:

```
-- Insert routes
INSERT INTO route
VALUES (NULL, DEFAULT, DEFAULT, NULL, @latest_bo_id);
SET @temp_id = last_insert_id();
CALL InsertMCPToRoute(1, @temp_id, 1);
CALL InsertMCPToRoute(3, @temp_id, 2);
CALL InsertMCPToRoute(4, @temp_id, 3);
CALL InsertMCPToRoute(8, @temp_id, 4);
SET @route1 = @temp_id;

INSERT INTO route
VALUES (NULL, DEFAULT, DEFAULT, NULL, @latest_bo_id);
SET @temp_id = last_insert_id();
CALL InsertMCPToRoute(2, @temp_id, 1);
CALL InsertMCPToRoute(6, @temp_id, 2);
CALL InsertMCPToRoute(7, @temp_id, 3);
CALL InsertMCPToRoute(9, @temp_id, 4);
SET @route2 = @temp_id;

INSERT INTO route
VALUES (NULL, DEFAULT, DEFAULT, NULL, @latest_bo_id);
SET @temp_id = last_insert_id();
CALL InsertMCPToRoute(1, @temp_id, 1);
CALL InsertMCPToRoute(9, @temp_id, 2);
CALL InsertMCPToRoute(2, @temp_id, 3);
CALL InsertMCPToRoute(4, @temp_id, 4);

INSERT INTO route
VALUES (NULL, DEFAULT, DEFAULT, NULL, @latest_bo_id);
SET @temp_id = last_insert_id();
CALL InsertMCPToRoute(1, @temp_id, 1);
CALL InsertMCPToRoute(3, @temp_id, 2);
CALL InsertMCPToRoute(4, @temp_id, 3);
CALL InsertMCPToRoute(2, @temp_id, 4);
```

```
INSERT INTO route
VALUES (NULL, DEFAULT, DEFAULT, NULL, @latest_bo_id);
SET @temp_id = last_insert_id();
CALL InsertMCPToRoute(2, @temp_id, 1);
CALL InsertMCPToRoute(8, @temp_id, 2);
CALL InsertMCPToRoute(4, @temp_id, 3);
CALL InsertMCPToRoute(1, @temp_id, 4);
```

Insert into the **employee** table and the corresponded **worktime**, **user** and as well as **janitor** or **collector** based on their job:

```
-- Insert employee and let one back officer manages them
SET @dummy = NULL;
CALL InsertEmployee(@dummy,'trangbku123','trangthaomai212',
'Doan Thi Trang',@latest_bo_id,ROUND(RAND()),SYSDATE(),NULL,NULL,
'female','0123567980','trangdoan@gmail.com',NULL,0,6000000);
CALL AssignAreaToJanitor(200+RAND()*300, @mcp1, CURDATE(),
last_insert_id());
UPDATE employee SET vehicle_id = @troll1 WHERE user_id =
last_insert_id();
CALL InsertShift(@dummy,'00:00:00','02:00:00','Mon',
last_insert_id());

CALL InsertEmployee(@dummy,'giaunhucho999','lotto77777',
'Le Van Giau',@latest_bo_id,ROUND(RAND()),SYSDATE(),NULL,NULL,
'male','0987123789','ngheo12@gmail.com',NULL,1,7000000);
CALL AssignRouteToCollector(@route1,last_insert_id());
UPDATE employee SET vehicle_id = @truck1 WHERE user_id =
last_insert_id();
CALL InsertShift(@dummy,'05:00:00','07:00:00','Sat',
last_insert_id());

CALL InsertEmployee(@dummy,'Thythycute666','06052001Thy',
'Khuc Thy',@latest_bo_id,ROUND(RAND()),SYSDATE(),NULL,NULL,
'female','0987123789','thythy99@gmail.com',NULL,0,6000000);
CALL AssignAreaToJanitor(200+RAND()*300, @mcp2, CURDATE(),
last_insert_id());
UPDATE employee SET vehicle_id = @troll2 WHERE user_id =
last_insert_id();
CALL InsertShift(@dummy,'09:00:00','11:30:00','Thur',
last_insert_id());
```

The tables inserted using by these queries will be shown at the end of the report, in **Appendix A**.

2 Application Building

2.1 User Creation

This section is for demonstrating how we create user and grant permission when working with **MYSQL** database management system.

2.1.1 Create new user

Upon installation, MYSQL default created a **root** user account which have full priviledge over MYSQL server. So that it is avoid to use this account outside of adminstrative functions.
Open MYSQL shell and login into MYSQL root account. Create new user with your own username and password.

```
\connect root@localhost
```

```
CREATE USER 'uwcmanager'@'localhost'
IDENTIFIED BY '123456789';
```

Because we want to create a user locally connect to MYSQL, we specify **localhost**. We leave out the authentication_plugins so that MYSQL will use the **caching_sha2_password** as default.

2.1.2 Granting user permissions

Now we want to give our new user to have all privileges on our application's database. The syntax for granting a user permissions is the follow.

```
GRANT ALL PRIVILEGES ON uwc_2.0 TO 'uwcmanager'@'localhost';
```

The option we chose is ALL PRIVILEDGES which granted our user all the permissions to working our uwc_2.0 database.
Now we have created a new user 'uwcmanager' as well as granted all the permission needded for manipulating our database. To check the final result, use the following command to show all the granted permissions of our user.

```
SHOW GRANTS FOR 'uwcmanager'@'localhost';
```

```
MySQL  localhost:33060+ ssl  SQL > SHOW GRANTS FOR 'uwcmanager'@'localhost';
+-----------------------------------------------------------------+
| Grants for uwcmanager@localhost                                 |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO `uwcmanager`@`localhost`                  |
| GRANT ALL PRIVILEGES ON `uwc_2`.`0` TO `uwcmanager`@`localhost` |
+-----------------------------------------------------------------+
2 rows in set (0.0137 sec)
MySQL  localhost:33060+ ssl  SQL >
```

Figure 10: Show granted privileges

2.2 App Features

Our application is built as a website with a remote server connected to our database that allow the back officer to accesss and manage their employees through any web browser on their . The app include some important features.

2.2.1 Authentication

Before accessing any information, Back officers are required to login by providing their account credentials including username and password. They are then delivered to the main page if and only if their credentials are corrected. Under the hood, the password is hashed before storing to database so that preventing many security issues.

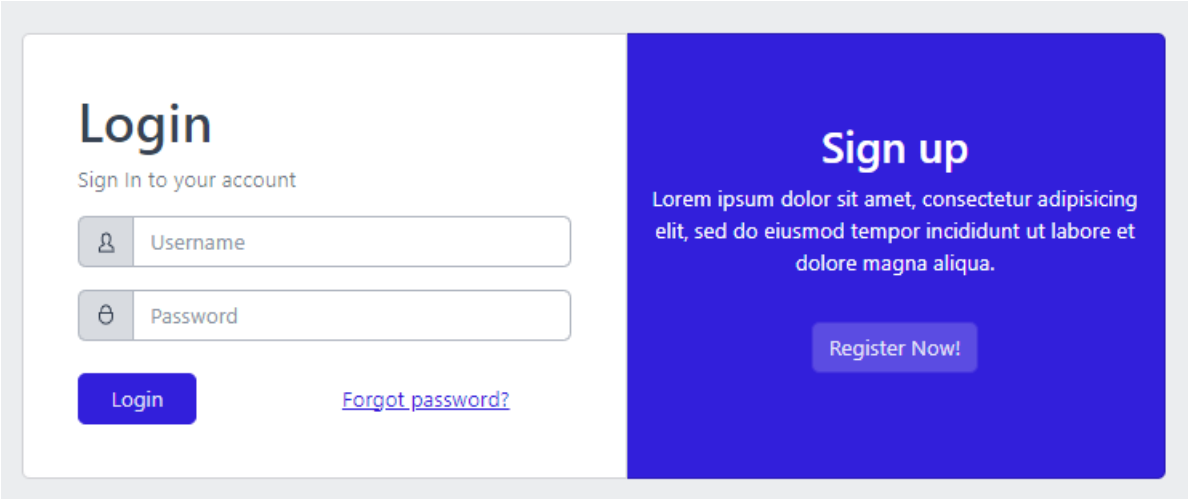


Figure 11: Login page

2.2.2 Employee Management

After authenticated by the login system, back officers are granted all permissions to manage and work with their own staffs.

On the user view, the browser display all employees who work under the current user (aka back officer) management. User can update, assign task or delete the employee user information as well as creating new one.

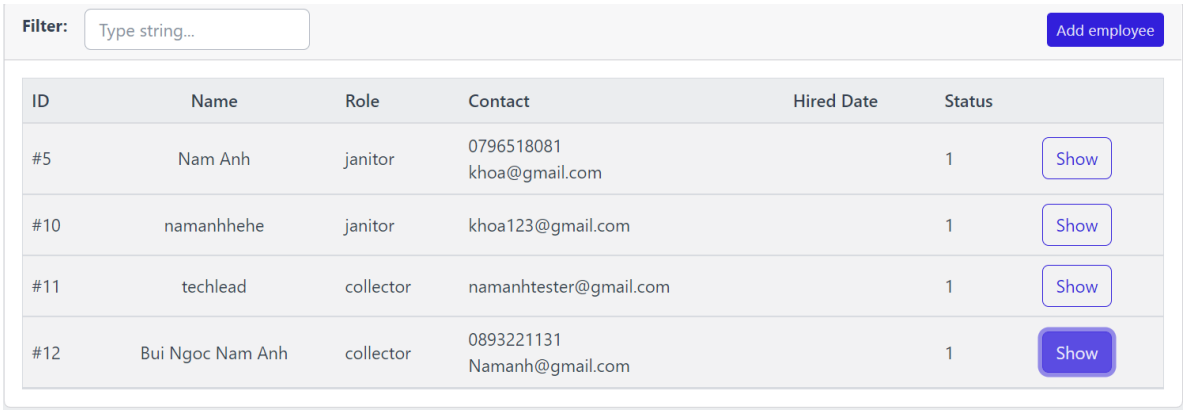


Figure 12: Employee dashboard

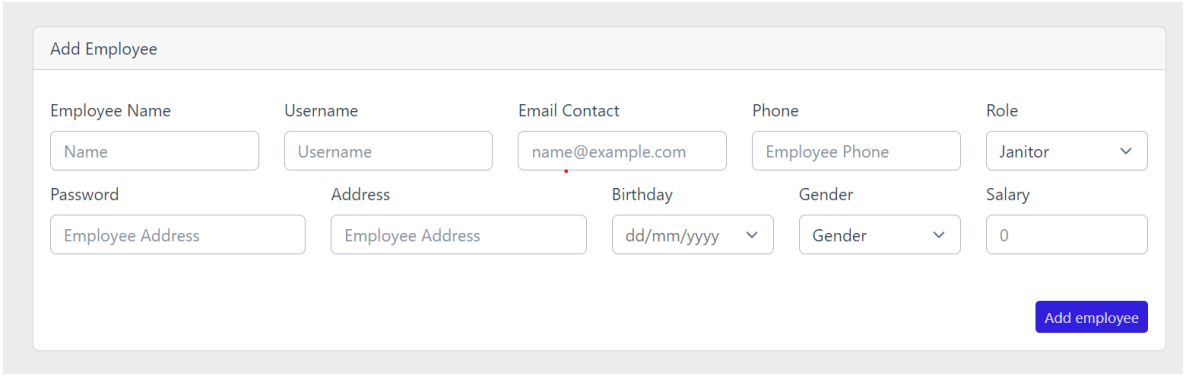
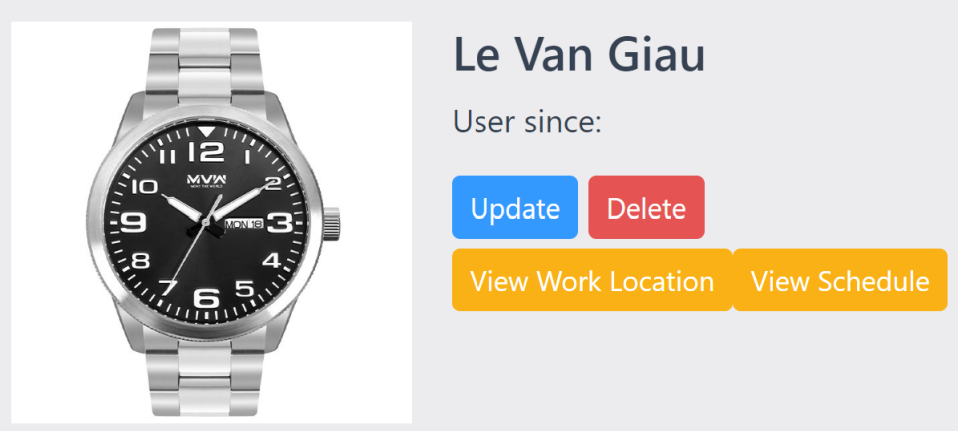


Figure 13: Employee update-add

Options are presented to navigate between profiles and tasks, as shown in the following figure.



The View Work Location option (as well as Map) leads to the Route/Map management modules.

2.2.3 Route Map

This feature is designed mainly for working with collectors. It provides a map that displays the locations of MCPs and their paths. Back officers can update or create a new route by picking a list of MCPs on the map. Behind the scene, algorithms are used to find the best convenient route path through picked MCPs and store them in the database.

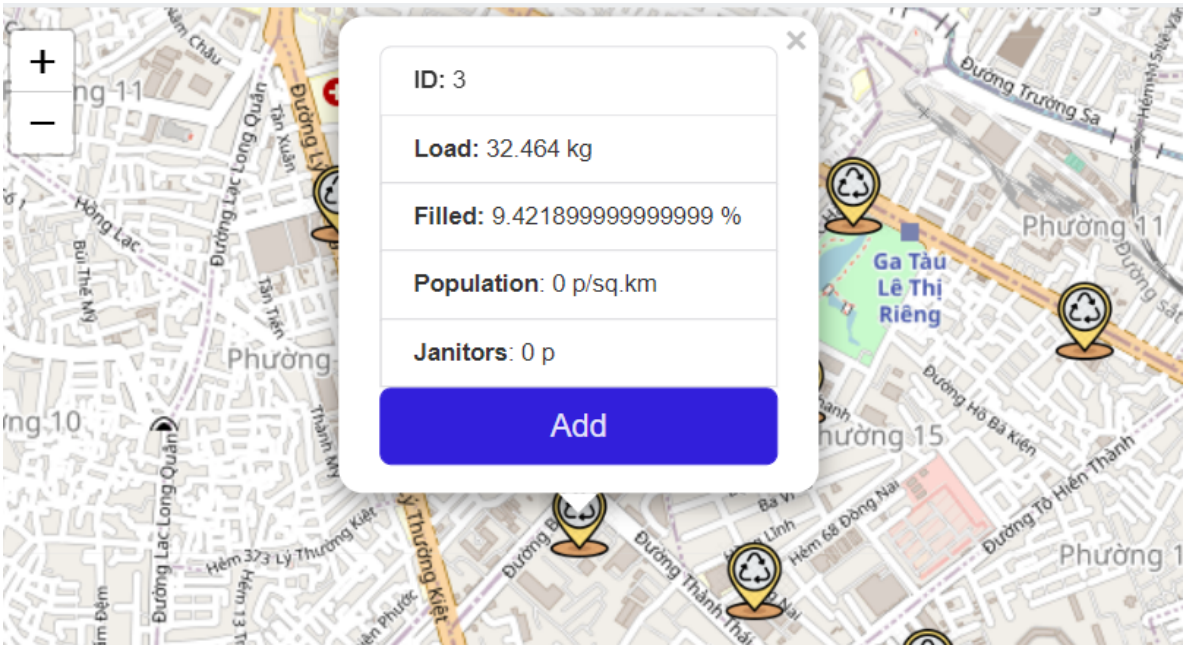


Figure 14: Picking an MCP

Below is an example display of a route from the application.

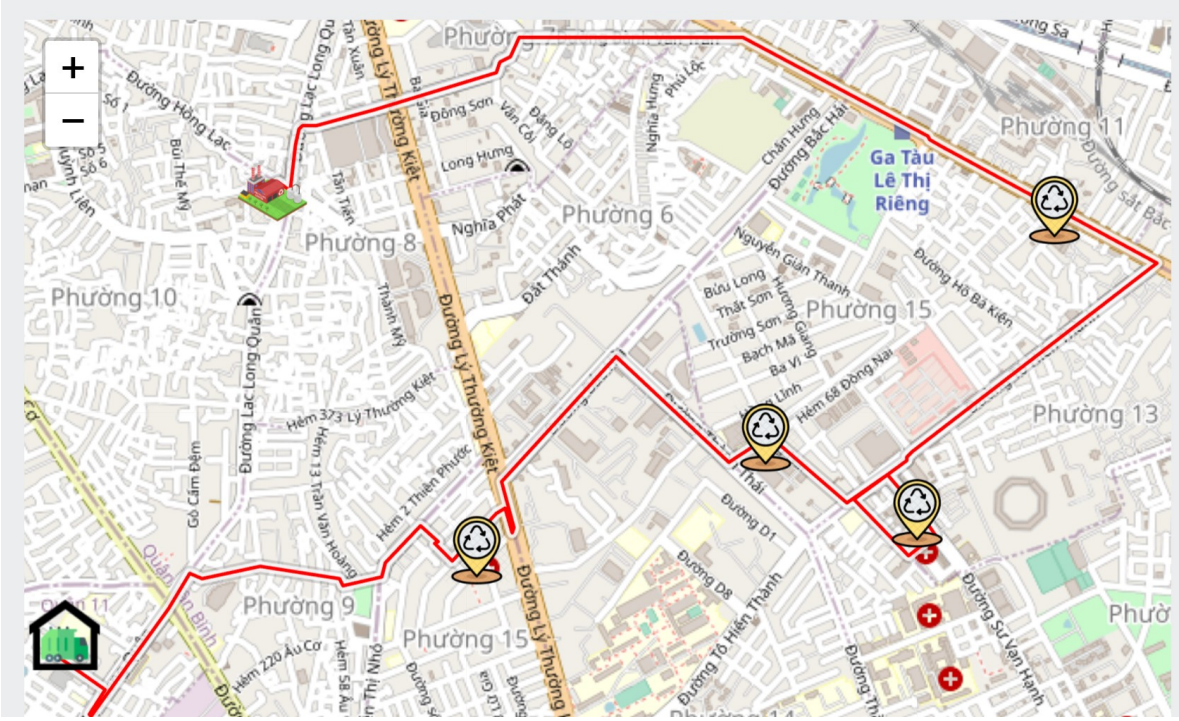
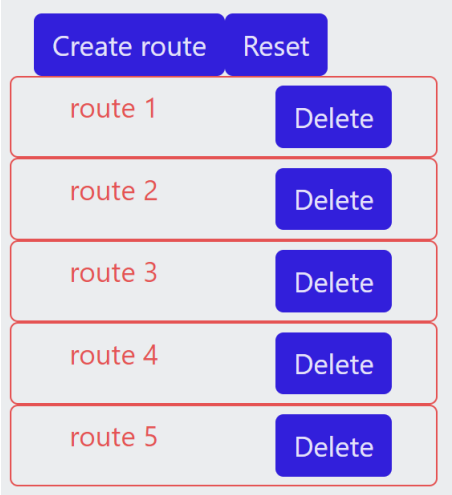
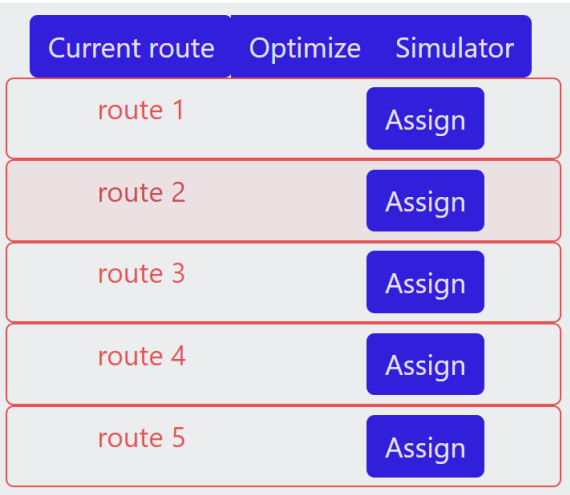


Figure 15: Route layout

It has all options stated in our specifications in the previous sections despite being aesthetically unsatisfactory.



Map options



Route options

We even add a simulator for realtime waste management.

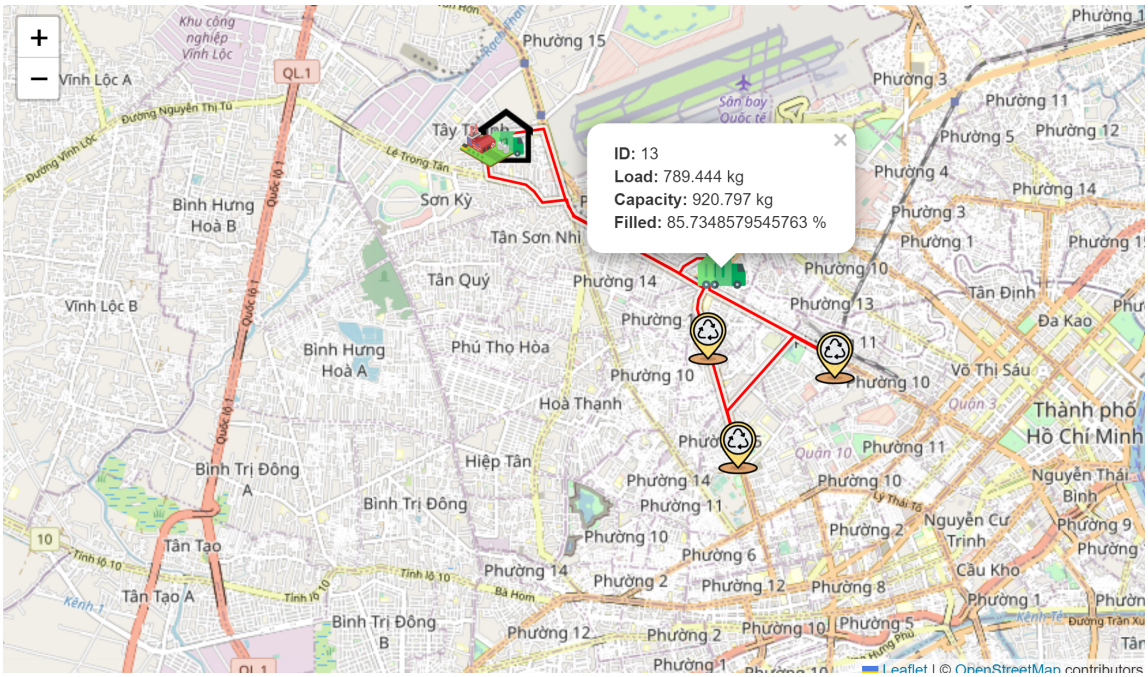


Figure 17: Simulator

Appendix A: Data Tables

id	is_vehicle	longtitude	latitude	load	capacity
1	0	106.6575010	10.7730450	86.297	391.228
2	0	106.6537450	10.7866540	12.570	371.095
3	0	106.6588210	10.7800860	75.549	324.445
4	0	106.6634210	10.7758280	4.558	336.053
5	0	106.6599600	10.7647150	24.796	324.209
6	0	106.6625890	10.7787420	60.648	363.276
7	0	106.6662570	10.7769350	82.349	308.439
8	0	106.6634590	10.7828160	50.427	366.688
9	0	106.6696090	10.7842080	10.733	307.188
10	0	106.6646900	10.7868000	97.142	374.485
11	0	106.6579610	10.7832080	81.548	364.737
12	0	106.6533470	10.7708700	0.988	367.820
13	1	106.6257860	10.8106760	0.000	977.216
14	1	106.6257860	10.8106760	0.000	977.677
15	1	106.6257860	10.8106760	0.000	956.736
16	1	106.6257860	10.8106760	0.000	150.648
17	1	106.6257860	10.8106760	0.000	183.034
18	1	106.6257860	10.8106760	0.000	163.225
NULL	NULL	NULL	NULL	NULL	NULL

asset table

user_id	employee_count	route_count	vehicle_count	MCP_count
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
NULL	NULL	NULL	NULL	NULL

back officer table

employee_id	route_id
14	NULL
15	NULL
7	1
11	1
9	2
NULL	NULL

collector table

mcp_start_date	work_radius	employee_id	mcp_id
2022-12-08	336.347	6	1
2022-12-08	481.982	8	2
2022-12-08	477.188	10	3
2022-12-08	NULL	12	NULL
2022-12-08	NULL	13	NULL
NULL	NULL	NULL	NULL

janitor table

order	mcp_id	route_id
1	1	1
2	3	1
3	4	1
4	8	1
1	2	2
2	6	2
3	7	2
4	9	2
1	1	3
2	9	3
3	2	3
4	4	3
1	1	4
2	3	4
3	4	4
4	2	4
1	2	5
2	8	5
3	4	5
4	1	5
NULL	NULL	NULL

contain_mcp table

asset_id	backofficer_id
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1

asset_supervisor table

user_id	manager_id	vehicle_id	is_working	is_collector	start_date	salary
6	5	16	1	0	NULL	6000000
7	5	13	0	1	NULL	7000000
8	5	17	1	0	NULL	6000000
9	5	14	1	1	NULL	7000000
10	5	18	1	0	NULL	6000000
11	5	15	0	1	NULL	7000000
12	5	21	0	0	NULL	7000000
13	5	22	1	0	NULL	7000000
14	5	19	1	1	NULL	7000000
15	5	20	0	1	NULL	7000000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

employee table

asset_id	pop_density	janitor_count
1	58.351	0
2	10.660	0
3	57.347	0
4	39.955	0
5	27.993	0
6	20.661	0
7	57.089	0
8	49.296	0
9	2.245	0
10	48.600	0
11	47.424	0
12	21.681	0
NULL	NULL	NULL

mcp table

asset_id	type
13	truck
14	truck
15	truck
16	trolley
17	trolley
18	trolley
19	truck
20	truck
21	trolley
22	trolley
NULL	NULL

vehicle table

id	last_update	created_at	distance	manager_id
1	2022-12-08 18:55:22	2022-12-08 18:55:22	NULL	5
2	2022-12-08 18:55:22	2022-12-08 18:55:22	NULL	5
3	2022-12-08 18:55:22	2022-12-08 18:55:22	NULL	5
4	2022-12-08 18:55:22	2022-12-08 18:55:22	NULL	5
5	2022-12-08 18:55:22	2022-12-08 18:55:22	NULL	5
NULL	NULL	NULL	NULL	NULL

route table

id	username	passw	name	is_bac	is_acti	date_joine	addı	birtl	ger	phone	email	last_login
1	VitNgu...	b5c...	Le Van ...	1	0	2022-1...	NULL	NULL	m...	NULL	duyvt...	NULL
2	Toinuoi...	799...	Pham Mi...	1	0	2022-1...	NULL	NULL	m...	NULL	minhc...	NULL
3	phucnh...	29d...	Nguyen...	1	0	2022-1...	NULL	NULL	m...	NULL	phucn...	NULL
4	TienMin...	923...	Dang Ti...	1	0	2022-1...	NULL	NULL	m...	NULL	tienmi...	NULL
5	ChienH...	141...	Ho Tha...	1	0	2022-1...	NULL	NULL	m...	NULL	BukBu...	NULL
6	trangb...	e7f...	Doan T...	0	1	2022-1...	NULL	NULL	f...	012...	trangd...	NULL
7	giaunh...	298...	Le Van ...	0	0	2022-1...	NULL	NULL	m...	098...	ngheo...	NULL
8	Thythy...	8ee...	Khuc Thy	0	1	2022-1...	NULL	NULL	f...	098...	thythy...	NULL
9	VanAvt...	15e...	Nguyen...	0	1	2022-1...	NULL	NULL	m...	097...	Bomav...	NULL
10	BBBboy...	132...	Nguyen...	0	1	2022-1...	NULL	NULL	m...	092...	Brherj...	NULL
11	123Csu...	3a2...	Nguyen...	0	0	2022-1...	NULL	NULL	m...	012...	Hiehei...	NULL
12	huyhoa...	b91...	Nguyen...	0	0	2022-1...	NULL	NULL	m...	014...	huyho...	NULL
13	quocba...	d37...	Tran Qu...	0	1	2022-1...	NULL	NULL	m...	042...	quocb...	NULL
14	phivu123	7b7...	Nguyen...	0	1	2022-1...	NULL	NULL	m...	076...	phivun...	NULL
15	huyan123	535...	Nguyen...	0	0	2022-1...	NULL	NULL	m...	045...	anhuy...	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

user table

start	end	weekday	employee_id	id
00:00:00	02:00:00	Mon	6	1
05:00:00	07:00:00	Sat	7	2
09:00:00	11:30:00	Thur	8	3
10:00:00	12:00:00	Tue	9	4
09:00:00	11:10:00	Fri	10	5
00:00:00	02:00:00	Wed	11	6
NULL	NULL	NULL	NULL	NULL

work time table

Appendix B: Source Code

The url below leads to our team project repo.

https://github.com/minhtrungcm181/Software_Engineering_UWC2.0_Project

To run the code, first clone the repo to a local site. Set up the BackEnd according to the README guideline in the `Controller` folder. Set up the FrontEnd side by installing npm and run `npm install` then `npm start` in the `View` folder.