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

Software & Engineering
Assignment Report

Urban Waste Collection Aid
UWC 2.0

Team Fail The Bloody Course

Lecturer: Prof. Truong Tuan Anh

Team members: Le Nguyen Hoang Nhan – 2052625
Trinh Minh Trung – 1852825
Le Khanh Duy – 2052003
Le Duc Thuan – 2053467
Dang Quoc Thinh – 1852761

Ho Chi Minh City, April 26, 2023

Contents

1 Requirement Elicitation

1.1 Business Context

Urban waste management is one of several significant problems faced by many countries in the world and thus considered one of the important points to be improved in Sustainable Development Goal (SDG) 11: sustainable cities and communities and SDG 6: clean water and sanitation. Particular attention is given to developing countries that continue to prioritize development and economic growth. In urban contexts, solid waste management is costly and ineffective. Improvement of waste collection and management is emphasized by governments and organizations for positive impacts on cities, societies and environments.

Waste collection is often designated to an organization that provides professional waste management services. A typical waste collection process involves (1) back officers, who operate a central system, (2) collectors who drive trucks to transfer waste, and (3) janitors who manually collect using trollers. Back officers have a general view of all vehicles and MCPs. In this context, an MCP is an intelligent major collection point which comprises several garbage containers, as illustrated by Figure 1.

Figure 1: A simple collection point

An MCP can regularly report its load back to the management system via its specialized hardware. Schedules and tasks were assigned among teams of janitors and collectors and coordinated by back officers. These assignments are often arranged on a weekly basis. Everyday, the back officers sent messages with information about collecting routes, work

areas and time to collectors and janitors. Collectors will start from a depot, pick up garbage from all janitors at some MCPs , and drive to the treatment plant (disposal facility). These MCPs that a collector drives through make up a route, which is included in tasks that are predetermined by some back officer. The routing scheme is demonstrated in Figure 2. We make an assumption that there is only one treatment plant and one depot. When the collector goes on work, the system optimizes his/her predetermined route by dropping out the assigned MCPs with load less than 15% their capacity. The collector travels the shortest paths between the MCPs. This optimization is also performed by the system.

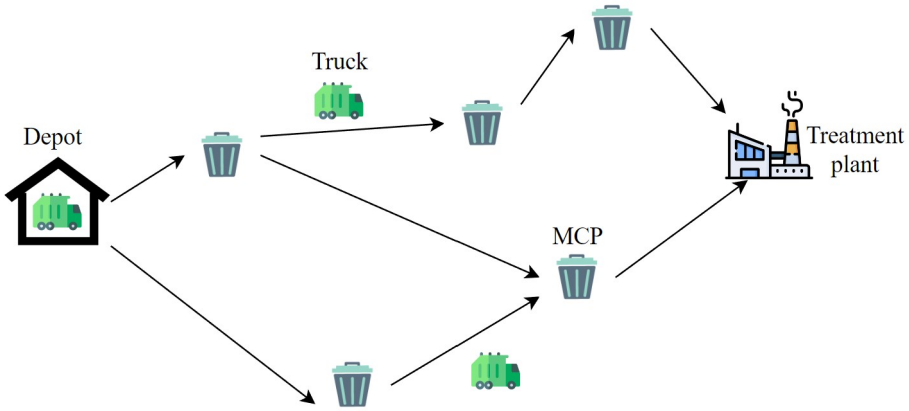


Figure 2: A simplified routing scheme

Janitors are hired based on the MCPs' locations and population density of the surrounding areas. The more busy an MCP is, the more janitors work on it, with each of them collecting garbage within a 500m-radius of their home. Customers who demand the waste management service can sign up so that their location is directed to the closest on-demand janitor. For simplicity, we will not model these customers in our upcoming data model.

To accompany all these business requirements, we need a specialized application and a corresponding database. While a piece of software UWC 1.0 is assumed to have already existed and aided the business, there are a lot of constraints to account for; therefore the need for a better system arises in form of detailed Front End and Back End Architecture. The following sections captures the necessary steps to construct them. Before that, we summarize the system stakeholders and their needs into Table 1.

Table 1: System Stakeholders and Needs

Stakeholders	Potential Problems	Needs
Back officers	- Communication delay is significant - Management module is labor intensive - Data query is slow - Insufficient amount of features and statistics	- Better communication system - More user-friendly interface - Optimal query - More features supporting work-flow
Employees	- Assigned tasks are unoptimized - Task delivery is slow - Lacking displayed features	- Real-time optimized workload - Faster real-time notification - More user-friendly interface
The community	- Customer service is lackluster - Area sanitation barely improves	- Better customer service - Higher system performance

1.2 System Requirements

1.2.1 Functional Requirements

Interaction within the system revolves around Back officers, Employees (Janitors and Collectors), and MCPs. So naturally, all of our system functionality can be categorized into three groups as follows.

Back officers are enabled to:

- View/Update employees' personal profile and schedule.
- Register new employees and remove those that hopped out.
- Assign areas to janitors and routes to collectors.
- View/Update MCPs and vehicles technical details, including their weight, load, capacity, fuel consumptions, etc.
- Register new assets (MCPs and vehicles) and remove when needed.
- Assign vehicles to employees, that is, trucks to collectors and trolleys to janitors.

Janitors and collectors are enabled to:

- View/Update their own personal profiles.
- View their own schedules and tasks.
- Check in/out task.
- Be notified about fully-loaded MCPs.
- Communicate with their colleagues and back officers.
- Update their status and locations when in work.

MCPs are equipped with hardware to:

- Update their loads.
- Notify system users when fully loaded.

1.2.2 Non-functional Requirements

- *Scalability*. The system should be able to handle real-time data from at least 1000 MCPs at the moment and 10.000 MCPs in five years.
- *Reliability*. Communication channel (messages, notification) works with at most 1 second delay.

  Query and update to the database work with at most 5 minute delay.
- *Availability*. Information/status of janitors and collectors, vehicles, MCPs must be updated every 15 mins with the availability of at least 95% of their operating time.

  The system must be operational through out normal working hours, from 8:30 am to 17:30 pm.
- *Supportability*. The system UI supports desktop view for back officers and mobile view for employees.

  UWC 2.0 system interfaces should be in Vietnamese, with an opportunity to switch to English in the future.

Figure 3: System Use Case Diagram

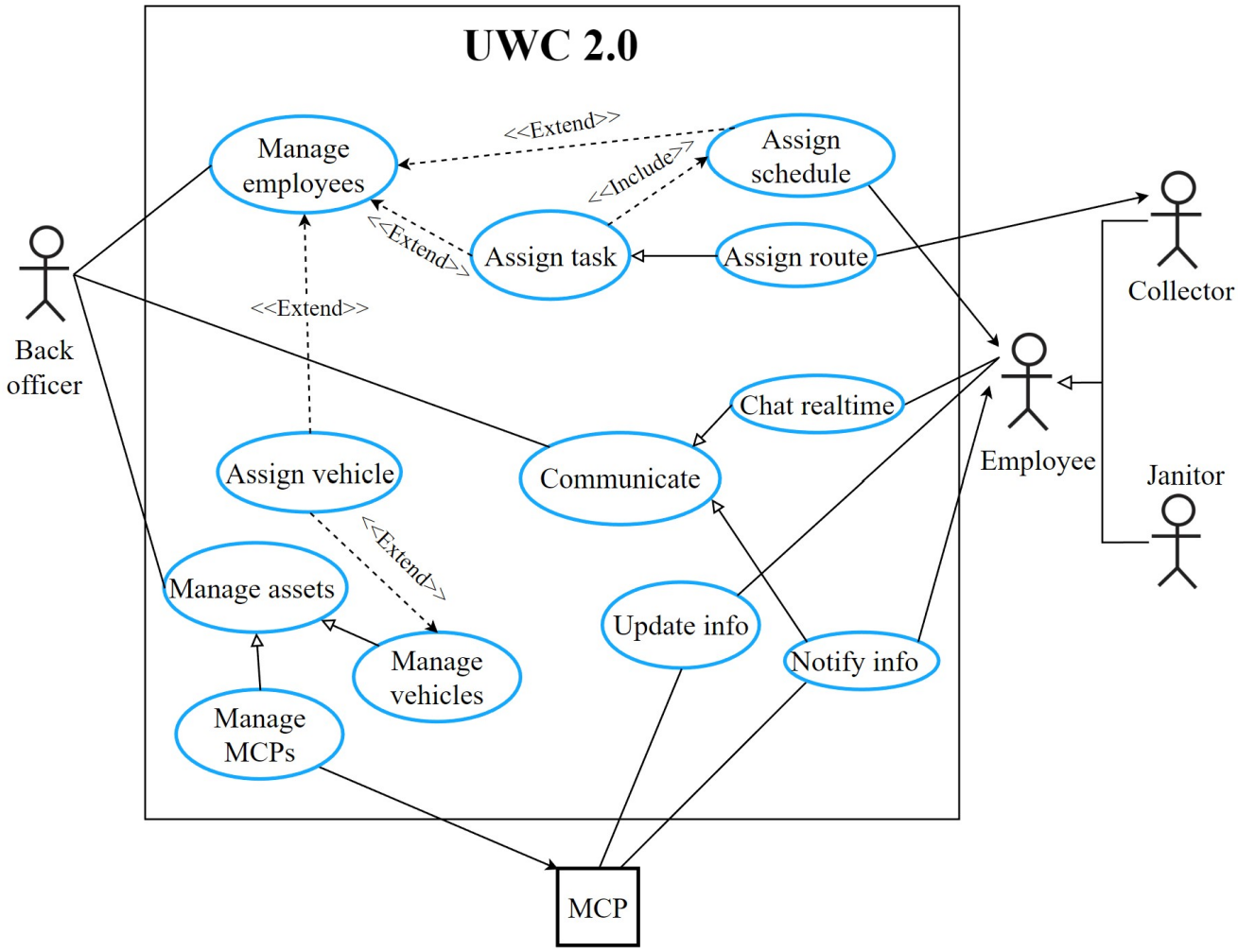
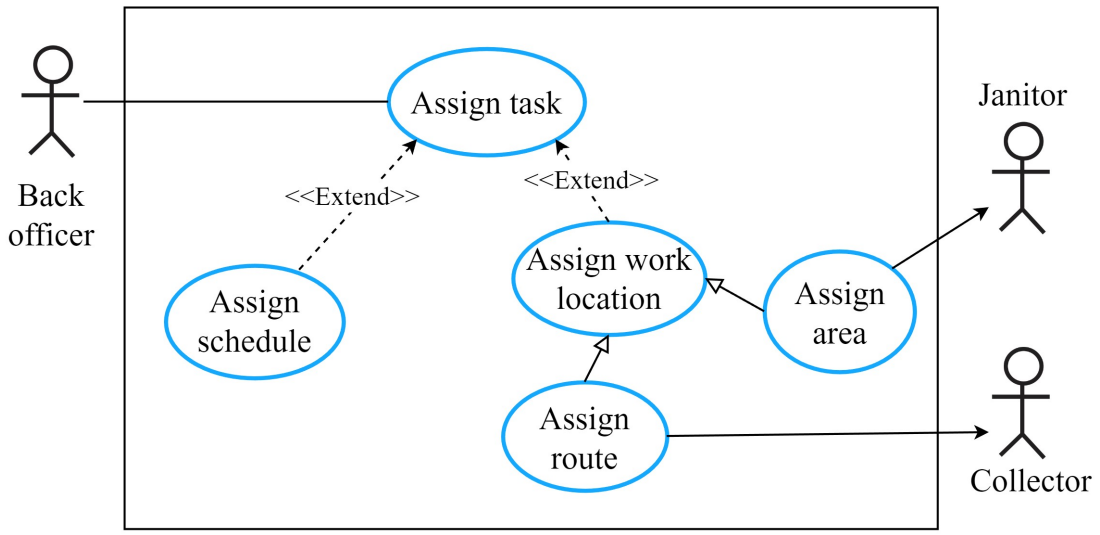


Figure 3 illustrates a Use Case Diagram for our system. The general view includes three main use cases - `Manage employees`, `Manage assets`, and `Communicate` - which are further specified into sub use cases. The main actors of the system are back officers, janitors and collectors (which specialize employees), and MCPs. Back officers have access to all use cases while the other are mostly on the receiving end.

1.3 Task Assignment Module

Figure 4: Task Assignment Use Case



Use case name	**Assign Task**
Description	The back officer specifies which employee to assign the task. If the chosen is Janitor, the Back Officer needs to assign MCP where the Janitor works and the time. Otherwise, If the chosen is a collector, the Back Officer needs to assign an optimized route
Actor	Back officers
Preconditions	The back officer is logged in The back officer has chosen an employee from the dashboard
Postconditions	The back officer assigns a new schedule, OR The back officer assigns a new work location, either route or area

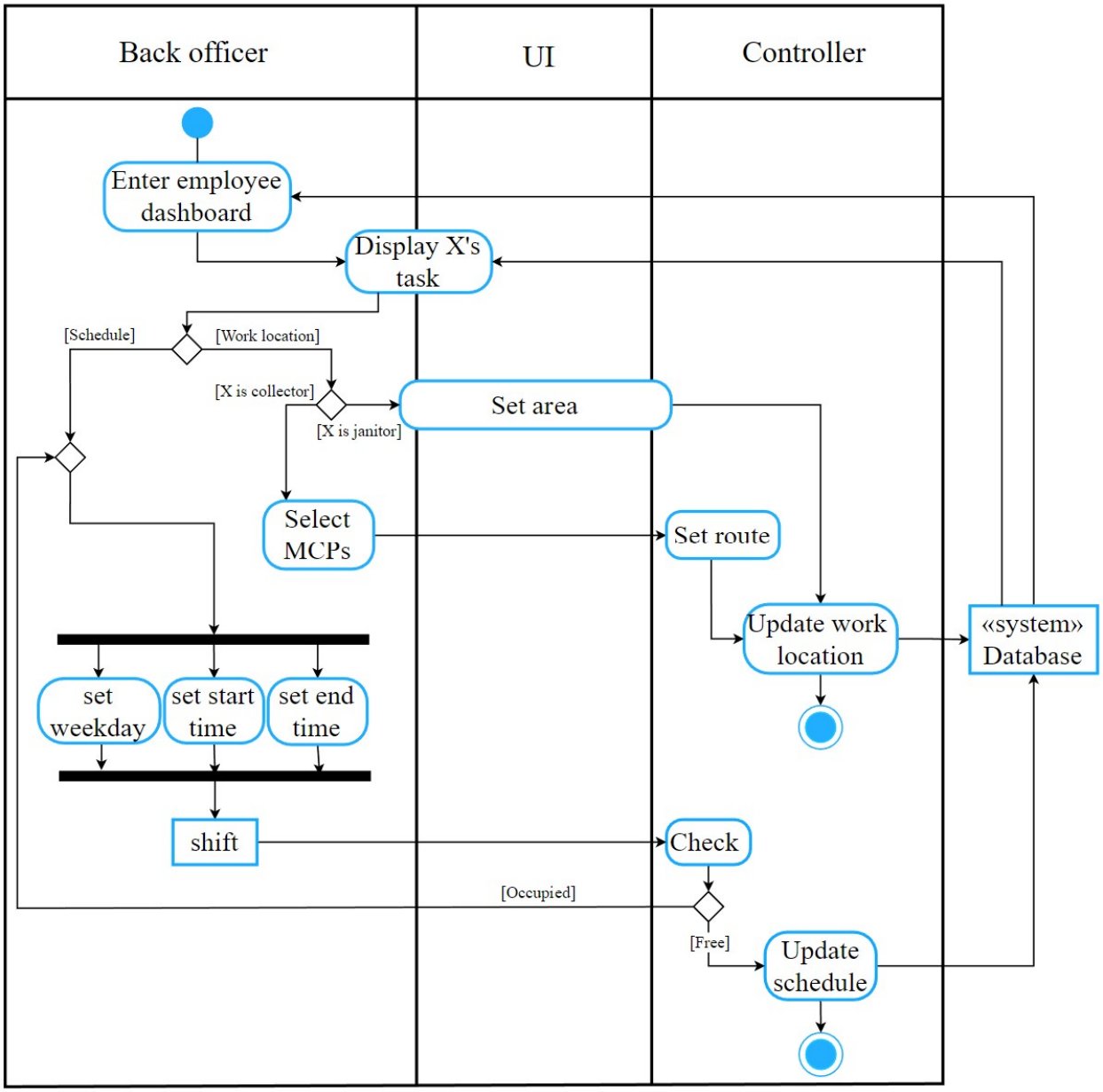
Normal flow	1. The system displays options, either `Schedule` or `Work location` 2. The back officer chooses either 3. If the back officer chooses Schedule 3.1. The back officer `assign schedule` 3.2. The system updates the schedule to the database 4. If the back officer chooses `Work location` 4.1. If the employee is a janitor 4.1.1. The back officer `assign area` 4.1.2. The system updates the area to the database 4.2. If the employee is a collector 4.2.1. The back officer `assign route` 4.2.2. The system updates the area to the database
Exceptions	None
Alternative flows	None

2 System Modeling

2.1 Task Assignment from Business Perspective

Detailed in Figure 5 is an activity diagram for the `Task Assignment` module. It is shown using a business perspective, that is, via interaction between a back officer and the system. The activity starts at the emloyee dashboard from the back officer view, where they choose to further down an employee profile, called X. They will then face with multiple options to advance, of which there are `Schedule` and `Work location` tabs. If the back officer goes with the former, he could then directly alter X's calendar, which comprises multiple weekly work shifts. For the latter, they will enter a map resolution of X's collecting area or route, depending on X's role as an employee being either a janitor or a collector. The back officer can adjust the collecting location to his will and submit new change to the system database, which also involves in display and query retrieval.

Figure 5: Task Assignment Activity Diagram



2.2 Route Planning Concept

The activity diagram sketched above only depicts partially what the *route planning module*, known simply as `Set route` in Figure 5, looks like from the perspective of a back officer. It includes route assignment from the back officer and optimization from the system without involvement from the collector, which is insufficient. In this section, we delve into the complete details of the module.

General Description

A route is a set of MCPs with paths between some of them. Each collector follows one route, started from the depot, to collect waste from the MCPs and transfer them to the treatment plant. This has been illustrated by Figure 2.

A back officer decides the set of MCPs that a collector is allowed to go through and store that as a route in the database. Also, we separate the notion of route from that of collector, so that a route can be created and stored first without a collector assigned to it. On duty, the collector queries the layout of those MCPs and optionally requires the system to re-route so that the following conditions are satisfied:

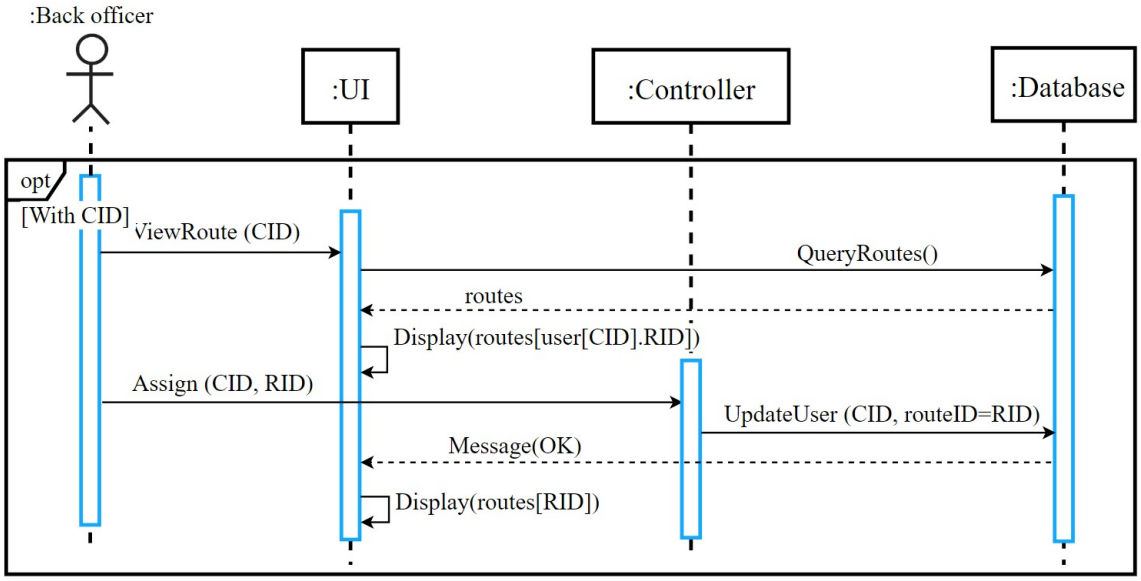
- Only MCPs with load more than 15% their capacity are chosen.
- The total load of the displayed MCPs doesn't exceed the capacity of the truck.
- The paths between connected MCPs are the shortest.

This optimization is done based on the information presumably updated by MCPs every 15 minutes. The path between two consecutive MCPs in the route is determined by the system to optimize fuel consumption, distance, etc. Dropping out some MCPs does not violate the precept the back officer made since their planned route is used as a premise and no new MCP is added to it. This mechanism softens the rigidity of predetermining a fixed route since the back officer does not know beforehand the precise load of every MCP each day of the upcoming week.

Sequence Diagram

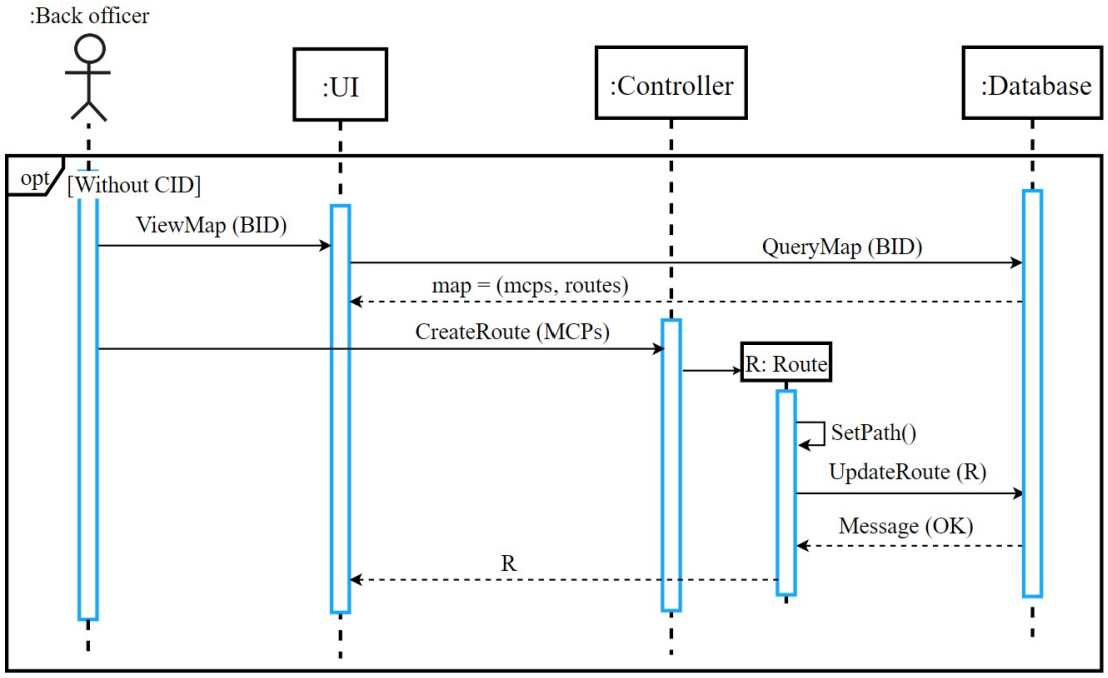
Below is the interaction between a back officer and the system when performing route planning, with a Collector CID that is. In this case, the back officer accesses the collector profile and manages their route by assigning an already existing route.

Figure 6: Back Planning with CID



Without a Collector CID, the back officer may choose to create and edit routes independently. This is done via the Map module, where back officers manage all MCPs and routes data. The below diagram illustrates planning using Map.

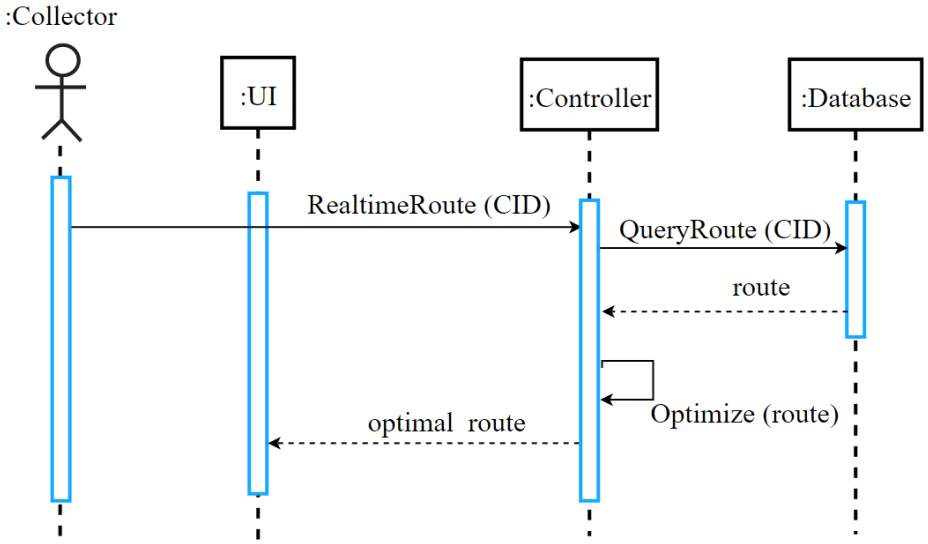
Figure 7: Back Planning with Map



Note how the back officer only picks the MCPs and shoots them back to the controller, where the `SetPath()` function is triggered. Specifically, this function takes care of the orders of MCPs within routes by solving the famous Travelling Salesman Problem to determine the shortest closed path that starts at the depot, traverses all of the assigned MCPs, and ends at the treatment plant.

Then there is the realtime re-routing requested by collectors:

Figure 8: Realtime Re-routing



2.3 Task Assignment Class Diagram

Below, Figure 9 is a diagram depicting complete interaction between the system classes, including entities, UI and controller, within the task assignment module. Generally, entity classes make up an abstract layer that updates data to the UI, which in turn sends user events to controllers to handle, which is enabled to make change to the data. Task in this context refers to both schedule and work location, or route/area depending on the employee role. We separate the assignment of either objects and divide the UI as well as controller into sub components that handle schedule, route and area without interplay. This choice of design was reflected in our activity diagram, Figure 5, as back officers have options to display between schedule and work location before assigning them anew.

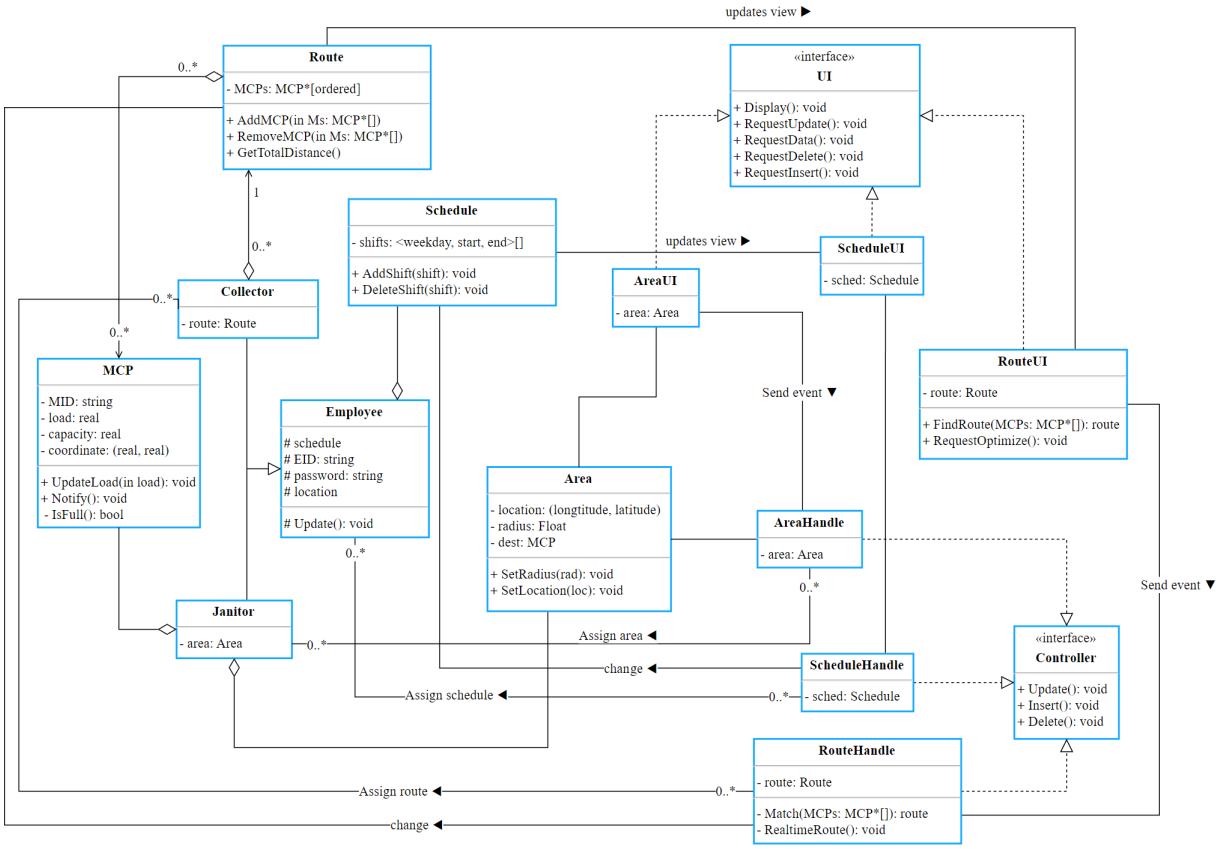


Figure 9: Task Assignment Class Diagram

3 Architecture Design

3.1 Architectural Approach

In this project, our group choose to design the UWC2.0 by using MVC pattern, as shown in Figure 10. There,

- View or User Interface takes responsibility for user to make contact with system. The contact from user becomes request and is sent to the controller. Otherwise, View takes another responsibility to render the content from Controller.
- Controller plays the most important role in the system. It will receive request passed from View, retrieve the right data, and return it back to the view. The major task here is that Controller needs classify requests and determine what to return. It is our responsibility to make Controller work properly.
- Model specifies how data is structured, and updates the view. More importantly, it

will be the main part that requests and receives directly from Database. Model might encompass business logic, data abstraction, etc. and be used to design Database first.

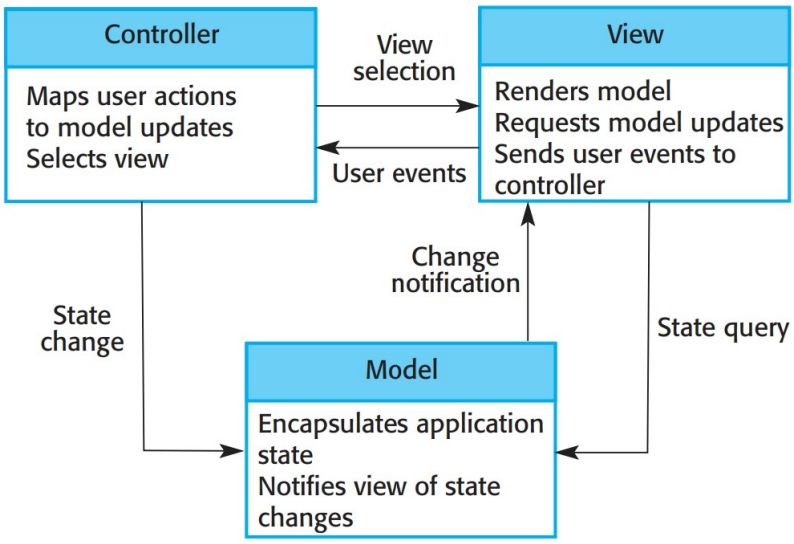


Figure 10: The MVC Architecture

The reason that our group setup this architecture is that:

- Easy to understand and implement. This architecture divides functionality clearly into parts, each has its own mission.
- Because of division in architecture, the system is easy to debug and maintain. Coding will have reusability, and avoiding hardcode is one of advantage.
- This architect the most common.

Modules

Name	Communicate
Description	The module transfers notifications between employees, back officers, and MCPs. It can also conduct chat sessions between system users.
Functions	`CreateSession` (UID, UID): bool `Notify` (UID, event_type): bool `SendMessage` (UID, message): bool

Name	Access Control
Description	The module performs user entry authentication by matching correct username and authenticate password
Functions	`Match` (username): employee `Authenticate` (password): bool `Login` (username, password): bool `SignOut` (UID): bool

Name	Vehicle Management
Description	The module manages vehicle
Functions	`CreateVehicle` (ID, capacity, load, location, type): vehicle `Update` (vehicle): bool `Edit` (vehicle): bool `Delete` (vehicle): bool `QueryVehicles` (): vehicle[ ]

Name	Employee Management
Description	The module manages employee information
Functions	`CreateEmployee` (ID, name, username, password, age, sex, salary) : employee `Update` (employee): bool `Edit` (employee): bool `Delete` (employee): bool `QueryEmployees` (): employee[ ]

Name	Map
Description	The module manages a grand map containing all MCPs, the depot, and the treatment plant of the company. It can perform changes to both MCPs and Routes on the map.
Functions	`DisplayMap` (MCPs): void `CreateMCP` (ID, location, capacity, load = 0): MCP `DeleteMCP` (ID): bool `CreateRoute` (MCPs): route `EditRoute` (route): bool `DeleteRoute` (route): bool `UpdateRoute` (route): bool `MatchRoute` (MCPs): route

Name	Task Management
Description	The module manages the schedule and area/route of employees
Functions	`LayoutSchedule` (shifts): void `AddShift` (shift): bool `EditShift` (shift): bool `DeleteShift` (shift): bool `MatchRoute` (MCPs): route `CreateRoute` (MCPs): route `AssignRoute` (route, UID): bool `LayoutRoute` (route): void

3.2 Implementation Diagram

Illustrated in Figure 11 is the implementation perspective of the task assignment module, enclosed in an MVC architecture. The `view` corresponds directly to a back officer UI. It includes options to assign tasks to an employee. Specifically, if one enters an employee's profile, they can choose to view and edit either that person's `schedule` or `work location`, which is a `route` or `area` depending on their role. The back officer can otherwise choose the `map` tab and fiddle with routes stored in the system.

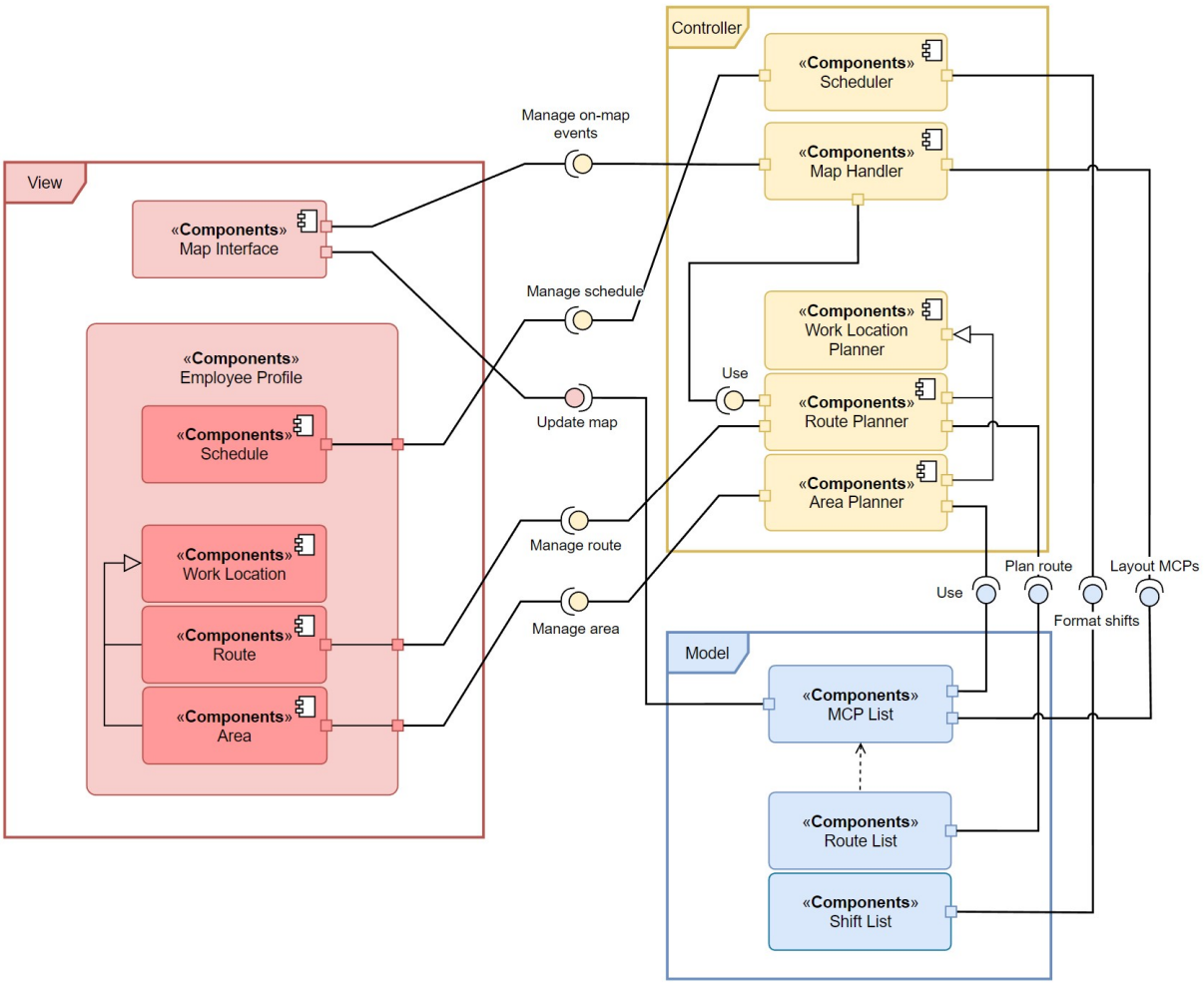


Figure 11: Task Assignment Implementation Diagram

In the diagram, components in `view` are handled by controller components, in which `Map Handler` is allowed to use `Route Planner` to aid route management without specifying any employee ID. It is designed so that the map interface can not mess with working areas of janitors. This is because each area is associated with a janitor location which is not displayed by the map. The controller performs its functionality using data provided by the database.

4 Implementation Sprint 1

4.1 Setting up version control system

We chose Git as our version control system and Github as a platform to store the respository online. This link goes to the online respository.

4.2 Update Respository

To see the whole timeline of the resposity, check the command `git log` on a Git Bash shell. The result is a long list of changes we have made to the respository on different members' computers. Therefore, only the main changes are reported here.

- Initialize the repository

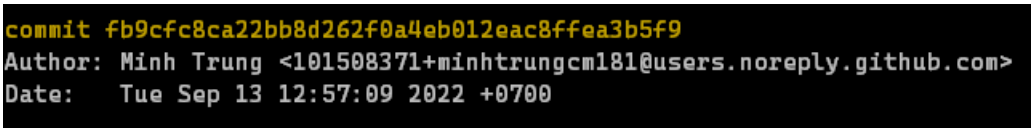


Figure 12: Init commit

- Add frontend code for the website

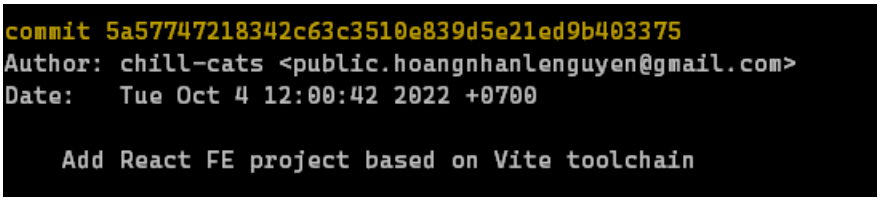


Figure 13: Init frontend code for website

- Reformat the respository description through `README.md` file.

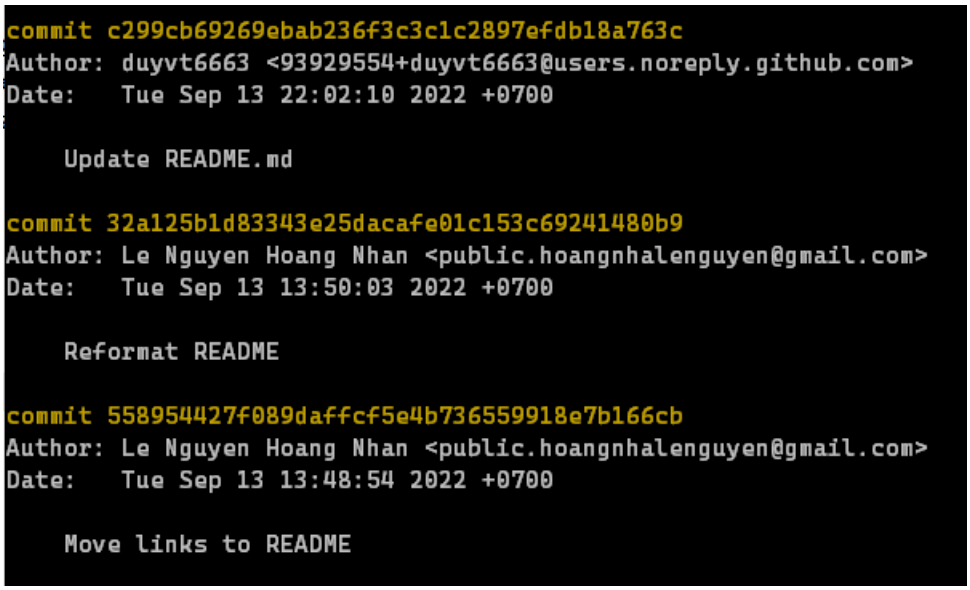


Figure 14: Update respository description

- Add system modeling as well as report files.

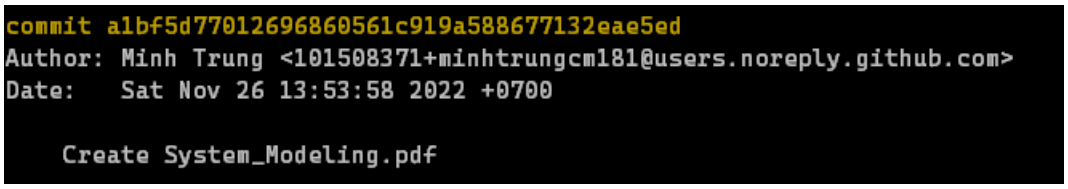


Figure 15: System modeling document

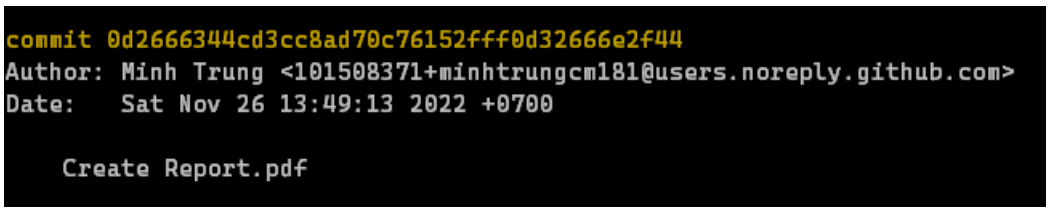


Figure 16: Project report document

- Add backend code for the website

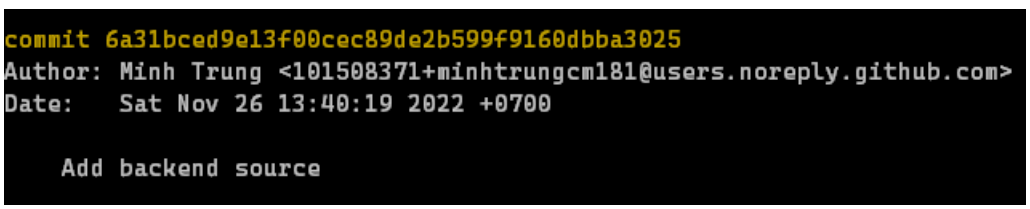


Figure 17: Add backend code for website

4.3 MVP1

Figma was used to construct a desktop-view central dashboard for Task Management for back-officers. There are two choices to work with Figma, eitheir on a website or on an application. Following this link to our wireframe design.

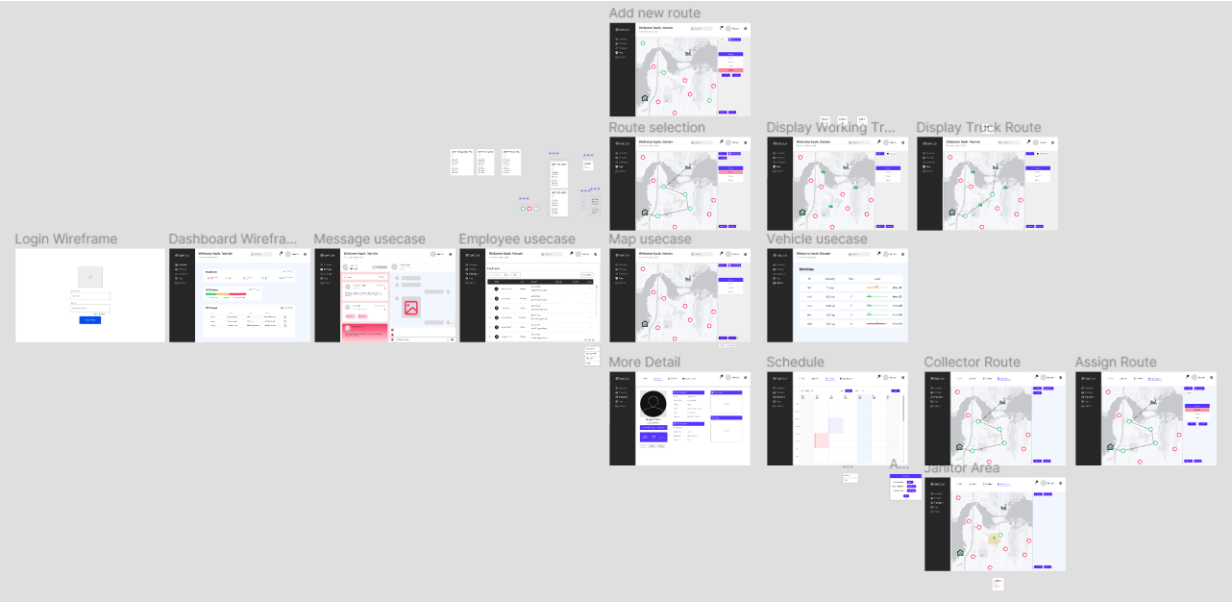


Figure 18: Wireframe design with Figma

4.3.1 Detail Wireframe Views

- Dashboard gives general statistics on the waste management operation.

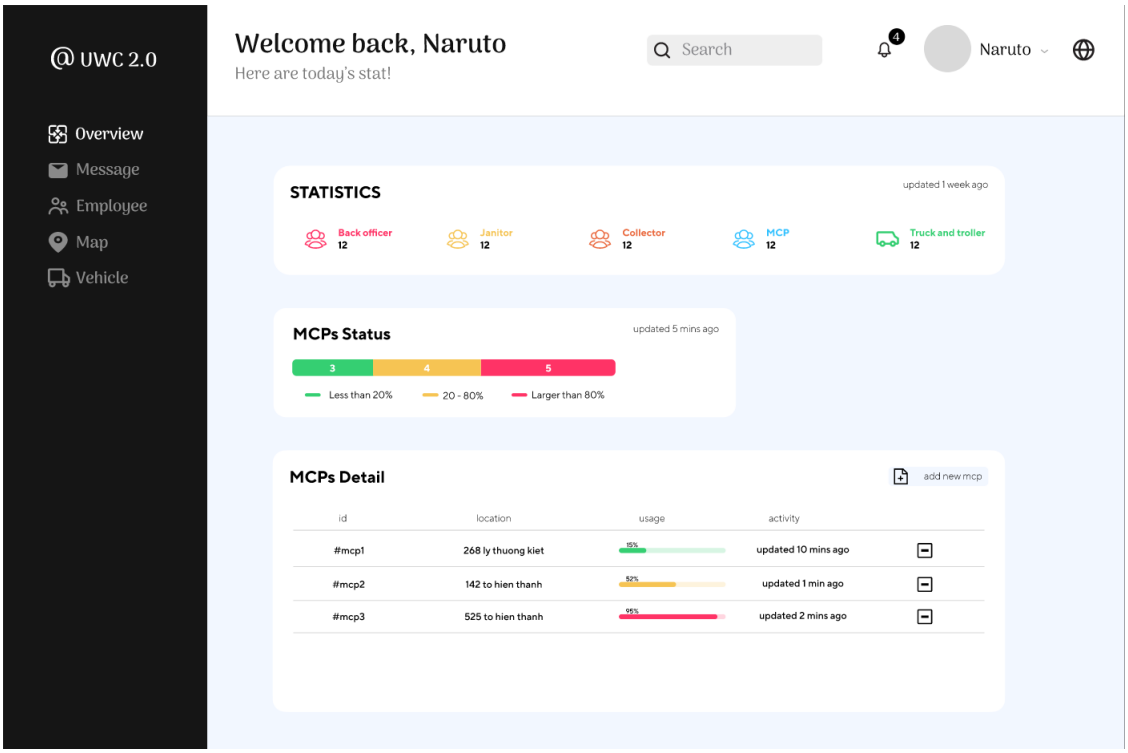


Figure 19: Dashboard view

- Messaging feature for communication between back-officiers, janitors and collectors.

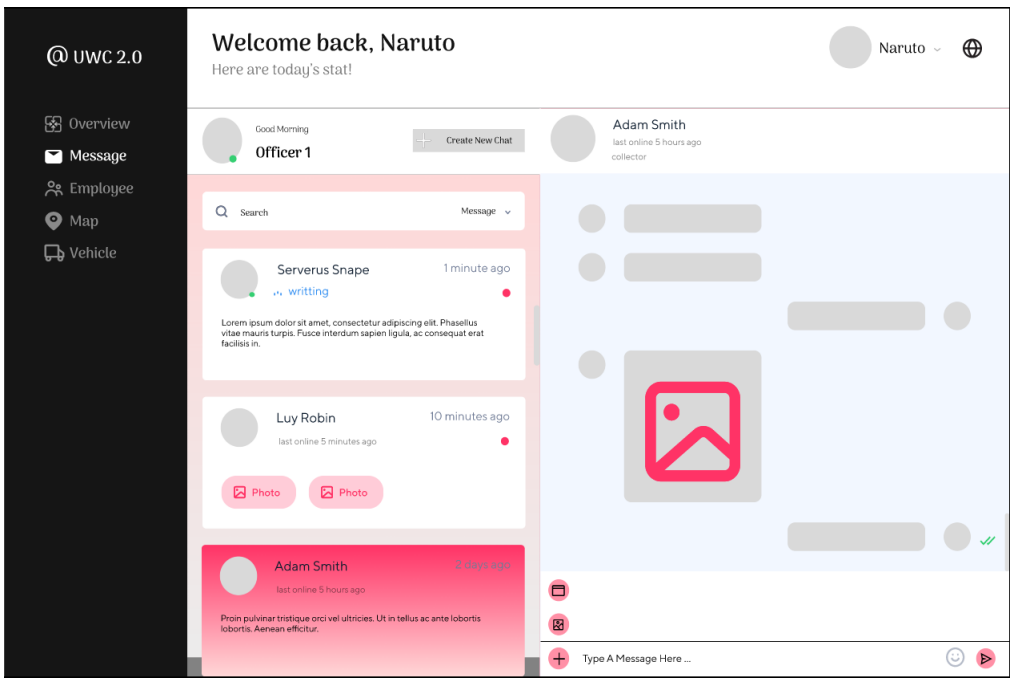


Figure 20: Messaging feature

- Employee management feature displays personal information and buttons that allow getting into a more detailed page, view his/her task on a calendar format and assigned working route on a live map.

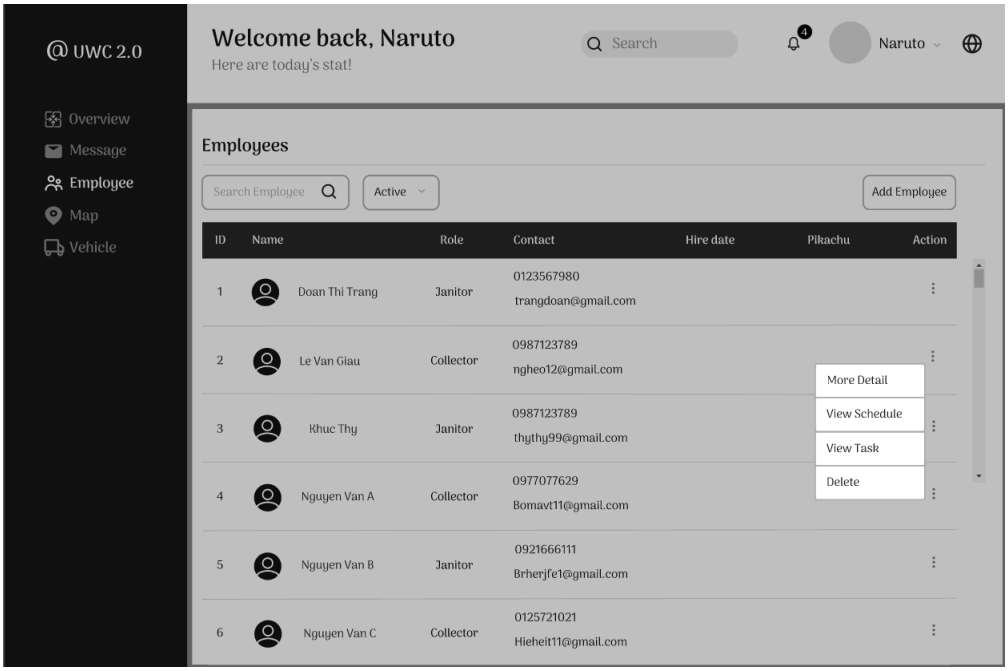


Figure 21: Employee management

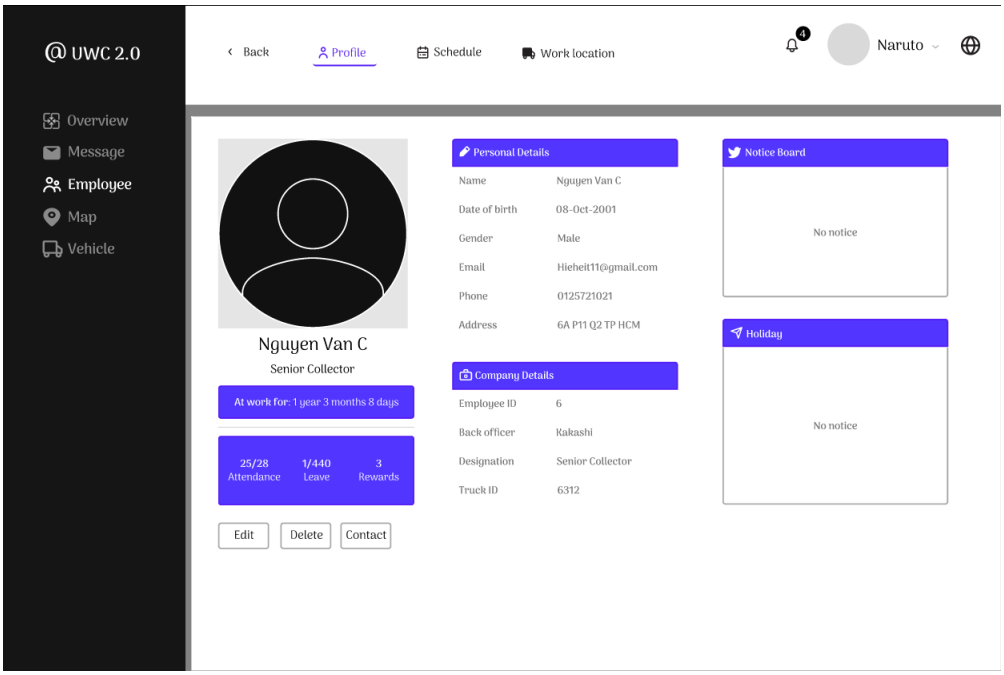


Figure 22: Employee details

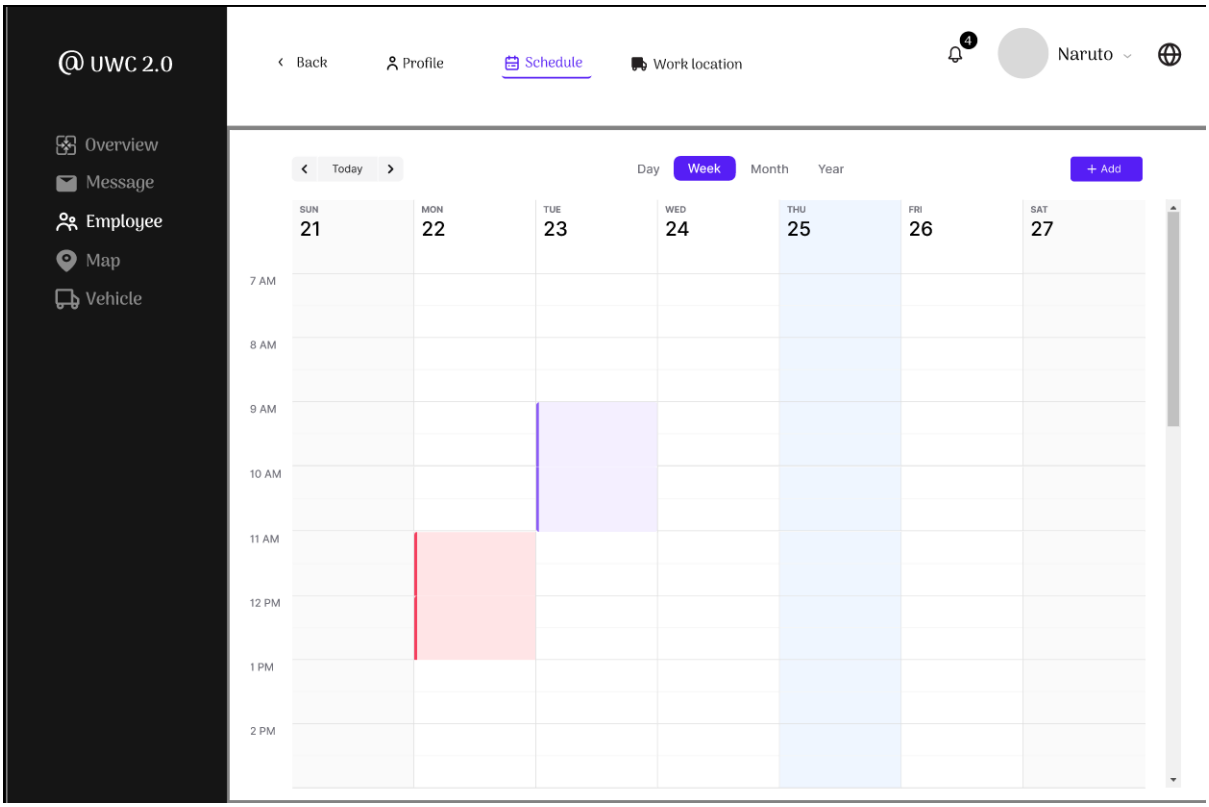


Figure 23: Employee working calendar

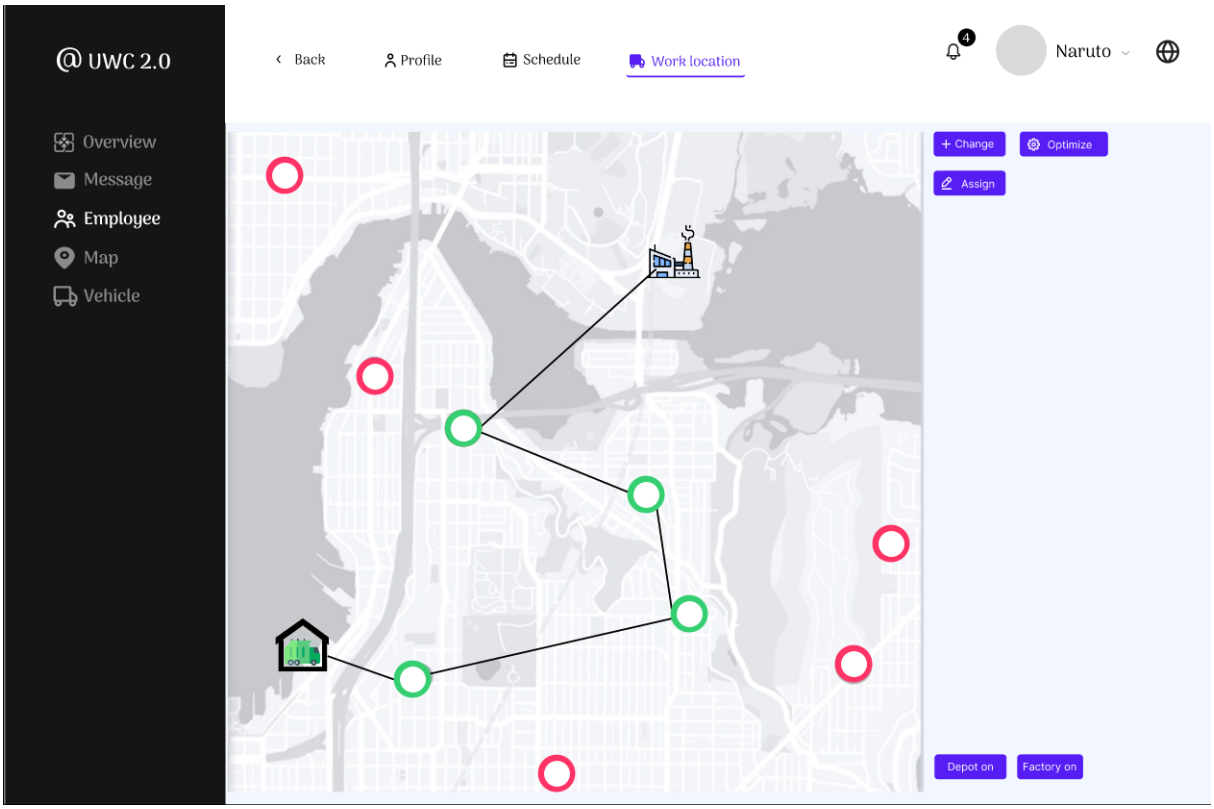


Figure 24: Employee working route

- Map page is dedicated to manage previously-created routes as well as giving the ability to create new ones. MCPs on hovering will show a popup window that has its status and lists of collectors and janitors working on it.

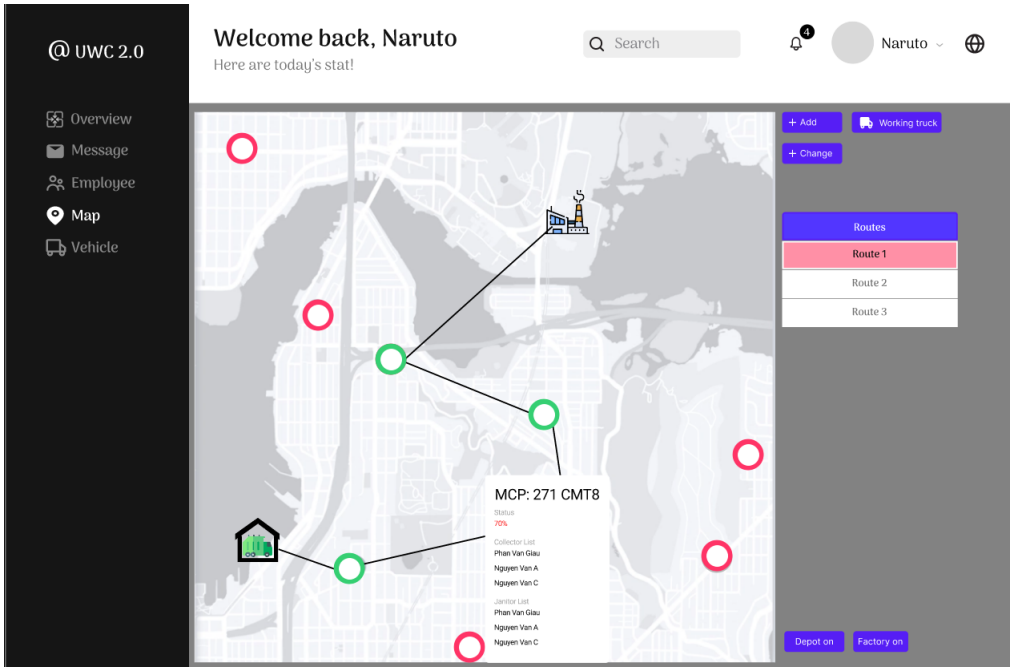


Figure 25: Manage routes

- One page for vehicles management

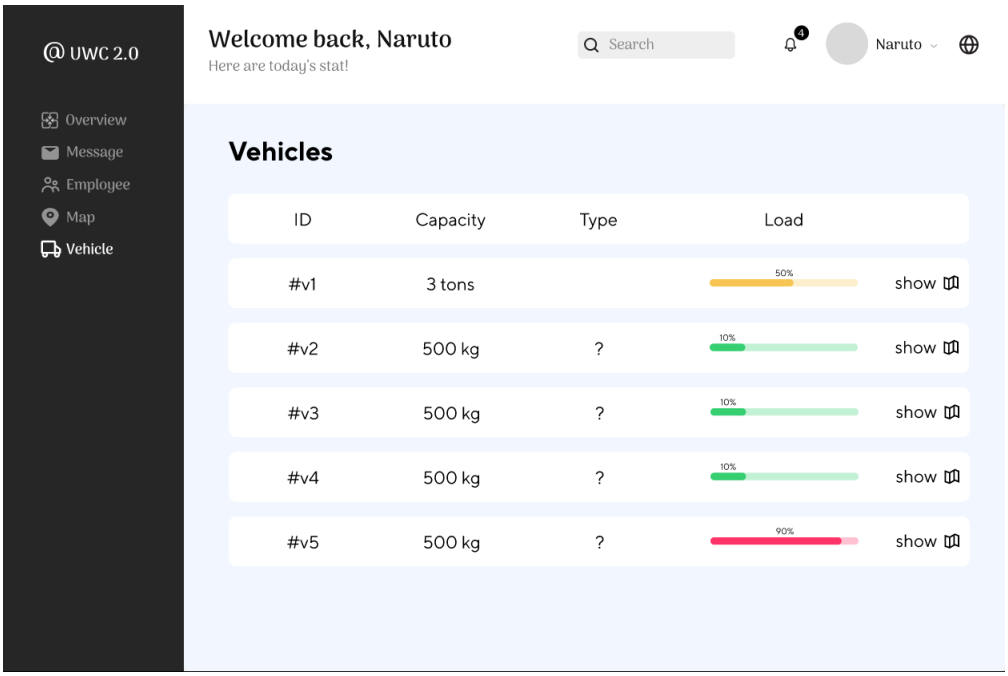


Figure 26: Manage vehicles

5 Implementation MVP2

5.1 Task Management Dashboard for back-officers

In this section, we present the final implementation of our application.

- Authentication prevents unintended users from manipulating the system.

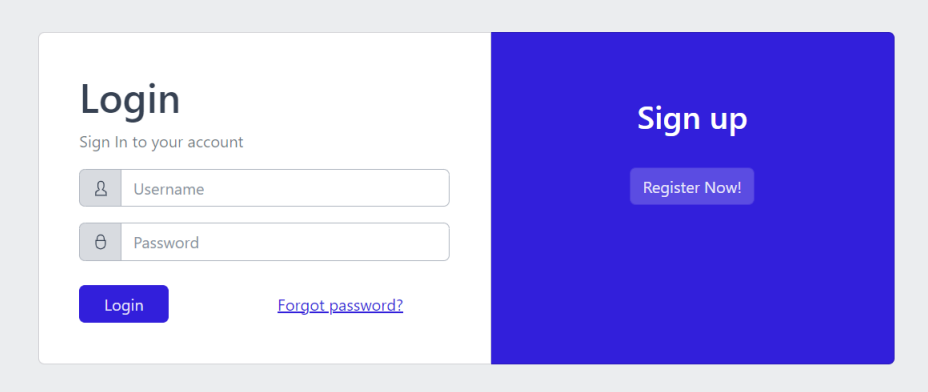


Figure 27: Authorizing legal back-officers

- Our dashboard gives some basic statistics on the number of employees as well as vehicles. There is also a chart that summarizes the status of MCPs.

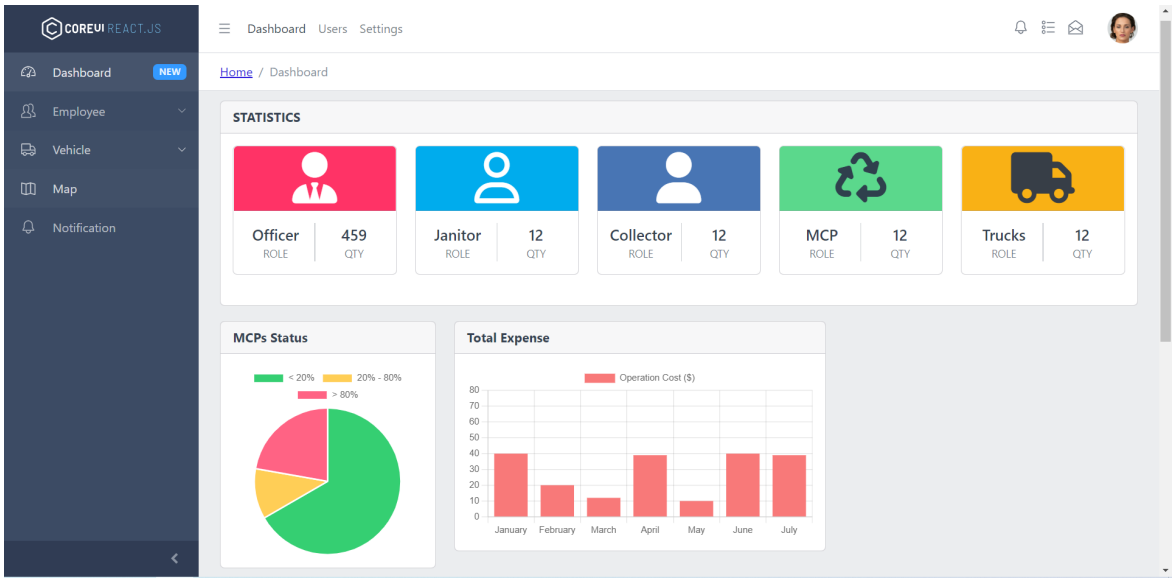


Figure 28: Dashboard view

- Back officers may enter the employee dashboard to manage their profiles and tasks.

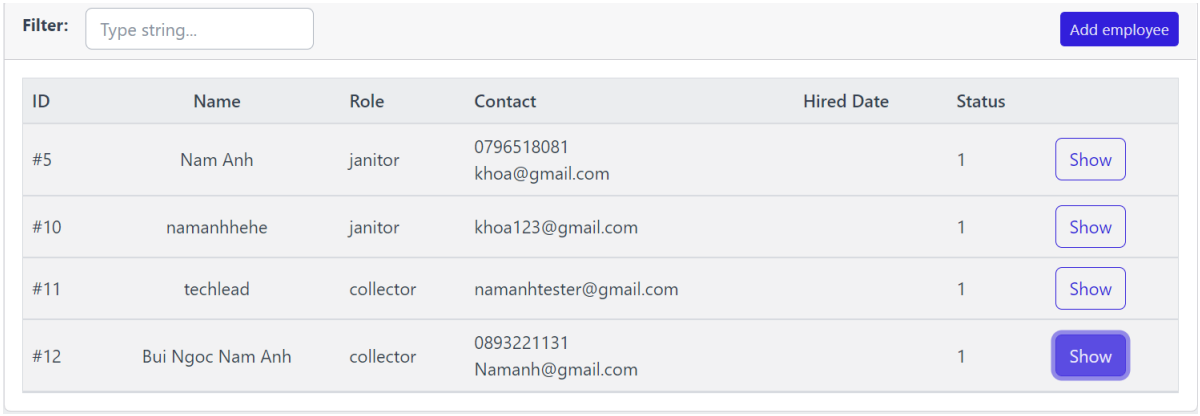


Figure 29: Employee dashboard

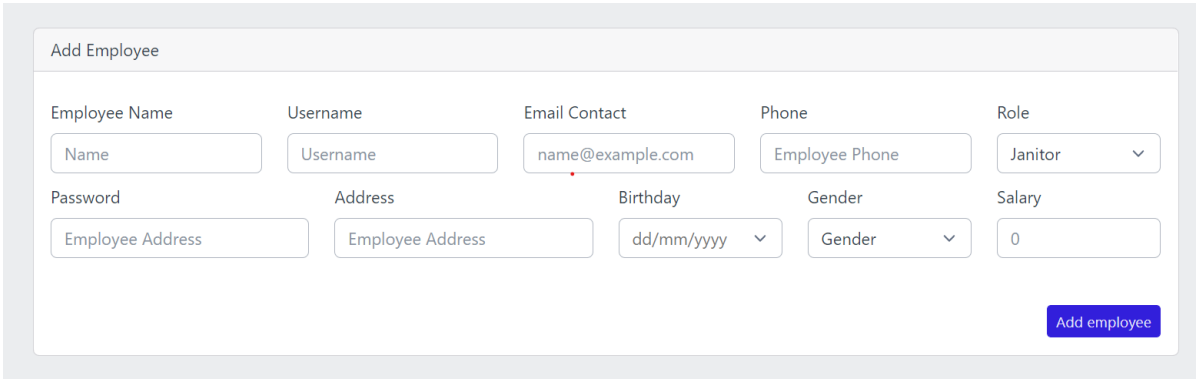
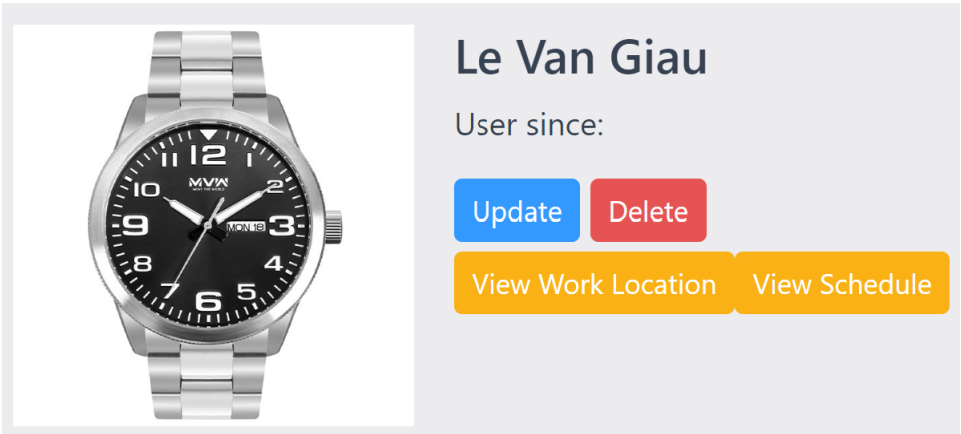


Figure 30: Add/Update form

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



- Our schedule is not quite complete.

- Our Map and Route module, however, match our expectation. Data are channeled from the database to display onscreen.

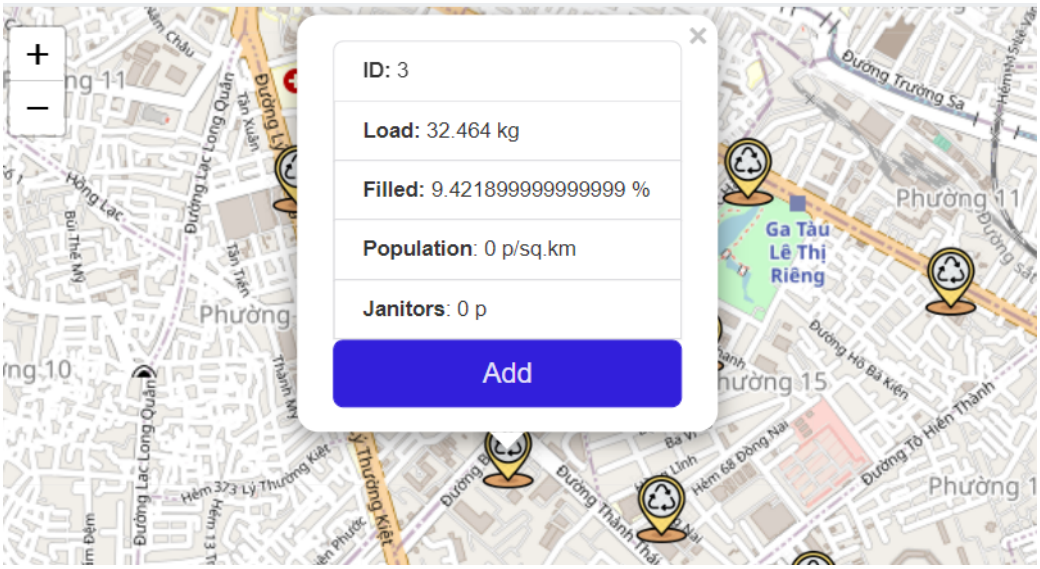


Figure 31: MCP display

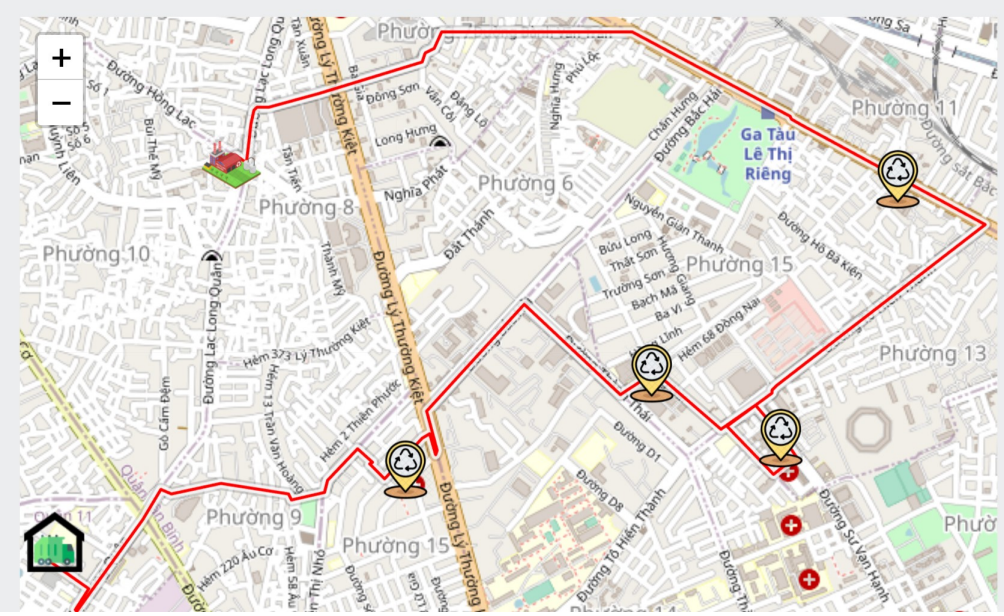
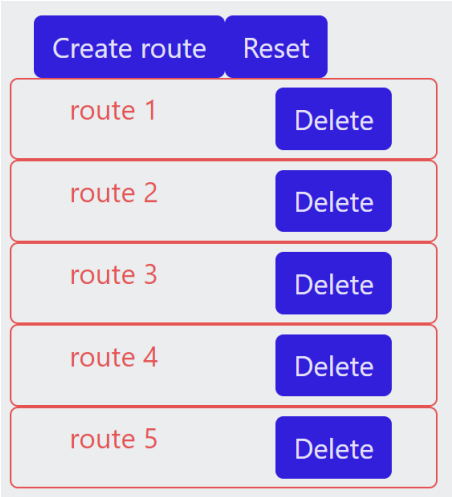
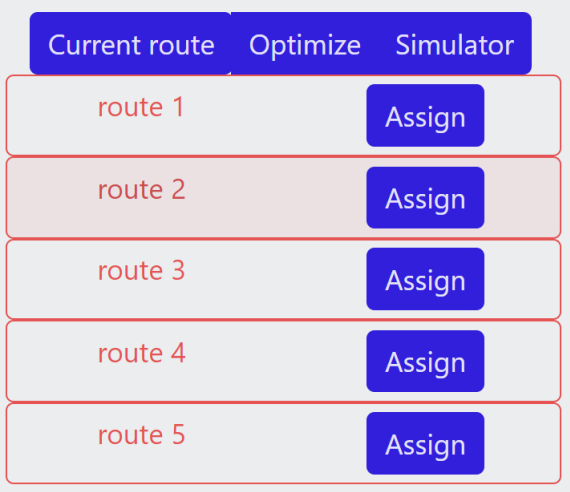


Figure 32: Route layout

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.

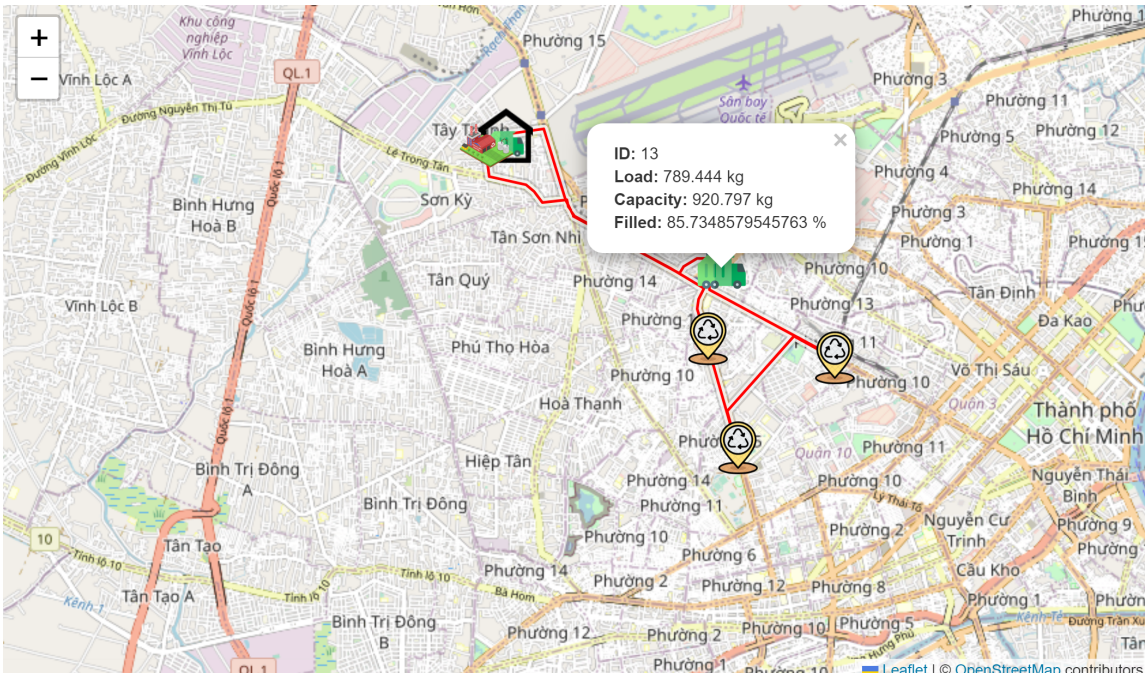


Figure 34: Simulator

Appendix A: How to set up

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.

Reference