GROUP ASSIGNMENT

Technology Park Malaysia

CT038-3-2-OODJ

OBJECT-ORIENTED DEVELOPMENT WITH JAVA

APU2F2211CS(DA)

HAND OUT DATE : 24 DECEMBER 2022

HAND IN DATE : 24 FEBRUARY 2023

WEIGHTAGE : 50%

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the administrative counter**
2. **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing)**
3. **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld**
4. **Cases of plagiarism will be penalized**
5. **The assignment should be bound in an appropriate style (comb bound or stapled).**
6. **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**
7. **You must obtain 50% overall to pass this module.**

Group 47	
Student ID	Student Name
TP065116	Neaw Aik Ka
TP067345	Edmund Chen Siang Zuan
TP067349	Nadila Binti Ahmad Shahrul Nizam

Table of Contents

1 Abstract

This paper provides an in-depth discussion of Object-Oriented Programming (OOP) and its fundamental concepts and principles. Specifically, it focuses on the implementation of OOP in a property management system for Parkhill Residence, Bukit Jalil. The paper presents a visualization of the system through the use-case diagram, use-case specification, and a class diagram, which helps to illustrate the code and the overall structure of the system. By implementing OOP, the system benefits from abstraction, encapsulation, inheritance, and polymorphism, which provide greater security, flexibility, and code reusability. Overall, the paper highlights the benefits of OOP and its application in developing complex systems.

2 Introduction

Parkhill Residence Condominium's property management system is a piece of software that performs as a centrally controlled system for coordinating, monitoring, and managing a condominium establishment's daily activities using object-oriented programming. Eight different types of users can access the property management system built and deploy for the Parkhill Residence Condominium. The eight types of users include the building manager, account executive, admin executive, building executive, resident or tenant, vendor, security guard, and visitors. The primary aim of creating the system was to alleviate users' workload by developing a unified system.

To implement OOP in the Parkhill Residence Condominium's property management system, individual classes will be created for each user type, such as Building_Manager, Account_Executive, Admin_Executive, Building_Executive, Resident, Vendor, SecurityGuard, and others. These classes have properties and methods that define each user's responsibilities and role. These classes can interact with one another through inheritance, encapsulation, abstraction, modularity and polymorphism to create a cohesive and flexible system. This technique can increase the system's modularity, maintainability, and scalability, making it simpler to add new features or modify ones in the future.

Since each of the eight users serves a different purpose as a user, they each have separate access points to the system. Aside from the visitors, the remaining seven users can log in and out of the system to perform various responsibilities they have been assigned. Higher-level management typically has greater access to the system. The building manager, for example, can manage the system's numerous users, develop, and monitor various reports, plan operations

and budgets for fund distribution and maintenance, and manage the team structure. Account executives can send invoices, receipts, and statements to residents, tenants, and vendors, record, view, edit payments and collect delinquent fees. As an admin executive, one may oversee the condominium's many units, renters, residents, complaints, personnel, facilities, and facility booking. Finally, the building executive can assign and prioritize employee tasks, complaints, patrolling schedules, and work reports.

Considering lower-level management has less responsibility in the management system, they may not require access to certain technical functionalities. For instance, residents and tenants are limited to accessing and editing their profiles, making payments or deposits, examining their payment history, pending fees, statements, bills, and receipts, applying for facility booking and visiting cards and registering complaints. Likewise, vendors are restricted from accessing and editing their profile, paying for rent, utilities, and services, examining their payment history, pending fees, bills, statements, and receipts, and registering complaints. On the other hand, security guards have different functionalities, such as inspecting visitor passes, recording and updating visitor admission and checkpoint check-in, and recording and updating an incident.

3 Assumptions

A few notable circumstances in this system are not mentioned in the question paper and should be clarified in this paragraph. For the authentication process of users, the system mandates that prospective residents, vendors, and security guards undergo an authentication process by an admin executive before they can become registered users. Another situation is that the report was generated by the building manager. Upon generating a report on the building management page, the system will produce an additional report in the "png" format, sending it to a specific program file and the building manager can save it for further usage. When a user adds a team leader or vice team leader on the team structure page, the system will trigger a prompt to confirm the addition of a team leader or vice leader is added more than one.

In this system, when the user makes the payment, the payment status will reflect as "paid" in the resident's interface. However, only upon account executive confirmation will it be updated in the payment history. In the event of a facility booking, the system will automatically check for time conflicts, and if one arises, it will request the user to reschedule. Meanwhile, the opening time of the facility is from 9 am to 10 pm. The user will not be able to book the facility out of this time scope.

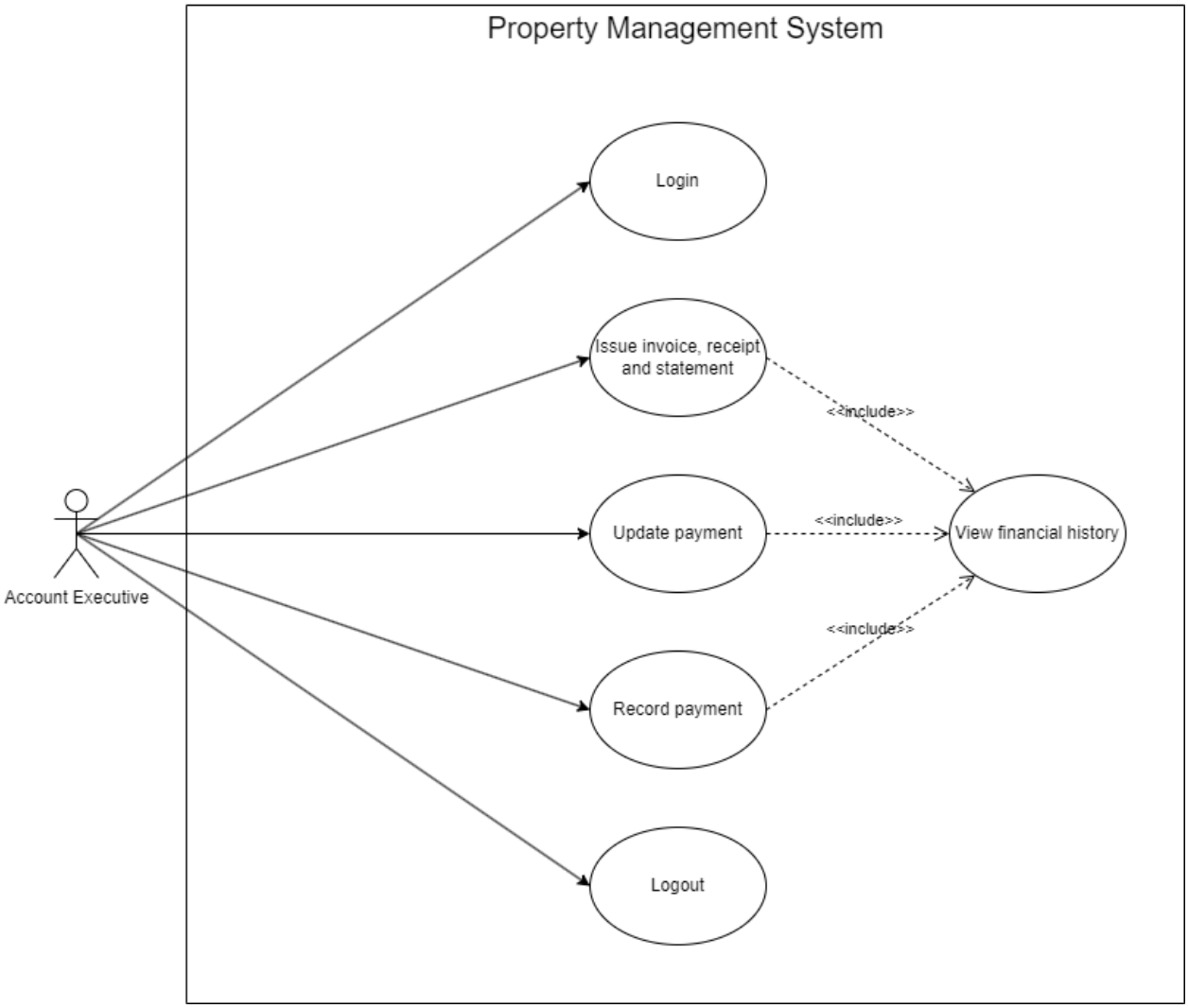
While the vendor and resident pages have separate units, the admin executive must enter the vendor's unit information.

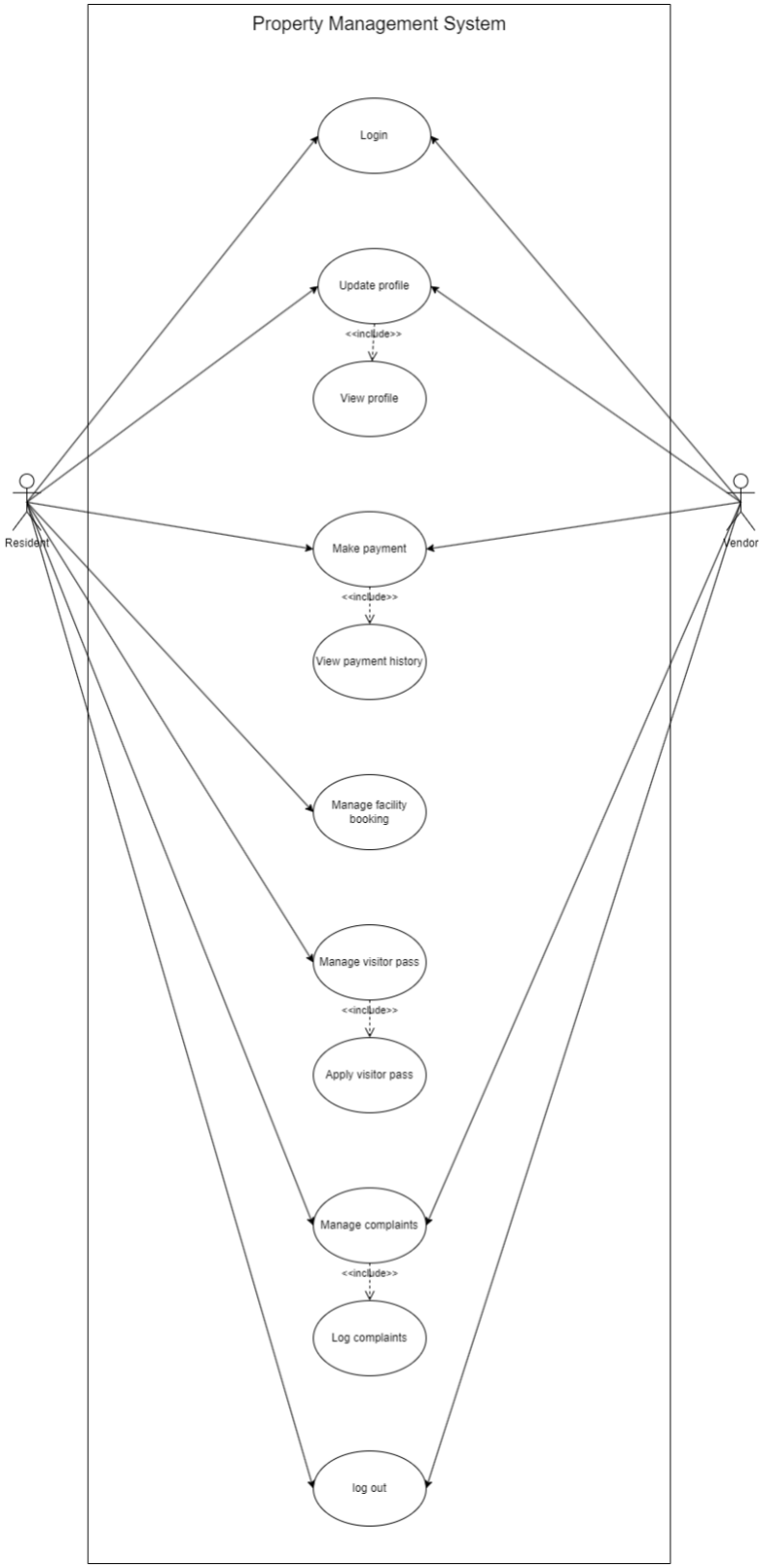
When a resident introduces a new unit, the system will automatically verify whether the unit and resident information exist. One resident can possess only one unit at the same time, but each unit can have a maximum of one resident and one owner at any given time. In the security system, the patrolling schedule can have multiple security guards patrol in the area concurrently.

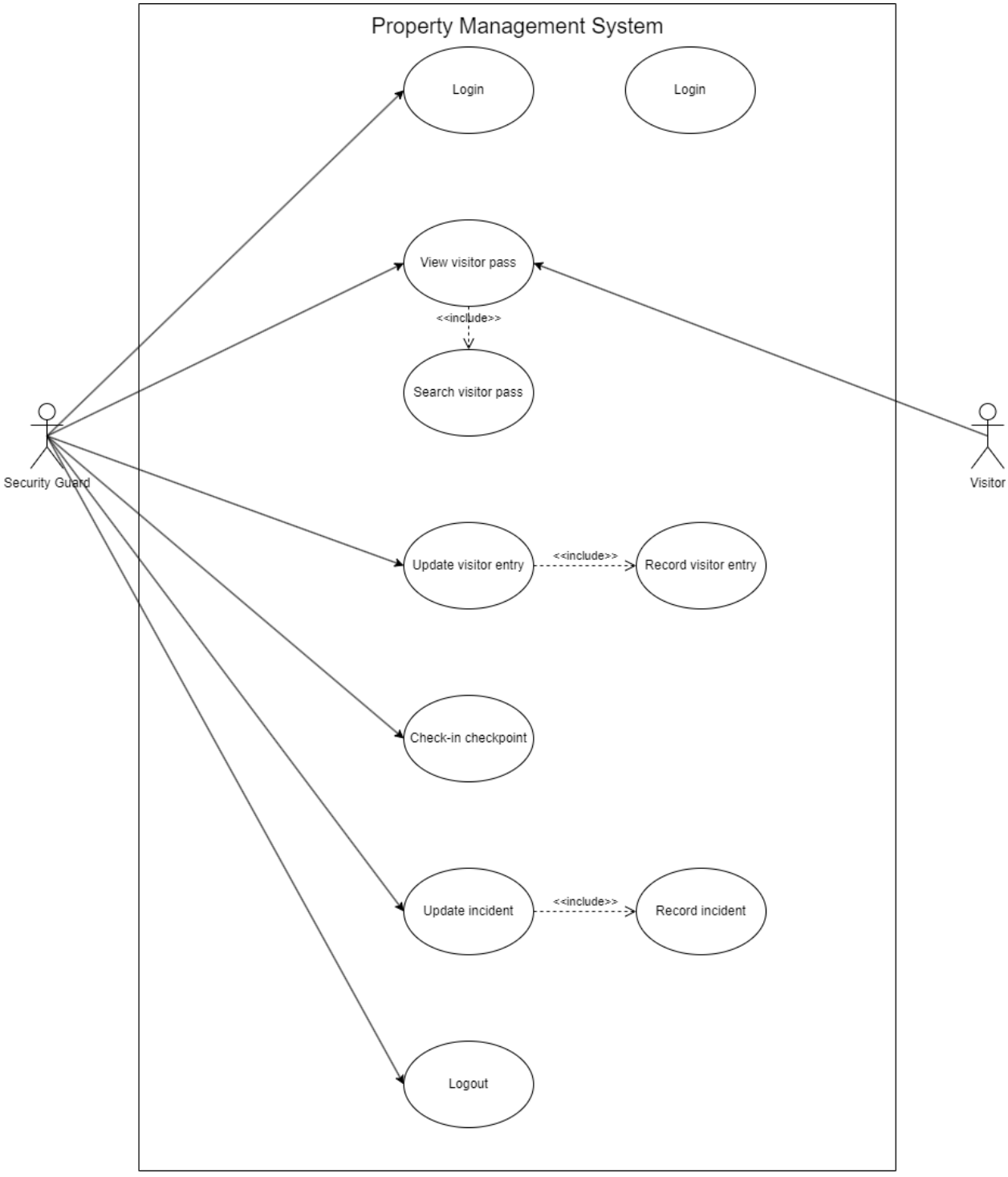
In user management, after adding a user ID, all executives and managers cannot change their own IDs. Each of them has a unique ID. This means that once their IDs are assigned and created, they cannot be changed. This is to ensure the uniqueness of user IDs in the system and the integrity of the data. Therefore, managers and executives need to carefully consider when creating IDs and ensure that they use unique and easily identifiable IDs.

4 Unified Modelling Language Diagram

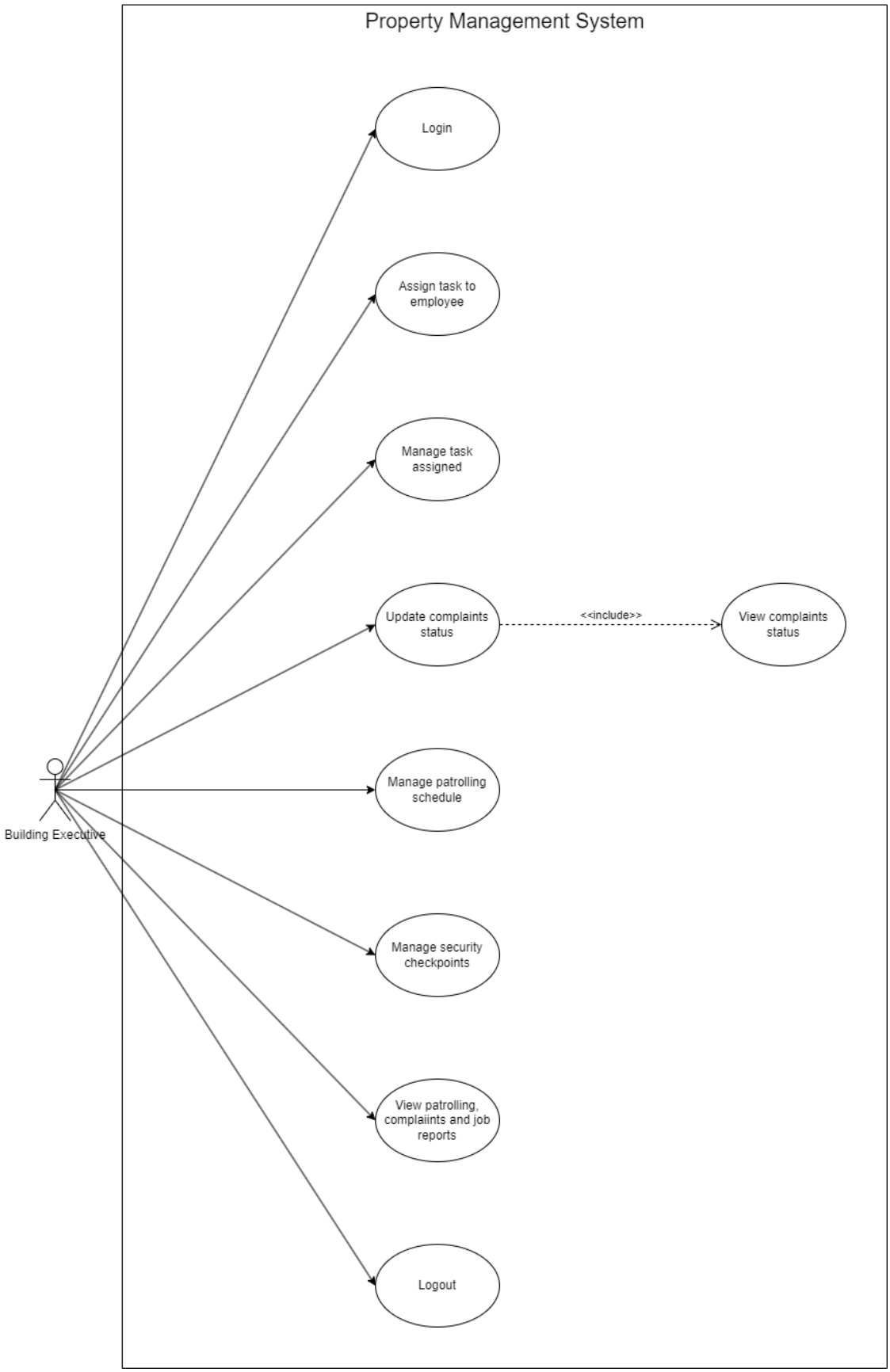
4.1 Use-Case Diagram

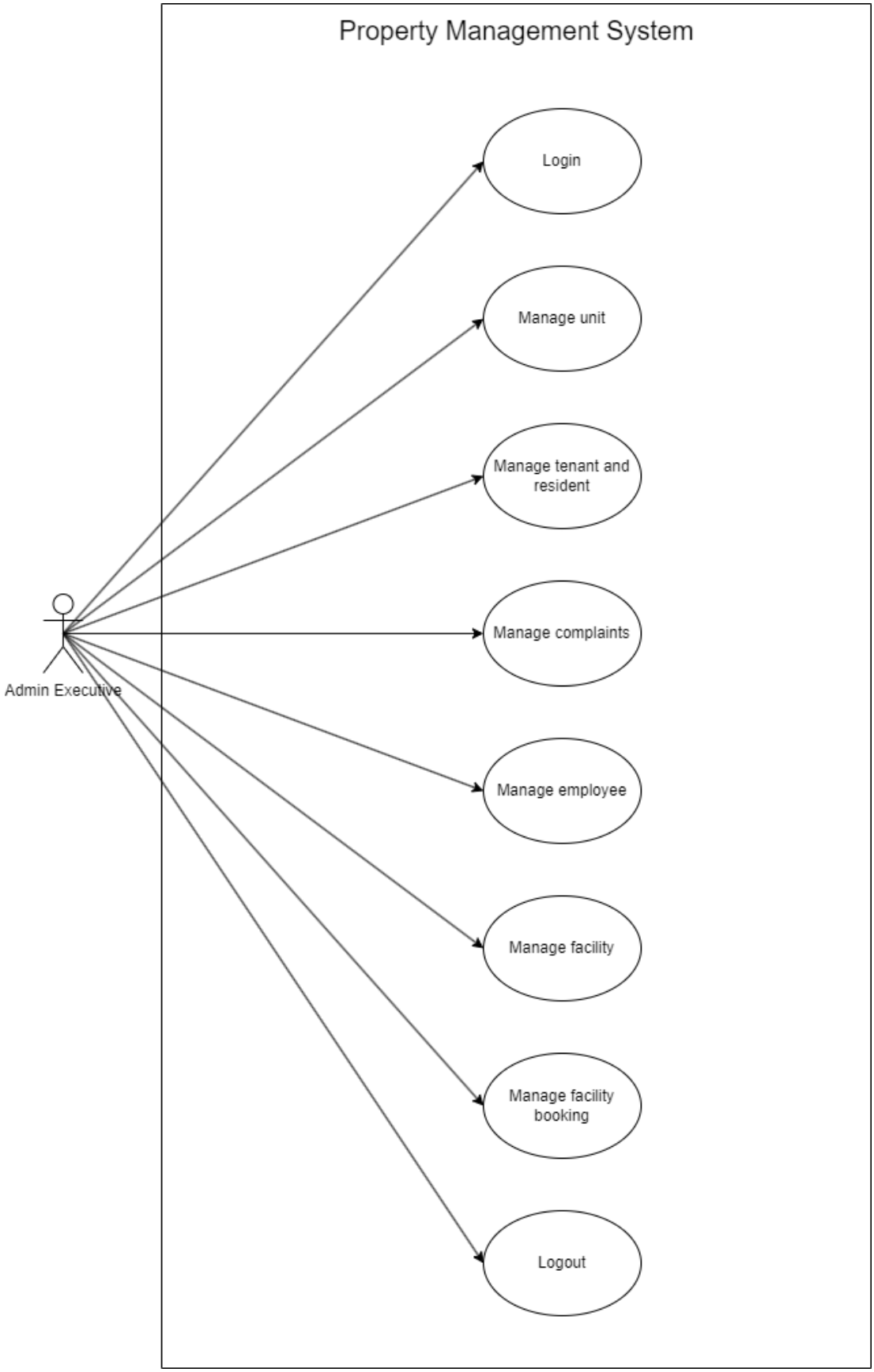



Property Management System
Login
Update profile
<<include>>
View profile
Resident
Vendor
Make payment
<<include>>
View payment history
Manage facility booking
Manage visitor pass
<<include>>
Apply visitor pass
Manage complaints
<<include>>
Log complaints
log out

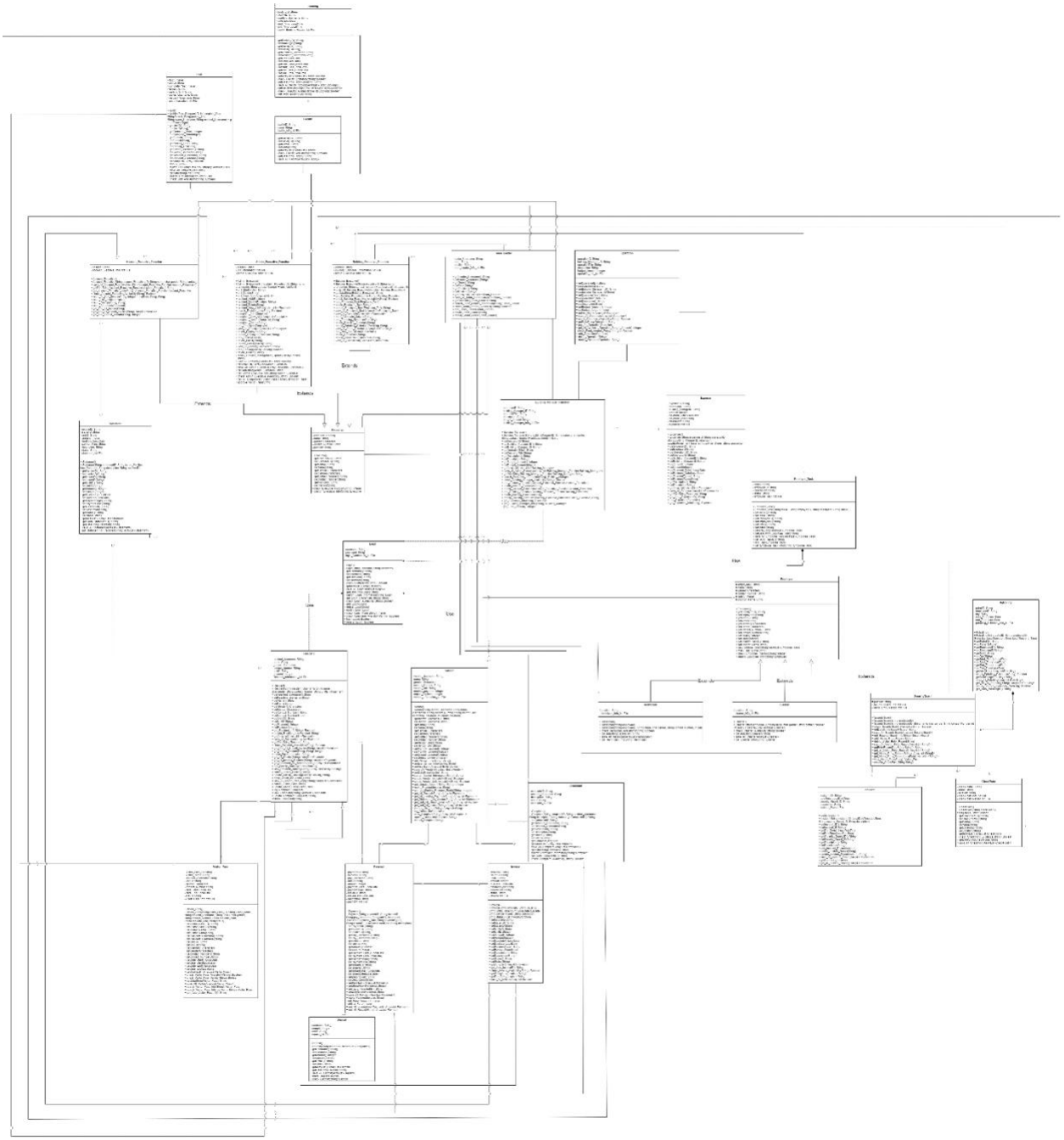

Property Management System
Login
Login
View visitor pass
<<include>>
Search visitor pass
Security Guard
Visitor
Update visitor entry
<<include>>
Record visitor entry
Check-in checkpoint
Update incident
<<include>>
Record incident
Logout


Property Management System
Login
Manage user's account
View report
<<include>>
Generate report
Allocate fund for maintenance
<<include>>
View operation and budget planning
Manage team structure
Log out
Building Manager

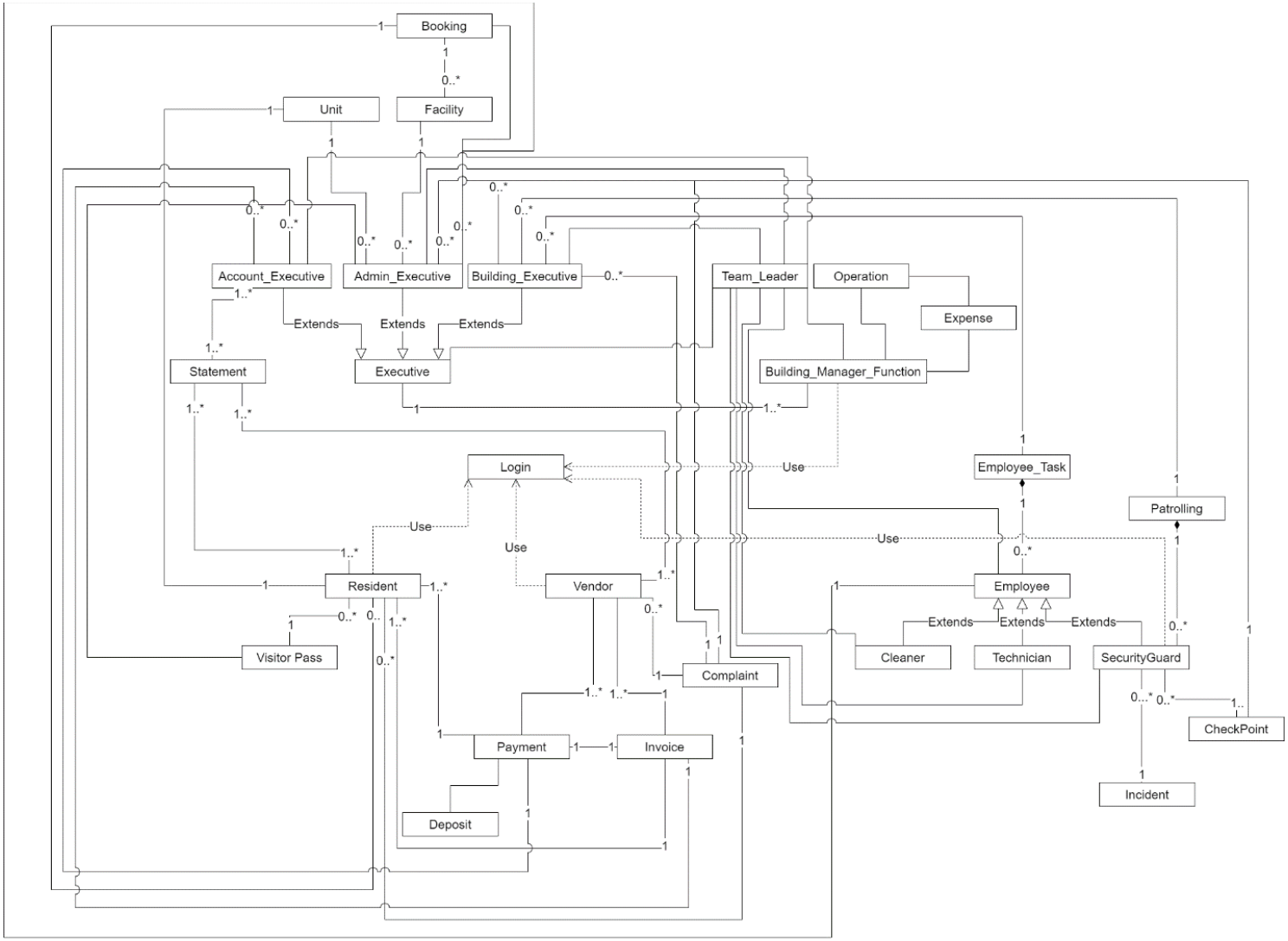

Property Management System
Login
Assign task to employee
Manage task assigned
Update complaints status
<<include>>
View complaints status
Manage patrolling schedule
Manage security checkpoints
View patrolling, complaiints and job reports
Logout
Building Executive


Property Management System
Login
Manage unit
Manage tenant and resident
Manage complaints
Manage employee
Manage facility
Manage facility booking
Logout
Admin Executive

4.2 Class Diagram

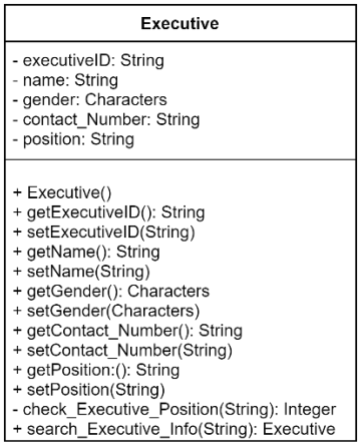


full diagram

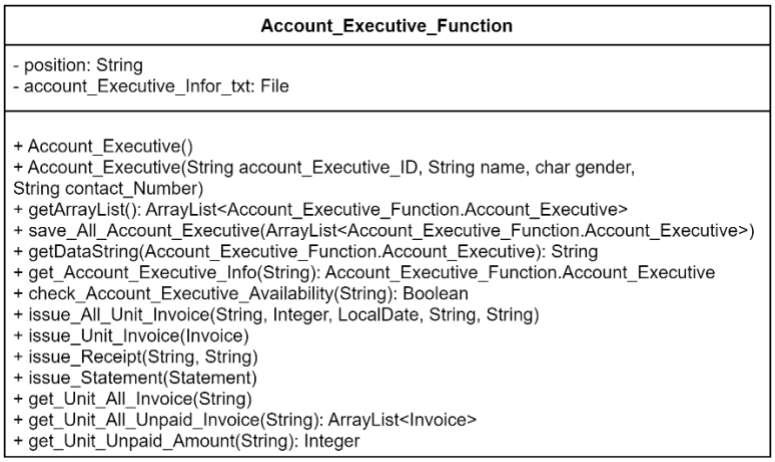


Class diagram (relationship only)

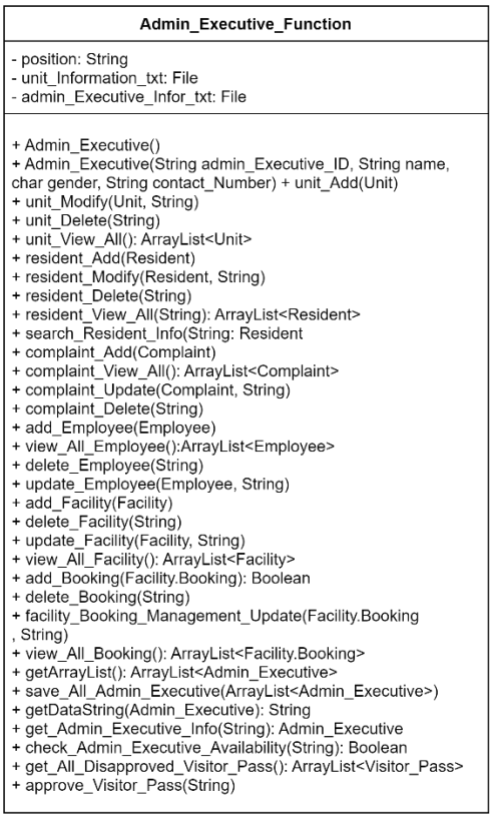
4.2.1 Executive Function



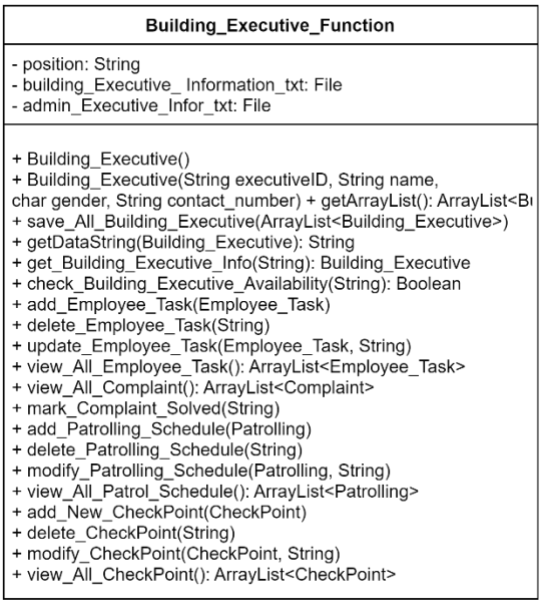
4.2.2 Account Executive Function



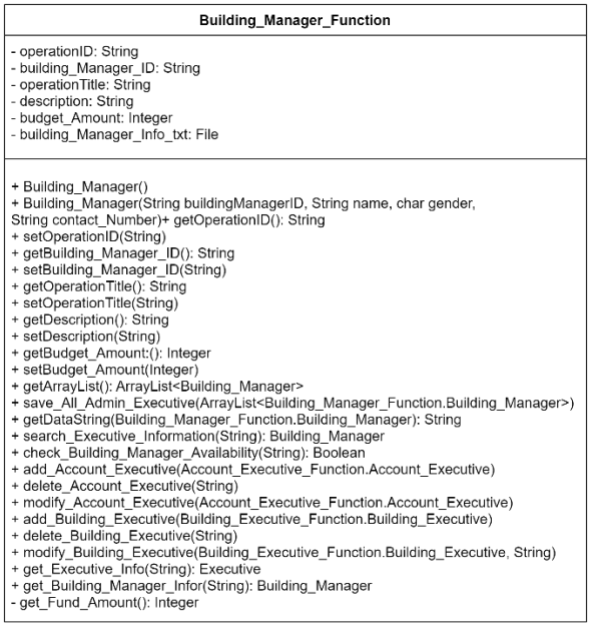
4.2.3 Admin Executive Function



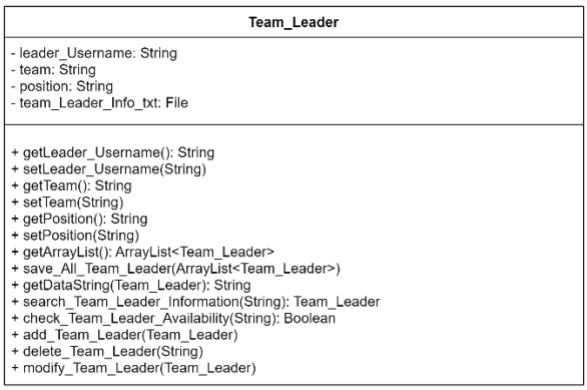
4.2.4 Building Executive Function



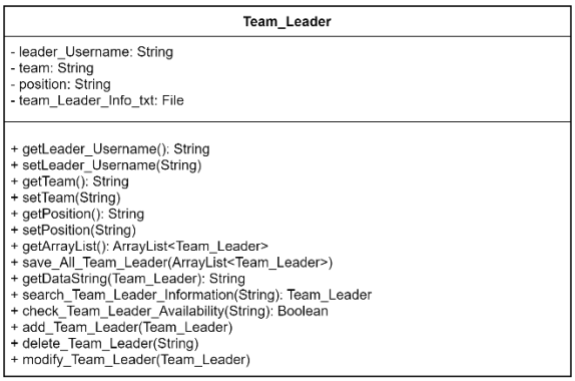
4.2.5 Building Manager Function



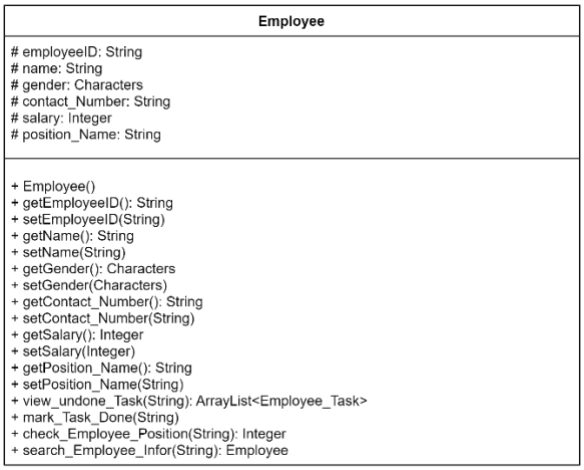
4.2.6 Team Leader



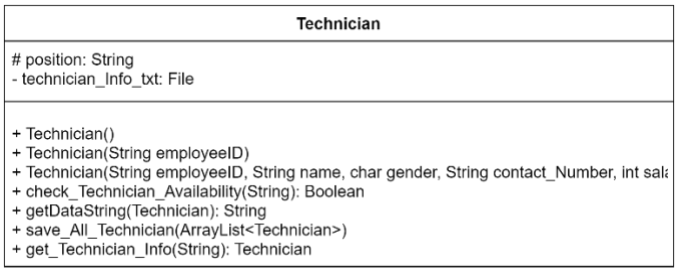
4.2.7 Operation



4.2.8 Employee



4.2.9 Technician



4.2.10 Cleaner

Cleaner
position: String - cleaner_Info_txt: File
+ Cleaner() + Cleaner(String employeeID, String name, char gender, String contact_Number, int salary)+ getArrayList(): ArrayList<Cleaner> + check_Cleaner_Availability(String): Boolean + getDataString(Cleaner): String + save_All_Cleaner(ArrayList<Cleaner>) + get_Cleaner_Infor(String): Cleaner

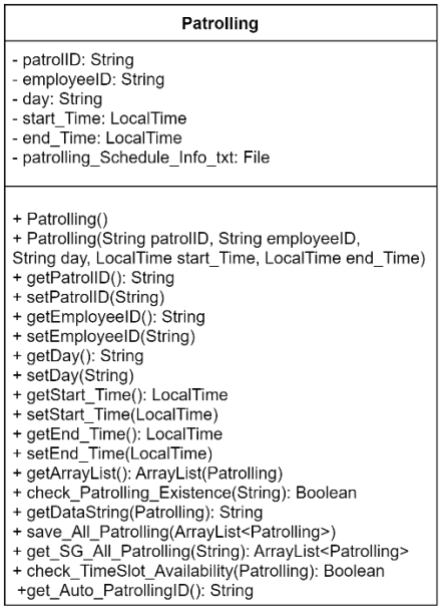
4.2.11 Security Guard

SecurityGuard
position: String - security_Guard_Info_txt: File - visitor_Entry_Record_txt: File
+ SecurityGuard() + SecurityGuard(String employeeID) + SecurityGuard(String employeeID, String name, char gender, String contact_Number, int + check_SecurityGuard_Availability(String): Boolean + getDataString(SecurityGuard): String + save_All_SecurityGuard(ArrayList<SecurityGuard>) + get_Security_Guard_Info(String): SecurityGuard + add_Visitor_Entry_Record(String) + delete_Visitor_Entry_Record(String) + save_All_Visitor_Entry_Record(ArrayList<String[]>) + getEntryRecordDataString(String[]): String + get_all_Visitor_Entry_Record(): ArrayList<String[]> + get_Visitor_Entry_Record(String): ArrayList<String[]> + get_Visitor_Entry_Record(LocalDate): ArrayList<String[]> + view_Visitor_Pass(String): Visitor_Pass + checkIn_CheckPoint(String,String)

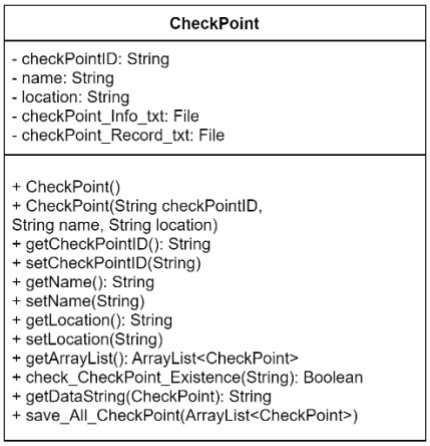
4.2.12 Employee Task

Employee_Task
- taskID: String - employeeID: String - description: String - status: String - employee_Task_txt: File
+ Employee_Task() + Employee_Task(String taskID, String employeeID, String description, String status) + getTaskID(): String + setTaskID(String) + getEmployeeID(): String + setEmployeeID(String) + getStatus(): String + setStatus(String) + getArrayList(): ArrayList<Employee_Task> + getDataString(Employee_Task): String + save_All_Employee_Task(ArrayList<Employee_Task> + get_Auto_TaskID(): String + add_Task(Employee_Task) + get_Employee_Task_Info(String): Employee_Task

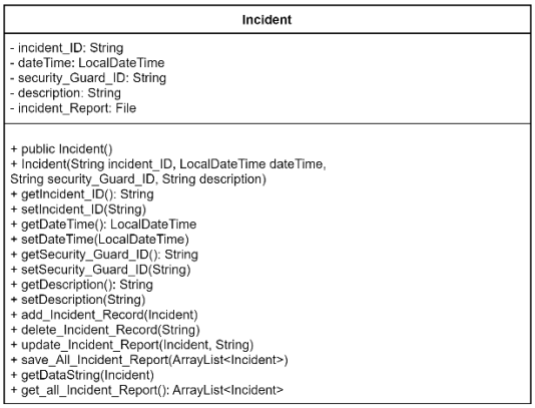
4.2.13 Patrolling



4.2.14 Check Point



4.2.15 Incident

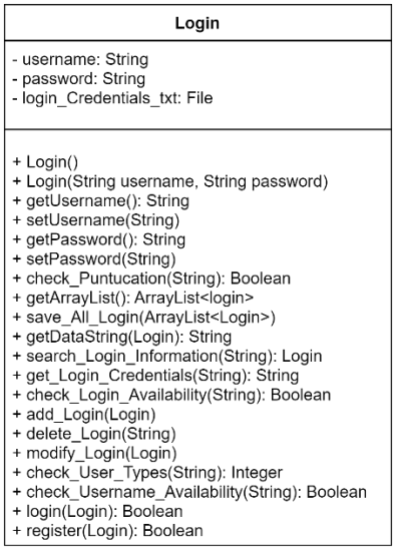


4.2.16 Resident

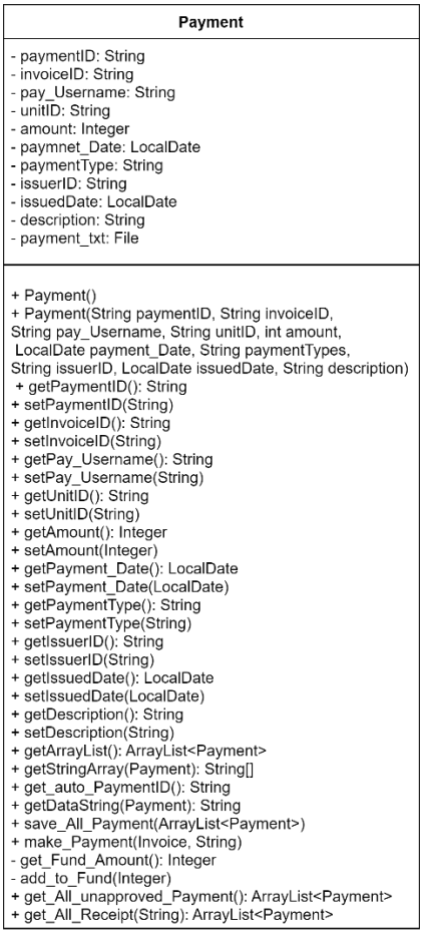
4.2.17 Vendor


Vendor
- vendor_Username: String
- name: String
- gender: Characters
- contact_Number: String
- vendor_Unit: String
- monthly_payment: Integer
- unpaid_payment: Integer
- vendor_Information_txt: File
+ Vendor()
+ Vendor(String vendor_Username, String name,
char gender, String contact_Number, String vendor_Unit,
int monthly_payment, int unpaid_payment)
+ getVendor_Username(): String
+ setVendor_Username(String)
+ getName(): String
+ setName(String)
+ getGender(): Characters
+ setGender(Characters)
+ getContact_Number(): String
+ setContact_Number(String)
+ getVendor_Unit(): String
+ setVendor_Unit(String)
+ getMonthly_payment(): Integer
+ setMonthly_payment(Integer)
+ getUnpaid_payment(): Integer
+ setUnpaid_payment(Integer)
+ get_Vendor_Info(String): Venodr
+ update_Vendor_Infor(Vendor, String)
+ getArrayList(): ArrayList<Entity.Vendor>
+ save_All_Vendor(ArrayList<Entity.Vendor>
+ getDataString(Vendor): String
+ search_Vendor_Information(String): Vendor
+ check_Vendor_Availability(String): Boolean
+ check_Vendor_Unit_Availability(String): Boolean
+ get_Unpaid_Amount(String, String): Integer
+ make_Payment(Invoice, String)
+ change_Monthly_Payment_Rental(String, Integer)
+ get_All_Receipt(String): ArrayList(Payment)
+ get_All_pending_Payment(String): ArrayList<Payment>
+ get_Statement_for_Vendor(String): ArrayList<Statement>
+ get_Unit_All_Unpaid_Invoice(String): ArrayList<Invoice>
+ get_Unit_All_Invoice(String): ArrayList<Invoice>
+ vendor_Unit_rent(String): ArrayList<String>
+ log_Complaint(Complaint)
+ view_Complaint(String): ArrayList<Complaint>
+ update_Complaint(Complaint, String)
+ cancel_Complaint(String)

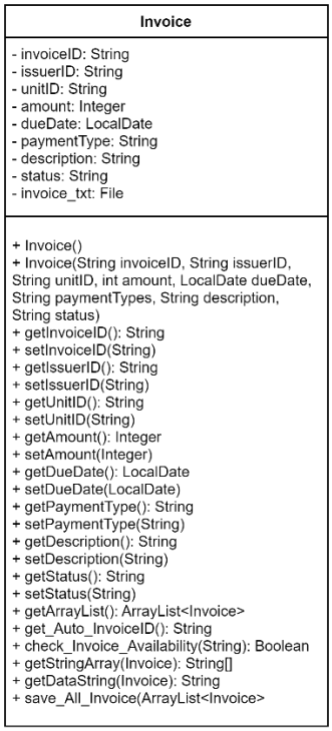
4.2.18 Login



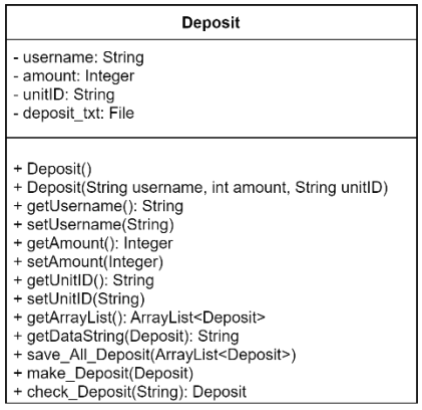
4.2.19 Payment



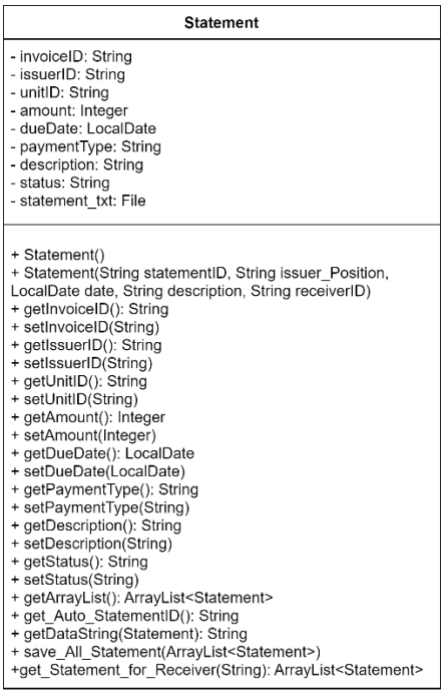
4.2.20 Invoice



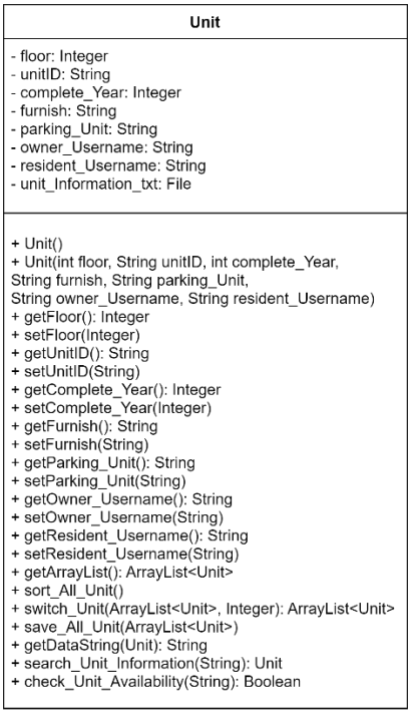
4.2.21 Deposit



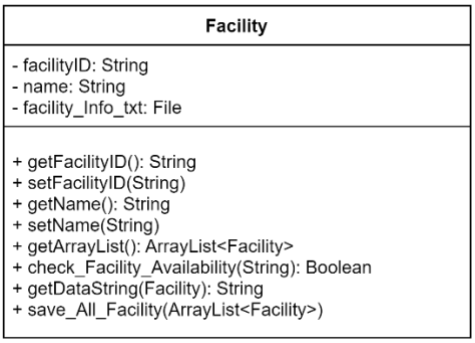
4.2.22 Statement



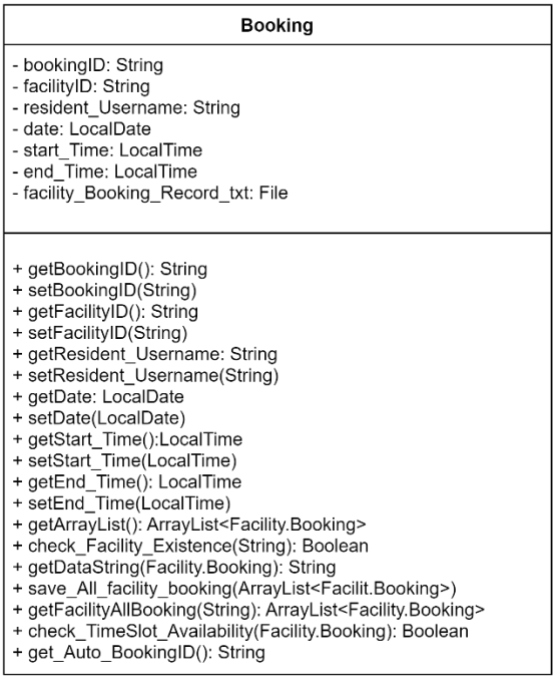
4.2.23 Unit



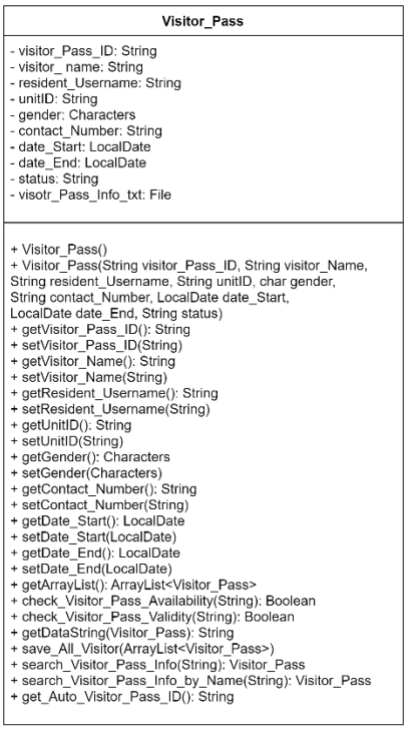
4.2.24 Facility



4.2.25 Booking



4.2.26 Visitor Pass



4.3 Use-Case Specification

4.3.1 User log in and out

Use Case Name	Login System
Use Case ID	UC-1
Description	Login to the system
Actors	Building manager, Account executive, Admin executive, Building executive, Resident
Pre-Condition	User ID and password will need to be insert
Post-Condition	Login successfully with valid user ID and password
Basic Path	1. Insert user ID and password. 2. Click “Login” button. 3. System will check the user ID and password if match in accordance with the text file. 4. “Login Successfully” message will be sent out when information entered is correct.
Alternative Path	
Exceptional Path	At basic path 3, the system displays error message to let user type an accurate user ID and password. At basic path 4, the system displays “Please try again” when information is wrong.

Use Case Name	Logout System
Use Case ID	UC-2
Description	Log out of the system
Actors	Building manager, Account executive, Admin executive, Building executive, Resident
Pre-Condition	Login to the system
Post-Condition	Click “Logout” button
Basic Path	1. Login to the system.

	2. Click the “Logout” button. 3. Send confirmation message. 4. Logout successfully.
Alternative Path	
Exceptional Path	

4.3.2 Invoice, Receipt, and Statement Management

Use Case Name	View invoice
Use Case ID	UC-3
Description	View invoice that has been issued
Actors	Account executive, Resident
Pre-Condition	Click on the invoice the user wishes to view on the account executive or resident page
Post-Condition	The system will display the details of the invoice chosen.
Basic Path	1. Click on one of the invoices displayed in the table. 2. Click on the “View invoice” button. 3. The system will the display information regarding the invoice the user clicked on.
Alternative Path	At basic path 3, users can choose to close the invoice details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Issue invoice
Use Case ID	UC-4
Description	Issue a new invoice
Actors	Account executive
Pre-Condition	Click on the “Issue invoice” button on the account executive page
Post-Condition	New invoice details written as a new row of data in the invoice text file.

Basic Path	1. Click on the “Issue invoice” button. 2. Fill in the details and click the “Issue invoice” button. 3. System will add the issued invoice into a new row of data.
Alternative Path	At basic path 3, users can choose to close the issued invoice details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	View receipt
Use Case ID	UC-5
Description	View receipt that has been issued
Actors	Account executive, Resident
Pre-Condition	Click on the receipt the user wishes to view button on the account executive page
Post-Condition	The system will display the details of the receipt chosen.
Basic Path	1. Click on one of the receipts displayed in the table. 2. Click on the “View receipt” button. 3. The system will the display information regarding the receipt the user clicked on.
Alternative Path	At basic path 3, users can choose to close the receipt details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	View statement
Use Case ID	UC-6
Description	View statement that has been issued
Actors	Account executive, Resident
Pre-Condition	Click on the statement the user wishes to view button on the account executive page
Post-Condition	The system will display the details of the financial statement chosen.

Basic Path	1. Click on one of the statements displayed in the table. 2. Click on the “View statement” button. 3. The system will the display information regarding the statement the user clicked on.
Alternative Path	At basic path 3, users can choose to close the statement details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Issue statement
Use Case ID	UC-7
Description	Issue a new statement
Actors	Account executive
Pre-Condition	Click on the “Issue statement” button on the account executive page
Post-Condition	New statement details written as a new row of data in the statement text file.
Basic Path	1. Click on the “Issue statement” button. 2. Fill in the details and click the “Issue statement” button. 3. System will add the issued statement in to a new row of data.
Alternative Path	At basic path 3, users can choose to close the issued statement details by clicking on the “Close” button.
Exceptional Path	

4.3.3 Payment Management

Use Case Name	View payment
Use Case ID	UC-8
Description	View payment that has been issued
Actors	Account executive
Pre-Condition	Click on the invoice the user wishes to view button on the account executive page
Post-Condition	Click on the "View payment" button on the account executive page
Basic Path	1. Click on one of the payments displayed in the table. 2. Click on the "View payment" button. 3. The system will the display information regarding the payment the user clicked on.
Alternative Path	At basic path 3, users can choose to close the payment details by clicking on the "Close" button.
Exceptional Path	

Use Case Name	Confirm payment
Use Case ID	UC-9
Description	Approved payment that has received
Actors	Account executive
Pre-Condition	Click on the payment status the user wishes to approve on the account executive page
Post-Condition	The system will update the payment's status from "PENDING to "RECEIVED".
Basic Path	1. Click on one of the payments' statuses displayed in the table. 2. Click on the "Confirm payment" button. 3. The system will then send a message notifying regarding the change of status.
Alternative Path	

Exceptional Path	At path 2, if the status of the payment is set to “RECEIVED”, system will notify user regarding this.

4.3.4 Pending Fee Management

Use Case Name	View pending fee
Use Case ID	UC-10
Description	View pending fee that has been issued
Actors	Account executive
Pre-Condition	Click on the pending fee the user wishes to view button on the account executive page
Post-Condition	Click on the “View pending fee” button on the account executive page
Basic Path	1. Click on one of the pending fees displayed in the table. 2. Click on the “View pending fee” button. 3. The system will the display information regarding the pending fee the user clicked on.
Alternative Path	At basic path 3, users can choose to close the pending fee details by clicking on the “Close” button.
Exceptional Path	

4.3.5 Deposit Management

Use Case Name	View deposit
Use Case ID	UC-11
Description	View deposit that has been issued
Actors	Account executive
Pre-Condition	Click on the deposit the user wishes to view button on the account executive page
Post-Condition	Click on the "View deposit" button on the account executive page
Basic Path	1. Click on one of the deposits displayed on the table. 2. Click on the "View deposit" button. 3. The system will the display information regarding the deposit the user clicked on.
Alternative Path	At basic path 3, users can choose to close the deposit details by clicking on the "Close" button.
Exceptional Path	

4.3.6 Unit Management

Use Case Name	Add new unit
Use Case ID	UC-12
Description	Add a new unit into the system
Actors	Admin executive
Pre-Condition	Click on the "Add new unit" button on the admin executive page
Post-Condition	New unit details written as a new row of data in the unit text file.
Basic Path	1. Click on the "Add new unit" button. 2. Fill in the details and click the "Add new unit" button. 3. System will add the newly added unit into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new unit by clicking on the "Close" button.

Exceptional Path	At basic path 2, if the users entered owner or resident username does not exist, the system will notify about the lack of data on the resident or owner.

Use Case Name	Modify unit info
Use Case ID	UC-13
Description	Modify unit information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Unit Info” button on the admin executive page
Post-Condition	New unit details written as will be modified into the unit text file.
Basic Path	1. Click on the “Modify Unit Info” button. 2. Fill in the details and click the “Add new unit” button. 3. System will add the modified unit onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying unit info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete unit
Use Case ID	UC-14
Description	Delete unit information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Unit” button on the admin executive page
Post-Condition	Deleted unit details will be deleted from the unit text file.
Basic Path	1. Click on one of the unit data displayed on the table. 2. Click on the “Delete Unit” button. 3. System will notify users regarding confirmation before officially deleting the unit info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting unit info details by clicking on the “Cancel” button.

Exceptional Path	

Use Case Name	View unit details
Use Case ID	UC-15
Description	View unit details that is available
Actors	Admin executive
Pre-Condition	Click on the unit the user wishes to view on the admin executive page
Post-Condition	Click on the "View unit details" button on the admin executive page
Basic Path	1. Click on one of the unit details displayed on the table. 2. Click on the "View unit details" button. 3. The system will the display information regarding the unit details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the unit details by clicking on the "Close" button.
Exceptional Path	

4.3.7 Resident Management

Use Case Name	Add new resident
Use Case ID	UC-16
Description	Add a new resident into the system
Actors	Admin executive
Pre-Condition	Click on the "Add new resident" button on the admin executive page
Post-Condition	New resident details written as a new row of data in the resident text file.
Basic Path	1. Click on the "Add new resident" button. 2. Fill in the details and click the "Add new resident" button. 3. The system will notify the user once the addition of a new resident is completed.

	4. System will add the newly added resident into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new resident by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify resident info
Use Case ID	UC-17
Description	Modify resident information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Resident Info” button on the admin executive page
Post-Condition	New resident details written as will be modified into the resident text file.
Basic Path	1. Choose any resident info that the user wish to modify. 2. Click on the “Modify Resident Info” button. 3. Fill in the details and click the “Add new resident” button. 4. System will add the modified resident onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying resident info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete resident
Use Case ID	UC-18
Description	Delete resident information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Resident” button on the admin executive page
Post-Condition	Deleted resident details will be deleted from the resident text file.
Basic Path	1. Click on one of the resident data displayed on the table.

	2. Click on the “Delete Resident” button. 3. The system will notify users regarding confirmation before officially deleting the resident info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting resident info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View resident details
Use Case ID	UC-19
Description	View resident details that is available
Actors	Admin executive
Pre-Condition	Click on the resident the user wishes to view on the admin executive page
Post-Condition	Click on the “View resident details” button on the admin executive page
Basic Path	1. Click on one of the resident details displayed on the table. 2. Click on the “View resident details” button. 3. The system will the display information regarding the resident details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the resident details by clicking on the “Close” button.
Exceptional Path	

4.3.8 Complaint Management

Use Case Name	New complaint
Use Case ID	UC-20
Description	Add a new complaint into the system
Actors	Admin executive

Pre-Condition	Click on the “Add new complaint” button on the admin executive page
Post-Condition	New complaint details written as a new row of data in the complaint text file.
Basic Path	1. Click on the “Add new complaint” button. 2. Fill in the details and click the “Add new complaint” button. 3. The system will notify the user once the data for the new complaint is added. 4. System will add the newly added complaint into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new complaint by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify complaint info
Use Case ID	UC-21
Description	Modify complaint information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Complaint Info” button on the admin executive page
Post-Condition	New complaint details written as will be modified into the complaint text file.
Basic Path	1. Choose one of the complaints the user wishes to modify. 2. Click on the “Modify Complaint Info” button. 3. Fill in the details and click the “Add new complaint” button. 4. System will add the modified complaint onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying complaint info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete complaint
Use Case ID	UC-22
Description	Delete complaint information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Complaint” button on the admin executive page
Post-Condition	Deleted complaint details will be deleted from the complaint text file.
Basic Path	1. Click on one of the complaint data displayed on the table. 2. Click on the “Delete Complaint” button. 3. The system will notify users regarding confirmation before officially deleting the complaint info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting complaint info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View complaint details
Use Case ID	UC-23
Description	View complaint details that is available
Actors	Admin executive, Building executive, Resident
Pre-Condition	Click on the complaint the user wishes to view on the resident, admin and building executive page
Post-Condition	The system will display the details of the complaint chosen.
Basic Path	1. Click on one of the complaint details displayed on the table. 2. Click on the “View complaint details” button. 3. The system will the display information regarding the complaint details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the complaint details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Complaint solved
Use Case ID	UC-24
Description	Approved complaint that has been solved
Actors	Admin executive, Building executive
Pre-Condition	Click on the complaint status the user wishes to change the status on the admin or building executive page
Post-Condition	The system will update the complaint's status from "UNSOLVED to "SOLVED".
Basic Path	1. Click on one of the complaints' statuses displayed in the table. 2. Click on the "Complaint solved" button. 3. The user will send to ask for confirmation before changing the complaint status. 4. The system will then send a message notifying regarding the change of status.
Alternative Path	At basic path 3, users can choose to not change the complaint status to "SOLVED" and click "No.
Exceptional Path	At basic path 2, if the status is originally "SOLVED", users will be notified regarding this.

4.3.9 Employee Management

Use Case Name	Add new employee
Use Case ID	UC-25
Description	Add a new employee into the system
Actors	Admin executive
Pre-Condition	Click on the "Add new employee" button on the admin executive page
Post-Condition	New employee details written as a new row of data in the employee text file.
Basic Path	1. Click on the "Add new employee" button.

	2. The user is required to add and specify the type of employee the user wishes to add. 3. Fill in the details and click the “Add new employee” button. 4. The user will be notified once the addition of a new employee is completed. 5. System will add the newly added employee into a new row of data.
Alternative Path	At basic path 4, users can choose to cancel the addition of a new employee by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Modify employee info
Use Case ID	UC-26
Description	Modify employee information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Employee Info” button on the admin executive page
Post-Condition	New employee details written will be modified into the employee text file.
Basic Path	1. Choose one of the employee info the user wishes to make changes to. 2. Click on the “Modify Employee Info” button. 3. Fill in the details and click the “Add new employee” button. 4. System will add the modified employee onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying employee info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete employee
Use Case ID	UC-27

Description	Delete employee information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Employee” button on the admin executive page
Post-Condition	Deleted employee details will be deleted from the employee text file.
Basic Path	1. Click on one of the employee data displayed on the table. 2. Click on the “Delete Employee” button. 3. The system will notify users regarding confirmation before officially deleting the employee info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting employee info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View employee details
Use Case ID	UC-28
Description	View employee details that is available
Actors	Admin executive
Pre-Condition	Click on the employee the user wishes to view on the admin executive page
Post-Condition	Click on the “View employee details” button on the admin executive page
Basic Path	1. Choose one of the employee details the user wishes to view. 2. Click on one of the employee details displayed on the table. 3. Click on the “View employee details” button. 4. The system will the display information regarding the employee details the user clicked on.
Alternative Path	At basic path 2, users can choose to close the employee details by clicking on the “Close” button.
Exceptional Path	

4.3.10 Facility Management

Use Case Name	Add new facility
Use Case ID	UC-29
Description	Add a new facility into the system
Actors	Admin executive
Pre-Condition	Click on the “Add new facility” button on the admin executive page
Post-Condition	New facility details written as a new row of data in the facility text file.
Basic Path	1. Click on the “Add new facility” button. 2. Fill in the details and click the “Add new facility” button. 3. System will add the newly added facility into a new row of data.
Alternative Path	At basic path 3, users can choose to not add a new facility by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify facility info
Use Case ID	UC-30
Description	Modify facility information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Facility Info” button on the admin executive page
Post-Condition	New facility details written as will be modified into the facility text file.
Basic Path	1. Choose one of the facility info the users wish to modify. 2. Click on the “Modify Facility Info” button. 3. Fill in the details and click the “Add new facility” button. 4. System will add the modified facility onto the rows of data.

Alternative Path	At basic path 3, users can choose to cancel modifying facility info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete facility
Use Case ID	UC-31
Description	Delete facility information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Facility” button on the admin executive page
Post-Condition	Deleted facility details will be deleted from the facility text file.
Basic Path	1. Click on one of the facility data displayed on the table. 2. Click on the “Delete Facility” button. 3. System will notify users regarding confirmation before officially deleting the facility info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting facility info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View facility details
Use Case ID	UC-32
Description	View facility details that is available
Actors	Admin executive
Pre-Condition	Click on the facility the user wishes to view on the admin executive page
Post-Condition	Click on the “View facility details” button on the admin executive page
Basic Path	1. Click on one of the facility details displayed on the table. 2. Click on the “View facility details” button.

	3. The system will the display information regarding the facility details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the facility details by clicking on the “Close” button.
Exceptional Path	

4.3.11 Facility Booking Management

Use Case Name	Make new booking
Use Case ID	UC-33
Description	Add a new facility booking into the system
Actors	Admin executive
Pre-Condition	Click on the “Add new facility booking” button on the admin executive page
Post-Condition	New facility booking details written as a new row of data in the facility booking text file.
Basic Path	1. Click on the “Add new facility booking” button. 2. Fill in the details and click the “Add new facility booking” button. 3. The system will notify the user once the addition of a new booking is added. 4. System will add the newly added facility booking into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new facility booking by clicking on the “Cancel” button.
Exceptional Path	At basic path 2, the users will be notified if the facility they entered does not exist in the system.

Use Case Name	Modify facility booking info
Use Case ID	UC-34
Description	Modify facility booking information in the system
Actors	Admin executive
Pre-Condition	Click on the "Modify Facility Booking Info" button on the admin executive page
Post-Condition	New facility booking details written as will be modified into the facility booking text file.
Basic Path	1. Choose one of the facilities booking info the user wish to make changes to. 2. Click on the "Modify Facility booking Info" button. 3. Fill in the details and click the "Add new facility booking" button. 4. System will add the modified facility booking onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying facility booking info details by clicking on the "Cancel" button.
Exceptional Path	

Use Case Name	Delete facility booking
Use Case ID	UC-35
Description	Delete facility booking information in the system
Actors	Admin executive
Pre-Condition	Click on the "Delete Facility booking" button on the admin executive page
Post-Condition	Deleted facility booking details will be deleted from the facility booking text file.
Basic Path	1. Click on one of the facilities booking data displayed on the table. 2. Click on the "Delete Facility booking" button.

	3. The system will notify users regarding confirmation before officially deleting the facility booking info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting facility booking info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View facility booking details
Use Case ID	UC-36
Description	View facility booking details that is available
Actors	Admin executive, Resident
Pre-Condition	Click on the facility booking the user wishes to view on the admin executive or resident page
Post-Condition	The system will display the details of the facility booking chosen.
Basic Path	1. Click on one of the facilities booking details displayed on the table. 2. Click on the “View facility booking details” button. 3. The system will the display information regarding the facility booking details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the facility booking details by clicking on the “Close” button.
Exceptional Path	

4.3.12 Vendor Management

Use Case Name	Add new vendor
Use Case ID	UC-37
Description	Add a new vendor into the system
Actors	Admin executive
Pre-Condition	Click on the “Add new vendor” button on the admin executive page

Post-Condition	New vendor details written as a new row of data in the vendor text file.
Basic Path	1. Click on the “Add new vendor” button. 2. Fill in the details and click the “Add new vendor” button. 3. The system will notify users regarding the addition of a new vendor. 4. System will add the newly added vendor into a new row of data.
Alternative Path	At basic path 3, users can choose not to add a new vendor by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify vendor info
Use Case ID	UC-38
Description	Modify vendor information in the system
Actors	Admin executive
Pre-Condition	Click on the “Modify Vendor Info” button on the admin executive page
Post-Condition	New vendor details written as will be modified into the vendor text file.
Basic Path	1. Choose one of the vendor info the user wish to modify. 2. Click on the “Modify Vendor Info” button. 3. Fill in the details and click the “Add new vendor” button. 4. System will add the modified vendor onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying vendor info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete vendor
Use Case ID	UC-39

Description	Delete vendor information in the system
Actors	Admin executive
Pre-Condition	Click on the “Delete Vendor” button on the admin executive page
Post-Condition	Deleted vendor details will be deleted from the vendor text file.
Basic Path	1. Click on one of the vendor data displayed on the table. 2. Click on the “Delete Vendor” button. 3. The system will notify users regarding confirmation before officially deleting the vendor info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting vendor info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View vendor details
Use Case ID	UC-40
Description	View vendor details that is available
Actors	Admin executive
Pre-Condition	Click on the vendor the user wishes to view on the admin executive page
Post-Condition	Click on the “View vendor details” button on the admin executive page
Basic Path	1. Click on one of the vendor details displayed on the table. 2. Click on the “View vendor details” button. 3. The system will the display information regarding the vendor details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the vendor details by clicking on the “Close” button.
Exceptional Path	

4.3.13 Visitor Pass Management

Use Case Name	Approve visitor pass
Use Case ID	UC-41
Description	Approved visitors pass application that has received
Actors	Admin executive
Pre-Condition	Click on the visitor pass application status the user wishes to approve on the admin executive page
Post-Condition	The system will update the visitor pass application's status from "DISAPPROVED to "APPROVED".
Basic Path	1. Click on one of the visitors pass application statuses displayed in the table. 2. Click on the "Approve visitor pass" button. 3. The system will then send a message notifying regarding the change of status.
Alternative Path	
Exceptional Path	At basic path 2, the users will be notified if the visitor pass, they chose is already approved.

Use Case Name	View visitor pass details
Use Case ID	UC-42
Description	View visitor pass that is available
Actors	Admin executive, Resident, Security Guard
Pre-Condition	Click on the visitor pass the user wishes to view on the security guard ,admin executive or resident page
Post-Condition	The system will display the details of the visitor pass chosen.
Basic Path	1. Click on one of the visitor pass details displayed on the table. 2. Click on the "View visitor pass" button. 3. The system will the display information regarding the visitor pass details the user clicked on.

Alternative Path	At basic path 3, users can choose to close the visitor pass details by clicking on the “Close” button.
Exceptional Path	

4.3.14 Employee Task Management

Use Case Name	Assign new employee task
Use Case ID	UC-43
Description	Add a new employee task into the system
Actors	Building executive
Pre-Condition	Click on the “Add new employee task” button on the building executive page
Post-Condition	New employee task details written as a new row of data in the employee task text file.
Basic Path	1. Click on the “Add new employee task” button. 2. Fill in the details and click the “Add new employee task” button. 3. The system will send a notification once the addition of a new employee task is added. 4. System will add the newly added employee task into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new employee task by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify employee task info
Use Case ID	UC-44
Description	Modify employee task information in the system
Actors	Building executive

Pre-Condition	Click on the "Modify Employee task Info" button on the building executive page
Post-Condition	New employee task details written as will be modified into the employee task text file.
Basic Path	1. Choose one of the employee task the user wish to modify. 2. Click on the "Modify Employee task Info" button. 3. Fill in the details and click the "Add new employee task" button. 4. System will add the modified employee task onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying employee task info details by clicking on the "Cancel" button.
Exceptional Path	

Use Case Name	Delete employee task
Use Case ID	UC-45
Description	Delete employee task information in the system
Actors	Building executive
Pre-Condition	Click on the "Delete Employee task" button on the building executive page
Post-Condition	Deleted employee task details will be deleted from the employee task text file.
Basic Path	1. Click on one of the employee task data displayed on the table. 2. Click on the "Delete Employee task" button. 3. The system will notify users regarding confirmation before officially deleting the employee task info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting employee task info details by clicking on the "Cancel" button.
Exceptional Path	

Use Case Name	View employee task details
Use Case ID	UC-46
Description	View employee task details that is available
Actors	Building executive
Pre-Condition	Click on the employee task the user wishes to view on the building executive page
Post-Condition	Click on the "View employee task details" button on the building executive page
Basic Path	1. Click on one of the employee task details displayed on the table. 2. Click on the "View employee task details" button. 3. The system will the display information regarding the employee task details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the employee task details by clicking on the "Close" button.
Exceptional Path	

4.3.15 Patrolling Management

Use Case Name	Set up new patrolling
Use Case ID	UC-47
Description	Add a new patrolling into the system
Actors	Building executive
Pre-Condition	Click on the "Add new patrolling" button on the building executive page
Post-Condition	New patrolling details written as a new row of data in the patrolling text file.
Basic Path	1. Click on the "Add new patrolling" button. 2. Fill in the details and click the "Add new patrolling" button.

	3. The system will then notify users once a new patrolling schedule is set up. 4. System will add the newly added patrolling into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new patrolling schedule by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Modify patrolling info
Use Case ID	UC-48
Description	Modify patrolling information in the system
Actors	Building executive
Pre-Condition	Click on the “Modify Patrolling Info” button on the building executive page
Post-Condition	New patrolling details written will be modified into the patrolling text file.
Basic Path	1. Choose one of the patrolling info the user wishes to modify. 2. Click on the “Modify Patrolling Info” button. 3. Fill in the details and click the “Add new patrolling” button. 4. System will add the modified patrolling onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying patrolling info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete patrolling
Use Case ID	UC-49
Description	Delete patrolling information in the system
Actors	Building executive
Pre-Condition	Click on the “Delete Patrolling” button on the building executive page

Post-Condition	Deleted patrolling details will be deleted from the patrolling text file.
Basic Path	1. Click on one of the patrolling data displayed on the table. 2. Click on the “Delete Patrolling” button. 3. The system will notify users regarding confirmation before officially deleting the patrolling info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting patrolling info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View patrolling details
Use Case ID	UC-50
Description	View patrolling details that is available
Actors	Building executive
Pre-Condition	Click on the patrolling the user wishes to view on the building executive page
Post-Condition	Click on the “View patrolling details” button on the building executive page
Basic Path	1. Click on one of the patrolling details displayed on the table. 2. Click on the “View patrolling details” button. 3. The system will the display information regarding the patrolling details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the patrolling details by clicking on the “Close” button.
Exceptional Path	

4.3.16 Checkpoint Management

Use Case Name	Add new checkpoint
Use Case ID	UC-51
Description	Add a new checkpoint into the system
Actors	Building executive
Pre-Condition	Click on the "Add new checkpoint" button on the building executive page
Post-Condition	New checkpoint details written as a new row of data in the checkpoint text file.
Basic Path	1. Click on the "Add new checkpoint" button. 2. Fill in the details and click the "Add new checkpoint" button. 3. System will add the newly added checkpoint into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new checkpoint by clicking on the "Cancel" button.
Exceptional Path	

Use Case Name	Modify checkpoint info
Use Case ID	UC-52
Description	Modify checkpoint information in the system
Actors	Building executive
Pre-Condition	Click on the "Modify Checkpoint Info" button on the building executive page
Post-Condition	New checkpoint details written as will be modified into the checkpoint text file.
Basic Path	1. Click on the "Modify Checkpoint Info" button. 2. Fill in the details and click the "Add new checkpoint" button. 3. System will add the modified checkpoint onto the rows of data.

Alternative Path	At basic path 3, users can choose to cancel modifying checkpoint info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete checkpoint
Use Case ID	UC-53
Description	Delete checkpoint information in the system
Actors	Building executive
Pre-Condition	Click on the “Delete Checkpoint” button on the building executive page
Post-Condition	Deleted checkpoint details will be deleted from the checkpoint text file.
Basic Path	1. Click on one of the checkpoint data displayed on the table. 2. Click on the “Delete Checkpoint” button. 3. The system will notify users regarding confirmation before officially deleting the checkpoint info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting checkpoint info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View checkpoint details
Use Case ID	UC-54
Description	View checkpoint details that is available
Actors	Building executive
Pre-Condition	Click on the checkpoint the user wishes to view on the building executive page
Post-Condition	The system will display the chosen checkpoint details.
Basic Path	1. Click on one of the checkpoint details displayed on the table. 2. Click on the “View checkpoint details” button.

	3. The system will the display information regarding the checkpoint details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the checkpoint details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	View checkpoint record
Use Case ID	UC-55
Description	View checkpoint record that is available
Actors	Building executive
Pre-Condition	Click on the checkpoint record the user wishes to view on the building executive page
Post-Condition	The system will display the details regarding the checkpoint record the users chose.
Basic Path	1. Click on one of the checkpoint records displayed on the table. 2. Click on the “View checkpoint record” button. 3. The system will the display information regarding the checkpoint record the user clicked on.
Alternative Path	At basic path 3, users can choose to close the checkpoint record by clicking on the “Close” button.
Exceptional Path	

4.3.17 Job Report Management

Use Case Name	View entry record
Use Case ID	UC-56
Description	View entry record that is available
Actors	Building executive
Pre-Condition	Click on the entry record the user wishes to view on the building executive page
Post-Condition	The system will display the details regarding the entry record the users chose.
Basic Path	1. Click on one of the entry records displayed on the table. 2. Click on the "View entry record" button. 3. The system will the display information regarding the entry record the user clicked on.
Alternative Path	At basic path 3, users can choose to close the entry record by clicking on the "Close" button.
Exceptional Path	

Use Case Name	View incident report
Use Case ID	UC-57
Description	View incident report that is available
Actors	Building executive
Pre-Condition	Click on the incident report the user wishes to view on the building executive page
Post-Condition	Click on the "View incident report" button on the building executive page
Basic Path	1. Click on one of the incident reports displayed on the table. 2. Click on the "View incident report" button. 3. The system will the display information regarding the incident report the user clicked on.

Alternative Path	At basic path 3, users can choose to close the incident report by clicking on the “Close” button.
Exceptional Path	

Use Case Name	View report
Use Case ID	UC-58
Description	View report details that is available
Actors	Building manager
Pre-Condition	Click on the report the report wishes to view on the building manager page
Post-Condition	The system will display the report the user wishes to view.
Basic Path	1. Click on one of the report details displayed on the table. 2. Click on the “View report details” button. 3. The system will the display information regarding the report details the report clicked on.
Alternative Path	At basic path 3, reports can choose to close the report details by clicking on the “Close” button.
Exceptional Path	

4.3.18 User Management

Use Case Name	Add new user
Use Case ID	UC-59
Description	Add a new user into the system
Actors	Building manager
Pre-Condition	Click on the “Add new user” button on the building manager page
Post-Condition	New user details written as a new row of data in the user text file.
Basic Path	1. Click on the “Add new user” button. 2. Fill in the details and click the “Add new user” button.

	3. The system will then notify the user regarding the addition of a new user. 4. System will add the newly added user into a new row of data.
Alternative Path	At basic path 3, users can choose to not add a new user detail by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Modify user info
Use Case ID	UC-60
Description	Modify user information in the system
Actors	Building manager
Pre-Condition	Click on the “Modify User Info” button on the building manager page
Post-Condition	New user details written as will be modified into the user text file.
Basic Path	1. Choose one of the user, the user wishes to modify. 2. Click on the “Modify User Info” button. 3. Fill in the details and click the “Add new user” button. 4. System will add the modified user onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying user info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete user
Use Case ID	UC-61
Description	Delete user information in the system
Actors	Building manager
Pre-Condition	Click on the “Delete User” button on the building manager page
Post-Condition	Deleted user details will be deleted from the user text file.
Basic Path	1. Click on one of the user data displayed on the table. 2. Click on the “Delete User” button.

	3. The system will notify users regarding confirmation before officially deleting the user info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting user info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View user details
Use Case ID	UC-62
Description	View user details that is available
Actors	Building manager
Pre-Condition	Click on the user the user wishes to view on the building manager page
Post-Condition	The system will display details on regarding the user that the user chosen.
Basic Path	1. Click on one of the user details displayed on the table. 2. Click on the “View user details” button. 3. The system will the display information regarding the user details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the user details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Search user
Use Case ID	UC-63
Description	Search user details that is available
Actors	Building manager
Pre-Condition	Click on the search bar
Post-Condition	User data will be displayed
Basic Path	1. Click on the search bar.

	2. Type in the username for the user wants to find. 3. Click on which user the user wants to search for. 4. The system will the display information regarding the user details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the user details by clicking on the "Close" button.
Exceptional Path	

4.3.19 Operation Budgeting Management

Use Case Name	Add operation budgeting
Use Case ID	UC-64
Description	Add a new operation budgeting into the system
Actors	Building manager
Pre-Condition	Click on the "Add new operation budgeting" button on the building manager page
Post-Condition	New operation budgeting details written as a new row of data in the operation budgeting text file.
Basic Path	1. Click on the "Add new operation budgeting" button. 2. Fill in the details and click the "Add new operation budgeting" button. 3. System will add the newly added operation budgeting into a new row of data.
Alternative Path	At basic path 3, operation budgeting can choose to cancel the addition of a new operation budgeting details by clicking on the "Close" button.
Exceptional Path	

Use Case Name	Modify operation budgeting info
Use Case ID	UC-65

Description	Modify operation budgeting information in the system
Actors	Building manager
Pre-Condition	Click on the "Modify Operation budgeting Info" button on the building manager page
Post-Condition	New operation budgeting details written as will be modified into the operation budgeting text file.
Basic Path	1. Choose one of the operation budgeting, the user wishes to modify. 2. Click on the "Modify Operation budgeting Info" button. 3. Fill in the details and click the "Add new operation budgeting" button. 4. System will add the modified operation budgeting onto the rows of data.
Alternative Path	At basic path 3, operation budgeting can choose to cancel modifying operation budgeting info details by clicking on the "Cancel" button.
Exceptional Path	

Use Case Name	Delete operation budgeting
Use Case ID	UC-66
Description	Delete operation budgeting information in the system
Actors	Building manager
Pre-Condition	Click on the "Delete Operation budgeting" button on the building manager page
Post-Condition	Deleted operation budgeting details will be deleted from the operation budgeting text file.
Basic Path	1. Click on one of the operation budgeting data displayed on the table. 2. Click on the "Delete Operation budgeting" button.

	3. The system will notify operation budgeting regarding confirmation before officially deleting the operation budgeting info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, operation budgeting can choose to cancel deleting operation budgeting info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View operation budgeting details
Use Case ID	UC-67
Description	View operation budgeting details that is available
Actors	Building manager
Pre-Condition	Click on the operation budgeting the operation budgeting wishes to view on the building manager page
Post-Condition	The system will display the operation budgeting information the user wishes to view.
Basic Path	1. Click on one of the operation budgeting details displayed on the table. 2. Click on the “View operation budgeting details” button. 3. The system will the display information regarding the operation budgeting details.
Alternative Path	At basic path 3, operation budgeting can choose to close the operation budgeting details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Fund operation
Use Case ID	UC-68
Description	Fund operation budgeting details that is available
Actors	Building manager

Pre-Condition	Click on the operation budgeting the operation budgeting wishes to fund on the building manager page
Post-Condition	The system will make payment to the operation budgeting the user wishes to fund.
Basic Path	1. Click on one of the operation budgeting details displayed on the table. 2. Click on the "Fund operation" button. 3. The system will notify user once the transaction is completed. 4. The system will the display information regarding the change of operation budgeting details.
Alternative Path	At basic path 3, operation budgeting can choose to close the operation budgeting details by clicking on the "Close" button.
Exceptional Path	

4.3.20 Team Leader Management

Use Case Name	Add team leader
Use Case ID	UC-69
Description	Add a new team leader into the system
Actors	Building manager
Pre-Condition	Click on the "Add new team leader" button on the building manager page
Post-Condition	New team leader details written as a new row of data in the team leader text file.
Basic Path	1. Click on the "Add new team leader" button. 2. Fill in the details and click the "Add new team leader" button. 3. System will add the newly added team leader into a new row of data.
Alternative Path	At basic path 3, team leader can choose to cancel the addition of a new team leader by clicking on the "Cancel" button.

Exceptional Path	

Use Case Name	Modify team leader info
Use Case ID	UC-70
Description	Modify team leader information in the system
Actors	Building manager
Pre-Condition	Click on the “Modify Team leader Info” button on the building manager page
Post-Condition	The system will display the information regarding the team leader the user wishes to see.
Basic Path	1. Click on the “Modify Team leader Info” button. 2. Fill in the details and click the “Add new team leader” button. 3. System will add the modified team leader onto the rows of data.
Alternative Path	At basic path 3, team leader can choose to cancel modifying team leader info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Delete team leader
Use Case ID	UC-71
Description	Delete team leader information in the system
Actors	Building manager
Pre-Condition	Click on the “Delete Team leader” button on the building manager page
Post-Condition	Deleted team leader details will be deleted from the team leader text file.
Basic Path	1. Click on one of the team leader data displayed on the table. 2. Click on the “Delete Team leader” button. 3. The system will notify team leader regarding confirmation before officially deleting the team leader info.

	4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, team leader can choose to cancel deleting team leader info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View team leader details
Use Case ID	UC-72
Description	View team leader details that is available
Actors	Building manager
Pre-Condition	Click on the team leader the team leader wishes to view on the building manager page
Post-Condition	The system will the display the information regarding the team leader that the user chose.
Basic Path	1. Click on one of the team leader details displayed on the table. 2. Click on the “View team leader details” button. 3. The system will the display information regarding the team leader details the team leader clicked on.
Alternative Path	At basic path 3, team leader can choose to close the team leader details by clicking on the “Close” button.
Exceptional Path	

4.3.21 Security Guard Functionalities

Use Case Name	Update visitor pass info
Use Case ID	UC-73
Description	Update visitor pass information in the system
Actors	Security Guard
Pre-Condition	Click on the “Update Visitor pass Info” button on the security guard page

Post-Condition	New visitor pass details written as will be modified into the visitor pass text file.
Basic Path	1. Click on the “Update Visitor pass Info” button. 2. Fill in the details and click the “Update new visitor pass” button. 3. System will add the modified visitor pass onto the rows of data.
Alternative Path	At basic path 3, visitor pass can choose to cancel modifying visitor pass info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Check-in Check Point
Use Case ID	UC-74
Description	Checking in at a checkpoint
Actors	Security Guard
Pre-Condition	Click on the “Check-In Check Point” button on the security guard page
Post-Condition	Checkpoint information will then be updated.
Basic Path	1. Click on the “Check-In Check Point” button. 2. Click on the checkpoint user wants to check in to. 3. System will notify once the user has check-in to the specific checkpoint.
Alternative Path	At basic path 2, visitor pass can choose to cancel modifying visitor pass info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	View checkpoint details
Use Case ID	UC-75
Description	View checkpoint details that is available
Actors	Security Guard

Pre-Condition	Click on the checkpoint details the user wishes to view on the security guard page
Post-Condition	Click on the “View checkpoint details” button on the security guard page
Basic Path	1. Click on one of the checkpoint details displayed on the table. 2. Click on the “View checkpoint details” button. 3. The system will the display information regarding the checkpoint details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the checkpoint details by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Record incident
Use Case ID	UC-76
Description	Add a new incident into the system
Actors	Security guard
Pre-Condition	Click on the “Record incident” button on the security guard page
Post-Condition	New incident details written as a new row of data in the incident text file.
Basic Path	1. Click on the “Record incident” button. 2. Fill in the details and click the “Record incident” button. 3. System will add the newly recorded incident into a new row of data.
Alternative Path	At basic path 3, incident can choose to cancel the addition of a new incident by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Update incident info
Use Case ID	UC-77
Description	Update incident information in the system

Actors	Security guard
Pre-Condition	Click on the “Update incident” button on the security guard page
Post-Condition	New incident details written as will be modified into the incident text file.
Basic Path	1. Click on the “Update incident Info” button. 2. Fill in the details and click the “Update incident” button. 3. System will add the modified incident onto the rows of data.
Alternative Path	At basic path 3, incident can choose to cancel modifying incident info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Check incident details
Use Case ID	UC-78
Description	Check incident details that is available
Actors	Security Guard
Pre-Condition	Click on the incident details the user wishes to view on the security guard page
Post-Condition	Users can view details of the incident they wish to check.
Basic Path	1. Click on one of the incident details displayed on the table. 2. Click on the “Check incident details” button. 3. The system will the display information regarding the checkpoint details the user clicked on.
Alternative Path	At basic path 3, users can choose to close the incident details by clicking on the “Close” button.
Exceptional Path	

4.3.22 Resident Functionalities

Use Case Name	Make payment
Use Case ID	UC-79
Description	Make payments that needs to be made
Actors	Resident
Pre-Condition	Click on the payment that the user wishes to pay on the resident page
Post-Condition	Payment status will be updated
Basic Path	1. Click on one of the payment details displayed on the table. 2. Click on the “Make payment” button. 3. The system will notify the user of the payment made. 4. The payment status will change from “UNPAID” to ‘PAID”.
Alternative Path	At basic path 3, users can choose to close the system notification by clicking on the “ok” button.
Exceptional Path	

Use Case Name	Check deposit payment
Use Case ID	UC-80
Description	Check deposit payment details that is required to pay
Actors	Resident
Pre-Condition	Click on the “Check deposit payment” button on resident page
Post-Condition	Users can view details of the deposit payment they wish to check.
Basic Path	1. Click on the “Check deposit payment” button. 2. Payment will be made immediately. 3. The system will then notify users regarding the payment they made.
Alternative Path	At basic path 2, users can choose to not make payment by clicking on the “Close” button.
Exceptional Path	

Use Case Name	Book facility
Use Case ID	UC-81
Description	Make a new facility booking into the system
Actors	Resident
Pre-Condition	Click on the “Book facility” button on the resident page
Post-Condition	The system will update the details of the facility booking.
Basic Path	1. Click on the “Book facility” button. 2. Fill in the details and click the “Book facility” button. 3. System will add the newly added facility booking into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel the addition of a new facility booking by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Update booking info
Use Case ID	UC-82
Description	Update facility booking information in the system
Actors	Resident
Pre-Condition	Click on the “Update booking” button on the resident page
Post-Condition	New facility booking details written will be modified into the facility booking text file.
Basic Path	1. Click on the “Update booking” button. 2. Fill in the details and click the “Update booking” button. 3. System will add the modified facility booking onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel modifying facility booking info details by clicking on the “Cancel” button.
Exceptional Path	At basic path 2, if the users select a time slot that is booked, they will be notified regarding this by the system.

Use Case Name	Cancel booking
Use Case ID	UC-83
Description	Cancel booking information in the system
Actors	Resident
Pre-Condition	Click on the “Cancel booking” button on the resident page
Post-Condition	Cancelled facility booking details will be deleted from the facility booking text file.
Basic Path	1. Click on one of the facilities booking data displayed on the table. 2. Click on the “Cancel booking” button. 3. The system will notify users regarding confirmation before officially deleting the facility booking info. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting facility booking info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Apply visitor pass
Use Case ID	UC-84
Description	Apply for visitor pass to be added to the system
Actors	Resident
Pre-Condition	Click on the “Apply visitor pass” button on the resident page
Post-Condition	New visitor pass details written as a new row of data in the visitor pass text file waiting to be approved.
Basic Path	1. Click on the “Apply visitor pass” button. 2. Fill in the details and click the “Apply visitor pass” button. 3. System will add the new application of visitor pas into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel applying for visitor pass details by clicking on the “Cancel” button.

Exceptional Path	

Use Case Name	Update visitor pass
Use Case ID	UC-85
Description	Update visitor pass information in the system
Actors	Resident
Pre-Condition	Click on the “Update visitor pass” button on the resident page
Post-Condition	Updated visitor pass details written as will be modified into the facility booking text file.
Basic Path	1. Choose which visitor passes the user wish to update. 2. Click on the “Update visitor pass” button. 3. Fill in the details and click the “Update visitor pass” button. 4. System will add the modified visitor pass onto the rows of data.
Alternative Path	At basic path 3, users can choose to cancel updating visitor pass info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Cancel visitor pass
Use Case ID	UC-86
Description	Cancel visitor pass in the system
Actors	Resident
Pre-Condition	Click on the “Cancel visitor pass” button on the resident page
Post-Condition	Cancelled visitor pass details will be deleted from the visitor pass text file.
Basic Path	1. Click on one of the visitor data displayed on the table. 2. Click on the “Cancel visitor pass” button. 3. The system will notify users regarding confirmation before officially deleting the visitor pass. 4. System will delete the info from the rows of data.

Alternative Path	At basic path 3, users can choose to cancel deleting visitor pass details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Log complaint
Use Case ID	UC-87
Description	Log complaint regarding anything that bothers the users
Actors	Resident
Pre-Condition	Click on the “Log complaint” button on the resident page
Post-Condition	New complaint details written as a new row of data in the complaint text file.
Basic Path	1. Click on the “Log complaint” button. 2. Fill in the details and click the “Log complaint” button. 3. System will add the new complaint into a new row of data.
Alternative Path	At basic path 3, users can choose to cancel logging complaint details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Update complaint
Use Case ID	UC-88
Description	Update complaint information in the system
Actors	Resident
Pre-Condition	Click on the “Update complaint” button on the resident page
Post-Condition	Updated complaint details written as will be modified into the complaint text file.
Basic Path	1. Choose one of the complaint the user wish to update. 2. Click on the “Update complaint” button. 3. Fill in the details and click the “Update complaint” button. 4. System will add the modified complaint onto the rows of data.

Alternative Path	At basic path 3, users can choose to cancel updating complaint info details by clicking on the “Cancel” button.
Exceptional Path	

Use Case Name	Cancel complaint
Use Case ID	UC-89
Description	Cancel complaint in the system
Actors	Resident
Pre-Condition	Click on the “Cancel complaint” button on the resident page
Post-Condition	Cancelled complaint details will be deleted from the complaint text file.
Basic Path	1. Click on one of the complaints displayed on the table. 2. Click on the “Cancel complaint” button. 3. The system will notify users regarding confirmation before officially deleting the complaint. 4. System will delete the info from the rows of data.
Alternative Path	At basic path 3, users can choose to cancel deleting complaint details by clicking on the “Cancel” button.
Exceptional Path	

5 Object-Oriented Concepts

5.1 Attribute

Attributes, in OOP, are an essential element of classes and instances that define the data a class can hold. The attributes encapsulate the internal state of an object and represent its characteristics or properties. They are typically declared within a class and can hold various possible values that class instances may have. The code snippet illustrates how attributes can be defined in a class to represent a Patrolling Schedule object. (Agha, 1990)

```
private String patrolID;
    private String employeeID;
    private String day;
    private LocalTime start_Time;
    private LocalTime end_Time;
    private final File patrolling_Schedule_Info_txt = new File("src/Database/Patrolling_Schedule_Information.txt");
```

In the given code, the class "Patrolling" includes attributes such as 'patrolID', 'employeeID', 'day', 'start_Time', 'end_Time', and 'patrolling_Schedule_Info_txt'. Each attribute has a specific purpose in representing a particular property of the Patrolling Schedule object. For instance, the 'patrolID' attribute represents the unique identification of the Patrolling Schedule, while the 'employeeID' attribute represents the employee responsible for the patrolling.

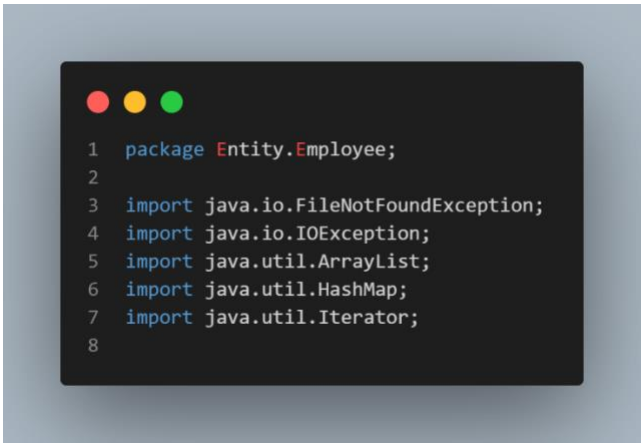
In addition, 'day' represents the day of the week for which the patrolling is scheduled, while 'start_Time' and 'end_Time' represent the time interval for the patrolling. The 'patrolling_Schedule_Info_txt' attribute points to the file location that contains the information about patrolling schedules, which can be accessed using the appropriate getter and setter methods.

In summary, the attributes in the 'Patrolling' class define and represent the internal state of a Patrolling Schedule object, which is essential for the system to function correctly. Properly defined attributes within a class provide a reliable and efficient way to manage data and ensure the integrity of the object-oriented design.

5.2 Classes

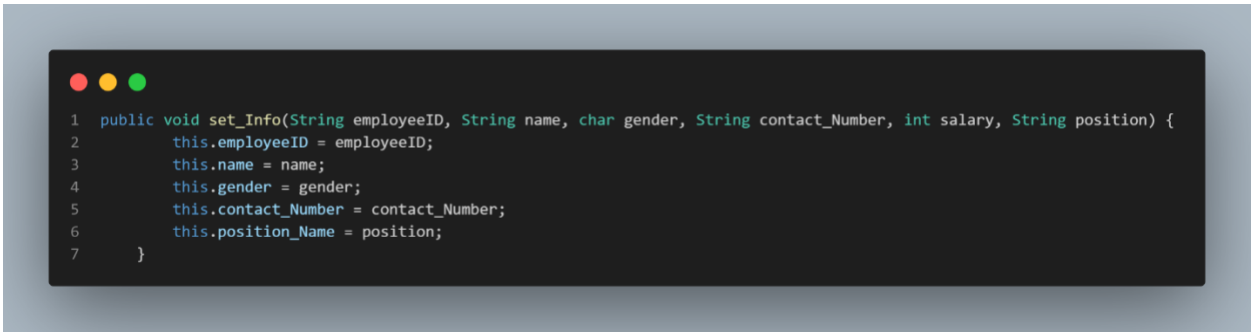
In OOP, a class is a framework or outline that defines the behavior or state for creating objects. It defines the properties, methods, and behaviors that objects of that class will have. Classes are

applied to build things by serving as models. A class is described as a combination of elements that are logical quantities such as fields, methods, and constructors. These elements are the building blocks of a class and are used to define the behavior and state of objects created from the class. (Agha, 1990)



```java
package Entity.Employee;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

```

```java
public clas Employee {
    protected String employeeID;
    protected String name;
    protected char gender;
    protected String contact_Number;
    protected int salary;
    protected String position_Name;

    public Employee() {
    }
```



```java
public void set_Info(String employeeID, String name, char gender, String contact_Number, int salary, String position) {
        this.employeeID = employeeID;
        this.name = name;
        this.gender = gender;
        this.contact_Number = contact_Number;
        this.position_Name = position;
    }
```

To demonstrate this concept, look closely at the Property Management System built for Parkhill Residence. Among the many classes created, one is called "Employee," representing the various employees working in Parkhill Residence. The Property Management System is based on this class.

The "Employee" class is a framework that provides a blueprint for creating and managing employee objects in the system. It has attributes such as employeeID, name, gender, contact number, salary, and position name, which store relevant information about each employee. These attributes can have unique values for each employee object created in the system, allowing for more precise tracking and management of employee-related data.

The class is designed to be flexible and scalable, and it can be customized to meet specific business needs. For instance, the set_Info method can be used to set specific values for the attributes of the Employee object, making it easier to create new employees and update their information as necessary. Additionally, the class can be expanded by adding more methods and fields to enhance its functionality.

In summary, the Employee class is an example of how classes can be used to automate and streamline various processes in a company. By providing a framework for creating and managing employee objects, the class helps reduce errors, increase efficiency, and improve data accuracy, which can significantly impact a business's overall success.

5.3 Object

Objects are groups of actions that have the same state. As is well known, everything is an object, and each comprises Attributes and Actions. Objects formed from a class in any programming language are always referred to as instances of that class. They have states as well as behaviors. These things are always connected to real-world objects, i.e., actual humans. As a consequence, they are frequently referred to as global run-time entities. These self-contained units include methods and attributes that allow data to be used. Physical and logical data, for example, may both be objects. It stores addresses and consumes memory space. (Agha, 1990)

```java
public String get_Auto_InvoiceID() throws FileNotFoundException {
    FileReader fileReader = new FileReader(this.invoice_txt);
    Scanner scanner = new Scanner(fileReader);
    scanner.nextLine();

    Integer num;
    for(num = 0; scanner.hasNextLine(); num = num + 1) {
        String[] data = scanner.nextLine().split(":", 8);
        String number = data[0].substring(3);
        num = Integer.parseInt(number);
    }

    String str = "INV" + num.toString();
    System.out.println(str);
    return str;
}
```

The code provided is an example of OOP, specifically using an object in Java. The method get_Auto_InvoiceID() creates an object of the FileReader class, which reads data from a file. Additionally, an object of the Scanner class is created to parse the data read from the file. The variables num and str are objects of the Integer and String classes, respectively.

This highlights the versatility and power of objects in OOP, as they allow for creating complex systems by breaking down significant problems into smaller, more manageable components. In addition, encapsulating data and functionality within objects allows code to be organized and reused more efficiently, leading to more maintainable and scalable software systems.

5.4 Operation (Method)

In OOP, an operation is a function that can be called on an object to modify its behavior. It is a crucial concept in OOP because it allows developers to create dynamic and interactive programs responding to user input and other external events. Each operation is assigned a signature, which defines the parameters that can be used. An object may have no operations, one operation, or many operations.

```
1  public String getName() {  return this.name;   }
2
3  public void setName(String name) { this.name = name; }
4
5  public char getGender() { return this.gender; }
6
7  public void setGender(char gender) { this.gender = gender; }
8
9  public String getContact_Number() { return this.contact_Number; }
10
11 public void setContact_Number(String contact_Number) { this.contact_Number = contact_Number; }
12
13 public String getPosition() { return this.position; }
14
15 public void setPosition(String position) { this.position = position; }
16
```

In our code, we have implemented various operations in different parts of the program to provide a way to manipulate the data stored in an object. For example, the code snippet above shows a series of methods used to get and set various attributes of an executive object, such as ExecutiveID, Name, Gender, Contact_Number, and Position.

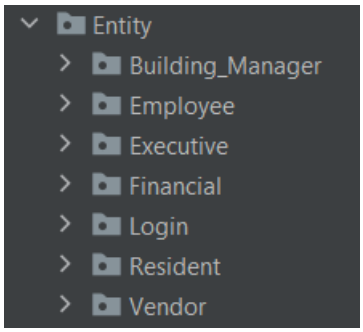
The setExecutiveID method sets the executive's ID, while the getName method returns the executive's name. Similarly, setContact_Number sets the executive's contact number, and get gender returns the executive's gender. These operations allow for accessing and modifying the attributes of an executive object, providing greater control over its properties and behavior.

By using operations to manipulate the data stored in an object, we can create more dynamic and interactive programs that can respond to user input and other external events. These operations enable us to interact with the data in an executive object, making it easier to manage and control.

By providing a way to manipulate the data stored in an object, we can create programs that are more flexible and adaptable to changing conditions.

5.5 Package

A package in OOP has the primary function of facilitating the grouping of fundamental elements. A model element may include additional model elements inside a package it creates, making it easier to organize and manage the software system. Typically, the model presently being developed is organized into a package. A package also serves as the configuration management unit, allowing the developers to specify which files belong to a package and how they relate to each other.



When we write "package Entity;" in our code, we identify the package to which the Java class belongs. This line helps organize classes related to each other and create a namespace for them. For instance, the "Entity" package identifies that the Java class in this file belongs to the Entity package. Therefore, other classes within the same package can access the class, but those outside the package cannot unless they are imported.

In summary, the package statement is a fundamental aspect of OOP that enables developers to organize their code more structurally, making it easier to manage and maintain large-scale software systems. Furthermore, by using packages, developers can create a logical grouping of classes, functions, and methods that can be used together, simplifying software systems' development and maintenance. Thus, the use of packages is highly recommended in OOP.

6 Object-Oriented Principles

6.1 Modularity

Modularity is a crucial principle in OOP that involves breaking a system down into smaller, independent parts that can be developed, tested, and maintained separately. Modularity aims to simplify the development process and make it easier to manage complex systems. In modular design, each component is designed to perform a specific function and can be tested and modified independently of other components.

Implementation

```
public Team_Leader search_Team_Leader_Information(String leader_Username) throws FileNotFoundException {
    ArrayList<Team_Leader> leaderArrayList = this.getArrayList();
    Team_Leader result = new Team_Leader();
    result.setLeader_Username("0");
    Iterator var4 = leaderArrayList.iterator();

    while(true) {
        Team_Leader leader;
        do {
            if (!var4.hasNext()) {
                return result;
            }

            leader = (Team_Leader)var4.next();
        } while(!leader.getLeader_Username().equals(leader_Username) && !leader.getTeam().equals(leader_Username));

        result = leader;
    }
}
```

```
public boolean check_Team_Leader_Availability(String team, String position) throws FileNotFoundException {
    boolean result = false;
    ArrayList<Team_Leader> leaderArrayList = this.getArrayList();
    Iterator var5 = leaderArrayList.iterator();

    while(var5.hasNext()) {
        Team_Leader leader1 = (Team_Leader)var5.next();
        if (leader1.getTeam().equals(team) && leader1.getPosition().equals(position)) {
            result = true;
        }
    }
```

In the provided code, modularity is implemented through static methods in the Team_Leader class. The two methods, search_Team_Leader_Information and check_Team_Leader_Availability, are designed to perform specific tasks related to searching for information on team leaders and checking their availability, respectively.

By encapsulating these tasks in their methods, the Team_Leader class becomes more modular and easier to maintain. For example, if the logic for searching for team leader information needs to be updated, it can be done within the search_Team_Leader_Information method without affecting the rest of the code that uses the Team_Leader class.

Additionally, using exceptions in the method signatures, such as FileNotFoundException, allows for better error handling and easier debugging. This helps to further improve the modularity of the code by separating error handling from the main logic of the methods.

In summary, the use of static methods and exception handling in the Team_Leader class demonstrates the implementation of modularity by separating tasks into their methods and improving the maintainability and error handling of the code.

6.2 Abstraction

Abstraction is a fundamental concept in OOP that serves to present only the necessary information while hiding irrelevant details. The primary goal of abstraction is to provide a simplified interface for the user while concealing the inner workings of an object. This approach helps to reduce programming complexity and effort by presenting only the essential features of an object. In OOP, abstract classes and methods are used to achieve this goal. Abstract classes define one or more abstract methods with a method signature but no implementation. These methods are helpful when multiple subclasses perform a similar activity in different ways and with different implementations. (Wegner, 1990)

Implementation

To demonstrate the implementation of abstraction, consider the following code snippet:

```java
public ArrayList<Statement> getArrayList() throws FileNotFoundException {
    ArrayList<Statement> statementArrayList = new ArrayList();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM.dd.yyyy");
    Scanner scanner = new Scanner(this.statement_txt);
    scanner.nextLine();

    while(scanner.hasNextLine()) {
        String[] data = scanner.nextLine().split(":", 5);
        Statement statement = new Statement(data[0], data[1], LocalDate.parse(data[2], formatter), data[3], data[4]);
        statementArrayList.add(statement);
    }

    scanner.close();
    return statementArrayList;
}
```

In the provided code, modularity is implemented through static methods in the Team_Leader class. The two methods, search_Team_Leader_Information, and check_Team_Leader_Availability, are designed to perform specific tasks related to searching for information on team leaders and checking their availability, respectively.

By encapsulating these tasks in their methods, the Team_Leader class becomes more modular and easier to maintain. For example, if the logic for searching for team leader information needs to be updated, it can be done within the search_Team_Leader_Information method without affecting the rest of the code that uses the Team_Leader class. Additionally, using exceptions in the method signatures, such as FileNotFoundException, allows for better error handling and easier debugging.

This helps to further improve the modularity of the code by separating error handling from the main logic of the methods.

In summary, the use of static methods and exception handling in the Team_Leader class demonstrates the implementation of modularity by separating tasks into their methods and improving the maintainability and error handling of the code.

6.3 Encapsulation

Encapsulation is a key concept in OOP that combines data and code into a single entity shielded from outside interference and abuse. This approach conceals the data from other classes, and only the current class's methods can access it. This technique is known as data hiding. In addition, encapsulation provides a layer of protection that helps prevent program crashes by stopping the modification of individual variables and data during program execution. Encapsulation is accomplished by declaring variables as private and providing setter and getter methods as public to modify and retrieve variable values while keeping them secure.

Implementation

Encapsulation can be implemented in many ways. The most common method involves declaring variables as private and providing getter and setter methods as public to modify and retrieve variable values while keeping them secure. The provided code demonstrates encapsulation by using private instance variables and public getter and setter methods to access and modify those variables.

```java
public String getInvoiceID() { return this.invoiceID; }

public void setInvoiceID(String invoiceID) { this.invoiceID = invoiceID; }

public String getIssuerID() { return this.issuerID; }

public void setIssuerID(String issuerID) { this.issuerID = issuerID; }

public String getUnitID() { return this.unitID; }

public void setUnitID(String unitID) { this.unitID = unitID; }

public int getAmount() { return this.amount; }

public void setAmount(int amount) { this.amount = amount; }

public String getInvoiceID() { return this.invoiceID; }

public void setInvoiceID(String invoiceID) { this.invoiceID = invoiceID; }

public String getIssuerID() { return this.issuerID; }

public void setIssuerID(String issuerID) { this.issuerID = issuerID; }

public String getUnitID() { return this.unitID; }

public void setUnitID(String unitID) { this.unitID = unitID; }

public int getAmount() { return this.amount; }

public void setAmount(int amount) { this.amount = amount; }

public LocalDate getDueDate() { return this.dueDate; }

public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }

public LocalDate getDueDate() { return this.dueDate; }

public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }

```

The Invoice class utilizes private instance variables, such as invoiceID, issuerID, unitID, amount, dueDate, paymentTypes, description, and status, which cannot be directly accessed or modified from outside the class. Instead, public getter and setter methods are provided for each variable to

allow controlled access and modification. For example, the getInvoiceID() method returns the value of the invoiceID variable, while the setInvoiceID(String invoiceID) method sets the value of invoiceID to the provided string. Similarly, the other methods return the values of their respective private instance variables, while the corresponding set methods set the values of those variables to the provided values.

In summary, encapsulation is a crucial concept in OOP that helps maintain code quality, security, and flexibility. It can be implemented using private instance variables and public getter and setter methods, which provide controlled access to the data and code while keeping them secure. However, encapsulation has advantages and disadvantages, and developers must weigh these factors when deciding whether to use it in their programs.

6.4 Inheritance

In Java, inheritance is a feature that allows a class to obtain another class's properties, including the variables and methods. This enables the creation of hierarchical relationships between classes by allowing one object to inherit the attributes of another object. Inheritance is a simple yet powerful concept, allowing developers to reuse existing code in a previous class. Instead of creating new code from scratch, developers can create a new class that inherits the existing class and uses all its inherited properties and functions. (Wegner, 1990)

Implementation

```
public class Building_Executive_Function {
    public Building_Executive_Function() {
    }

    public static class Building_Executive extends Executive {
        private String position = "Building Executive";
        private File building_Executive_Info_txt = new File("src/Database/Building_Executive_Information.txt");

        public Building_Executive() {
        }

        public Building_Executive(String executiveID, String name, char gender, String contact_number) {
            super.set_Info(executiveID, name, gender, contact_number, this.position);
        }
    }
}
```

To illustrate the implementation of inheritance, consider the provided code that defines the Building_Executive_Function class. In this code, the Building_Executive class is defined to extend the Executive class, thereby inheriting all its non-private attributes, such as ID, name, gender, contact number, and position. Moreover, the Building_Executive class adds its specific attribute, position, to the inherited attributes of the Executive class, assigning a "Building Executive" value by default. Finally, the class also defines its file object, building_Executive_Info_txt, which stores the information specific to a building executive.

The implementation of inheritance in this code provides several benefits, including code reusability, hierarchical categorization of classes, and the ability to build upon the functionality of a superclass when developing a subclass. By avoiding duplicating the code for the attributes and methods of the Executive class in the Building_Executive class, the implementation of inheritance enables the Building_Executive class to reuse and build upon the functionality of the Executive class. Additionally, inheritance allows for hierarchical categorization of classes, with more specific classes (such as Building_Executive) inheriting from more general classes (such as Executive).

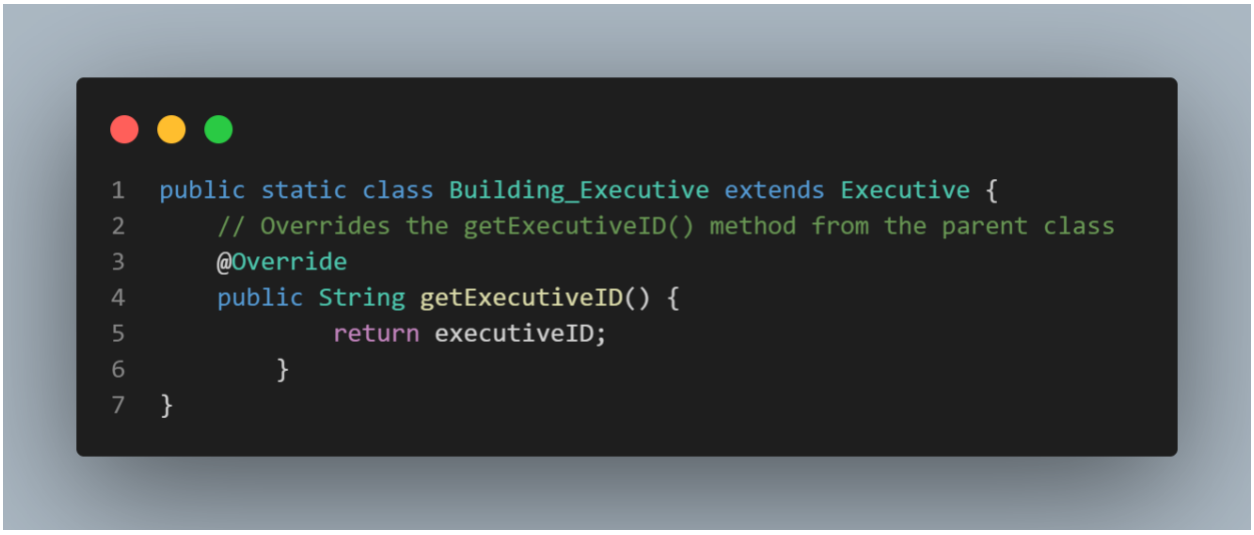
In conclusion, the implementation of inheritance in the provided code exemplifies the fundamental OOP concept of class inheritance, which is a powerful tool for developers to create hierarchical relationships between classes, reuse existing code, and enhance the organization and structure of software. However, developers should also be aware of the disadvantages of inheritance, such as the close dependence between base and super classes, the need to modify child classes when updating the functionality of the base class, and the complexity of maintaining child classes' functionality when methods in the superclass are removed.

6.5 Polymorphism

Polymorphism seeks to construct a function that performs numerous instructions based on the object sent to it as input. The transformation of an entity into a derived type is referred to as polymorphism. Polymorphism is a method of creating code that can be utilized in several ways to make several objects. Polymorphism is often performed by using functions that take class or interference parameters. This parameter's location in the object supplied as an input to this function influences its behavior.

Implementation

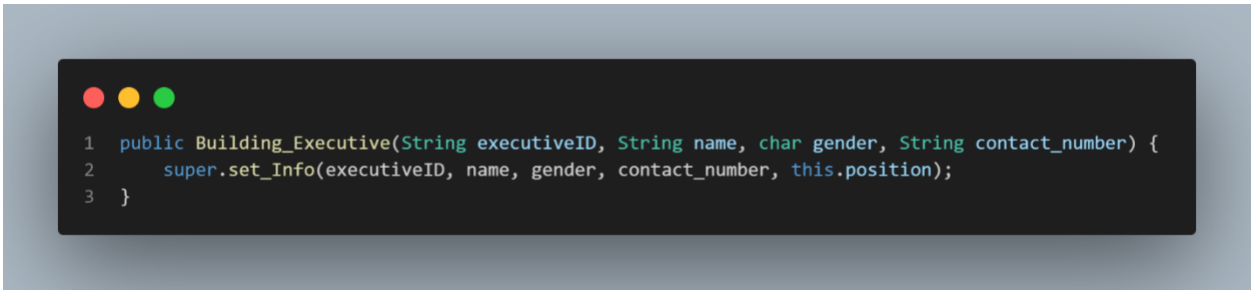
The implementation of polymorphism in the provided Java code is illustrated by the Building_Executive class, a subclass of the Executive class. The Building_Executive class overrides some of the parent class methods, allowing it to provide a specific implementation of these methods. This is demonstrated in the code snippet below, where the Building_Executive class overrides the getExecutiveID() method from the Executive class to return its executiveID attribute:



```
1  public static class Building_Executive extends Executive {
2      // Overrides the getExecutiveID() method from the parent class
3      @Override
4      public String getExecutiveID() {
5              return executiveID;
6          }
7  }
```

To further explain the implementation of polymorphism in the provided Java code, the Building_Executive class is used to create objects of both its type and the parent class Executive type. This is because the Building_Executive class inherits all the methods of the parent class Executive, including its constructor, which is used to create objects of type Executive. This demonstrates polymorphism in the ability of the Building_Executive class to take on multiple

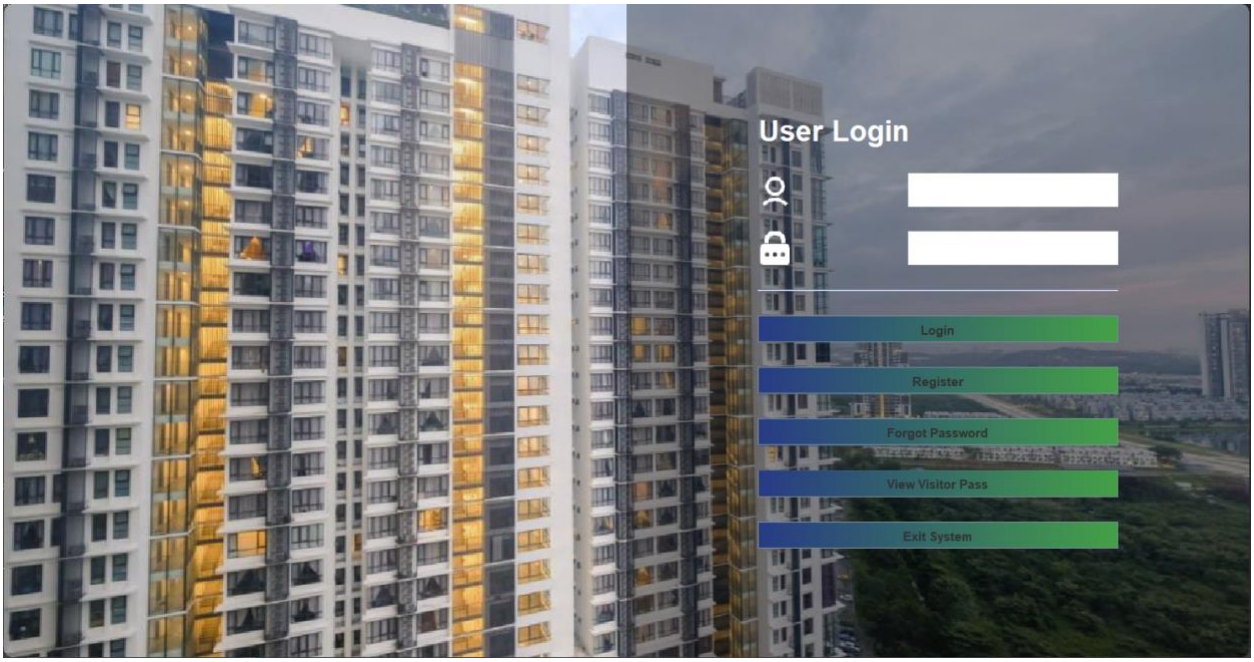
forms, specifically the form of the parent class Executive, which is used in the code below to create an object of type Executive:



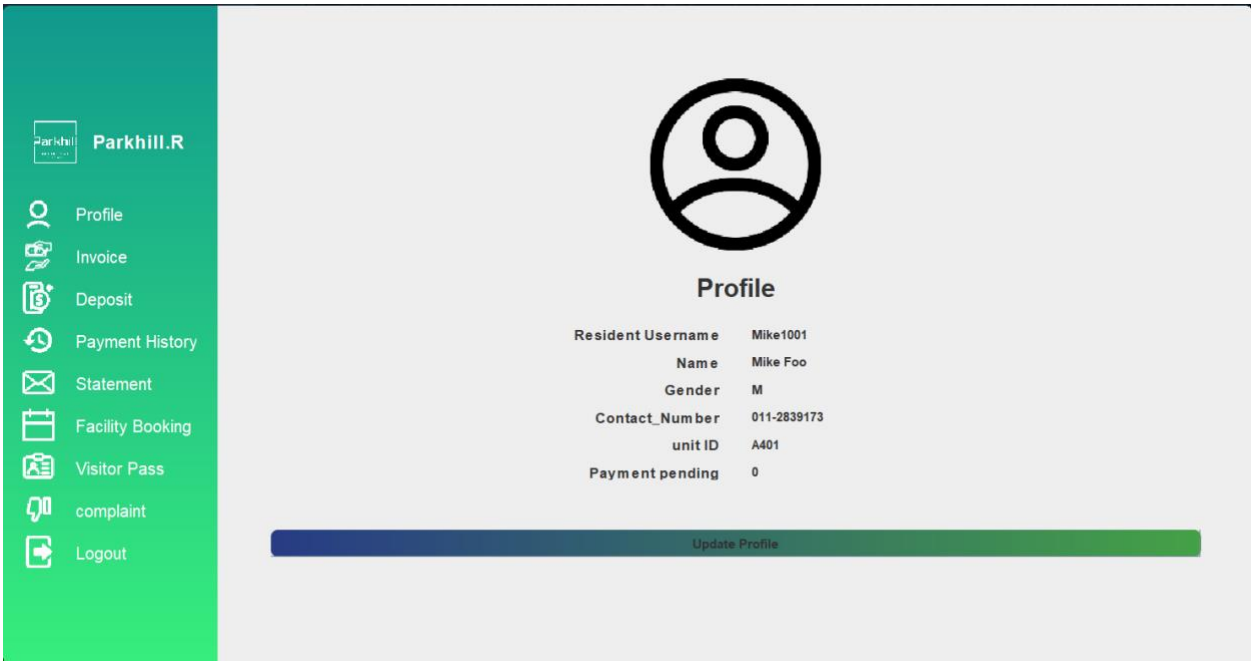
```java
public Building_Executive(String executiveID, String name, char gender, String contact_number) {
    super.set_Info(executiveID, name, gender, contact_number, this.position);
}
```

7 TP067349 Nadila Binti Ahmad Shahrul Nizam

7.1 User-level access, logging, and logout activity



The presented interface showcases the system's login page, which is a unified entry point for all user types. After entering their unique credentials, different user types will be redirected to their respective user dashboards. It is crucial to note that successful login requires inputting valid user IDs and passwords. Additionally, residents who wish to register within the system can do so via the "Register" button and await approval from an authorized administrator.



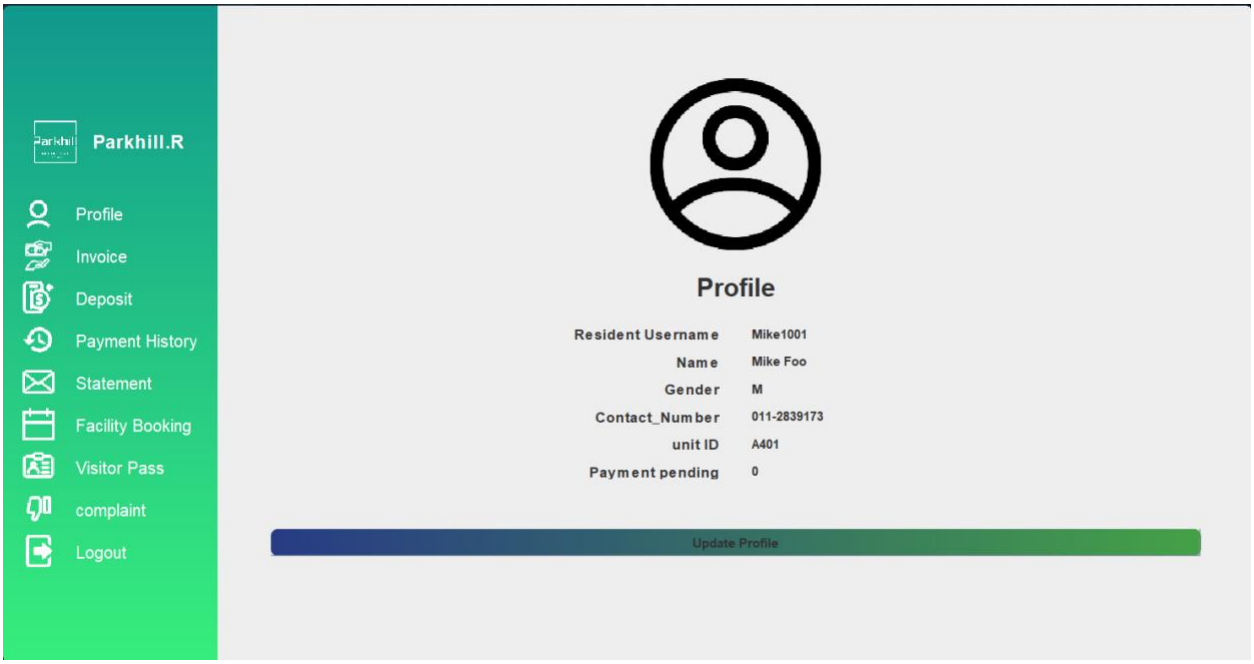
After logging into the system, all user interfaces have a "Logout" button in the bottom left corner. By clicking this button, each user can return to the login page.

7.2 Resident / Tenant

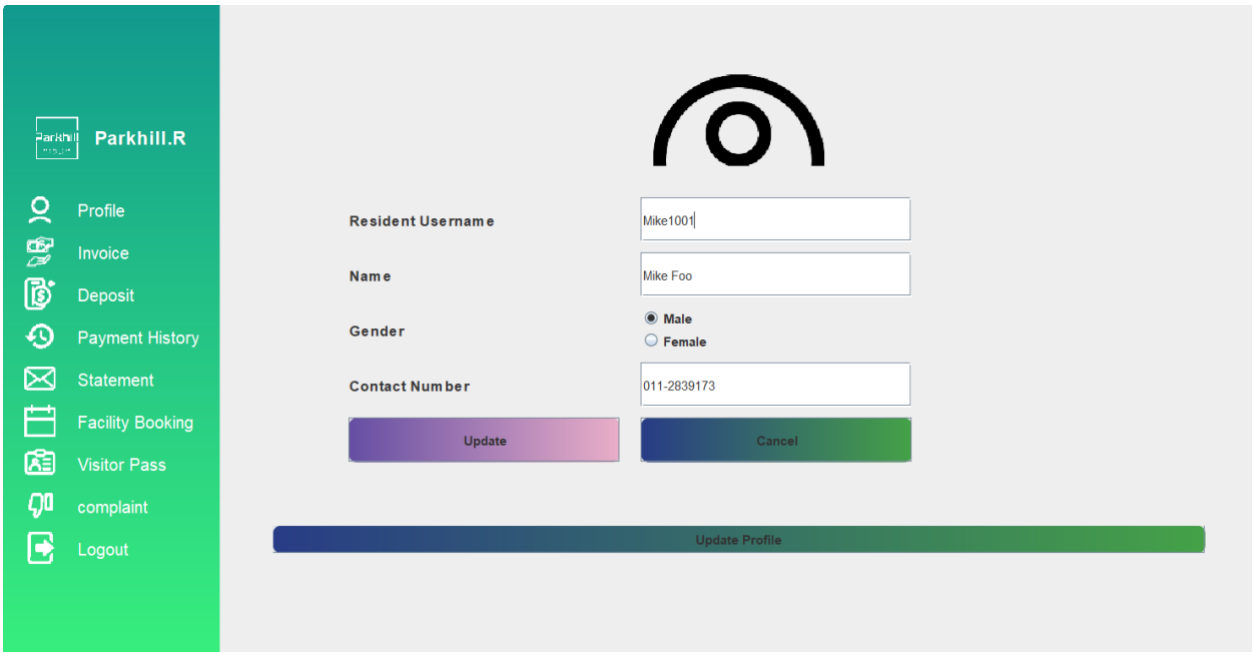
7.2.1 Functional Requirement

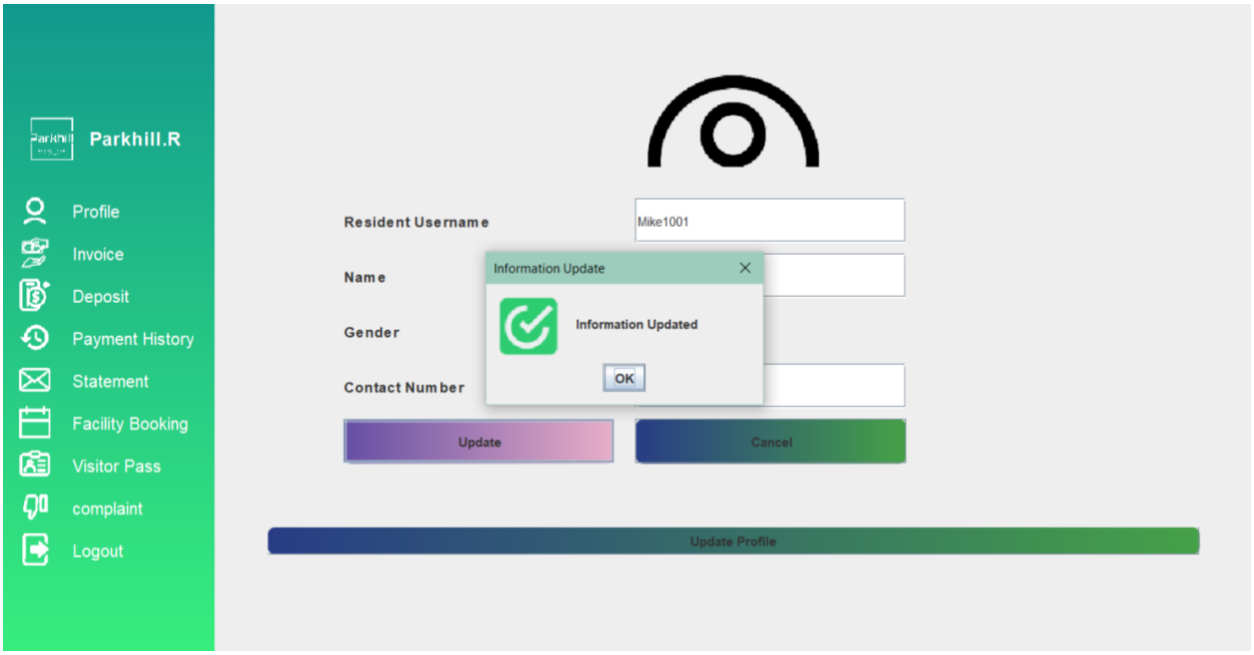
1. User profile interface
2. Invoice function
3. Deposit function
4. Payment history interface
5. Statement interface
6. Facility booking function
7. Visitor pass interface
8. Complaint interface

7.2.2 Profile



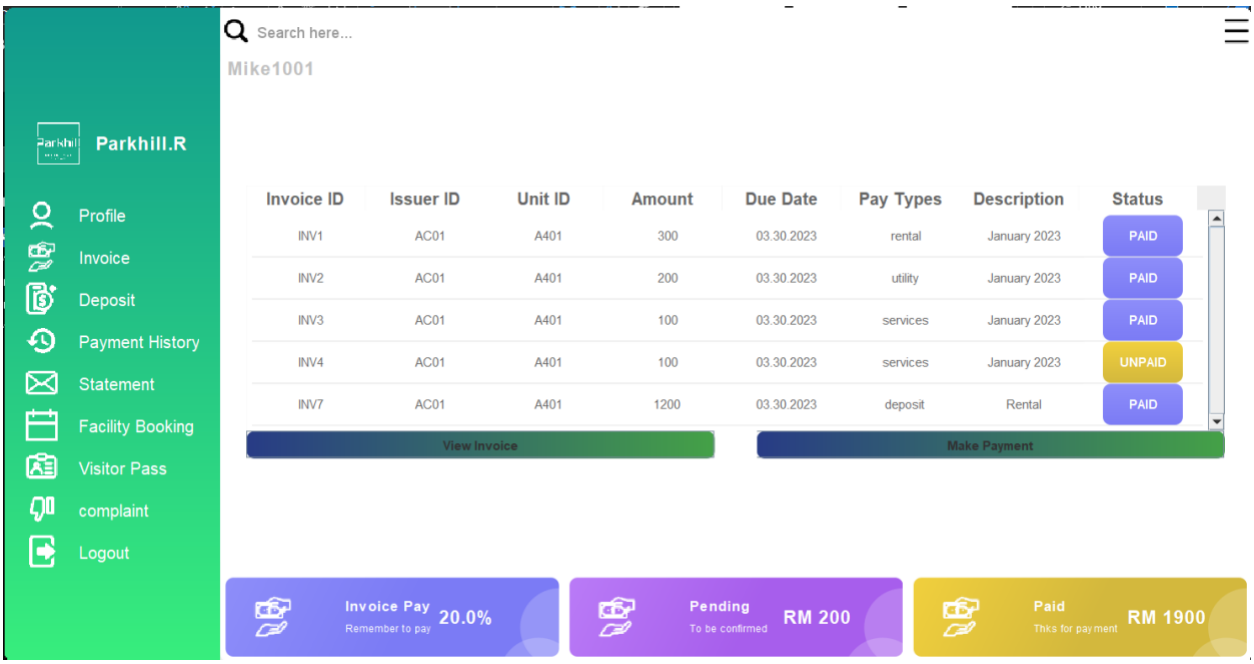
Once the Resident has successfully logged into the system, they will be directed to their designed interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the resident profile page is accessible through this interface and allows users to modify their profile information.





By clicking on the update profile button, the user will be led to this interface which allows them to change their username, name, gender, and contact number. After clicking the "Update" button, a confirmation notice will alert users that the information is updated.

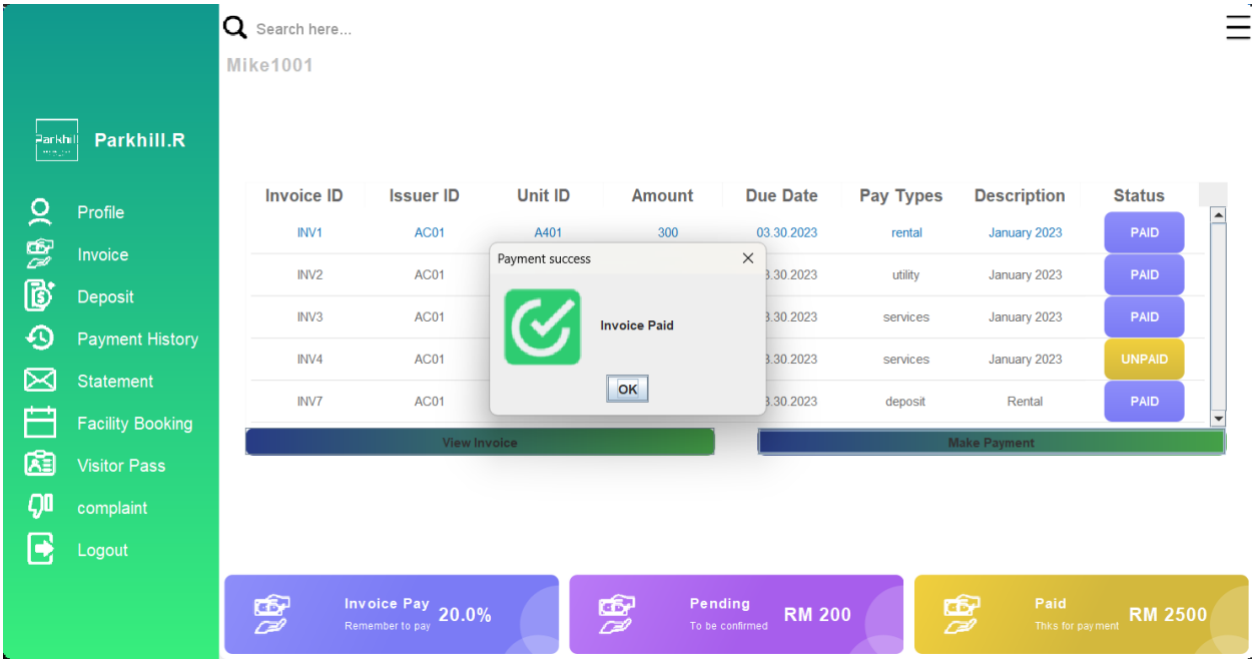
7.2.3 Invoice



When users access the invoice tab, they will be directed to a user interface that displays all their invoices. The bottom part of the interface contains boxes with reminders that show users their

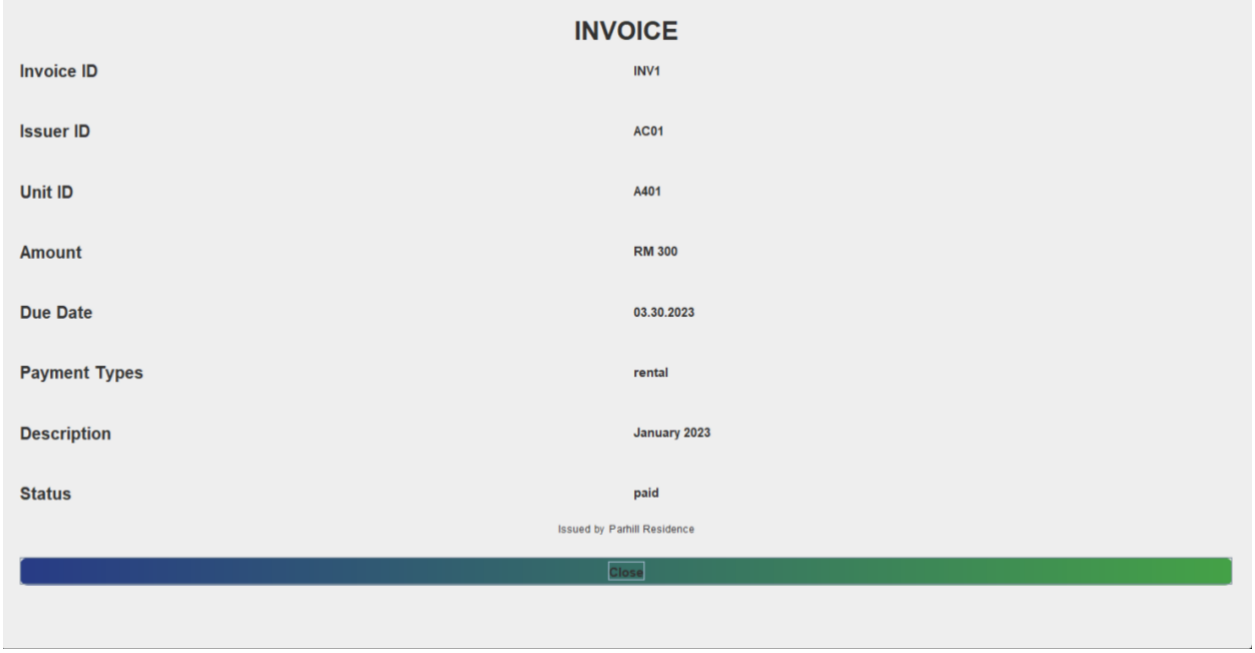
pending, paid, and unpaid invoices. The interface also features button functions allowing users to search their financial statements, view invoices, and make payments.

7.2.3.1 Make Payment



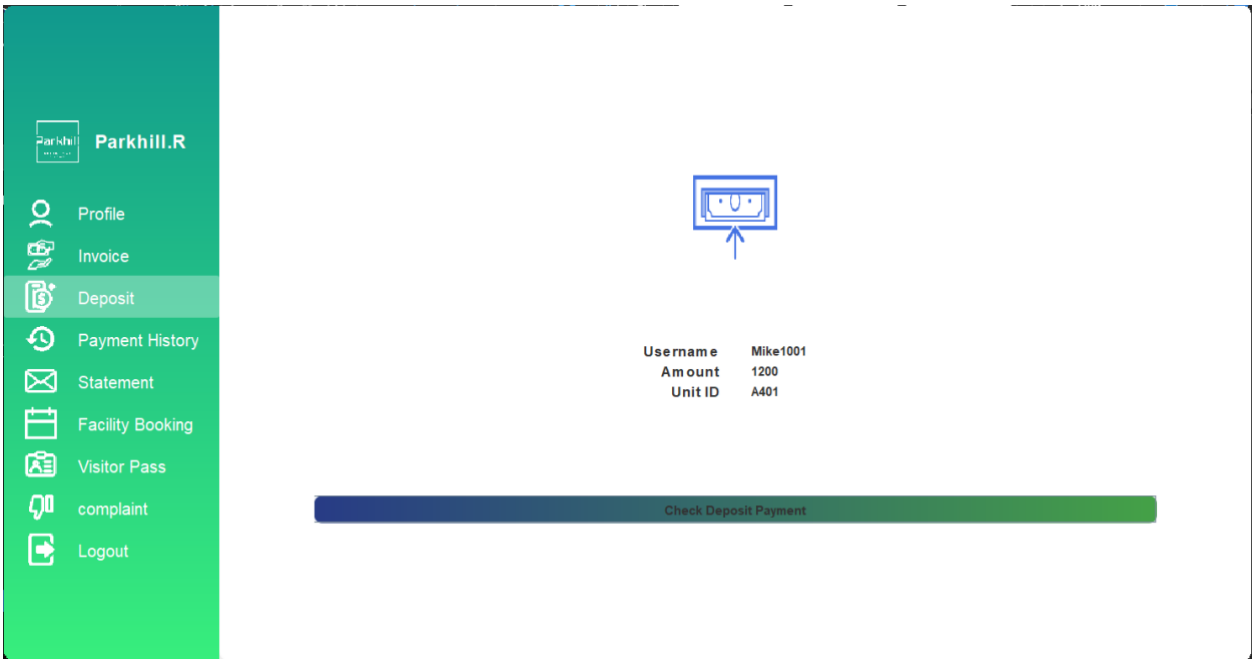
If the user clicks on the "Make Payment" button, they will be alerted once the payment has been made.

7.2.3.2 View Invoice



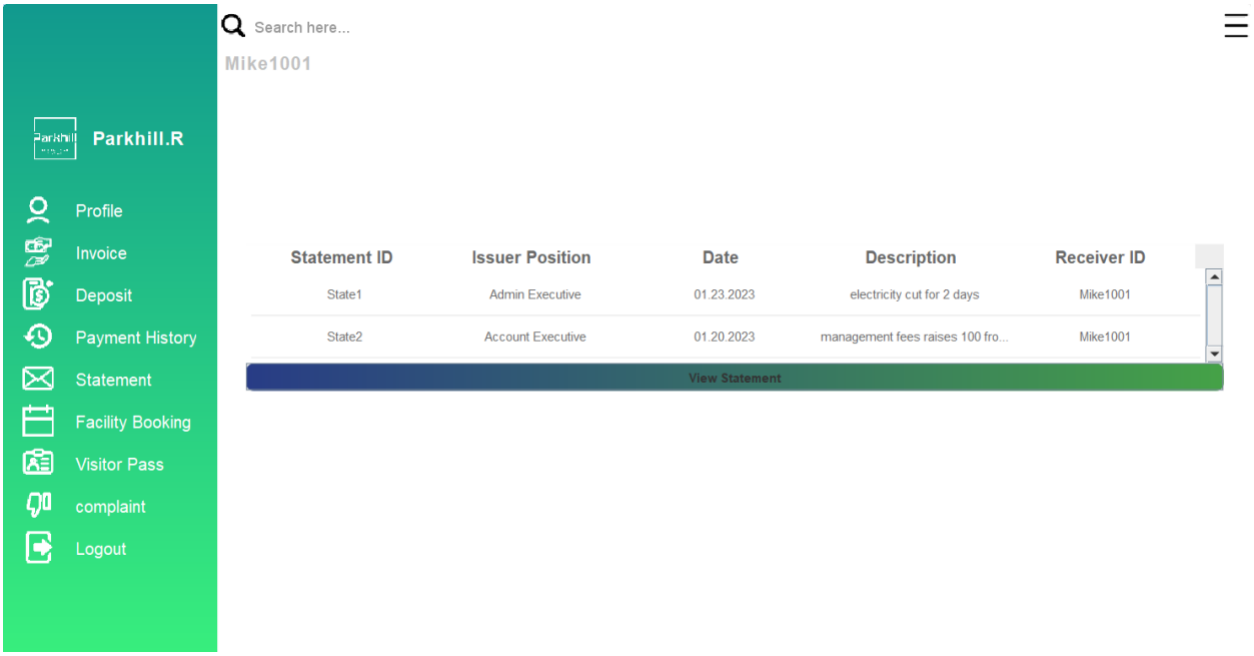
If the users choose to click on one of the payment histories displayed and then view it, they can click the "View Invoice" button, showing the user the invoice for the particular invoice they chose.

7.2.4 Deposit



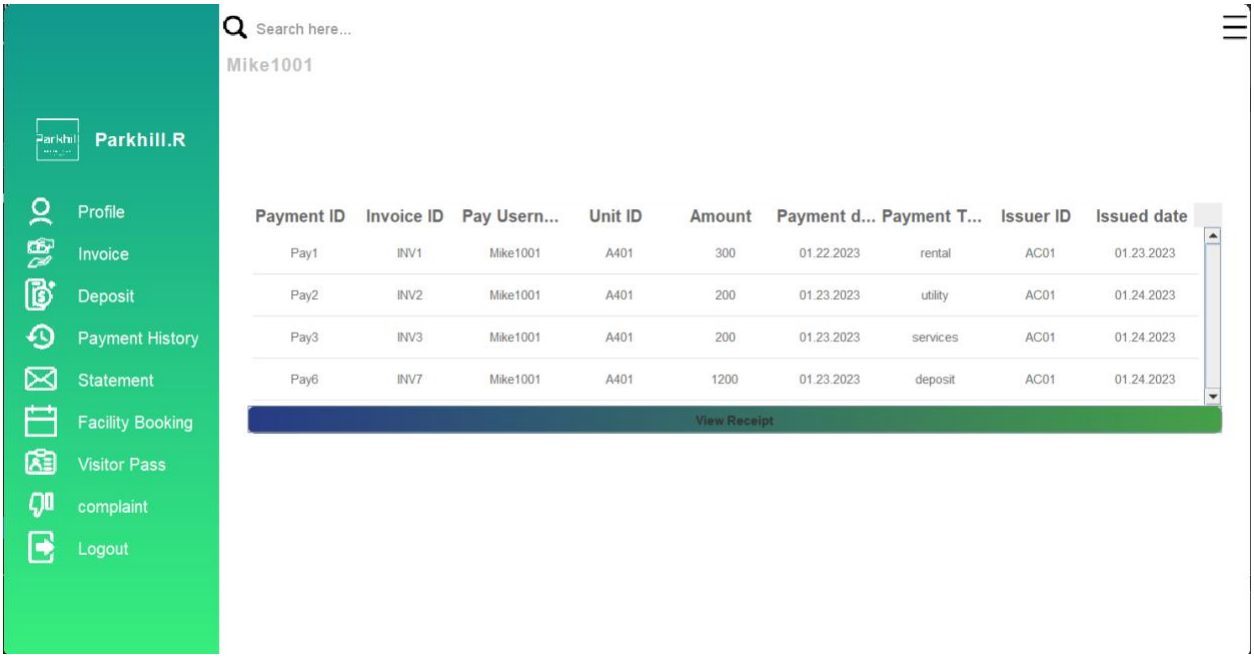
When users navigate to the deposit section in the menu, they will be directed to this interface. Here, they can view the payment amount that is due. The interface also contains a button labeled "check deposit payment", which takes the user back to the previous invoice section.

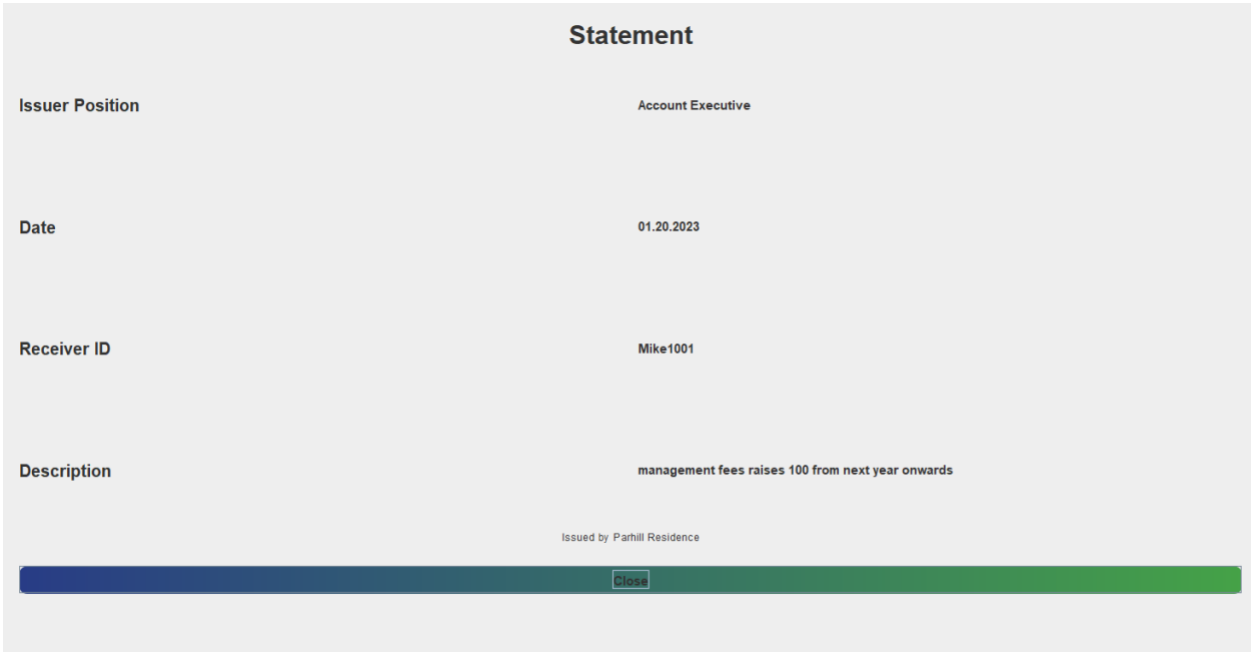
7.2.5 Payment history



Users will be directed to this interface when browsing the payment history section in the menu. Here, they can view the users' payment history. In addition, the interface also allows users to search through their payment history and view the receipt for their selected payment history.

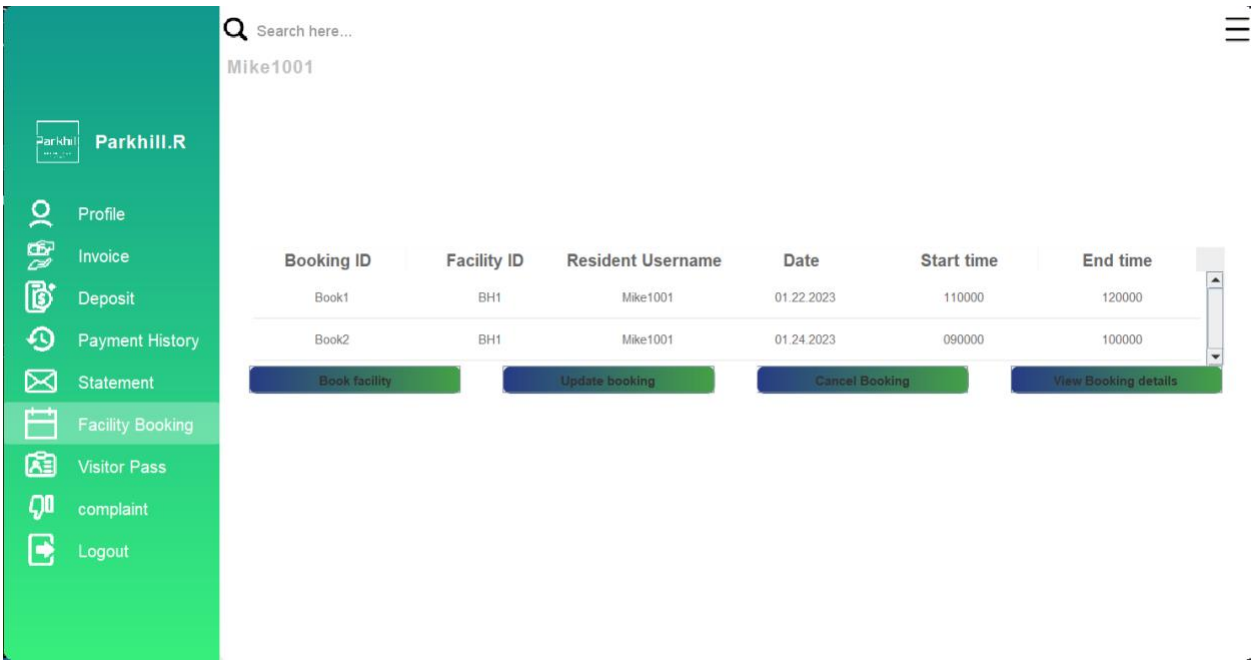
7.2.6 Statement





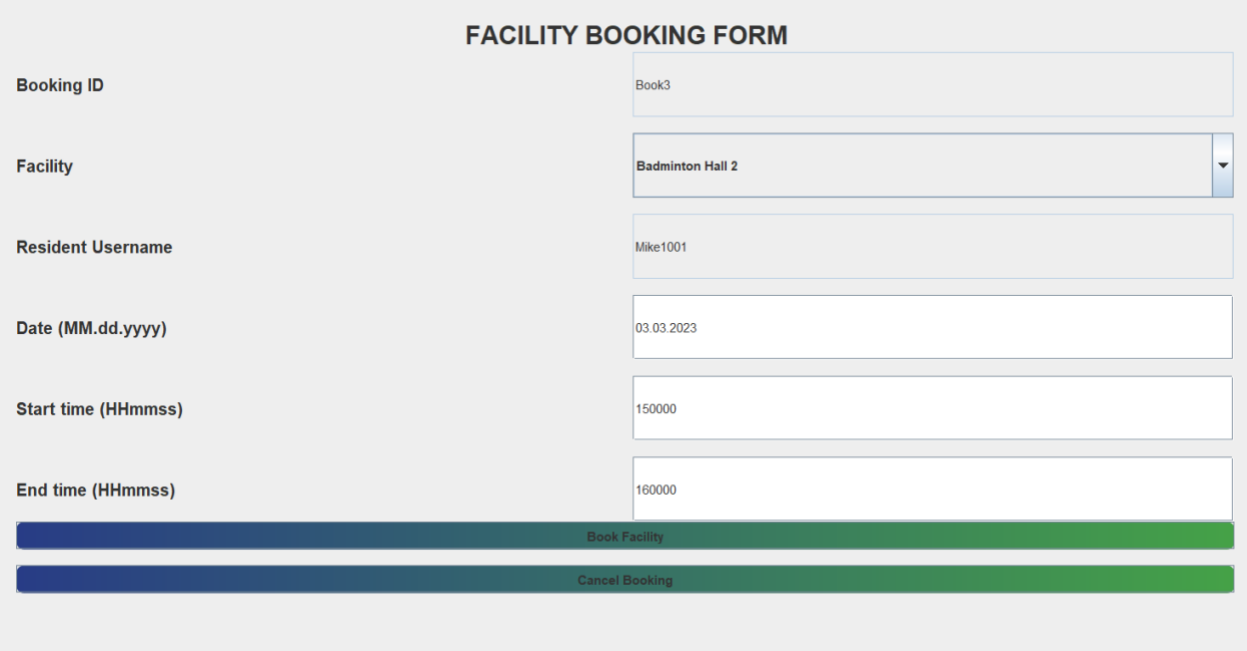
Users who navigate the statement section in the menu will be taken to this interface to access and view the financial statements. The interface also allows users to search through their financial statements and view the statement they have selected from the table.

7.2.7 Facility booking



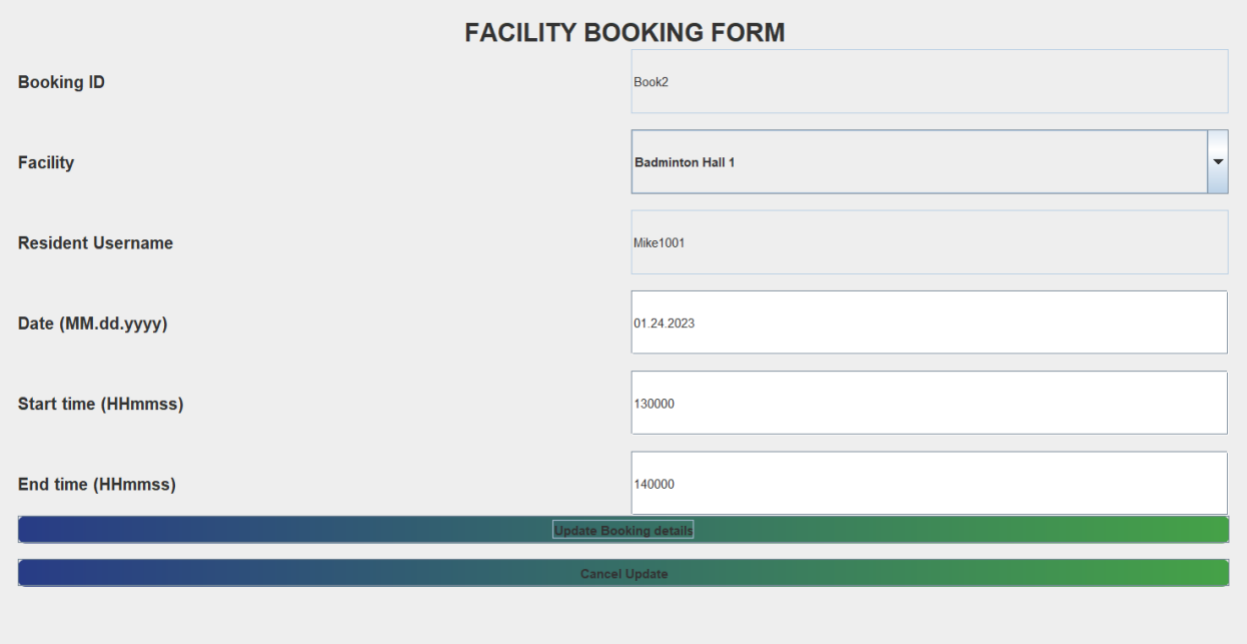
When users access the facility booking section in the menu, they will be directed to this interface to view and access their facility booking. The interface enables users to browse their facility booking history, reserve a facility, modify, or cancel a reservation, and examine the specifics of their booking.

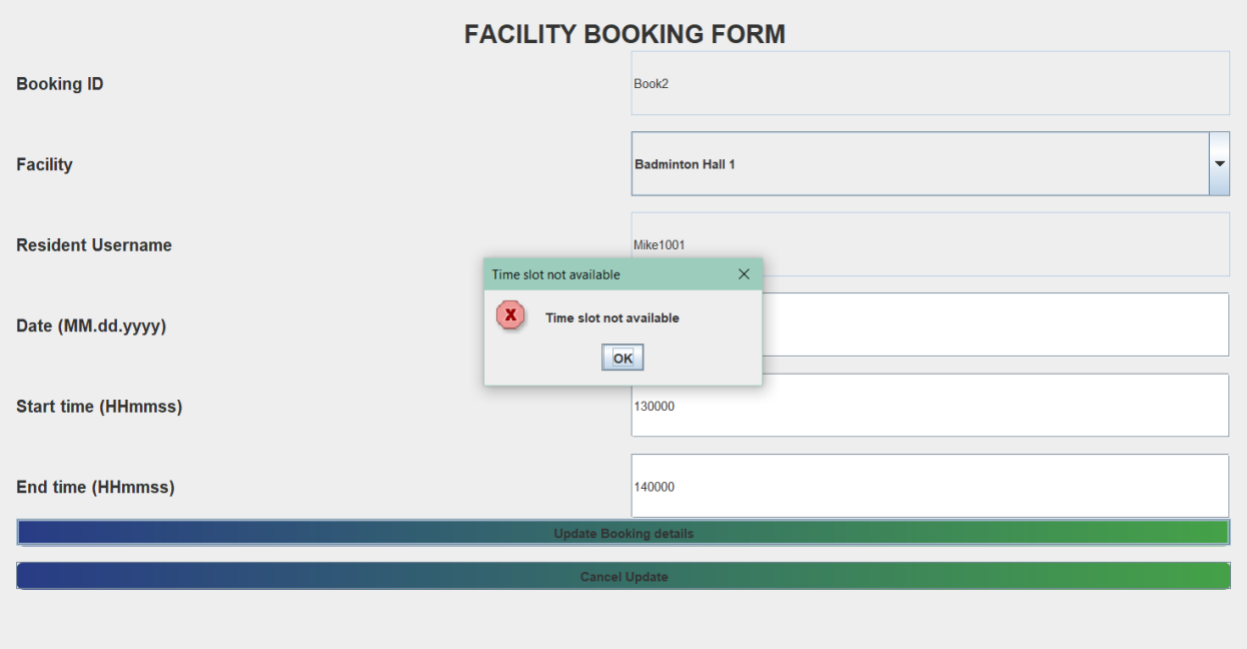
7.2.7.1 Book facility



By clicking on the "book facility" button, the users will be led to a facility booking form allowing them to fill in their details to book a facility. By clicking on the "book facility" button on this form, there will be a confirmation alerting the users regarding their booking.

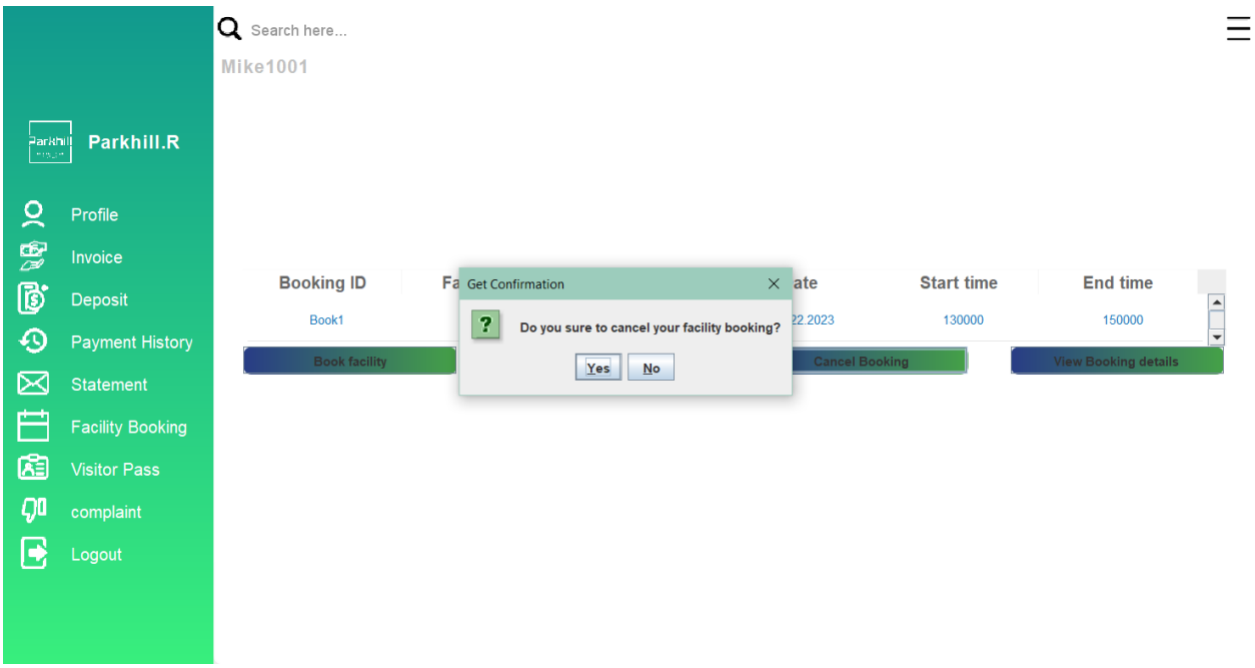
7.2.7.2 Update booking





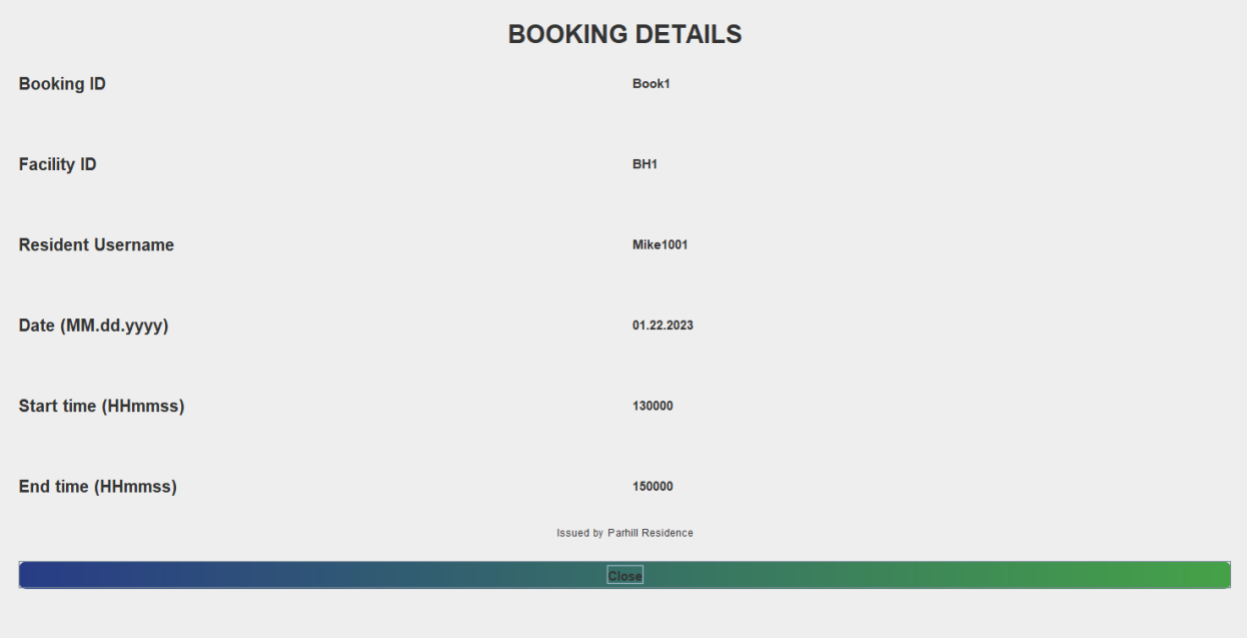
In order to update a facility booking, the users must click on the booking they wish to update and then proceed by clicking on the "update booking" button. The users will then be brought to this interface which allows them to update their booking details. Afterward, the users can click on the "update booking" button to submit their updated booking form. If the users select a time slot that is not currently available, the system will notify the users regarding this.

7.2.7.3 Cancel Booking



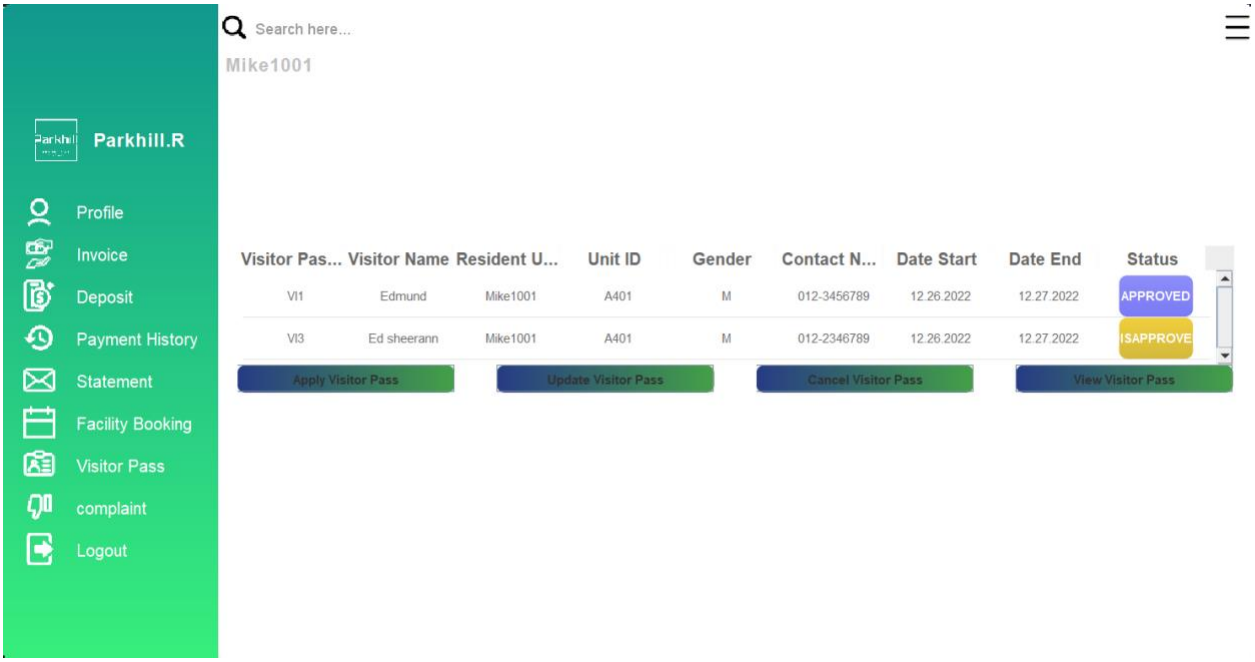
To cancel a facility booking, users must select the booking they want to cancel and click the "cancel booking" button. This action will lead them to a new interface where they can confirm the cancellation of their booking details. Users will be notified to verify their decision before the booking is officially canceled.

7.2.7.4 View Booking Details



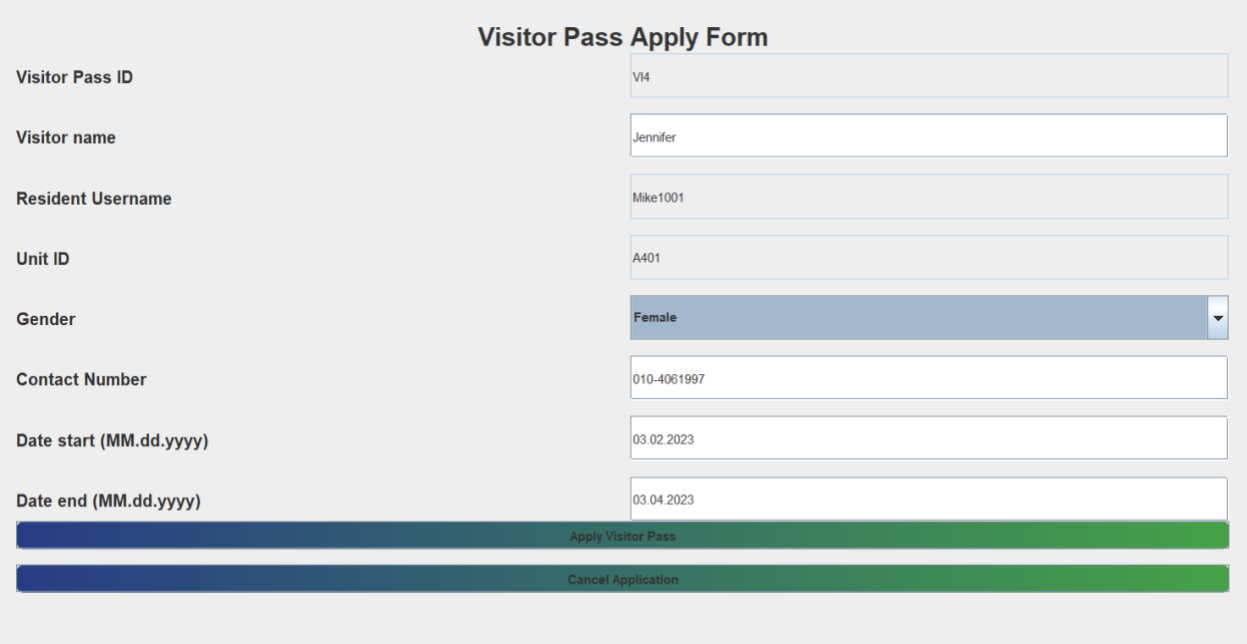
In order to access and view a facility booking, users need to first choose the booking they want to view, followed by clicking on the "view booking" button. This will direct them to a new interface that displays the details of their chosen booking.

7.2.8 Visitor pass



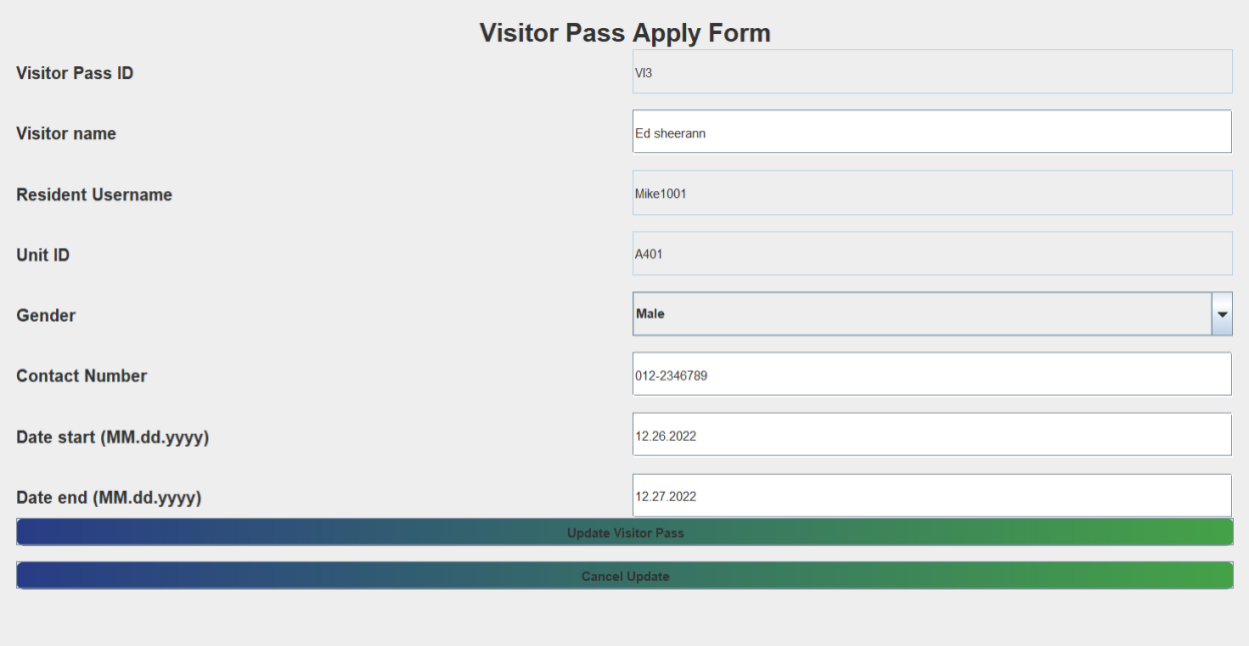
When users access the visitor pass section in the menu, they will be directed to this interface to view and access their visitor pass. In addition, the interface enables users to browse their visitor pass history, apply, update, cancel and view visitor passes.

7.2.8.1 Apply Visitor Pass



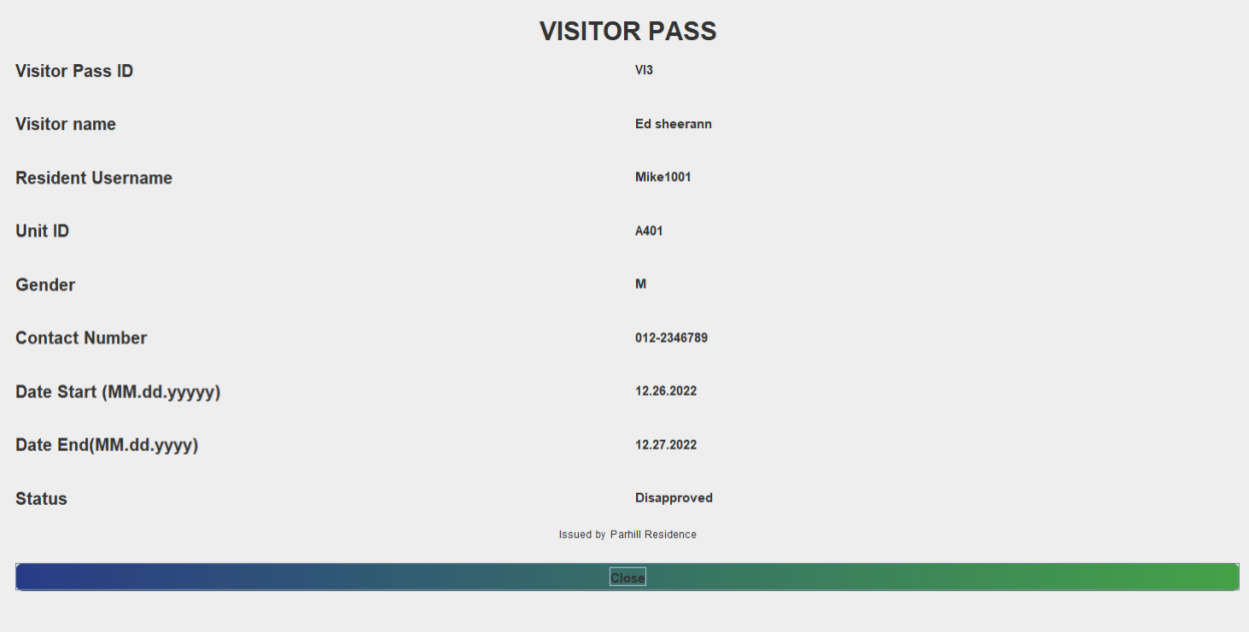
By clicking the "apply visitor pass" button, the users will be led to a visitor pass form, allowing them to fill in their details to apply for a visitor pass. By clicking on the "apply visitor pass" button on this form, there will be a confirmation alerting the users regarding their application.

7.2.8.2 Update Visitor Pass



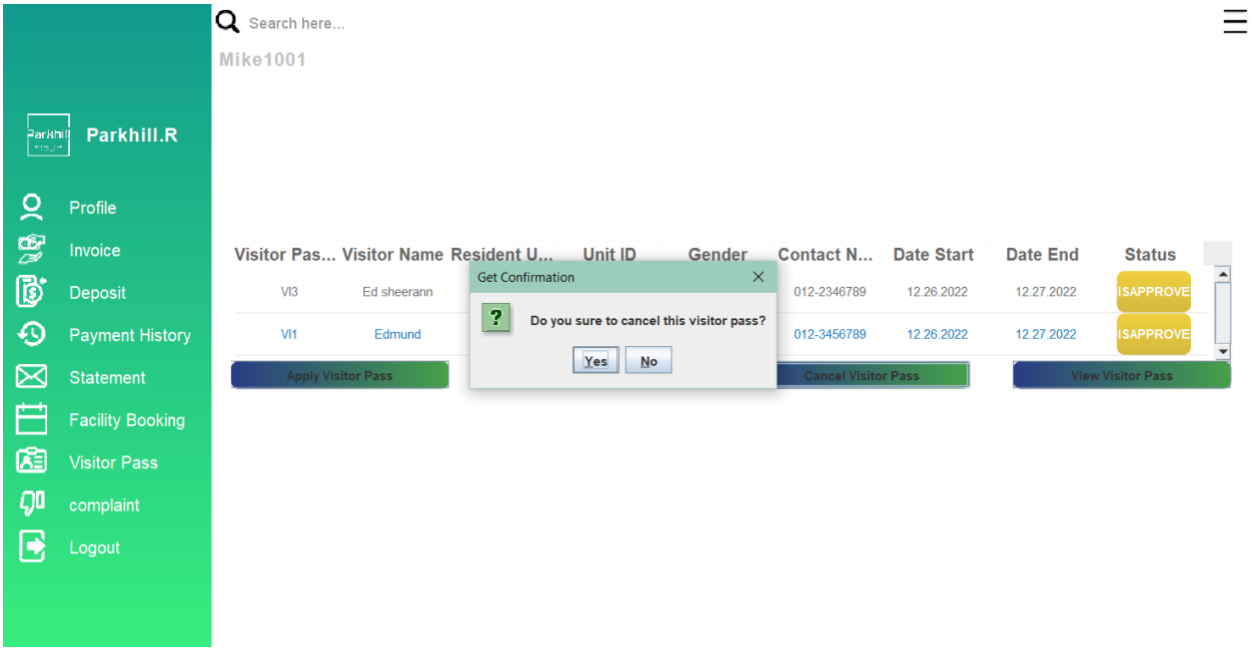
To update a visitor pass, the users must click on the visitor pass they wish to update and then proceed by clicking the "update visitor pass" button. The users will then be brought to this interface which allows them to update their visitor pass details. Afterward, the users can click the "update visitor pass" button to submit their updated visitor pass form.

7.2.8.3 View Visitor Pass



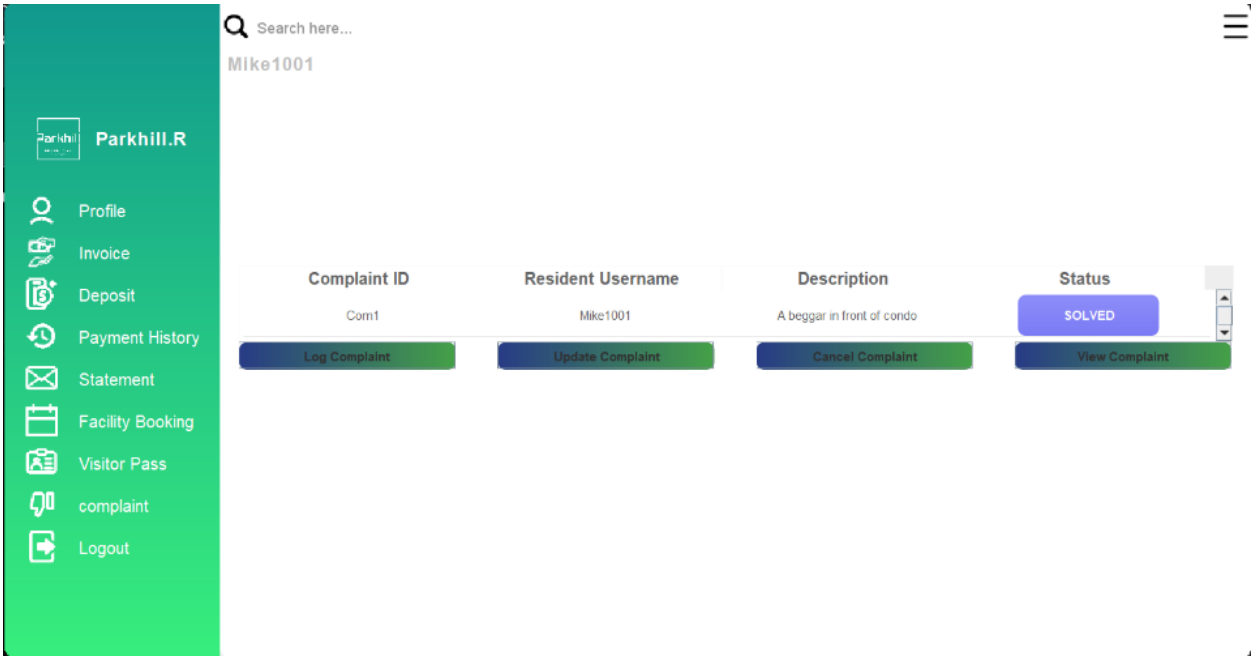
Users must select the visitor pass they want to cancel and click the “cancel visitor pass” button to cancel a visitor pass. This action will lead them to a new interface where they can confirm the cancellation of their pass details. Users will be notified to verify their decision before the visitor pass is officially canceled.

7.2.8.4 Cancel Visitor Pass



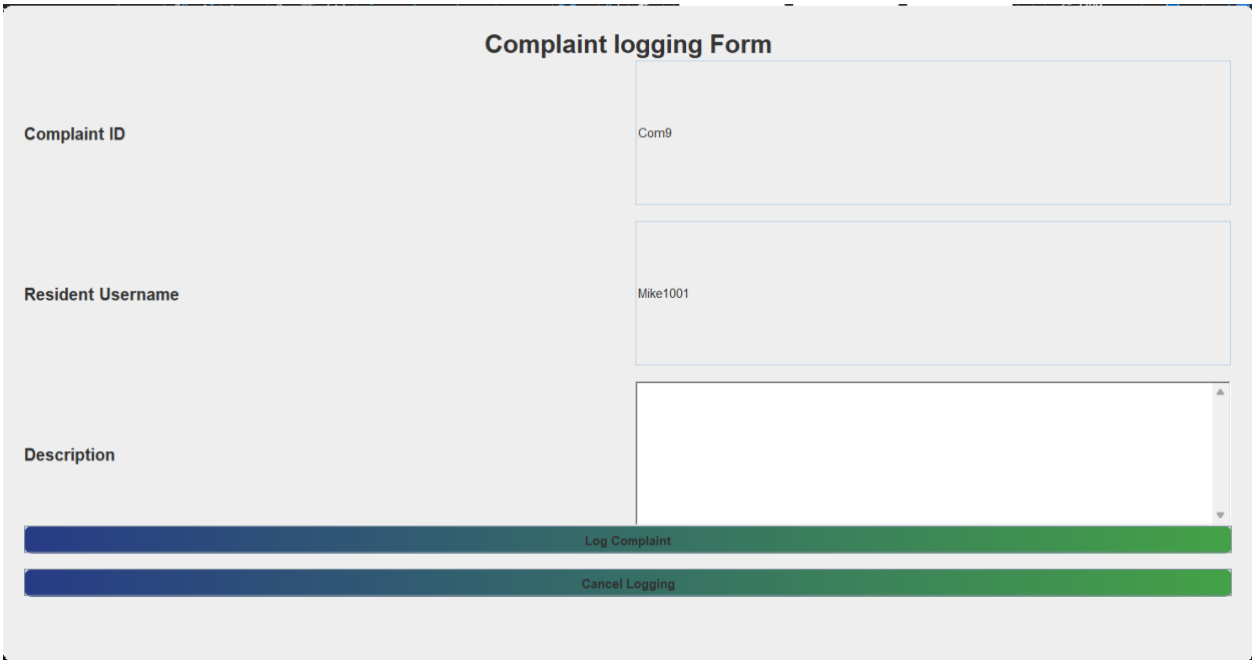
In order to access and view a visitor pass, users must first choose the visitor pass they want to view, followed by clicking on the “view pass” button. This will direct them to a new interface that displays the details of their chosen pass.

7.2.9 Complaint



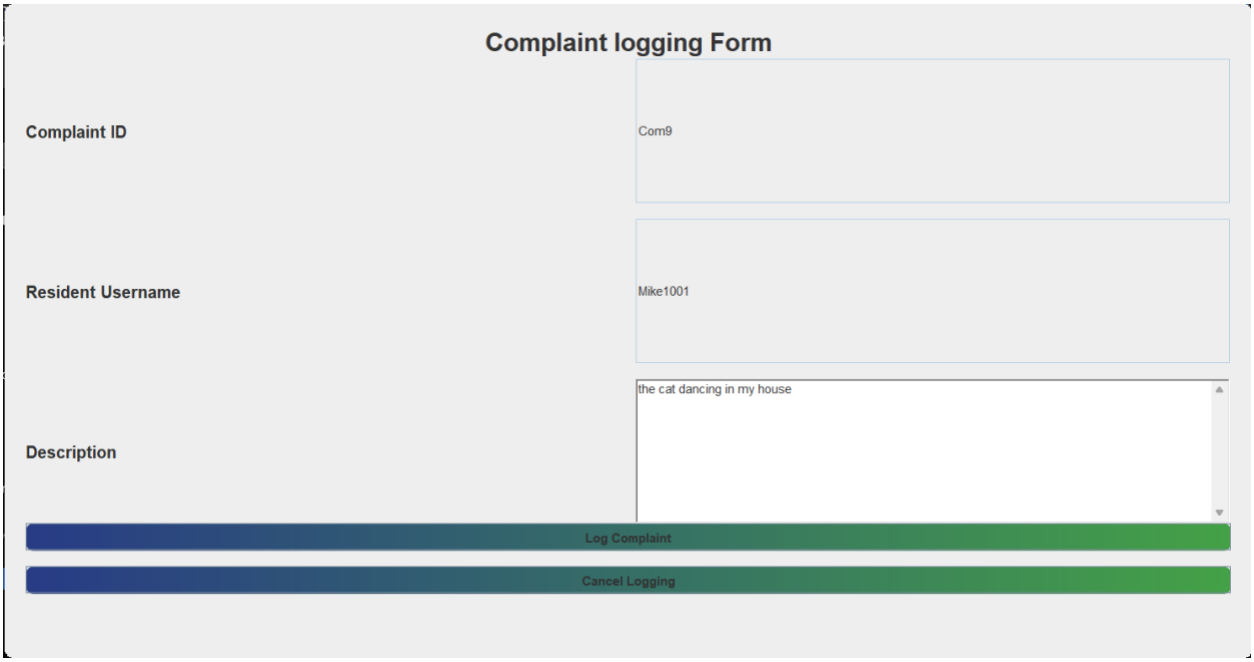
When users access the complaint section in the menu, they will be directed to this interface to view and access their complaints. In addition, the interface enables users to browse their complaint history, track, log, update, cancel, and view the complaint.

7.2.9.1 Log complaint



By clicking the "log complaint" button, users will be directed to a complaint form, allowing them to fill in their details to file a complaint. Clicking the "log complaint" button on this form will trigger a confirmation alerting the users about their complaint submission.

7.2.9.2 Update Complaint

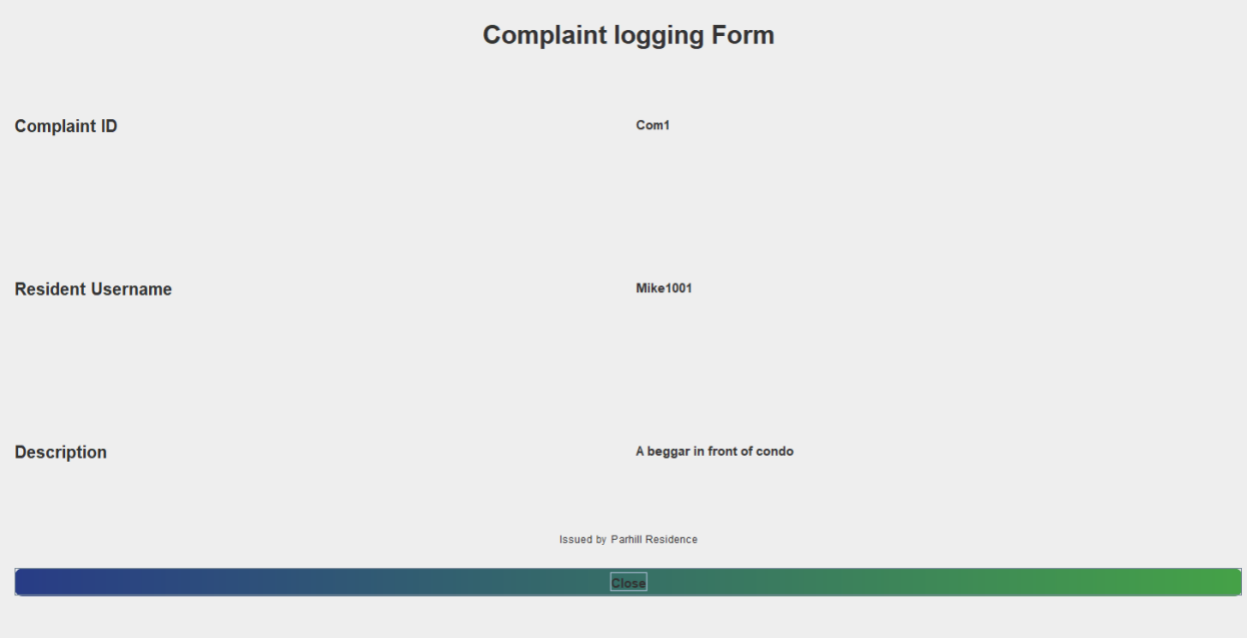


To update a complaint, users must click on the complaint they wish to update and then proceed by clicking the "update complaint" button. The users will then be directed to an interface that allows them to update their complaint details. Afterwards, users can click the "update complaint" button to submit their updated complaint form.

7.2.9.3 Cancel Complaint

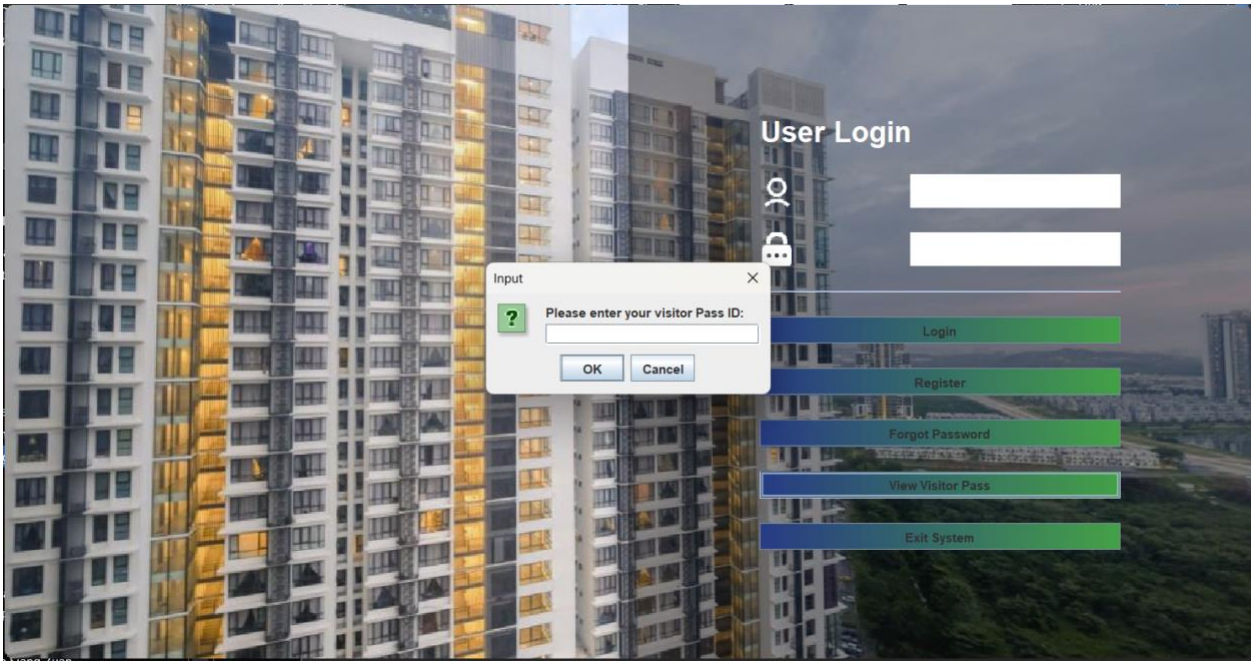
To cancel a complaint, users must select the complaint they want to cancel and click the "cancel complaint" button. This action will lead them to a new interface where they can confirm the cancellation of their complaint details. Users will receive a notification to verify their decision before the complaint is officially cancelled.

7.2.9.4 View Complaint

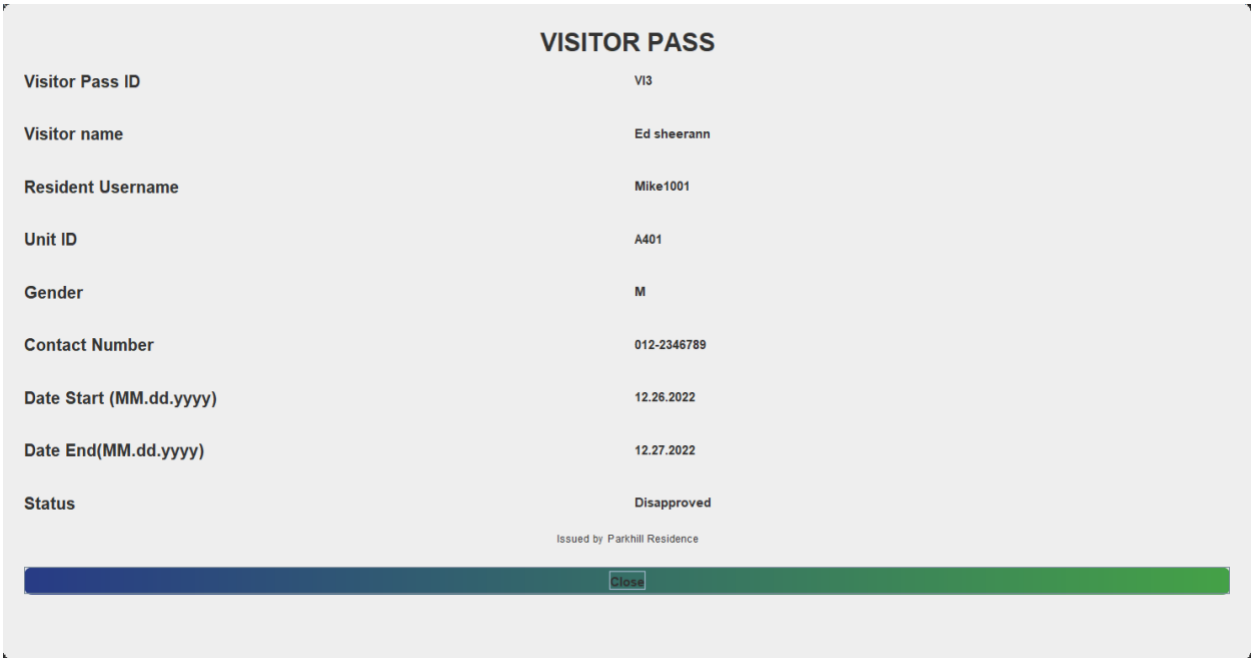


To view a complaint, users must first choose the one they want to view, then click on the "view complaint" button. This will direct them to a new interface that displays the details of their chosen complaint.

7.3 Visitor



Visitors can obtain their relevant information by clicking on "View Visitor Pass" and entering their Visitor Pass ID to conduct a search.



After entering the Visitor Pass ID, the system will enter this interface where the visitor's visiting time, location, and other relevant information will be displayed.

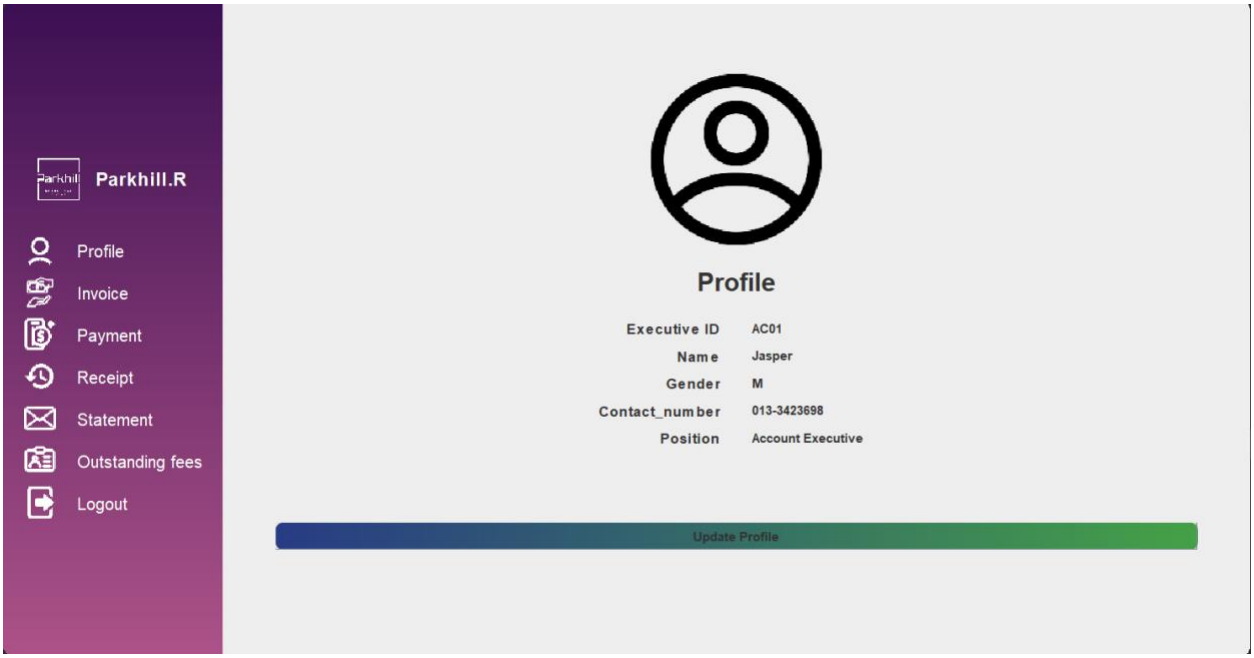
8 TP065116 Neaw Aik Ka

8.1 Account Executive

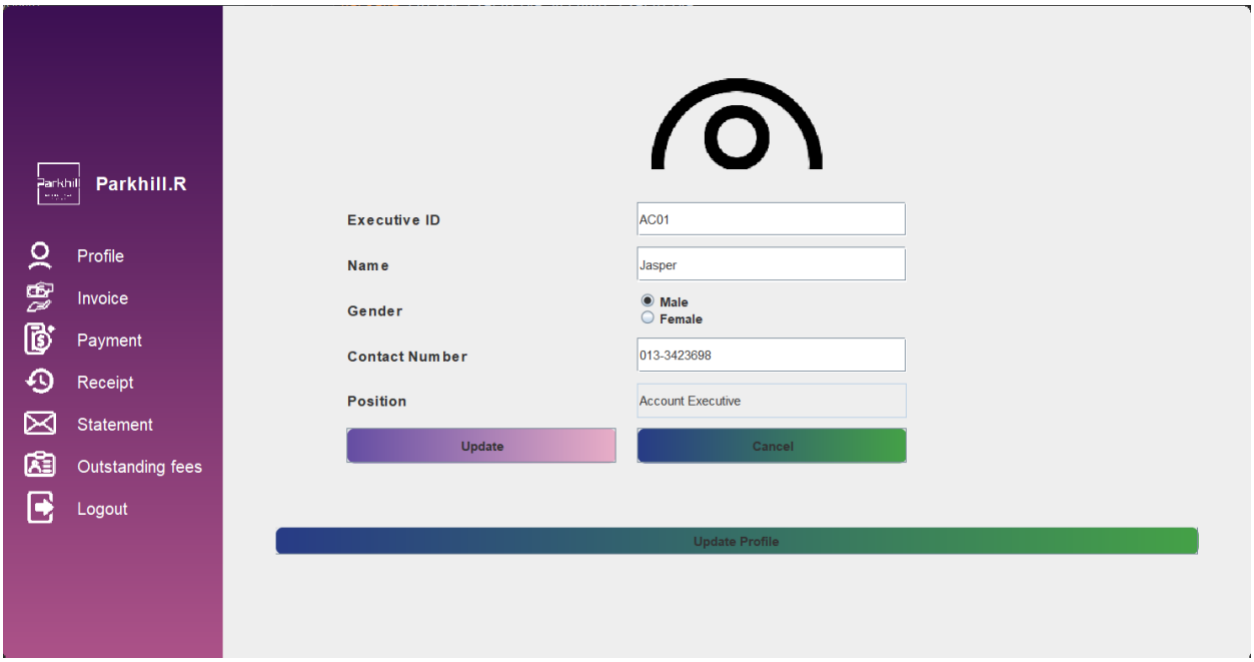
8.1.1 Functional requirements

1. Profile Interface
2. Invoice Management
3. Payment Management
4. Receipt Interface
5. Statement Management
6. Outstanding fees Management
7. Logout

8.1.2 Profile

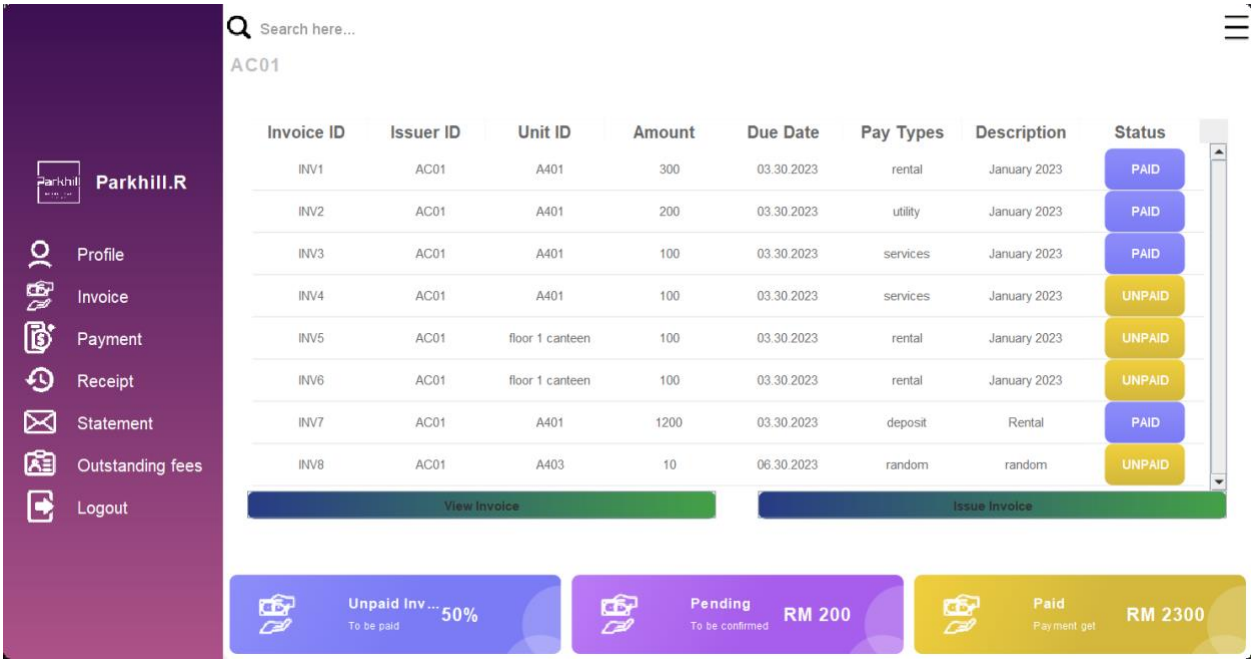


Once the Account executive has successfully logged into the system, they will be directed to their designated interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the account executive profile page is accessible through this interface and allows users to modify their profile information.



By clicking on the update profile button, the user will be led to this interface which allows them to change their username, name, gender and contact number. After clicking the "Update" button, a confirmation notice will alert users that the information is updated.

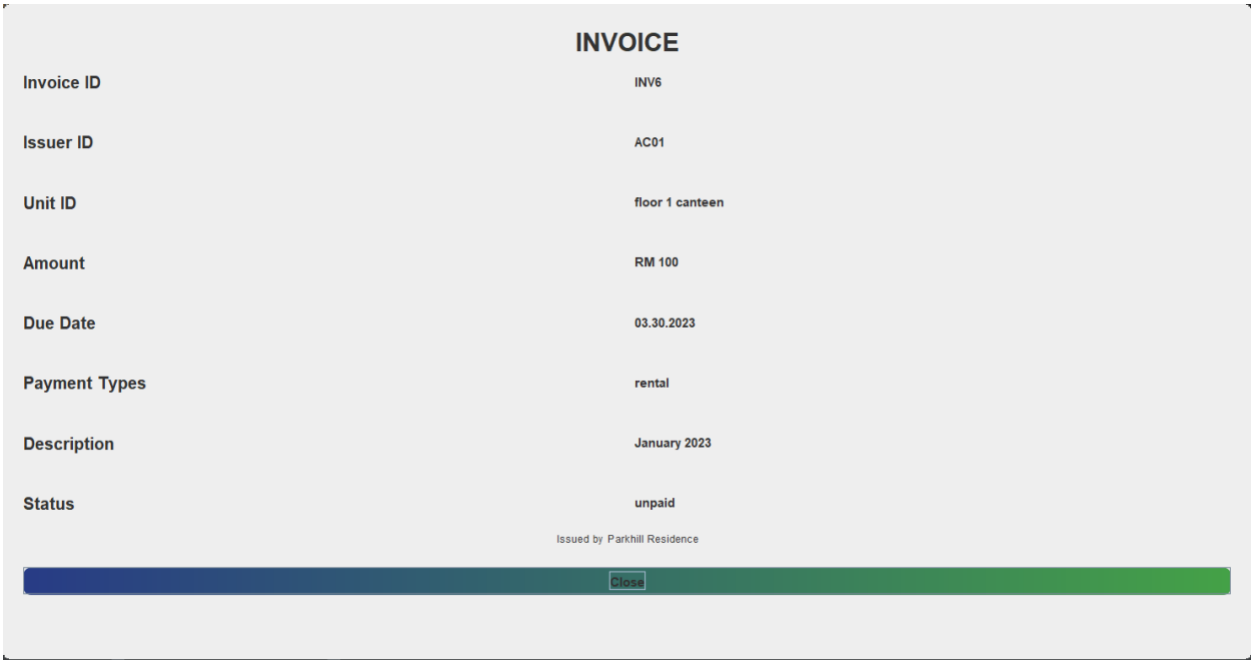
8.1.3 Invoice



When users access the invoice tab, they will be directed to a user interface that displays all their invoices. The bottom part of the interface contains boxes with reminders that show users their

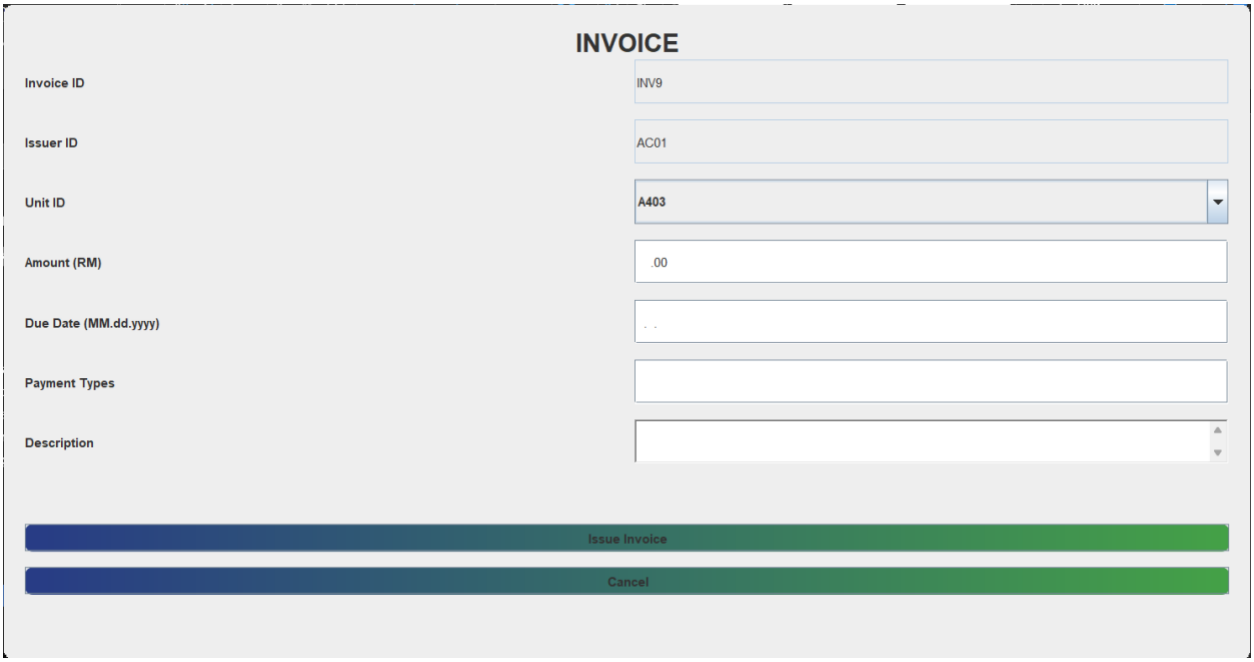
pending, paid, and unpaid invoices. The interface also features button functions allowing users to search their financial statements, view invoices, and make payments.

8.1.3.1 View Invoice



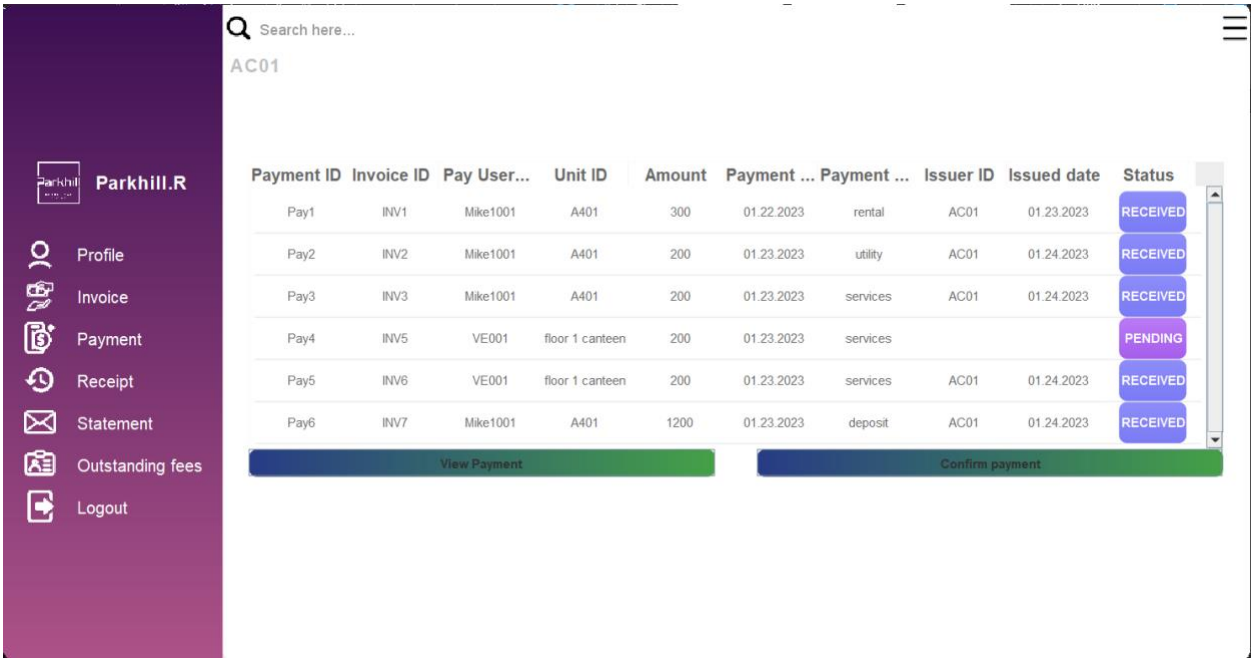
To view a payment, users must choose the one they want to view and click the "view invoice" button. This will direct them to a new interface that displays their invoice details.

8.1.3.2 Issue Invoice



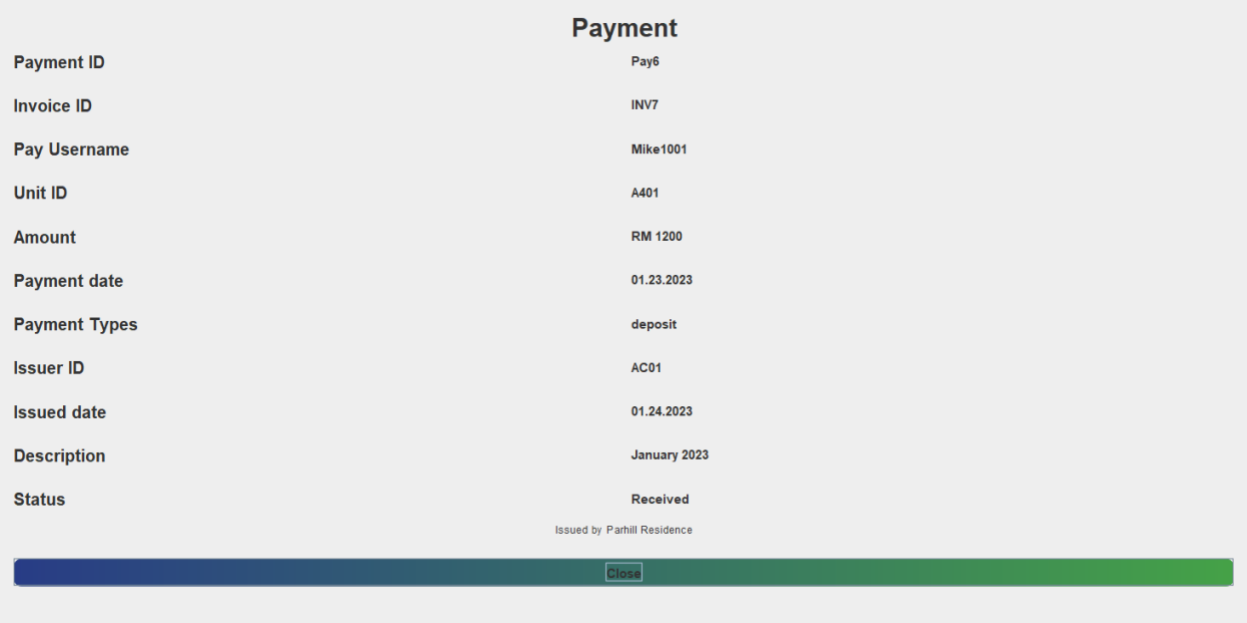
By clicking the "issue Invoice" button, users will be directed to an invoice form, allowing them to fill in their details to issue an invoice. Clicking the "issue invoice" button on this form will trigger a confirmation alerting the users about their issued invoice.

8.1.4 Payment



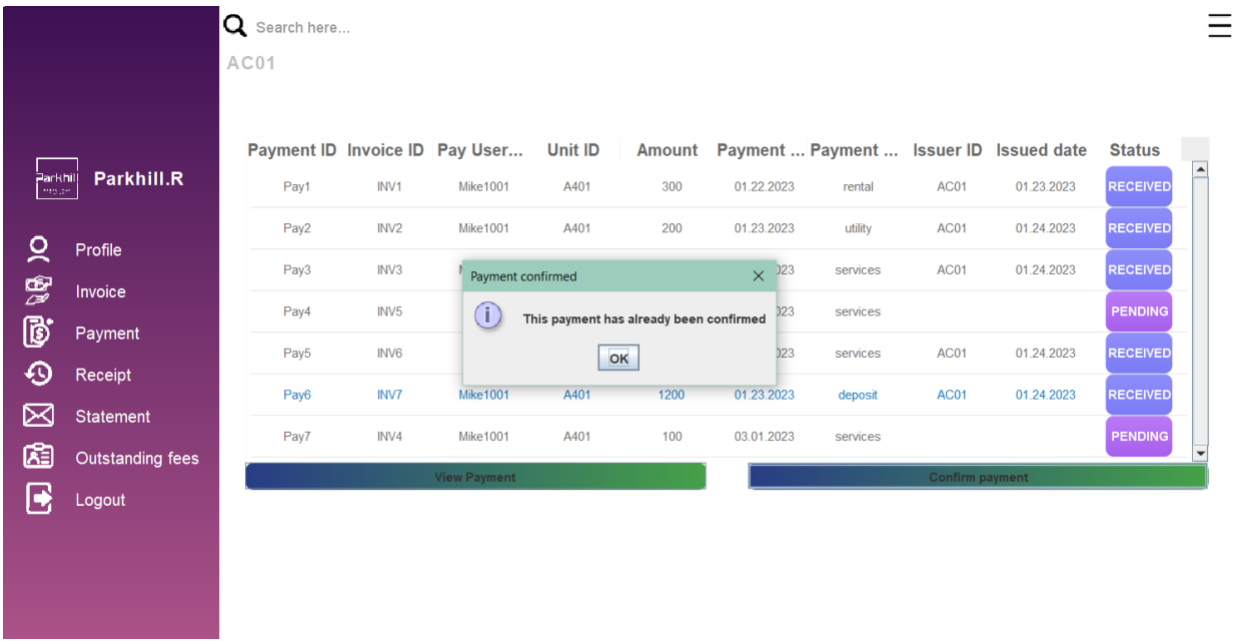
When users access the payment tab, they will be directed to a user interface that displays all payment that is waiting to be confirmed by the user. The interface also features button functions allowing users to search through everyone's payment status, view payment, and confirm payment.

8.1.4.1 View Payment



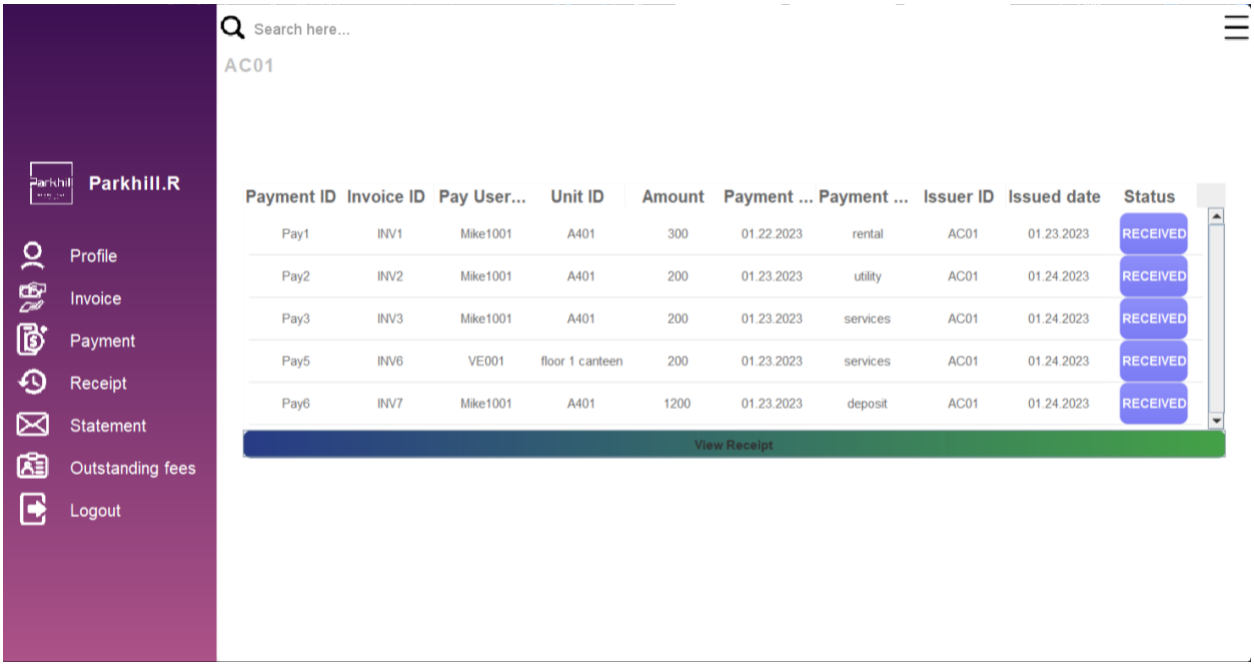
To view a payment, users must choose the one they want to view and click the "view payment" button. This will direct them to a new interface that displays their payment details.

8.1.4.2 Confirm Payment

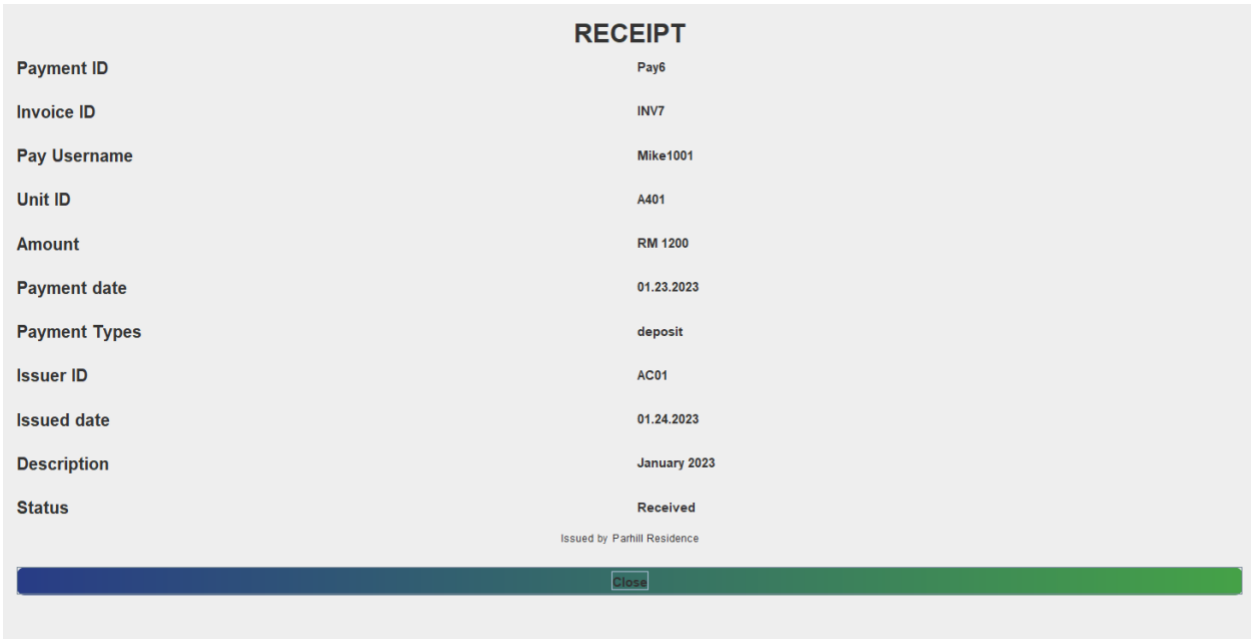


To confirm payment, users must choose the one they want to confirm and click the "confirm payment" button. Users will receive a notification to verify their decision before the payment is officially confirmed.

8.1.5 Receipt

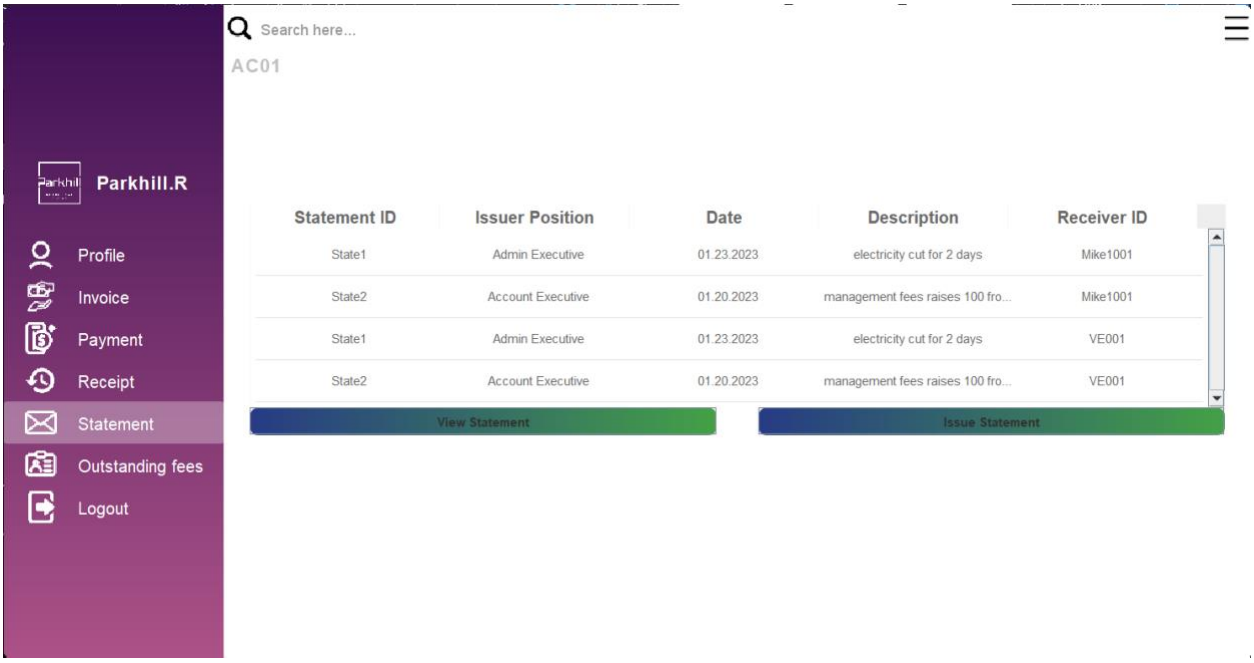


When users access the receipt tab, they will be directed to a user interface that displays all payments made by residents and vendors. The interface also features button functions allowing users to search through everyone's payment and view receipts.



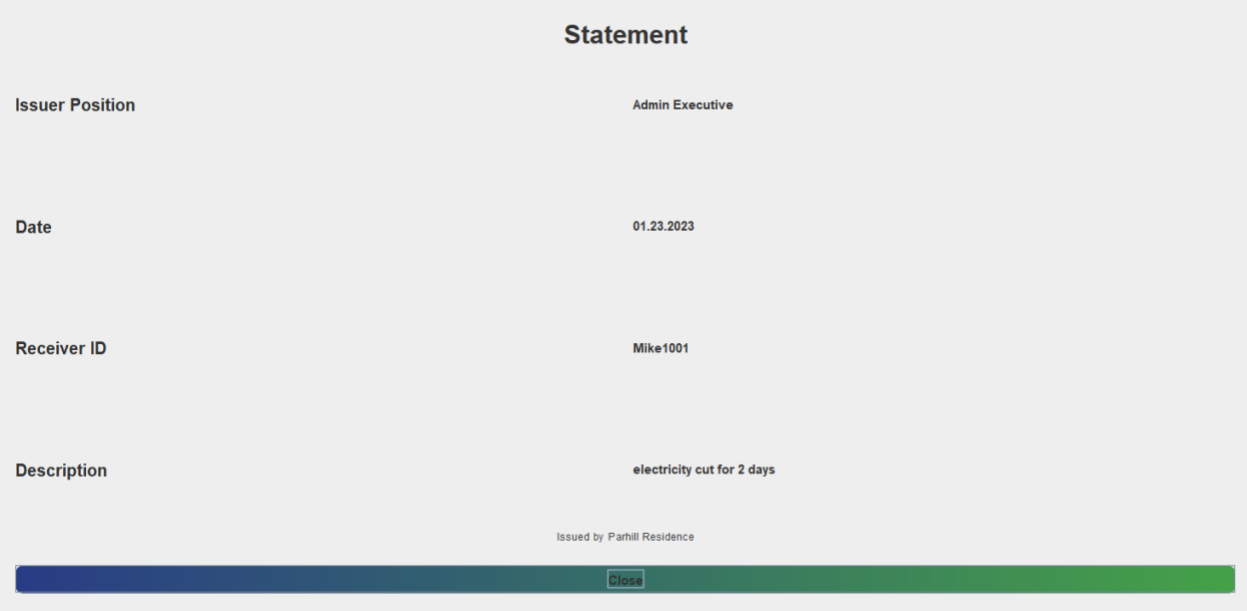
To view a receipt, users must choose the one they want to view and click the "view receipt" button. This will direct them to a new interface that displays the receipt details.

8.1.6 Statement



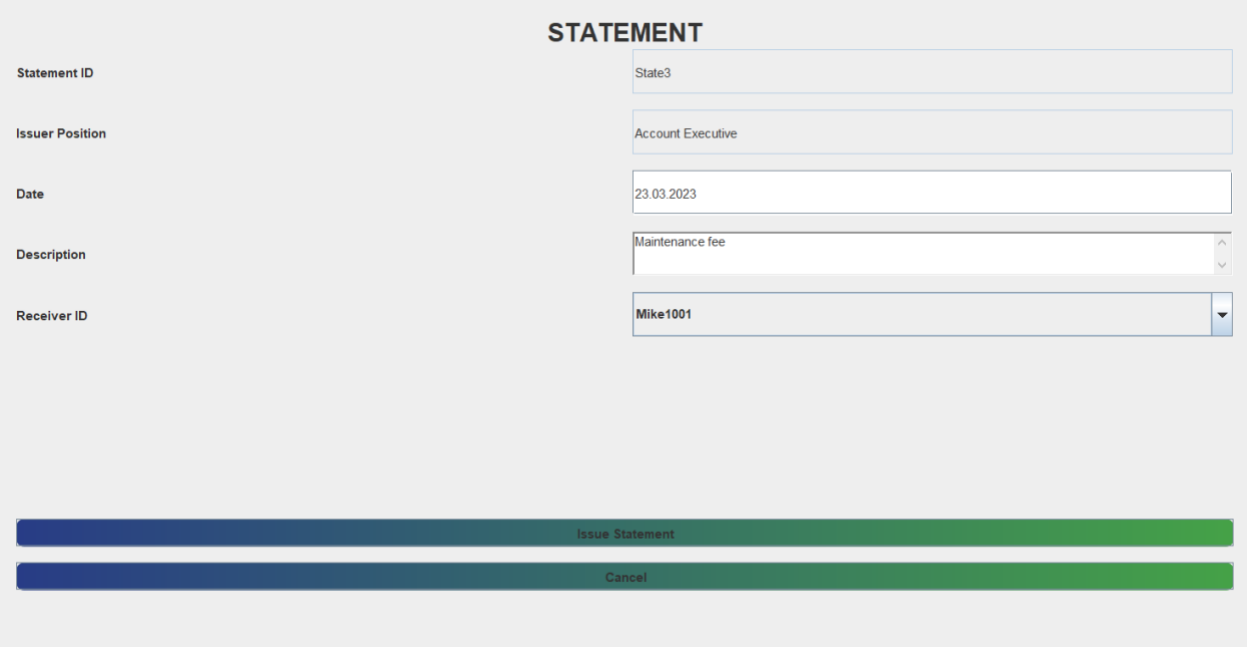
Users who navigate the statement section in the menu will be taken to this interface to access and view the financial statements. The interface also allows users to browse, view and issue a financial statement they have selected from the table.

8.1.6.1 View Statement



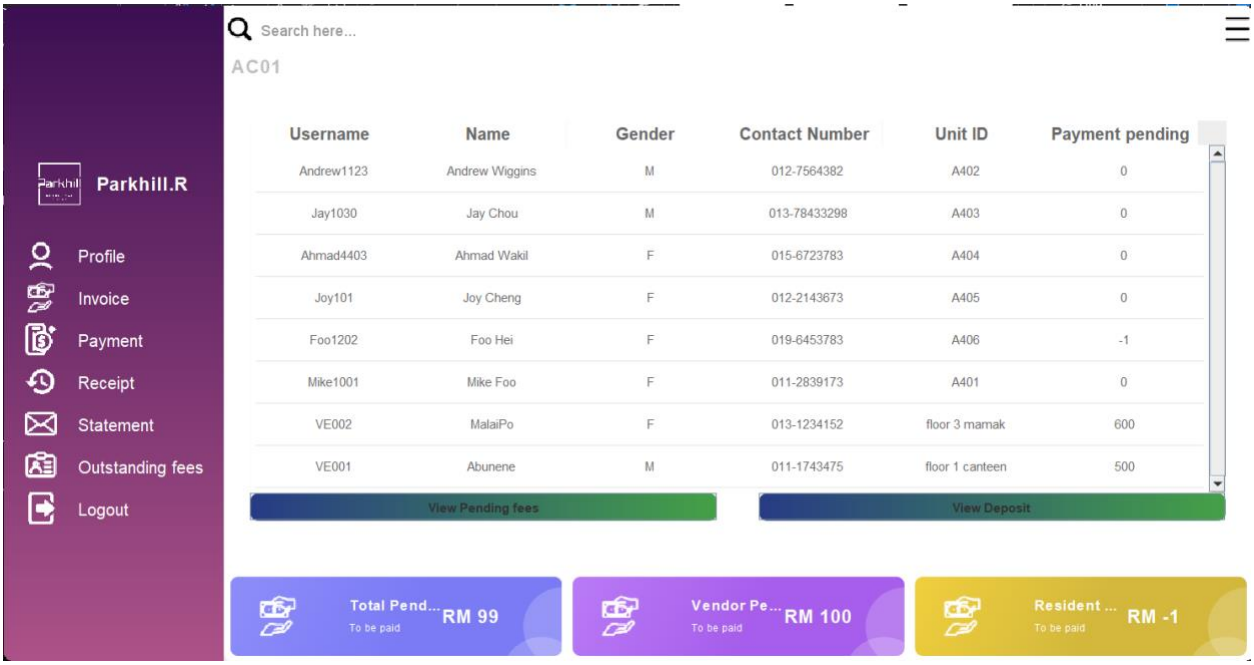
To view a statement, users must choose the one they want to view and click the "view statement" button. This will direct them to a new interface that displays the financial statement details.

8.1.6.2 Issue Statement



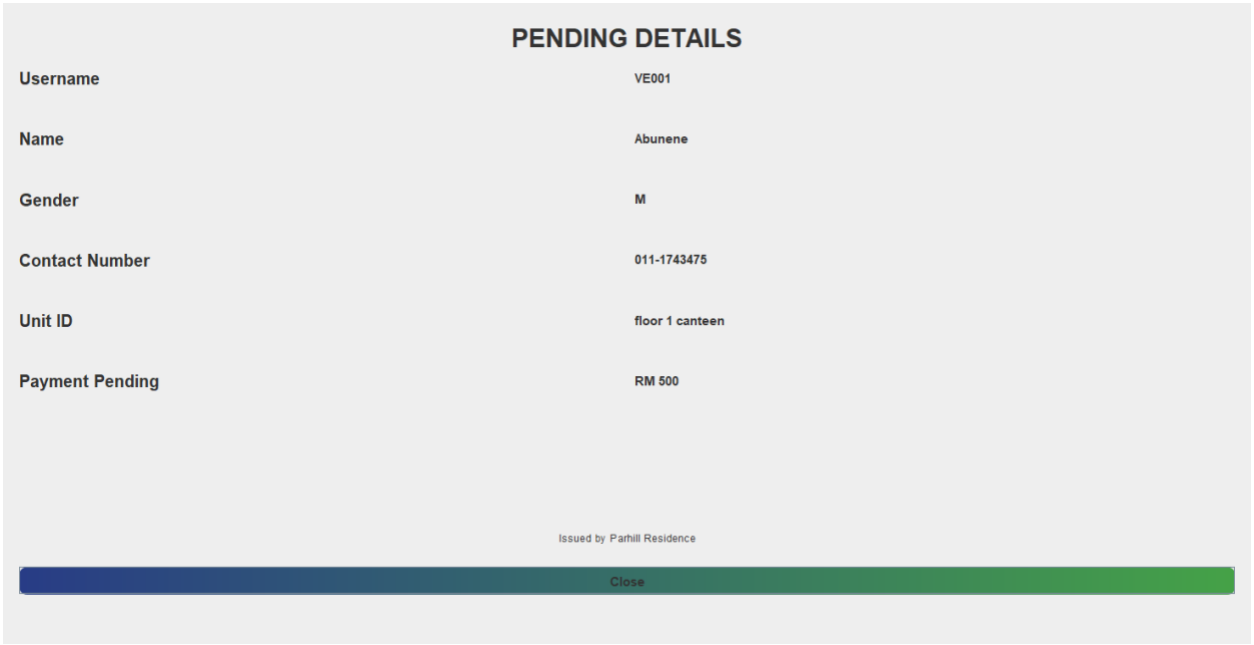
By clicking the "issue statement" button, users will be directed to a statement form, allowing them to fill in their details to issue a statement. Clicking the "issue statement" button on this form will trigger a confirmation alerting the users about their issued statement.

8.1.7 Outstanding fee



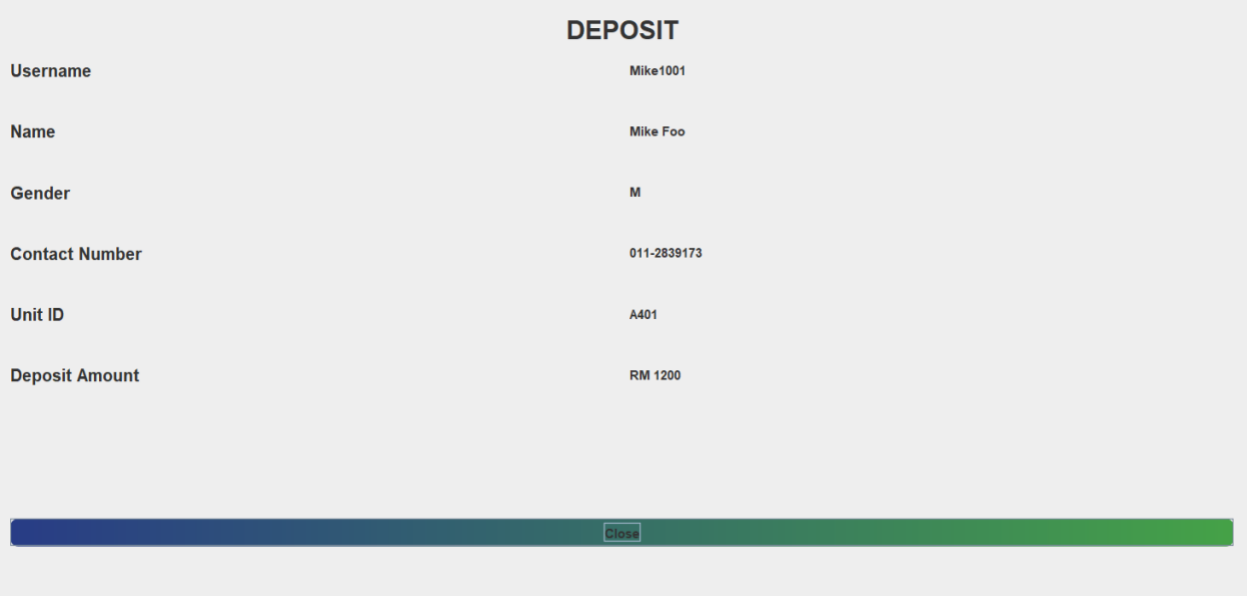
The bottom part of the interface contains boxes with reminders showing users' vendor and resident ending amounts and total pending amounts. The interface also features button functions allowing users to search pending fee records and view pending fees and deposits.

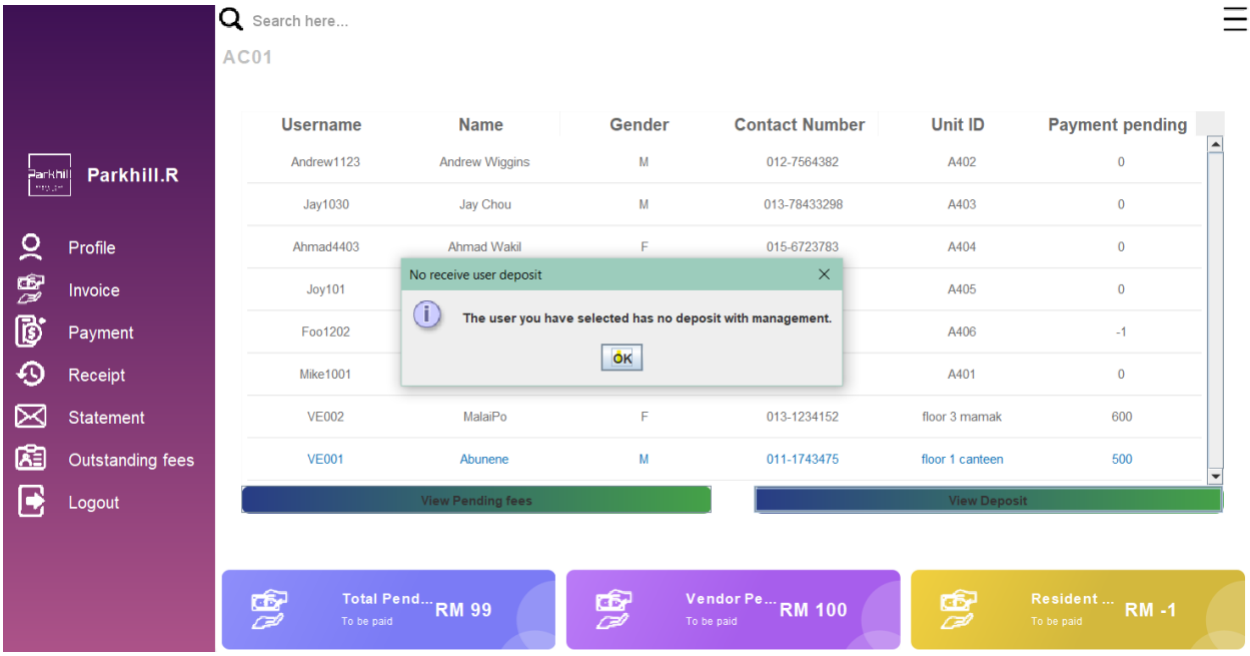
8.1.7.1 View Pending fees



To view a pending fee, users must choose the one they want to view and click the "view pending fee" button. This will direct them to a new interface that displays the pending details.

8.1.7.2 View Deposit





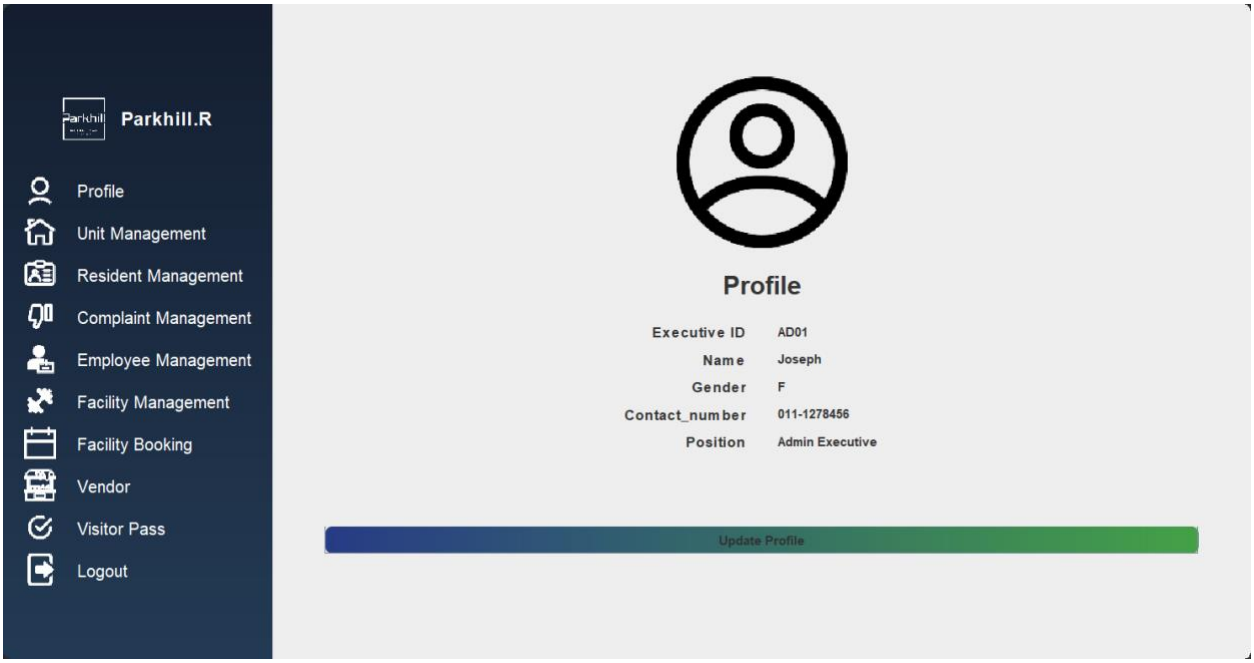
To view a deposit, users must choose the one they want to view and click the "view deposit" button. This will direct them to a new interface that displays the deposit details if there is one. Otherwise, the system will notify the users about the lack of deposit info.

8.2 Admin Executive

8.2.1 Functional requirements

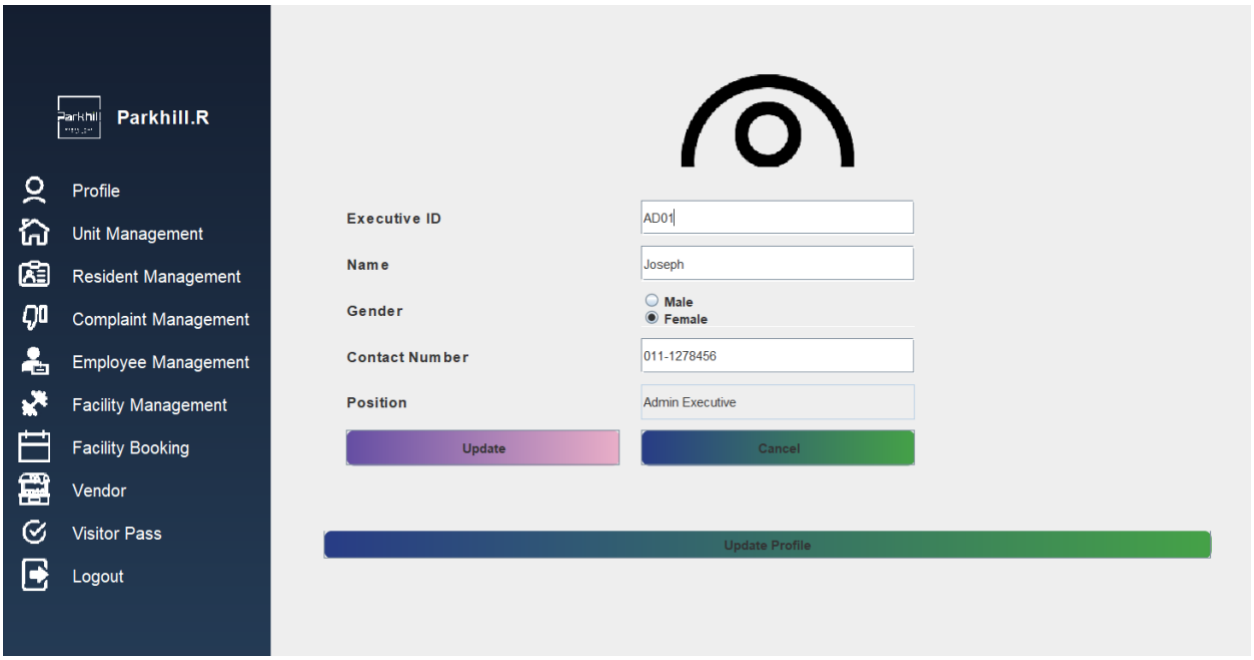
1. Profile Interface
2. Unit Management
3. Resident Management
4. Complaint Management
5. Employee Management
6. Facility Management
7. Facility Booking Management
8. Vendor Management
9. Visitor Pass Application
10. Logout

8.2.2 Profile



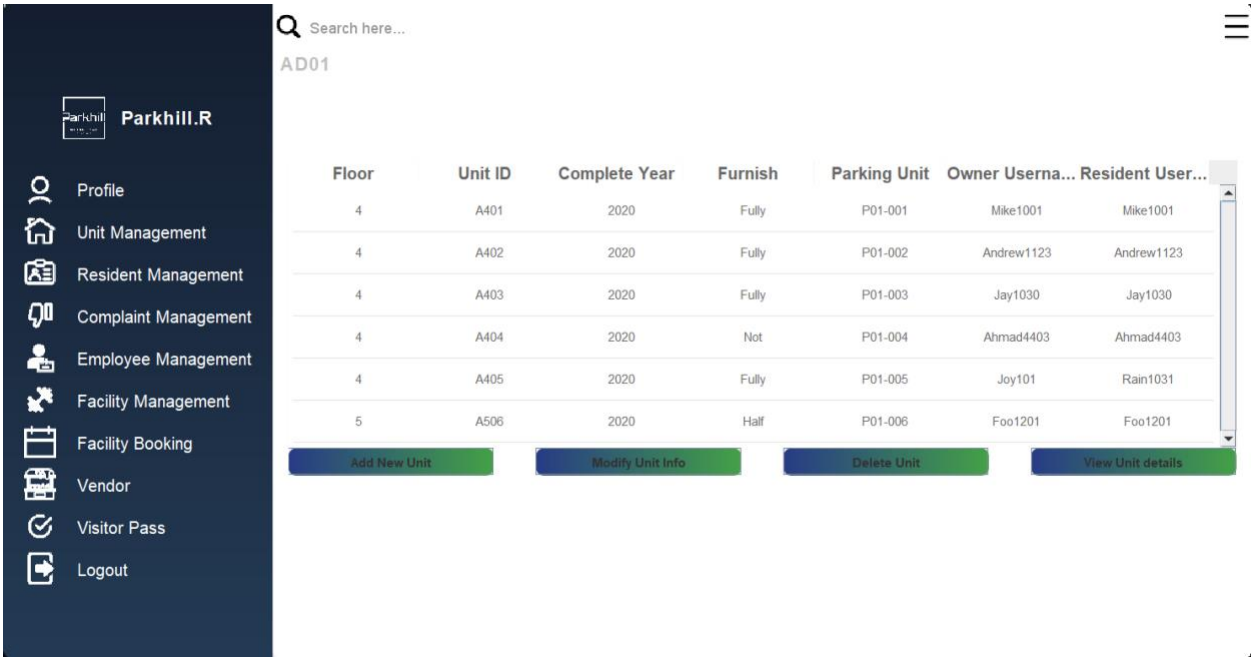
Once the admin management has successfully logged into the system, they will be directed to their designated interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the admin

management profile page is accessible through this interface and allows users to modify their profile information.



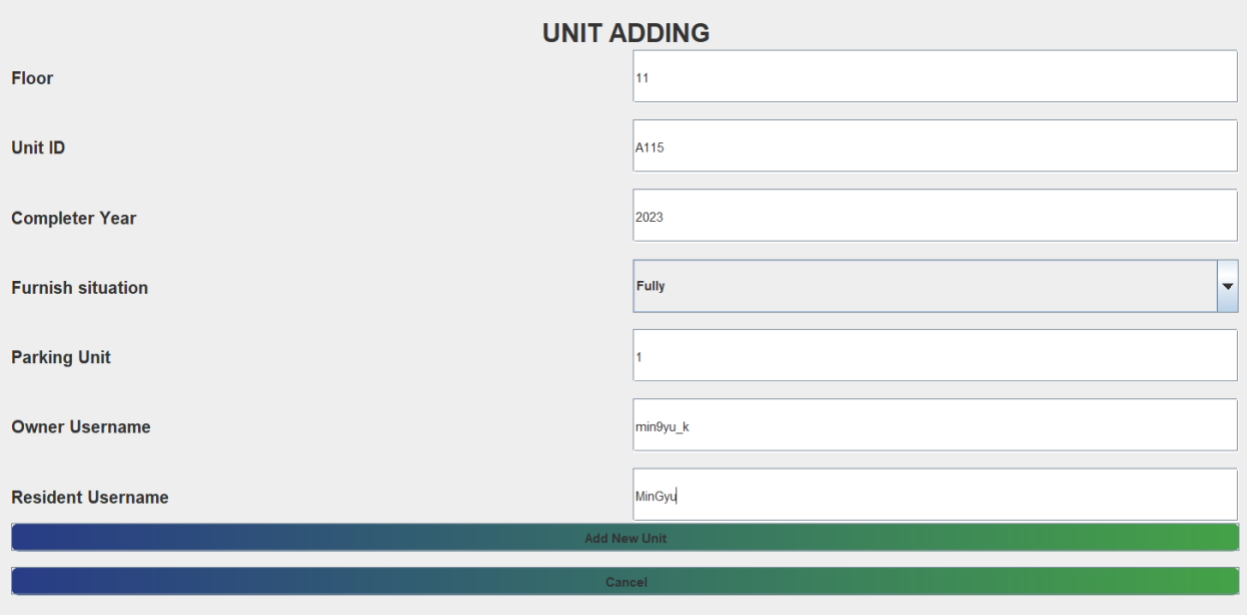
By clicking on the update profile button, the user will be led to this interface which allows them to change their username, name, gender and contact number. After clicking the "Update" button, a confirmation notice will alert users that the information is updated.

8.2.3 Unit Management



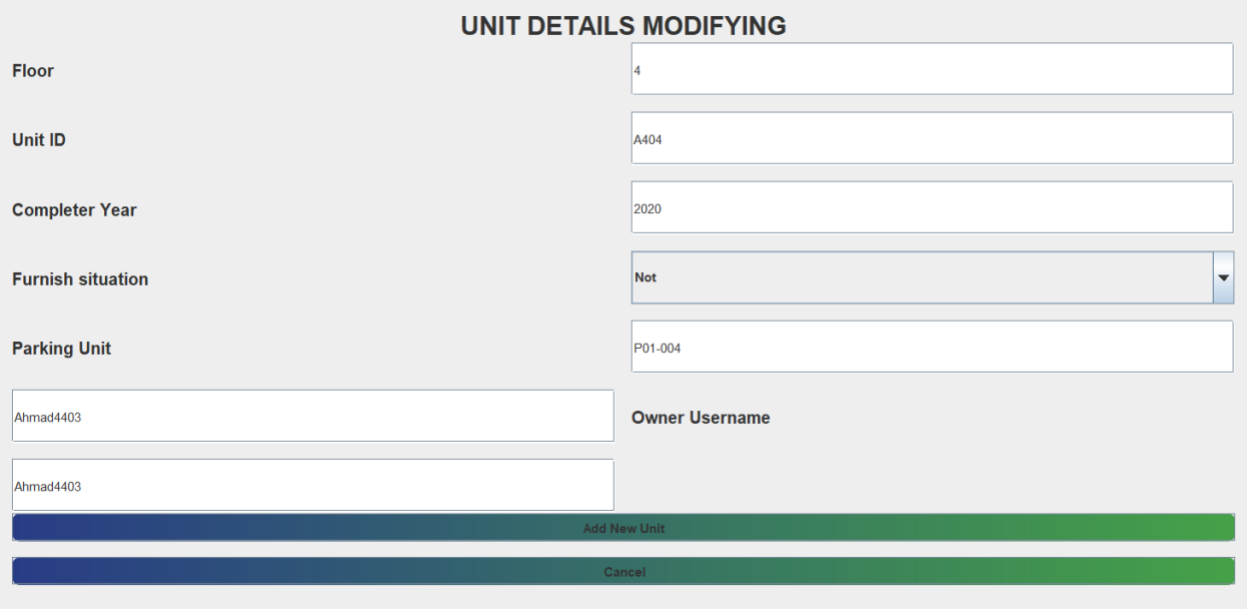
Users who navigate the unit management section in the menu will be taken to this interface to access and view the list of unit records. The interface also allows users to search for specific units, add a new unit, modify unit info, delete a unit, and view unit details on the unit they selected from the table.

8.2.3.1 Add New Unit



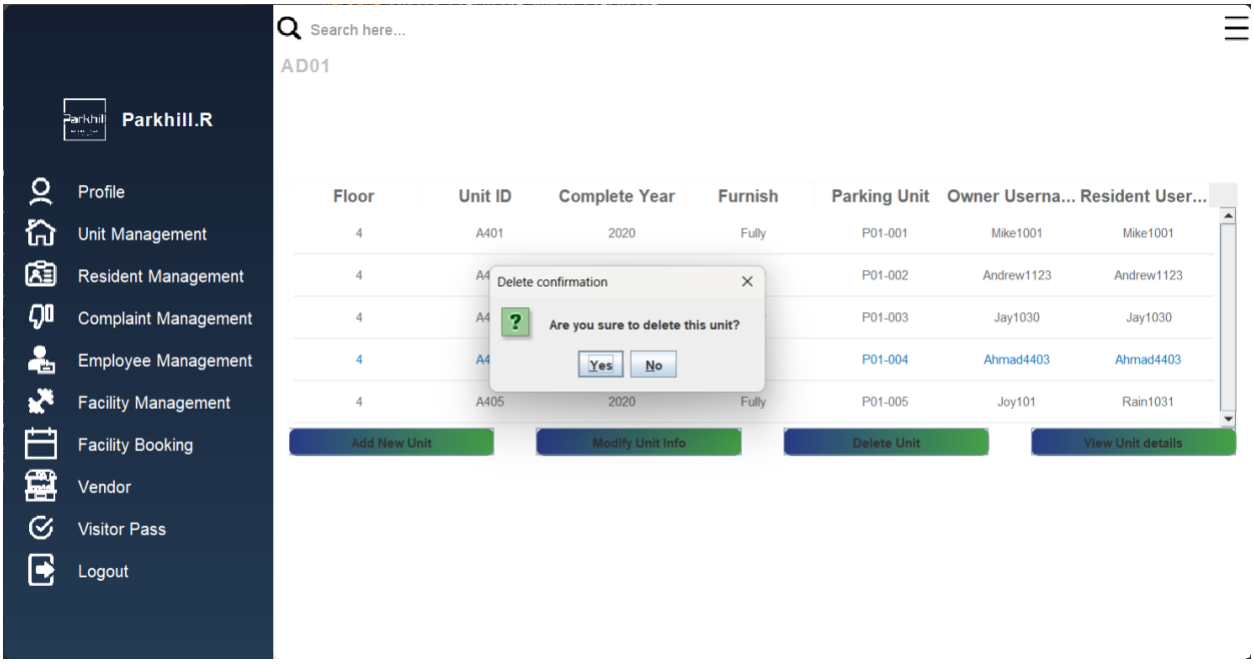
When users click the "add new unit" button, they will be taken to a form where they can input the necessary details to create a new unit. Once they have filled in the form and clicked the "add new unit" button, a confirmation message will notify the users that the unit has been added successfully. If the unit does not belong to a registered user, the system will send a notification indicating that the resident information is incomplete.

8.2.3.2 Modify Unit Info



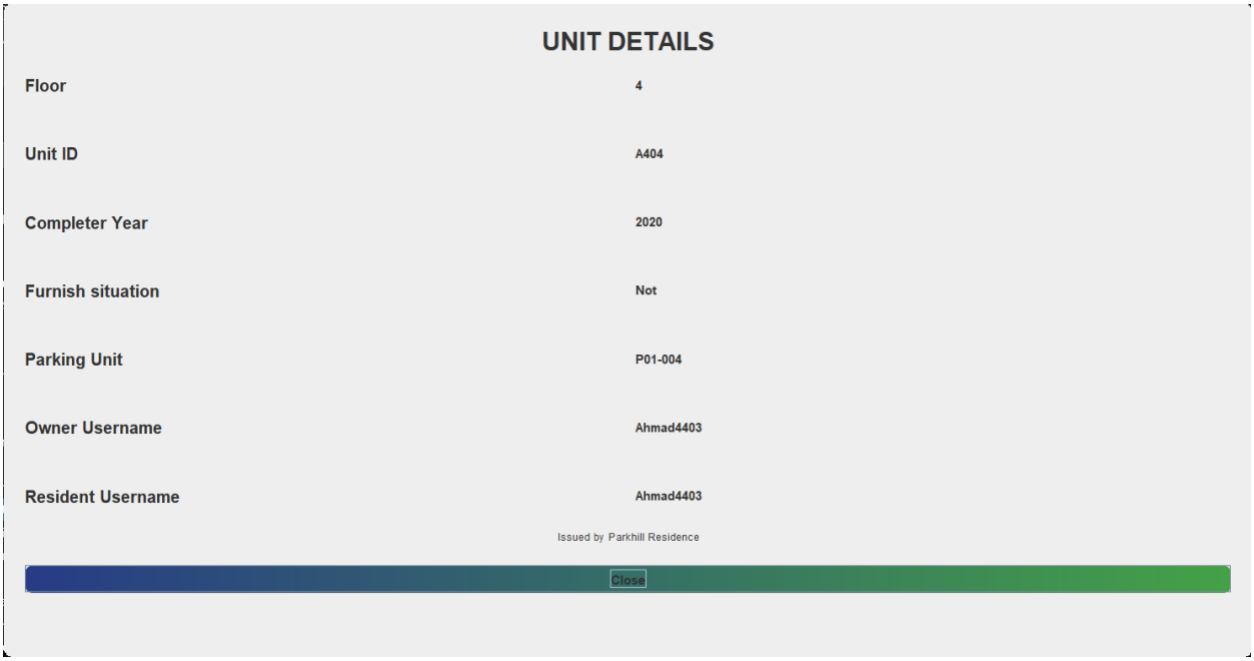
To modify a unit, users must click on the unit they wish to modify and then proceed by clicking the "modify unit" button. The users will then be directed to an interface to modify unit details. Afterwards, users can click the "add new unit" button to submit their modified unit details.

8.2.3.3 Delete Unit



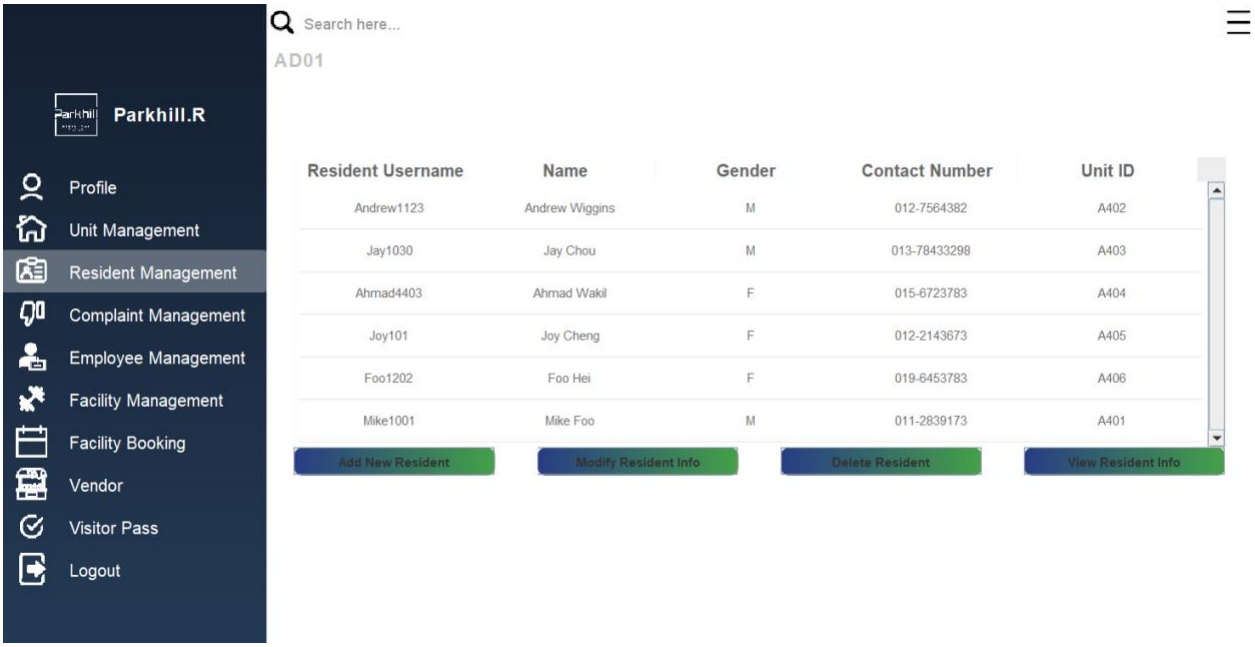
Users must select the unit they want to delete and click the "delete unit" button to delete a unit. This action will lead them to a new interface where they can confirm the deletion of their unit details. Users will receive a notification to verify their decision before the unit is officially deleted.

8.2.3.4 View Unit Details



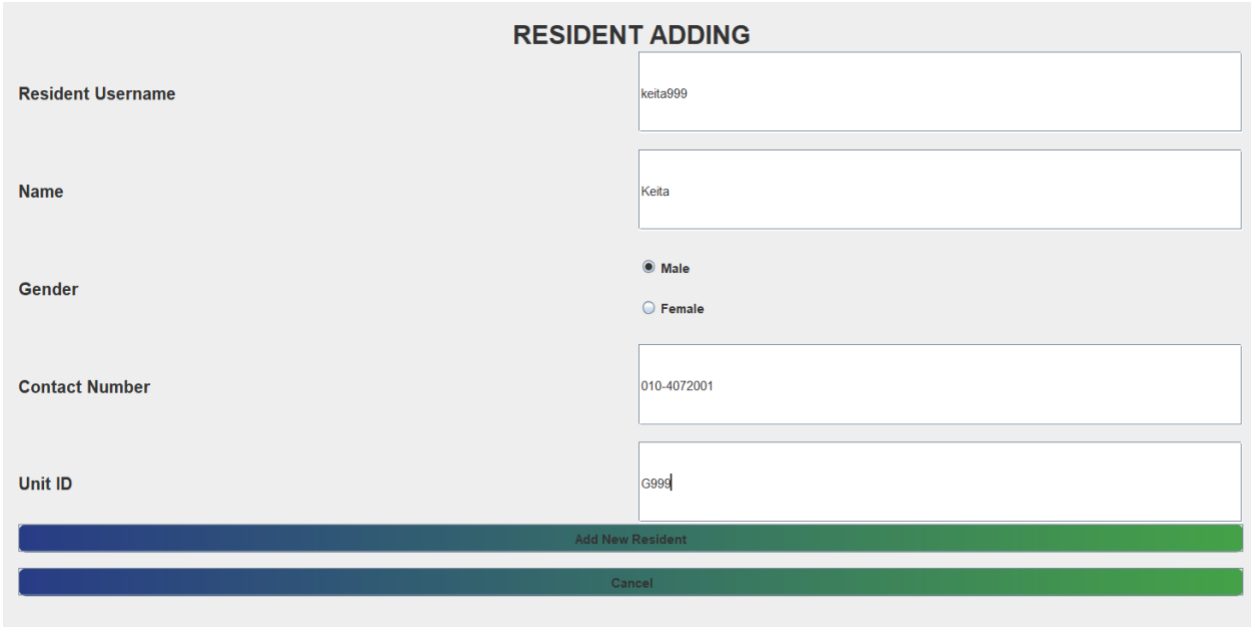
To view a unit, users must choose the one they want to view and click the "view unit detail" button. This will direct them to a new interface that displays the unit details.

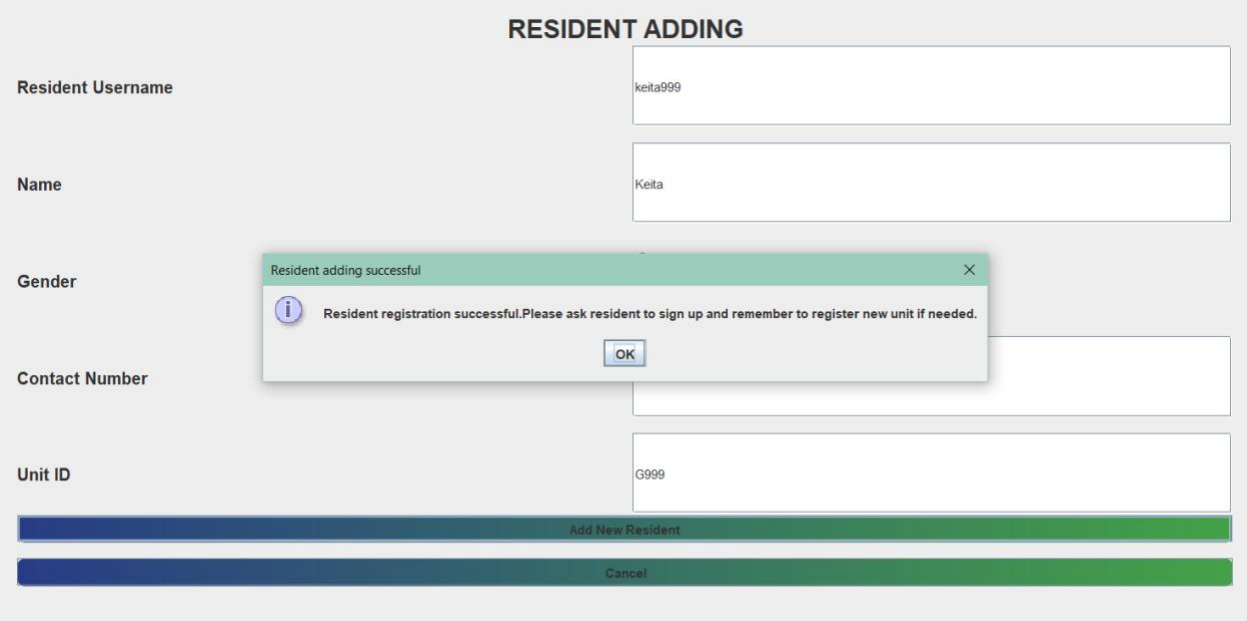
8.2.4 Resident management



Users who navigate the resident management section in the menu will be taken to this interface to access and view the list of residents' records. The interface also allows users to search for specific residents, add a new resident, modify resident info, delete resident info, and view resident details on the unit they selected from the table.

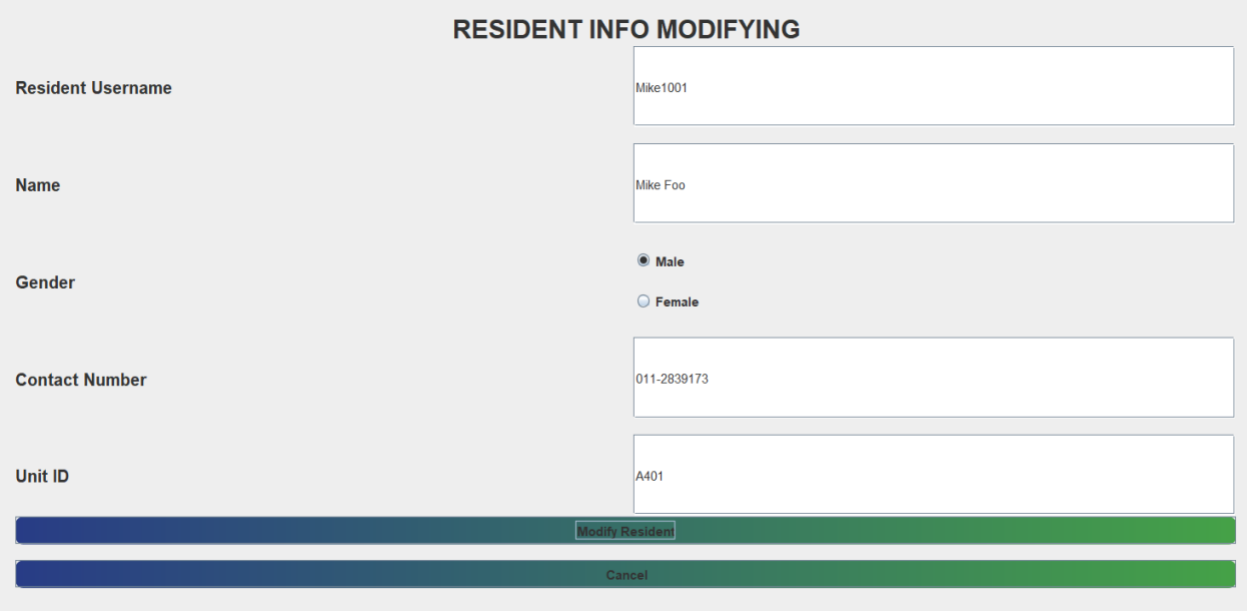
8.2.4.1 Add New Resident





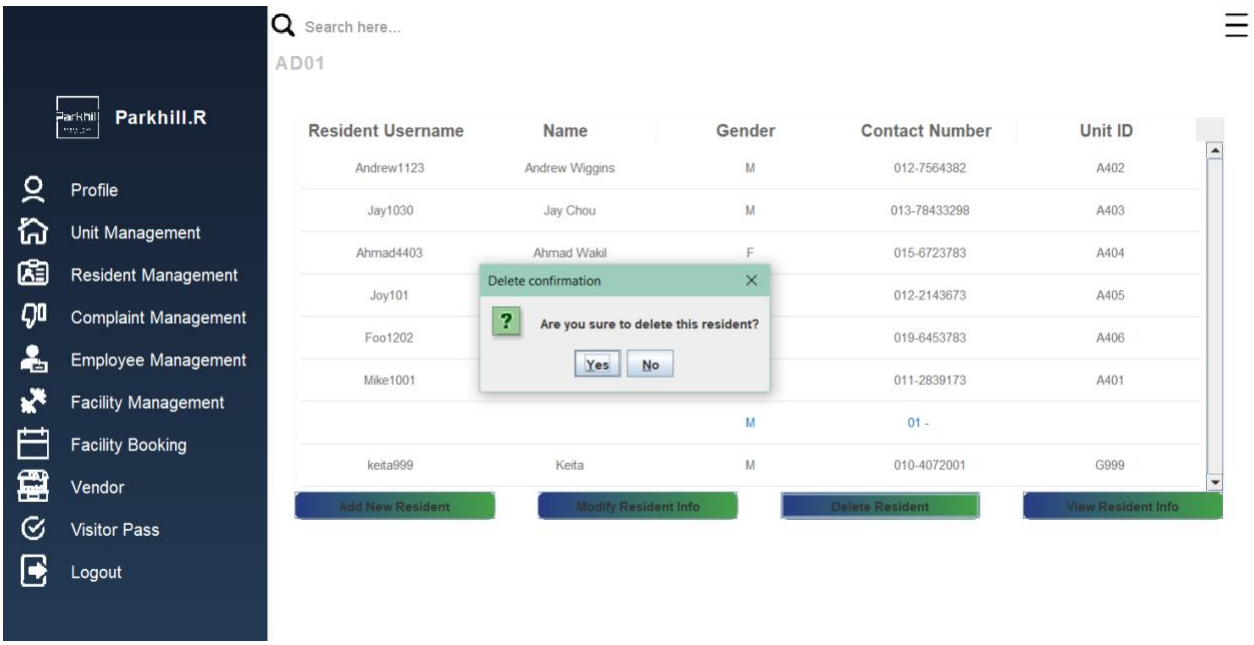
When users click the "add new resident" button, they will be taken to a form where they can input the necessary details to add a new resident. Once they have filled in the form and clicked the "add new resident" button, a confirmation message will notify the users that the resident has been added successfully.

8.2.4.2 Modify Resident Info



To modify a resident, users must click on the resident they wish to modify and then proceed by clicking the "modify resident" button. The users will then be directed to an interface to modify resident details. Afterwards, users can click the "add new resident" button to submit their modified resident details.

8.2.4.3 Delete Resident

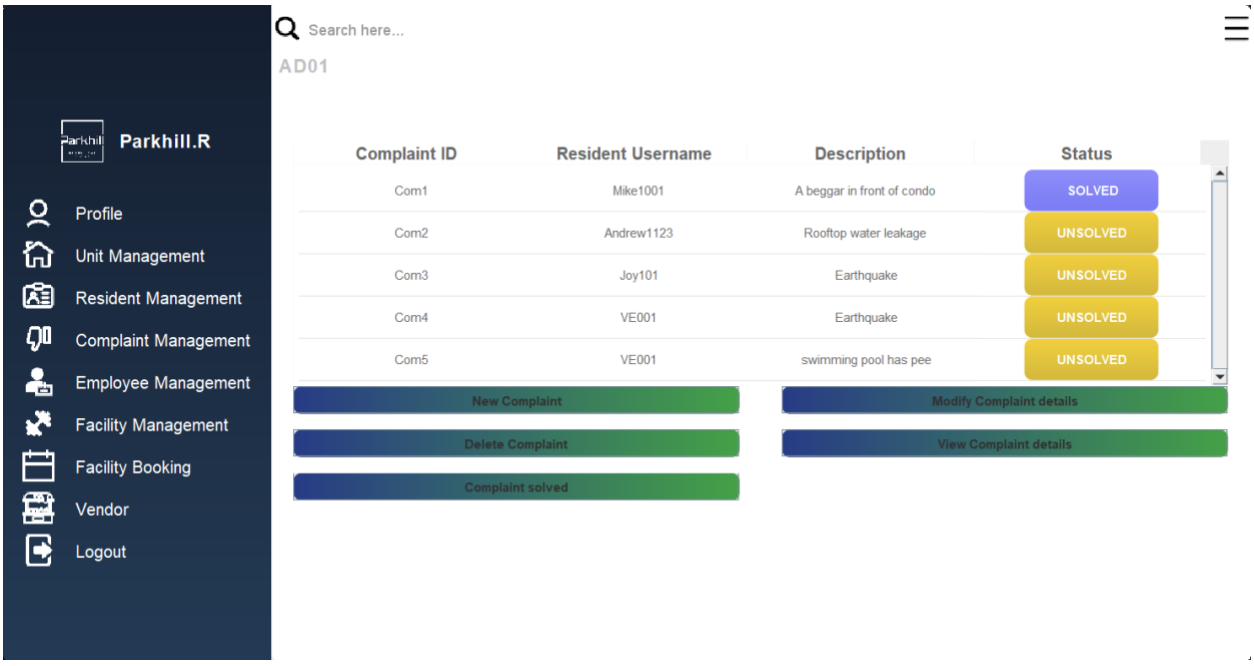


Users must select the resident they want to delete and click the "delete resident" button to delete a resident. This action will lead them to a new interface where they can confirm the deletion of their resident details. Users will receive a notification to verify their decision before the resident is officially deleted.

8.2.4.4 View Resident Info

To view a resident, users must choose the one they want to view and click the "view resident" button. This will direct them to a new interface that displays the resident details.

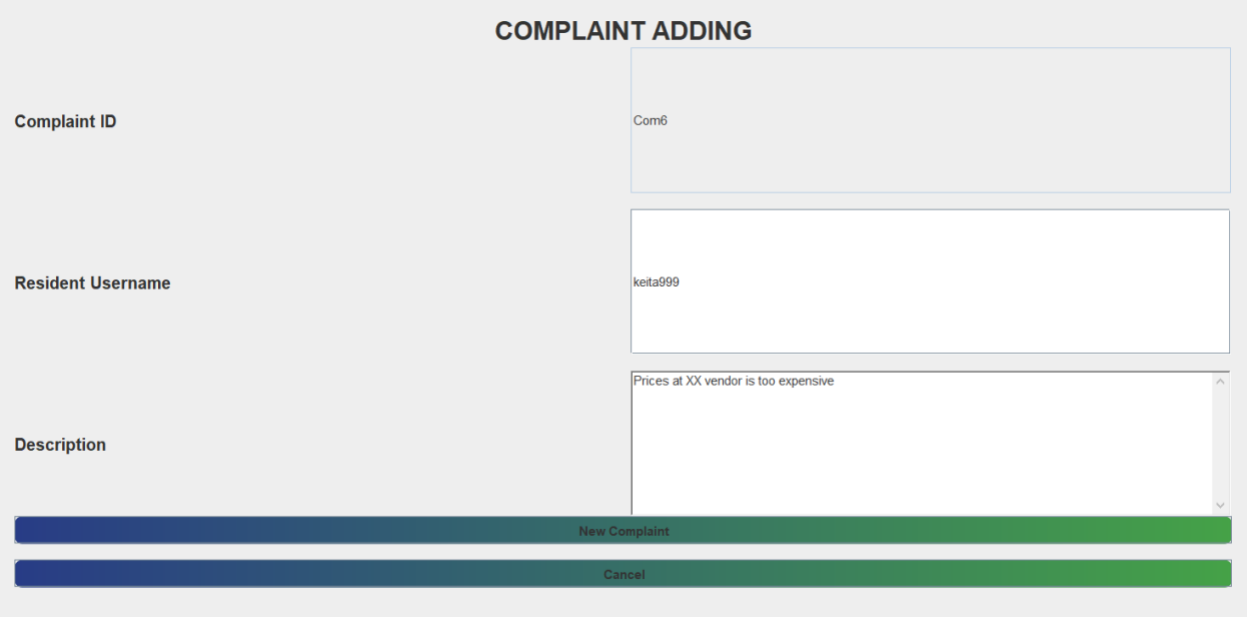
8.2.5 Complaint management

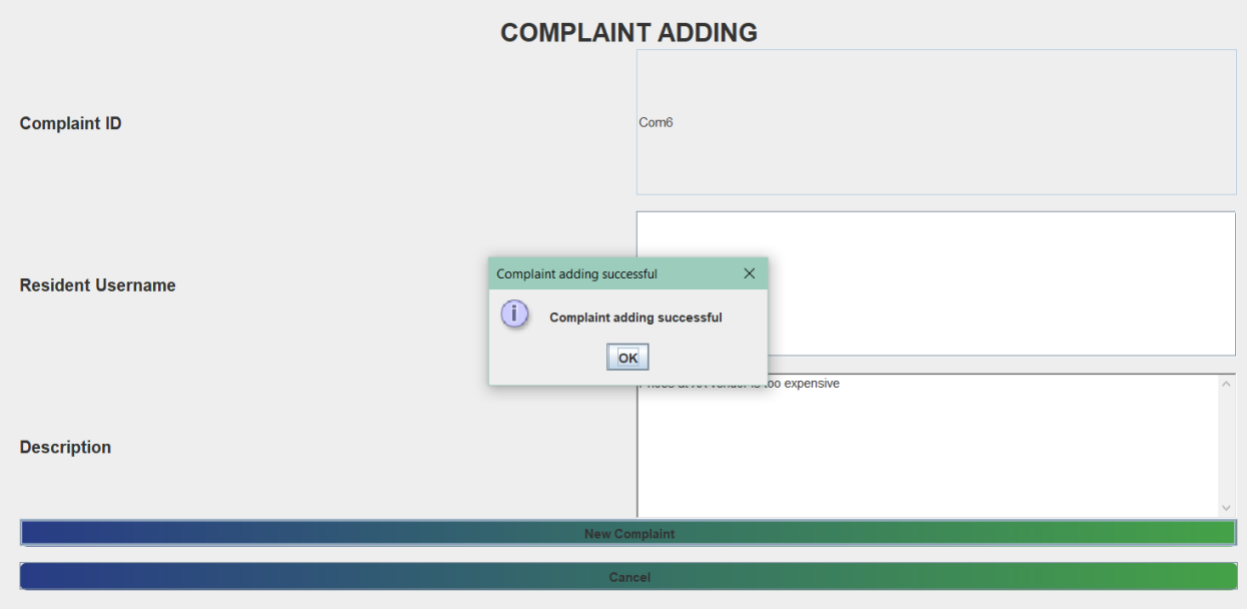


Users who navigate the complaint management section in the menu will be taken to this interface to access and view the list of complaint records. The interface also allows users to search for

specific complaints, add a new complaint, modify complaint details, delete complaints, view complaint details, and approve complaint solved on the record they selected from the table.

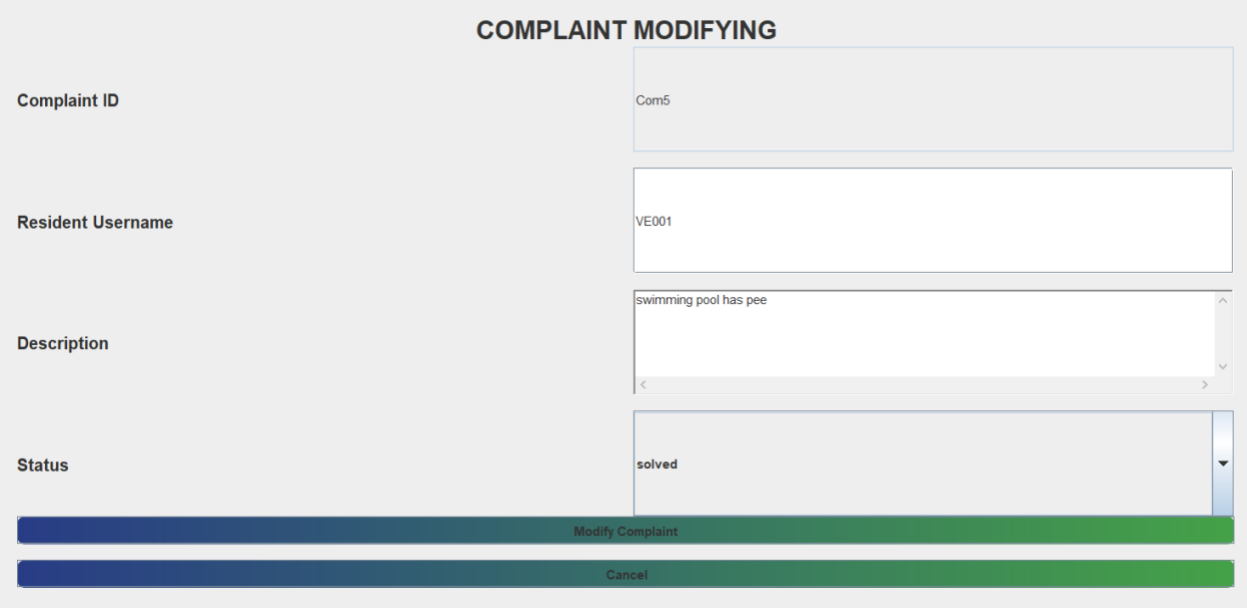
8.2.5.1 New Complaint





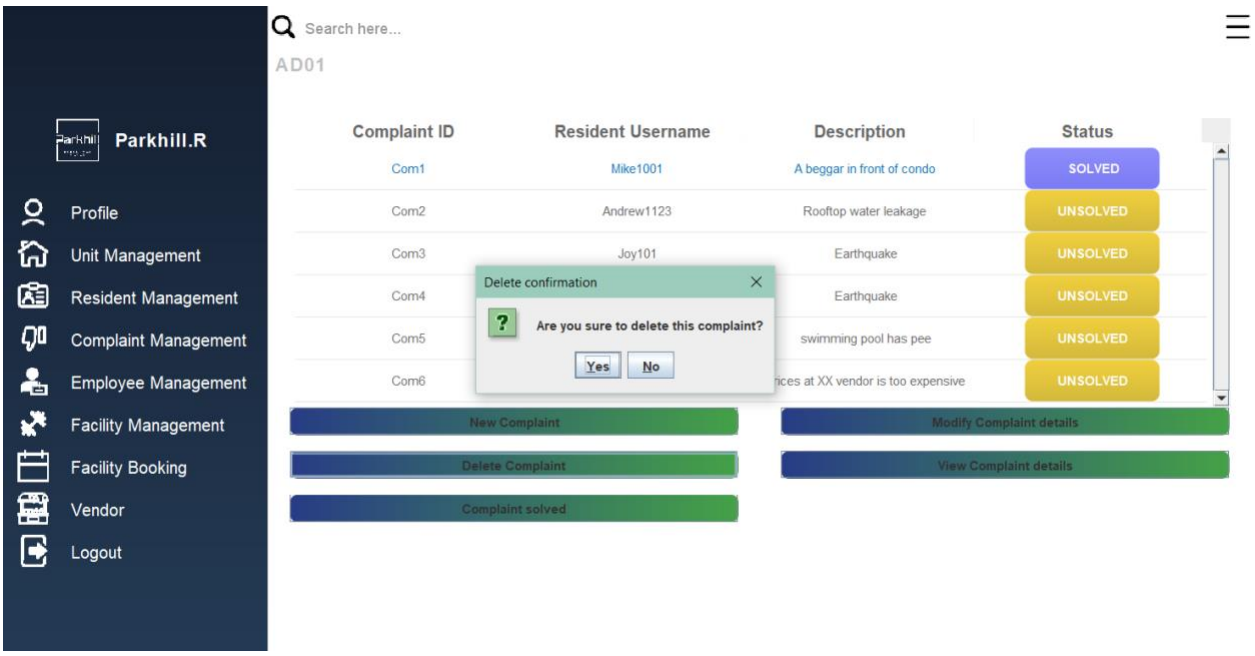
When users click the "add new complaint" button, they will be taken to a form where they can input the necessary details to add a new complaint. Once they have filled in the form and clicked the "add new complaint" button, a confirmation message will notify the users that the complaint has been added successfully.

8.2.5.2 Modify Complaint details



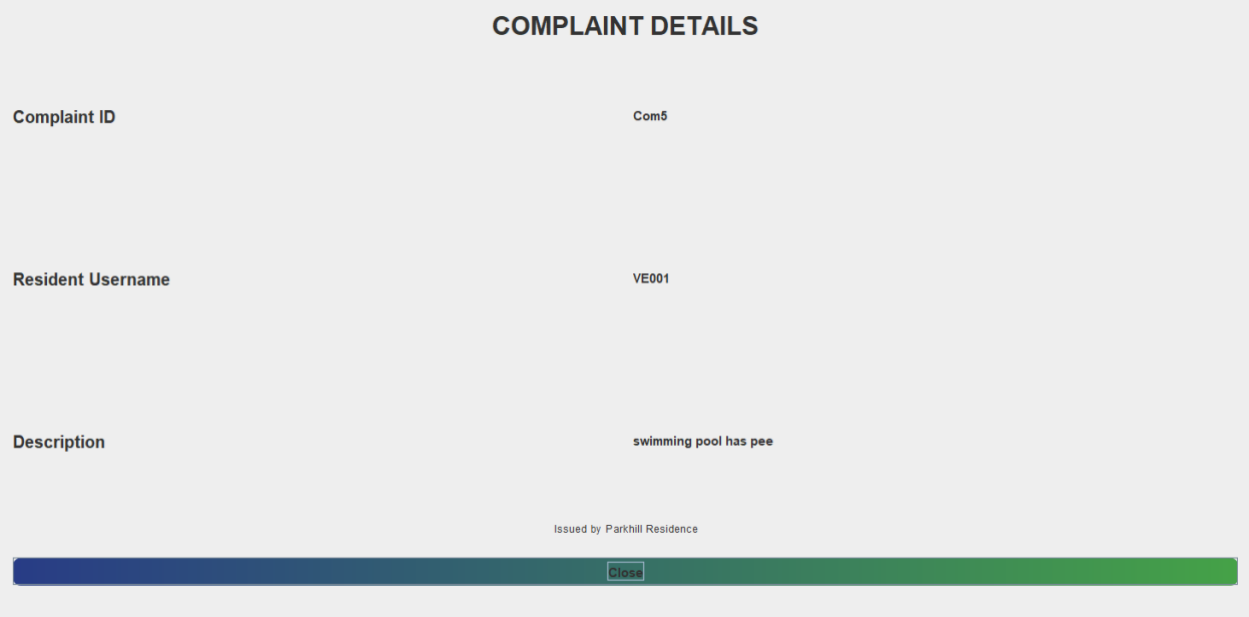
To modify a complaint, users must click on the complaint they wish to modify and then proceed by clicking the "modify complaint" button. The users will then be directed to an interface to modify complaint details. Afterwards, users can click the "modify complaint" button to submit their modified complaint details.

8.2.5.3 Delete Complaint



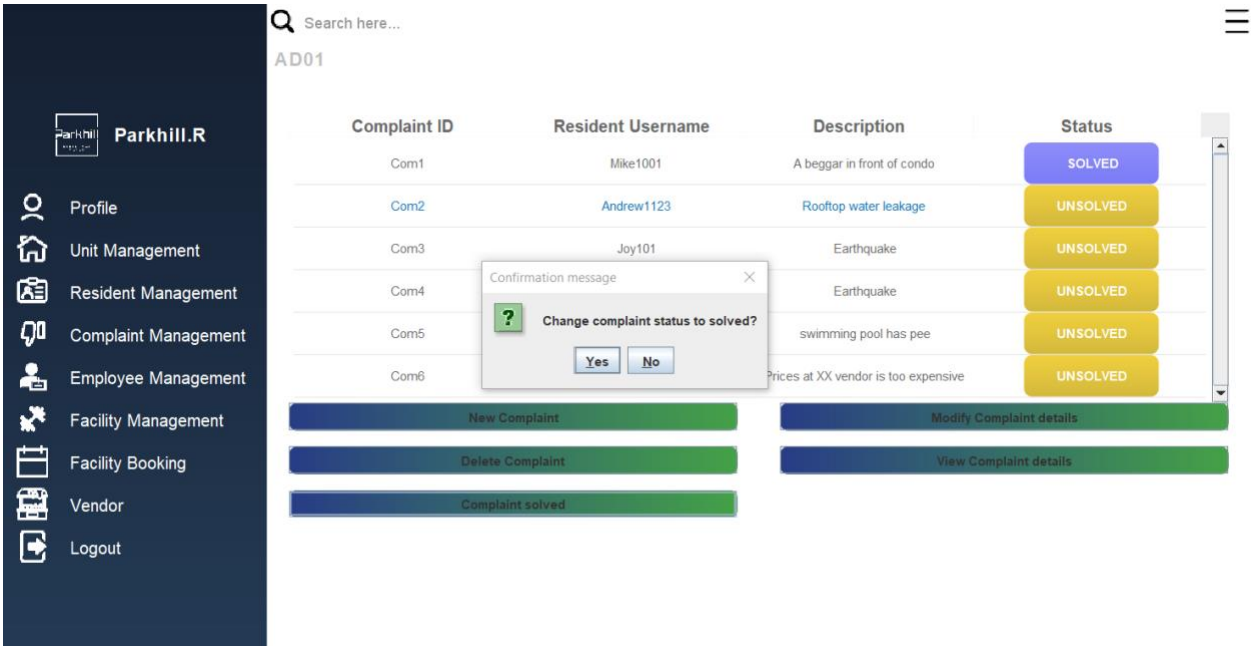
Users must select the complaint they want to delete and click the "delete complaint" button to delete a complaint. This action will lead them to a new interface where they can confirm the deletion of their complaint details. Users will receive a notification to verify their decision before the complaint is officially deleted.

8.2.5.4 View Complaint details



To view a complaint, users must choose the one they want to view and click the "view complaint" button. This will direct them to a new interface that displays the complaint details.

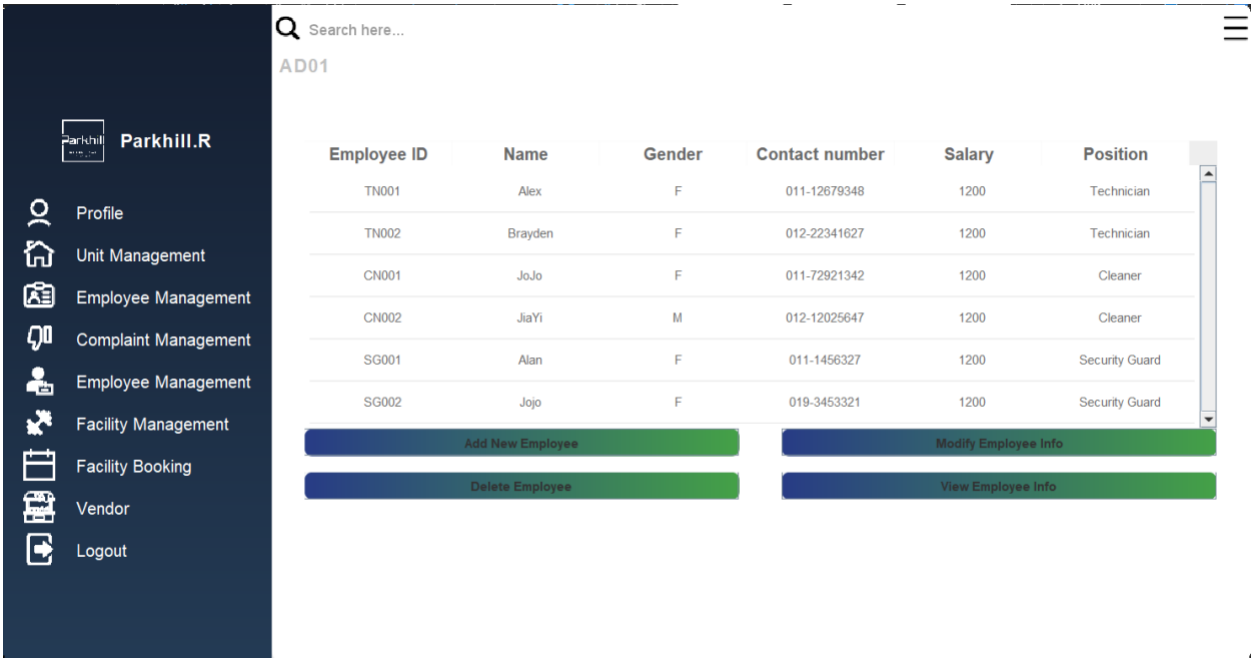
8.2.5.5 Complaint solved



To change the status of a complaint to solve, the user must first pick which complaint they would want to change the status to solve. Then, by clicking the "complaint solved" button, the system

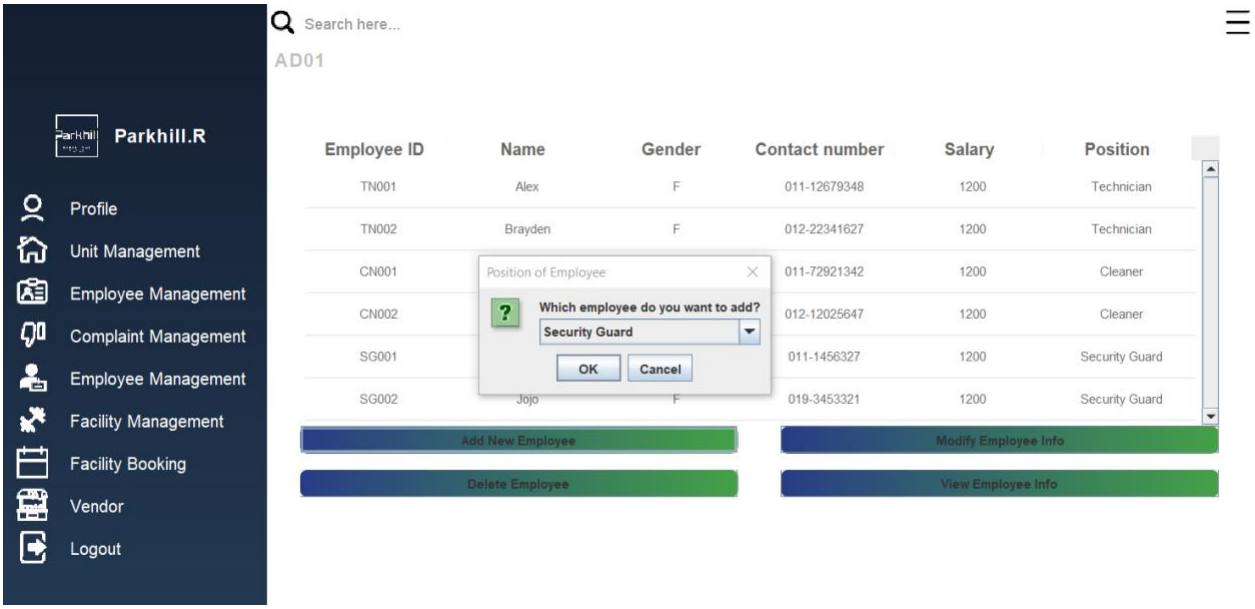
will change the complaint to solved. Users will receive a notification to verify their decision before the complaint status officially changes.

8.2.6 Employee management

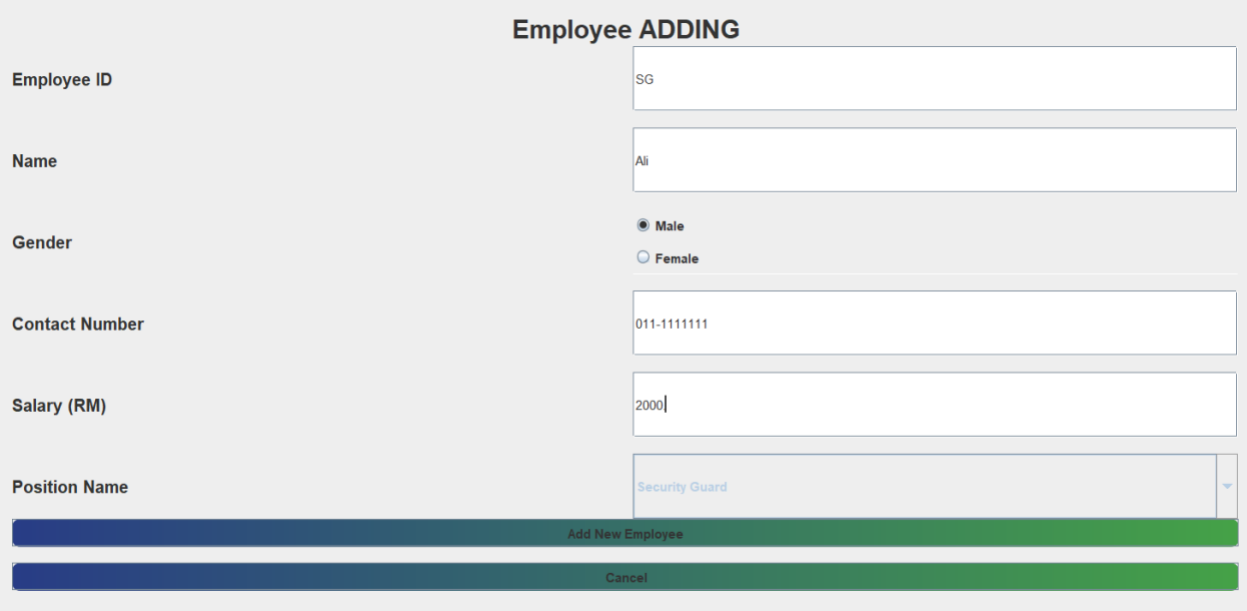


Users who navigate the employee management section in the menu will be taken to this interface to access and view the list of employee records. The interface also allows users to search for specific employees, add new employees, modify employee details, delete employees, and view employee details on the record they selected from the table.

8.2.6.1 Add New Employee

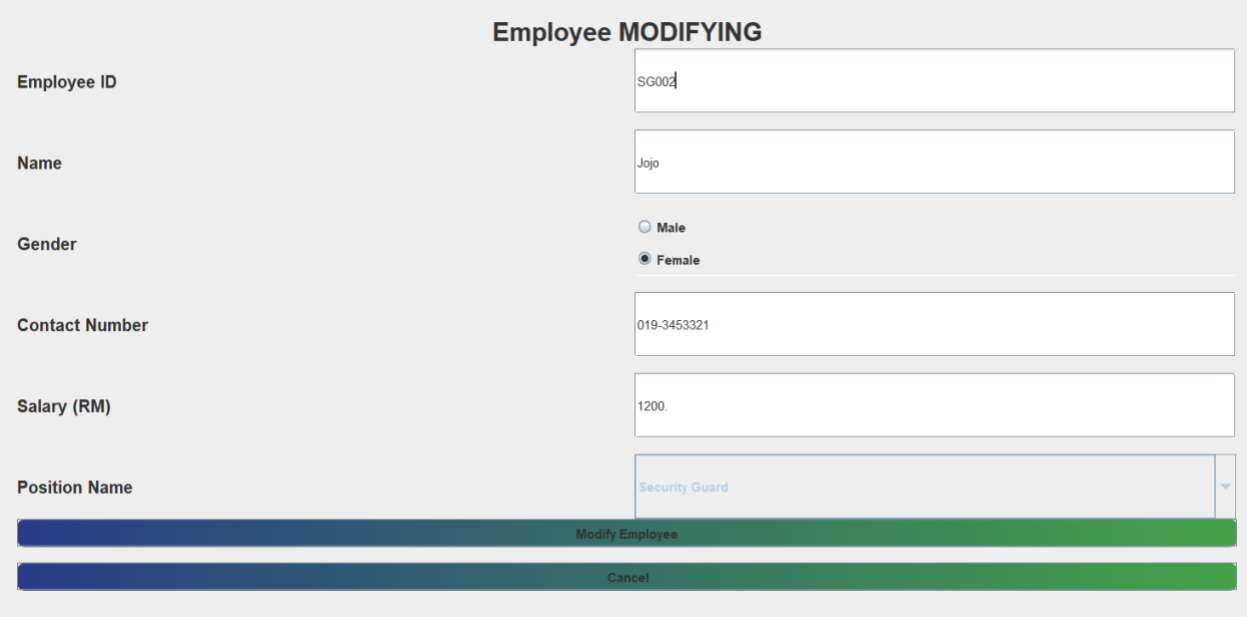


When users click the "add new employee" button, they will be given a choice of the type of employee they want to add.



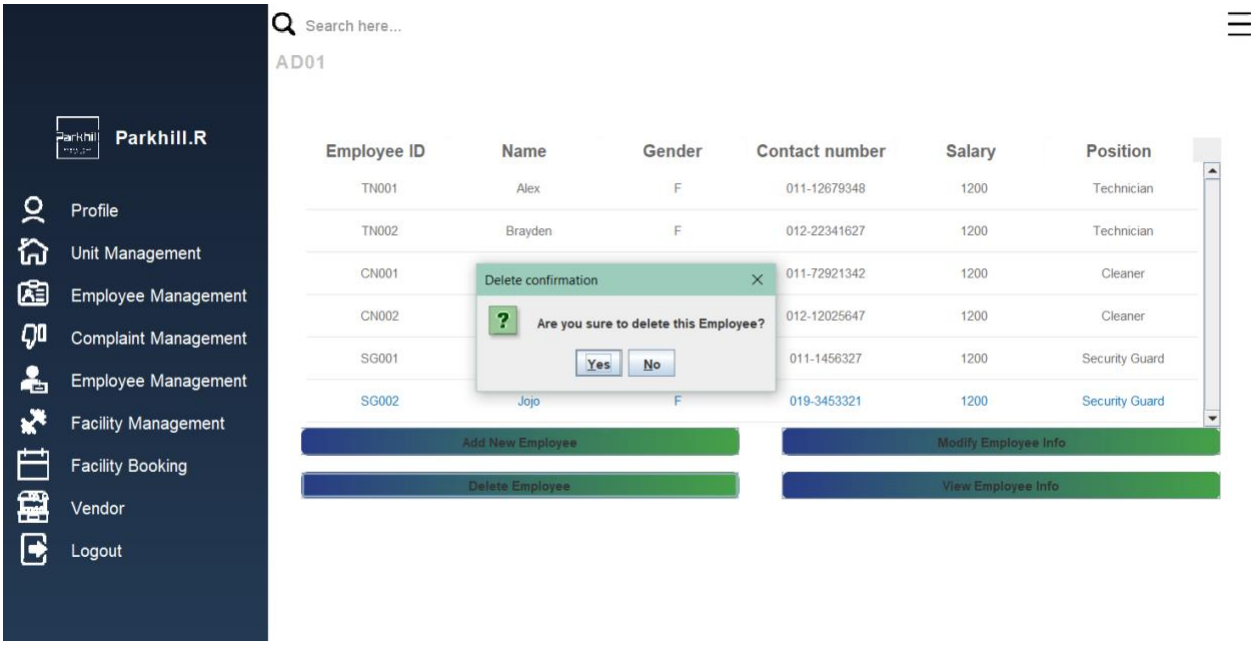
Once they have filled in the form and clicked the "add new employee" button, a confirmation message will notify the users that the employee has been added successfully.

8.2.6.2 Modify Employee Info



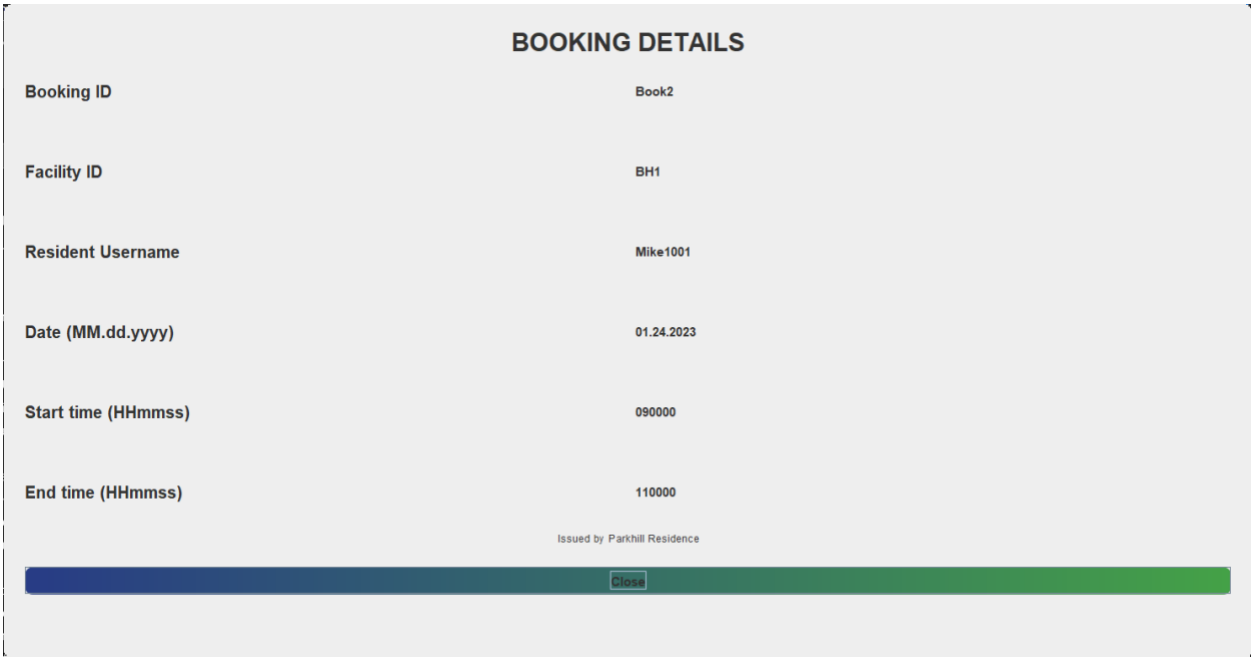
To modify an employee record, users must click on the employee they wish to modify and then proceed by clicking the "modify employee" button. The users will then be directed to an interface to modify employee details. Afterwards, users can click the "modify employee" button to submit their modified employee details.

8.2.6.3 Delete Employee



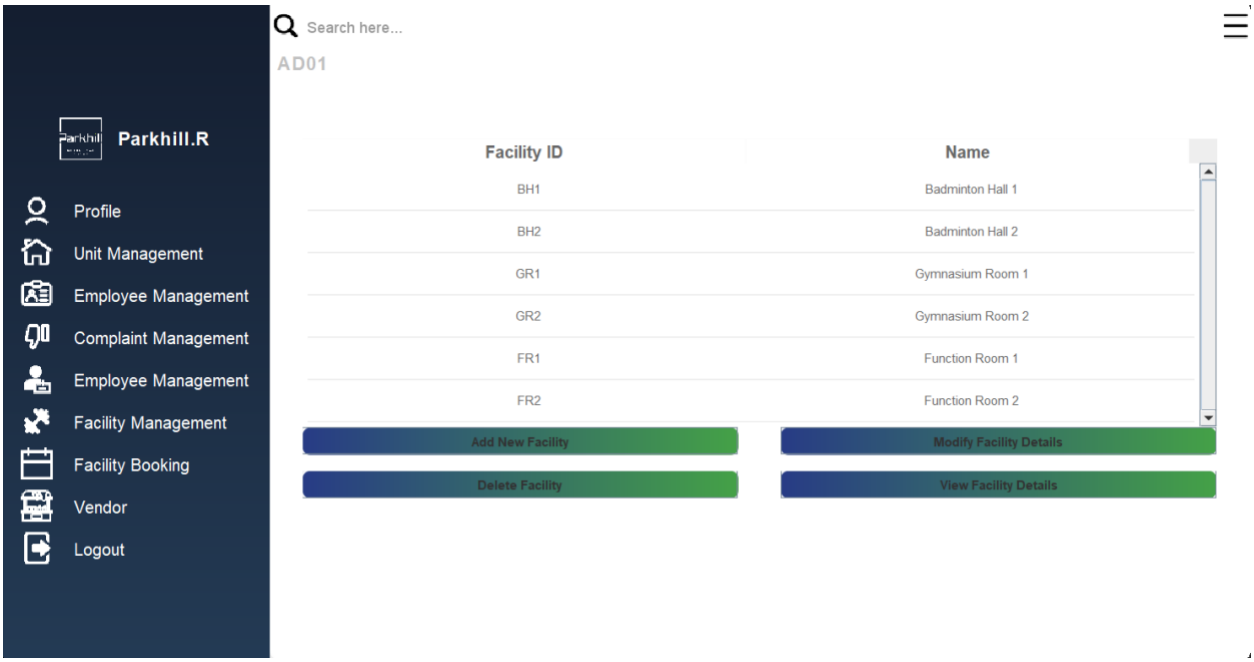
Users must select the employee they want to delete and click the "delete employee" button to delete an employee record. This action will lead them to a new interface where they can confirm the deletion of their employee details. Users will receive a notification to verify their decision before the employee record is officially deleted.

8.2.6.4 View Employee Info



To view an employee record, users must choose the one they want to view and click the "view employee" button. This will direct them to a new interface that displays the employee details.

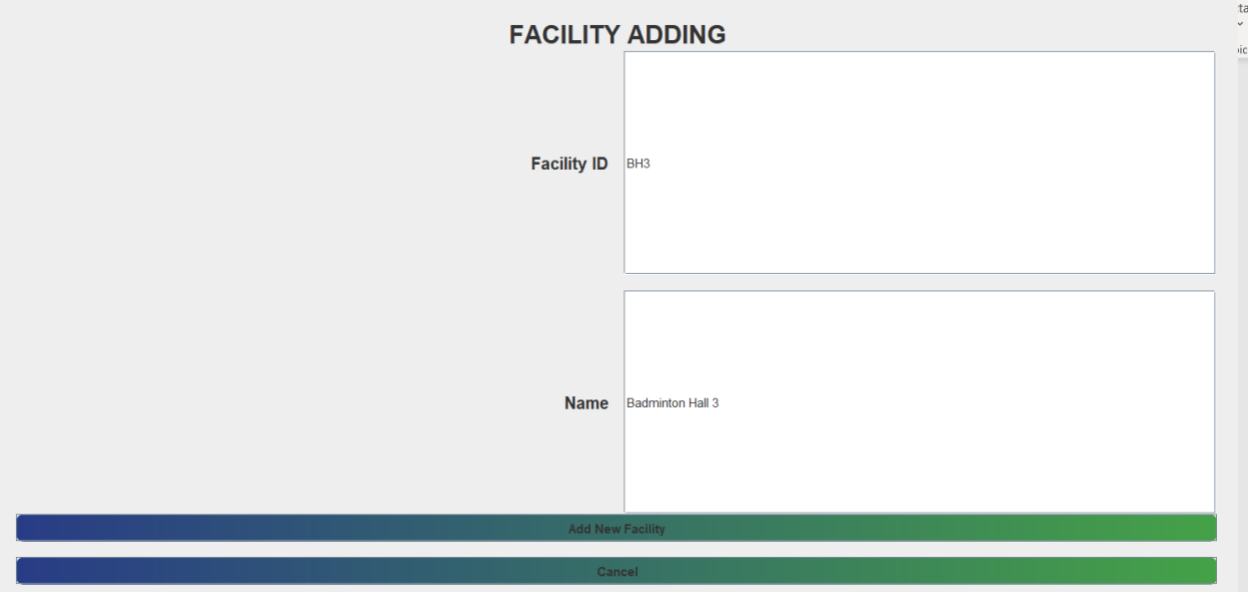
8.2.7 Facility management



Users who navigate the facility management section in the menu will be taken to this interface to access and view the list of facility records. The interface also allows users to search for specific

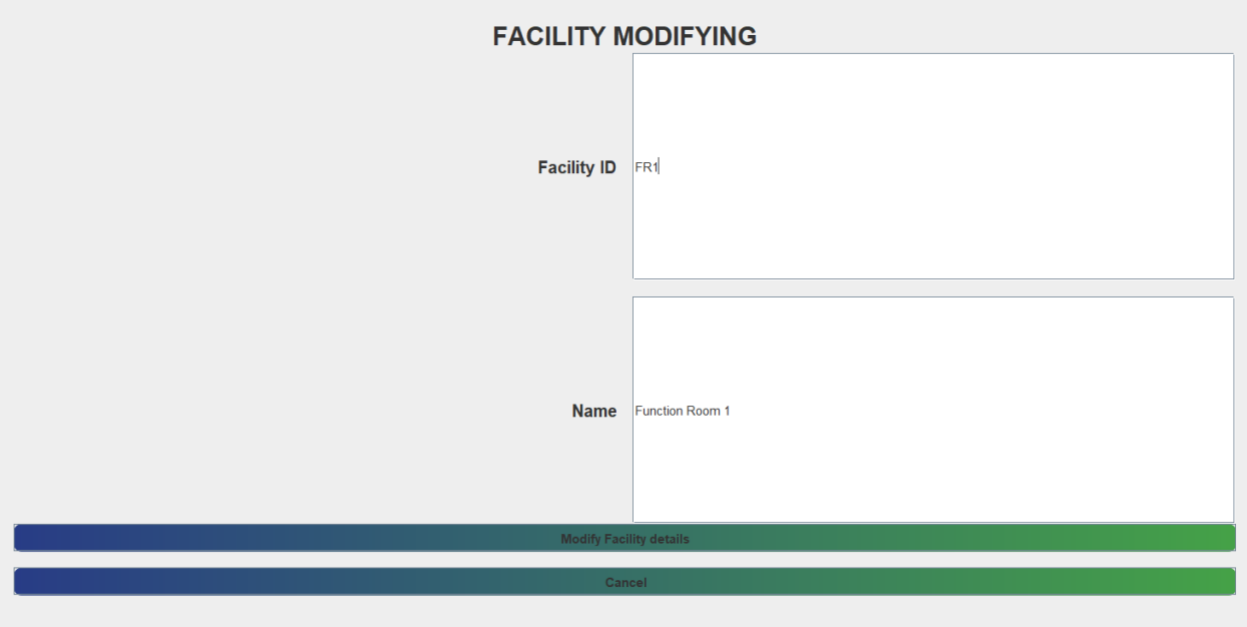
facilities, add a new facility request, modify facility details, delete facility records, view facility details, and approve resolved facility requests on the record they selected from the table.

8.2.7.1 Add New Facility



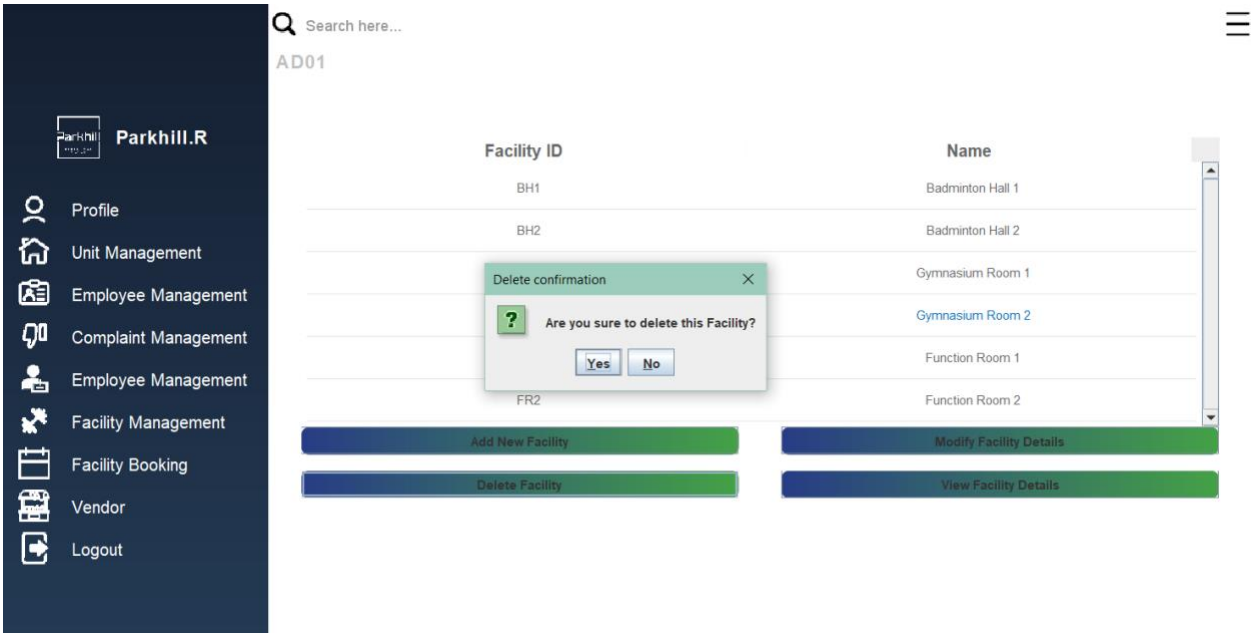
When users click the "add new facility request" button, they will be taken to a form where they can input the necessary details to add a new facility request. Once they have filled in the form and clicked the "add new facility request" button, a confirmation message will notify the users that the facility request has been added successfully.

8.2.7.2 Modify Facility Details



To modify a facility request, users must click on the facility request they wish to modify and then proceed by clicking the "modify facility request" button. The users will then be directed to an interface to modify facility details. Afterward, users can click the "modify facility request" button to submit their modified facility details.

8.2.7.3 Delete Facility

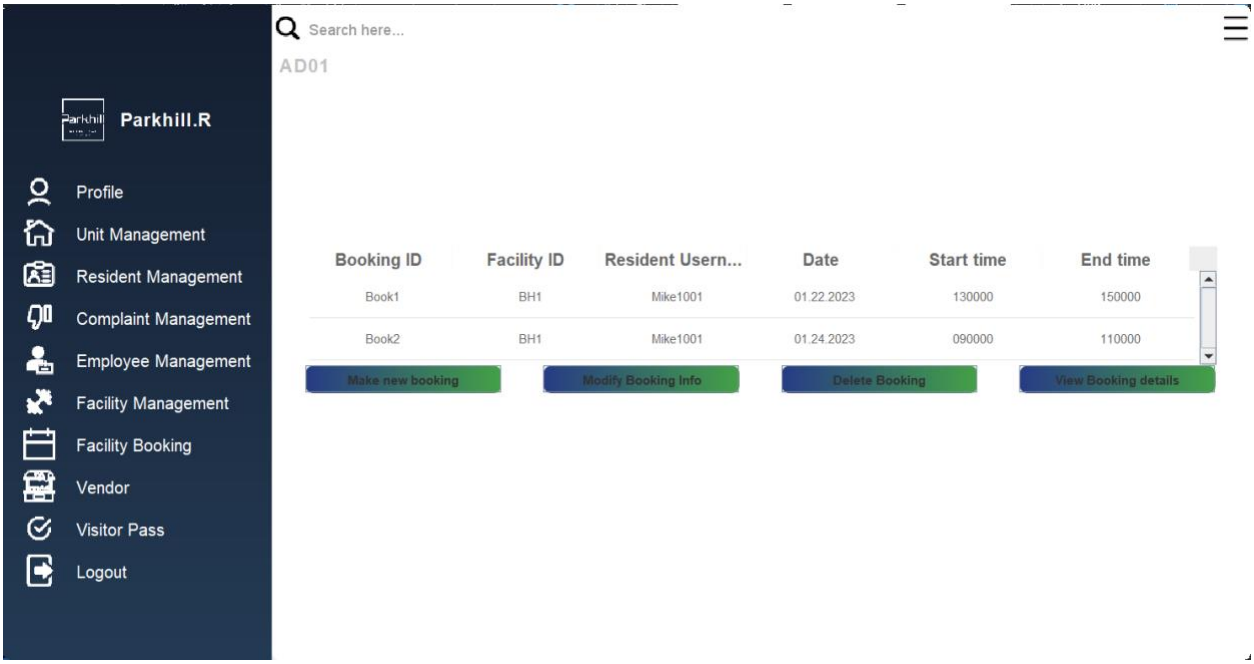


Users must select the facility request they want to delete and click the "delete facility request" button to delete a facility request. This action will lead them to a new interface where they can confirm the deletion of their facility request details. Users will receive a notification to verify their decision before the facility request is officially deleted.

8.2.7.4 View Facility Details

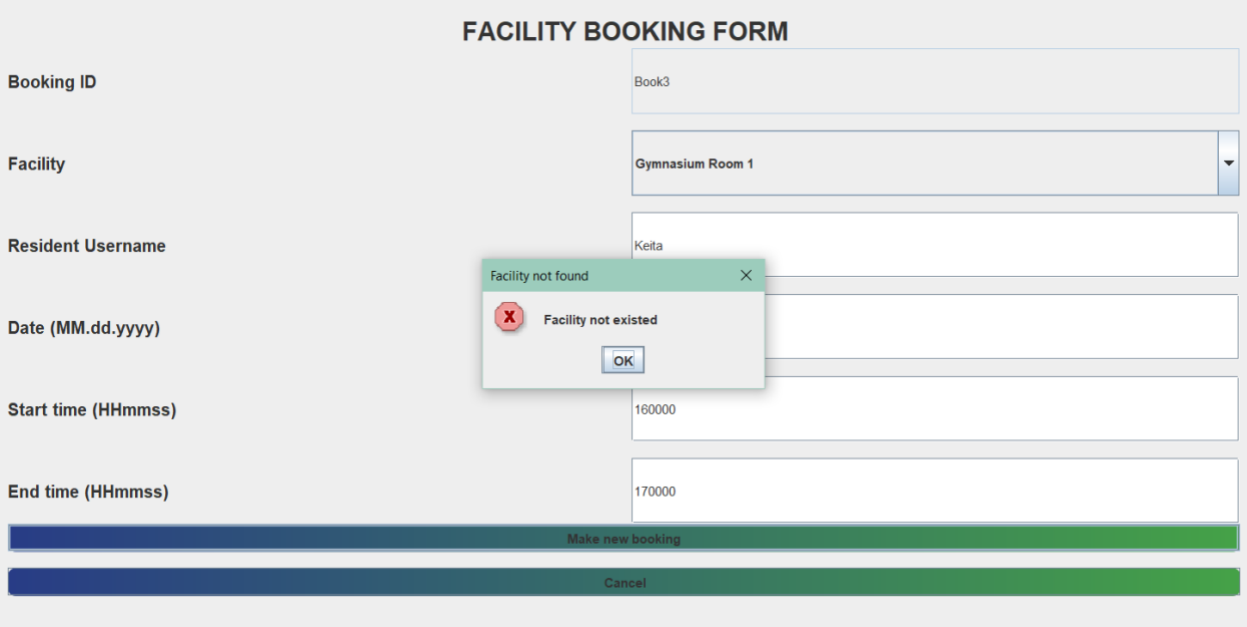
To view a facility request, users must choose the one they want to view and click the "view facility request" button. This will direct them to a new interface displaying facility request details.

8.2.8 Facility booking



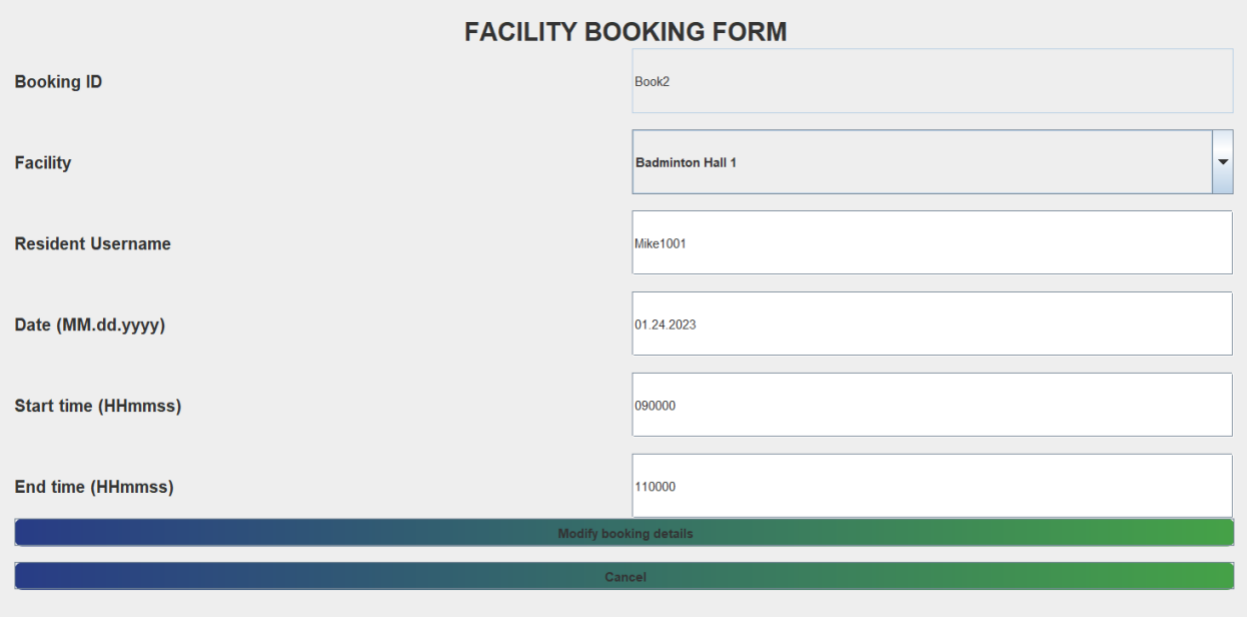
Users who navigate the facility booking management section in the menu will be taken to this interface to access and view the list of facility booking records. The interface also allows users to search for specific bookings, make new bookings, modify booking details, delete bookings, view booking details, and approve booking completion on the record they selected from the table.

8.2.8.1 Make New Booking



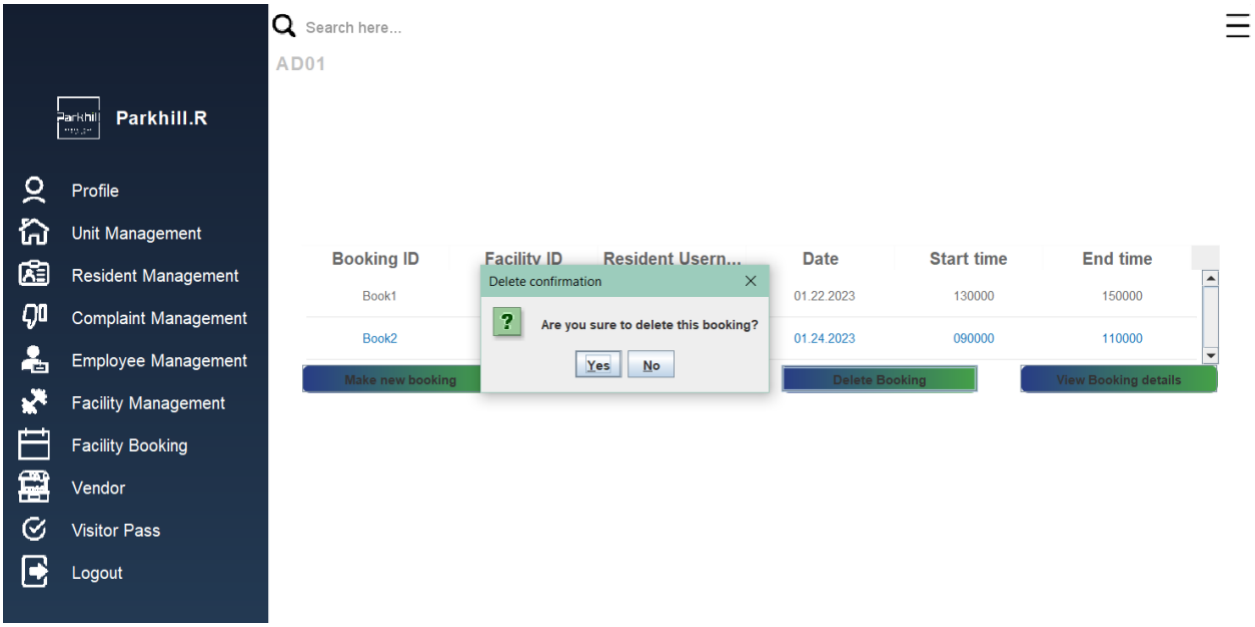
When users click the "make new booking" button, they will be taken to a form where they can input the necessary details to add a new booking. Once they have filled in the form and clicked the "make new booking" button, a confirmation message will notify the users if the booking has been added successfully. Otherwise, the system will also pop out a notification stating that the facility is unavailable.

8.2.8.2 Modify Booking Info



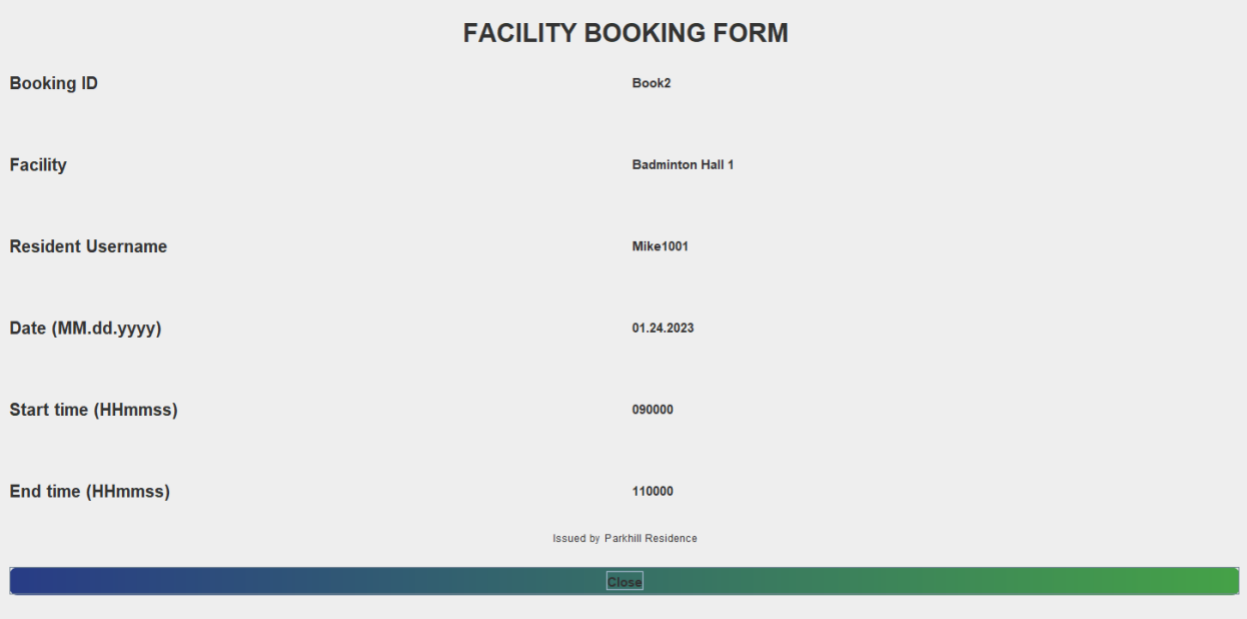
To modify a booking, users must click on the booking they wish to modify and then proceed by clicking the "modify booking" button. The users will then be directed to an interface to modify booking detail. Afterward, users can click the "modify booking" button to submit their modified booking details.

8.2.8.3 Delete Booking



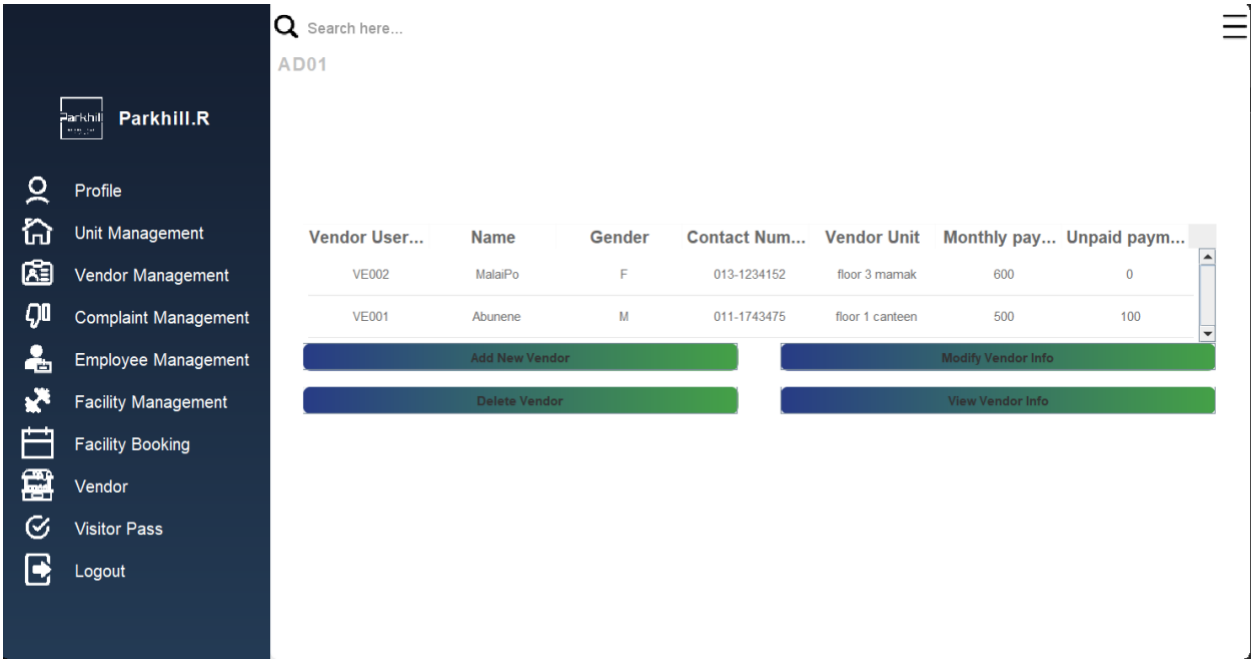
Users must select the booking they want to delete and click the "delete booking" button to delete a booking. This action will lead them to a new interface where they can confirm the deletion of their booking details. Users will receive a notification to verify their decision before the booking is officially deleted.

8.2.8.4 View Booking Details



To view a booking, users must choose the one they want to view and click the "view booking" button. This will direct them to a new interface that displays the booking details.

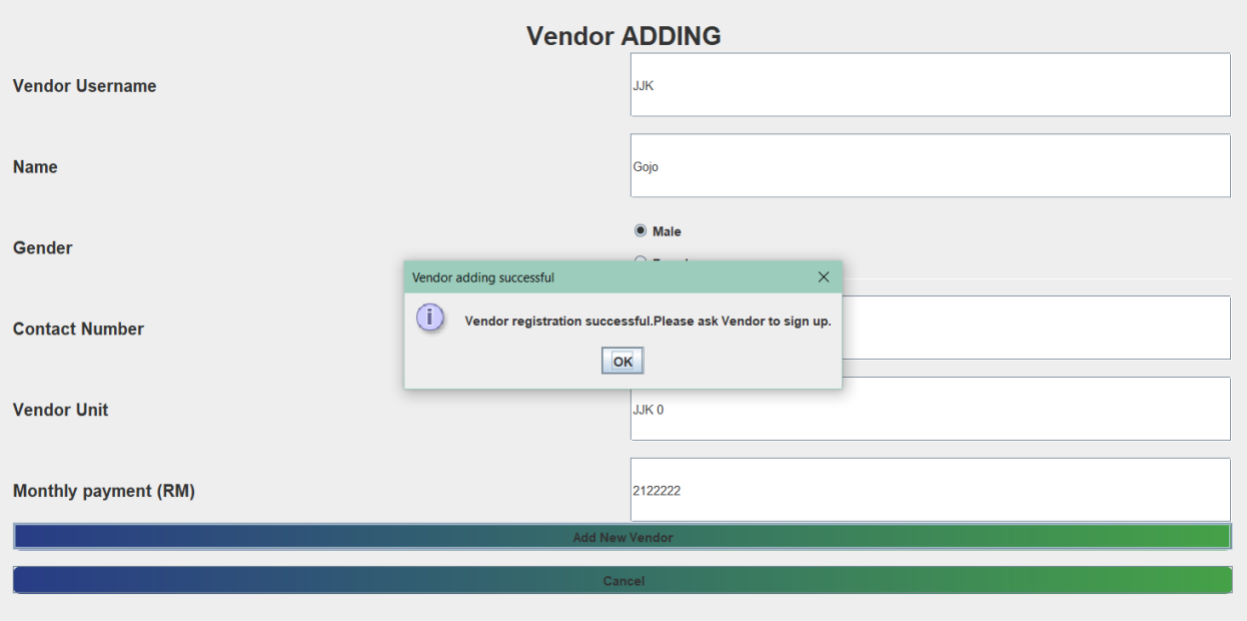
8.2.9 Vendor



Users who navigate the Vendor management section in the menu will be taken to this interface to access and view the list of vendor records. The interface also allows users to search for specific

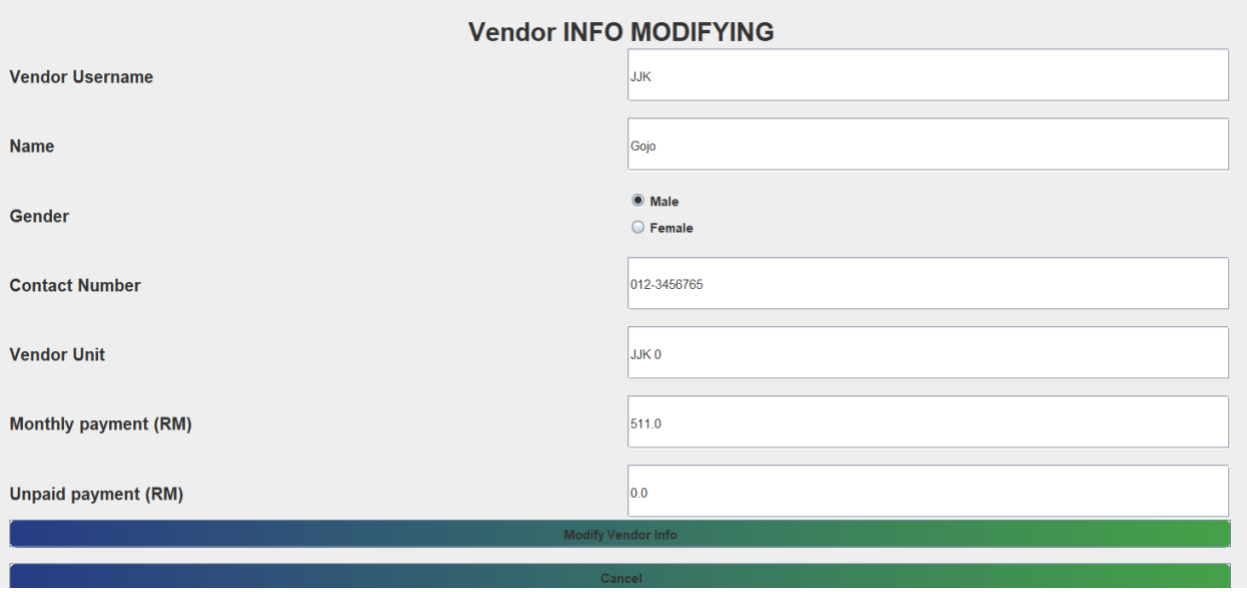
vendors, add a new vendor, modify vendor details, delete vendors, view vendor details, and approve vendor registration.

8.2.9.1 Add New Vendor



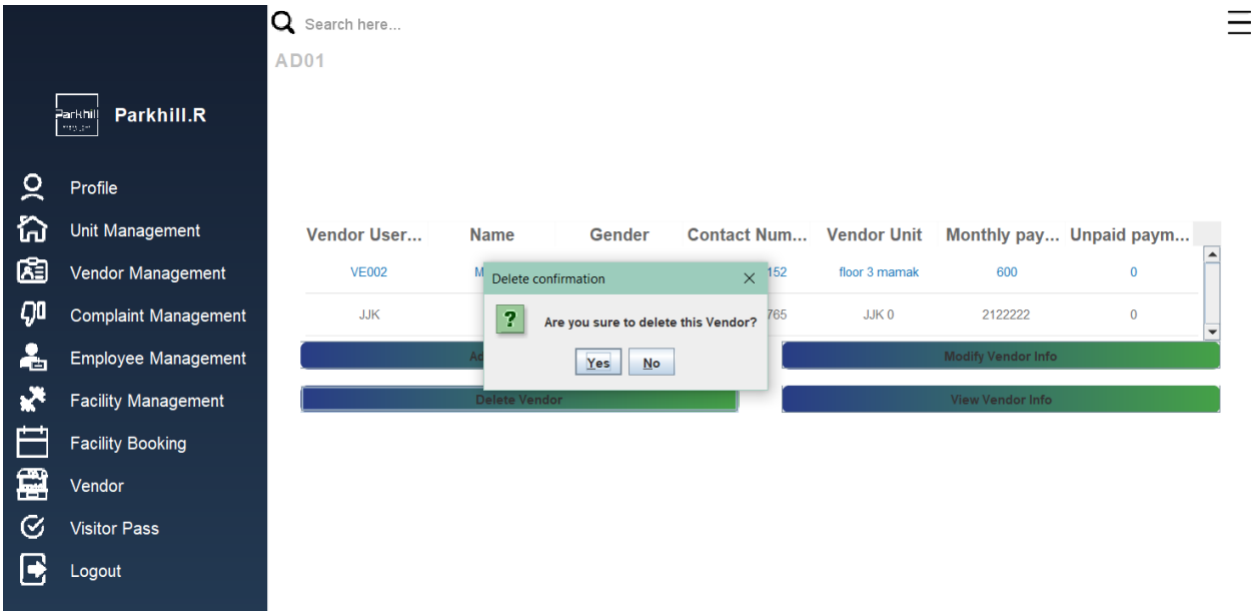
When users click the "add new vendor" button, they will be taken to a form where they can input the necessary details to add a new vendor. Once they have filled in the form and clicked the "add new vendor" button, a confirmation message will notify the users that the vendor has been added successfully.

8.2.9.2 Modify Vendor Info



To modify a vendor, users must click on the vendor they wish to modify and then proceed by clicking the "modify vendor" button. The users will then be directed to an interface to modify vendor details. Afterward, users can click the "modify vendor" button to submit their modified vendor details.

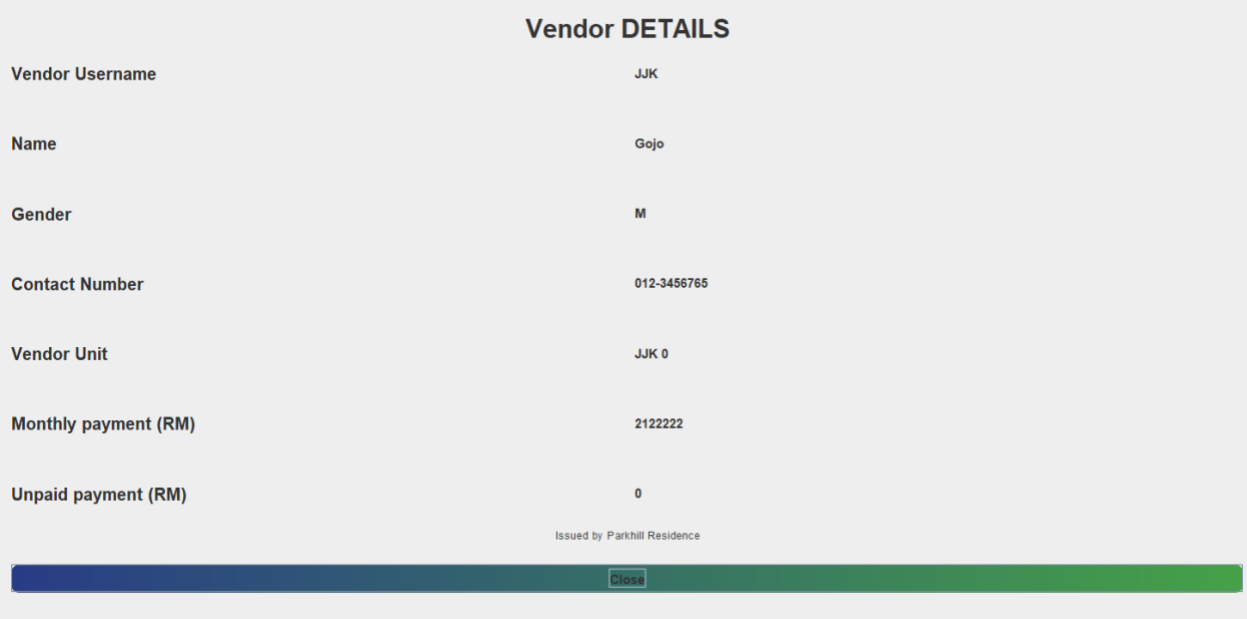
8.2.9.3 Delete Vendor



Users must select the vendor they want to delete and click the "delete vendor" button to delete a vendor. This action will lead them to a new interface where they can confirm the deletion of their

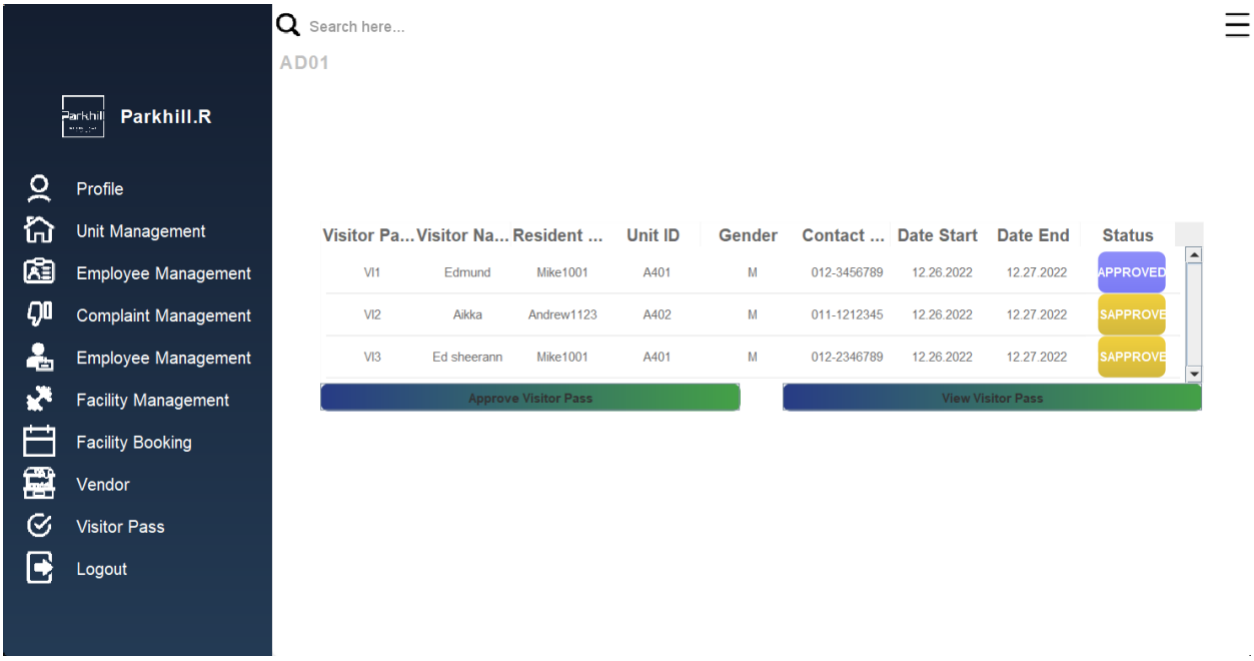
vendor details. Users will receive a notification to verify their decision before the vendor is officially deleted.

8.2.9.4 View Info



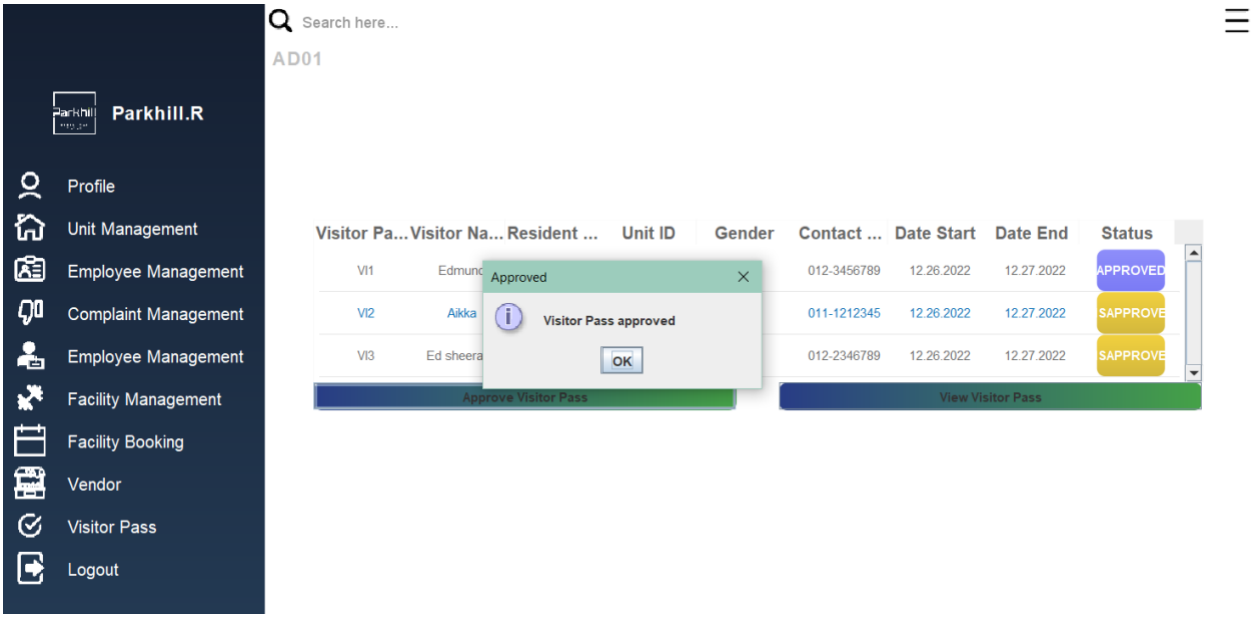
To view a vendor, users must choose the one they want to view and click the "view vendor" button. This will direct them to a new interface that displays the vendor details.

8.2.10 Visitor pass



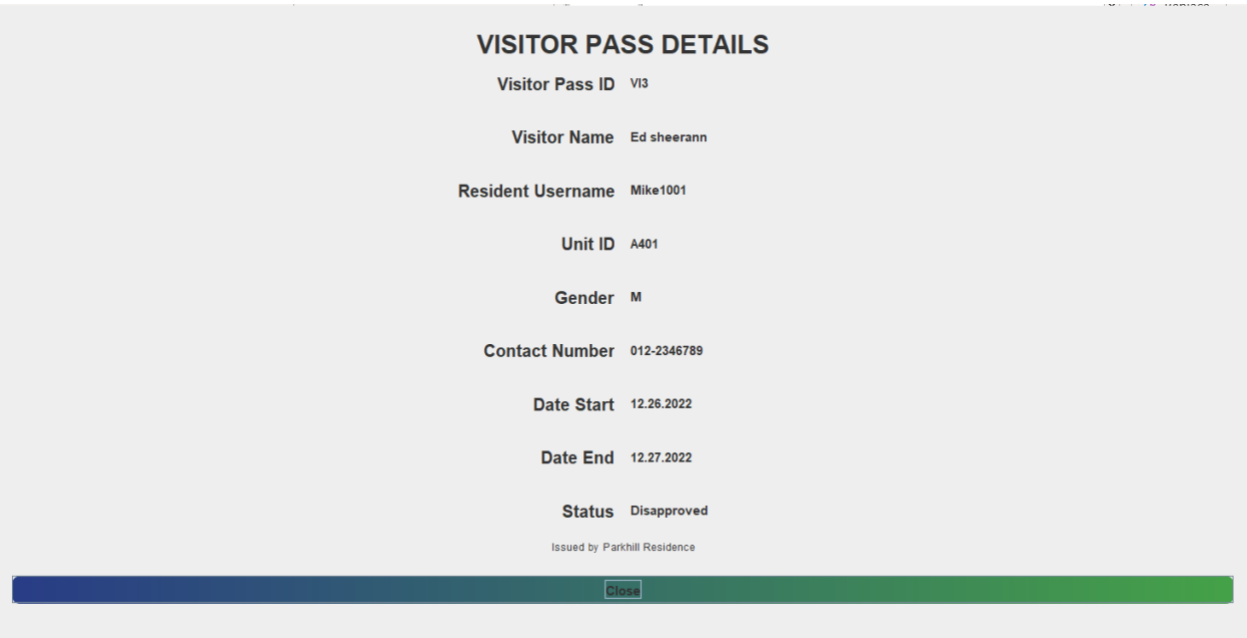
Users who navigate the visitor pass management section in the menu will be taken to this interface to access and view the list of visitor pass records. The interface also allows users to search for a specific visitor pass, approve visitor pass and view visitor pass details.

8.2.10.1 Approve Visitor Pass



In order to change the status of a visitor pass to approve, the user must first pick which visitor pass they want to approve. Then, by clicking the "approve visitor pass" button, the system will change the visitor pass to approved. Users will receive a notification to verify their decision before the visitor pass status officially changes.

8.2.10.2 View Visitor Pass



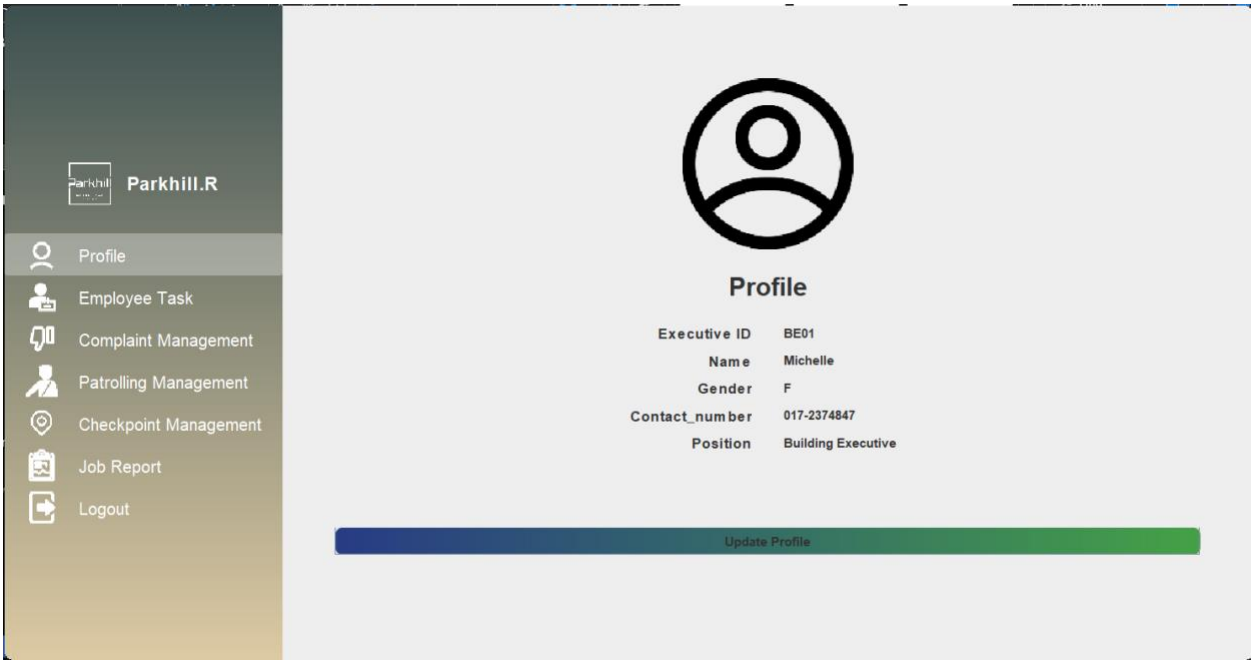
To view a visitor pass, users must choose the one they want to view and click the "view visitor pass " button. This will direct them to a new interface displaying visitor pass details.

8.3 Building Executive

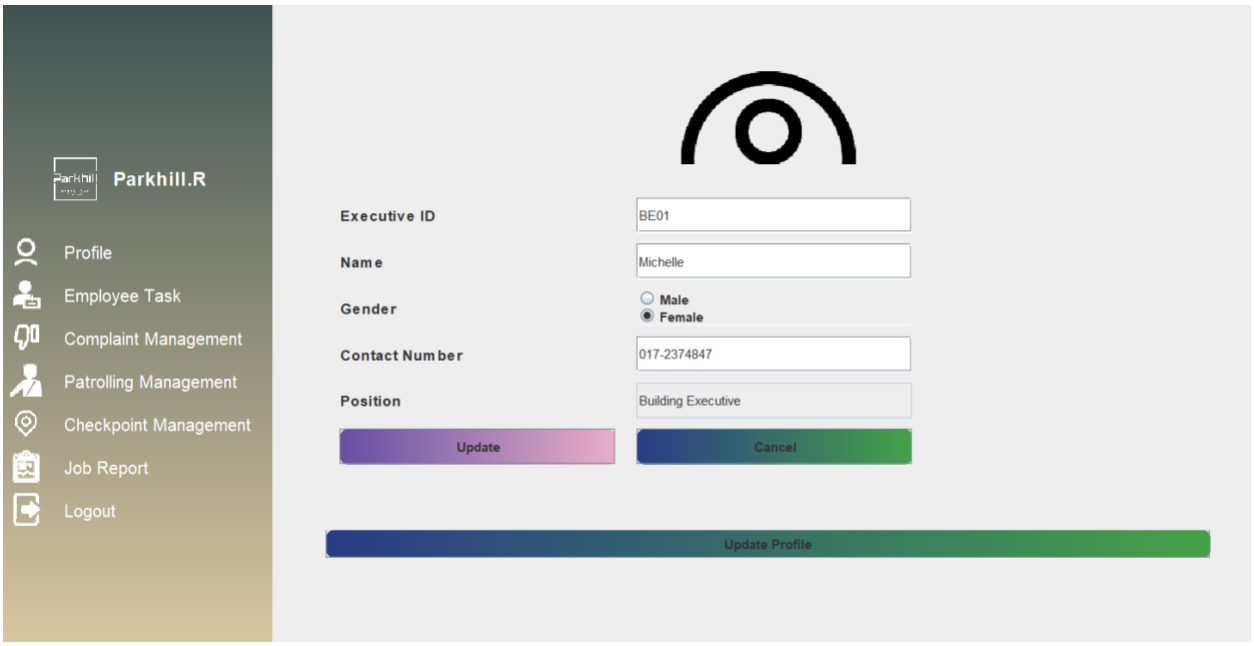
8.3.1 Functional requirement

1. Profile Interface
2. Employee Task Interface
3. Complaint Management
4. Patrolling Management
5. Checkpoint Management
6. Job Report Interface
7. Logout

8.3.2 Profile

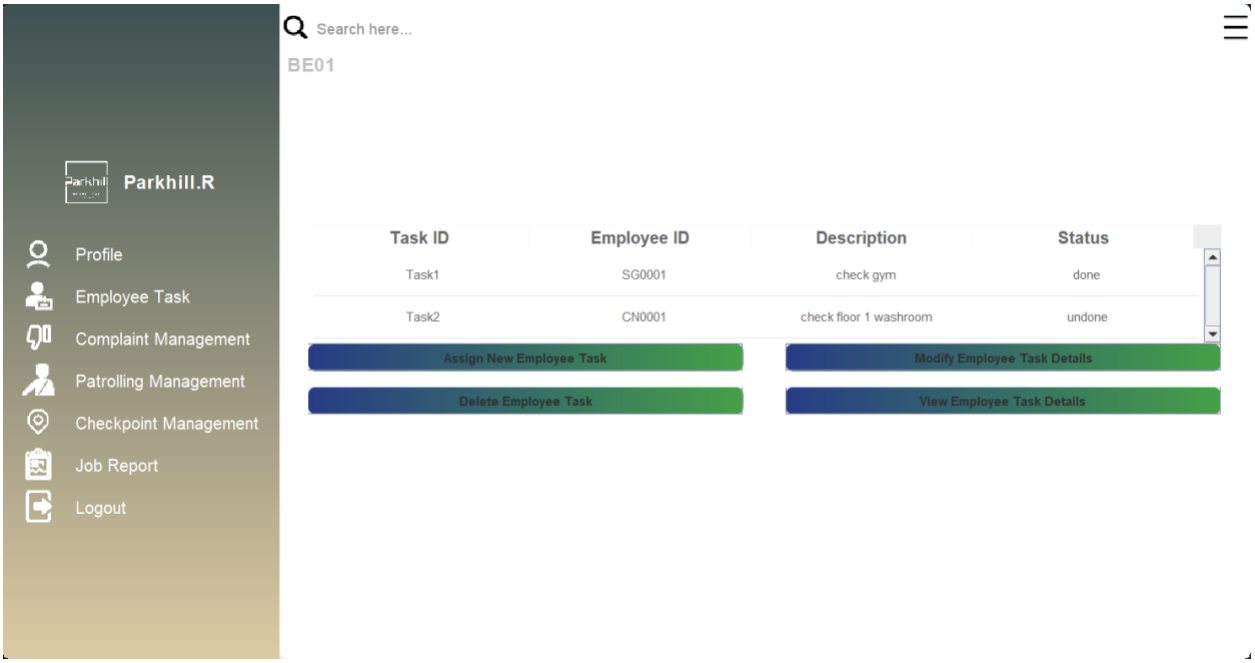


Once the building executive has successfully logged into the system, they will be directed to their designated interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the building executive profile page is accessible through this interface and allows users to modify their profile information.



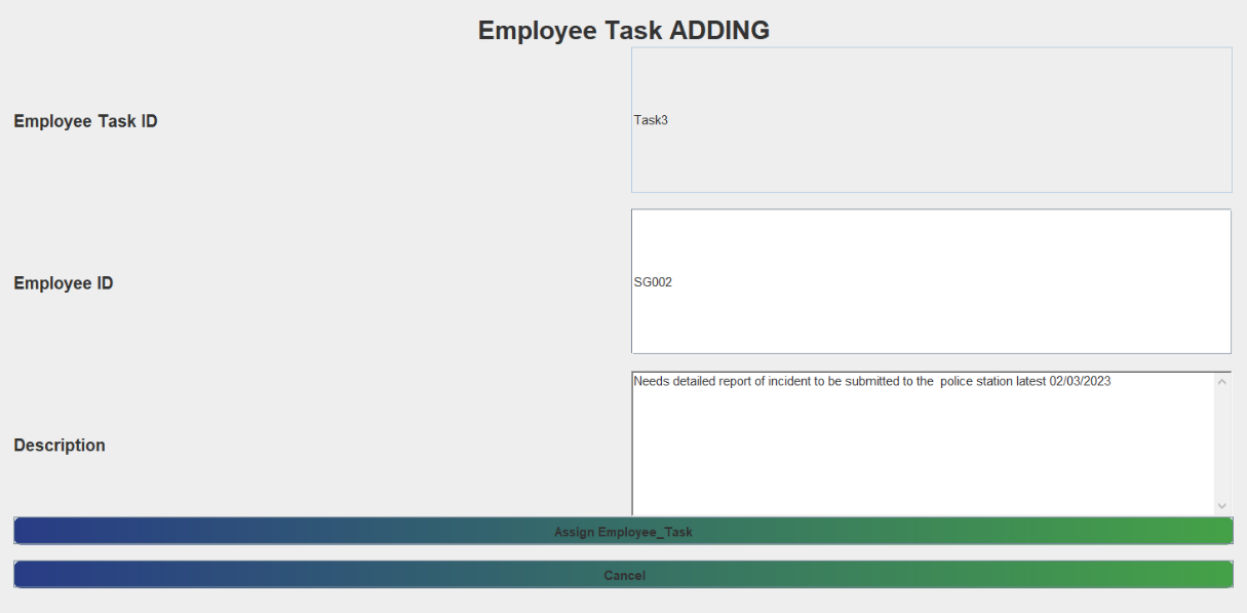
By clicking on the update profile button, the user will be led to this interface which allows them to change their username, name, gender and contact number. After clicking the "Update" button, a confirmation notice will alert users that the information is updated.

8.3.3 Employee task



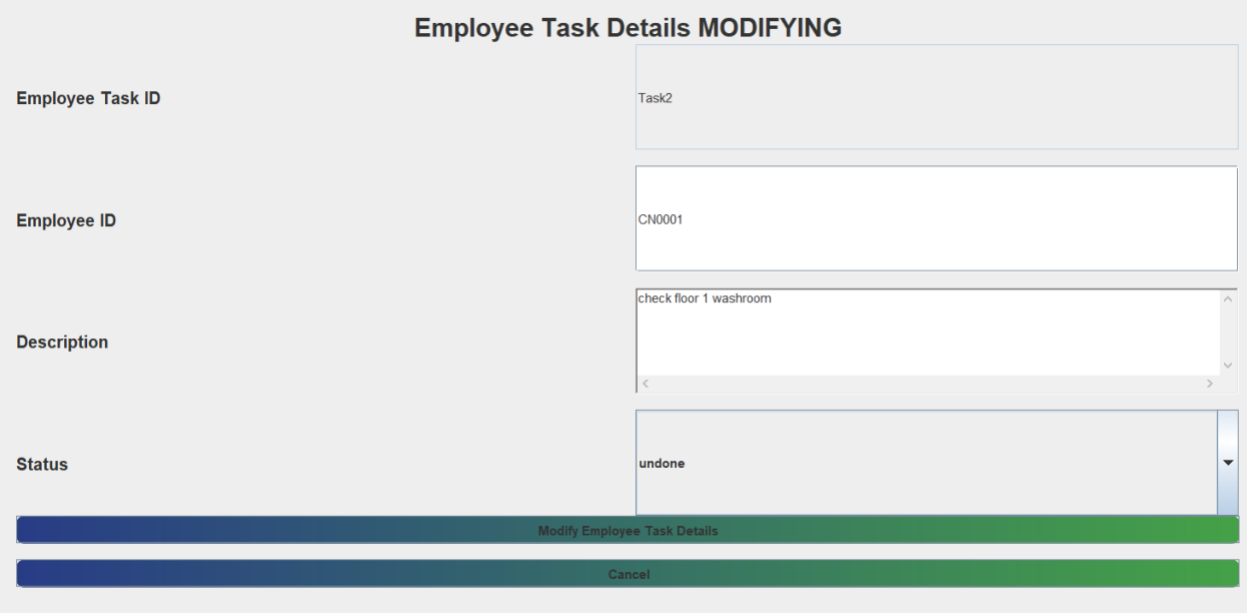
Users who navigate the employee task management section in the menu will be directed to this interface to access and view the list of employee tasks. The interface also allows users to search for specific employee tasks, add a new employee task, modify task details, delete employee tasks, view task details, and approve completed tasks on the record they selected from the table.

8.3.3.1 Assign New Employee Task



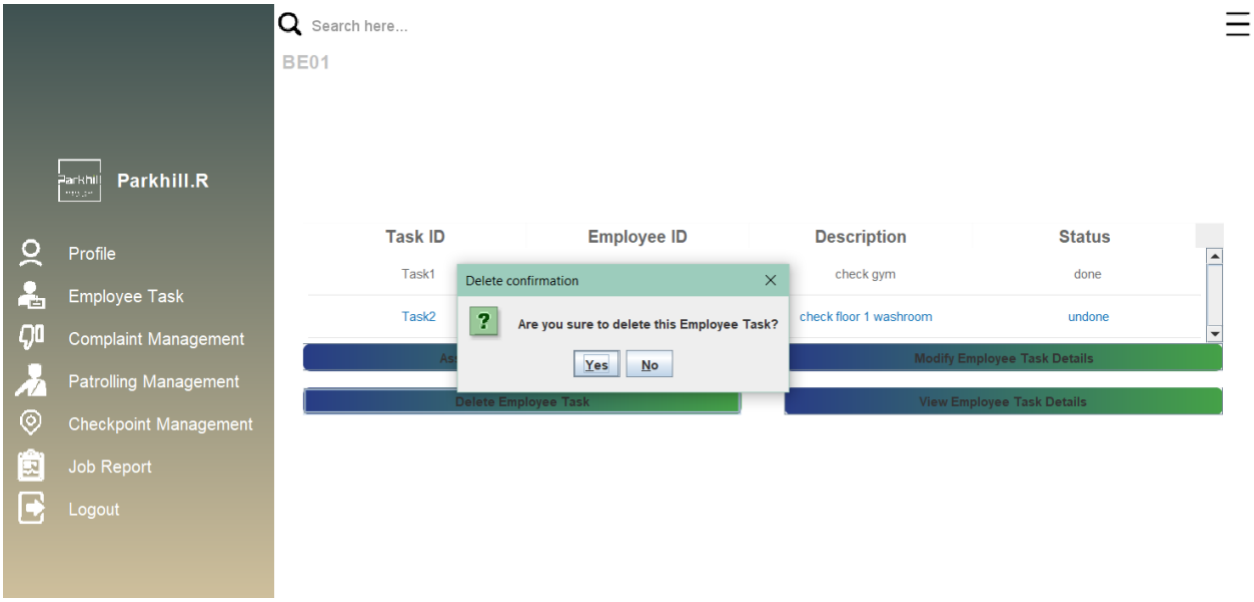
When users click the "assign new employee task" button, they will be directed to a form where they can input the necessary details to assign a new employee task. Once they have filled in the form and clicked the "assign new employee task" button, a confirmation message will notify the users that the task has been added successfully.

8.3.3.2 Modify Employee Task Details



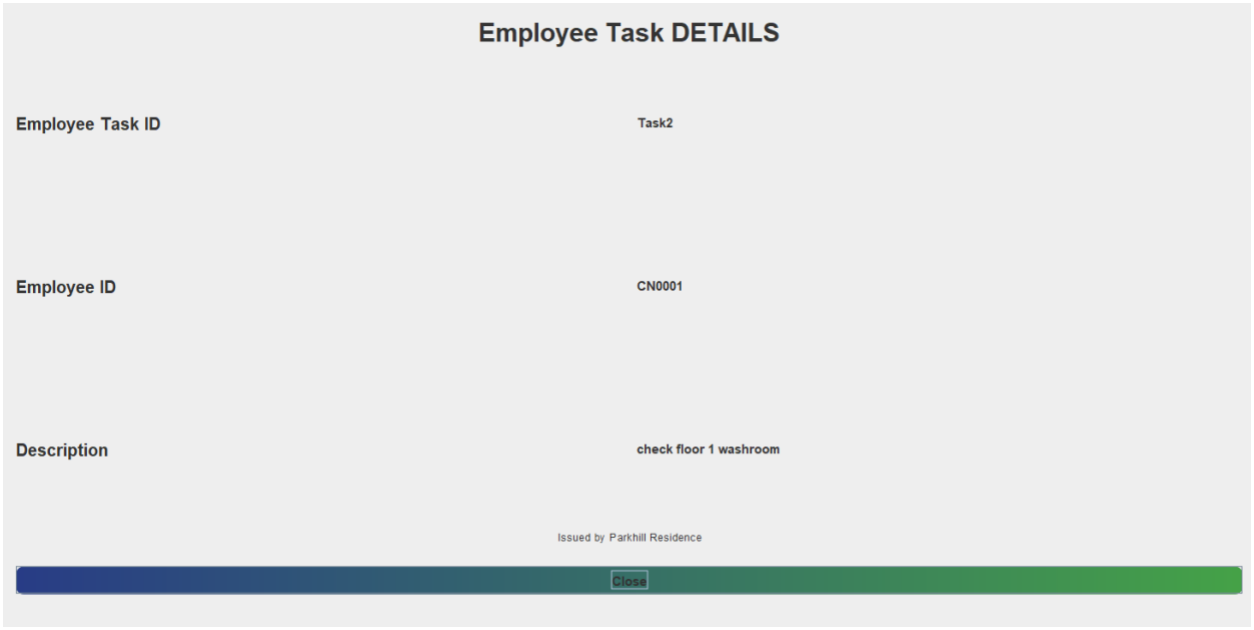
To modify a task, users must click on the task they wish to modify and then proceed by clicking the "modify employee task" button. The users will then be directed to an interface to modify task details. Afterward, users can click the "modify employee task" button to submit their modified task details.

8.3.3.3 Delete Employee Task



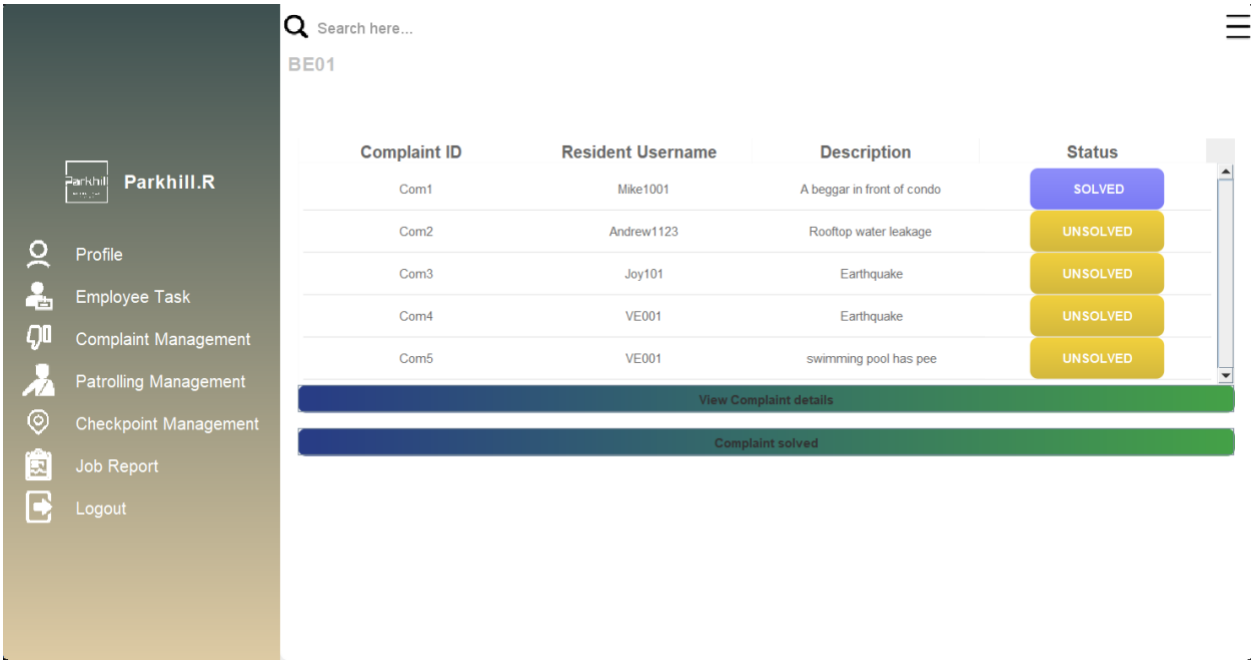
Users must select the task they want to delete and click the "delete task" button to delete a task. This action will lead them to a new interface where they can confirm the deletion of their task details. Users will receive a notification to verify their decision before the task is officially deleted.

8.3.3.4 View Employee Task Details



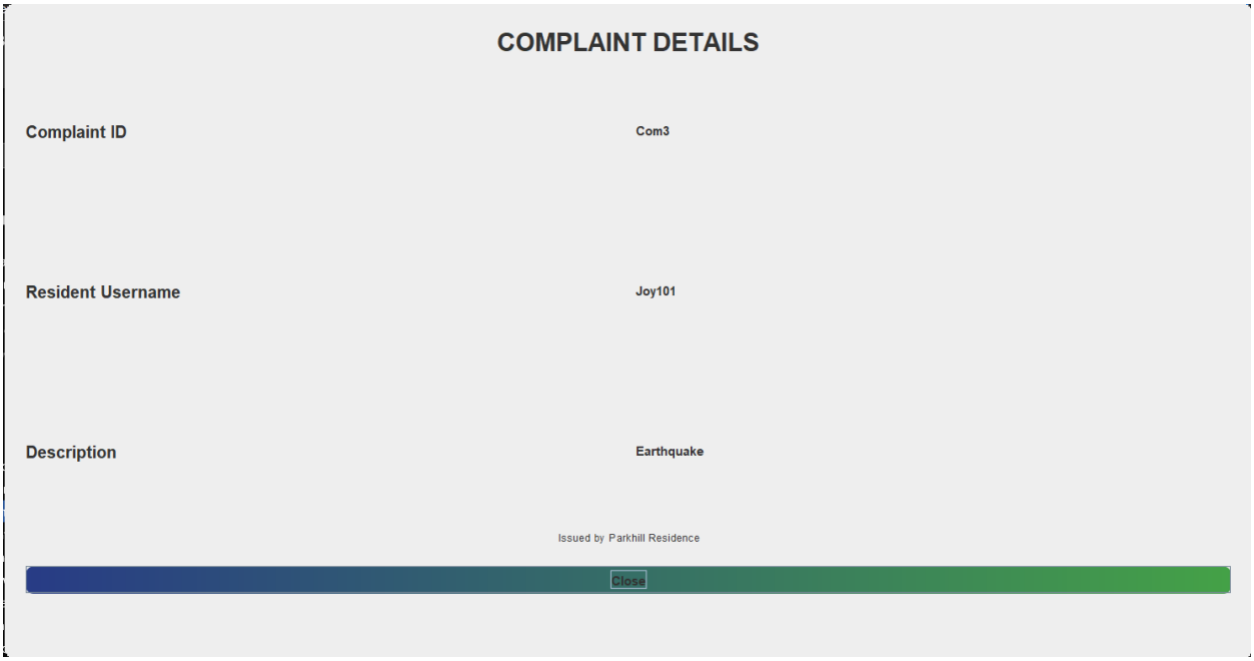
To view a task, users must choose the one they want to view and click the "view task" button. This will direct them to a new interface that displays the task details.

8.3.4 Complaint management



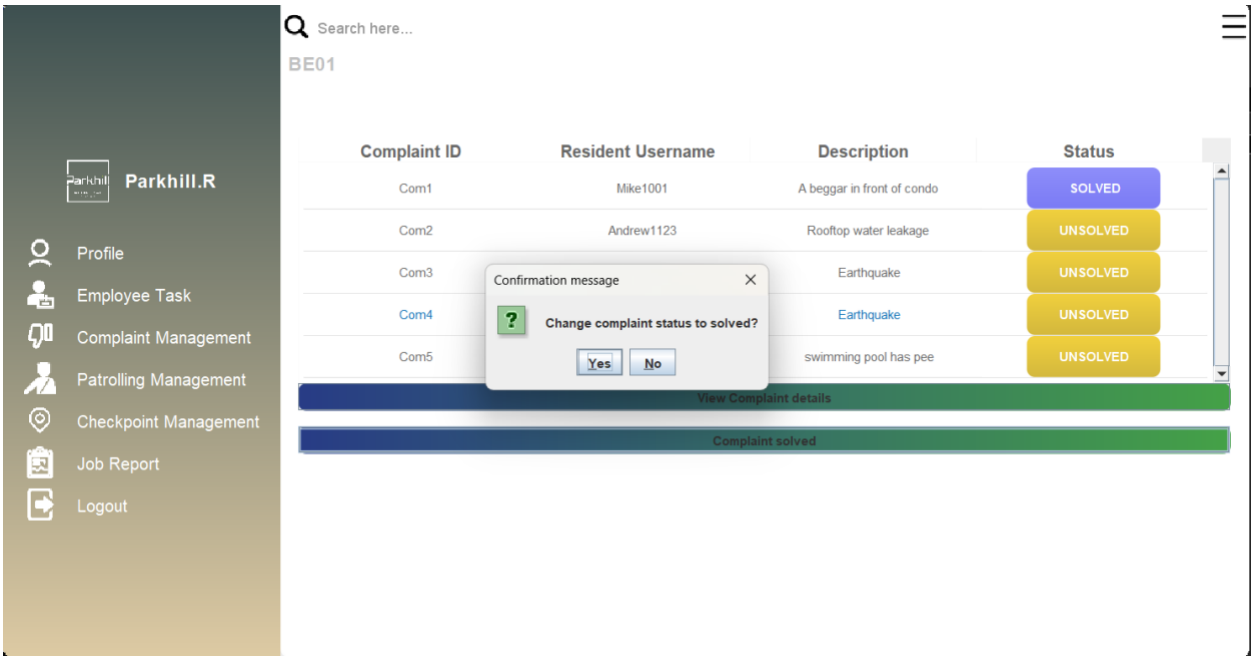
This is the complaint management section for building executives. Here, they can access all complaints related to residents and vendors, including the complaint ID and resident/vendor name. They can perform various operations such as viewing complaint details and marking a complaint as solved.

8.3.4.1 View Complaint details



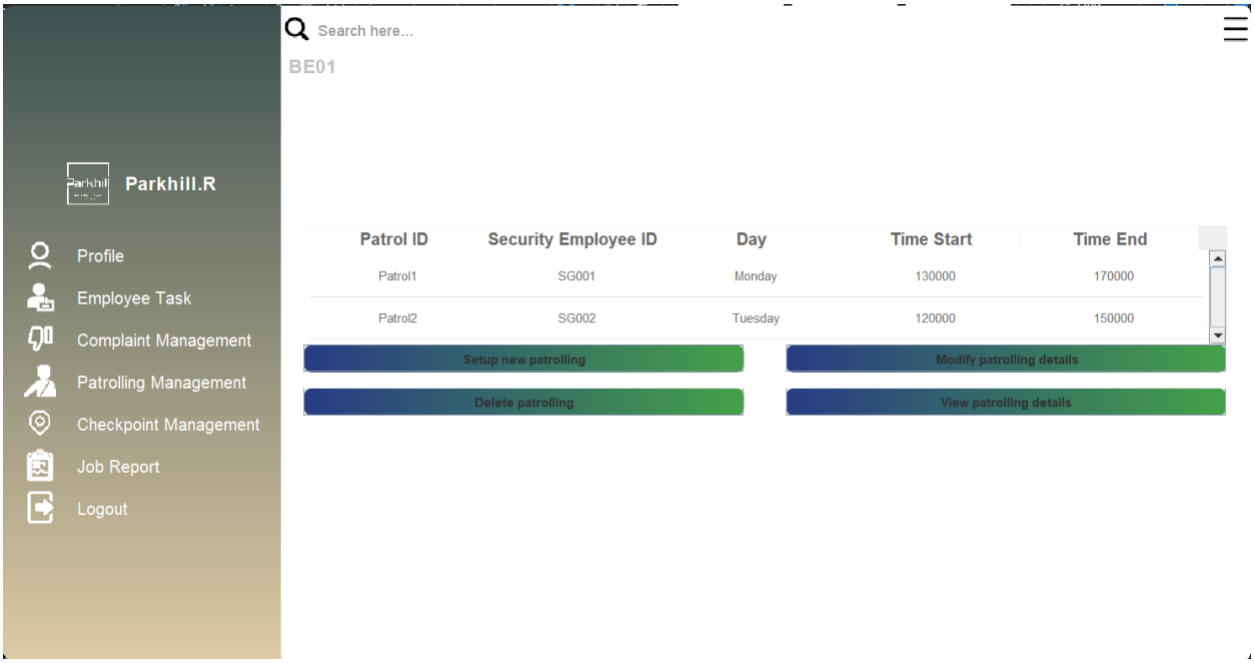
After selecting a specific complaint, clicking on "View Complaint Details" will allow the building executive to view detailed information about the complaint.

8.3.4.2 Complaint solved



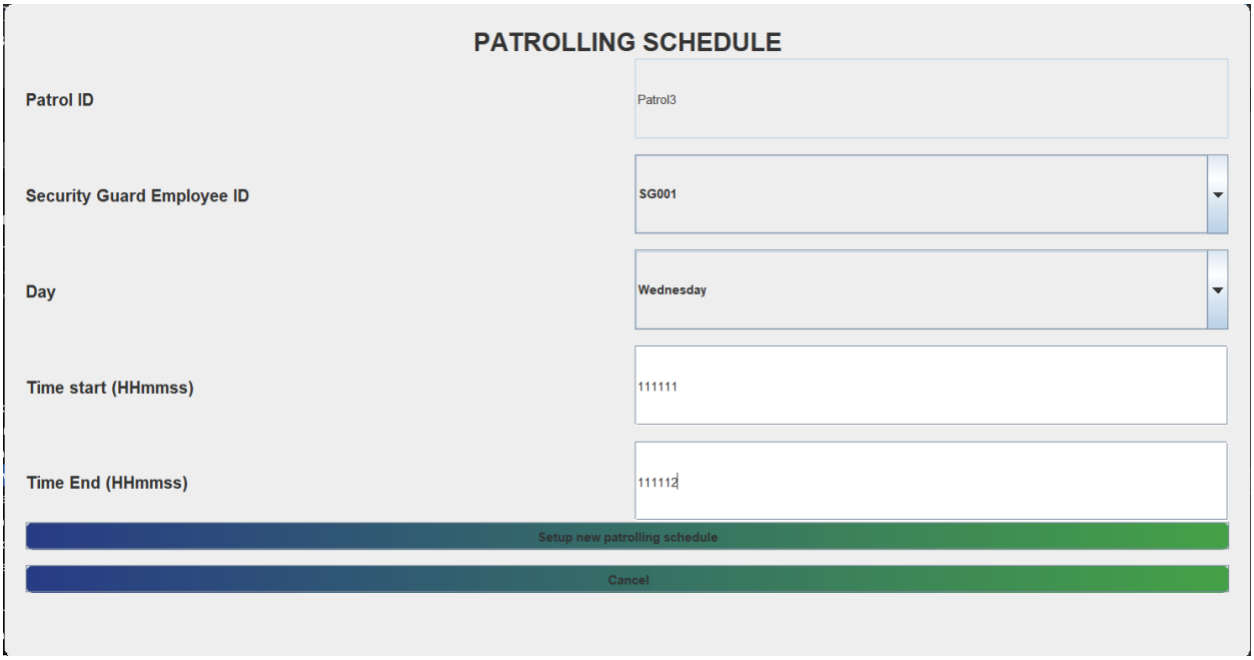
Users can change the status of a complaint from "Unsolved" to "SOLVED" by clicking on the "Complaint solved" button and confirming the change.

8.3.5 Patrolling management

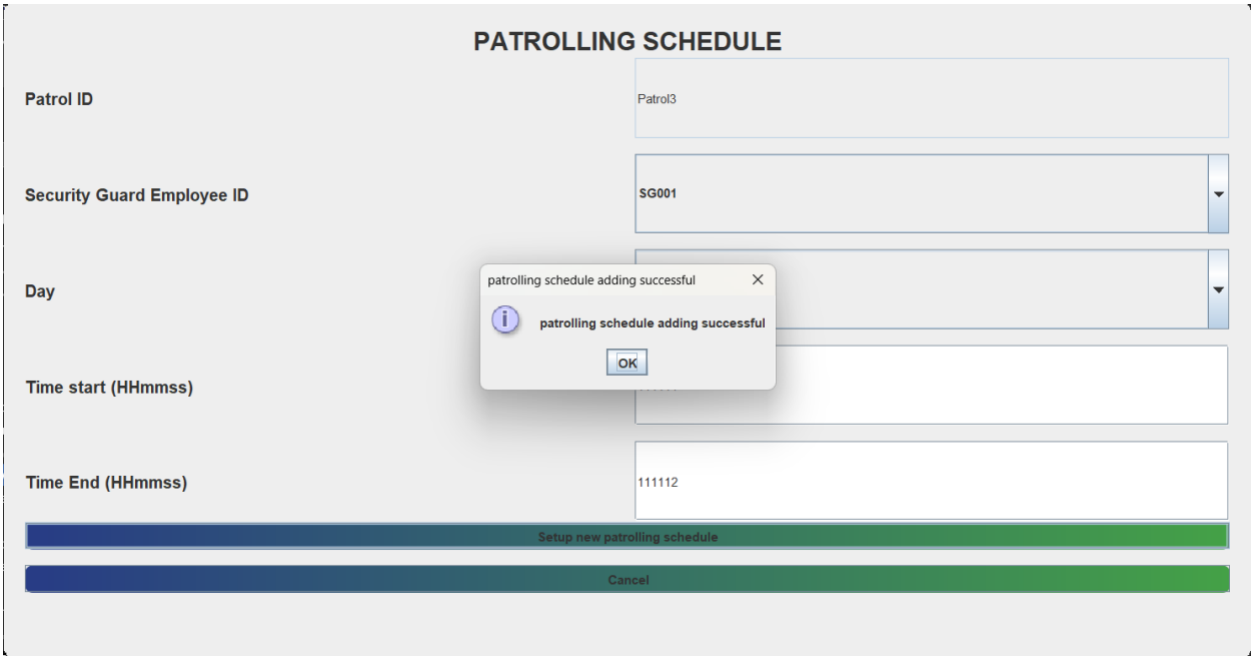


The interface displayed above is the "Patrolling Management" section of the Building Executive page. Here, Building Executives can easily access and review all the data related to the security guard patrolling, such as the time of patrol, security guard ID, and other relevant actions. This interface provides a comprehensive platform for Building Executives to efficiently manage and monitor the security operations of their buildings.

8.3.5.1 Setup new patrolling

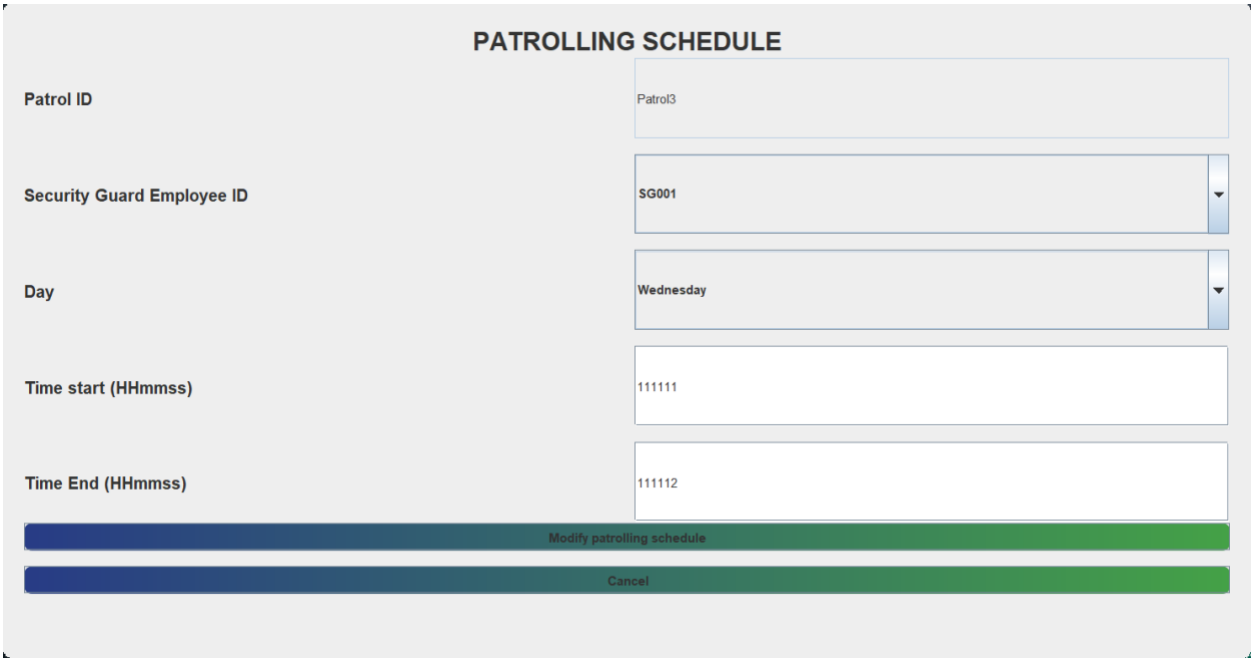


Building executives can add a new patrolling schedule by clicking on the "Setup new patrolling" button. The system will prompt the user to input relevant information such as the start and end time for the patrolling schedule.



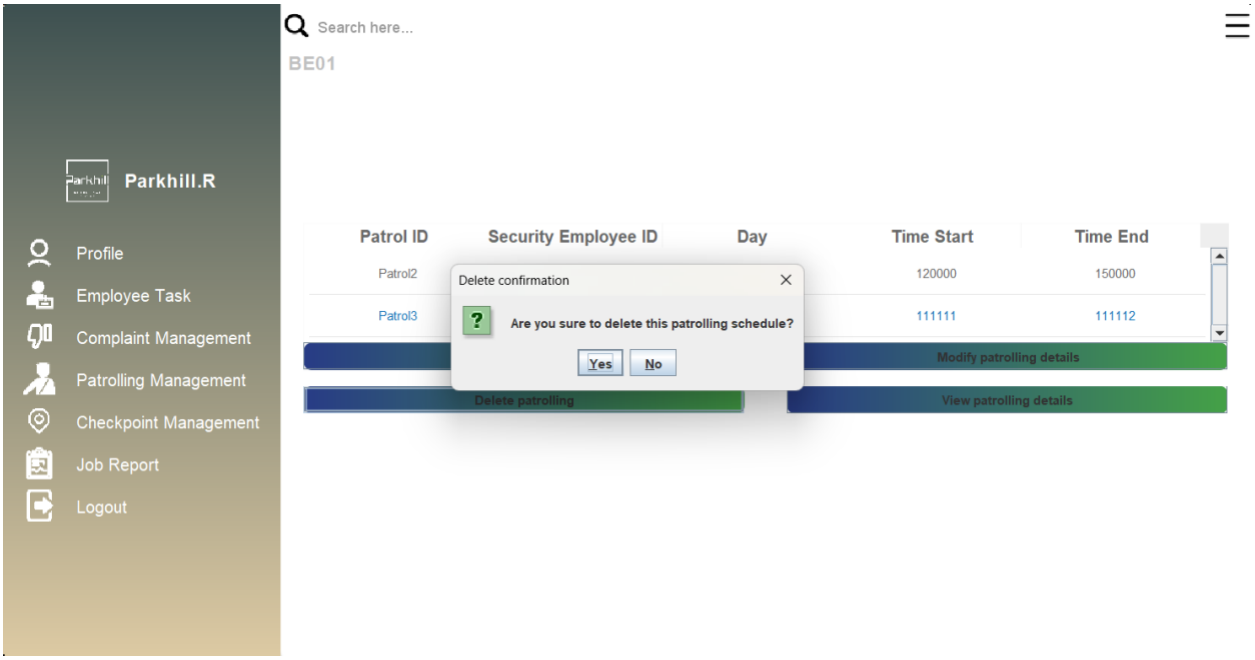
When a new schedule is successfully added to the system, the system will automatically notify the user.

8.3.5.2 Modify patrolling details



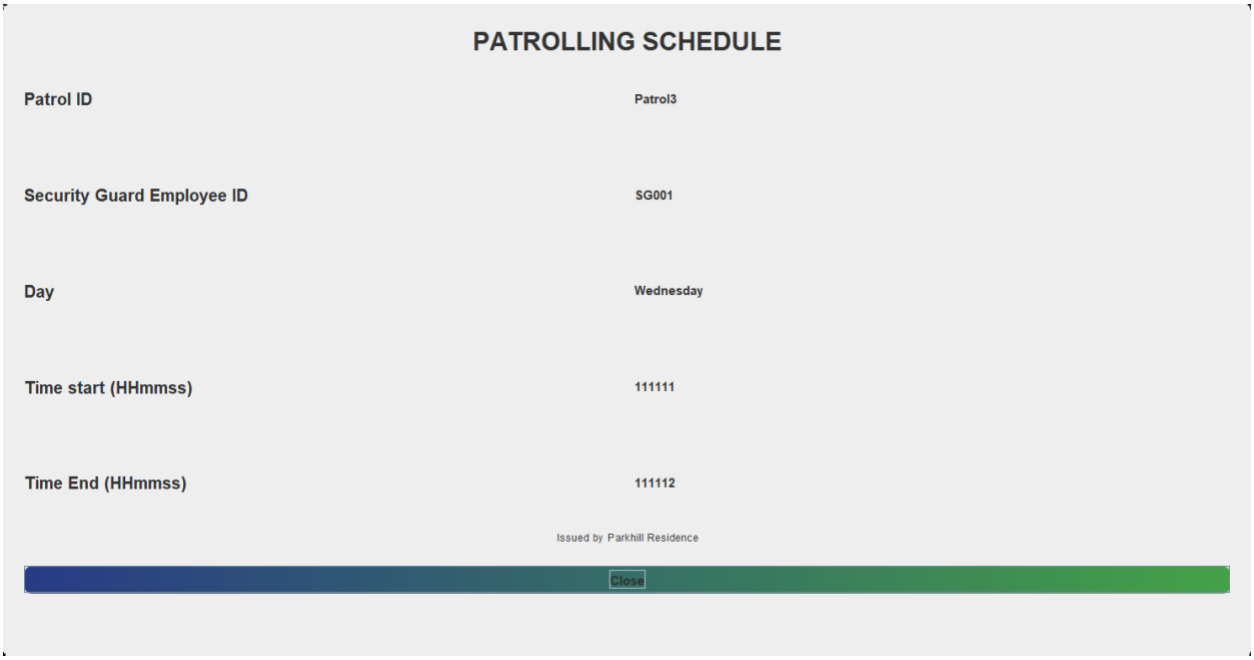
Users can modify specific patrolling schedules by clicking "Modify patrolling details". This will take them to an interface where they can edit the relevant information. Once the modifications are complete, the user can click "Modify patrolling details" to save the changes.

8.3.5.3 Delete patrolling.



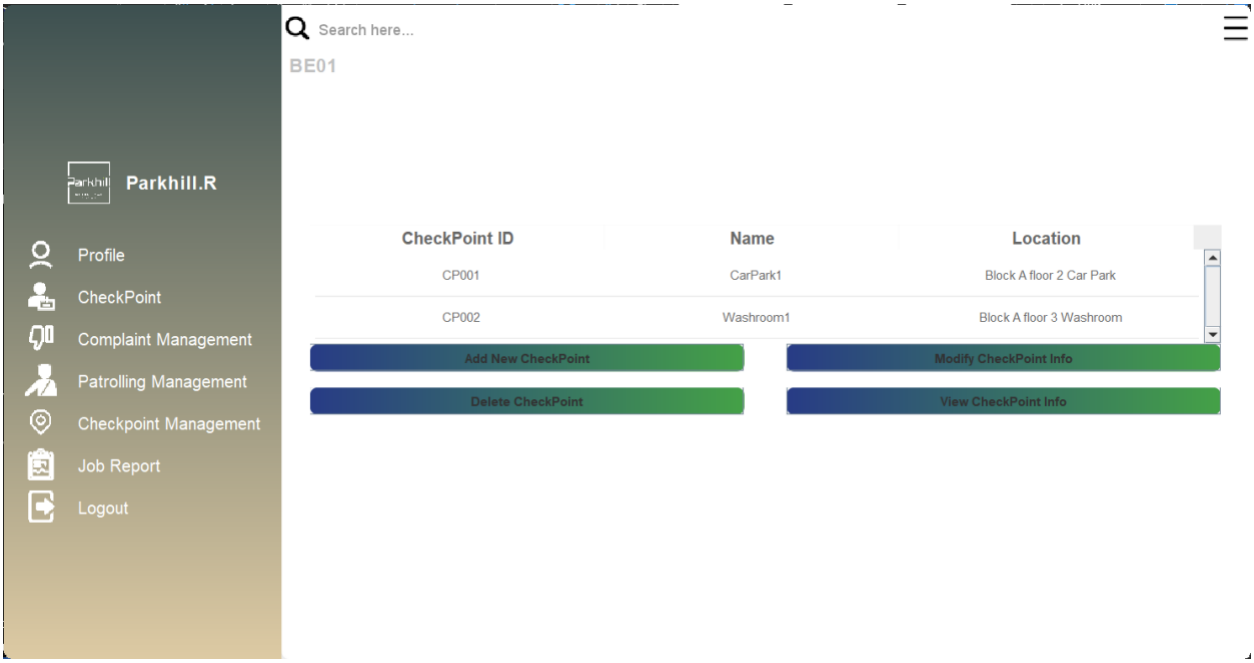
Users can delete a specific patrolling schedule by clicking on "Delete patrolling". After clicking on it, the system will prompt the user to confirm whether they want to delete the schedule. If the user clicks "Yes", the schedule will be deleted.

8.3.5.4 View Patrolling



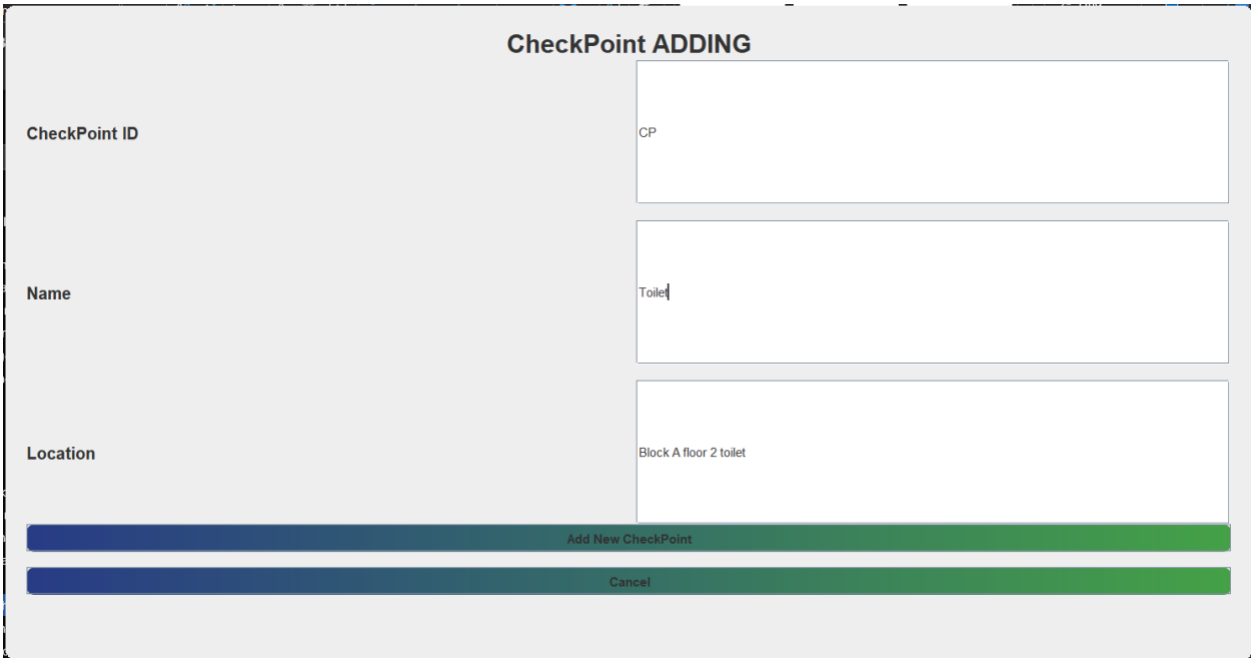
Users can select a specific patrolling schedule and click on "View Patrolling" to view more detailed information.

8.3.6 Checkpoint management

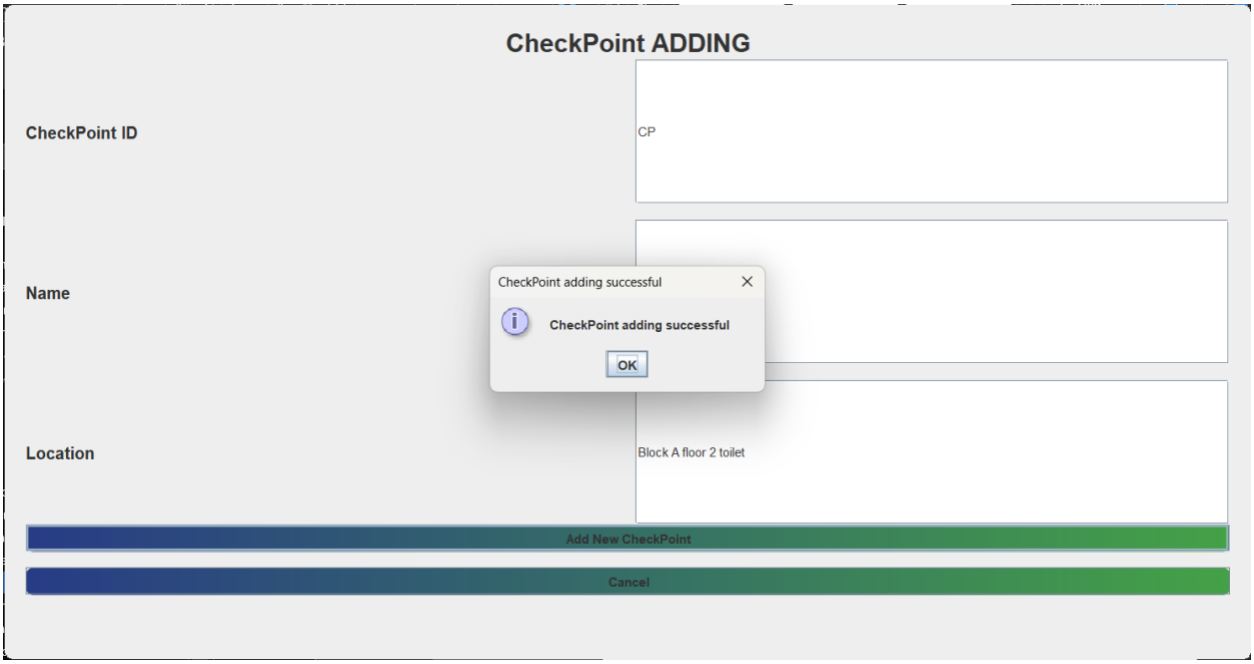


Building executives can navigate to the checkpoint management page by clicking on the corresponding button. On this page, they can view all information related to security check points, such as the checkpoint ID and its specific location.

8.3.6.1 Add New CheckPoint

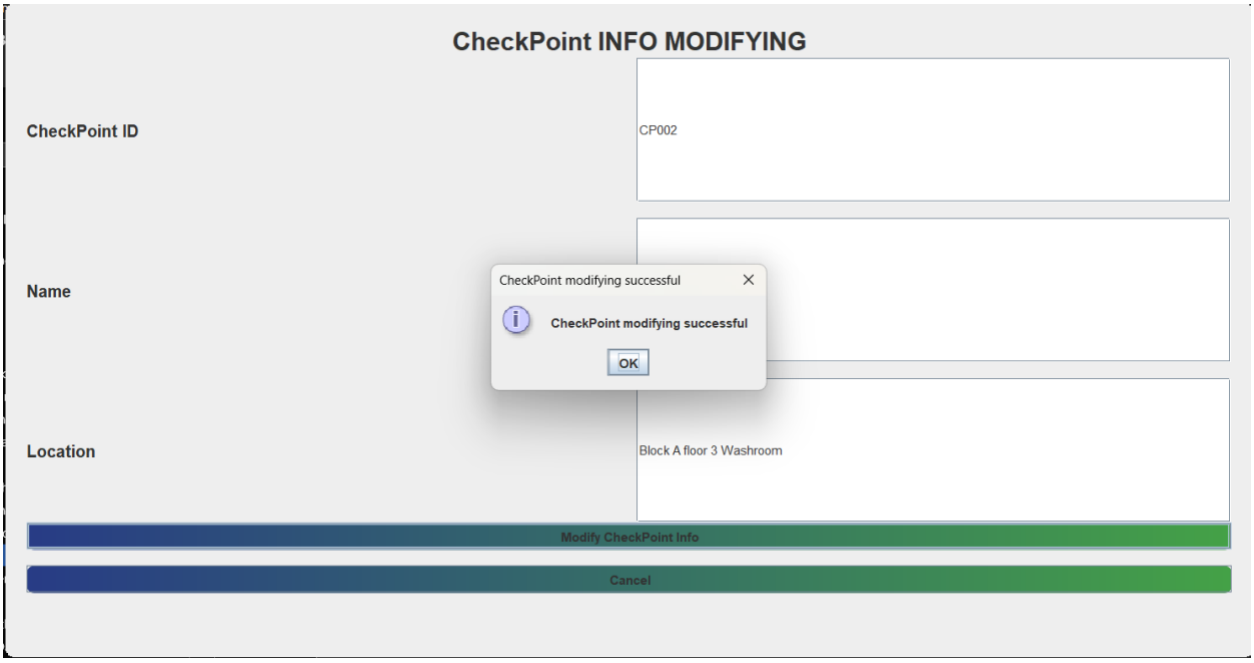


Users can add new checkpoints by clicking on "Add New CheckPoint" on the checkpoint management page. They will be prompted to fill in information such as the checkpoint name and location. Once the information is entered, they can click on "Add New CheckPoint" again to create the checkpoint.

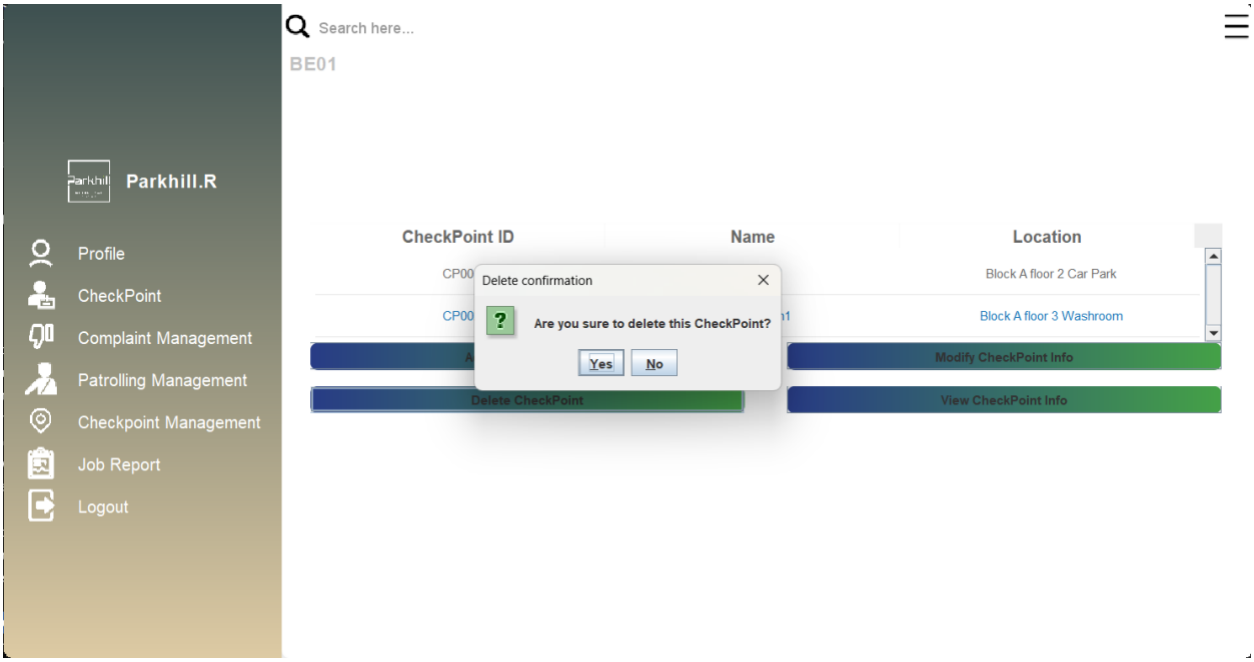


After adding a new checkpoint using the "Add New CheckPoint" button, the system will automatically display a prompt confirming the successful addition of the checkpoint.

8.3.6.2 Modify CheckPoint Info

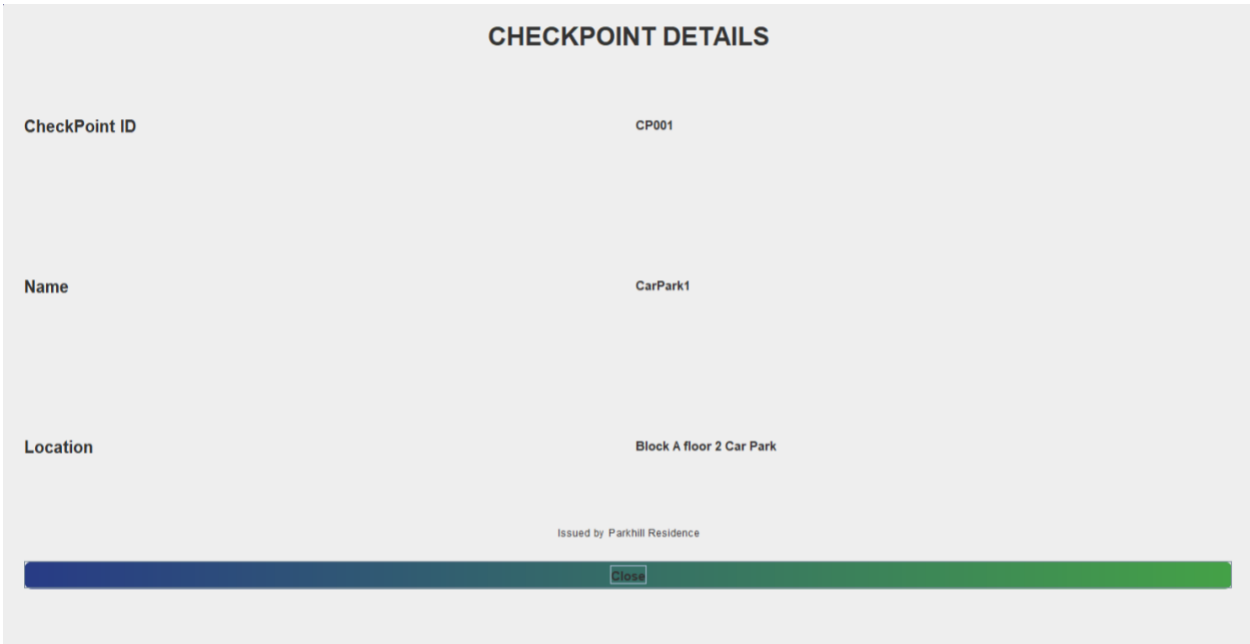


8.3.6.3 Delete CheckPoint



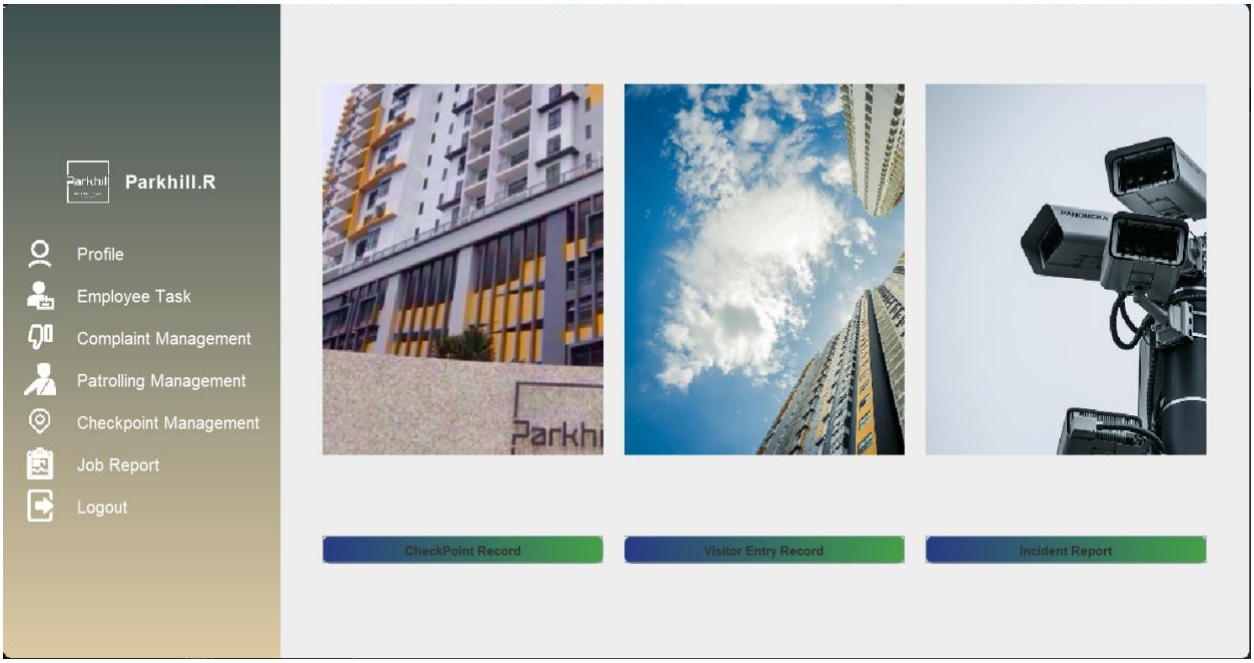
The user can select the row or column they want to delete and then click "Delete Check Point" to remove it from the list.

8.3.6.4 View CheckPoint Info

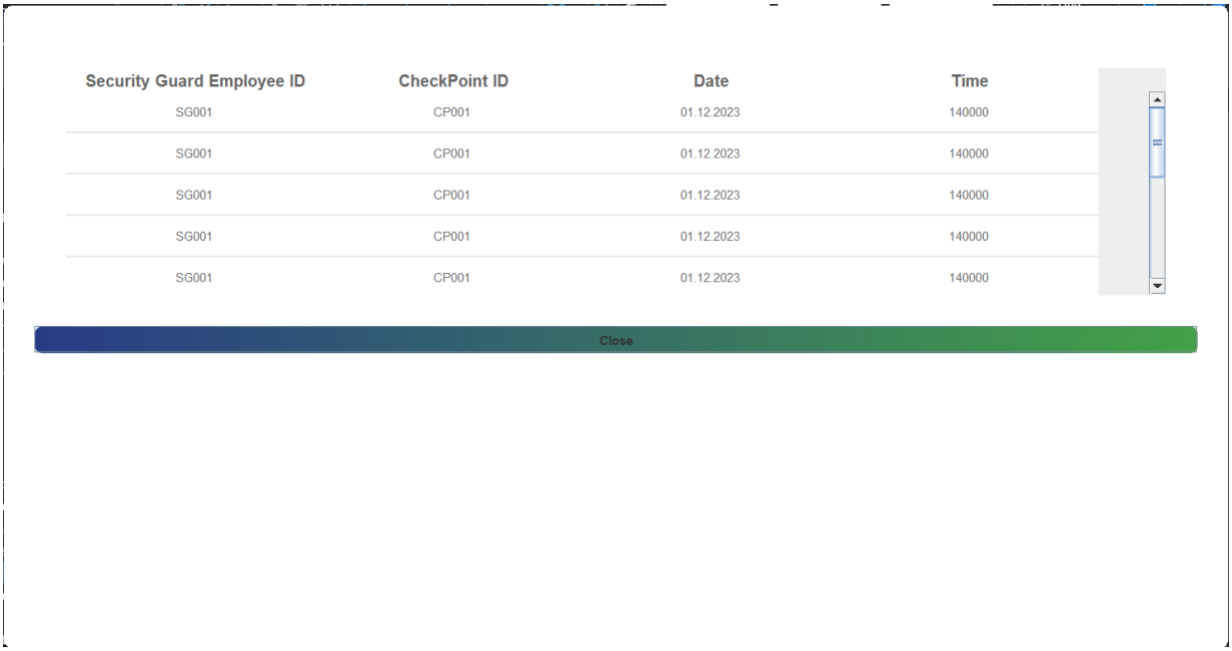


After selecting the desired rows and columns, the user can click on "View CheckPoint Info" to view the corresponding details of the checkpoint.

8.3.7 Job report



When the building executive clicks on the "job report" section, they can view different types of records such as checkpoint record, visitor entry record, and incident record.

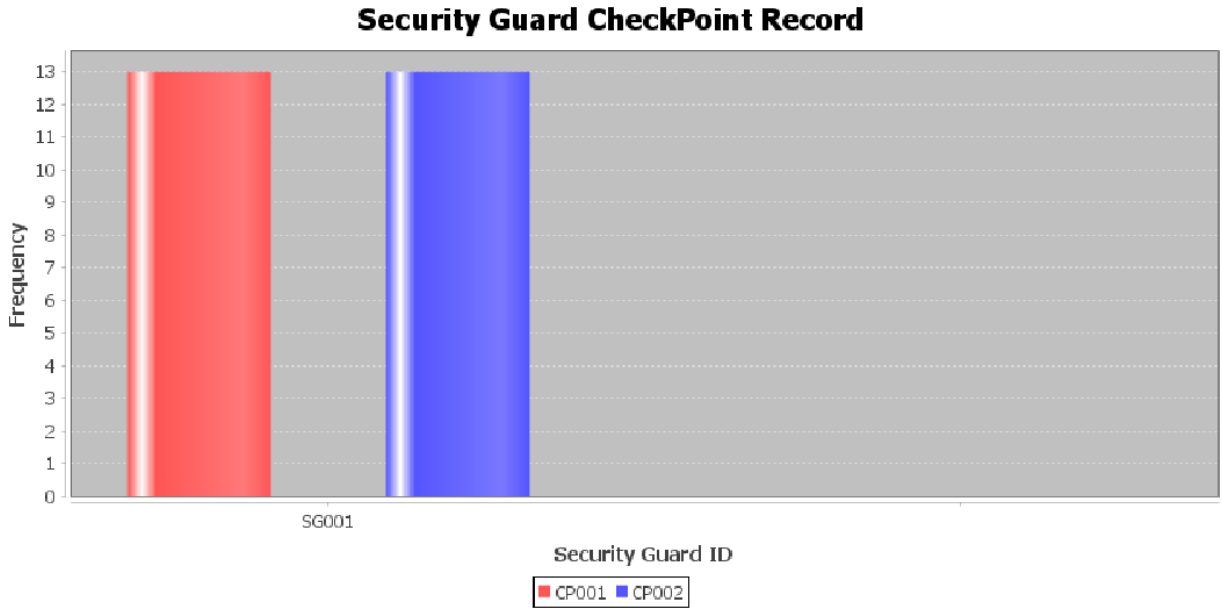


Upon clicking on various buttons, the Building Executive can access different types of records, such as Check Point Record, Visitor Entry Record, and Incident Record. For instance, upon clicking on the "CheckPoint Record" button, a new interface is displayed, presenting all the checkpoint data reported by the security guards in a clear and organized manner.

8.4 Report or File Generation

8.4.1 System Report Panel

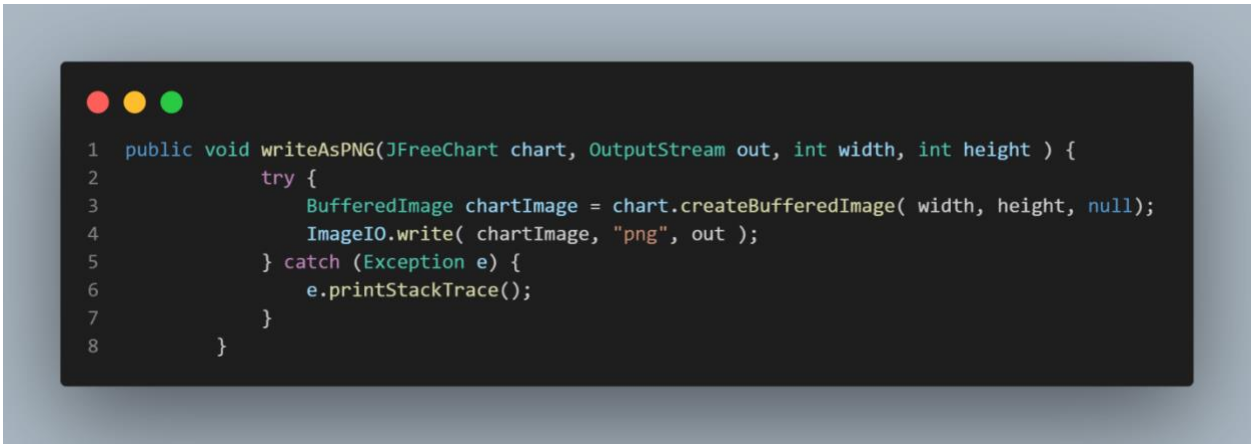
```
public reportPanel() throws IOException, ClassNotFoundException {
            JFreeChart checkPointRecordChart = ChartFactory.createBarChart("Security Guard CheckPoint Record", "Security Guard ID", "Frequency", getCheckPointDataset(), PlotOrientation.VERTICAL, true, true, false);
            JFreeChart executivePieChart = ChartFactory.createPieChart("Executive User Number", getExecutiveDataset(), true, true, false);
            JFreeChart complaintPieChart = ChartFactory.createPieChart("User Complaint Frequency", getUserComplaintDataset(), true, true, false);
            JFreeChart taskPieChart = ChartFactory.createPieChart("Employee Task Amount", getEmployeeTaskDataset(), true, true, false);
            JFreeChart bookingChart = ChartFactory.createBarChart("Resident Booking Frequency", "Resident Username", "Frequency", getBookingDataset(), PlotOrientation.VERTICAL, true, true, false);
            JFreeChart visitorPieChart = ChartFactory.createPieChart("Visitor Pass Visiting Frequency", getVisitorDataset(), true, true, false);

            ChartPanel checkPointPanel = new ChartPanel(checkPointRecordChart);
            ChartPanel executivePanel = new ChartPanel(executivePieChart);
            ChartPanel bookingPanel = new ChartPanel(bookingChart);
            ChartPanel complaintPanel = new ChartPanel(complaintPieChart);
            ChartPanel visitorPanel = new ChartPanel(visitorPieChart);
            ChartPanel taskPanel = new ChartPanel(taskPieChart);

            JPanel contentPane = new JPanel();
            contentPane.setLayout(new GridLayout(6, 1, 10, 10));
            contentPane.add(checkPointPanel);
            contentPane.add(executivePanel);
            contentPane.add(bookingPanel);
            contentPane.add(complaintPanel);
            contentPane.add(visitorPanel);
            contentPane.add(taskPanel);

            setViewportView(contentPane);

            OutputStream out = Files.newOutputStream(Paths.get("src/ReportPNG/checkPoint.png"));
            writeAsPNG(checkPointRecordChart, out, 1280, 720);
            out = Files.newOutputStream(Paths.get("src/ReportPNG/executive.png"));
            writeAsPNG(executivePieChart, out, 1280, 720);
            out = Files.newOutputStream(Paths.get("src/ReportPNG/complaint.png"));
            writeAsPNG(complaintPieChart, out, 1280, 720);
            out = Files.newOutputStream(Paths.get("src/ReportPNG/task.png"));
            writeAsPNG(taskPieChart, out, 1280, 720);
            out = Files.newOutputStream(Paths.get("src/ReportPNG/booking.png"));
            writeAsPNG(bookingChart, out, 1280, 720);
            out = Files.newOutputStream(Paths.get("src/ReportPNG/visitor pass.png"));
            writeAsPNG(visitorPieChart, out, 1280, 720);
        }
```



This code creates a report panel that displays multiple charts using the JFreeChart library. The panel consists of six charts, each showing data that is obtained without the use of a database. The charts include a bar chart for security guard checkpoint records, a pie chart for executive user numbers, a bar chart for resident booking frequency, a pie chart for user complaint frequency, a pie chart for visitor pass visiting frequency, and a pie chart for employee task amount. The code

creates a JPanel and adds the six chart panels to it using a grid layout. Finally, the content pane of the report panel is set to be the JPanel. All of the charts are created using datasets such as VisitorDataset, BookingDataset, and UserComplaintDataset, and are presented in a visually appealing manner through the use of charts

8.4.2 writeAsPNG



```
public void writeAsPNG(JFreeChart chart, OutputStream out, int width, int height ) {
            try {
                BufferedImage chartImage = chart.createBufferedImage( width, height, null);
                ImageIO.write( chartImage, "png", out );
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

This method takes a JFreeChart object, creates a PNG image of the chart with specified dimensions, and writes the image to an OutputStream.

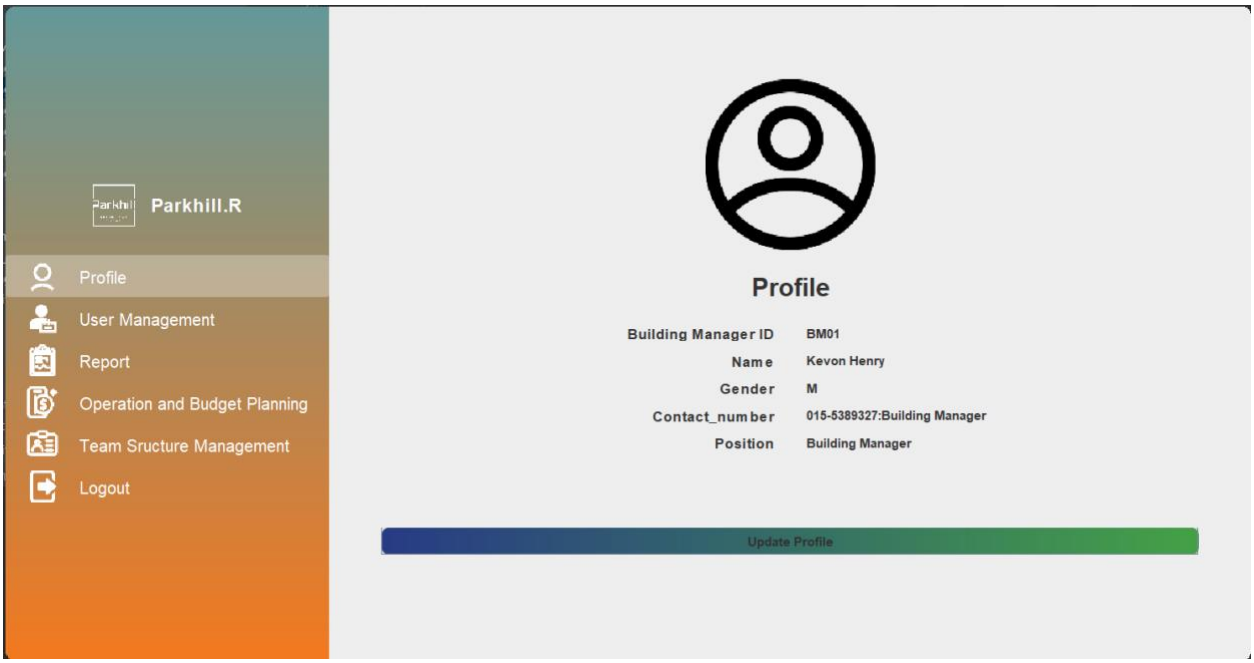
9 TP067435 Edmund Chen Siang Zuan

9.1 Building manager

9.1.1 Functional requirements

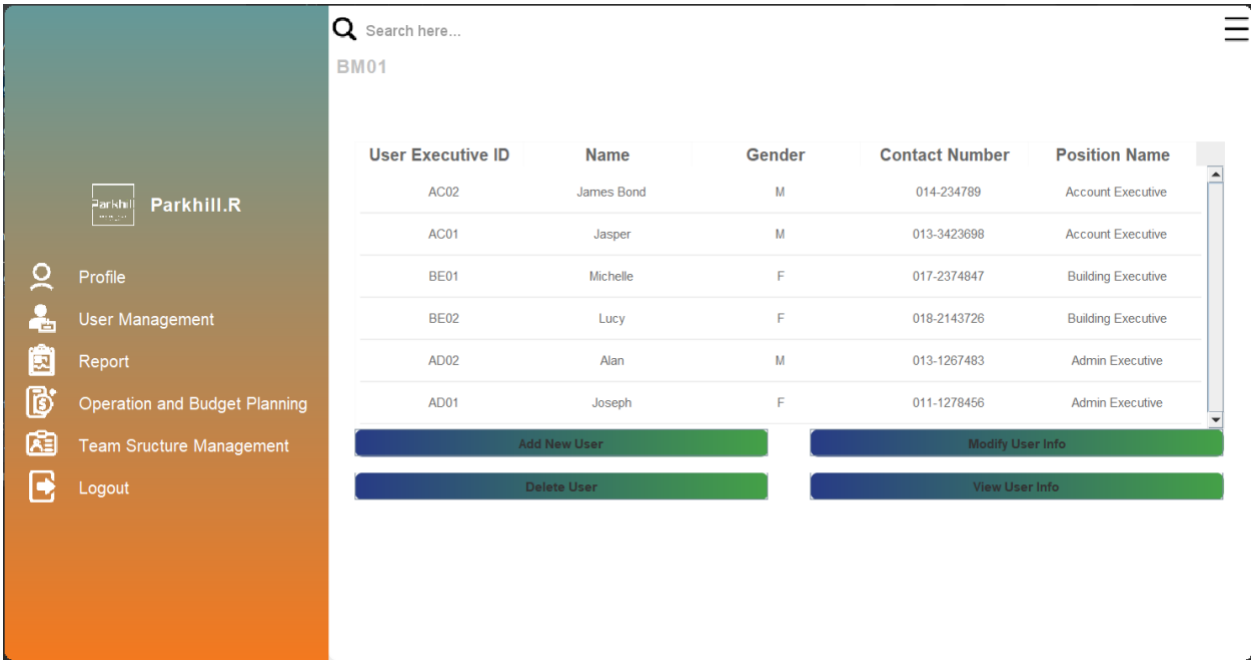
1. Profile interface
2. User Management
3. Report Interface
4. Operation and Budget Planning Interface
5. Logout

9.1.2 Profile



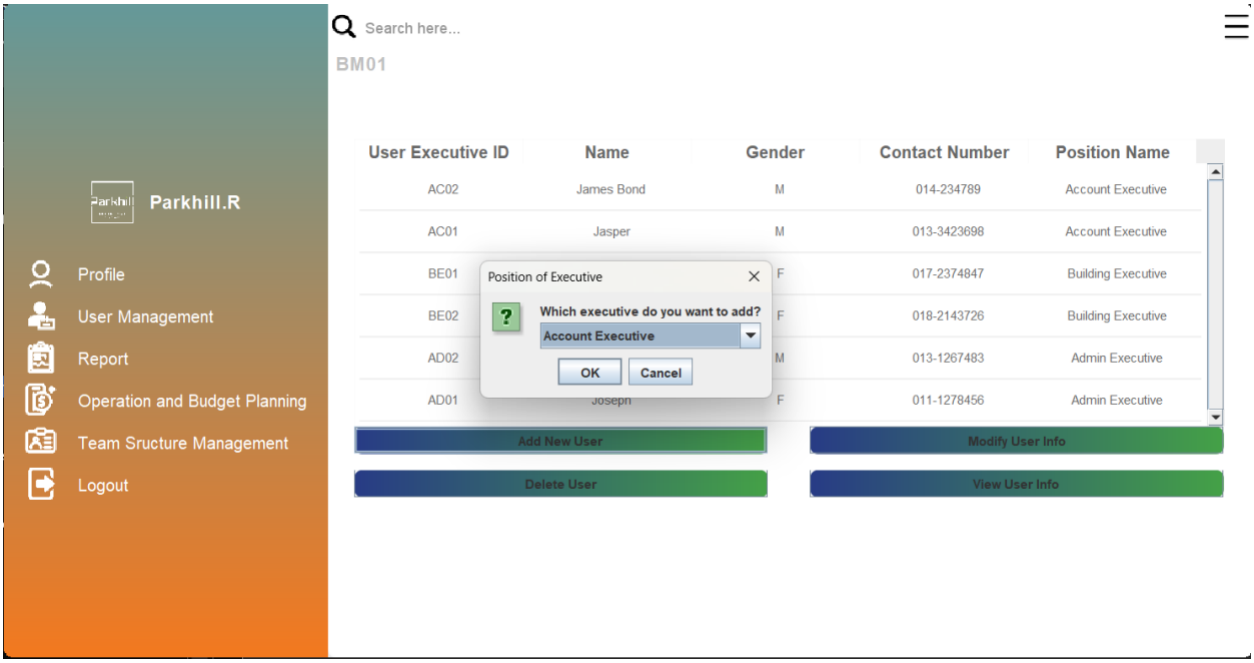
Once the building manager has successfully logged into the system, they will be directed to their designated interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the building manager profile page is accessible through this interface and allows users to modify their profile information.

9.1.3 User management

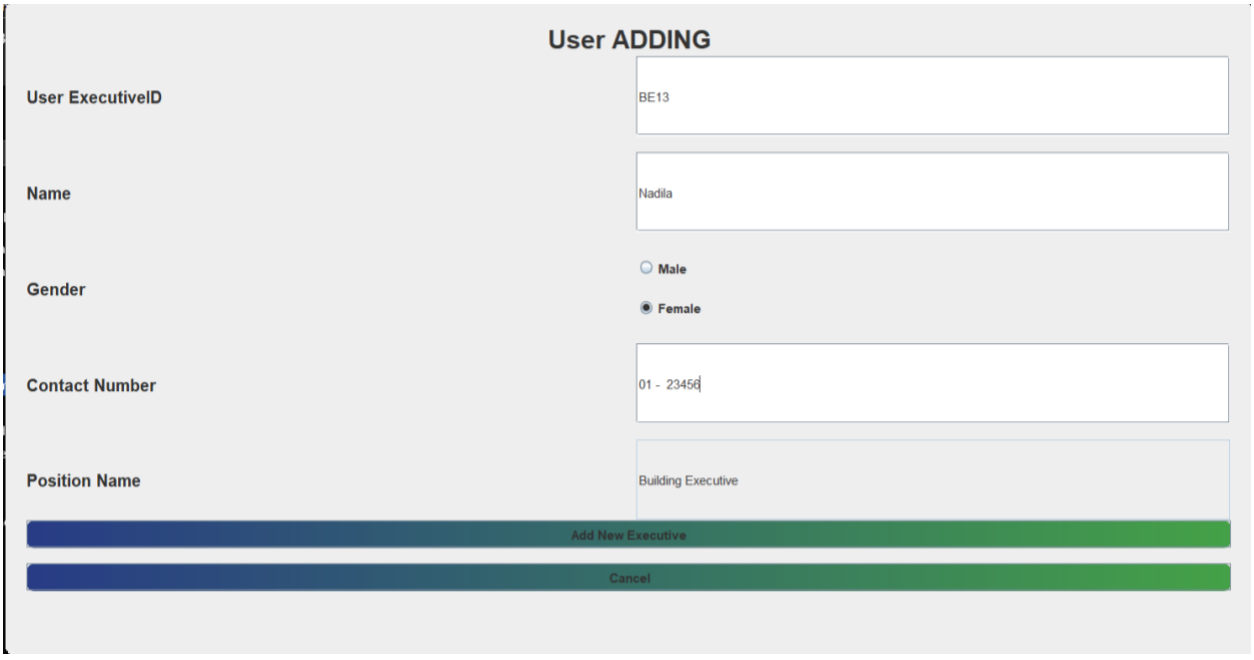


The user management interface allows building managers to view and manage user executive information through a table format. The table includes details such as the user executive ID, name, and gender. Managers can perform various user-related operations on this page, including adding new users, modifying user information, deleting users, and viewing user details.

9.1.3.1 Add New User

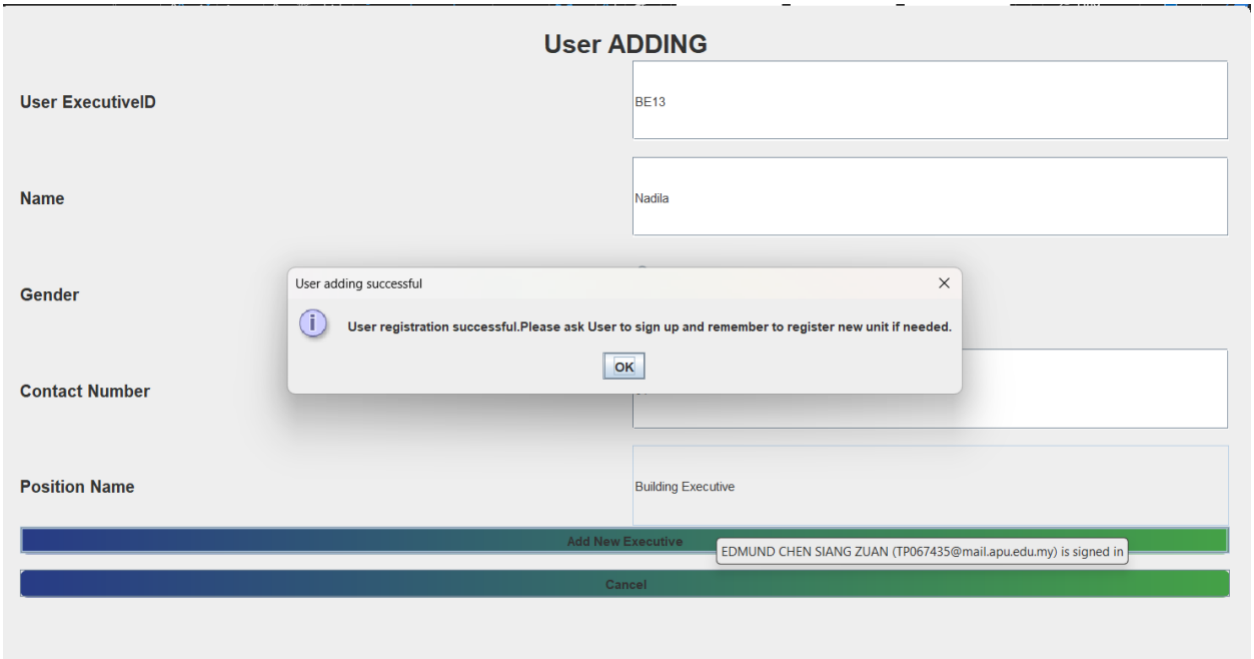


When the building manager clicks on the “Add New User" button, a window will pop up with options to select the type of user to be added. After selecting the desired user type, the manager can click the "OK" button to proceed with adding the user.



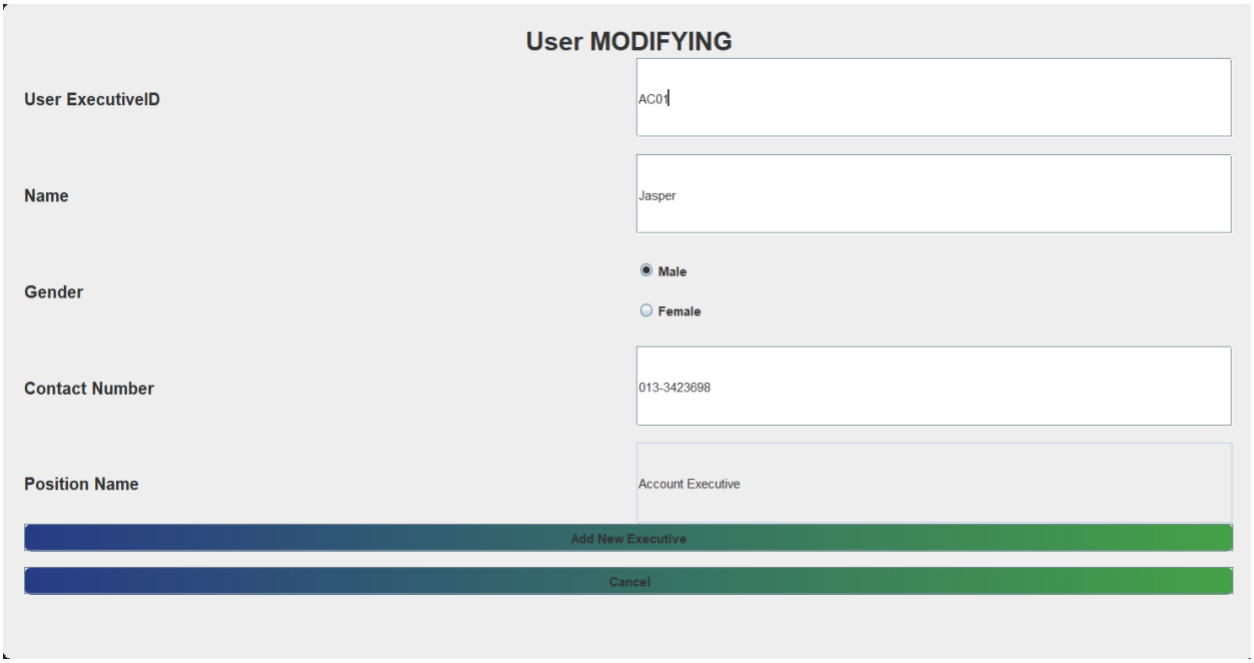
To add a new executive, the user first selects the executive type from a drop-down menu and clicks "OK". Then, a form is displayed where the user can enter the executive's details, such as ID, name,

gender, and contact number. After filling in the information, the user clicks "Add" to save the new executive's data.



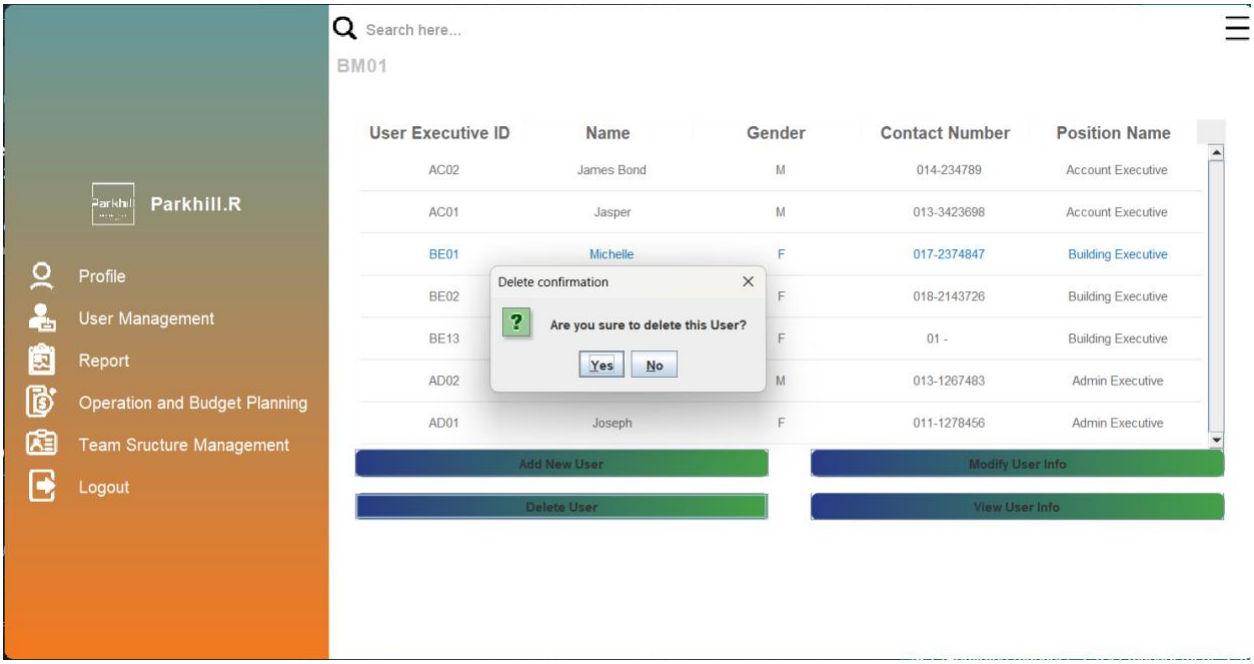
Once the new executive has joined successfully, a pop-up window will appear and indicate that the executive has joined successfully, and the original interface will be skipped.

9.1.3.2 Modify User Info



Please provide more context about the code you are referring to. Without the code, I cannot provide an accurate response to your question.

9.1.3.3 Delete User

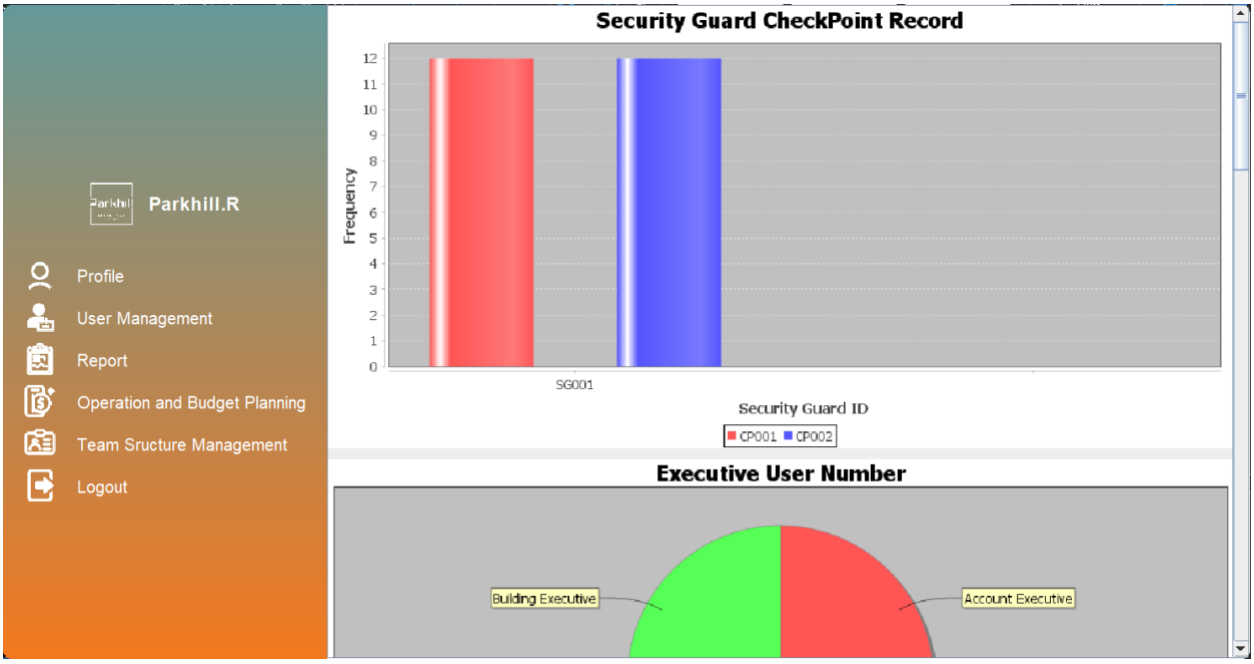


When the user selects the desired executive and presses "Delete User", the system will prompt the user to confirm the deletion of the executive, press to confirm, the executive will be deleted successfully.

9.1.3.4 View User Info

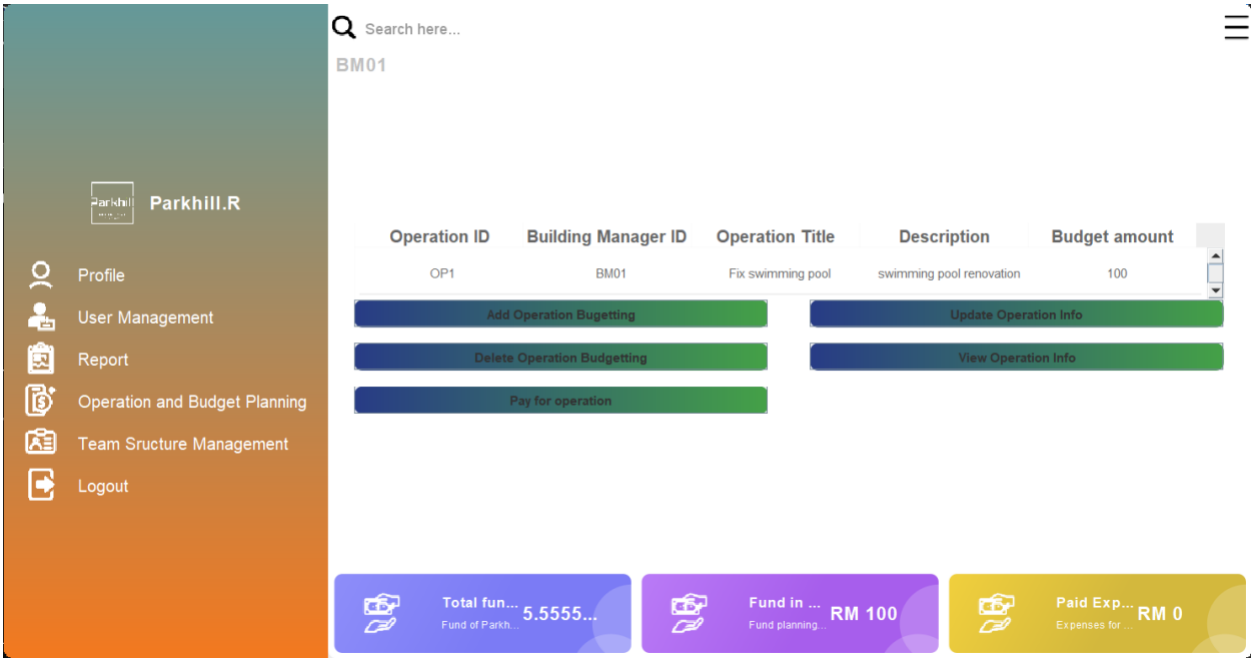
When the user selects an executive and clicks on "View User Info", a pop-up window will display detailed information about the selected executive, including their name, ID, gender, and other relevant information.

9.1.4 Report



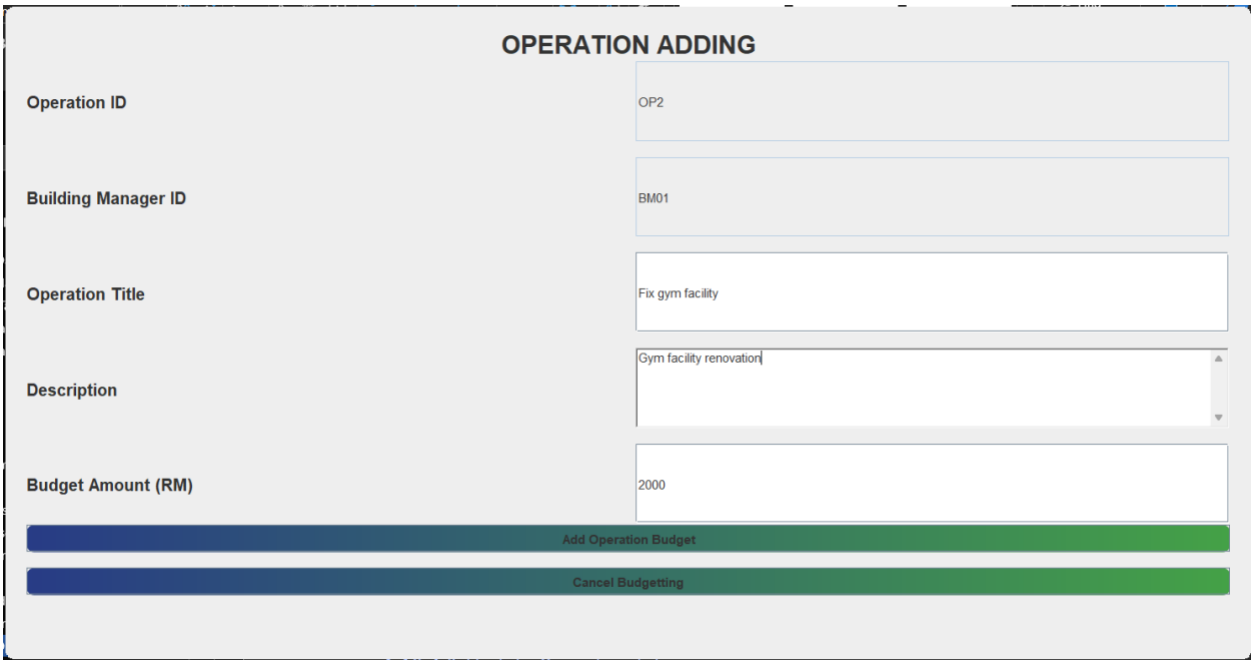
The building manager has access to the report section, which displays recent information in an easy-to-read visual format. The section features dynamic bar and pie charts that update automatically, providing real-time data to help the building manager make informed decisions.

9.1.5 Operation and budget planning

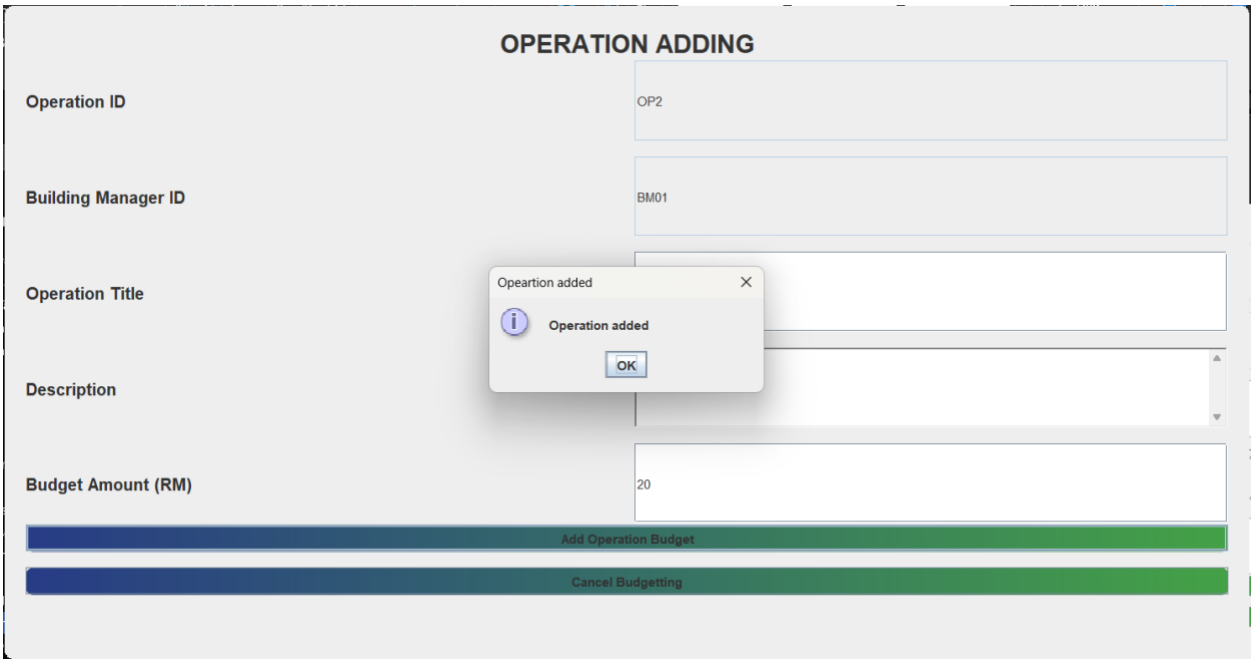


This page is the "Operation and Budget Planning" section for the Building Manager. Here, users can report facility and maintenance costs, and view all the information reported by other Building Managers. This includes the ID of other Building Managers, the required maintenance budget, and the description of the issue.

9.1.5.1 Add Operation Budgeting

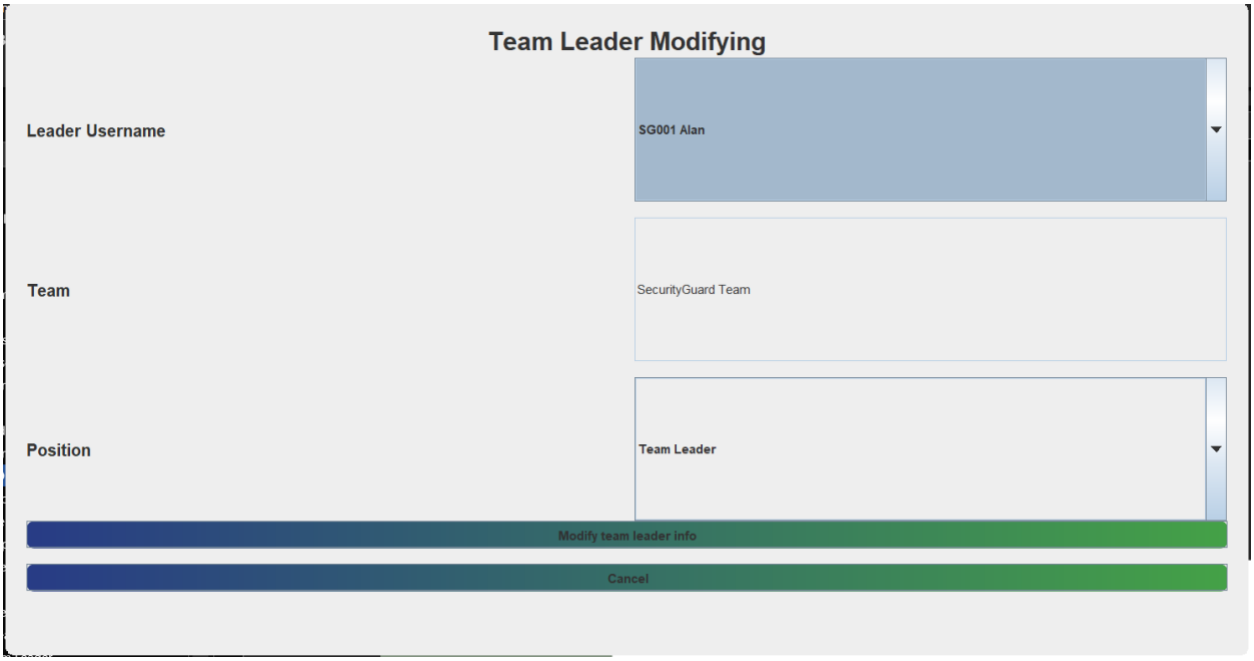


When a user wants to add a new budget planning report, they can click on "Add Operation Budgeting" to enter another interface where they will be prompted to fill in the relevant information. Once the information is filled in, the user can click on "Add Operation Budgeting" to complete the report.



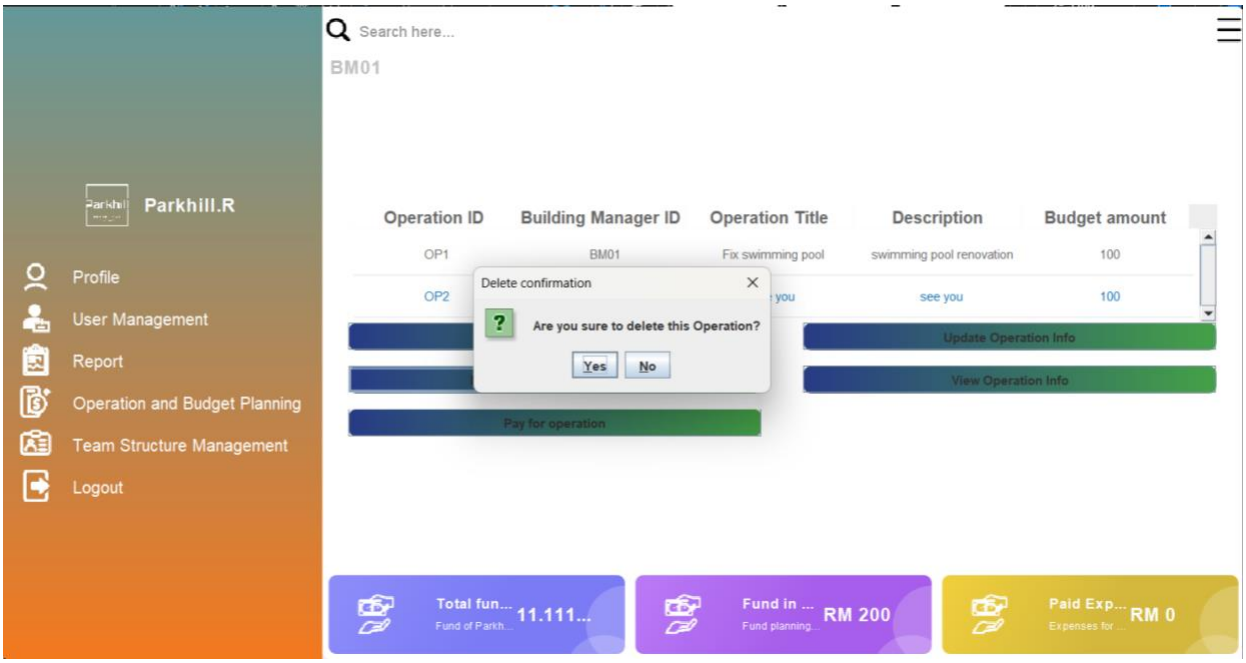
After successfully adding a new budget planning report, the system will display a pop-up window confirming the successful addition and return the user to the main page.

9.1.5.2 Update Operation Info



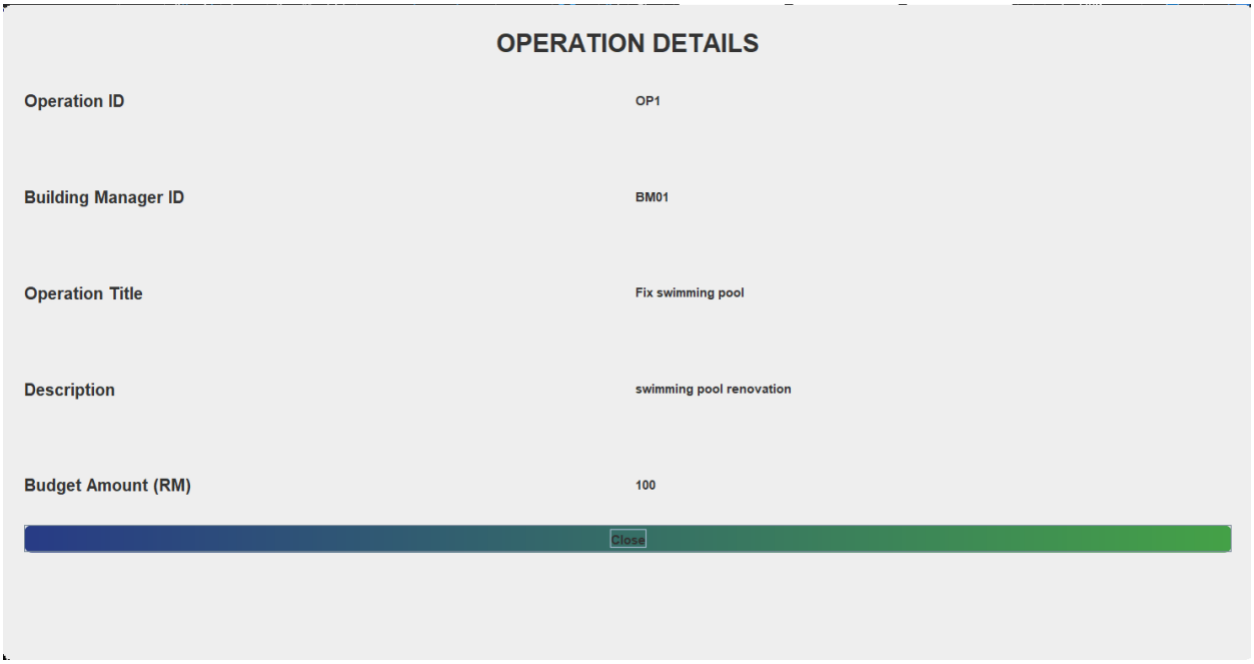
When a building manager wants to modify the team structure, they can do so by clicking on "Update Operation Info" to make the changes.

9.1.5.3 Delete Operation Budgeting



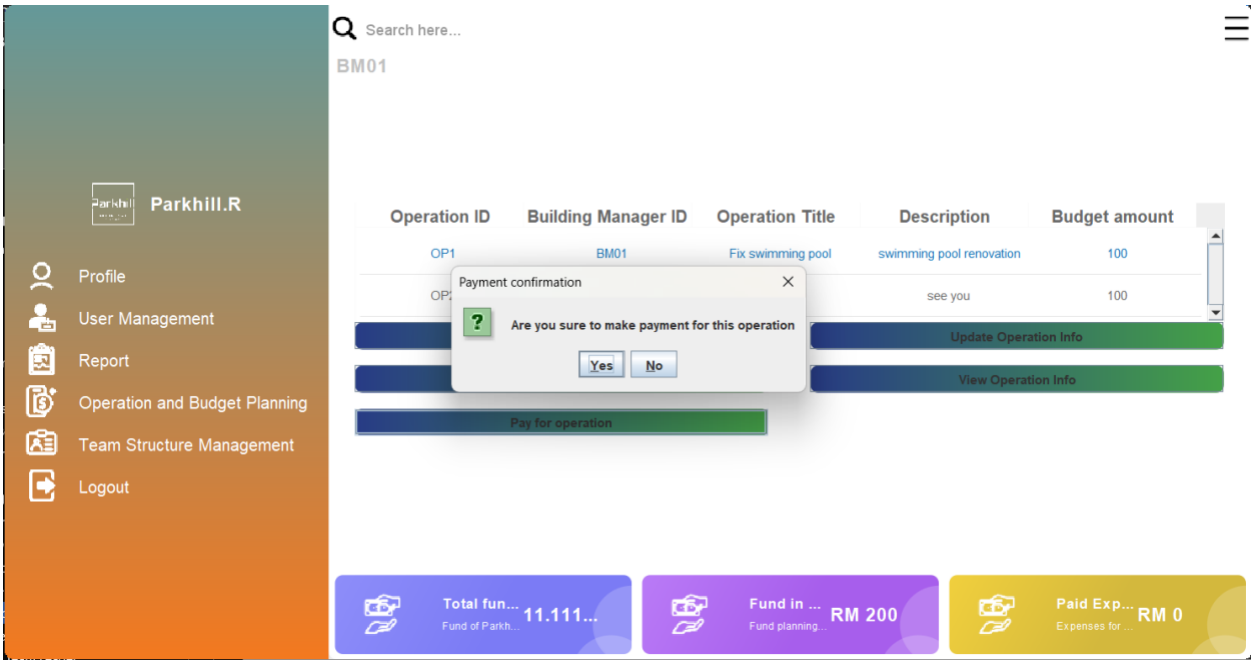
Building Manager can delete a row by clicking on "Delete Operation Budgeting". After clicking, the system will prompt for confirmation of the deletion.

9.1.5.4 View Operation Info



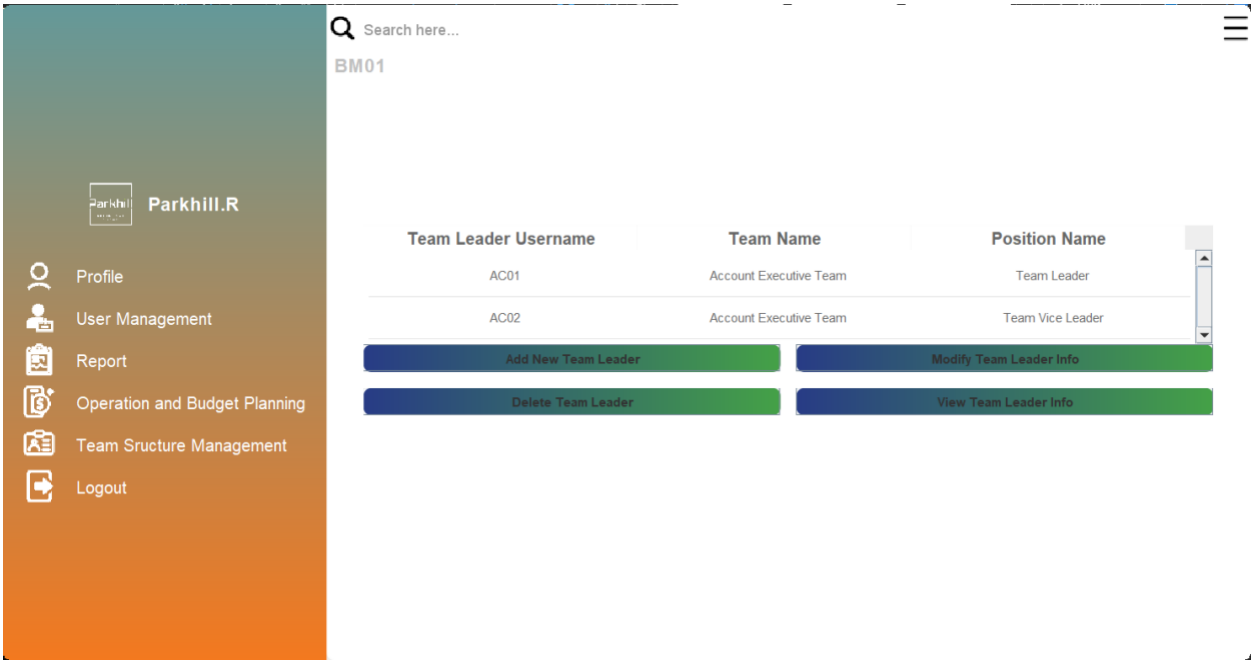
Building managers can select the desired rows and columns and click on "View Operation Info" to view the related information, such as operation ID, building manager ID, and operation title.

9.1.5.5 Pay for Operation



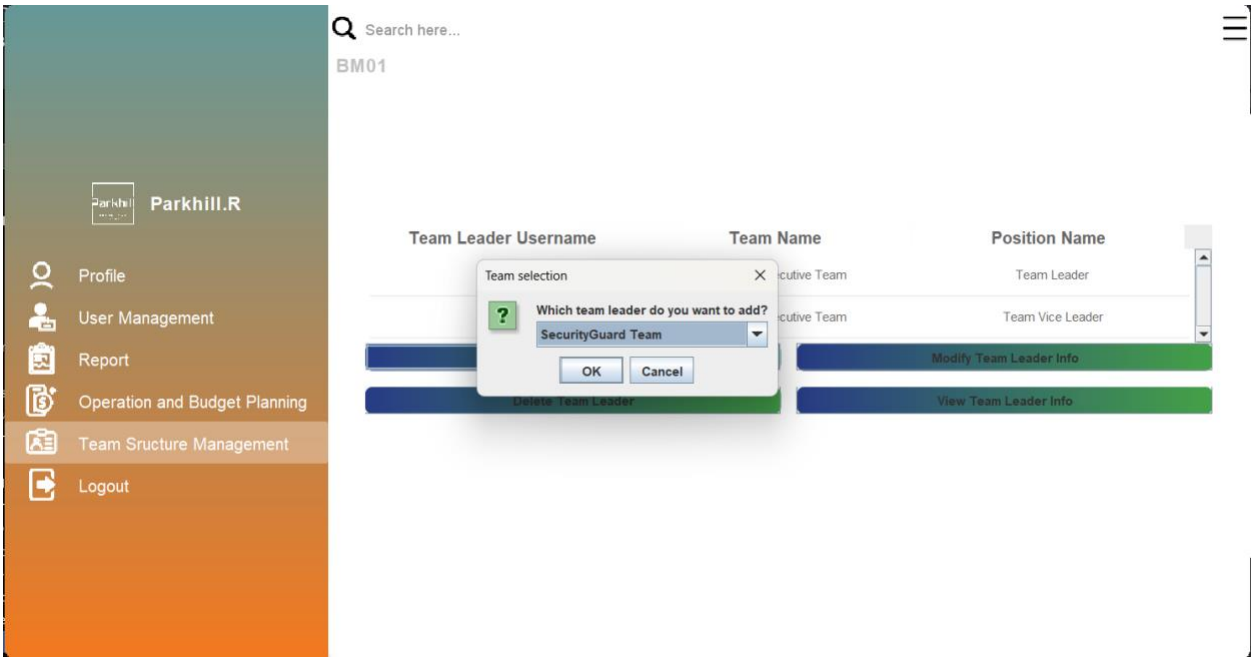
A building manager can initiate a payment for a specific operation by clicking on the "Pay for Operation" button. The system will prompt the user to confirm the payment before processing it.

9.1.6 Team structure management

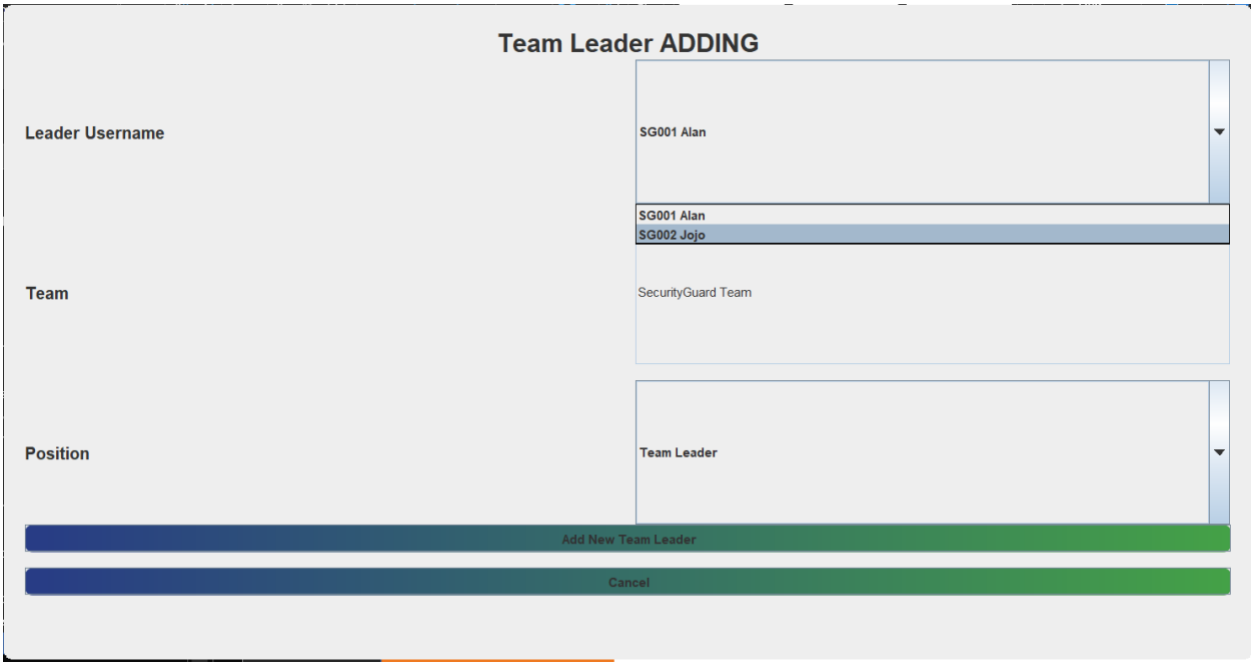


This text appears to be missing context and content to provide a summary or to be made shorter. Please provide more information or specify which text you are referring to.

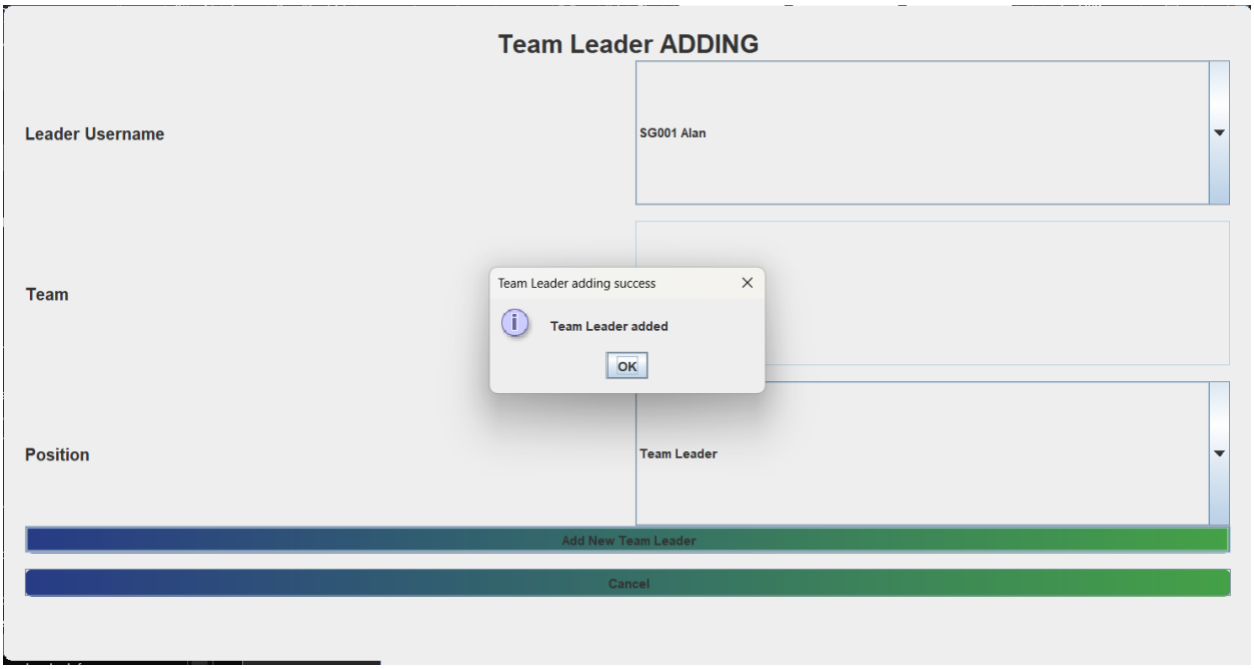
9.1.6.1 Add New Team Leader



After clicking on "Add New Team Leader", a window will pop up for the user to select the type of team leader they want to add.

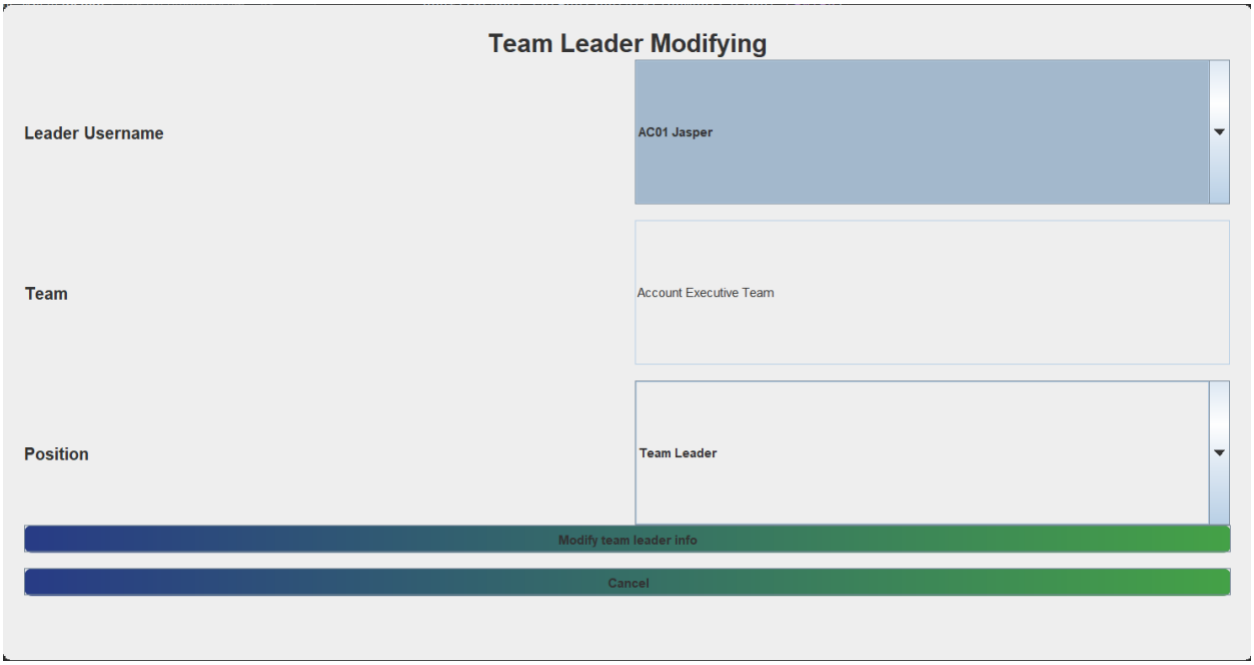


After clicking "Add New Team Leader", a window will pop up asking the user to choose the type of team leader they want to add. Then, the system prompts the user to fill in the relevant information of the team leader to be added. Finally, clicking "Add New Team Leader" will add the team leader to the system.



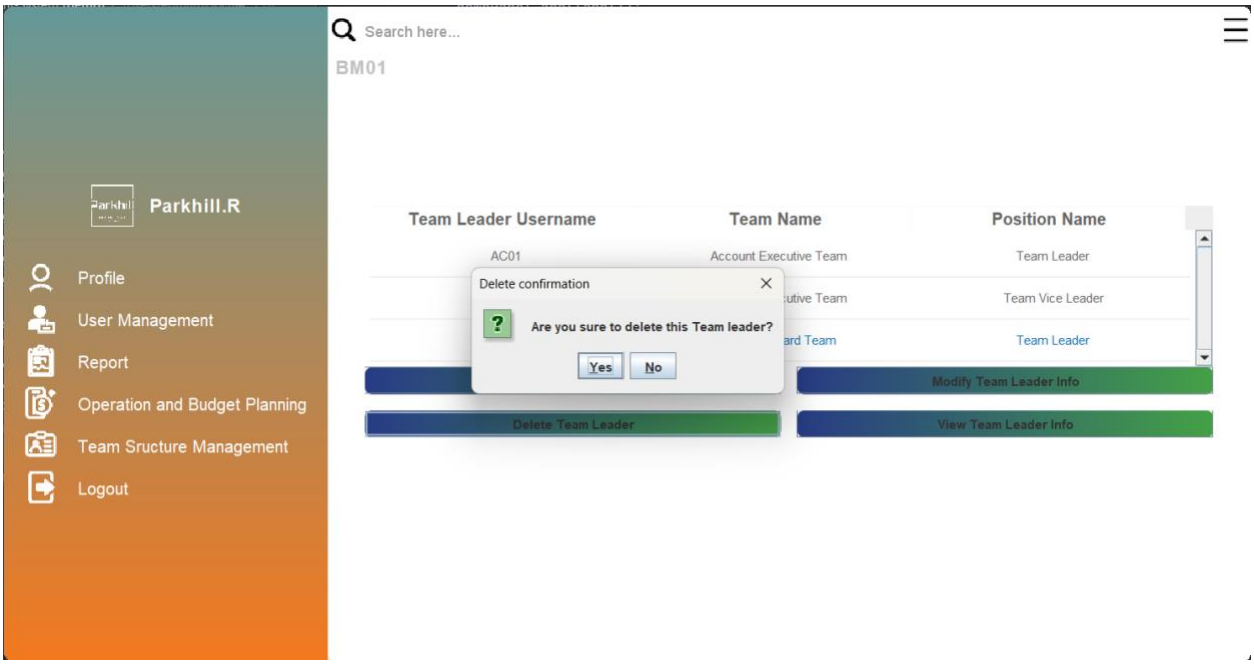
Finally, the system will display a message to confirm that the team leader has been successfully added.

9.1.6.2 Modify Team Leader Info



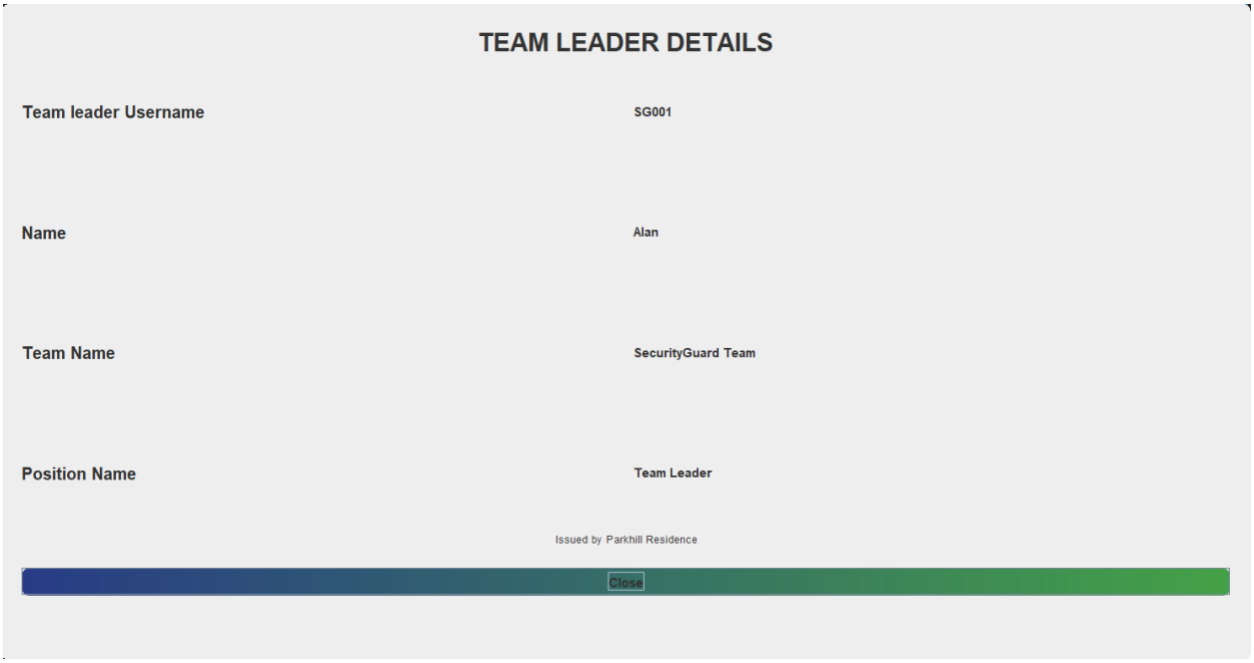
After clicking on "Modify Team Leader Info", the building manager can make modifications to the relevant information and then click on the button again to save the changes.

9.1.6.3 Delete Team Leader



Building managers can delete selected rows by clicking on "Delete Team Leader".

9.1.6.4 View Team Leader



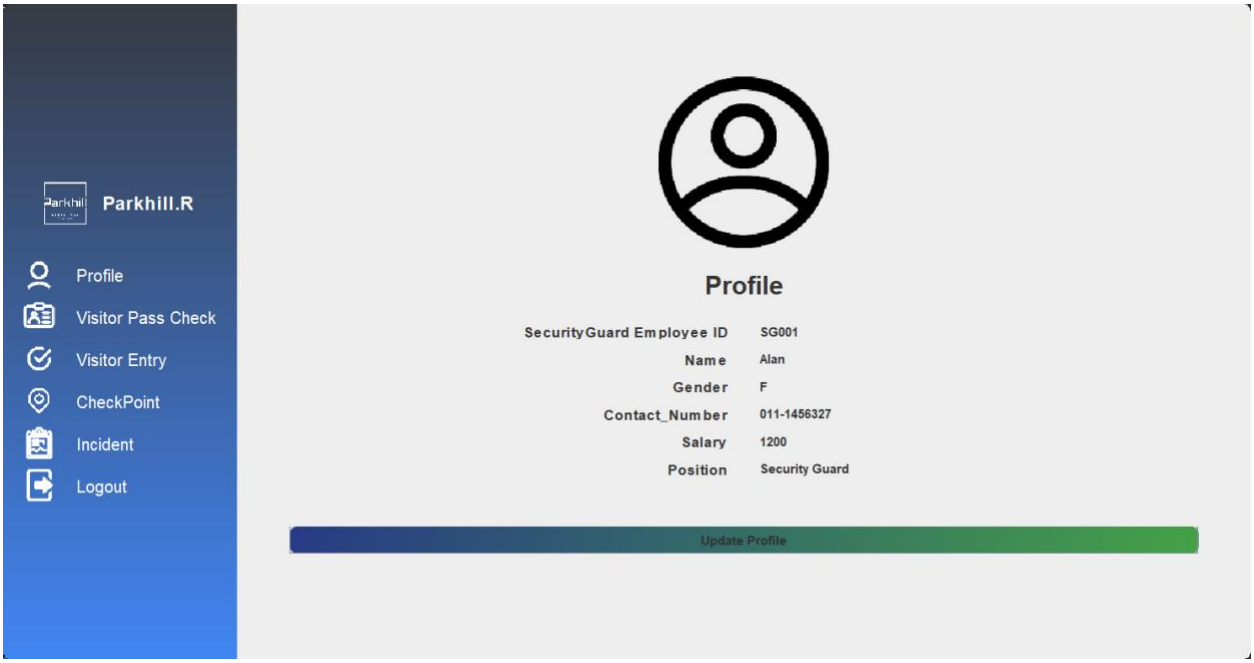
The building manager can click "View Team Leader" to see the relevant information for the selected row.

9.2 Security guard

9.2.1 Functional requirements

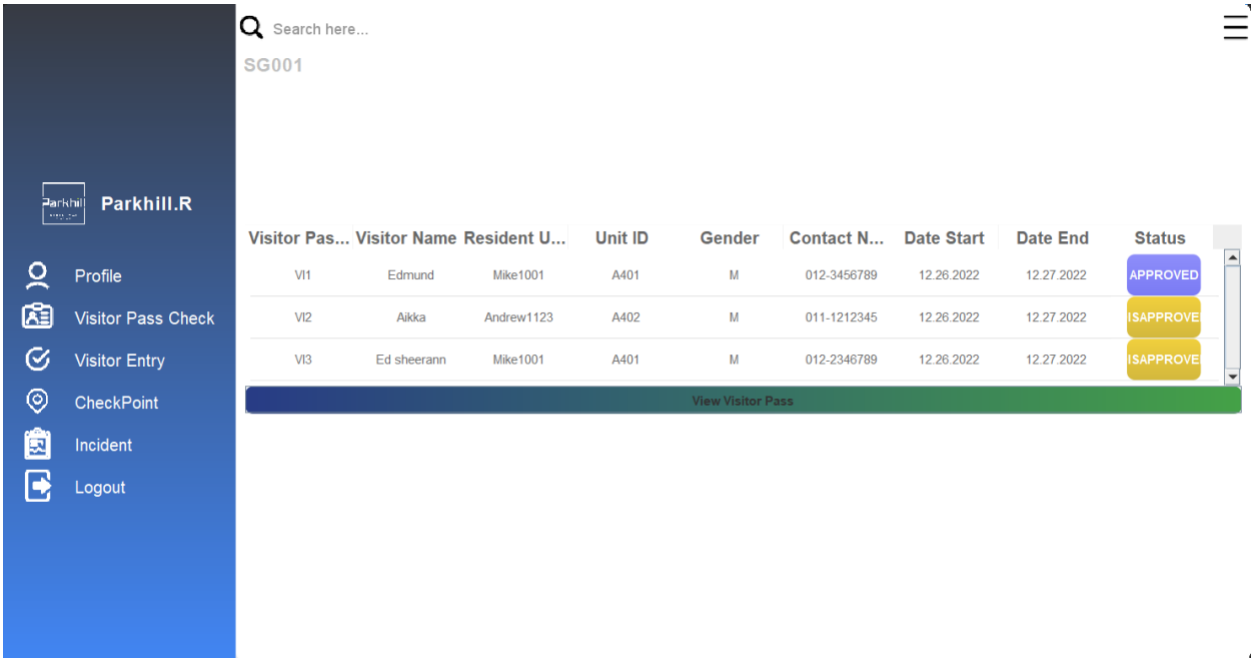
1. Profile Interface
2. Visitor Pass Interface
3. Visitor Entry Interface
4. Incident Interface
5. Logout

9.2.2 Profile



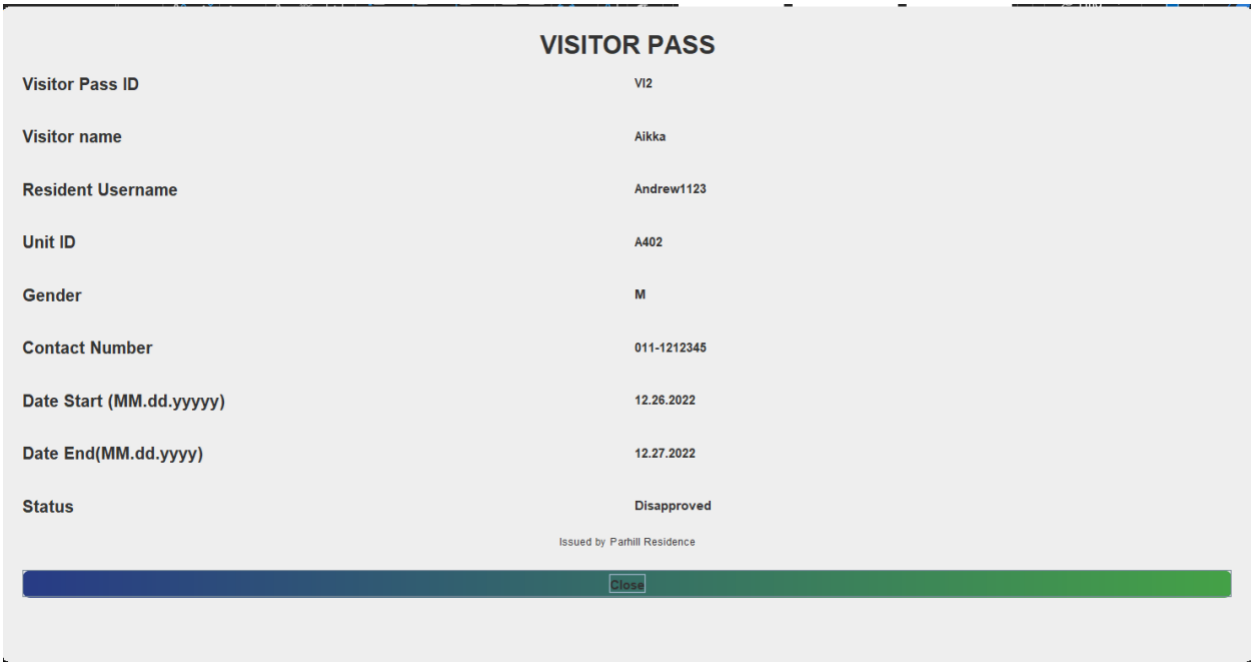
Once the security guard has successfully logged into the system, they will be directed to their designated interface. This interface consists of a menu area on the left-hand side, allowing the user to navigate between various sections and log out of the system. Additionally, the security guard profile page is accessible through this interface and allows users to modify their profile information.

9.2.3 Visitor pass check



When the user clicks on the 'Visitor Pass Check' button, they will be directed to a screen displaying information related to the visitor pass, such as ID and gender.

9.2.3.1 View Visitor Pass

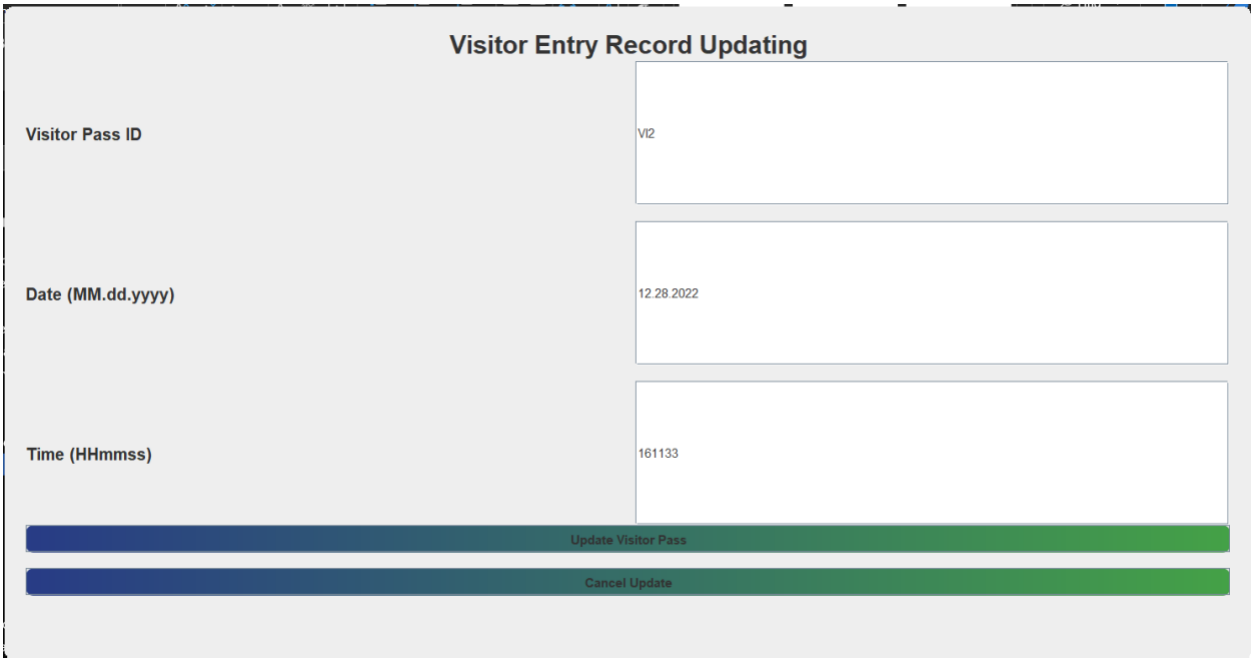


When a security guard needs to view information about a specific visitor, they can click on the corresponding visitor pass and select 'View Visitor Pass'. The page will then display all relevant information about the visitor, such as their name, ID, gender, etc.

9.2.4 Visitor entry

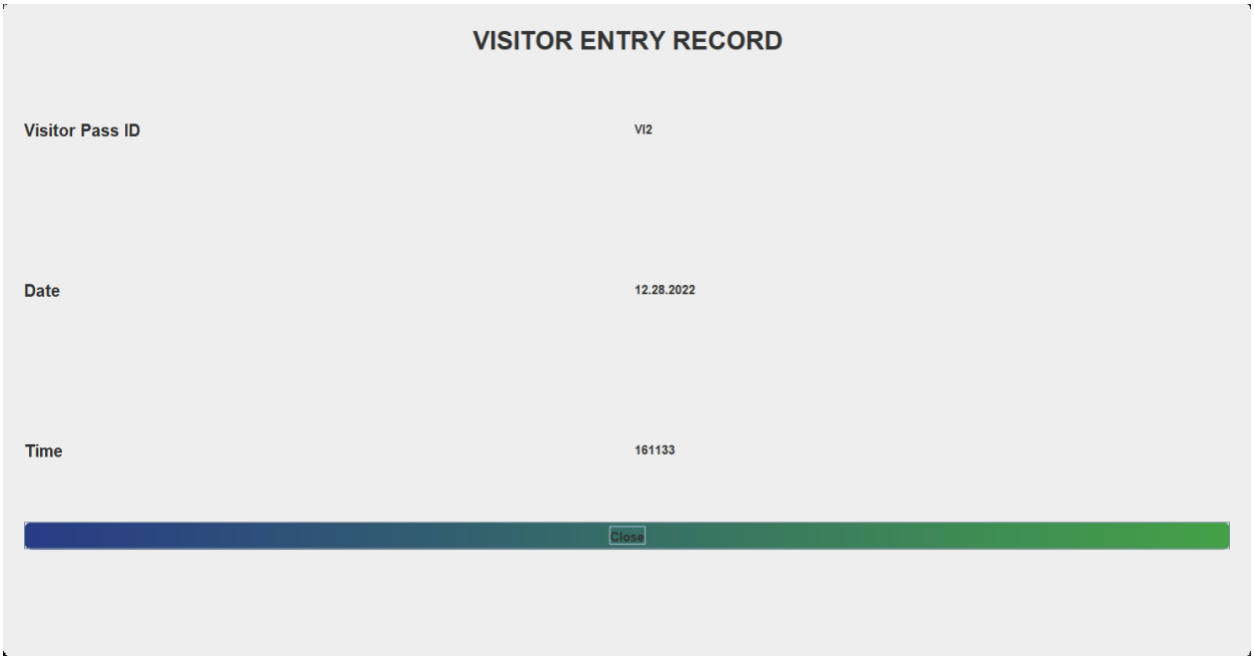
When accessing the Visitor Entry section, the user will be able to view the details of all the visits made by visitors, including the exact time of their entry.

9.2.4.1 Update Visitor Pass



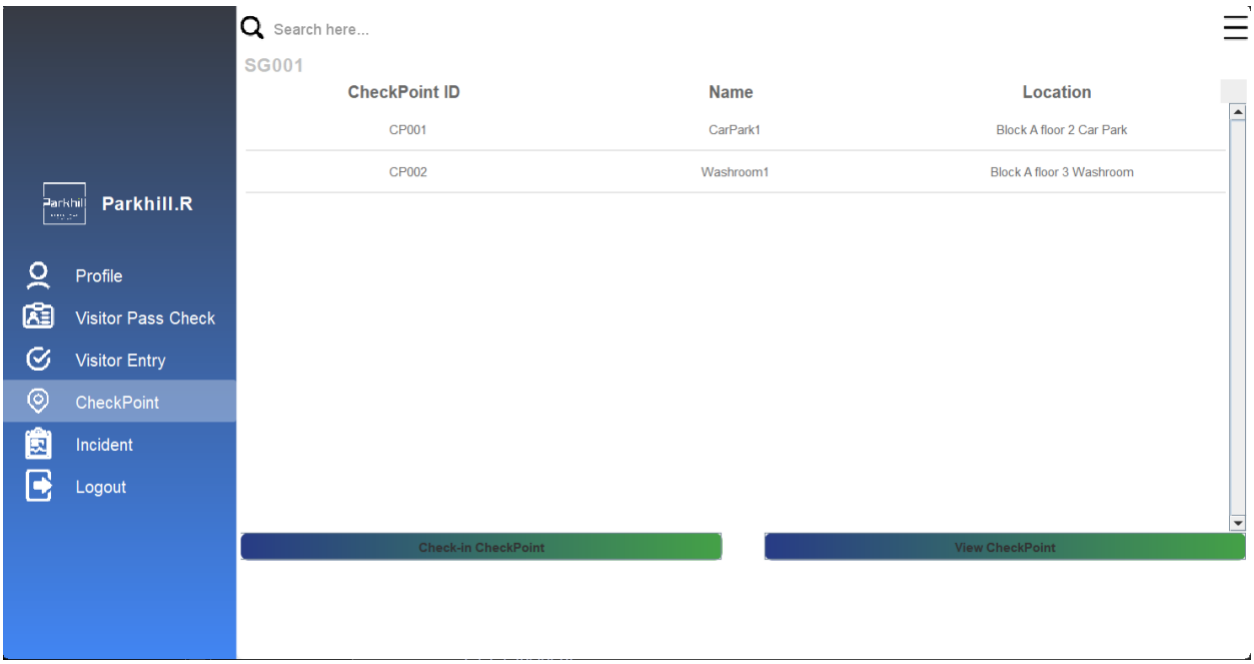
When the user selects a visitor and clicks on "Update Visitor Pass," the system will display the visitor's information in a form that allows the user to make changes. Once changes are made, the user can click "Update Visitor Pass" to save them, or "Cancel Update" to discard them and return to the previous page.

9.2.4.2 View Visitor Pass



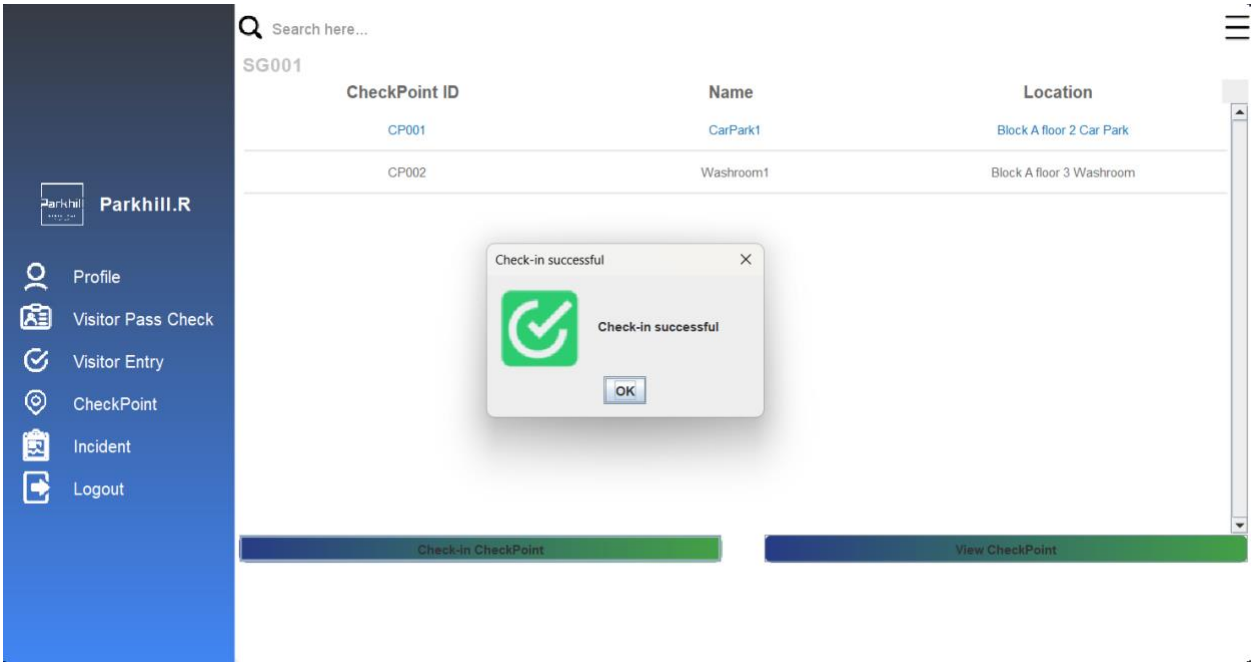
By selecting the checkpoint, users can click on the 'View CheckPoint' button to access a more detailed interface with information such as the checkpoint location and related records.

9.2.5 Checkpoint



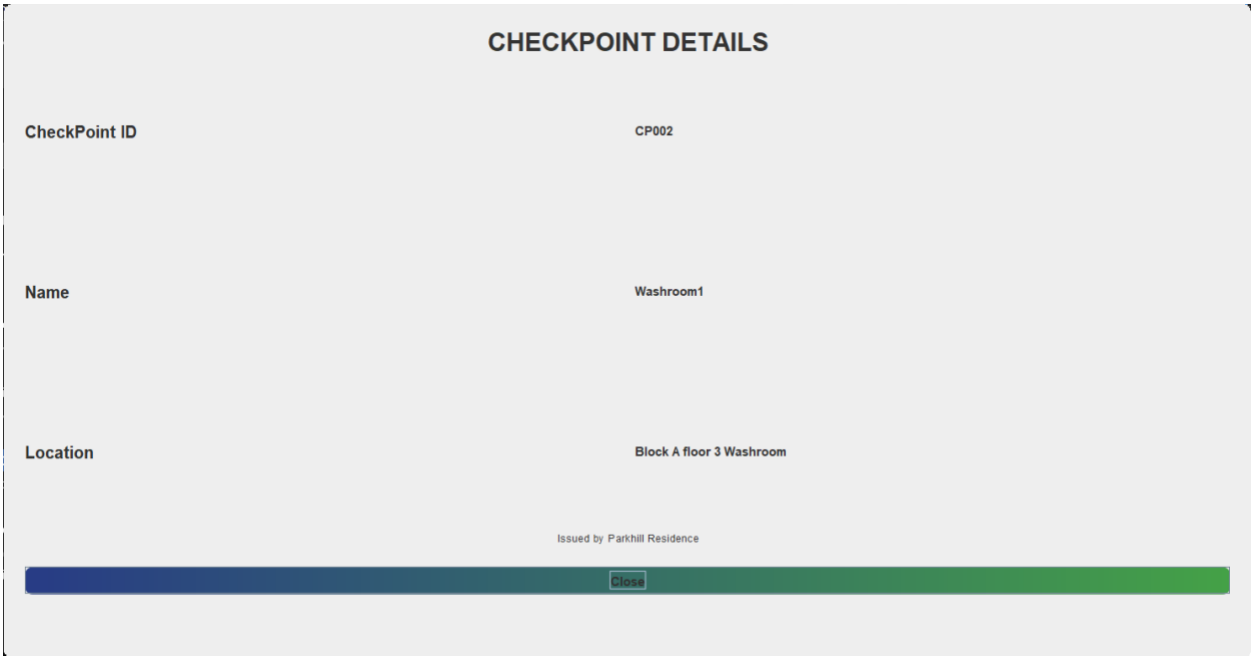
When the user wants to access a checkpoint, they can navigate to the checkpoint section where they will see a list of all checkpoints with their specific information, such as location, ID, and status.

9.2.5.1 Check-in CheckPoint



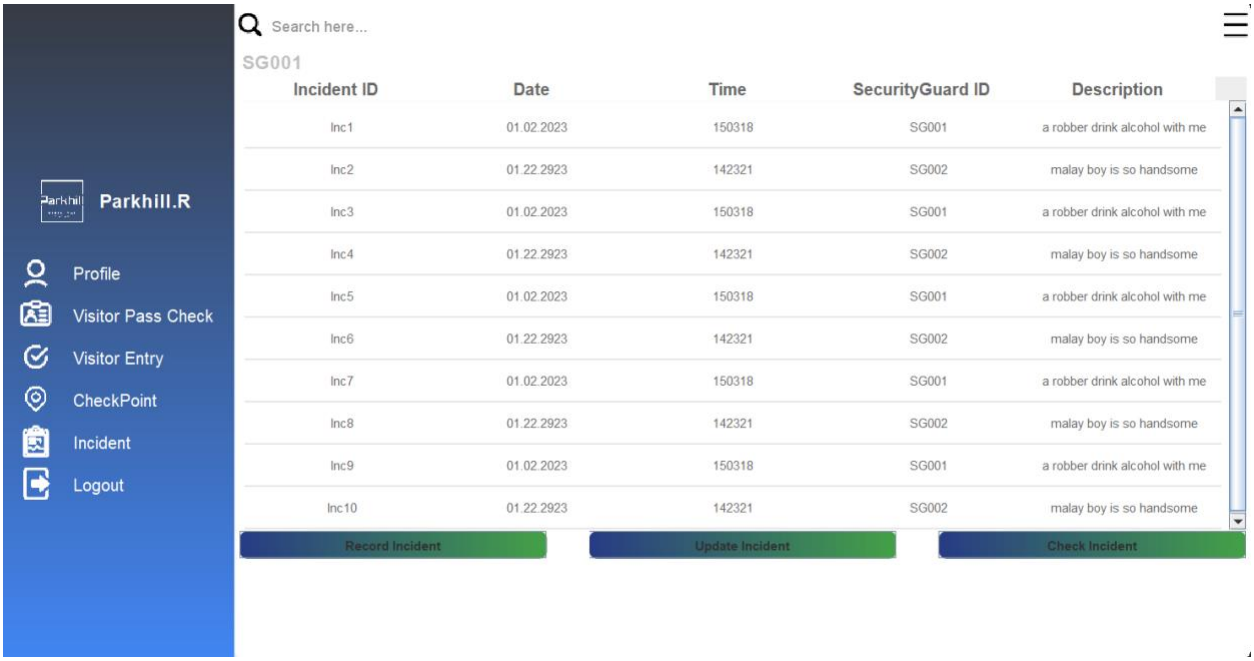
When users select a checkpoint to check in, they can click on the 'Check-in CheckPoint' button. The system will then automatically display a message indicating that the check-in was successful.

9.2.5.2 View CheckPoint



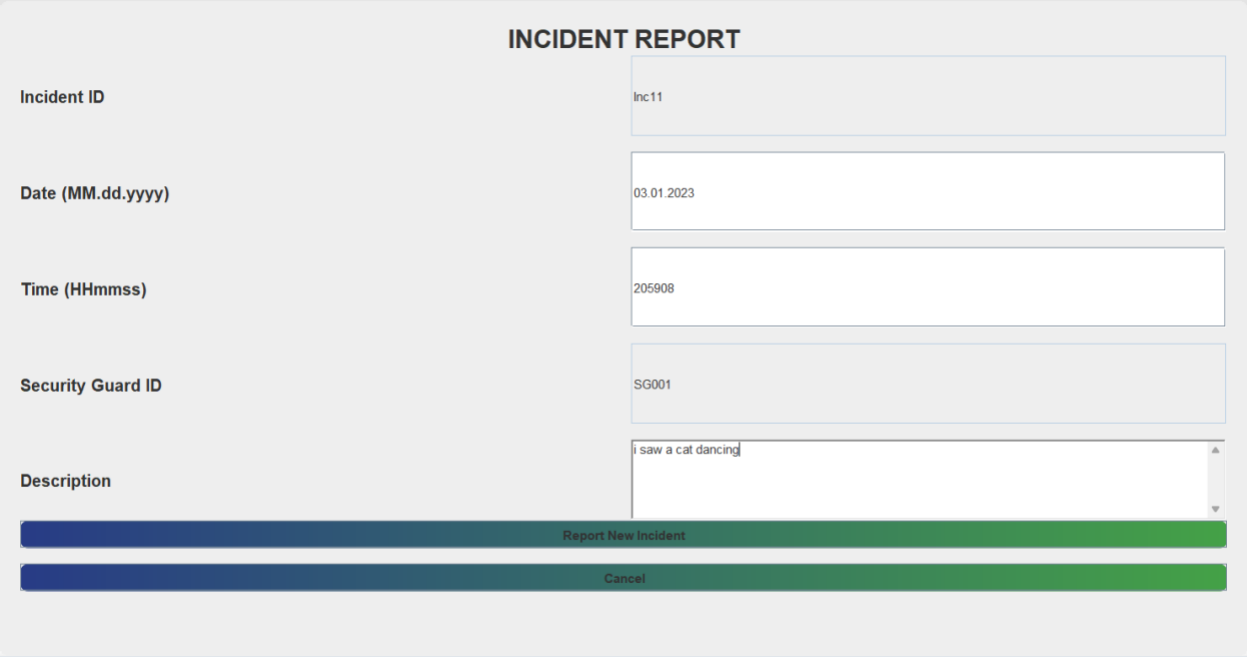
By selecting the checkpoint, users can click on the 'View CheckPoint' button to access a more detailed interface with information such as the checkpoint location and related records.

9.2.6 Incident



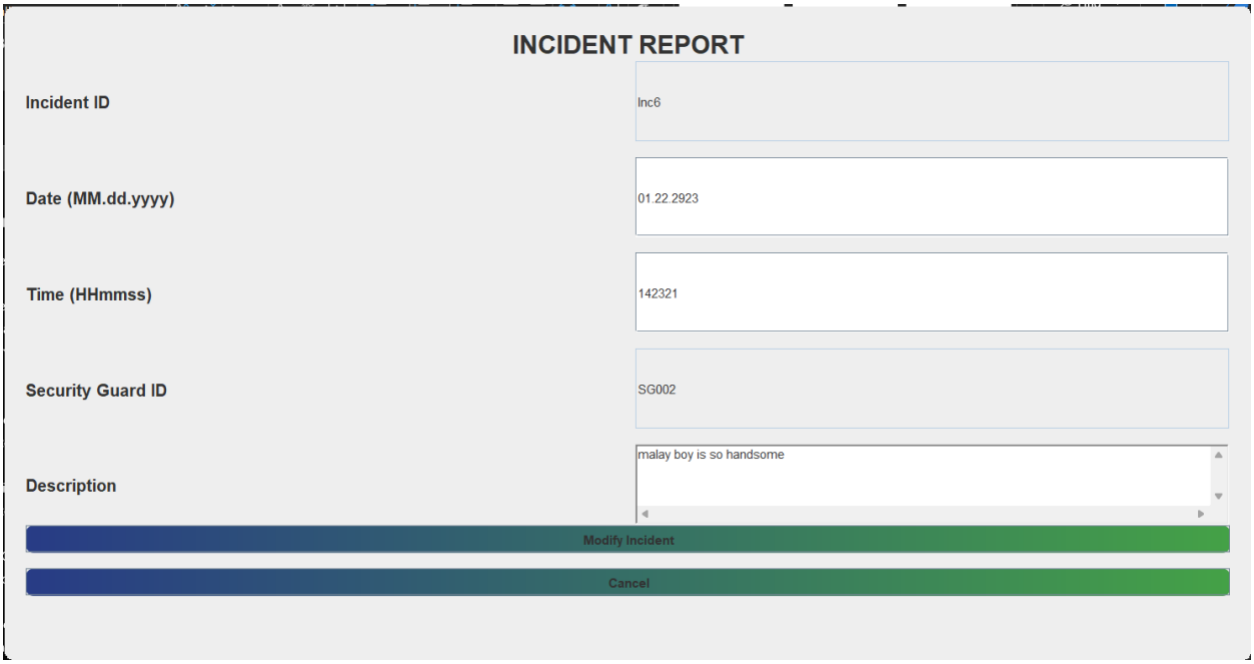
On the "Incident" page, users can view all recent security guard reports, including details such as the date and time of the incident, the ID of the security guard involved, and a brief description of the incident.

9.2.6.1 Record incident



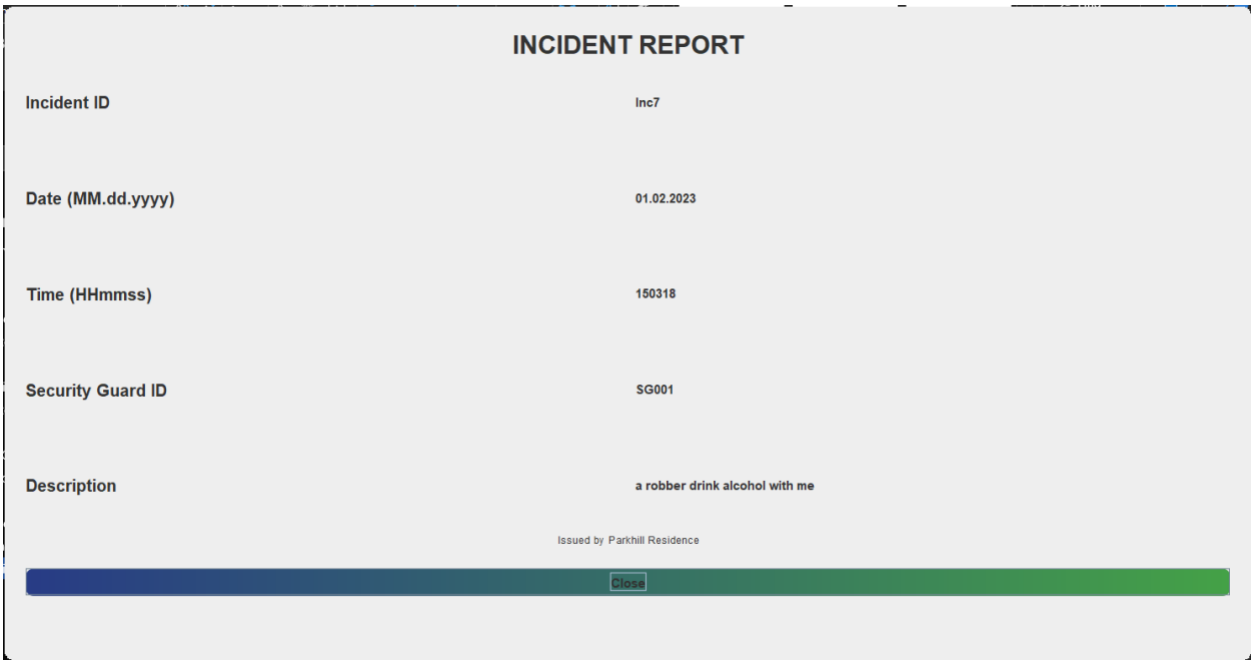
To report an incident, users can click on the "Record Incident" button, which will prompt them to enter relevant information about the incident. Once the required details are filled in, they can click on "Report New Incident" to submit the report.

9.2.6.2 Update Incident



After selecting the incident record to modify, the user can click on "Update Incident" to enter another interface where the user can modify the relevant information such as date, time, and description.

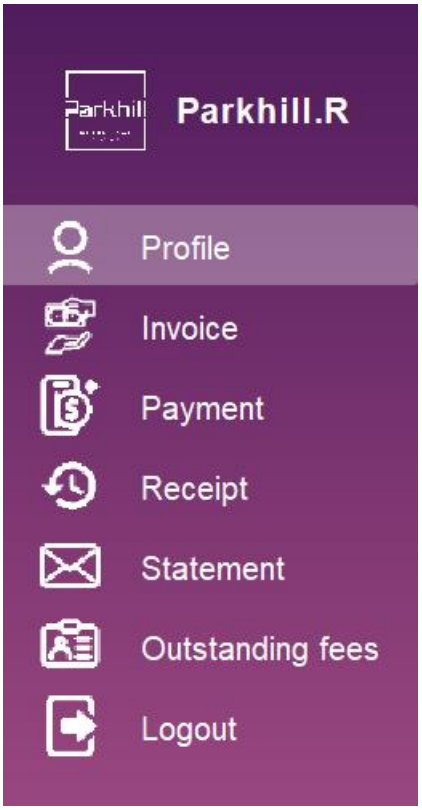
9.2.6.3 Check Incident



When users choose to view an incident report, they can click on the row, click on "check incident", and the page will jump to another interface, where users can get all relevant information, such as specific time, place, date, and description.

9.3 Other features

9.3.1 Menu

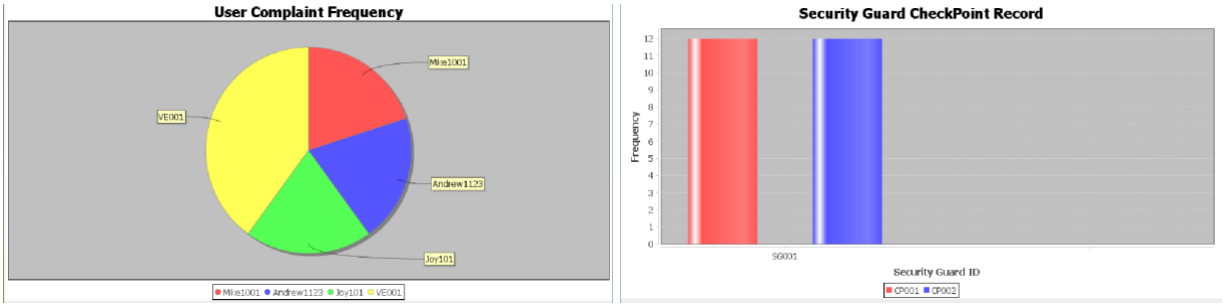


The user interface features a dynamic visual cue to assist users in navigating between sections. When a user selects a different section, the background color of that section will change to a crisp white, providing a clear and intuitive indication of the user's current location within the interface.

9.3.2 Loading

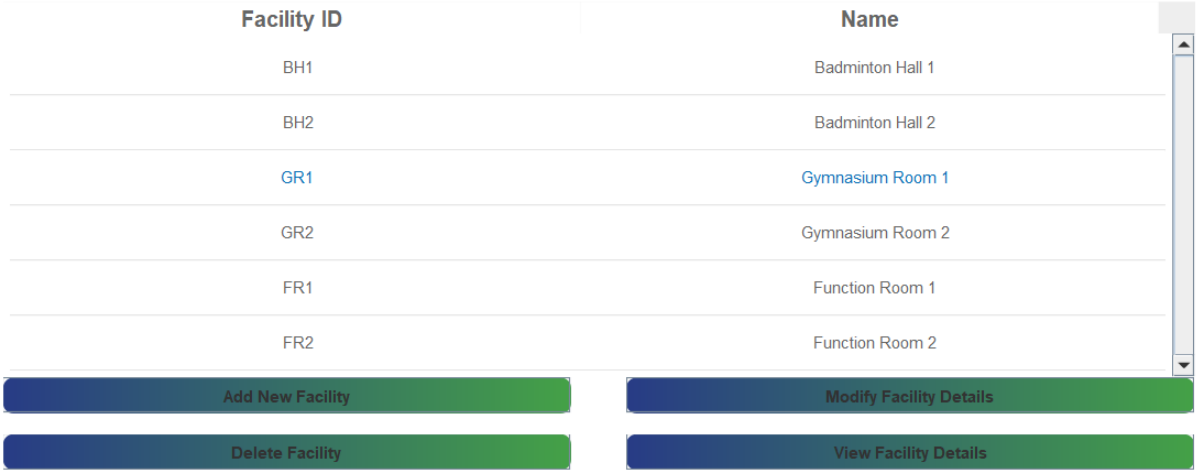
Upon initialization of the system, an elegant and informative loading bar will be presented to the user, displaying the precise percentage of program completion. This provides the user with the assurance that the program is currently in the process of starting up.

9.3.3 Report Graph



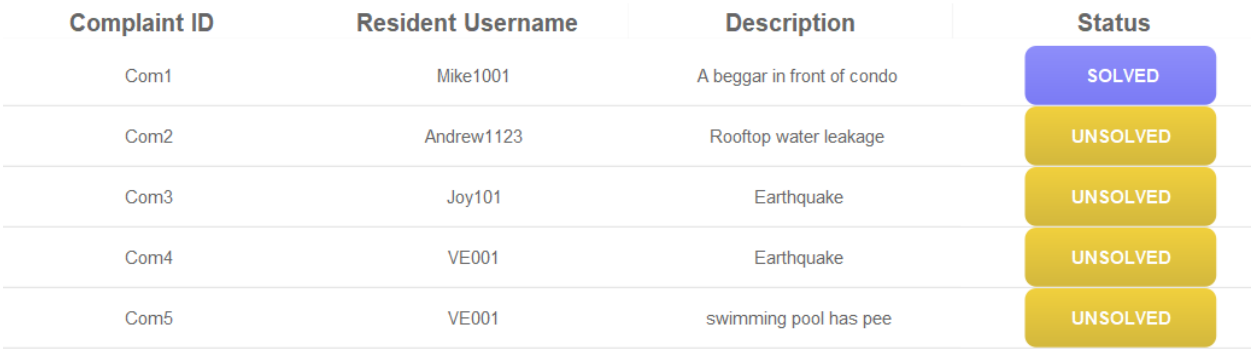
In the "job report" section of the building executive page, the information is dynamically presented in a visually appealing table format. This allows users to intuitively view and comprehend the data with ease.

9.3.4 Selected Row color



Upon clicking on a desired row, the interface intelligently highlights the row in a striking blue hue, ensuring that the user is fully cognizant of their selection.

9.3.5 Status color



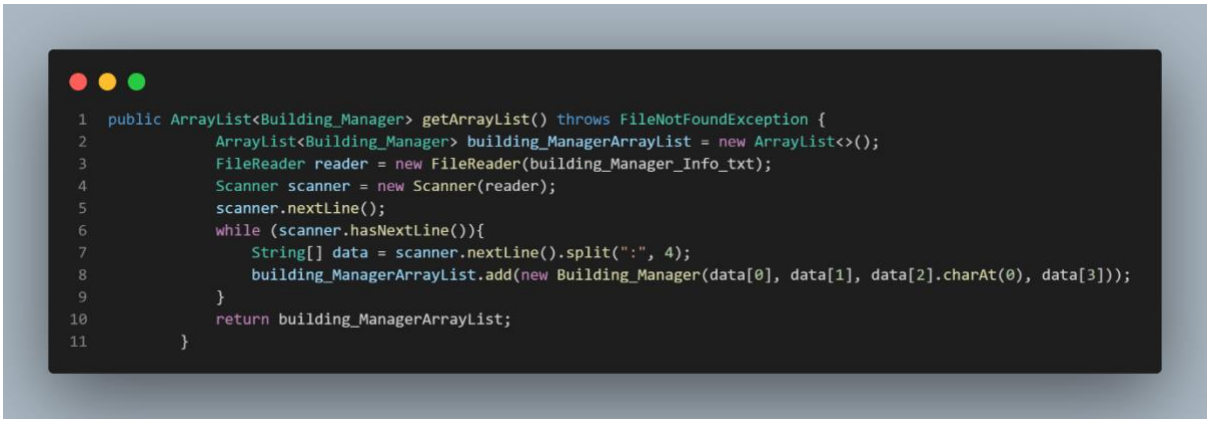
In some pages, when a request has not yet been completed, its status will be displayed as "solved" or "unsolved" along with different colors to indicate its status. For example, in the above example, the "solved" button appears in purple while the "unsolved" button appears in yellow. This provides a clear visual cue for the user to quickly understand the current status of the request.

10 System Code

10.1 Building Manager

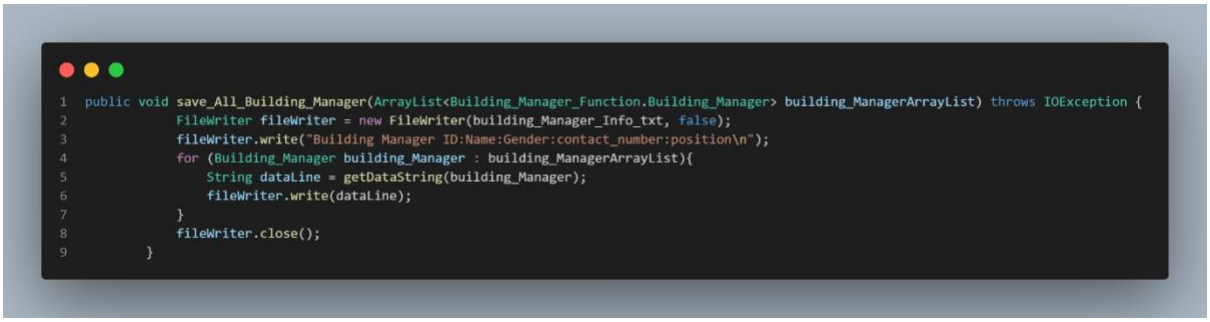
10.1.1 Building_Manager_Function

```java
public class Building_Manager_Function{
    public static class Building_Manager{
        private String buildingManagerID;
        private String name;
        private char gender;
        private String contact_Number;
        private String position = "Building Manager";
        private final File building_Manager_Info_txt = new File("src/Database/Building_Manager_Information.txt");
        public Building_Manager(){}
        public Building_Manager(String buildingManagerID, String name, char gender, String contact_Number) {
            this.buildingManagerID = buildingManagerID;
            this.name = name;
            this.gender = gender;
            this.contact_Number = contact_Number;
        }
```



```java
public ArrayList<Building_Manager> getArrayList() throws FileNotFoundException {
            ArrayList<Building_Manager> building_ManagerArrayList = new ArrayList<>();
            FileReader reader = new FileReader(building_Manager_Info_txt);
            Scanner scanner = new Scanner(reader);
            scanner.nextLine();
            while (scanner.hasNextLine()){
                String[] data = scanner.nextLine().split(":", 4);
                building_ManagerArrayList.add(new Building_Manager(data[0], data[1], data[2].charAt(0), data[3]));
            }
            return building_ManagerArrayList;
        }
```

10.1.2 Save all building manager



```java
public void save_All_Building_Manager(ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList) throws IOException {
            FileWriter fileWriter = new FileWriter(building_Manager_Info_txt, false);
            fileWriter.write("Building Manager ID:Name:Gender:contact_number:position\n");
            for (Building_Manager building_Manager : building_ManagerArrayList){
                String dataLine = getDataString(building_Manager);
                fileWriter.write(dataLine);
            }
            fileWriter.close();
        }
```

10.1.3 getarraylist

```java
public ArrayList<Building_Manager> getArrayList() throws FileNotFoundException {
            ArrayList<Building_Manager> building_ManagerArrayList = new ArrayList<>();
            FileReader reader = new FileReader(building_Manager_Info_txt);
            Scanner scanner = new Scanner(reader);
            scanner.nextLine();
            while (scanner.hasNextLine()){
                String[] data = scanner.nextLine().split(":", 4);
                building_ManagerArrayList.add(new Building_Manager(data[0], data[1], data[2].charAt(0), data[3]));
            }
            return building_ManagerArrayList;
        }
```

10.1.4 Search executive information

```java
public Building_Manager search_Executive_Information(String executiveID) throws FileNotFoundException {
            ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList = getArrayList();
            Building_Manager_Function.Building_Manager result = new Building_Manager_Function.Building_Manager();
            result.setBuildingManagerID("0");
            for (Building_Manager_Function.Building_Manager accountExecutive : building_ManagerArrayList) {
                if (accountExecutive.getBuildingManagerID().equals(executiveID)) {
                    result = accountExecutive;
                }
            }
            return result;
        }
```

10.1.5 Check building manager availability

```java
public boolean check_Building_Manager_Availability(String buildingManagerID) throws FileNotFoundException {
            boolean result = false;
            Building_Manager buildingManager = new Building_Manager();
            ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList = buildingManager.getArrayList();
            for (Building_Manager buildingManager1 : building_ManagerArrayList) {
                if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
                    result = true;
            }
            return result;
        }
```

10.1.6 Update building manager info

```java
public void update_Building_Manager_Info(Building_Manager buildingManager, String buildingManagerID) throws IOException {
            ArrayList<Building_Manager> buildingManagerArrayList = buildingManager.getArrayList();
            for (Building_Manager buildingManager1 : buildingManagerArrayList){
                if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
                    buildingManagerArrayList.remove(buildingManager1);
            }
            buildingManagerArrayList.add(buildingManager);
            buildingManager.save_All_Building_Manager(buildingManagerArrayList);
        }
```

10.1.7 Add account executive

```java
public void add_Account_Executive(Account_Executive_Function.Account_Executive accountExecutive) throws IOException {
            ArrayList<Account_Executive_Function.Account_Executive> account_executiveArrayList = accountExecutive.getArrayList();
            if (!(accountExecutive.check_Account_Executive_Availability(accountExecutive.getExecutiveID())))
                account_executiveArrayList.add(accountExecutive);
            accountExecutive.save_All_Account_Executive(account_executiveArrayList);
        }
```

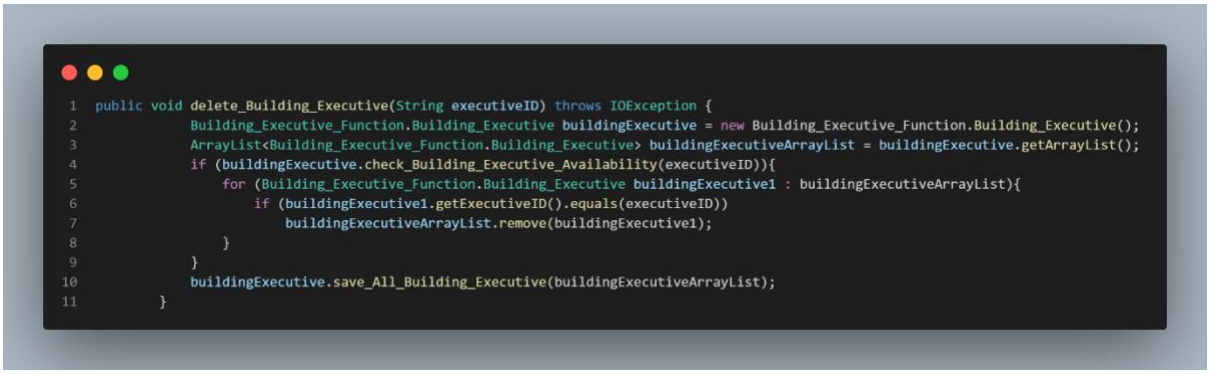
10.1.8 Delete account executive

```java
public void delete_Account_Executive(String executiveID) throws IOException {
            Account_Executive_Function.Account_Executive accountExecutive = new Account_Executive_Function.Account_Executive();
            ArrayList<Account_Executive_Function.Account_Executive> account_executiveArrayList = accountExecutive.getArrayList();
            if (accountExecutive.check_Account_Executive_Availability(executiveID)){
                for (Account_Executive_Function.Account_Executive accountExecutive1 : account_executiveArrayList){
                    if (accountExecutive1.getExecutiveID().equals(executiveID))
                        account_executiveArrayList.remove(accountExecutive1);
                }
            }
            accountExecutive.save_All_Account_Executive(account_executiveArrayList);
        }
```

10.1.9 Modify account executive

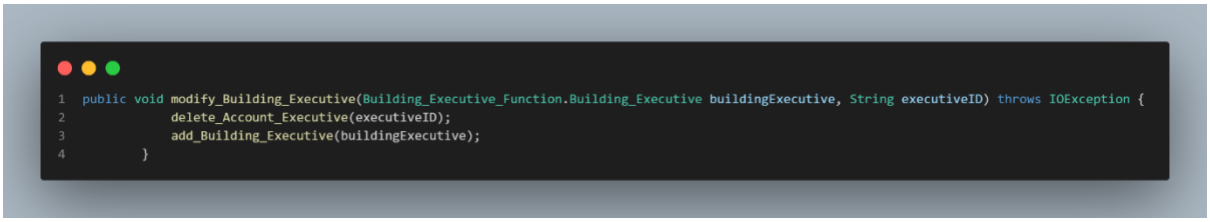
```java
public void modify_Account_Executive(Account_Executive_Function.Account_Executive accountExecutive, String executiveID) throws IOException {
            delete_Account_Executive(executiveID);
            add_Account_Executive(accountExecutive);
        }
```

10.1.10Delete building executive



```
public void delete_Building_Executive(String executiveID) throws IOException {
            Building_Executive_Function.Building_Executive buildingExecutive = new Building_Executive_Function.Building_Executive();
            ArrayList<Building_Executive_Function.Building_Executive> buildingExecutiveArrayList = buildingExecutive.getArrayList();
            if (buildingExecutive.check_Building_Executive_Availability(executiveID)){
                for (Building_Executive_Function.Building_Executive buildingExecutive1 : buildingExecutiveArrayList){
                    if (buildingExecutive1.getExecutiveID().equals(executiveID))
                        buildingExecutiveArrayList.remove(buildingExecutive1);
                }
            }
            buildingExecutive.save_All_Building_Executive(buildingExecutiveArrayList);
        }
```

10.1.11Modify building executive



```
public void modify_Building_Executive(Building_Executive_Function.Building_Executive buildingExecutive, String executiveID) throws IOException {
            delete_Account_Executive(executiveID);
            add_Building_Executive(buildingExecutive);
        }
```

10.1.12Add_admin_executive

```
public void add_Admin_Executive(Admin_Executive_Function.Admin_Executive adminExecutive) throws IOException {
            ArrayList<Admin_Executive_Function.Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
            if (!(adminExecutive.check_Admin_Executive_Availability(adminExecutive.getExecutiveID())))
                adminExecutiveArrayList.add(adminExecutive);
            adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList);
        }
```

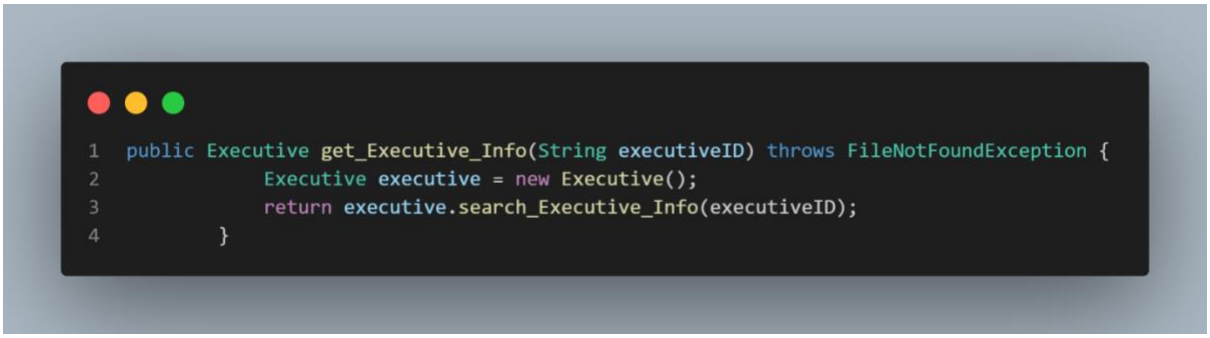
10.1.13Delete admin executive

```
public void delete_Admin_Executive(String executiveID) throws IOException {
            Admin_Executive_Function.Admin_Executive adminExecutive = new Admin_Executive_Function.Admin_Executive();
            ArrayList<Admin_Executive_Function.Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
            if (adminExecutive.check_Admin_Executive_Availability(executiveID)){
                for (Admin_Executive_Function.Admin_Executive adminExecutive1 : adminExecutiveArrayList){
                    if (adminExecutive1.getExecutiveID().equals(executiveID))
                        adminExecutiveArrayList.remove(adminExecutive1);
                }
            }
            adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList);
        }
```

10.1.14modify_Admin_Executive

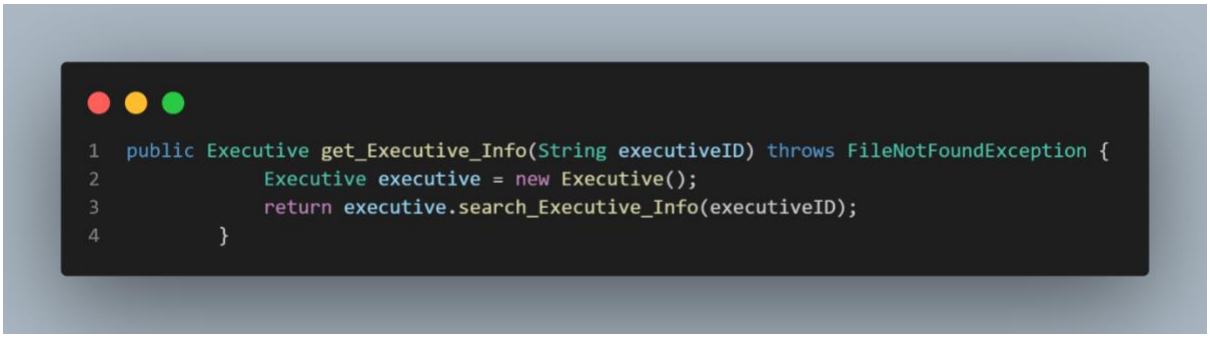
```
public void modify_Admin_Executive(Admin_Executive_Function.Admin_Executive adminExecutive, String executiveID) throws IOException {
            delete_Admin_Executive(executiveID);
            add_Admin_Executive(adminExecutive);
        }

```

10.1.15get_Executive_Info



```
public Executive get_Executive_Info(String executiveID) throws FileNotFoundException {
            Executive executive = new Executive();
            return executive.search_Executive_Info(executiveID);
        }
```

10.1.16get_Building_Manager_Info



```
public Executive get_Executive_Info(String executiveID) throws FileNotFoundException {
            Executive executive = new Executive();
            return executive.search_Executive_Info(executiveID);
        }
```

10.2 Account Executive

10.2.1 Account_Executive extends Executive

```java
public static class Account_Executive extends Executive {
        private String position = "Account Executive";
        private File account_Executive_Info_txt = new File("src/Database/Account_Executive_Information.txt");
        public Account_Executive(){}
        public Account_Executive(String account_Executive_ID, String name, char gender, String contact_Number){
            super();
            super.set_Info(account_Executive_ID, name, gender, contact_Number, position);
        }
```

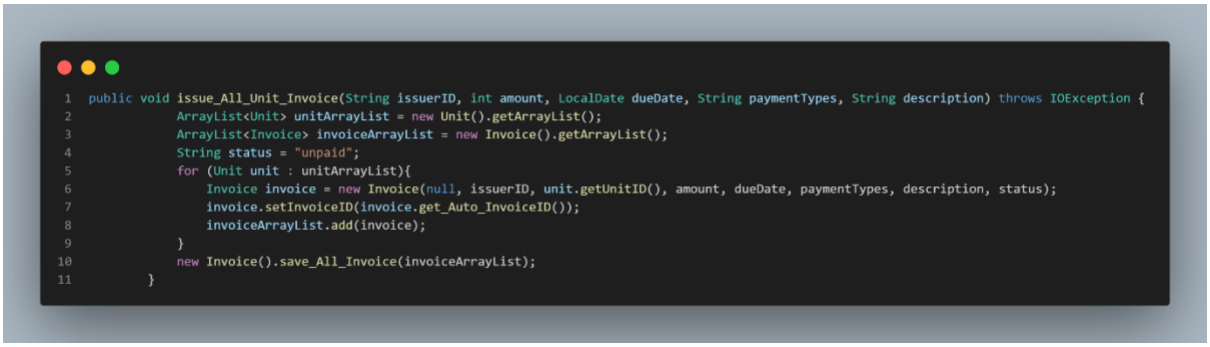
10.2.2 update_Account_Executive_Info

```java
public void update_Building_Manager_Info(Building_Manager buildingManager, String buildingManagerID) throws IOException {
            ArrayList<Building_Manager> buildingManagerArrayList = buildingManager.getArrayList();
            for (Building_Manager buildingManager1 : buildingManagerArrayList){
                if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
                    buildingManagerArrayList.remove(buildingManager1);
            }
            buildingManagerArrayList.add(buildingManager);
            buildingManager.save_All_Building_Manager(buildingManagerArrayList);
        }
```

10.2.3 get_Account_Executive_Info

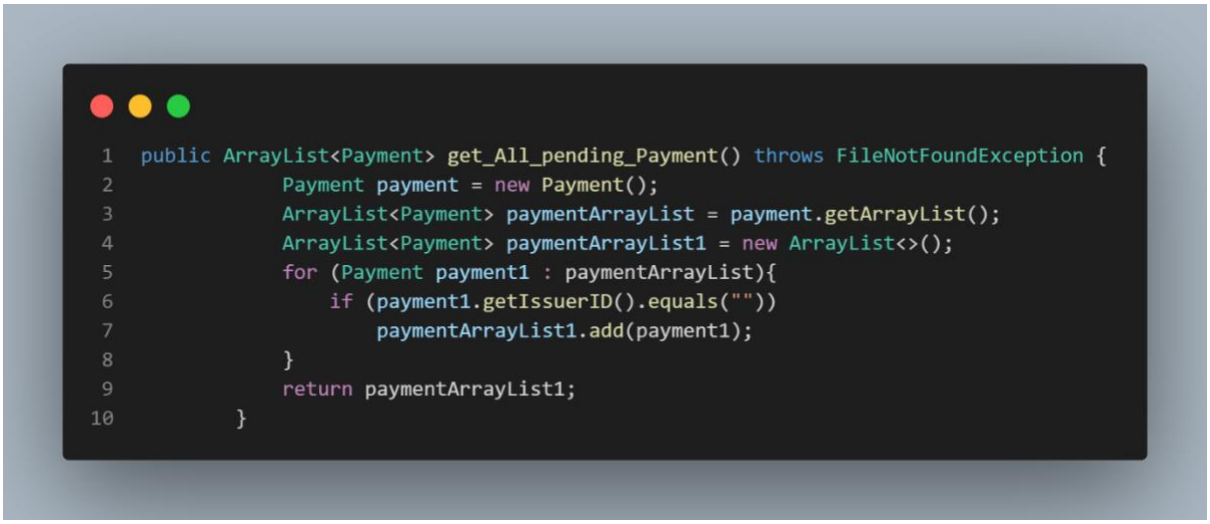
```java
public Account_Executive_Function.Account_Executive get_Account_Executive_Info(String executiveID) throws FileNotFoundException {
            ArrayList<Account_Executive_Function.Account_Executive> accountExecutiveArrayList = getArrayList();
            Account_Executive result = new Account_Executive();
            result.setExecutiveID("0");
            for (Account_Executive accountExecutive : accountExecutiveArrayList) {
                if (accountExecutive.getExecutiveID().equals(executiveID)) {
                    result = accountExecutive;
                }
            }
            return result;
        }
```

10.2.4 issue_All_Unit_Invoice



```
public void issue_All_Unit_Invoice(String issuerID, int amount, LocalDate dueDate, String paymentTypes, String description) throws IOException {
        ArrayList<Unit> unitArrayList = new Unit().getArrayList();
        ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
        String status = "unpaid";
        for (Unit unit : unitArrayList){
            Invoice invoice = new Invoice(null, issuerID, unit.getUnitID(), amount, dueDate, paymentTypes, description, status);
            invoice.setInvoiceID(invoice.get_Auto_InvoiceID());
            invoiceArrayList.add(invoice);
        }
        new Invoice().save_All_Invoice(invoiceArrayList);
    }
```

10.2.5 get_All_pending_Payment



```
public ArrayList<Payment> get_All_pending_Payment() throws FileNotFoundException {
        Payment payment = new Payment();
        ArrayList<Payment> paymentArrayList = payment.getArrayList();
        ArrayList<Payment> paymentArrayList1 = new ArrayList<>();
        for (Payment payment1 : paymentArrayList){
            if (payment1.getIssuerID().equals(""))
                paymentArrayList1.add(payment1);
        }
        return paymentArrayList1;
    }
```

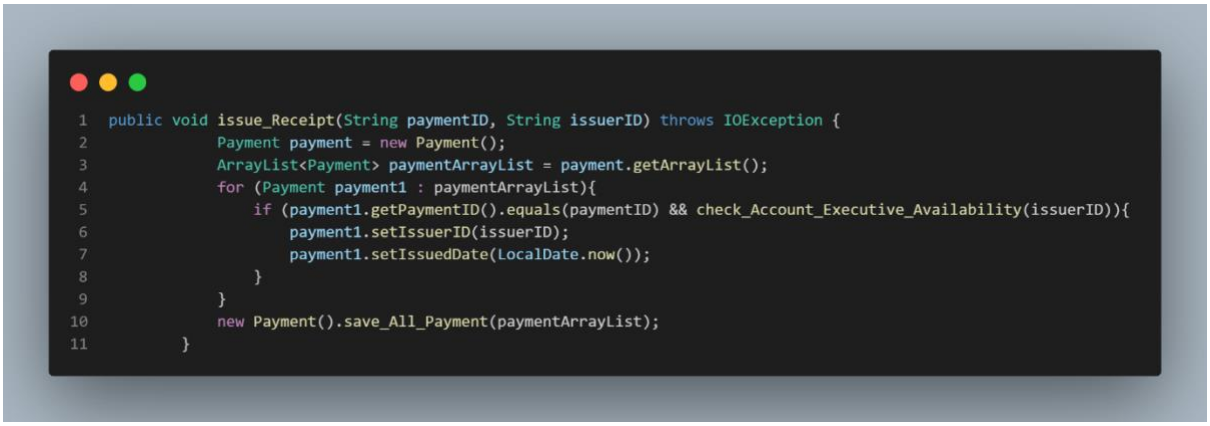
10.2.6 get_All_Unit_List

```java
public ArrayList<String> get_All_Unit_List() throws FileNotFoundException {
            ArrayList<Resident> residentArrayList = new Resident().getArrayList();
            ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
            ArrayList<String> all_Unit = new ArrayList<>();
            for (Resident resident : residentArrayList){
                all_Unit.add(resident.getUnitID());
            }
            for (Vendor vendor : vendorArrayList){
                all_Unit.add(vendor.getVendor_Unit());
            }
            for (int i = 0;i < all_Unit.size();i++){
                for (int j = 0;j < all_Unit.size();j++){
                    if (all_Unit.get(i).equals(all_Unit.get(j)))
                        all_Unit.remove(j);
                }
            }
            return all_Unit;
        }
```

10.2.7 issue_Unit_Invoice

```java
public void issue_Unit_Invoice(Invoice invoice) throws IOException {
            ArrayList<Invoice> invoiceArrayList = invoice.getArrayList();
            invoiceArrayList.add(invoice);
            invoice.save_All_Invoice(invoiceArrayList);
        }
```

10.2.8 issue_Receipt



```
public void issue_Receipt(String paymentID, String issuerID) throws IOException {
            Payment payment = new Payment();
            ArrayList<Payment> paymentArrayList = payment.getArrayList();
            for (Payment payment1 : paymentArrayList){
                if (payment1.getPaymentID().equals(paymentID) && check_Account_Executive_Availability(issuerID)){
                    payment1.setIssuerID(issuerID);
                    payment1.setIssuedDate(LocalDate.now());
                }
            }
            new Payment().save_All_Payment(paymentArrayList);
        }
```

10.2.9 issue_Statement

```
public void issue_Statement(Statement statement) throws IOException {
            ArrayList<Statement> statementArrayList = new Statement().getArrayList();
            statementArrayList.add(statement);
            new Statement().save_All_Statement(statementArrayList);
        }
```

10.2.10get_Unit_All_Invoice

```
public ArrayList<Invoice> get_Unit_All_Invoice(String unitID) throws FileNotFoundException {
            ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
            for (Invoice invoice : invoiceArrayList){
                if (!(invoice.getUnitID().equals(unitID)))
                    invoiceArrayList.remove(invoice);
            }
            return invoiceArrayList;
        }
```

10.2.11get_All_Unpaid_Invoice

```
public ArrayList<Invoice> get_All_Unpaid_Invoice() throws FileNotFoundException {
            ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
            ArrayList<Invoice> invoicesReturn = new ArrayList<>();
            for (Invoice invoice : invoiceArrayList){
                if (invoice.getStatus().equals("unpaid"))
                    invoicesReturn.add(invoice);
            }
            return invoicesReturn;
        }
```

10.2.12get_Unit_Unpaid_Amount

```
public int get_Unit_Unpaid_Amount(String unitID) throws FileNotFoundException {
            int total = 0;
            ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
            for (Invoice invoice : invoiceArrayList){
                if (invoice.getUnitID().equals(unitID) && invoice.getStatus().equals("unpaid"))
                    total += invoice.getAmount();
            }
            return total;
        }
```

10.3 Building Executive

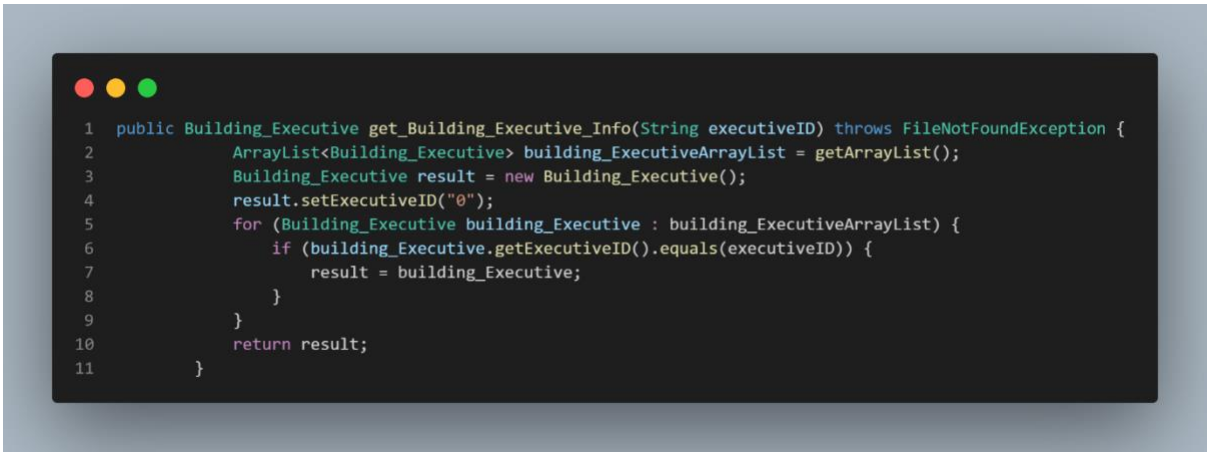
10.3.1 Account_Executive extends Executive

```
public static class Building_Executive extends Executive{
        private String position = "Building Executive";
        private File building_Executive_Info_txt = new File("src/Database/Building_Executive_Information.txt");
        public Building_Executive(){}
        public Building_Executive(String executiveID, String name, char gender, String contact_number){
            super();
            super.set_Info(executiveID, name, gender, contact_number, position);
        }
```

10.3.2 save_All_Building_Executive

```
public void save_All_Building_Executive(ArrayList<Building_Executive> building_ExecutiveArrayList) throws IOException {
        FileWriter fileWriter = new FileWriter(building_Executive_Info_txt, false);
        fileWriter.write("executiveID:Name:Gender:contact_number:position\n");
        for (Building_Executive building_Executive : building_ExecutiveArrayList){
            String[] data = {building_Executive.getExecutiveID(), building_Executive.getName(), Character.toString(building_Executive.getGender()), building_Executive.getContact_Number(), building_Executive.getPosition()};
            String dataLine = "";
            for (String dd : data){
                dataLine += dd + ":";
            }
            dataLine = dataLine.substring(0, dataLine.length() - 1);
            dataLine += '\n';
            fileWriter.write(dataLine);
        }
        fileWriter.close();
    }
```

10.3.3 get_Building_Executive_Info

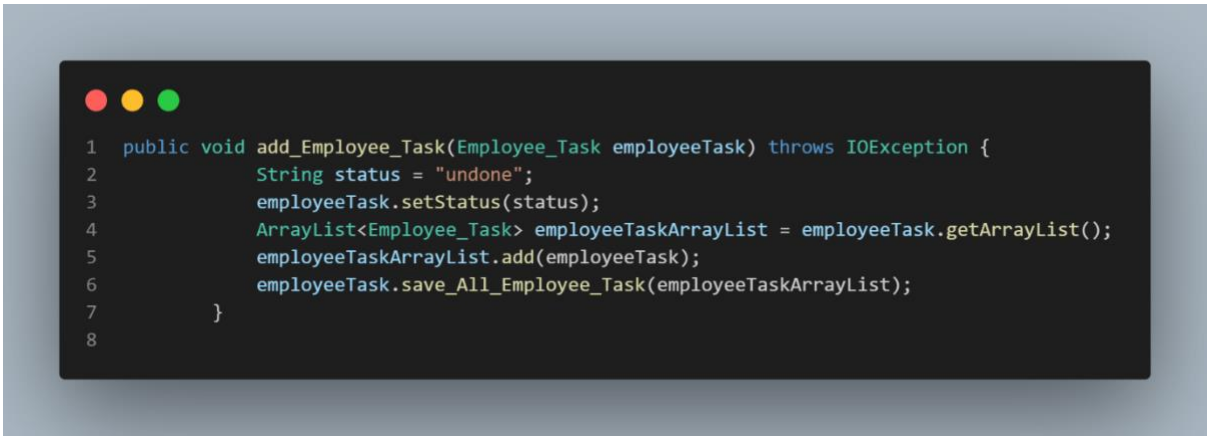


```
public Building_Executive get_Building_Executive_Info(String executiveID) throws FileNotFoundException {
            ArrayList<Building_Executive> building_ExecutiveArrayList = getArrayList();
            Building_Executive result = new Building_Executive();
            result.setExecutiveID("0");
            for (Building_Executive building_Executive : building_ExecutiveArrayList) {
                if (building_Executive.getExecutiveID().equals(executiveID)) {
                    result = building_Executive;
                }
            }
            return result;
        }
```

10.3.4 update_Building_Executive_Info

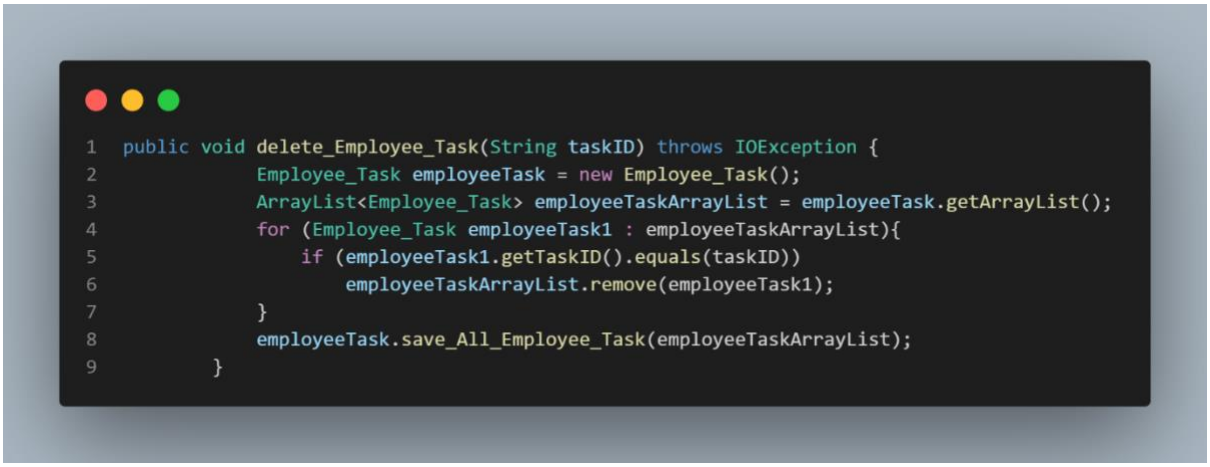
```
public void update_Building_Executive_Info(Building_Executive buildingExecutive, String executiveID) throws IOException {
            ArrayList<Building_Executive> buildingExecutiveArrayList = buildingExecutive.getArrayList();
            for (Building_Executive buildingExecutive1 : buildingExecutiveArrayList){
                if (buildingExecutive1.getExecutiveID().equals(executiveID))
                    buildingExecutiveArrayList.remove(buildingExecutive);
            }
            buildingExecutiveArrayList.add(buildingExecutive);
            buildingExecutive.save_All_Building_Executive(buildingExecutiveArrayList);
        }
```

10.3.5 add_Employee_Task



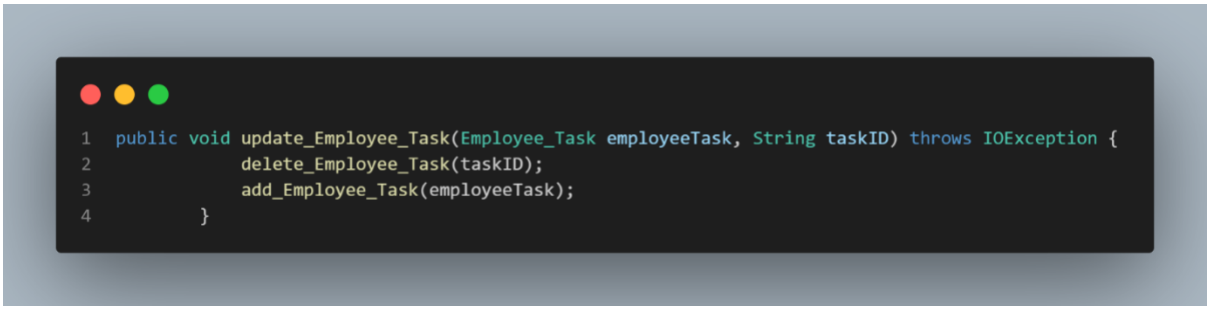
```
public void add_Employee_Task(Employee_Task employeeTask) throws IOException {
            String status = "undone";
            employeeTask.setStatus(status);
            ArrayList<Employee_Task> employeeTaskArrayList = employeeTask.getArrayList();
            employeeTaskArrayList.add(employeeTask);
            employeeTask.save_All_Employee_Task(employeeTaskArrayList);
        }

```

10.3.6 delete_Employee_Task



```
public void delete_Employee_Task(String taskID) throws IOException {
            Employee_Task employeeTask = new Employee_Task();
            ArrayList<Employee_Task> employeeTaskArrayList = employeeTask.getArrayList();
            for (Employee_Task employeeTask1 : employeeTaskArrayList){
                if (employeeTask1.getTaskID().equals(taskID))
                    employeeTaskArrayList.remove(employeeTask1);
            }
            employeeTask.save_All_Employee_Task(employeeTaskArrayList);
        }
```

10.3.7 update_Employee_Task



```
public void update_Employee_Task(Employee_Task employeeTask, String taskID) throws IOException {
            delete_Employee_Task(taskID);
            add_Employee_Task(employeeTask);
        }
```

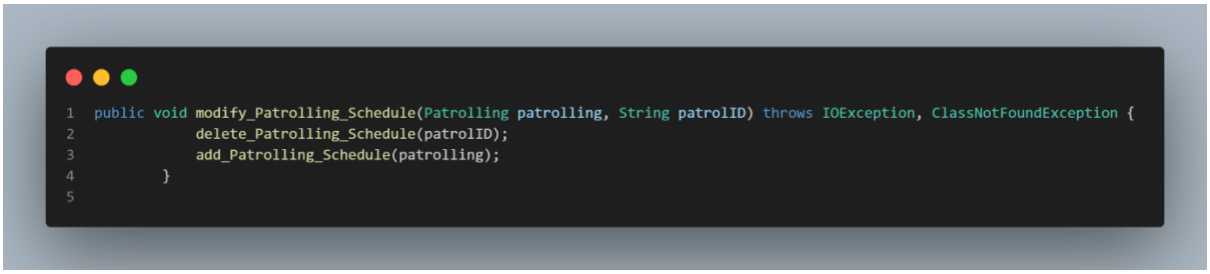
10.3.8 view_All_Complaint

```
public ArrayList<Complaint> view_All_Complaint() throws FileNotFoundException {
            ArrayList<Complaint> complaintArrayList = new Complaint().getArrayList();
            return complaintArrayList;
        }
```

10.3.9 add_Patrolling_Schedule

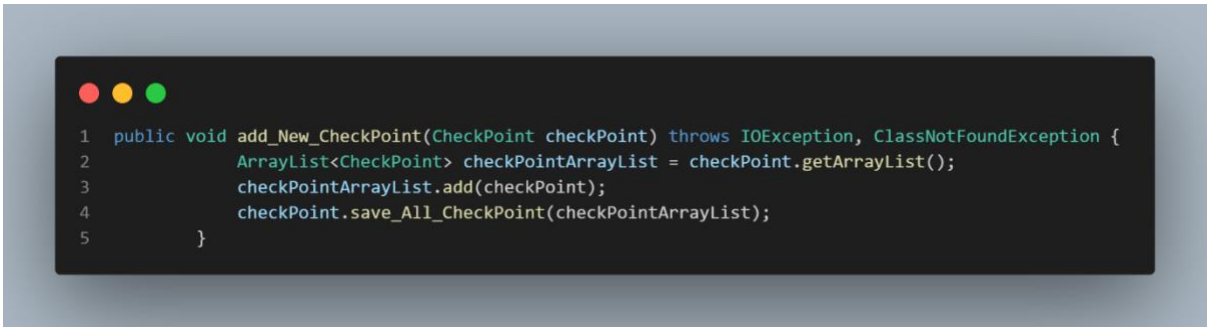
```
public void add_Patrolling_Schedule(Patrolling patrolling) throws IOException, ClassNotFoundException {
            SecurityGuard securityGuard = new SecurityGuard();
            boolean check = securityGuard.check_SecurityGuard_Availability(patrolling.getEmployeeID());
            if (check){
                ArrayList<Patrolling> patrollingArrayList = patrolling.getArrayList();
                patrollingArrayList.add(patrolling);
                patrolling.save_All_Patrolling(patrollingArrayList);
            }
        }
```

10.3.10delete_Patrolling_Schedule



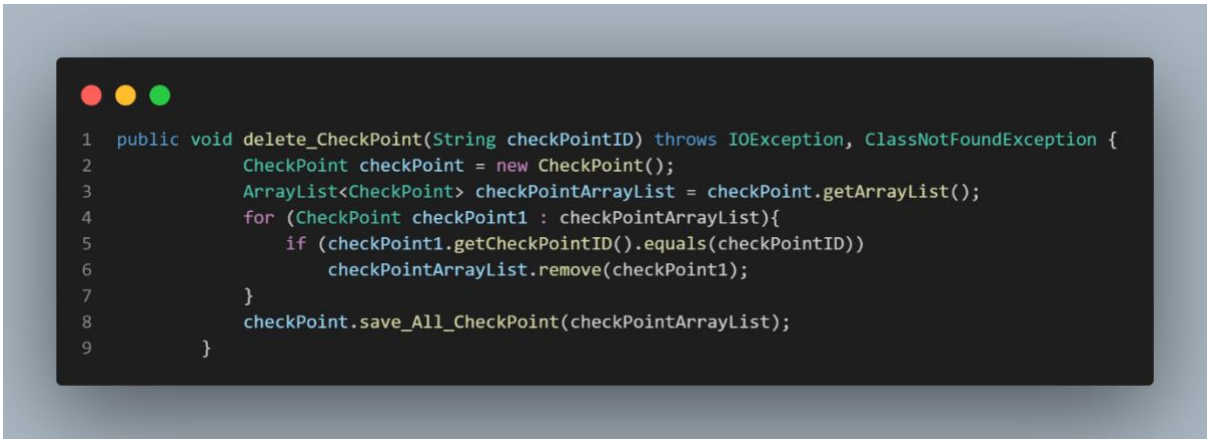
```
public void modify_Patrolling_Schedule(Patrolling patrolling, String patrolID) throws IOException, ClassNotFoundException {
            delete_Patrolling_Schedule(patrolID);
            add_Patrolling_Schedule(patrolling);
        }

```

10.3.11add_New_CheckPoint



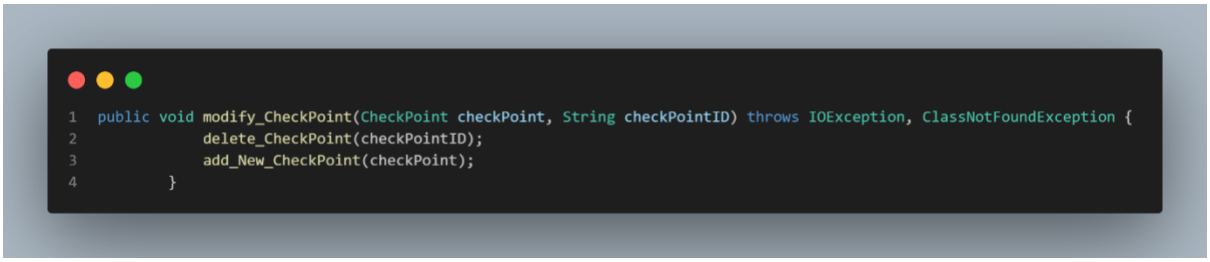
```
public void add_New_CheckPoint(CheckPoint checkPoint) throws IOException, ClassNotFoundException {
            ArrayList<CheckPoint> checkPointArrayList = checkPoint.getArrayList();
            checkPointArrayList.add(checkPoint);
            checkPoint.save_All_CheckPoint(checkPointArrayList);
        }
```

10.3.12delete_CheckPoint



```
public void delete_CheckPoint(String checkPointID) throws IOException, ClassNotFoundException {
            CheckPoint checkPoint = new CheckPoint();
            ArrayList<CheckPoint> checkPointArrayList = checkPoint.getArrayList();
            for (CheckPoint checkPoint1 : checkPointArrayList){
                if (checkPoint1.getCheckPointID().equals(checkPointID))
                    checkPointArrayList.remove(checkPoint1);
            }
            checkPoint.save_All_CheckPoint(checkPointArrayList);
        }
```

10.3.13modify_CheckPoint



```
public void modify_CheckPoint(CheckPoint checkPoint, String checkPointID) throws IOException, ClassNotFoundException {
            delete_CheckPoint(checkPointID);
            add_New_CheckPoint(checkPoint);
        }
```

10.4 Admin Executive

10.4.1 Admin_Executive extends Executive

```java
public static class Admin_Executive extends Executive {
        private String position = "Admin Executive";
        private File unit_Information_txt = new File("src/Database/Unit_Information.txt");
        private File admin_Executive_Info_txt = new File("src/Database/Admin_Executive_Information.txt");

        public Admin_Executive(){}
        public Admin_Executive(String admin_Executive_ID, String name, char gender, String contact_Number){
            super();
            super.set_Info(admin_Executive_ID, name, gender, contact_Number, position);
        }
```

10.4.2 update_Admin_Executive_Info

```java
public void update_Admin_Executive_Info(Admin_Executive adminExecutive, String executiveID) throws IOException {
            ArrayList<Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
            ArrayList<Admin_Executive> adminExecutiveArrayList1 = new ArrayList<>();
            for (Admin_Executive adminExecutive1 : adminExecutiveArrayList){
                if (!(adminExecutive1.getExecutiveID().equals(executiveID)))
                    adminExecutiveArrayList1.add(adminExecutive1);
            }
            adminExecutiveArrayList1.add(adminExecutive);
            adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList1);
        }
```

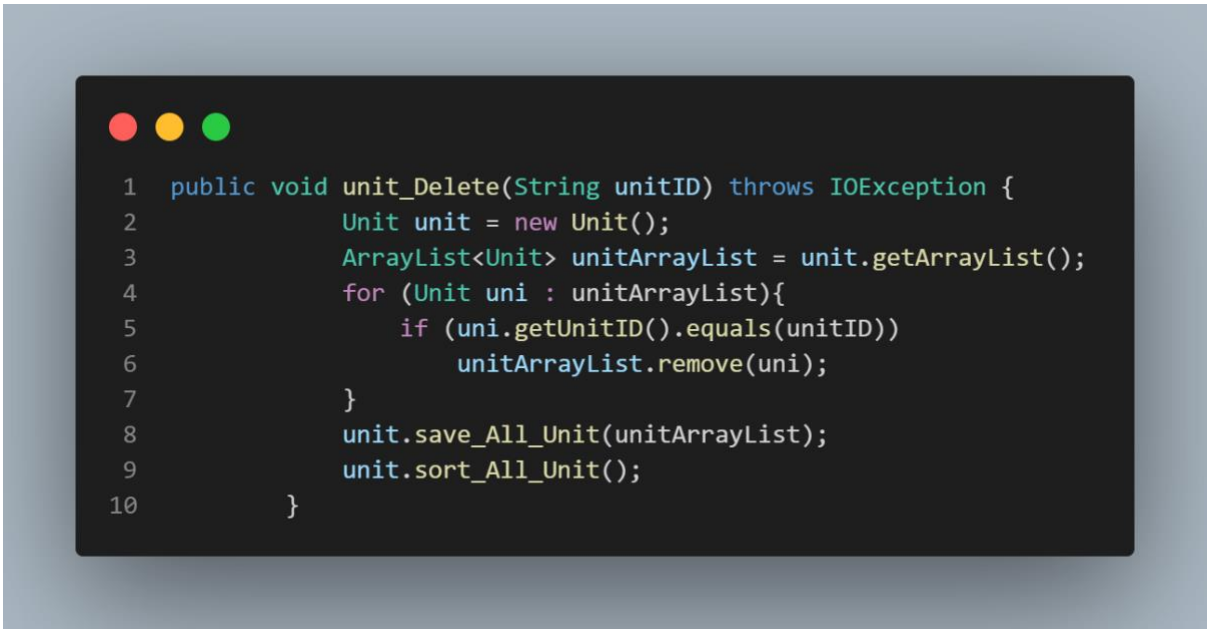
10.4.3 unit_Add

```java
public void unit_Add(Unit unit) throws IOException {
            ArrayList<Unit> unitArrayList = unit.getArrayList();
            unitArrayList.add(unit);
            unit.save_All_Unit(unitArrayList);
        }
```

10.4.4 unit_Modify

```
public void unit_Modify(Unit unit, String unitID) throws IOException {
        if (unit.check_Unit_Availability(unitID)){
            unit_Delete(unitID);
            unit_Add(unit);
            unit.sort_All_Unit();
            JOptionPane.showMessageDialog(null, "Unit Details Modification success", "Modifying success added", JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

10.4.5 unit_Delete



```
public void unit_Delete(String unitID) throws IOException {
            Unit unit = new Unit();
            ArrayList<Unit> unitArrayList = unit.getArrayList();
            for (Unit uni : unitArrayList){
                if (uni.getUnitID().equals(unitID))
                    unitArrayList.remove(uni);
            }
            unit.save_All_Unit(unitArrayList);
            unit.sort_All_Unit();
        }
```

10.4.6 resident_Add

```
public void resident_Add(Resident resident) throws IOException {
            ArrayList<Resident> residentArrayList = resident.getArrayList();
            residentArrayList.add(resident);
            resident.save_All_Resident(residentArrayList);
        }
```

10.4.7 resident_Modify

```
public void resident_Modify(Resident resident, String resident_Username) throws IOException {
        String dataLine = resident.getDataString(resident);
        if (resident.check_Resident_Availability(resident_Username)){
            resident_Delete(resident_Username);
            resident_Add(resident);
            JOptionPane.showMessageDialog(null, "Resident Information modifying success", "Resident information modifying success", JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

10.4.8 resident_Delete

```
public void resident_Delete(String resident_Username) throws IOException {
            Resident resident = new Resident();
            ArrayList<Resident> residentArrayList = resident.getArrayList();
            for (Resident re : residentArrayList){
                if (re.getResident_Username().equals(resident_Username))
                    residentArrayList.remove(re);
            }
            resident.save_All_Resident(residentArrayList);
        }
```

10.4.9 complaint_Add

```
public void complaint_Add(Complaint complaint) throws IOException {
            ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
            complaintArrayList.add(complaint);
            complaint.save_All_Complaint(complaintArrayList);
        }
```

10.4.10complaint_Update

```
public void complaint_Update(Complaint complaint, String complaintID) throws IOException {
            if (complaint.check_Complaint_Availability(complaintID)){
                complaint_Delete(complaintID);
                complaint_Add(complaint);
            }
        }
```

10.4.11complaint_Delete

```
public void complaint_Delete(String complaintID) throws IOException {
            Complaint complaint = new Complaint();
            ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
            for (Complaint com : complaintArrayList){
                if (com.getComplaintID().equals(complaintID))
                    complaintArrayList.remove(com);
            }
            complaint.save_All_Complaint(complaintArrayList);
        }
```

10.4.12add_Employee

```
public void add_Employee(Employee employee) throws IOException, ClassNotFoundException {
        String condition = employee.getEmployeeID().substring(0, 2);
        if (condition.equals("SG")){
            SecurityGuard securityGuard = new SecurityGuard();
            securityGuard.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
            ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
            securityGuardArrayList.add(securityGuard);
            securityGuard.save_All_SecurityGuard(securityGuardArrayList);
        }else if (condition.equals("CN")){
            Cleaner cleaner = new Cleaner();
            cleaner.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
            ArrayList<Cleaner> cleanerArrayList = cleaner.getArrayList();
            cleanerArrayList.add(cleaner);
            cleaner.save_All_Cleaner(cleanerArrayList);
        } else if (condition.equals("TN")){
            Technician technician = new Technician();
            technician.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
            ArrayList<Technician> technicianArrayList = technician.getArrayList();
            technicianArrayList.add(technician);
            technician.save_All_Technician(technicianArrayList);
        }
    }
```

10.4.13view_All_Employee

```java
public ArrayList<Employee> view_All_Employee() throws IOException, ClassNotFoundException {
            ArrayList<Employee> employeeArrayList = new ArrayList<>();
            ArrayList<Technician> technicianArrayList = new Technician().getArrayList();
            ArrayList<Cleaner> cleanerArrayList = new Cleaner().getArrayList();
            ArrayList<SecurityGuard>securityGuardArrayList = new SecurityGuard().getArrayList();
            employeeArrayList.addAll(technicianArrayList);
            employeeArrayList.addAll(cleanerArrayList);
            employeeArrayList.addAll(securityGuardArrayList);
            return employeeArrayList;
        }
```

10.4.14delete_Employee

```java
public void delete_Employee(String employeeID) throws IOException, ClassNotFoundException {
            String condition = employeeID.substring(0, 2);
            if (condition.equals("SG")){
                SecurityGuard securityGuard = new SecurityGuard();
                ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
                for (SecurityGuard securityGuard1 : securityGuardArrayList){
                    if (securityGuard1.getEmployeeID().equals(employeeID))
                        securityGuardArrayList.remove(securityGuard1);
                }
                securityGuard.save_All_SecurityGuard(securityGuardArrayList);
            }else if (condition.equals("CN")){
                Cleaner cleaner = new Cleaner();
                ArrayList<Cleaner> cleanerArrayList = cleaner.getArrayList();
                for (Cleaner cleaner1 : cleanerArrayList){
                    if (cleaner1.getEmployeeID().equals(employeeID))
                        cleanerArrayList.remove(cleaner1);
                }
                cleaner.save_All_Cleaner(cleanerArrayList);
            } else if (condition.equals("TN")){
                Technician technician = new Technician();
                ArrayList<Technician> technicianArrayList = technician.getArrayList();
                for (Technician technician1 : technicianArrayList){
                    if (technician1.getEmployeeID().equals(employeeID))
                        technicianArrayList.remove(technician1);
                }
                technician.save_All_Technician(technicianArrayList);
            }
        }
```

10.4.15update_Employee

```
public void update_Employee(Employee employee, String employeeID) throws IOException, ClassNotFoundException {
            delete_Employee(employeeID);
            add_Employee(employee);
        }
```

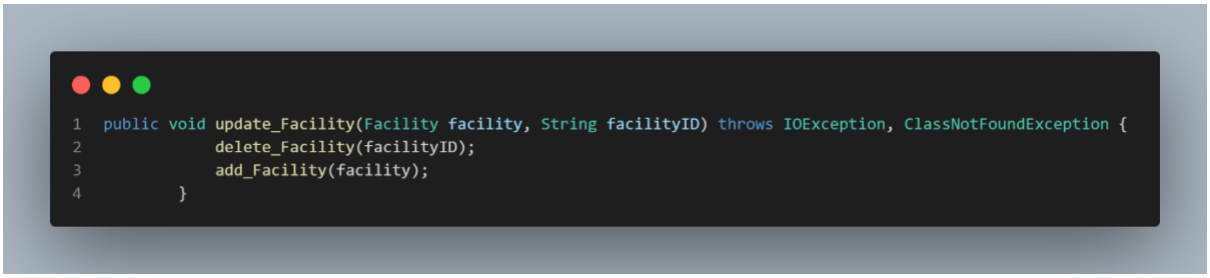
10.4.16add_Facility

```
public void add_Facility(Facility facility) throws IOException, ClassNotFoundException {
            ArrayList<Facility> facilityArrayList = facility.getArrayList();
            if (!facility.check_Facility_Availability(facility.getFacilityID()))
                facilityArrayList.add(facility);
            facility.save_All_Facility(facilityArrayList);
        }
```

10.4.17delete_Facility

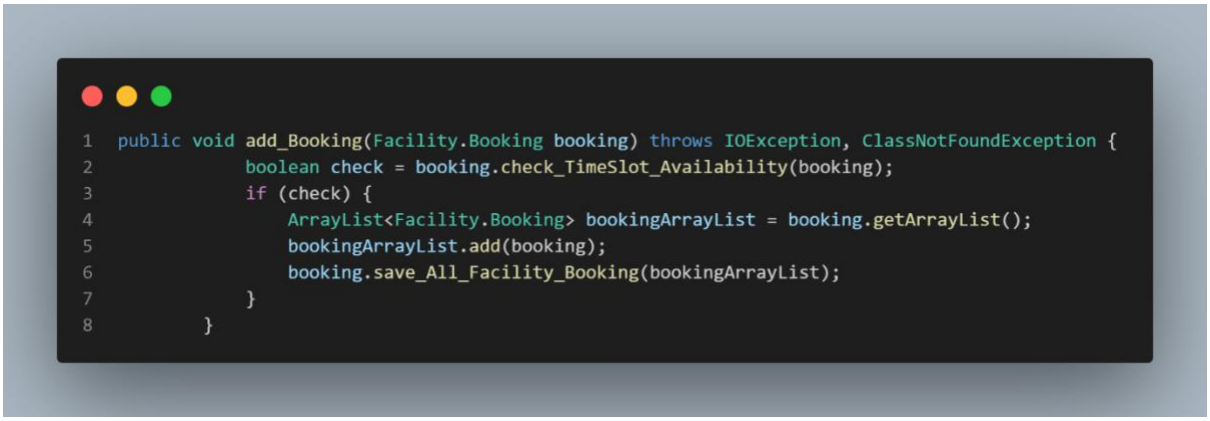
```
public void delete_Facility(String facilityID) throws IOException, ClassNotFoundException {
            Facility facility = new Facility();
            ArrayList<Facility> facilityArrayList = facility.getArrayList();
            for (Facility facility1 : facilityArrayList){
                if (facility1.getFacilityID().equals(facilityID))
                    facilityArrayList.remove(facility1);
            }
            facility.save_All_Facility(facilityArrayList);
        }
```

10.4.18update_Facility



```
public void update_Facility(Facility facility, String facilityID) throws IOException, ClassNotFoundException {
            delete_Facility(facilityID);
            add_Facility(facility);
        }
```

10.4.19add_Booking

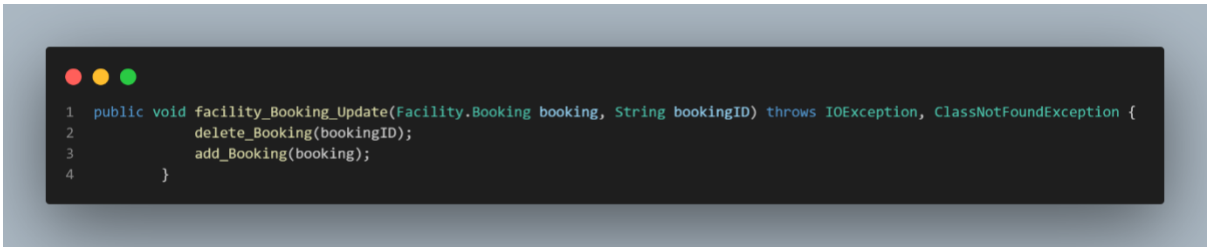


```
public void add_Booking(Facility.Booking booking) throws IOException, ClassNotFoundException {
            boolean check = booking.check_TimeSlot_Availability(booking);
            if (check) {
                ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();
                bookingArrayList.add(booking);
                booking.save_All_Facility_Booking(bookingArrayList);
            }
        }
```

10.4.20delete_Booking

```
public void delete_Booking(String bookingID) throws IOException, ClassNotFoundException {
            Facility.Booking booking = new Facility.Booking();
            ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();
            for (Facility.Booking booking1 : bookingArrayList){
                if (booking1.getBookingID().equals(bookingID))
                    bookingArrayList.remove(booking1);
            }
            booking.save_All_Facility_Booking(bookingArrayList);
        }
```

10.4.21facility_Booking_Update

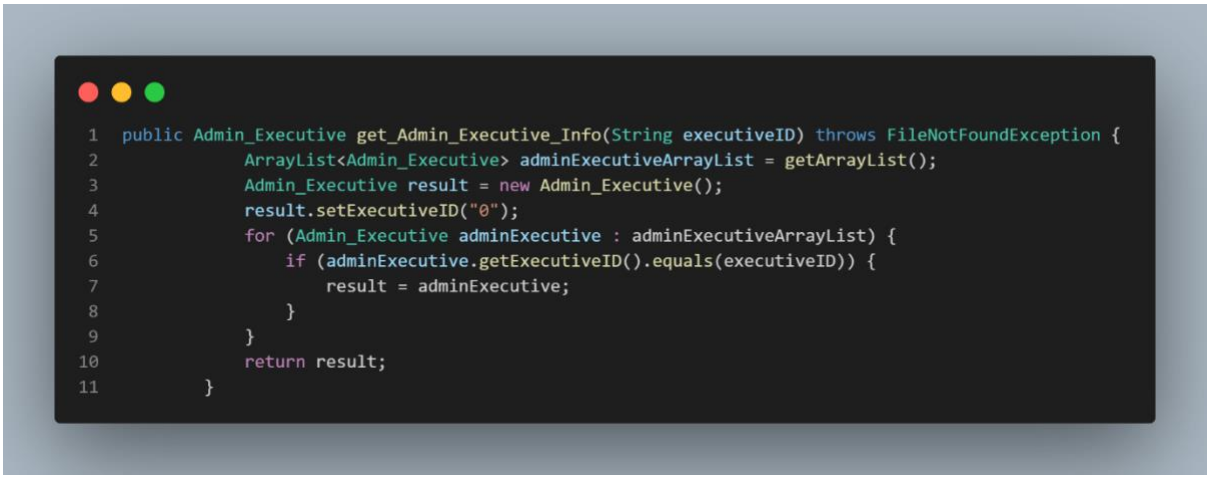


```
public void facility_Booking_Update(Facility.Booking booking, String bookingID) throws IOException, ClassNotFoundException {
            delete_Booking(bookingID);
            add_Booking(booking);
        }
```

10.4.22save_All_Admin_Executive

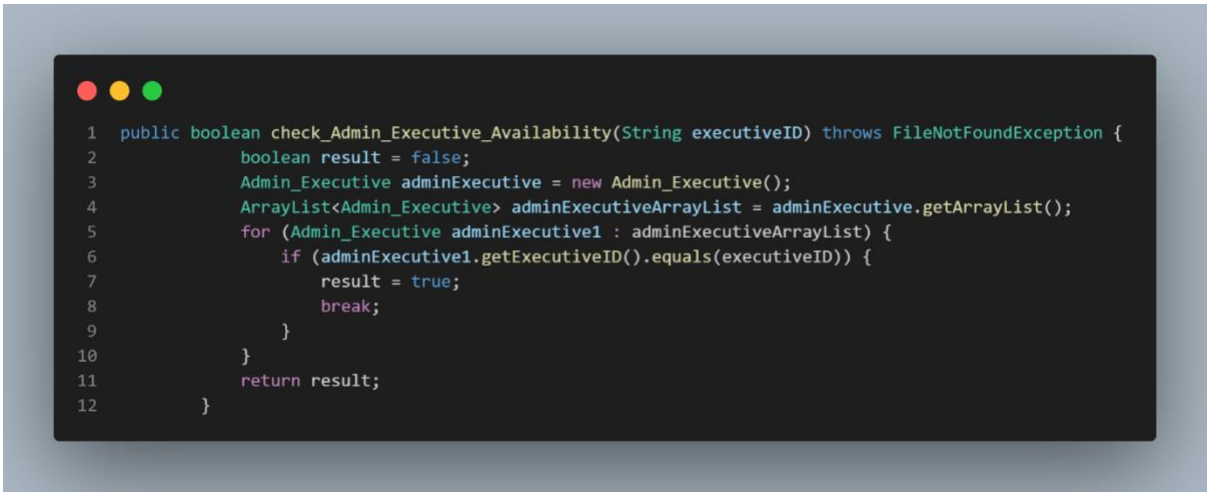
```
public void save_All_Admin_Executive(ArrayList<Admin_Executive> adminExecutiveArrayList) throws IOException {
        FileWriter fileWriter = new FileWriter(admin_Executive_Info_txt, false);
        fileWriter.write("executiveID:Name:Gender:contact_number:position\n");
        for (Admin_Executive adminExecutive : adminExecutiveArrayList){
            String[] data = {adminExecutive.getExecutiveID(), adminExecutive.getName(), Character.toString(adminExecutive.getGender()), adminExecutive.getContact_Number(), adminExecutive.getPosition()};
            String dataLine = "";
            for (String dd : data){
                dataLine += dd + ":";
            }
            dataLine = dataLine.substring(0, dataLine.length() - 1);
            dataLine += '\n';
            fileWriter.write(dataLine);
        }
        fileWriter.close();
    }
```

10.4.23get_Admin_Executive_Info



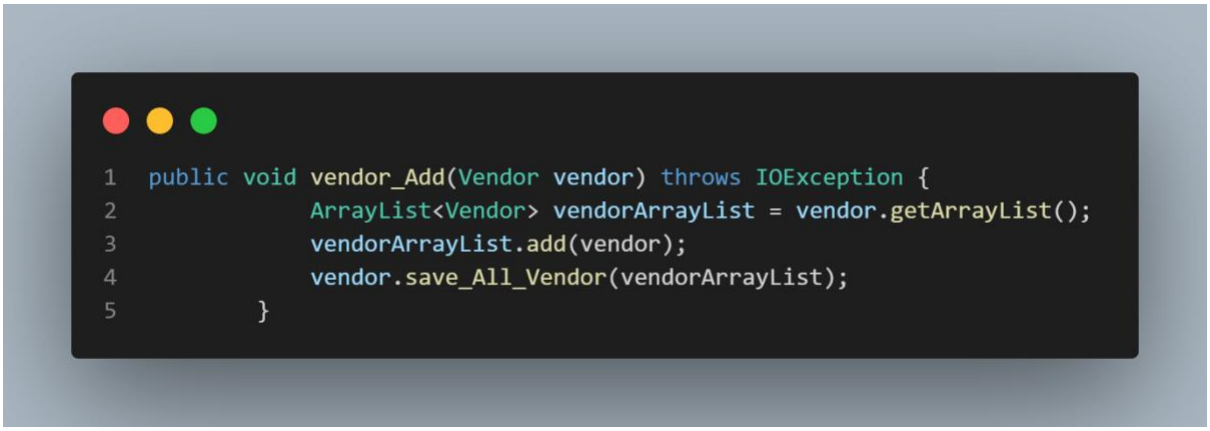
```
public Admin_Executive get_Admin_Executive_Info(String executiveID) throws FileNotFoundException {
            ArrayList<Admin_Executive> adminExecutiveArrayList = getArrayList();
            Admin_Executive result = new Admin_Executive();
            result.setExecutiveID("0");
            for (Admin_Executive adminExecutive : adminExecutiveArrayList) {
                if (adminExecutive.getExecutiveID().equals(executiveID)) {
                    result = adminExecutive;
                }
            }
            return result;
        }
```

10.4.24check_Admin_Executive_Availability



```
public boolean check_Admin_Executive_Availability(String executiveID) throws FileNotFoundException {
            boolean result = false;
            Admin_Executive adminExecutive = new Admin_Executive();
            ArrayList<Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
            for (Admin_Executive adminExecutive1 : adminExecutiveArrayList) {
                if (adminExecutive1.getExecutiveID().equals(executiveID)) {
                    result = true;
                    break;
                }
            }
            return result;
        }
```

10.4.25vendor_Add



```
public void vendor_Add(Vendor vendor) throws IOException {
            ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
            vendorArrayList.add(vendor);
            vendor.save_All_Vendor(vendorArrayList);
        }
```

10.4.26vendor_Modify

```
public void vendor_Modify(Vendor vendor, String vendor_Username) throws IOException {
        String dataLine = vendor.getDataString(vendor);
        if (vendor.check_Vendor_Availability(vendor_Username)){
            vendor_Delete(vendor_Username);
            vendor_Add(vendor);
            JOptionPane.showMessageDialog(null, "Vendor Information modifying success", "Vendor information modifying success", JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

10.4.27vendor_Delete

```java
public void vendor_Delete(String vendor_Username) throws IOException {
            Vendor vendor = new Vendor();
            ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
            for (Vendor vendor1 : vendorArrayList){
                if (vendor1.getVendor_Username().equals(vendor_Username))
                    vendorArrayList.remove(vendor1);
            }
            vendor.save_All_Vendor(vendorArrayList);
        }
```

10.5 Resident

10.5.1 Resident

```java
public Resident(String resident_Username, String name, char gender, String contact_Number, String unitID, int payment) {
        this.resident_Username = resident_Username;
        this.name = name;
        this.gender = gender;
        this.contact_Number = contact_Number;
        this.unitID = unitID;
        this.payment = payment;
    }
```

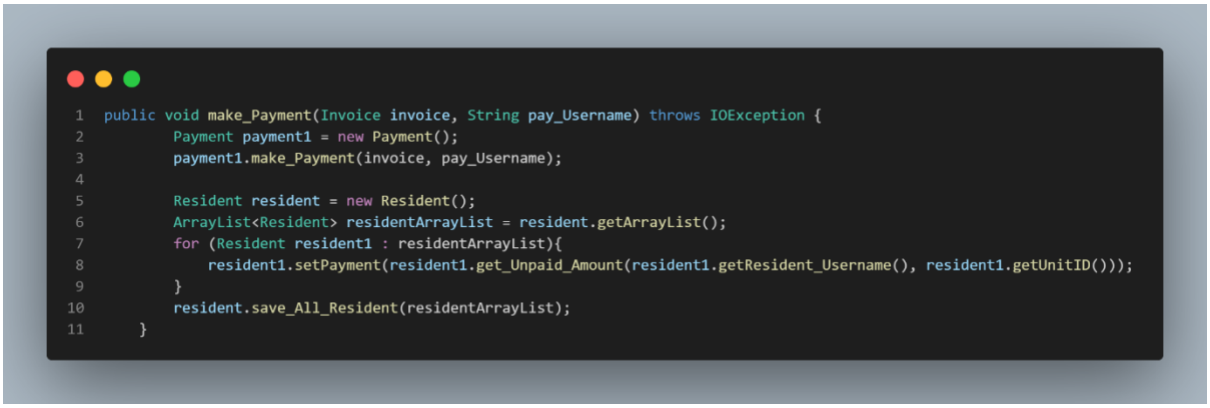
10.5.2 get_Resident_Info

```java
public Resident get_Resident_Info(String resident_Username) throws FileNotFoundException {
        Resident resident = new Resident();
        resident.setResident_Username("0");
        ArrayList<Resident> residentArrayList = resident.getArrayList();
        for (Resident resident1 : residentArrayList){
            if (resident1.getResident_Username().equals(resident_Username))
                return resident1;
        }
        return resident;
    }
```

10.5.3 update_Resident_Info

```java
public void update_Resident_Info(Resident resident, String resident_Username) throws IOException {
        ArrayList<Resident> residentArrayList = resident.getArrayList();
        ArrayList<Resident> residentArrayList1 = new ArrayList<>();
        for (Resident resident1 : residentArrayList){
            if (!(resident1.getResident_Username().equals(resident_Username)))
                residentArrayList1.add(resident1);
        }
        residentArrayList1.add(resident);
        resident.save_All_Resident(residentArrayList1);
    }

```

10.5.4 make_Payment

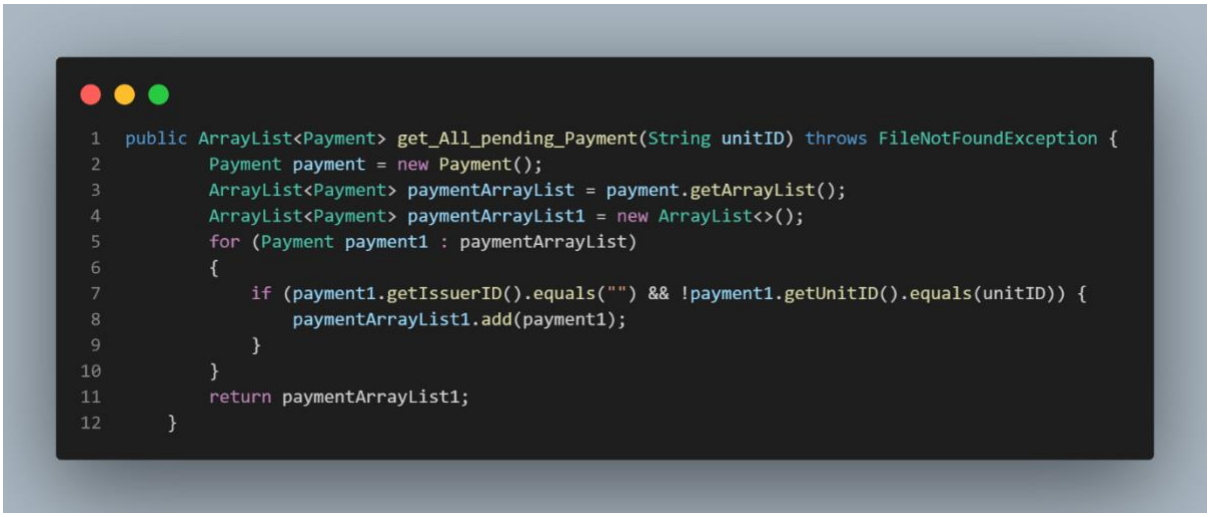


```java
public void make_Payment(Invoice invoice, String pay_Username) throws IOException {
        Payment payment1 = new Payment();
        payment1.make_Payment(invoice, pay_Username);

        Resident resident = new Resident();
        ArrayList<Resident> residentArrayList = resident.getArrayList();
        for (Resident resident1 : residentArrayList){
            resident1.setPayment(resident1.get_Unpaid_Amount(resident1.getResident_Username(), resident1.getUnitID()));
        }
        resident.save_All_Resident(residentArrayList);
    }
```

10.5.5 get_All_Receipt

```java
public ArrayList<Payment> get_All_Receipt(String unitID) throws FileNotFoundException {
        Payment payment = new Payment();
        ArrayList<Payment> paymentArrayList = payment.get_All_Receipt(unitID);
        return paymentArrayList;
    }
```

10.5.6 get_All_pending_Payment

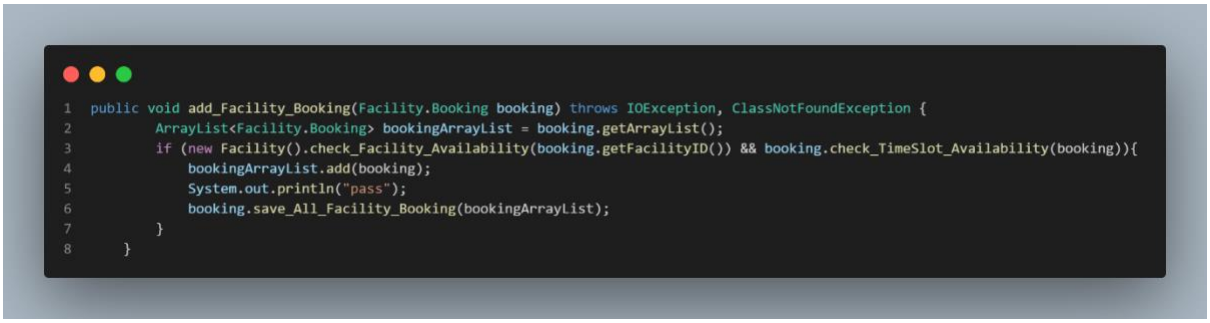


```java
public ArrayList<Payment> get_All_pending_Payment(String unitID) throws FileNotFoundException {
        Payment payment = new Payment();
        ArrayList<Payment> paymentArrayList = payment.getArrayList();
        ArrayList<Payment> paymentArrayList1 = new ArrayList<>();
        for (Payment payment1 : paymentArrayList)
        {
            if (payment1.getIssuerID().equals("") && !payment1.getUnitID().equals(unitID)) {
                paymentArrayList1.add(payment1);
            }
        }
        return paymentArrayList1;
    }
```

10.5.7 get_Statement_for_Resident

```java
public ArrayList<Statement> get_Statement_for_Resident(String resident_Username) throws FileNotFoundException {
        Statement statement = new Statement();
        return statement.get_Statement_for_Receiver(resident_Username);
    }
```

10.5.8 add_Facility_Booking



```java
public void add_Facility_Booking(Facility.Booking booking) throws IOException, ClassNotFoundException {
        ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();
        if (new Facility().check_Facility_Availability(booking.getFacilityID()) && booking.check_TimeSlot_Availability(booking)){
            bookingArrayList.add(booking);
            System.out.println("pass");
            booking.save_All_Facility_Booking(bookingArrayList);
        }
    }
```

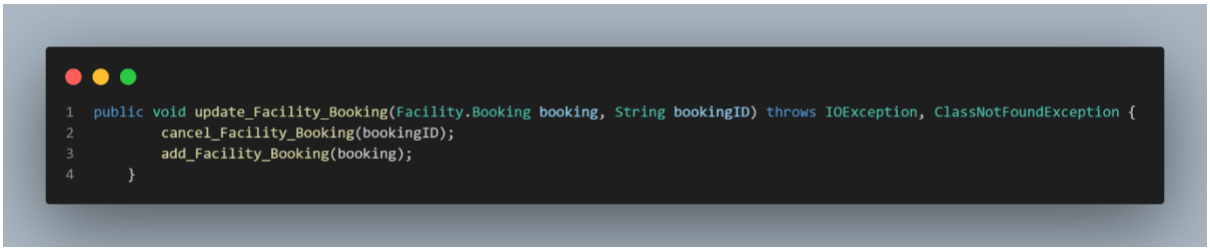
10.5.9 view_Resident_Booking

```
public ArrayList<Facility.Booking> view_Resident_Booking(String resident_Username) throws IOException, ClassNotFoundException {
        ArrayList<Facility.Booking> bookingArrayList = new Facility.Booking().getArrayList();
        ArrayList<Facility.Booking> bookingArrayList1 = new ArrayList<>();
        for (Facility.Booking booking : bookingArrayList) {
            if (booking.getResident_Username().equals(resident_Username))
                bookingArrayList1.add(booking);
        }
        return bookingArrayList1;
    }
```

10.5.10cancel_Facility_Booking

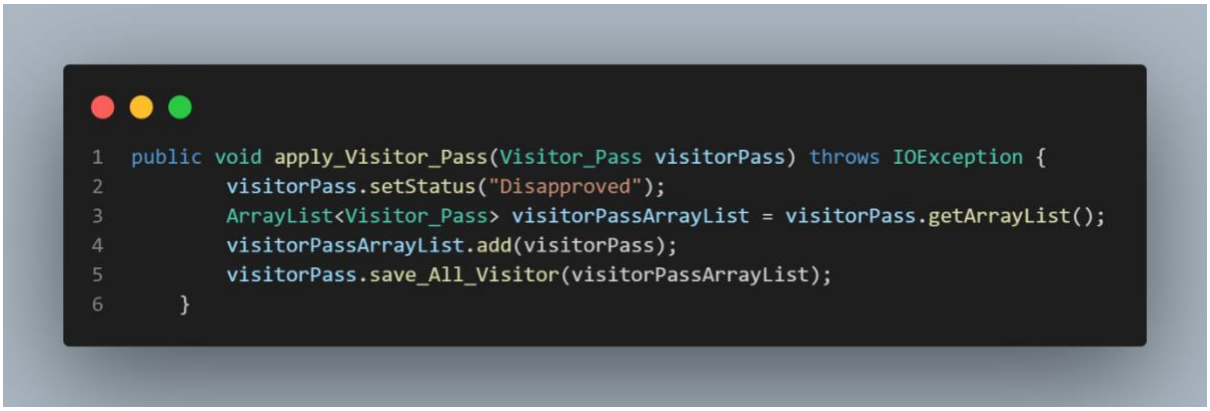
```
public void cancel_Facility_Booking(String bookingID) throws IOException, ClassNotFoundException {
        ArrayList<Facility.Booking> bookingArrayList = new Facility.Booking().getArrayList();
        ArrayList<Facility.Booking> bookingArrayList1 = new ArrayList<>();
        for (Facility.Booking booking : bookingArrayList){
            if (!(booking.getBookingID().equals(bookingID)))
                bookingArrayList1.add(booking);
        }
        new Facility.Booking().save_All_Facility_Booking(bookingArrayList1);
    }
```

10.5.11update_Facility_Booking



```
public void update_Facility_Booking(Facility.Booking booking, String bookingID) throws IOException, ClassNotFoundException {
        cancel_Facility_Booking(bookingID);
        add_Facility_Booking(booking);
    }
```

10.5.12apply_Visitor_Pass

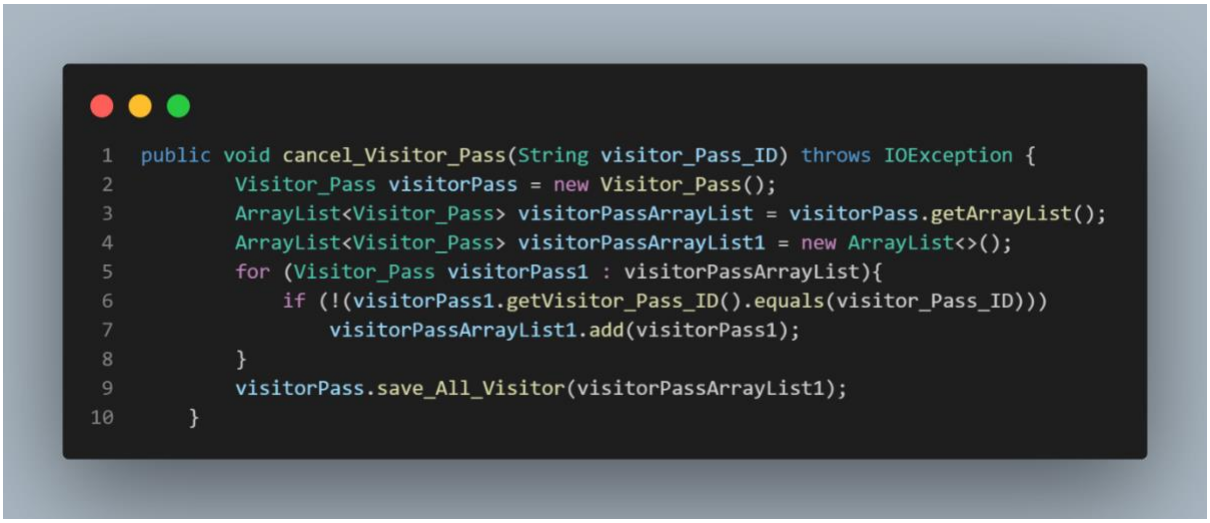


```java
public void apply_Visitor_Pass(Visitor_Pass visitorPass) throws IOException {
        visitorPass.setStatus("Disapproved");
        ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
        visitorPassArrayList.add(visitorPass);
        visitorPass.save_All_Visitor(visitorPassArrayList);
    }
```

10.5.13view_All_Visitor_Pass_Apply

```java
public ArrayList<Visitor_Pass> view_All_Visitor_Pass_Apply(String resident_Username) throws FileNotFoundException {
        Visitor_Pass visitorPass = new Visitor_Pass();
        ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
        ArrayList<Visitor_Pass> visitorPassesReturn = new ArrayList<>();
        for (Visitor_Pass visitorPass1 : visitorPassArrayList){
            if (visitorPass1.getResident_Username().equals(resident_Username))
                visitorPassesReturn.add(visitorPass1);
        }
        return visitorPassesReturn;
    }
```

10.5.14cancel_Visitor_Pass



```
public void cancel_Visitor_Pass(String visitor_Pass_ID) throws IOException {
        Visitor_Pass visitorPass = new Visitor_Pass();
        ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
        ArrayList<Visitor_Pass> visitorPassArrayList1 = new ArrayList<>();
        for (Visitor_Pass visitorPass1 : visitorPassArrayList){
            if (!(visitorPass1.getVisitor_Pass_ID().equals(visitor_Pass_ID)))
                visitorPassArrayList1.add(visitorPass1);
        }
        visitorPass.save_All_Visitor(visitorPassArrayList1);
    }
```

10.5.15update_Visitor_Pass

```
public void update_Visitor_Pass(Visitor_Pass visitorPass) throws IOException {
        cancel_Visitor_Pass(visitorPass.getVisitor_Pass_ID());
        apply_Visitor_Pass(visitorPass);
    }
```

10.5.16log_Complaint

```
public void log_Complaint(Complaint complaint) throws IOException {
        ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
        complaintArrayList.add(complaint);
        complaint.save_All_Complaint(complaintArrayList);
    }
```

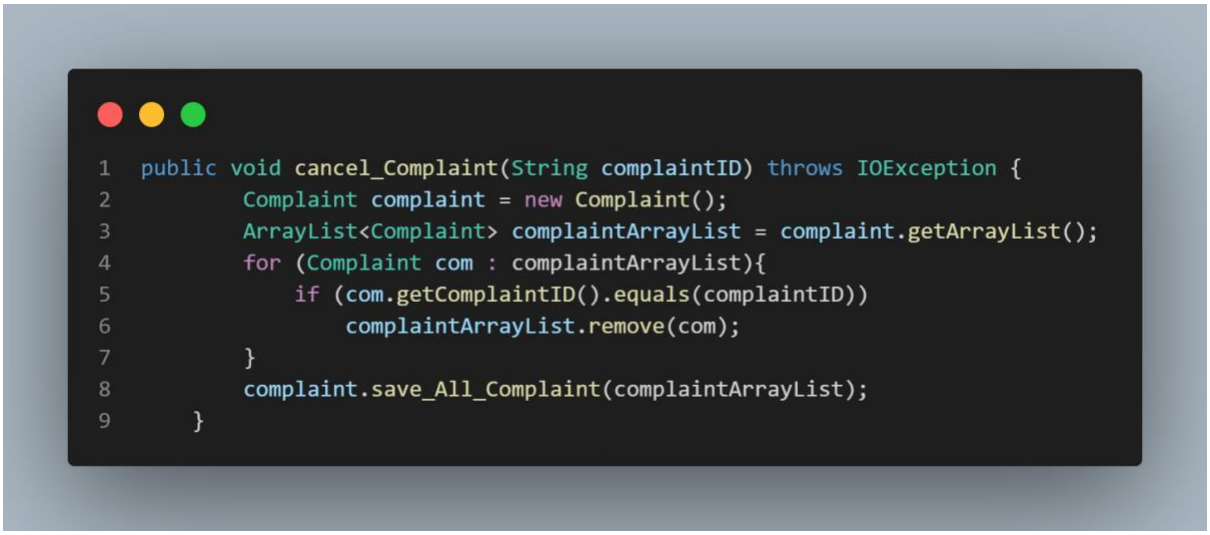
10.5.17view_Complaint

```java
public ArrayList<Complaint> view_Complaint(String resident_Username) throws FileNotFoundException {
        Complaint complaint = new Complaint();
        ArrayList<Complaint> complaintArrayList =  complaint.getArrayList();
        ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
        for (Complaint complaint1 : complaintArrayList){
            if (complaint1.getResident_Username().equals(resident_Username)){
                complaintArrayList1.add(complaint1);
            }
        }
        return complaintArrayList1;
    }
```

10.5.18update_Complaint

```java
public void update_Complaint(Complaint complaint) throws IOException {
        if (complaint.check_Complaint_Availability(complaint.getComplaintID())){
            cancel_Complaint(complaint.getComplaintID());
            log_Complaint(complaint);
        }
    }
```

10.5.19cancel_Complaint



```java
public void cancel_Complaint(String complaintID) throws IOException {
        Complaint complaint = new Complaint();
        ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
        for (Complaint com : complaintArrayList){
            if (com.getComplaintID().equals(complaintID))
                complaintArrayList.remove(com);
        }
        complaint.save_All_Complaint(complaintArrayList);
    }
```

10.6 Vendor

10.6.1 Vendor

```java
public Vendor(String vendor_Username, String name, char gender, String contact_Number, String vendor_Unit, int monthly_payment, int unpaid_payment) {
        this.vendor_Username = vendor_Username;
        this.name = name;
        this.gender = gender;
        this.contact_Number = contact_Number;
        this.vendor_Unit = vendor_Unit;
        this.monthly_payment = monthly_payment;
        this.unpaid_payment = unpaid_payment;
    }
```

10.6.2 get_Vendor_Info

```java
public Vendor get_Vendor_Info(String vendor_Username) throws FileNotFoundException {
        Vendor vendor = new Vendor();
        vendor.setVendor_Username("0");
        ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
        for (Vendor vendor1 : vendorArrayList){
            if (vendor1.getVendor_Username().equals(vendor_Username))
                return vendor1;
        }
        return vendor;
    }
```

10.6.3 update_Vendor_Info

```java
public void update_Vendor_Info(Vendor vendor, String vendor_Username) throws IOException {
        ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
        ArrayList<Vendor> vendorArrayList1 = new ArrayList<>();
        for (Vendor vendor1 : vendorArrayList){
            if (!(vendor1.getVendor_Username().equals(vendor_Username)))
                vendorArrayList1.add(vendor1);
        }
        vendorArrayList1.add(vendor);
        vendor.save_All_Vendor(vendorArrayList1);
    }
```

10.6.4 make_Payment

```java
public void make_Payment(Invoice invoice, String pay_Username) throws IOException {
        Payment payment1 = new Payment();
        payment1.make_Payment(invoice, pay_Username);

        Vendor vendor = new Vendor();
        ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
        for (Vendor vendor1 : vendorArrayList){
            vendor1.setUnpaid_payment(vendor1.get_Unpaid_Amount(vendor1.getVendor_Username(), vendor1.getVendor_Unit()));
        }
        vendor.save_All_Vendor(vendorArrayList);
    }
```

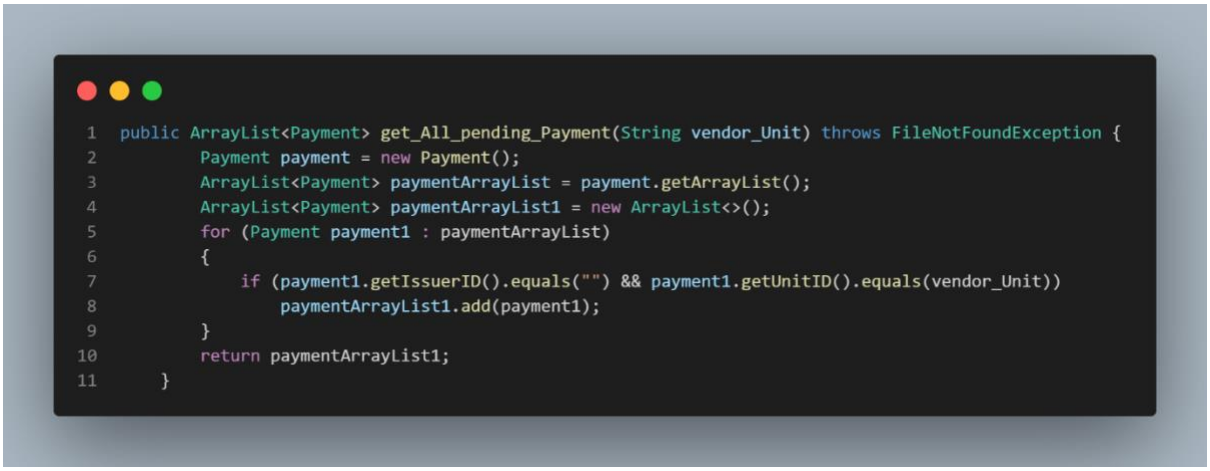
10.6.5 change_Monthly_Payment_Rental

```java
public void change_Monthly_Payment_Rental(String vendor_Unit, int rental_Amount) throws IOException {
        ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
        for (Vendor vendor : vendorArrayList){
            if (vendor.getVendor_Unit().equals(vendor_Unit)){
                vendor.setMonthly_payment(rental_Amount);
            }
        }
        save_All_Vendor(vendorArrayList);
    }
```

10.6.6 get_All_Receipt

```java
public ArrayList<Payment> get_All_Receipt(String vendor_Unit) throws FileNotFoundException {
        Payment payment = new Payment();
        ArrayList<Payment> paymentArrayList = payment.get_All_Receipt(vendor_Unit);
        return paymentArrayList;
    }
```

10.6.7 get_All_pending_Payment

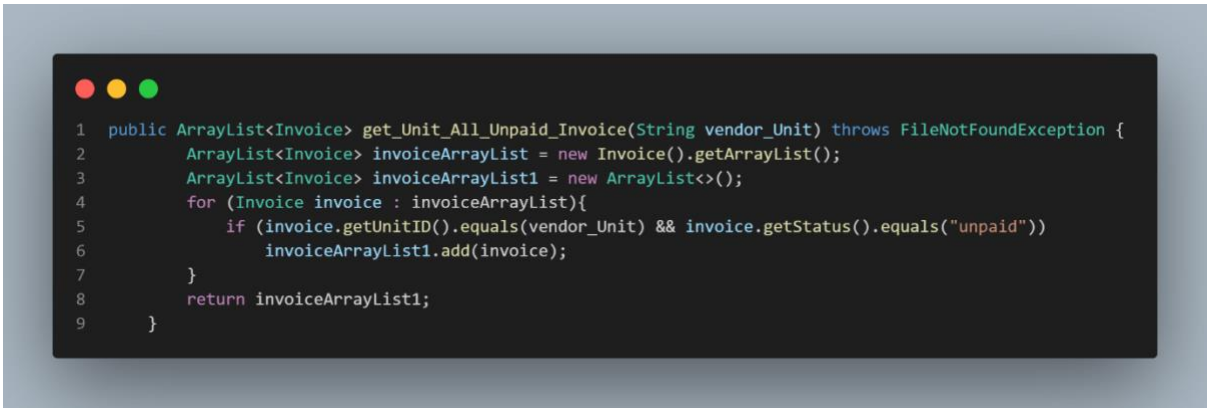


```java
public ArrayList<Payment> get_All_pending_Payment(String vendor_Unit) throws FileNotFoundException {
        Payment payment = new Payment();
        ArrayList<Payment> paymentArrayList = payment.getArrayList();
        ArrayList<Payment> paymentArrayList1 = new ArrayList<>();
        for (Payment payment1 : paymentArrayList)
        {
            if (payment1.getIssuerID().equals("") && payment1.getUnitID().equals(vendor_Unit))
                paymentArrayList1.add(payment1);
        }
        return paymentArrayList1;
    }
```

10.6.8 get_Statement_for_Vendor

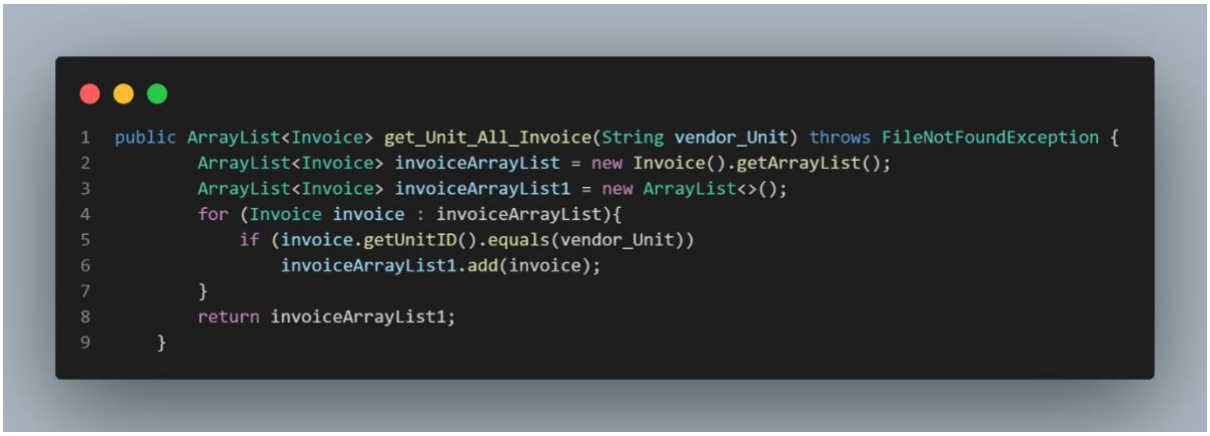
```java
public ArrayList<Statement> get_Statement_for_Vendor(String vendor_Username) throws FileNotFoundException {
        Statement statement = new Statement();
        return statement.get_Statement_for_Receiver(vendor_Username);
    }

```

10.6.9 get_Unit_All_Unpaid_Invoice



```java
public ArrayList<Invoice> get_Unit_All_Unpaid_Invoice(String vendor_Unit) throws FileNotFoundException {
        ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
        ArrayList<Invoice> invoiceArrayList1 = new ArrayList<>();
        for (Invoice invoice : invoiceArrayList){
            if (invoice.getUnitID().equals(vendor_Unit) && invoice.getStatus().equals("unpaid"))
                invoiceArrayList1.add(invoice);
        }
        return invoiceArrayList1;
    }
```

10.6.10get_Unit_All_Invoice



```
public ArrayList<Invoice> get_Unit_All_Invoice(String vendor_Unit) throws FileNotFoundException {
        ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
        ArrayList<Invoice> invoiceArrayList1 = new ArrayList<>();
        for (Invoice invoice : invoiceArrayList){
            if (invoice.getUnitID().equals(vendor_Unit))
                invoiceArrayList1.add(invoice);
        }
        return invoiceArrayList1;
    }
```

10.6.11vendor_Unit_rent

```
public ArrayList<String> vendor_Unit_rent(String vendor_Username) throws FileNotFoundException {
        ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
        ArrayList<String> data = new ArrayList<>();
        for (Vendor vendor1 : vendorArrayList){
            if (vendor1.getVendor_Username().equals(vendor_Username))
                data.add(vendor1.getVendor_Unit());
        }
        return data;
    }
```

10.6.12log_Complaint

```
public void log_Complaint(Complaint complaint) throws IOException {
        ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
        complaintArrayList.add(complaint);
        complaint.save_All_Complaint(complaintArrayList);
    }
```

10.6.13view_Complaint

```java
public ArrayList<Complaint> view_Complaint(String resident_Username) throws FileNotFoundException {
        Complaint complaint = new Complaint();
        ArrayList<Complaint> complaintArrayList =  complaint.getArrayList();
        ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
        for (Complaint complaint1 : complaintArrayList){
            if (complaint1.getResident_Username().equals(resident_Username))
                complaintArrayList1.add(complaint1);
        }
        return complaintArrayList1;
    }
```

10.6.14update_Complaint

```java
public void update_Complaint(Complaint complaint, String complaintID) throws IOException {
        if (complaint.check_Complaint_Availability(complaintID)){
            cancel_Complaint(complaintID);
            log_Complaint(complaint);
        }
    }
```

10.6.15cancel_Complaint

```java
public void cancel_Complaint(String complaintID) throws IOException {
        Complaint complaint = new Complaint();
        ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
        ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
        for (Complaint com : complaintArrayList){
            if (!(com.getComplaintID().equals(complaintID)))
                complaintArrayList1.add(com);
        }
        complaint.save_All_Complaint(complaintArrayList1);
    }
```

10.7 Security Guard

10.7.1 SecurityGuard extends Employee

```java
public class SecurityGuard extends Employee implements Serializable{
    protected String position = "Security Guard";
    private File security_Guard_Info_txt = new File("src/Database/SecurityGuard_Information.txt");
    private File visitor_Entry_Record_txt = new File("src/Database/Visitors_Entry_Record.txt");
    public  SecurityGuard(){}
    public SecurityGuard(String employeeID){}
    public SecurityGuard(String employeeID, String name, char gender, String contact_Number, int salary) throws IOException, ClassNotFoundException, NotSerializableException {
        super();
        super.set_Info(employeeID, name, gender, contact_Number, salary, position);
    }
```

10.7.2 update_SecurityGuard_Info

```java
public void update_SecurityGuard_Info(SecurityGuard securityGuard, String employeeID) throws IOException, ClassNotFoundException {
        ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
        for (SecurityGuard securityGuard1 : securityGuardArrayList){
            if (securityGuard1.getEmployeeID().equals(employeeID))
                securityGuardArrayList.remove(securityGuard1);
        }
        securityGuardArrayList.add(securityGuard);
        securityGuard.save_All_SecurityGuard(securityGuardArrayList);
    }
```

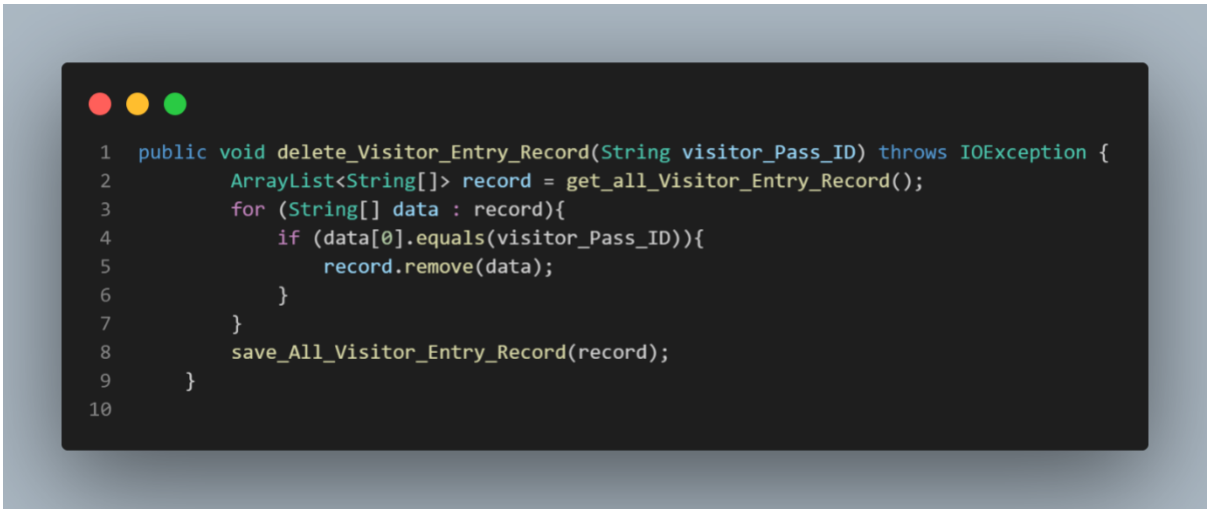
10.7.3 get_Security_Guard_Info

```java
public SecurityGuard get_Security_Guard_Info(String employeeID) throws IOException, ClassNotFoundException {
        SecurityGuard securityGuard = new SecurityGuard();
        securityGuard.setEmployeeID("0");
        ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
        for (SecurityGuard securityGuard1 : securityGuardArrayList){
            if (securityGuard1.getEmployeeID().equals(employeeID)){
                return securityGuard1;
            }
        }
        return securityGuard;
    }
```

10.7.4 add_Visitor_Entry_Record

```
public void add_Visitor_Entry_Record(String visitor_Pass_ID) throws IOException {
        LocalDateTime myDateObj = LocalDateTime.now();
        DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("MM.dd.yyyy:HHmmss");
        String data = visitor_Pass_ID + ":" + myDateObj.format(myFormatObj) + '\n';
        FileWriter fileWriter = new FileWriter(visitor_Entry_Record_txt, true);
        fileWriter.write(data);
        fileWriter.close();
        JOptionPane.showMessageDialog(null, "Visitor Entry Record add success", "Record success added", JOptionPane.INFORMATION_MESSAGE);
    }
```

10.7.5 delete_Visitor_Entry_Record

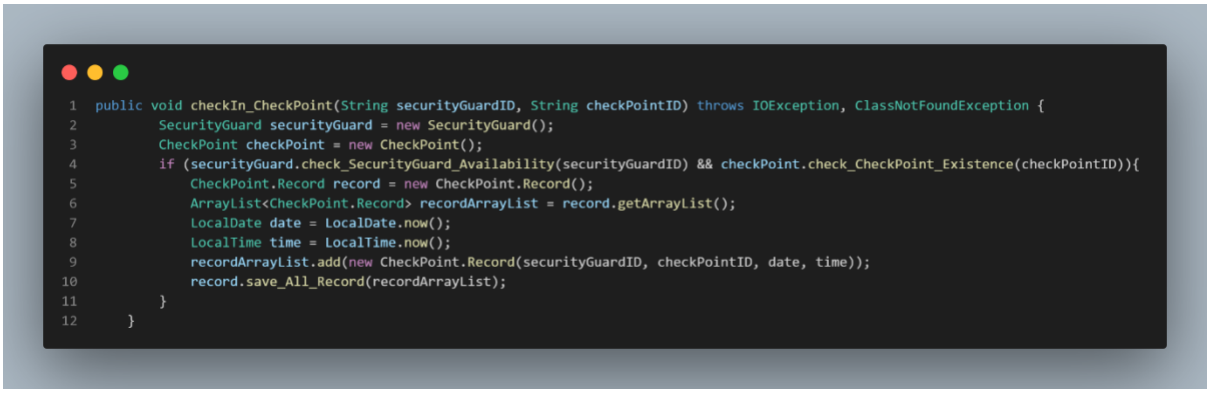


```
public void delete_Visitor_Entry_Record(String visitor_Pass_ID) throws IOException {
        ArrayList<String[]> record = get_all_Visitor_Entry_Record();
        for (String[] data : record){
            if (data[0].equals(visitor_Pass_ID)){
                record.remove(data);
            }
        }
        save_All_Visitor_Entry_Record(record);
    }

```

10.7.6 update_Visitor_Entry_Record

```
public void update_Visitor_Entry_Record(String visitor_Pass_ID, String dateTime) throws IOException {
        delete_Visitor_Entry_Record(visitor_Pass_ID);
        add_Visitor_Entry_Record(visitor_Pass_ID, dateTime);
    }

```

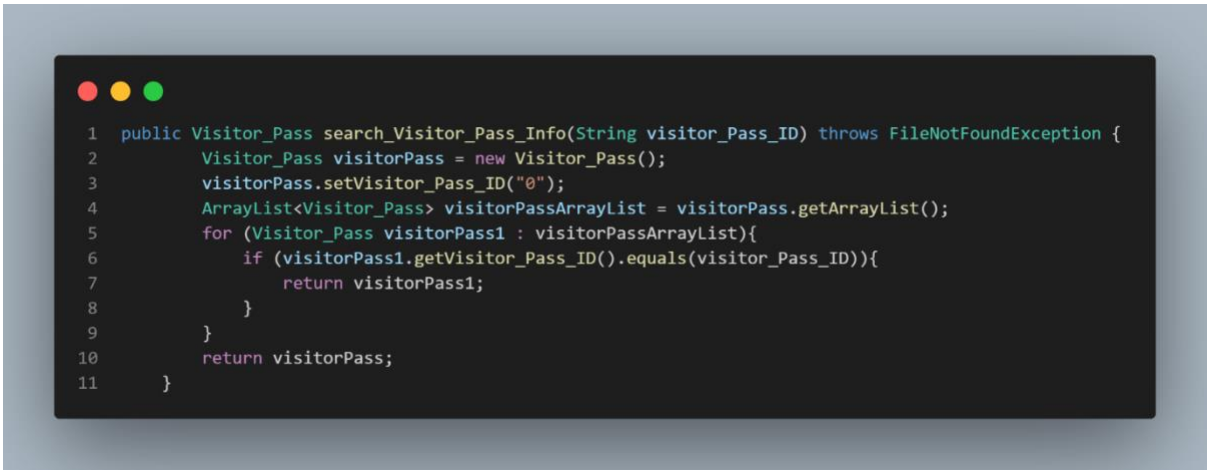
10.7.7 checkIn_CheckPoint



```java
public void checkIn_CheckPoint(String securityGuardID, String checkPointID) throws IOException, ClassNotFoundException {
        SecurityGuard securityGuard = new SecurityGuard();
        CheckPoint checkPoint = new CheckPoint();
        if (securityGuard.check_SecurityGuard_Availability(securityGuardID) && checkPoint.check_CheckPoint_Existence(checkPointID)){
            CheckPoint.Record record = new CheckPoint.Record();
            ArrayList<CheckPoint.Record> recordArrayList = record.getArrayList();
            LocalDate date = LocalDate.now();
            LocalTime time = LocalTime.now();
            recordArrayList.add(new CheckPoint.Record(securityGuardID, checkPointID, date, time));
            record.save_All_Record(recordArrayList);
        }
    }
```

10.8 Visitor

10.8.1 search_Visitor_Pass_Info



```java
public Visitor_Pass search_Visitor_Pass_Info(String visitor_Pass_ID) throws FileNotFoundException {
        Visitor_Pass visitorPass = new Visitor_Pass();
        visitorPass.setVisitor_Pass_ID("0");
        ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
        for (Visitor_Pass visitorPass1 : visitorPassArrayList){
            if (visitorPass1.getVisitor_Pass_ID().equals(visitor_Pass_ID)){
                return visitorPass1;
            }
        }
        return visitorPass;
    }
```

10.8.2 search_Visitor_Pass_Info_by_Name

```java
public Visitor_Pass search_Visitor_Pass_Info_by_Name(String visitor_Name) throws FileNotFoundException {
        Visitor_Pass visitorPass = new Visitor_Pass();
        visitorPass.setVisitor_Pass_ID("0");
        ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
        for (Visitor_Pass visitorPass1 : visitorPassArrayList){
            if (visitorPass1.getVisitor_Name().equals(visitor_Name)){
                return visitorPass1;
            }
        }
        return visitorPass;
    }
```

10.8.3 get_Auto_Visitor_Pass_ID

```java
public String get_Auto_Visitor_Pass_ID() throws FileNotFoundException {
        FileReader fileReader = new FileReader(visitor_Pass_Info_txt);
        Scanner scanner = new Scanner(fileReader);
        scanner.nextLine();
        Integer num = 0;
        while (scanner.hasNextLine()){
            String[] data = scanner.nextLine().split(":", 9);
            String number = data[0].substring(2);
            num = Integer.parseInt(number);
            num += 1;
        }
        String str = "VI" + num.toString();
        return str;
    }
```

11 Conclusion

In conclusion, the implementation of OOP in the Parkhill Residence Condominium's property management system has significantly improved the system's functionality, flexibility, and maintainability. The use of OOP concepts has allowed the system to be organized and structured in a way that each user's functionalities have been clearly defined. In addition, by creating individual classes for each user type, the system has become more modular, making it easier to add new features and modify existing ones in the future, thus making the system more scalable.

The OOP concepts of inheritance, encapsulation, abstraction, modularity, and polymorphism have enabled different user classes to interact with each other seamlessly. This interaction has facilitated better communication and coordination between users, which has improved the system's overall efficiency. Moreover, the system's modular design has made it easier to debug and maintain, reducing the likelihood of errors and downtime.

The Parkhill Residence Condominium's property management system is an excellent example of how OOP can create a complex, efficient, and user-friendly software system. The system's implementation has resulted in software that is easy to use and maintain. In addition, as the condominium establishment grows and changes, the system can be scaled to meet its needs.

In summary, implementing OOP concepts in the Parkhill Residence Condominium's property management system has been beneficial in many ways. It has improved the system's functionality, flexibility, and maintainability, making it more efficient and user-friendly. With the ability to scale and adapt to the changing needs of the establishment, the system will continue to provide excellent service to its users.

12 References

Wegner, P. (1990). Concepts and paradigms of object-oriented programming. *OOPS Messenger*, *1*(1), 7–87. https://doi.org/10.1145/382192.383004

Agha, G. (1990). Concurrent object-oriented programming. *Communications of the ACM*, *33*(9), 125–141. https://doi.org/10.1145/83880.84528