Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

Software Engineering (CO3001)

Assignment Report

URBAN WASTE COLLECTION AID UWC 2.0

Lecturer: Quản Thành Thơ

Group name: Take the trash out!!!

Team members: Trương Tân Hào Hiệp – 2011211

Nguyễn Phạm Hoàng Vũ – 2052324

Lê Phước Gia Lộc – 2052155

Nguyễn Quốc Huy – 2053044

Trương Huỳnh Đăng Khoa – 2053145

Ho Chi Minh city, 21 September, 2022

TABLE OF CONTENTS

TASK 1

1.1. Requirement elicitation

1.1.1. Identify the context of this project.

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. Therefore, Urban waste collection aid - UWC 2.0 is created to help management and solid waste collection processes be more effective and economical.

1.1.2. Who are relevant stakeholders? What are their current needs? What could be their current problem?

Stakeholders	Problems	Needs
Back officers, employee (janitors & collectors)	- Communication between co-workers is ineffective. - Non-optimal waste-collecting route. - Application doesn't have enough features for employee & back officers to work on.	- Better communication channel. - More user-friendly interfaces in application. - New features in managing the flow of work.
Application provider company	- Complaints about the inefficiency of the application.	- Update the application to meet the new requirements.

1.1.3. In your opinion, what benefits UWC 2.0 will be for each stakeholder?

UWC 2.0 will be an improvement of UWC 1.0 in terms of performance, accessibility and convenience thereby satisfying the stakeholders requirements and bringing benefits to its users (including community, employees and employers).

1.2. Use-case diagram

1.2.1. Describe all functional and non-functional requirements that can be inferred from the project description.

Functional Requirements:

General requirements:

– Account management for the back officers and employee (log-in, manage personal information, ...)

– Real-time communication (through messages) between staffs.

– Changes in route (in case of busy MCPs, better route found, …) must notify employee in advance.

Back officers:

– Collectors and Janitors management.

+ View/justify collectors and janitors' personal information.

+ View collectors and janitors work calendar.

+ Add/ Remove collectors and janitors.

– Vehicles management.

+ View/ Justify technical information of vehicles (weight, capacity, ...).

+ View current status of vehicles (position, fuel, ...).

+ Assign vehicles to collectors, trollers to janitors.

+ Add/ Remove vehicles.

– MCPs management.

+ View/ Justify information of MCPs (position, capacity, ...).

+ View current status of MCPs (empty/non-empty status, ...).

– Send notification (when MCP is busy, announce route for janitors and collectors, ...).

– Task assignment.

+ Assign route for collectors.

+ Assign janitors to MCPs.

Janitors and collectors:

– Check in/ out work.

– Have an overview of work location on virtual map.

– View work calendar:

+ View general work calendar.

+ View work calendar in details (daily, weekly tasks and important informations regarding the task such as location, time, etc.)

Non-functional Requirements:

Performance

- Communication should be carried out with a delay of less than 1 second.
- At most 2 sec delays for system's responses to any interaction with users.

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.

Usability

The user interface is designed to be simple, appealing and convenient. The system has a friendly interface with simple interactive button for every particular feature to adapt all the function of the software. Furthermore, it has minimal yet enough information and images; in order for our software to be easy to use and accessible - so that users can conveniently use it without needing any prerequisite experience or knowledge.

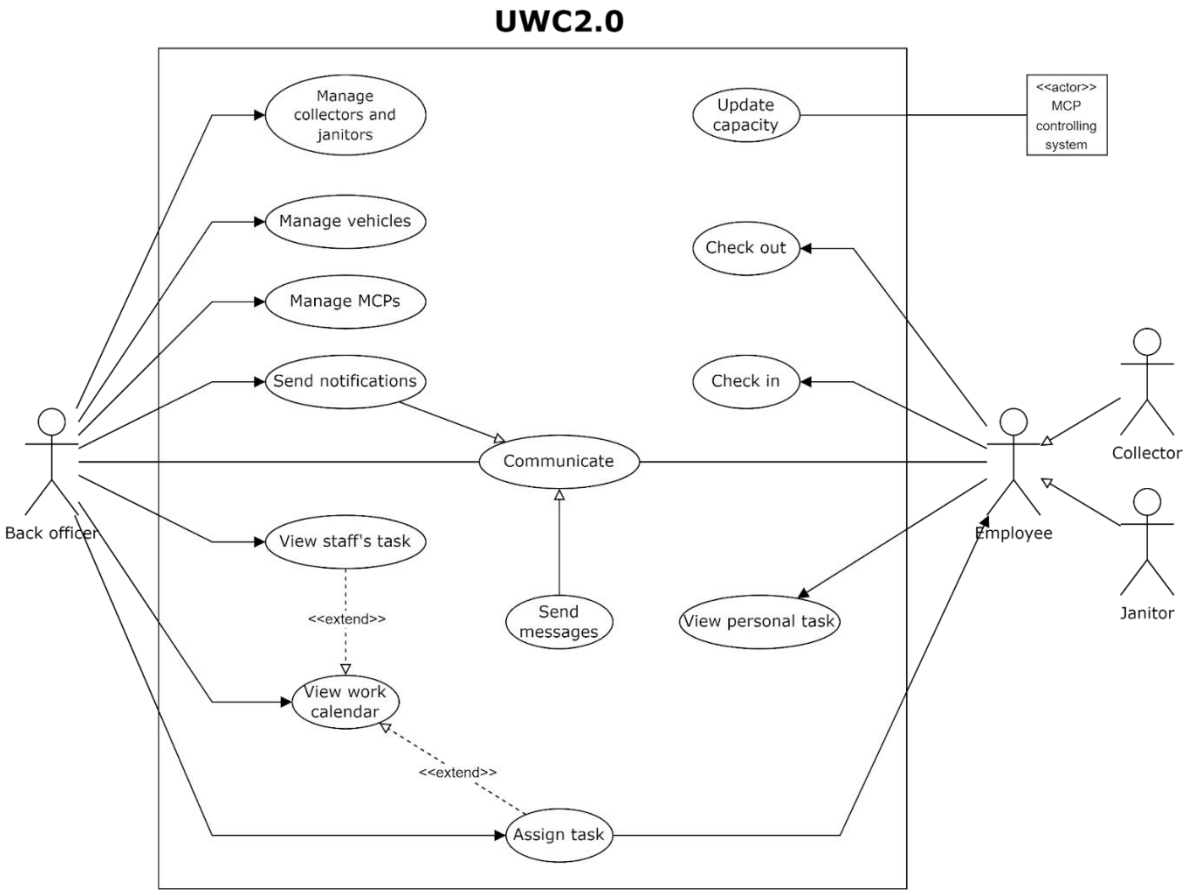
Availability

- The system must be accessible throughout regular business hours (from 7 a.m to 6 p.m). There can be no more than five seconds of downtime during regular business hours on any given day. Therefore, data is available by 7 am local time after an overnight update.
- Information should be updated from MCPs every 15 minutes with the availability of at least 95% of their operating time.

Adaptability

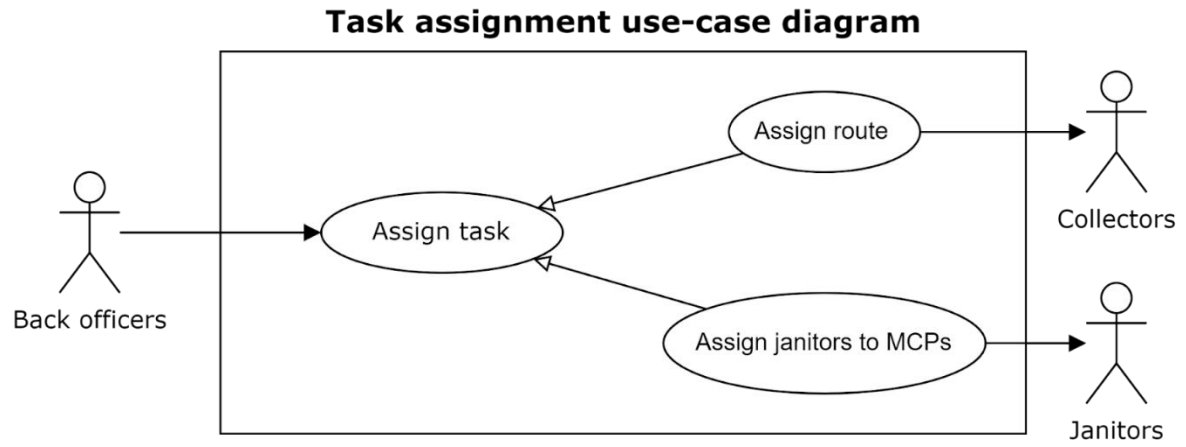
- UWC 2.0 system interfaces should be in Vietnamese, with an opportunity to switch to English in the future.
- It is expected that the Task Management be interoperable with the UWC 1.0 as much as possible.

1.2.2. Draw a use-case diagram for the whole system



1.3. Task assignment module

1.3.1. Draw the Task assignment module use-case diagram



1.3.2. Describe the use-case using a table format

Use-case name	**Assign task**
Actor	Back officers
Description	Back officers assign tasks (of a chosen area) to collectors and janitors
Trigger	Back officers chose an area in the areas list (from “Task management menu”)
Preconditions	Back officers have signed in using legitimate system accounts commensurate with their position.
Normal flow	1. The system displays the “Area task assignment menu”, which includes: - Map of area’s MCPs. - Area’s task calendar. 2. Back officers interact with either component in the menu. 3. System changes/modifies the interface based on the user interactions and moves to the relevant use case.
Postconditions	- “MCP symbol” selected: Move to “Assign route” use-case - Work shift entries (in area’s task calendar) selected: Move to “Assign janitors to MCPs” use-case
Exceptions	
Alternate flows	

Use-case name	**Assign route**
Actor	Back officers

Description	Assign route to collectors in designated work shifts.
Trigger	Back officer clicked on a “work shift” entry (in the task calendar from “Area’s task assignment menu”).
Preconditions	Back officers have signed in using legitimate system accounts commensurate with their position.
Normal flow	1. System displays a list of available collectors. 2. Back officer assigns a collector from the list to the selected work shift. 3. System assigns an optimized route to the collector. 4. System notifies the collector about the newly assigned work.
Postconditions	Collectors are assigned to selected work shift with optimized routes.
Exceptions	Exception 1: at step 2 2a. Back officer clicks the ‘X’ button to close the list of available collectors. 2b. Continue with the “Area’s task assignment menu”.
Alternative Flows	Alternative Flow 1: at step 2 2a. Back officer clicks a different work shift. 2b. Continue step 1 in normal flow.

Use-case name	**Assign janitors to MCPs**
Actor	Back officers
Description	The back officers assign janitors to MCPs.
Trigger	Back officer clicks the “MCP symbol” (on the map from “Area’s task assignment menu”).
Preconditions	Back officers have signed in using legitimate system accounts commensurate with their position.

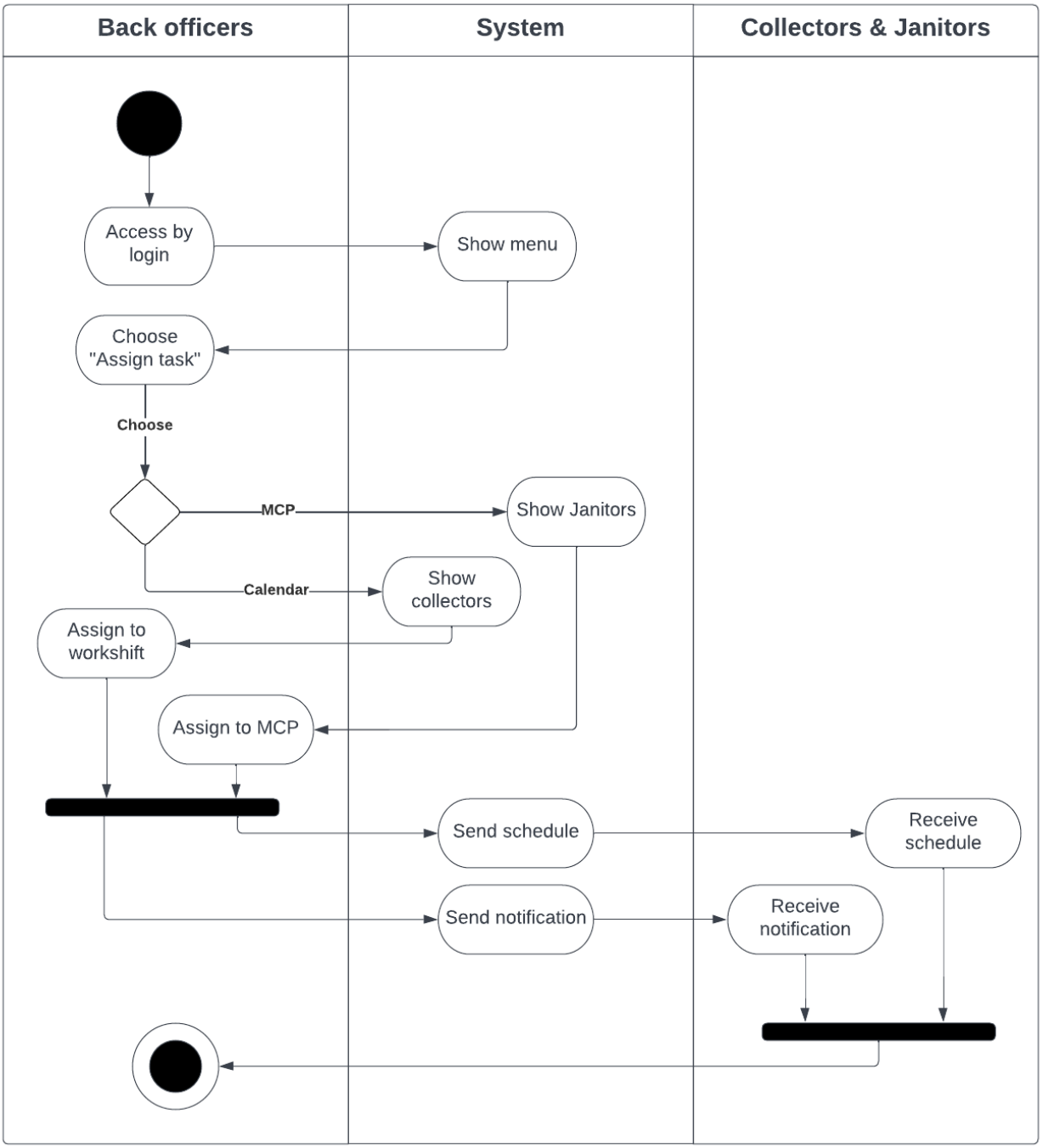
Normal flow	1. The system displays the “MCP management menu”, which includes: - MCP’s information (capacity, status, …). - MCP’s task calendar. 2. The Back Officer chooses a work shift from MCP’s task calendar. 3. The system shows a list of available janitors to that work shift. 4. The Back Officer assigns janitors to that work shift. 5. System notifies the janitor about the newly assigned work.
Postconditions	Collectors are assigned to selected work shifts and MCPs.
Exceptions	Exception 1: at step 2 2a. Back officer clicks the ‘X’ button to close the MCP’s management menu 2b. Continue with the “Area’s task assignment menu”. Exception 1: at step 4 2a. Back officer clicks the ‘X’ button to close the list of available janitors. 2b. Continue with the “MCP management menu”.
Alternative Flows	Alternative Flow 1: at step 2 2a. Back officer clicks on the symbol of different MCP 2b. Continue step 1 in normal flow. Alternative Flow 2: at step 4 4a. Back officer clicks on a different work shift. 4b. Continue step 3 in normal flow.

TASK 2

2.1. Activity diagram

2.1.1. Draw an activity diagram to capture the business process between systems and the stakeholders in Task Assignment module

Activity Diagram:



2.1.2. Description

– Back Officer's identification has been verified, and the system will now display the “Area task assignment menu” menu selections which has “Map of area’s MCPs” and “Area’s task calendar”.

– The task's recipient is determined by the back officer. If “Map of area’s MCPs” is selected, the system will display the “MCP management menu”, which includes: MCP’s information (capacity, status, …) and MCP’s task calendar. The back officer chooses a work shift in the MCP’s task calendar. Then, the system shows a list of available janitors to that work shift and the Back Officer assigns janitors to that work shift.

– If “Area’s task calendar” is selected instead, the system will display a list of available collectors and Back Officer assigns a collector from the list to the selected work shift. System will then assign an optimized route to the collector.

– System will assign schedule to the employee then send notification about task assigned to the employee’s calendar.

2.2. Conceptual solution for route planning task

2.2.1. Conceptual solution

Assuming that a city has many areas.

– System will show the list consisting of all areas and its status of the tasks assigned.

– Back officers can choose one of the areas to assign task.

– System shows all work shifts of every weekday of that area.

+ Work shifts can be changed (still guaranteed during prime time) based on MCPs data from previous weeks (to optimize efficiency for the most waste concentrated time only).

– Back officers will assign available collectors to shifts (one collector each). This assignment only notifies the collector his shift.

– The route will be optimized by the system, then it will be updated and sent to the collector along with a notification before his/ her shift.

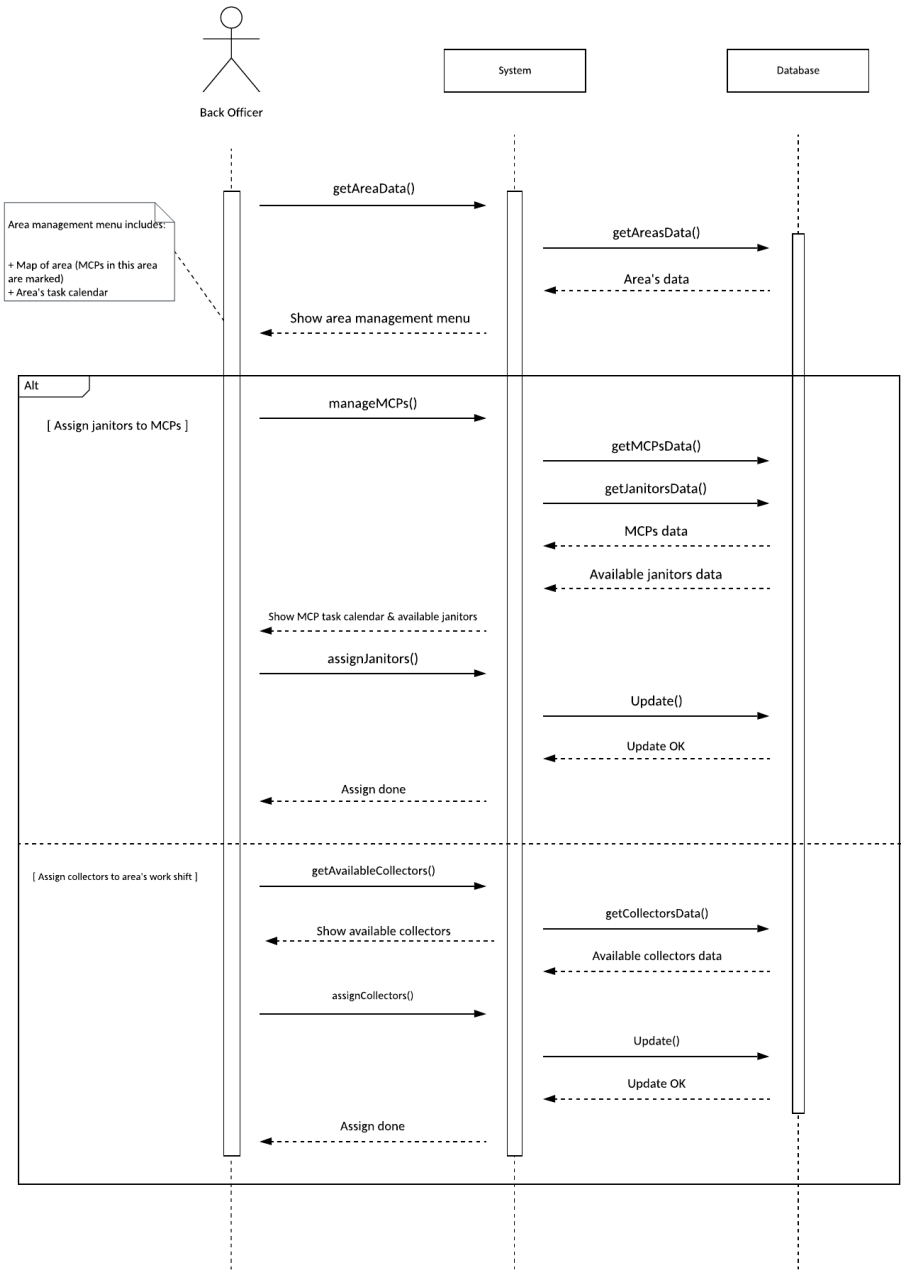
– The MCPs’ criteria to be selected for one route will be based on:

+ The MCPs that reach the threshold (real-time data).

+ The MCPs that is likely to reach the threshold soon (The prediction will be based on the previous weeks data).

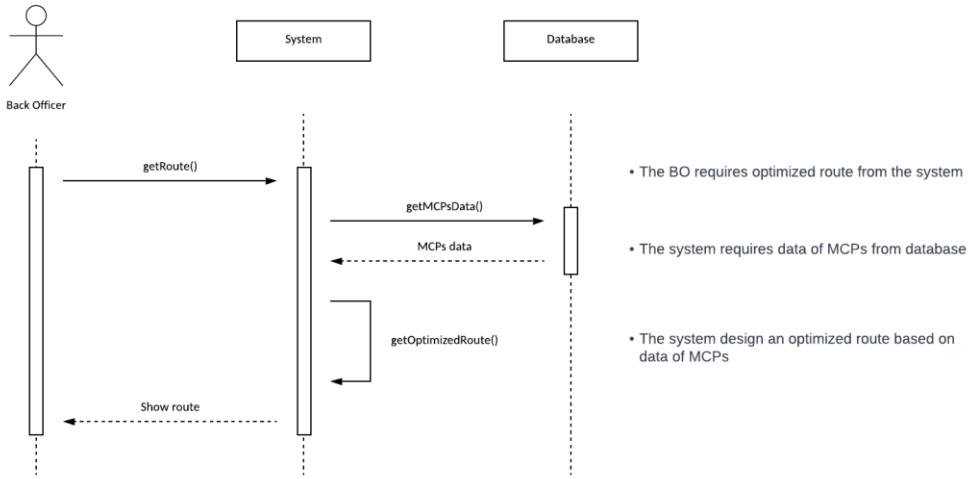
2.2.2. Sequence diagram

Task assignment:

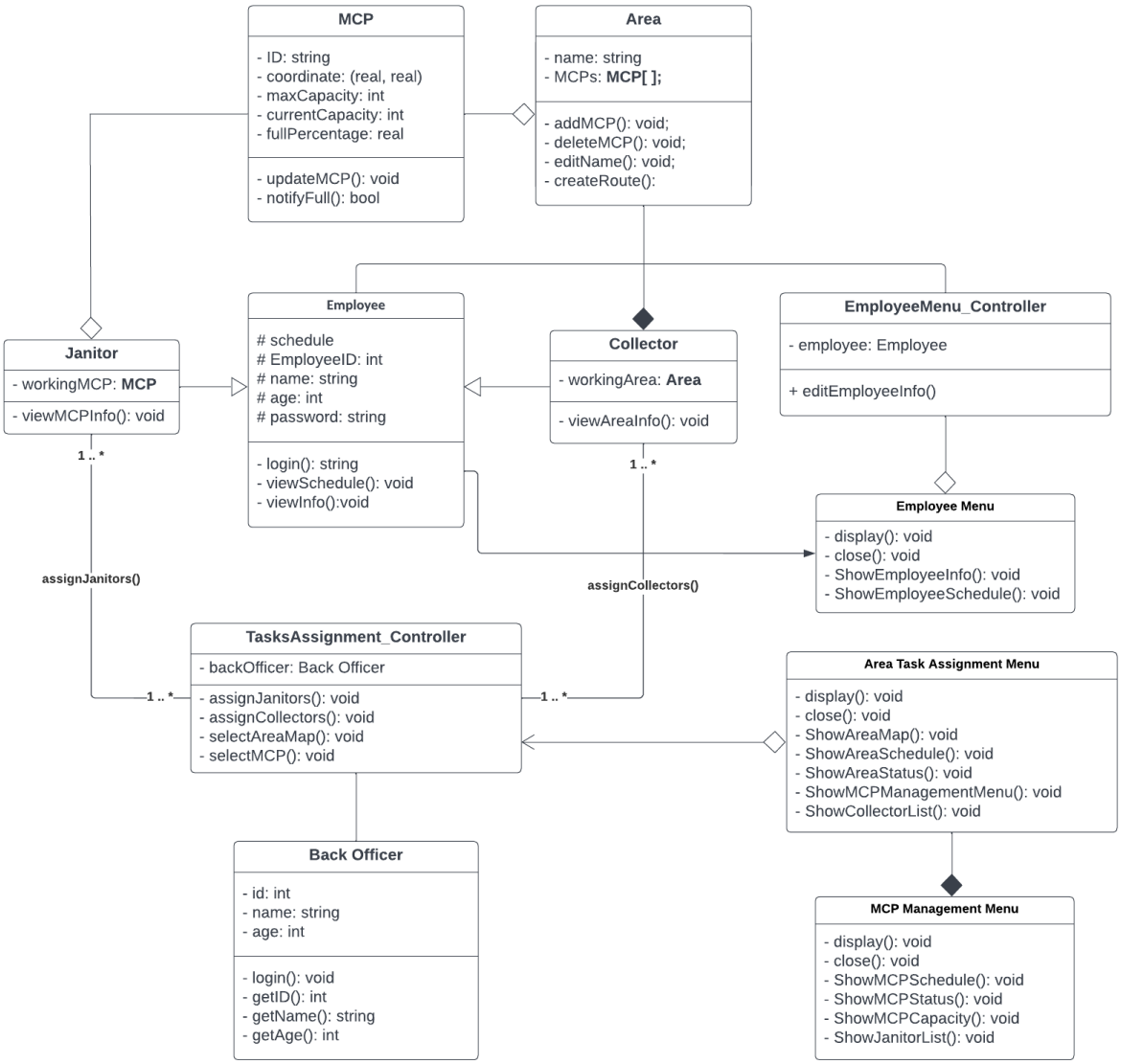


- The BO start the assignment by choosing the needed area
- In case the user clicks on an MCP (from the MCPs map), the system will show that MCP's task calendar and the list of available janitors
- The BO assign janitors to MCP's work shift (maybe by dragging and dropping the janitors in the given list into the work shift of MCP)
- In case, the user clicks on the area's task calendar below, the system will show the list of available collectors
- The BO assign collectors to area's work shift (maybe by dragging and dropping the collectors in the given list into the work shift of MCP)

Route planning:



2.3. Class diagram for Task Assignment Module



TASK 3

3.1. Architectural approach

3.1.1. Describe an architectural approach you will use to implement the system.

In the process of applying the design system, our team choose the MVC model which is an architectural pattern that separates an application into three main logical components such as Model, View, and Controller. Each architecture component is built to handle specific development aspect of an application, which is responsible for adding source code into 3 main parts, each part has its own task and is handled independently of the other parts of the software.

– Model (M): the component that stores all the data of the application and its related logic, is the bridge between the two components View and Controller. It responds to the request from the views and also responds to instructions from the controller to update itself. The model can be displayed as a database, it explicitly represents operations with the database such as viewing, retrieving, processing data, etc.

In our application, model is used to store information about janitor, collector, list of information of MCP, Vehicles, Routes, … Furthermore, it is also a method to call API, communicate with database to perform basic operations functions such as CREATE, READ, UPDATE, DELETE (CRUD).

– View (V): the component whose functionality is to manage the data presented to the user. A view requests directly to the model to give information or indirectly through the Controller so that it presents the output presentation to the user.

Our system can utilize this component for: login, registration, view tasks, view information, ... More details are displayed registration forms, login, buttons with their own functions such as submit user request or redirect, …

– Controller (C): component that handles user actions for the purpose of notifying the View as well as the Model if there is a change request.

Our application uses Controller to handle performing actions on the data through user interaction, returning the corresponding view from the request made.

Interaction flow:

User sends request -> Controller receives request -> Handle model if needed -> Return View result -> Display interface to user -> Wait for next request.

Our application works with a good number of features that require interactions between views and data so separates it into view, controller and model made it easier to implement and also further utilize the extensibility of the architectural approach (for any additional requirements in the model component do not affect what we defined in the view component).

The downside when it comes to this architectural pattern is for its difficulty to ensure the Model-View-Controller interaction which makes the implementation more complicated.

3.1.2. How many modules do you plan for the whole WMC 2.0 system? Briefly describe input, output and function of each module.

Module	Area Management
Input	name: string MCPs: MCP[ ] collectorID: string schedule: {name: string, day: enum Weekdays, workshift: int, collectorID} []
Output	newArea: Area existingArea: Area
Function	createArea(name, MCPs): newArea getAreaByName(name): existingArea updateAreaName(id, name): void updateAreaMCPs(name, MCPs): void deleteAreaByID(name): void updateAreaSchedule(schedule): void

Module	MCP Management
Input	coordinate: {x: float, y: float} maxCapacity: int currentCapacity: int ID: string janitorID: string schedule: {name, day, workshift, janitorID} []
Output	newMCP: MCP existingMCP: MCP
Function	createMCP(coordinate, maxCapacity, currentCapacity): newArea getMCPByID(ID): existingMCP

	updateMCPMaxCapacity(ID, newMaxCapacity): void updateMCPCurrentCapacity(ID, newCurrentCapacity): void updateMCPSchedule(schedule): void deleteMCPByID(ID): void

Module	Communicate

Input	backOfficerID: string
	employeeID: string
	id: string
	message: {
	content: string
	sender: “Back Officer” | “Employee”
	}

Output	conversation: Conversation
Function	createConversation(backOfficerID, EmployeeID): conversation getConversation(id): conversation addMessage(id, message): void

Module	Vehicle
Input	name: string capacity: float fuel: float image: string collectorID: string
Output	vehicle: Vehicle
Function	createVehicle(name, capacity, fuel, collectorID): vehicle getVehicle(id): vehicle deleteVehicle(id): void updateVehicle(id, name, capacity, fuel): void updateCollector(id, collectorID): void

Module	Employee Services

Input	name: string age: int role: “collector” | “janitor” password: string oldPassword: string newPassword: string id: string
Output	employee: Employee collectorSchedule: {area, day, workshift} [] janitorSchedule: {MCP, day, workshift} []
Function	login(name, id, password): collector getEmployeeInfo(id): employee updateInfo(id, name=””, age=0): void updatePassword(id, oldPassword, newPassword): void getOwnSchedule(id, role): collectorSchedule | janitorSchedule

Module	Back Officer
Input	name: string age: int password: string id: string janitorID: int collectorID: int employeeID: int
Output	existingBackOfficer: Back Officer collector: Collector janitor: Janitor areaSchedule: {name, day, workshift, collectorID} [] MCPSchedule: {name, day, workshift, collectorID} []+
Function	login(id, password): existingBackOfficer updateInfo(id, name =””, age = 0): void createEmployee(name, age, role): collector | janitor updateEmployeeInfo(id: collectorID | janitorID , name=””, age=0): void deleteEmployee(id): void sendMessage(employeeID: string, message: string)

Module	Task Assignment
Input	employeeID: int
Output	areaSchedule: {name, day, workshift, employeeID} [] MCPSchedule: {name, day, workshift, employeeID} []
Function	getAreaList(name): Area []

	getMCPList(id): MCP [] getCollectorList(): Employee[] getJanitorList(): Employee[] assignJanitorToMCP(name, employeeID): void assignCollectorToArea(name, employeeID): void

3.2. Implementation Diagram for Task Assignment module