



## 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

<b>TABLE OF CONTENTS.....</b>	<b>3</b>
<b>1 ABSTRACT .....</b>	<b>10</b>
<b>2 INTRODUCTION .....</b>	<b>10</b>
<b>3 ASSUMPTIONS .....</b>	<b>12</b>
<b>4 UNIFIED MODELLING LANGUAGE DIAGRAM.....</b>	<b>13</b>
4.1 USE-CASE DIAGRAM .....	13
4.2 CLASS DIAGRAM .....	19
4.2.1 <i>Executive Function</i> .....	21
4.2.2 <i>Account Executive Function</i> .....	21
4.2.3 <i>Admin Executive Function</i> .....	22
4.2.4 <i>Building Executive Function</i> .....	22
4.2.5 <i>Building Manager Function</i> .....	23
4.2.6 <i>Team Leader</i> .....	23
4.2.7 <i>Operation</i> .....	24
4.2.8 <i>Employee</i> .....	24
4.2.9 <i>Technician</i> .....	24
4.2.10 <i>Cleaner</i> .....	25
4.2.11 <i>Security Guard</i> .....	25
4.2.12 <i>Employee Task</i> .....	25
4.2.13 <i>Patrolling</i> .....	26
4.2.14 <i>Check Point</i> .....	26
4.2.15 <i>Incident</i> .....	27
4.2.16 <i>Resident</i> .....	27
4.2.17 <i>Vendor</i> .....	28
4.2.18 <i>Login</i> .....	29
4.2.19 <i>Payment</i> .....	29
4.2.20 <i>Invoice</i> .....	30
4.2.21 <i>Deposit</i> .....	30
4.2.22 <i>Statement</i> .....	31
4.2.23 <i>Unit</i> .....	31
4.2.24 <i>Facility</i> .....	32
4.2.25 <i>Booking</i> .....	32
4.2.26 <i>Visitor Pass</i> .....	33
4.3 USE-CASE SPECIFICATION .....	34
4.3.1 <i>User log in and out</i> .....	34
4.3.2 <i>Invoice, Receipt, and Statement Management</i> .....	35
4.3.3 <i>Payment Management</i> .....	38

4.3.4	<i>Pending Fee Management</i> .....	39
4.3.5	<i>Deposit Management</i> .....	40
4.3.6	<i>Unit Management</i> .....	40
4.3.7	<i>Resident Management</i> .....	42
4.3.8	<i>Complaint Management</i> .....	44
4.3.9	<i>Employee Management</i> .....	47
4.3.10	<i>Facility Management</i> .....	50
4.3.11	<i>Facility Booking Management</i> .....	52
4.3.12	<i>Vendor Management</i> .....	54
4.3.13	<i>Visitor Pass Management</i> .....	57
4.3.14	<i>Employee Task Management</i> .....	58
4.3.15	<i>Patrolling Management</i> .....	60
4.3.16	<i>Checkpoint Management</i> .....	63
4.3.17	<i>Job Report Management</i> .....	66
4.3.18	<i>User Management</i> .....	67
4.3.19	<i>Operation Budgeting Management</i> .....	70
4.3.20	<i>Team Leader Management</i> .....	73
4.3.21	<i>Security Guard Functionalities</i> .....	75
4.3.22	<i>Resident Functionalities</i> .....	79
<b>5</b>	<b>OBJECT-ORIENTED CONCEPTS</b> .....	<b>85</b>
5.1	ATTRIBUTE .....	85
5.2	CLASSES .....	85
5.3	OBJECT .....	87
5.4	OPERATION (METHOD) .....	89
5.5	PACKAGE .....	90
<b>6</b>	<b>OBJECT-ORIENTED PRINCIPLES</b> .....	<b>91</b>
6.1	MODULARITY .....	91
6.2	ABSTRACTION.....	93
6.3	ENCAPSULATION.....	94
6.4	INHERITANCE.....	96
6.5	POLYMORPHISM.....	98
<b>7</b>	<b>TP067349 NADILA BINTI AHMAD SHAHRUL NIZAM</b> .....	<b>100</b>
7.1	USER-LEVEL ACCESS, LOGGING, AND LOGOUT ACTIVITY .....	100
7.2	RESIDENT / TENANT.....	101
7.2.1	<i>Functional Requirement</i> .....	101
7.2.2	<i>Profile</i> .....	102
7.2.3	<i>Invoice</i> .....	103

7.2.4	<i>Deposit</i> .....	105
7.2.5	<i>Payment history</i> .....	106
7.2.6	<i>Statement</i> .....	107
7.2.7	<i>Facility booking</i> .....	108
7.2.8	<i>Visitor pass</i> .....	113
7.2.9	<i>Complaint</i> .....	118
7.3	VISITOR .....	123
<b>8</b>	<b>TP065116 NEAW AIK KA</b> .....	<b>124</b>
8.1	ACCOUNT EXECUTIVE .....	124
8.1.1	<i>Functional requirements</i> .....	124
8.1.2	<i>Profile</i> .....	124
8.1.3	<i>Invoice</i> .....	125
8.1.4	<i>Payment</i> .....	128
8.1.5	<i>Receipt</i> .....	130
8.1.6	<i>Statement</i> .....	131
8.1.7	<i>Outstanding fee</i> .....	133
8.2	ADMIN EXECUTIVE.....	135
8.2.1	<i>Functional requirements</i> .....	135
8.2.2	<i>Profile</i> .....	135
8.2.3	<i>Unit Management</i> .....	137
8.2.4	<i>Resident management</i> .....	141
8.2.5	<i>Complaint management</i> .....	145
8.2.6	<i>Employee management</i> .....	150
8.2.7	<i>Facility management</i> .....	154
8.2.8	<i>Facility booking</i> .....	159
8.2.9	<i>Vendor</i> .....	163
8.2.10	<i>Visitor pass</i> .....	167
8.3	BUILDING EXECUTIVE .....	169
8.3.1	<i>Functional requirement</i> .....	169
8.3.2	<i>Profile</i> .....	170
8.3.3	<i>Employee task</i> .....	172
8.3.4	<i>Complaint management</i> .....	176
8.3.5	<i>Patrolling management</i> .....	178
8.3.6	<i>Checkpoint management</i> .....	183
8.3.7	<i>Job report</i> .....	187
8.4	REPORT OR FILE GENERATION.....	189

8.4.1	<i>System Report Panel</i> .....	189
8.4.2	<i>writeAsPNG</i> .....	190
<b>9</b>	<b>TP067435 EDMUND CHEN SIANG ZUAN .....</b>	<b>191</b>
9.1	BUILDING MANAGER.....	191
9.1.1	<i>Functional requirements</i> .....	191
9.1.2	<i>Profile</i> .....	191
9.1.3	<i>User management</i> .....	192
9.1.4	<i>Report</i> .....	197
9.1.5	<i>Operation and budget planning</i> .....	198
9.1.6	<i>Team structure management</i> .....	202
9.2	SECURITY GUARD .....	205
9.2.1	<i>Functional requirements</i> .....	205
9.2.2	<i>Profile</i> .....	206
9.2.3	<i>Visitor pass check</i> .....	207
9.2.4	<i>Visitor entry</i> .....	208
9.2.5	<i>Checkpoint</i> .....	210
9.2.6	<i>Incident</i> .....	212
9.3	OTHER FEATURES .....	215
9.3.1	<i>Menu</i> .....	215
9.3.2	<i>Loading</i> .....	215
9.3.3	<i>Report Graph</i> .....	216
9.3.4	<i>Selected Row color</i> .....	217
9.3.5	<i>Status color</i> .....	217
<b>10</b>	<b>SYSTEM CODE .....</b>	<b>218</b>
10.1	BUILDING MANAGER.....	218
10.1.1	<i>Building_Manager_Function</i> .....	218
10.1.2	<i>Save all building manager</i> .....	218
10.1.3	<i>getarraylist</i> .....	219
10.1.4	<i>Search executive information</i> .....	219
10.1.5	<i>Check building manager availability</i> .....	219
10.1.6	<i>Update building manager info</i> .....	220
10.1.7	<i>Add account executive</i> .....	220
10.1.8	<i>Delete account executive</i> .....	220
10.1.9	<i>Modify account executive</i> .....	220
10.1.10	<i>Delete building executive</i> .....	221
10.1.11	<i>Modify building executive</i> .....	221

10.1.12	<i>Add_admin_executive.....</i>	221
10.1.13	<i>Delete admin executive.....</i>	221
10.1.14	<i>modify_Admin_Executive .....</i>	222
10.1.15	<i>get_Executive_Info .....</i>	222
10.1.16	<i>get_Building_Manager_Info .....</i>	222
10.2	ACCOUNT EXECUTIVE .....	223
10.2.1	<i>Account_Executive extends Executive.....</i>	223
10.2.2	<i>update_Account_Executive_Info .....</i>	223
10.2.3	<i>get_Account_Executive_Info .....</i>	223
10.2.4	<i>issue_All_Unit_Invoice .....</i>	224
10.2.5	<i>get_All_pending_Payment.....</i>	224
10.2.6	<i>get_All_Unit_List .....</i>	225
10.2.7	<i>issue_Unit_Invoice .....</i>	225
10.2.8	<i>issue_Receipt.....</i>	226
10.2.9	<i>issue_Statement .....</i>	226
10.2.10	<i>get_Unit_All_Invoice.....</i>	226
10.2.11	<i>get_All_Unpaid_Invoice.....</i>	227
10.2.12	<i>get_Unit_Unpaid_Amount.....</i>	227
10.3	BUILDING EXECUTIVE .....	227
10.3.1	<i>Account_Executive extends Executive.....</i>	227
10.3.2	<i>save_All_Building_Executive.....</i>	228
10.3.3	<i>get_Building_Executive_Info .....</i>	228
10.3.4	<i>update_Building_Executive_Info .....</i>	228
10.3.5	<i>add_Employee_Task.....</i>	229
10.3.6	<i>delete_Employee_Task.....</i>	229
10.3.7	<i>update_Employee_Task.....</i>	229
10.3.8	<i>view_All_Complaint.....</i>	230
10.3.9	<i>add_Patrolling_Schedule .....</i>	230
10.3.10	<i>delete_Patrolling_Schedule.....</i>	230
10.3.11	<i>add_New_CheckPoint .....</i>	231
10.3.12	<i>delete_CheckPoint.....</i>	231
10.3.13	<i>modify_CheckPoint .....</i>	231
10.4	ADMIN EXECUTIVE.....	232
10.4.1	<i>Admin_Executive extends Executive .....</i>	232
10.4.2	<i>update_Admin_Executive_Info.....</i>	232
10.4.3	<i>unit_Add .....</i>	232
10.4.4	<i>unit_Modify .....</i>	233
10.4.5	<i>unit_Delete .....</i>	233

10.4.6	<i>resident_Add</i> .....	233
10.4.7	<i>resident_Modify</i> .....	234
10.4.8	<i>resident_Delete</i> .....	234
10.4.9	<i>complaint_Add</i> .....	234
10.4.10	<i>complaint_Update</i> .....	235
10.4.11	<i>complaint_Delete</i> .....	235
10.4.12	<i>add_Employee</i> .....	235
10.4.13	<i>view_All_Employee</i> .....	236
10.4.14	<i>delete_Employee</i> .....	236
10.4.15	<i>update_Employee</i> .....	237
10.4.16	<i>add_Facility</i> .....	237
10.4.17	<i>delete_Facility</i> .....	237
10.4.18	<i>update_Facility</i> .....	238
10.4.19	<i>add_Booking</i> .....	238
10.4.20	<i>delete_Booking</i> .....	238
10.4.21	<i>facility_Booking_Update</i> .....	239
10.4.22	<i>save_All_Admin_Executive</i> .....	239
10.4.23	<i>get_Admin_Executive_Info</i> .....	239
10.4.24	<i>check_Admin_Executive_Availability</i> .....	240
10.4.25	<i>vendor_Add</i> .....	240
10.4.26	<i>vendor_Modify</i> .....	240
10.4.27	<i>vendor_Delete</i> .....	240
10.5	RESIDENT.....	241
10.5.1	<i>Resident</i> .....	241
10.5.2	<i>get_Resident_Info</i> .....	241
10.5.3	<i>update_Resident_Info</i> .....	242
10.5.4	<i>make_Payment</i> .....	242
10.5.5	<i>get_All_Receipt</i> .....	242
10.5.6	<i>get_All_pending_Payment</i> .....	243
10.5.7	<i>get_Statement_for_Resident</i> .....	243
10.5.8	<i>add_Facility_Booking</i> .....	243
10.5.9	<i>view_Resident_Booking</i> .....	244
10.5.10	<i>cancel_Facility_Booking</i> .....	244
10.5.11	<i>update_Facility_Booking</i> .....	244
10.5.12	<i>apply_Visitor_Pass</i> .....	245
10.5.13	<i>view_All_Visitor_Pass_Apply</i> .....	245
10.5.14	<i>cancel_Visitor_Pass</i> .....	246
10.5.15	<i>update_Visitor_Pass</i> .....	246
10.5.16	<i>log_Complaint</i> .....	246

10.5.17	<i>view_Complaint</i> .....	247
10.5.18	<i>update_Complaint</i> .....	247
10.5.19	<i>cancel_Complaint</i> .....	247
10.6	VENDOR.....	248
10.6.1	<i>Vendor</i> .....	248
10.6.2	<i>get_Vendor_Info</i> .....	248
10.6.3	<i>update_Vendor_Info</i> .....	248
10.6.4	<i>make_Payment</i> .....	249
10.6.5	<i>change_Monthly_Payment_Rental</i> .....	249
10.6.6	<i>get_All_Receipt</i> .....	249
10.6.7	<i>get_All_pending_Payment</i> .....	250
10.6.8	<i>get_Statement_for_Vendor</i> .....	250
10.6.9	<i>get_Unit_All_Unpaid_Invoice</i> .....	250
10.6.10	<i>get_Unit_All_Invoice</i> .....	251
10.6.11	<i>vendor_Unit_rent</i> .....	251
10.6.12	<i>log_Complaint</i> .....	251
10.6.13	<i>view_Complaint</i> .....	252
10.6.14	<i>update_Complaint</i> .....	252
10.6.15	<i>cancel_Complaint</i> .....	252
10.7	SECURITY GUARD.....	253
10.7.1	<i>SecurityGuard extends Employee</i> .....	253
10.7.2	<i>update_SecurityGuard_Info</i> .....	253
10.7.3	<i>get_Security_Guard_Info</i> .....	253
10.7.4	<i>add_Visitor_Entry_Record</i> .....	254
10.7.5	<i>delete_Visitor_Entry_Record</i> .....	254
10.7.6	<i>update_Visitor_Entry_Record</i> .....	254
10.7.7	<i>checkIn_CheckPoint</i> .....	255
10.8	VISITOR .....	255
10.8.1	<i>search_Visitor_Pass_Info</i> .....	255
10.8.2	<i>search_Visitor_Pass_Info_by_Name</i> .....	256
10.8.3	<i>get_Auto_Visitor_Pass_ID</i> .....	256
<b>11</b>	<b>CONCLUSION.....</b>	<b>257</b>
<b>12</b>	<b>REFERENCES .....</b>	<b>257</b>

## 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 deployed 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.

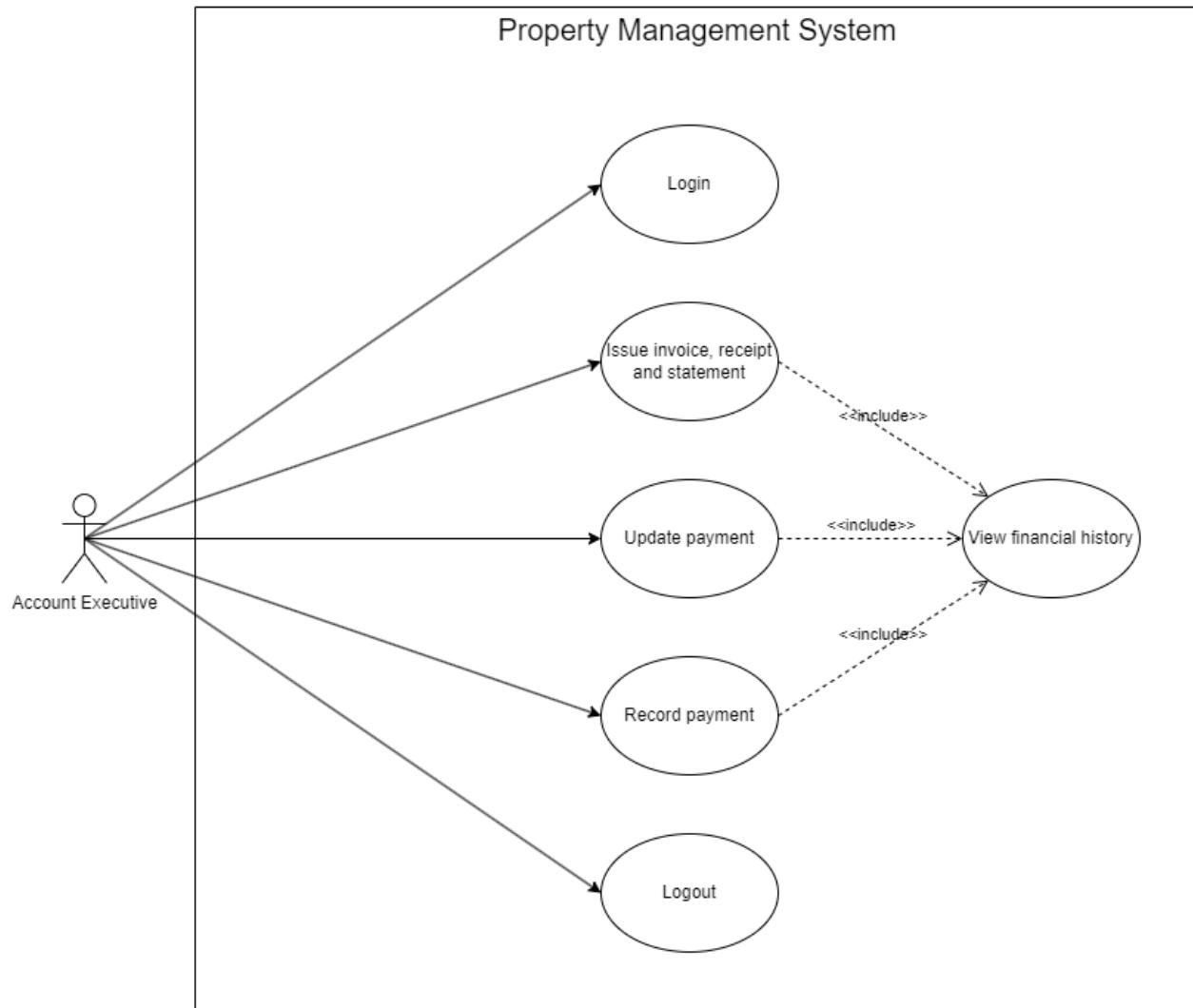
While the vendor and resident pages have separate units, the admin executive must enter the vendor's unit information.

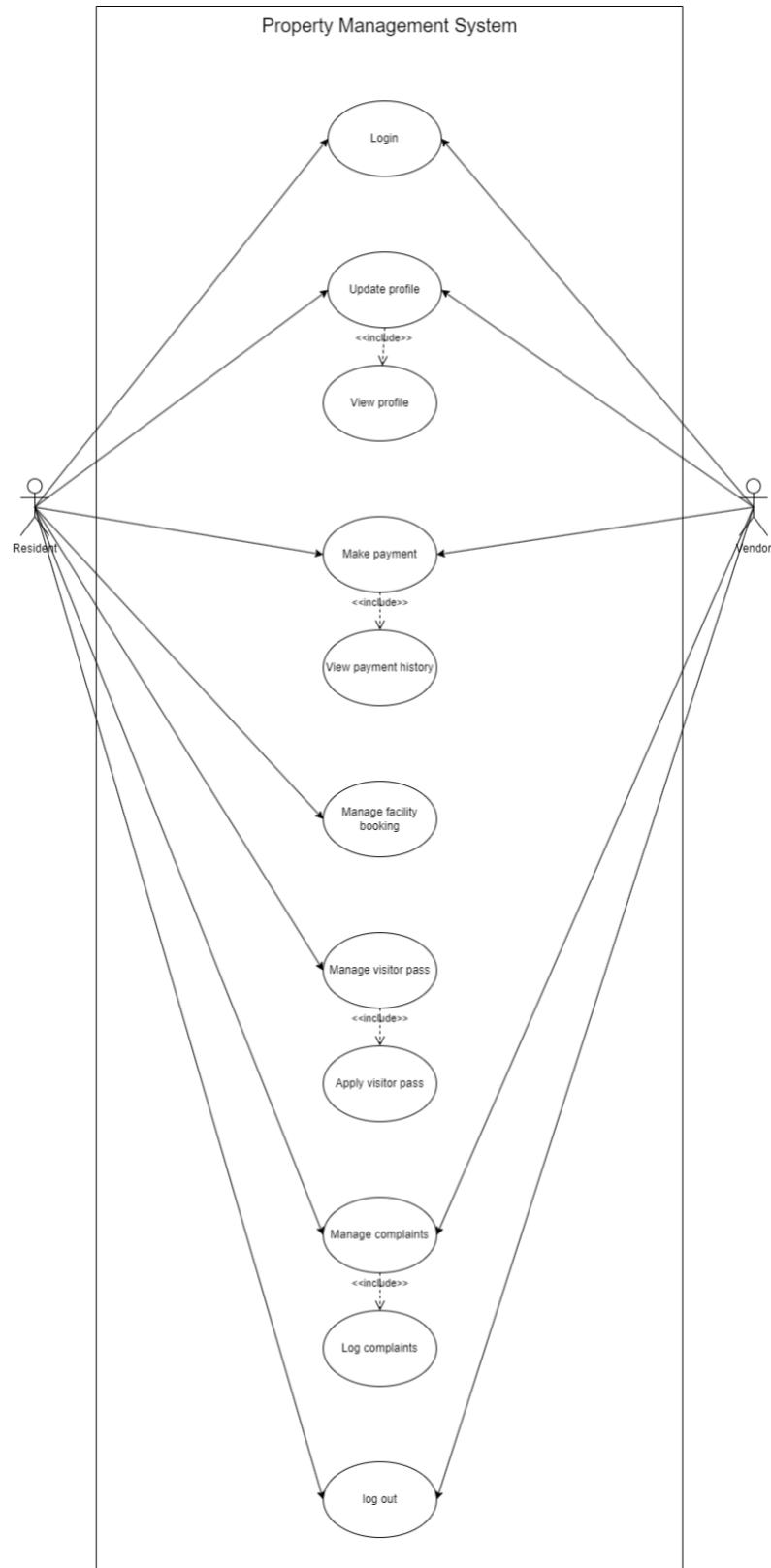
When a resident introduces a new unit, the system will automatically verify whether the unit and resident information exist. One resident can possess only one unit at the same time, but each unit can have a maximum of one resident and one owner at any given time. In the security system, the patrolling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

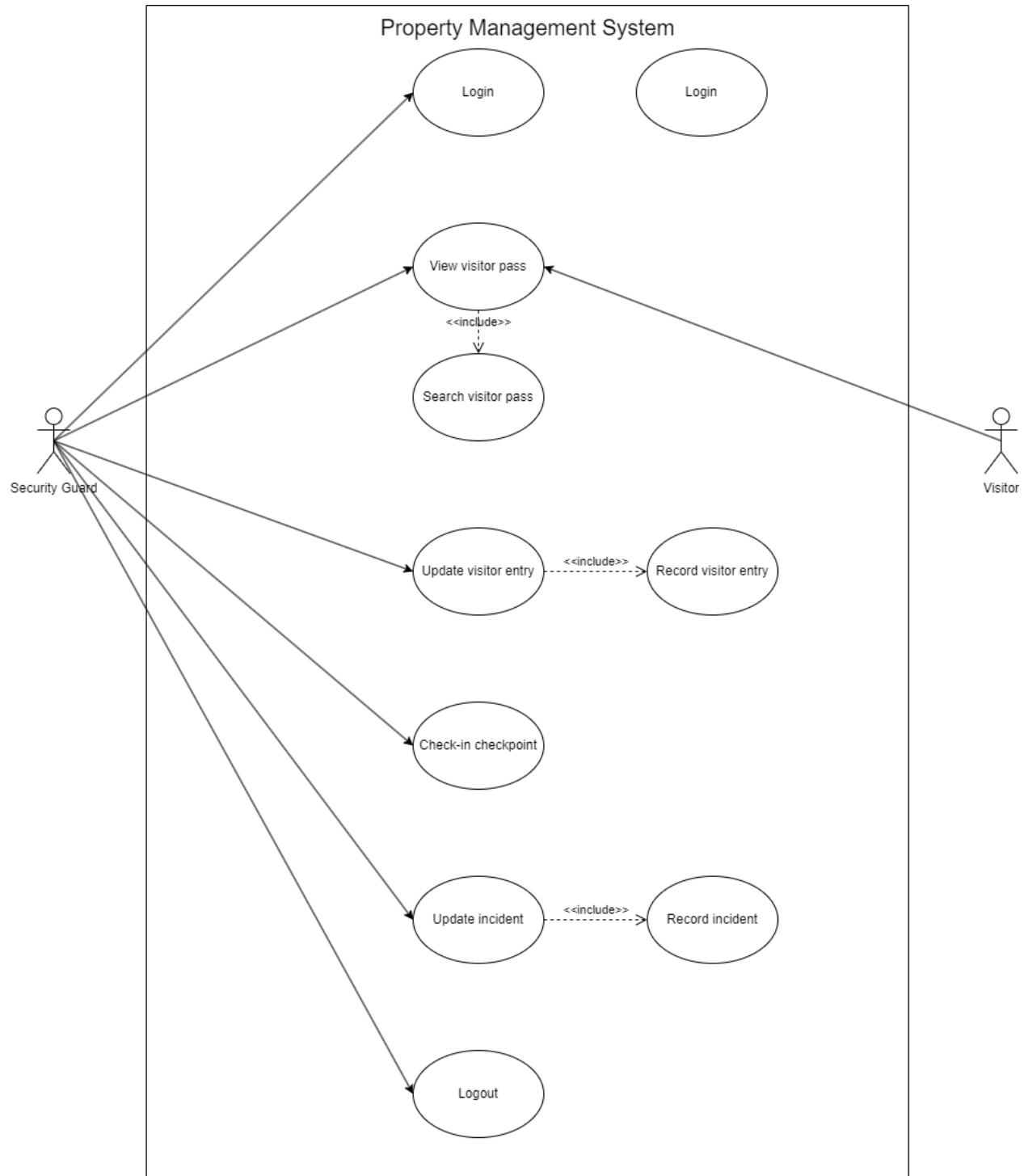
In user management, after adding a user ID, all executives and managers cannot change their own IDs. Each of them has a unique ID. This means that once their IDs are assigned and created, they cannot be changed. This is to ensure the uniqueness of user IDs in the system and the integrity of the data.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

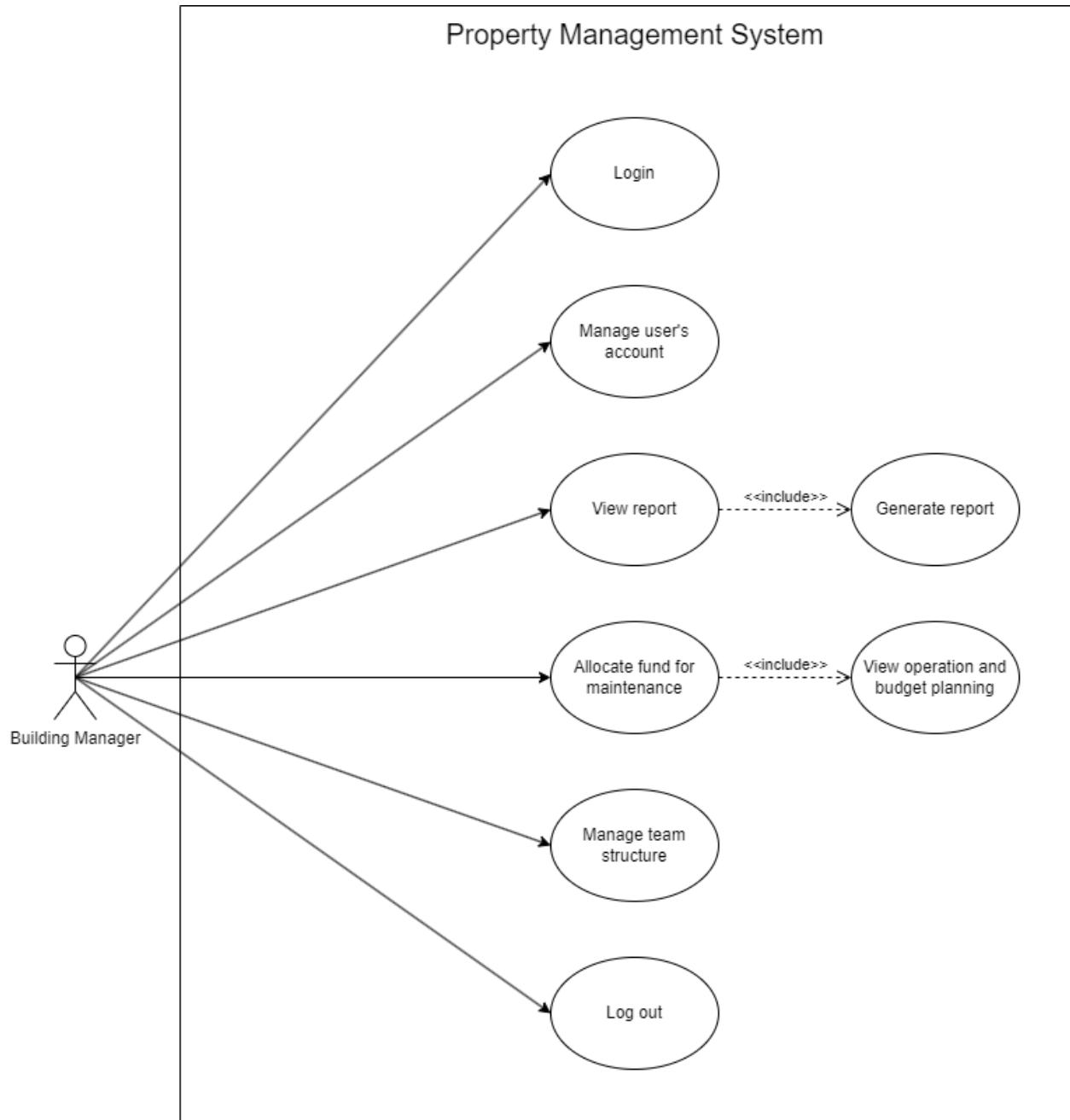
## 4 Unified Modelling Language Diagram

### 4.1 Use-Case Diagram





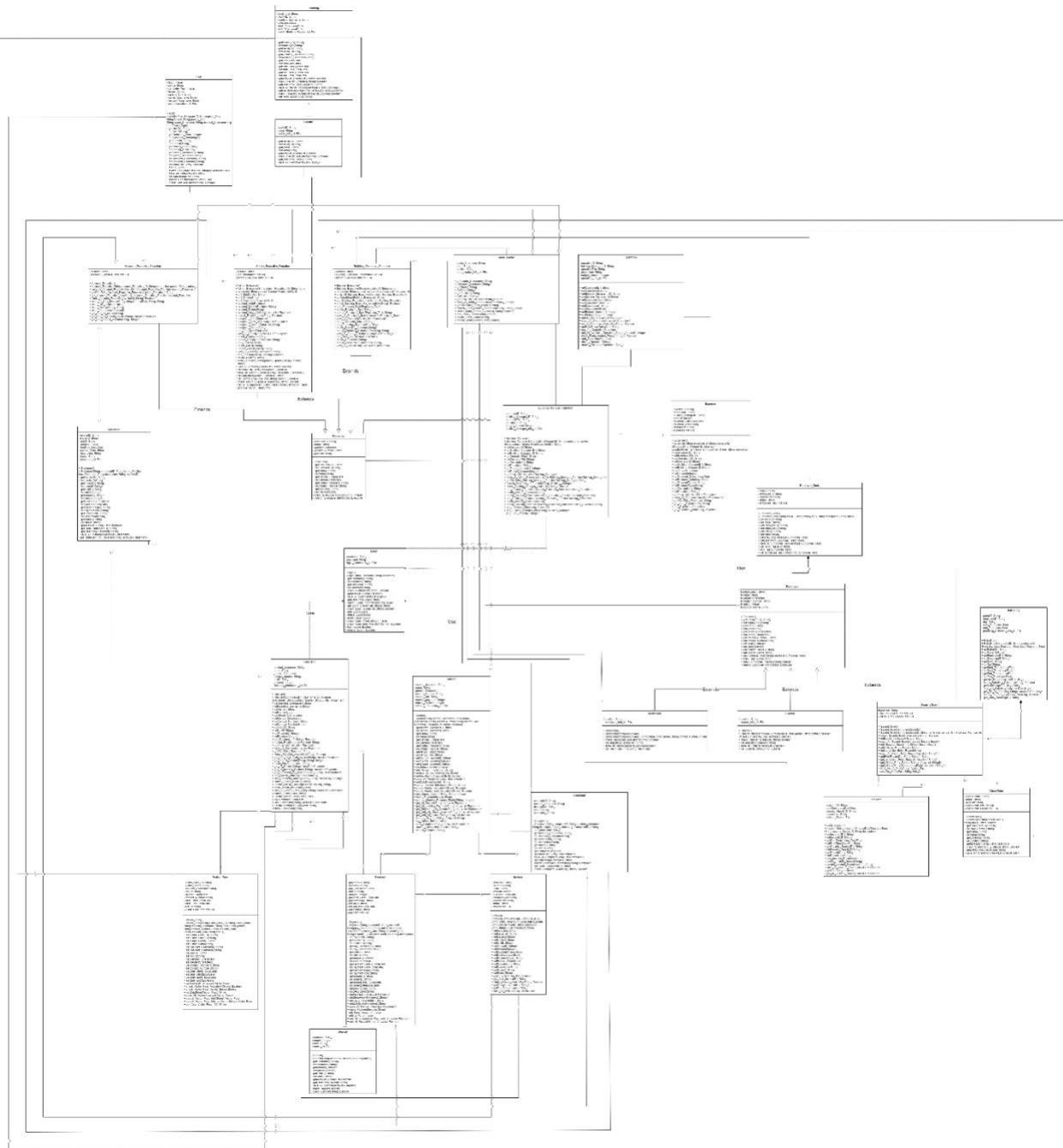




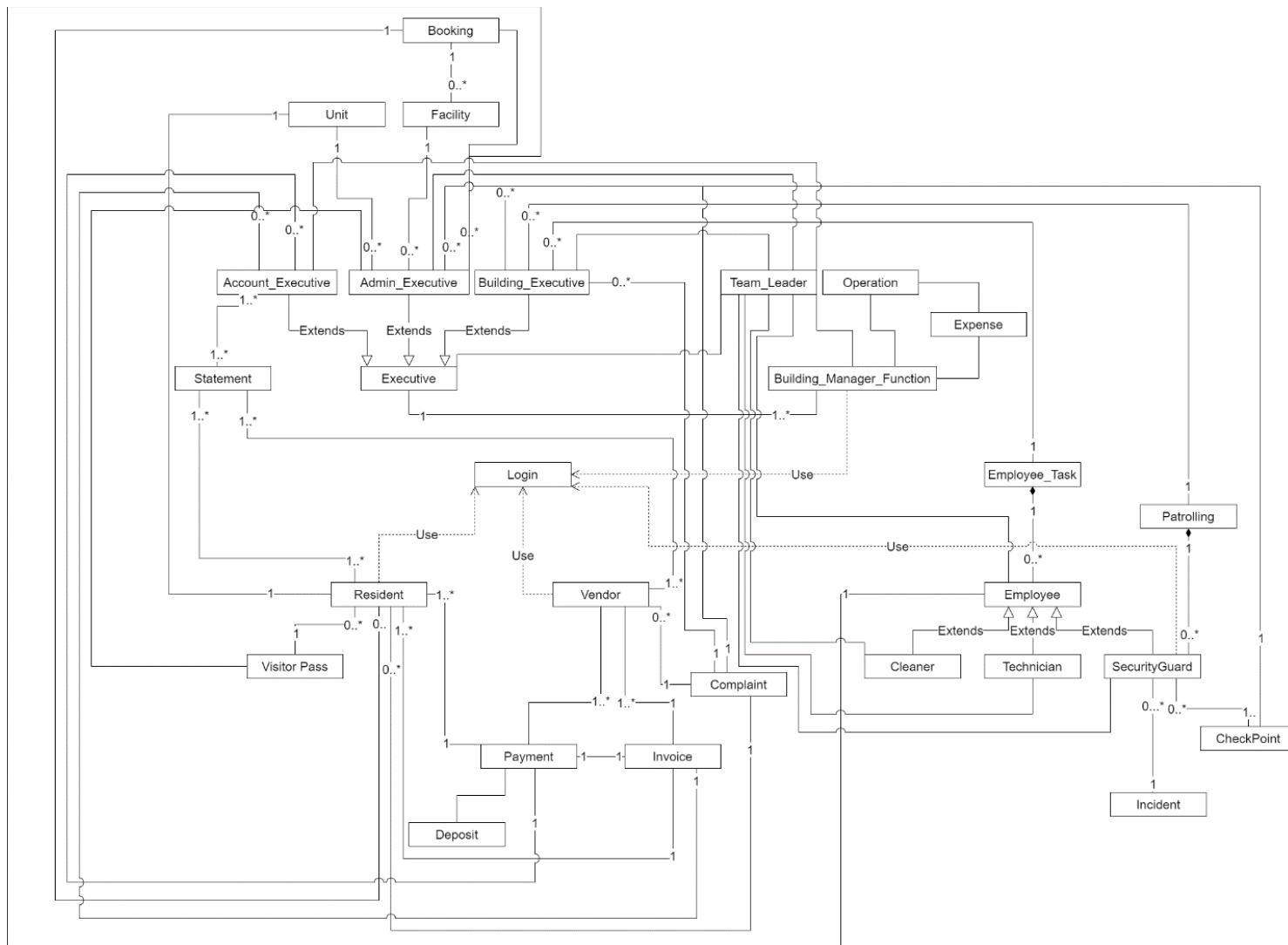




## 4.2 Class Diagram



### *full diagram*

*Class diagram (relationship only)*

### 4.2.1 Executive Function

Executive
- executiveID: String - name: String - gender: Characters - contact_Number: String - position: String  + Executive() + getExecutiveID(): String + setExecutiveID(String) + getName(): String + setName(String) + getGender(): Characters + setGender(Characters) + getContact_Number(): String + setContact_Number(String) + getPosition(): String + setPosition(String) - check_Executive_Position(String): Integer + search_Executive_Info(String): Executive

### 4.2.2 Account Executive Function

Account_Executive_Function
- position: String - account_Executive_Infor_txt: File  + Account_Executive() + Account_Executive(String account_Executive_ID, String name, char gender, String contact_Number) + getArrayList(): ArrayList<Account_Executive_Function.Account_Executive> + save_All_Account_Executive(ArrayList<Account_Executive_Function.Account_Executive>) + getDataString(Account_Executive_Function.Account_Executive): String + get_Account_Executive_Info(String): Account_Executive_Function.Account_Executive + check_Account_Executive_Availability(String): Boolean + issue_All_Unit_Invoice(String, Integer, LocalDate, String, String) + issue_Unit_Invoice(Invoice) + issue_Receipt(String, String) + issue_Statement(Statement) + get_Unit_All_Invoice(String) + get_Unit_All_Unpaid_Invoice(String): ArrayList<Invoice> + get_Unit_Unpaid_Amount(String): Integer

### 4.2.3 Admin Executive Function

Admin_Executive_Function
<pre> - position: String - unit_Information_txt: File - admin_Executive_Infor_txt: File  + Admin_Executive() + Admin_Executive(String admin_Executive_ID, String name, char gender, String contact_Number) + unit_Add(Unit) + unit_Modify(Unit, String) + unit_Delete(String) + unit_View_All(): ArrayList&lt;Unit&gt; + resident_Add(Resident) + resident_Modify(Resident, String) + resident_Delete(String) + resident_View_All(String): ArrayList&lt;Resident&gt; + search_Resident_Info(String: Resident + complaint_Add(Complaint) + complaint_View_All(): ArrayList&lt;Complaint&gt; + complaint_Update(Complaint, String) + complaint_Delete(String) + add_Employee(Employee) + view_All_Employee():ArrayList&lt;Employee&gt; + delete_Employee(String) + update_Employee(Employee, String) + add_Facility(Facility) + delete_Facility(String) + update_Facility(Facility, String) + view_All_Facility(): ArrayList&lt;Facility&gt; + add_Booking(Facility,Booking): Boolean + delete_Booking(String) + facility_Booking_Management_Update(Facility,Booking , String) + view_All_Booking(): ArrayList&lt;Facility.Booking&gt; + getArrayList(): ArrayList&lt;Admin_Executive&gt; + save_All_Admin_Executive(ArrayList&lt;Admin_Executive&gt;) + getDataString(Admin_Executive): String + get_Admin_Executive_Info(String): Admin_Executive + check_Admin_Executive_Availability(String): Boolean + get_All_Disapproved_Visitor_Pass(): ArrayList&lt;Visitor_Pass&gt; + approve_Visitor_Pass(String) </pre>

### 4.2.4 Building Executive Function

Building_Executive_Function
<pre> - position: String - building_Executive_Information_txt: File - admin_Executive_Infor_txt: File  + Building_Executive() + Building_Executive(String executiveID, String name, char gender, String contact_number) + getArrayList(): ArrayList&lt;Building_Executive&gt; + save_All_Building_Executive(ArrayList&lt;Building_Executive&gt;) + getDataString(Building_Executive): String + get_Building_Executive_Info(String): Building_Executive + check_Building_Executive_Availability(String): Boolean + add_Employee_Task(Employee_Task) + delete_Employee_Task(String) + update_Employee_Task(Employee_Task, String) + view_All_Employee_Task(): ArrayList&lt;Employee_Task&gt; + view_All_Complaint(): ArrayList&lt;Complaint&gt; + mark_Complaint_Solved(String) + add_Patrolling_Schedule(Patrolling) + delete_Patrolling_Schedule(String) + modify_Patrolling_Schedule(Patrolling, String) + view_All_Patrol_Schedule(): ArrayList&lt;Patrolling&gt; + add_New_CheckPoint(CheckPoint) + delete_CheckPoint(String) + modify_CheckPoint(CheckPoint, String) + view_All_CheckPoint(): ArrayList&lt;CheckPoint&gt; </pre>

#### 4.2.5 Building Manager Function

Building_Manager_Function
<pre> - operationID: String - building_Manager_ID: String - operationTitle: String - description: String - budget_Amount: Integer - building_Manager_Info_txt: File  + Building_Manager() + Building_Manager(String buildingManagerID, String name, char gender, String contact_Number) getOperationID(): String + setOperationID(String) + getBuilding_Manager_ID(): String + setBuilding_Manager_ID(String) + getOperationTitle(): String + setOperationTitle(String) + getDescription(): String + setDescription(String) + getBudget_Amount(): Integer + setBudget_Amount(Integer) + getArrayList(): ArrayList&lt;Building_Manager&gt; + save_All_Admin_Executive(ArrayList&lt;Building_Manager&gt;) + getDataString(Building_Manager_Function.Building_Manager): String + search_Executive_Information(String): Building_Manager + check_Building_Manager_Availability(String): Boolean + add_Account_Executive(Account_Executive_Function.Account_Executive) + delete_Account_Executive(String) + modify_Account_Executive(Account_Executive_Function.Account_Executive) + add_Building_Executive(Building_Executive_Function.Building_Executive) + delete_Building_Executive(String) + modify_Building_Executive(Building_Executive_Function.Building_Executive, String) + get_Executive_Info(String): Executive + get_Building_Manager_Infor(String): Building_Manager - get_Fund_Amount(): Integer </pre>

#### 4.2.6 Team Leader

Team_Leader
<pre> - leader_Username: String - team: String - position: String - team_Leader_Info_txt: File  + getLeader_Username(): String + setLeader_Username(String) + getTeam(): String + setTeam(String) + getPosition(): String + setPosition(String) + getArrayList(): ArrayList&lt;Team_Leader&gt; + save_All_Team_Leader(ArrayList&lt;Team_Leader&gt;) + getDataString(Team_Leader): String + search_Team_Leader_Information(String): Team_Leader + check_Team_Leader_Availability(String): Boolean + add_Team_Leader(Team_Leader) + delete_Team_Leader(String) + modify_Team_Leader(Team_Leader) </pre>

#### 4.2.7 Operation

Team_Leader
<pre> - leader_Username: String - team: String - position: String - team_Leader_Info_txt: File  + getLeader_Username(): String + setLeader_Username(String) + getTeam(): String + setTeam(String) + getPosition(): String + setPosition(String) + getArrayList(): ArrayList&lt;Team_Leader&gt; + save_All_Team_Leader(ArrayList&lt;Team_Leader&gt;) + getDataString(Team_Leader): String + search_Team_Leader_Information(String): Team_Leader + check_Team_Leader_Availability(String): Boolean + add_Team_Leader(Team_Leader) + delete_Team_Leader(String) + modify_Team_Leader(Team_Leader) </pre>

#### 4.2.8 Employee

Employee
<pre> # employeeID: String # name: String # gender: Characters # contact_Number: String # salary: Integer # position_Name: String  + Employee() + getEmployeeID(): String + setEmployeeID(String) + getName(): String + setName(String) + getGender(): Characters + setGender(Characters) + getContact_Number(): String + setContact_Number(String) + getSalary(): Integer + setSalary(Integer) + getPosition_Name(): String + setPosition_Name(String) + view undone_Task(String): ArrayList&lt;Employee_Task&gt; + mark_Task_Done(String) + check_Employee_Position(String): Integer + search_Employee_Infor(String): Employee </pre>

#### 4.2.9 Technician

Technician
<pre> # position: String - technician_Info_txt: File  + Technician() + Technician(String employeeID) + Technician(String employeeID, String name, char gender, String contact_Number, int sal + check_Technician_Availability(String): Boolean + getDataString(Technician): String + save_All_Technician(ArrayList&lt;Technician&gt;) + get_Technician_Info(String): Technician </pre>

#### 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 salary) + 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

Patrolling
<pre> - patrolID: String - employeeID: String - day: String - start_Time: LocalTime - end_Time: LocalTime - patrolling_Schedule_Info_txt: File  + Patrolling() + Patrolling(String patrolID, String employeeID, String day, LocalTime start_Time, LocalTime end_Time) + getPatrolID(): String + setPatrolID(String) + getEmployeeID(): String + setEmployeeID(String) + getDay(): String + setDay(String) + getStart_Time(): LocalTime + setStart_Time(LocalTime) + getEnd_Time(): LocalTime + setEnd_Time(LocalTime) + getArrayList(): ArrayList&lt;Patrolling&gt; + check_Patrolling_Existence(String): Boolean + getDataString(Patrolling): String + save_All_Patrolling(ArrayList&lt;Patrolling&gt;) + get_SG_All_Patrolling(String): ArrayList&lt;Patrolling&gt; + check_TimeSlot_Availability(Patrolling): Boolean + get_Auto_PatrollingID(): String </pre>

### 4.2.14 Check Point

CheckPoint
<pre> - checkPointID: String - name: String - location: String - checkPoint_Info_txt: File - checkPoint_Record_txt: File  + CheckPoint() + CheckPoint(String checkPointID, String name, String location) + getCheckPointID(): String + setCheckPointID(String) + getName(): String + setName(String) + getLocation(): String + setLocation(String) + getArrayList(): ArrayList&lt;CheckPoint&gt; + check_CheckPoint_Existence(String): Boolean + getDataString(CheckPoint): String + save_All_CheckPoint(ArrayList&lt;CheckPoint&gt;) </pre>

## 4.2.15 Incident

Incident
<pre> - incident_ID: String - dateTime: LocalDateTime - security_Guard_ID: String - description: String - incident_Report: File  + public Incident() + Incident(String incident_ID, LocalDateTime dateTIme, String security_Guard_ID, String description) + getIncident_ID(): String + setIncident_ID(String) + getDateTIme(): LocalDateTime + setDateTIme(LocalDateTime) + getSecurity_Guard_ID(): String + setSecurity_Guard_ID(String) + getDescription(): String + setDescription(String) + add_Incident_Record(Incident) + delete_Incident_Record(String) + update_Incident_Report(Incident, String) + save_All_Incident_Report(ArrayList&lt;Incident&gt;) + getDataString(Incident) + get_all_Incident_Report(): ArrayList&lt;Incident&gt; </pre>

## 4.2.16 Resident

Resident
<pre> - resident_Username: String - name: String - gender : Characters - contact_Number: String - unitID: String - payment: Integer - Resident_Information_txt: File  + Resident() + Resident(String resident_Username, String name, char gender, String contact_Number, String unitID, int payment) + getResident_Username(): String + setResident_Username(String) + getName(): String + setName(String) + getGender(): Characters + setGender(Characters) + getContact_Number(): String + setContact_Number(String) + getUnitID(): String + setUnitID(String) + getPayment(): Integer + setPayment(Integer) + get_Resident_Info(String): Resident + update_Resident_Info(Resident, String) + getArrayList(): ArrayList&lt;Resident&gt; + save_All_Resident(ArrayList&lt;Resident&gt;) + getDataString(Resident): String + check_Resident_Availability(String): Boolean + get_Unit_All_Unpaid_Invoice(String): ArrayList&lt;Invoice&gt; + get_Unpaid_Amount(String, String): Integer + make_Payment(Invoice, String) + get_All_Receipt(String): ArrayList&lt;Payment&gt; + get_All_pending_Payment(String): ArrayList&lt;Payment&gt; + get_Statement_for_Resident(String): ArrayList&lt;Statement&gt; + add_Facility_Booking(Facility.Booking) + view_Resident_Booking(String): ArrayList&lt;Facility.Booking&gt; + cancel_Facility_Booking(String) + update_Facility_Booking(Facility.Booking, String) + apply_Visitor_Pass(Visitor_Pass) + view_All_Visitor_Pass_Apply(String): ArrayList&lt;VisitorPass&gt; + cancel_Visitor_Pass(String) + update_Visitor_Pass(Visitor_Pass) + log_Complaint(Complaint) + view_Complaint(String): ArrayList&lt;Complaint&gt; + update_Complaint(Complaint, String) + cancel_Complaint(String) </pre>

#### 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

Login	
- username: String	
- password: String	
- login_Credentials_txt: File	
+ Login()	
+ Login(String username, String password)	
+ getUsername(): String	
+ setUsername(String)	
+ getPassword(): String	
+ setPassword(String)	
+ check_Punctuation(String): Boolean	
+ getArrayList(): ArrayList<Login>	
+ save_All_Login(ArrayList<Login>)	
+ getDataString(Login): String	
+ search_Login_Information(String): Login	
+ get_Login_Credentials(String): String	
+ check_Login_Availability(String): Boolean	
+ add_Login(Login)	
+ delete_Login(String)	
+ modify_Login(Login)	
+ check_User_Types(String): Integer	
+ check_Username_Availability(String): Boolean	
+ login(Login): Boolean	
+ register(Login): Boolean	

#### 4.2.19 Payment

Payment	
- paymentID: String	
- invoiceID: String	
- pay_Username: String	
- unitID: String	
- amount: Integer	
- paymentDate: LocalDate	
- paymentType: String	
- issuerID: String	
- issuedDate: LocalDate	
- description: String	
- payment_txt: File	
+ Payment()	
+ Payment(String paymentID, String invoiceID, String pay_Username, String unitID, int amount, LocalDate paymentDate, String paymentTypes, String issuerID, LocalDate issuedDate, String description)	
+ getPaymentID(): String	
+ setPaymentID(String)	
+ getInvoiceID(): String	
+ setInvoiceID(String)	
+ getPay_Username(): String	
+ setPay_Username(String)	
+ getUnitID(): String	
+ setUnitID(String)	
+ getAmount(): Integer	
+ setAmount(Integer)	
+ getPaymentDate(): LocalDate	
+ setPaymentDate(LocalDate)	
+ getPaymentType(): String	
+ setPaymentType(String)	
+ getIssuerID(): String	
+ setIssuerID(String)	
+ getIssuedDate(): LocalDate	
+ setIssuedDate(LocalDate)	
+ getDescription(): String	
+ setDescription(String)	
+ getArrayList(): ArrayList<Payment>	
+ getStringArray(Payment): String[]	
+ get_auto_PaymentID(): String	
+ getDataString(Payment): String	
+ save_All_Payment(ArrayList<Payment>)	
+ make_Payment(Invoice, String)	
- get_Fund_Amount(): Integer	
- add_to_Fund(Integer)	
+ get_All_unapproved_Payment(): ArrayList<Payment>	
+ get_All_Receipt(String): ArrayList<Payment>	

#### 4.2.20 Invoice

Invoice
<pre> - invoiceID: String - issuerID: String - unitID: String - amount: Integer - dueDate: LocalDate - paymentType: String - description: String - status: String - invoice_txt: File  + Invoice() + Invoice(String invoiceID, String issuerID, String unitID, int amount, LocalDate dueDate, String paymentTypes, String description, String status) + getInvoiceID(): String + setInvoiceID(String) + getIssuerID(): String + setIssuerID(String) + getUnitID(): String + setUnitID(String) + getAmount(): Integer + setAmount(Integer) + getDueDate(): LocalDate + setDueDate(LocalDate) + getPaymentType(): String + setPaymentType(String) + getDescription(): String + setDescription(String) + getStatus(): String + setStatus(String) + getArrayList(): ArrayList&lt;Invoice&gt; + get_Auto_InvoiceID(): String + check_Invoice_Availability(String): Boolean + getStringArray(Invoice): String[] + getDataString(Invoice): String + save_All_Invoice(ArrayList&lt;Invoice&gt; </pre>

#### 4.2.21 Deposit

Deposit
<pre> - username: String - amount: Integer - unitID: String - deposit_txt: File  + Deposit() + Deposit(String username, int amount, String unitID) + getUsername(): String + setUsername(String) + getAmount(): Integer + setAmount(Integer) + getUnitID(): String + setUnitID(String) + getArrayList(): ArrayList&lt;Deposit&gt; + getDataString(Deposit): String + save_All_Deposit(ArrayList&lt;Deposit&gt;) + make_Deposit(Deposit) + check_Deposit(String): Deposit </pre>

## 4.2.22 Statement

Statement
<pre> - invoiceID: String - issuerID: String - unitID: String - amount: Integer - dueDate: LocalDate - paymentType: String - description: String - status: String - statement_txt: File  + Statement() + Statement(String statementID, String issuer_Position, LocalDate date, String description, String receiverID) + getInvoiceID(): String + setInvoiceID(String) + getIssuerID(): String + setIssuerID(String) + getUnitID(): String + setUnitID(String) + getAmount(): Integer + setAmount(Integer) + getDueDate(): LocalDate + setDueDate(LocalDate) + getPaymentType(): String + setPaymentType(String) + getDescription(): String + setDescription(String) + getStatus(): String + setStatus(String) + getArrayList(): ArrayList&lt;Statement&gt; + get_Auto_StatementID(): String + getDataString(Statement): String + save_All_Statement(ArrayList&lt;Statement&gt;) + get_Statement_for_Receiver(String): ArrayList&lt;Statement&gt; </pre>

## 4.2.23 Unit

Unit
<pre> - floor: Integer - unitID: String - complete_Year: Integer - furnish: String - parking_Unit: String - owner_Username: String - resident_Username: String - unit_Information_txt: File  + Unit() + Unit(Integer floor, String unitID, int complete_Year, String furnish, String parking_Unit, String owner_Username, String resident_Username) + getFloor(): Integer + setFloor(Integer) + getUnitID(): String + setUnitID(String) + getComplete_Year(): Integer + setComplete_Year(Integer) + getFurnish(): String + setFurnish(String) + getParking_Unit(): String + setParking_Unit(String) + getOwner_Username(): String + setOwner_Username(String) + getResident_Username(): String + setResident_Username(String) + getArrayList(): ArrayList&lt;Unit&gt; + sort_All_Unit() + switch_Unit(ArrayList&lt;Unit&gt;, Integer): ArrayList&lt;Unit&gt; + save_All_Unit(ArrayList&lt;Unit&gt;) + getDataString(Unit): String + search_Unit_Information(String): Unit + check_Unit_Availability(String): Boolean </pre>

#### 4.2.24 Facility

Facility
<pre>- facilityID: String - name: String - facility_Info_txt: File  + getFacilityID(): String + setFacilityID(String) + getName(): String + setName(String) + getArrayList(): ArrayList&lt;Facility&gt; + check_Facility_Availability(String): Boolean + getDataString(Facility): String + save_All_Facility(ArrayList&lt;Facility&gt;)</pre>

#### 4.2.25 Booking

Booking
<pre>- bookingID: String - facilityID: String - resident_Username: String - date: LocalDate - start_Time: LocalTime - end_Time: LocalTime - facility_Booking_Record_txt: File  + getBookingID(): String + setBookingID(String) + getFacilityID(): String + setFacilityID(String) + getResident_Username: String + setResident_Username(String) + getDate: LocalDate + setDate(LocalDate) + getStart_Time(): LocalTime + setStart_Time(LocalTime) + getEnd_Time(): LocalTime + setEnd_Time(LocalTime) + getArrayList(): ArrayList&lt;Facility.Booking&gt; + check_Facility_Existence(String): Boolean + getDataString(Facility.Booking): String + save_All_facility_booking(ArrayList&lt;Facility.Booking&gt;) + getFacilityAllBooking(String): ArrayList&lt;Facility.Booking&gt; + check_TimeSlot_Availability(Facility.Booking): Boolean + get_Auto_BookingID(): String</pre>

#### 4.2.26 Visitor Pass

Visitor_Pass	
- visitor_Pass_ID: String	
- visitor_name: String	
- resident_Username: String	
- unitID: String	
- gender: Characters	
- contact_Number: String	
- date_Start: LocalDate	
- date_End: LocalDate	
- status: String	
- visotr_Pass_Info_txt: File	
+ Visitor_Pass()	
+ Visitor_Pass(String visitor_Pass_ID, String visitor_Name,	
String resident_Username, String unitID, char gender,	
String contact_Number, LocalDate date_Start,	
LocalDate date_End, String status)	
+ getVisitor_Pass_ID(): String	
+ setVisitor_Pass_ID(String)	
+ getVisitor_Name(): String	
+ setVisitor_Name(String)	
+ getResident_Username(): String	
+ setResident_Username(String)	
+ getUnitID(): String	
+ setUnitID(String)	
+ getGender(): Characters	
+ setGender(Characters)	
+ getContact_Number(): String	
+ setContact_Number(String)	
+ getDate_Start(): LocalDate	
+ setDate_Start(LocalDate)	
+ getDate_End(): LocalDate	
+ setDate_End(LocalDate)	
+ getArrayList(): ArrayList<Visitor_Pass>	
+ check_Visitor_Pass_Availability(String): Boolean	
+ check_Visitor_Pass_Validity(String): Boolean	
+ getDataString(Visitor_Pass): String	
+ save_All_Visitor(ArrayList<Visitor_Pass>)	
+ search_Visitor_Pass_Info(String): Visitor_Pass	
+ search_Visitor_Pass_Info_by_Name(String): Visitor_Pass	
+ get_Auto_Visitor_Pass_ID(): String	

## 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	<ol style="list-style-type: none"> <li>1. Insert user ID and password.</li> <li>2. Click “Login” button.</li> <li>3. System will check the user ID and password if match in accordance with the text file.</li> <li>4. “Login Successfully” message will be sent out when information entered is correct.</li> </ol>
Alternative Path	
Exceptional Path	<p>At basic path 3, the system displays error message to let user type an accurate user ID and password.</p> <p>At basic path 4, the system displays “Please try again” when information is wrong.</p>

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	<ol style="list-style-type: none"> <li>1. Login to the system.</li> </ol>

	<ol style="list-style-type: none"> <li>2. Click the “Logout” button.</li> <li>3. Send confirmation message.</li> <li>4. Logout successfully.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the invoices displayed in the table.</li> <li>2. Click on the “View invoice” button.</li> <li>3. The system will display information regarding the invoice the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Issue invoice” button.</li> <li>2. Fill in the details and click the “Issue invoice” button.</li> <li>3. System will add the issued invoice into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the receipts displayed in the table.</li> <li>2. Click on the “View receipt” button.</li> <li>3. The system will display information regarding the receipt the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the statements displayed in the table.</li> <li>2. Click on the “View statement” button.</li> <li>3. The system will display information regarding the statement the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Issue statement” button.</li> <li>2. Fill in the details and click the “Issue statement” button.</li> <li>3. System will add the issued statement in to a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the payments displayed in the table.</li> <li>2. Click on the “View payment” button.</li> <li>3. The system will display information regarding the payment the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the payments' statuses displayed in the table.</li> <li>2. Click on the “Confirm payment” button.</li> <li>3. The system will then send a message notifying regarding the change of status.</li> </ol>
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	<ol style="list-style-type: none"><li>1. Click on one of the pending fees displayed in the table.</li><li>2. Click on the “View pending fee” button.</li><li>3. The system will display information regarding the pending fee the user clicked on.</li></ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the deposits displayed on the table.</li> <li>2. Click on the “View deposit” button.</li> <li>3. The system will display information regarding the deposit the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new unit” button.</li> <li>2. Fill in the details and click the “Add new unit” button.</li> <li>3. System will add the newly added unit into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Modify Unit Info” button.</li> <li>2. Fill in the details and click the “Add new unit” button.</li> <li>3. System will add the modified unit onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the unit data displayed on the table.</li> <li>2. Click on the “Delete Unit” button.</li> <li>3. System will notify users regarding confirmation before officially deleting the unit info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the unit details displayed on the table.</li> <li>2. Click on the “View unit details” button.</li> <li>3. The system will display information regarding the unit details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new resident” button.</li> <li>2. Fill in the details and click the “Add new resident” button.</li> <li>3. The system will notify the user once the addition of a new resident is completed.</li> </ol>

	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	<ol style="list-style-type: none"> <li>1. Choose any resident info that the user wish to modify.</li> <li>2. Click on the “Modify Resident Info” button.</li> <li>3. Fill in the details and click the “Add new resident” button.</li> <li>4. System will add the modified resident onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the resident data displayed on the table.</li> </ol>

	<ol style="list-style-type: none"> <li>2. Click on the “Delete Resident” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the resident info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the resident details displayed on the table.</li> <li>2. Click on the “View resident details” button.</li> <li>3. The system will the display information regarding the resident details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new complaint” button.</li> <li>2. Fill in the details and click the “Add new complaint” button.</li> <li>3. The system will notify the user once the data for the new complaint is added.</li> <li>4. System will add the newly added complaint into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the complaints the user wishes to modify.</li> <li>2. Click on the “Modify Complaint Info” button.</li> <li>3. Fill in the details and click the “Add new complaint” button.</li> <li>4. System will add the modified complaint onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the complaint data displayed on the table.</li> <li>2. Click on the “Delete Complaint” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the complaint info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the complaint details displayed on the table.</li> <li>2. Click on the “View complaint details” button.</li> <li>3. The system will display information regarding the complaint details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the complaints' statuses displayed in the table.</li> <li>2. Click on the "Complaint solved" button.</li> <li>3. The user will send to ask for confirmation before changing the complaint status.</li> <li>4. The system will then send a message notifying regarding the change of status.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the "Add new employee" button.</li> </ol>

	<ol style="list-style-type: none"> <li>2. The user is required to add and specify the type of employee the user wishes to add.</li> <li>3. Fill in the details and click the “Add new employee” button.</li> <li>4. The user will be notified once the addition of a new employee is completed.</li> <li>5. System will add the newly added employee into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the employee info the user wishes to make changes to.</li> <li>2. Click on the “Modify Employee Info” button.</li> <li>3. Fill in the details and click the “Add new employee” button.</li> <li>4. System will add the modified employee onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the employee data displayed on the table.</li> <li>2. Click on the “Delete Employee” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the employee info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the employee details the user wishes to view.</li> <li>2. Click on one of the employee details displayed on the table.</li> <li>3. Click on the “View employee details” button.</li> <li>4. The system will the display information regarding the employee details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new facility” button.</li> <li>2. Fill in the details and click the “Add new facility” button.</li> <li>3. System will add the newly added facility into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the facility info the users wish to modify.</li> <li>2. Click on the “Modify Facility Info” button.</li> <li>3. Fill in the details and click the “Add new facility” button.</li> <li>4. System will add the modified facility onto the rows of data.</li> </ol>

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	<ol style="list-style-type: none"> <li>1. Click on one of the facility data displayed on the table.</li> <li>2. Click on the “Delete Facility” button.</li> <li>3. System will notify users regarding confirmation before officially deleting the facility info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the facility details displayed on the table.</li> <li>2. Click on the “View facility details” button.</li> </ol>

	3. The system will 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	<ol style="list-style-type: none"> <li>1. Click on the “Add new facility booking” button.</li> <li>2. Fill in the details and click the “Add new facility booking” button.</li> <li>3. The system will notify the user once the addition of a new booking is added.</li> <li>4. System will add the newly added facility booking into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the facilities booking info the user wish to make changes to.</li> <li>2. Click on the “Modify Facility booking Info” button.</li> <li>3. Fill in the details and click the “Add new facility booking” button.</li> <li>4. System will add the modified facility booking onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the facilities booking data displayed on the table.</li> <li>2. Click on the “Delete Facility booking” button.</li> </ol>

	<p>3. The system will notify users regarding confirmation before officially deleting the facility booking info.</p> <p>4. System will delete the info from the rows of data.</p>
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	<p>1. Click on one of the facilities booking details displayed on the table.</p> <p>2. Click on the “View facility booking details” button.</p> <p>3. The system will the display information regarding the facility booking details the user clicked on.</p>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new vendor” button.</li> <li>2. Fill in the details and click the “Add new vendor” button.</li> <li>3. The system will notify users regarding the addition of a new vendor.</li> <li>4. System will add the newly added vendor into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the vendor info the user wish to modify.</li> <li>2. Click on the “Modify Vendor Info” button.</li> <li>3. Fill in the details and click the “Add new vendor” button.</li> <li>4. System will add the modified vendor onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the vendor data displayed on the table.</li> <li>2. Click on the “Delete Vendor” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the vendor info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the vendor details displayed on the table.</li> <li>2. Click on the “View vendor details” button.</li> <li>3. The system will the display information regarding the vendor details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the visitors pass application statuses displayed in the table.</li> <li>2. Click on the “Approve visitor pass” button.</li> <li>3. The system will then send a message notifying regarding the change of status.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the visitor pass details displayed on the table.</li> <li>2. Click on the “View visitor pass” button.</li> <li>3. The system will the display information regarding the visitor pass details the user clicked on.</li> </ol>

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	<ol style="list-style-type: none"> <li>1. Click on the “Add new employee task” button.</li> <li>2. Fill in the details and click the “Add new employee task” button.</li> <li>3. The system will send a notification once the addition of a new employee task is added.</li> <li>4. System will add the newly added employee task into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the employee task the user wish to modify.</li> <li>2. Click on the “Modify Employee task Info” button.</li> <li>3. Fill in the details and click the “Add new employee task” button.</li> <li>4. System will add the modified employee task onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the employee task data displayed on the table.</li> <li>2. Click on the “Delete Employee task” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the employee task info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the employee task details displayed on the table.</li> <li>2. Click on the “View employee task details” button.</li> <li>3. The system will display information regarding the employee task details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new patrolling” button.</li> <li>2. Fill in the details and click the “Add new patrolling” button.</li> </ol>

	<p>3. The system will then notify users once a new patrolling schedule is set up.</p> <p>4. System will add the newly added patrolling into a new row of data.</p>
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	<ol style="list-style-type: none"> <li>Choose one of the patrolling info the user wishes to modify.</li> <li>Click on the “Modify Patrolling Info” button.</li> <li>Fill in the details and click the “Add new patrolling” button.</li> <li>System will add the modified patrolling onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the patrolling data displayed on the table.</li> <li>2. Click on the “Delete Patrolling” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the patrolling info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the patrolling details displayed on the table.</li> <li>2. Click on the “View patrolling details” button.</li> <li>3. The system will the display information regarding the patrolling details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new checkpoint” button.</li> <li>2. Fill in the details and click the “Add new checkpoint” button.</li> <li>3. System will add the newly added checkpoint into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Modify Checkpoint Info” button.</li> <li>2. Fill in the details and click the “Add new checkpoint” button.</li> <li>3. System will add the modified checkpoint onto the rows of data.</li> </ol>

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	<ol style="list-style-type: none"> <li>1. Click on one of the checkpoint data displayed on the table.</li> <li>2. Click on the “Delete Checkpoint” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the checkpoint info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the checkpoint details displayed on the table.</li> <li>2. Click on the “View checkpoint details” button.</li> </ol>

	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	<ol style="list-style-type: none"> <li>1. Click on one of the checkpoint records displayed on the table.</li> <li>2. Click on the “View checkpoint record” button.</li> <li>3. The system will the display information regarding the checkpoint record the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the entry records displayed on the table.</li> <li>2. Click on the “View entry record” button.</li> <li>3. The system will display information regarding the entry record the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the incident reports displayed on the table.</li> <li>2. Click on the “View incident report” button.</li> <li>3. The system will display information regarding the incident report the user clicked on.</li> </ol>

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	<ol style="list-style-type: none"> <li>1. Click on one of the report details displayed on the table.</li> <li>2. Click on the “View report details” button.</li> <li>3. The system will display information regarding the report details the report clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new user” button.</li> <li>2. Fill in the details and click the “Add new user” button.</li> </ol>

	<p>3. The system will then notify the user regarding the addition of a new user.</p> <p>4. System will add the newly added user into a new row of data.</p>
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	<p>1. Choose one of the user, the user wishes to modify.</p> <p>2. Click on the “Modify User Info” button.</p> <p>3. Fill in the details and click the “Add new user” button.</p> <p>4. System will add the modified user onto the rows of data.</p>
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	<p>1. Click on one of the user data displayed on the table.</p> <p>2. Click on the “Delete User” button.</p>

	<p>3. The system will notify users regarding confirmation before officially deleting the user info.</p> <p>4. System will delete the info from the rows of data.</p>
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	<ol style="list-style-type: none"> <li>1. Click on one of the user details displayed on the table.</li> <li>2. Click on the “View user details” button.</li> <li>3. The system will the display information regarding the user details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the search bar.</li> </ol>

	<ol style="list-style-type: none"> <li>2. Type in the username for the user wants to find.</li> <li>3. Click on which user the user wants to search for.</li> <li>4. The system will display information regarding the user details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new operation budgeting” button.</li> <li>2. Fill in the details and click the “Add new operation budgeting” button.</li> <li>3. System will add the newly added operation budgeting into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the operation budgeting, the user wishes to modify.</li> <li>2. Click on the “Modify Operation budgeting Info” button.</li> <li>3. Fill in the details and click the “Add new operation budgeting” button.</li> <li>4. System will add the modified operation budgeting onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the operation budgeting data displayed on the table.</li> <li>2. Click on the “Delete Operation budgeting” button.</li> </ol>

	<p>3. The system will notify operation budgeting regarding confirmation before officially deleting the operation budgeting info.</p> <p>4. System will delete the info from the rows of data.</p>
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	<ol style="list-style-type: none"> <li>1. Click on one of the operation budgeting details displayed on the table.</li> <li>2. Click on the “View operation budgeting details” button.</li> <li>3. The system will display information regarding the operation budgeting details.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the operation budgeting details displayed on the table.</li> <li>2. Click on the “Fund operation” button.</li> <li>3. The system will notify user once the transaction is completed.</li> <li>4. The system will display information regarding the change of operation budgeting details.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Add new team leader” button.</li> <li>2. Fill in the details and click the “Add new team leader” button.</li> <li>3. System will add the newly added team leader into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Modify Team leader Info” button.</li> <li>2. Fill in the details and click the “Add new team leader” button.</li> <li>3. System will add the modified team leader onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the team leader data displayed on the table.</li> <li>2. Click on the “Delete Team leader” button.</li> <li>3. The system will notify team leader regarding confirmation before officially deleting the team leader info.</li> </ol>

	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	<ol style="list-style-type: none"> <li>1. Click on one of the team leader details displayed on the table.</li> <li>2. Click on the “View team leader details” button.</li> <li>3. The system will the display information regarding the team leader details the team leader clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Update Visitor pass Info” button.</li> <li>2. Fill in the details and click the “Update new visitor pass” button.</li> <li>3. System will add the modified visitor pass onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Check-In Check Point” button.</li> <li>2. Click on the checkpoint user wants to check in to.</li> <li>3. System will notify once the user has check-in to the specific checkpoint.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the checkpoint details displayed on the table.</li> <li>2. Click on the “View checkpoint details” button.</li> <li>3. The system will display information regarding the checkpoint details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Record incident” button.</li> <li>2. Fill in the details and click the “Record incident” button.</li> <li>3. System will add the newly recorded incident into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Update incident Info” button.</li> <li>2. Fill in the details and click the “Update incident” button.</li> <li>3. System will add the modified incident onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the incident details displayed on the table.</li> <li>2. Click on the “Check incident details” button.</li> <li>3. The system will display information regarding the checkpoint details the user clicked on.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the payment details displayed on the table.</li> <li>2. Click on the “Make payment” button.</li> <li>3. The system will notify the user of the payment made.</li> <li>4. The payment status will change from “UNPAID” to ‘PAID’.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Check deposit payment” button.</li> <li>2. Payment will be made immediately.</li> <li>3. The system will then notify users regarding the payment they made.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Book facility” button.</li> <li>2. Fill in the details and click the “Book facility” button.</li> <li>3. System will add the newly added facility booking into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Update booking” button.</li> <li>2. Fill in the details and click the “Update booking” button.</li> <li>3. System will add the modified facility booking onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the facilities booking data displayed on the table.</li> <li>2. Click on the “Cancel booking” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the facility booking info.</li> <li>4. System will delete the info from the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on the “Apply visitor pass” button.</li> <li>2. Fill in the details and click the “Apply visitor pass” button.</li> <li>3. System will add the new application of visitor pas into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose which visitor passes the user wish to update.</li> <li>2. Click on the “Update visitor pass” button.</li> <li>3. Fill in the details and click the “Update visitor pass” button.</li> <li>4. System will add the modified visitor pass onto the rows of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Click on one of the visitor data displayed on the table.</li> <li>2. Click on the “Cancel visitor pass” button.</li> <li>3. The system will notify users regarding confirmation before officially deleting the visitor pass.</li> <li>4. System will delete the info from the rows of data.</li> </ol>

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	<ol style="list-style-type: none"> <li>1. Click on the “Log complaint” button.</li> <li>2. Fill in the details and click the “Log complaint” button.</li> <li>3. System will add the new complaint into a new row of data.</li> </ol>
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	<ol style="list-style-type: none"> <li>1. Choose one of the complaint the user wish to update.</li> <li>2. Click on the “Update complaint” button.</li> <li>3. Fill in the details and click the “Update complaint” button.</li> <li>4. System will add the modified complaint onto the rows of data.</li> </ol>

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	<ol style="list-style-type: none"><li>1. Click on one of the complaints displayed on the table.</li><li>2. Click on the “Cancel complaint” button.</li><li>3. The system will notify users regarding confirmation before officially deleting the complaint.</li><li>4. System will delete the info from the rows of data.</li></ol>
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)

```
1 private String patrolID;
2 private String employeeID;
3 private String day;
4 private LocalTime start_Time;
5 private LocalTime end_Time;
6 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)

```
1 package Entity.Employee;
2
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import java.util.ArrayList;
6 import java.util.HashMap;
7 import java.util.Iterator;
8
```

```
1 public clas Employee {
2     protected String employeeID;
3     protected String name;
4     protected char gender;
5     protected String contact_Number;
6     protected int salary;
7     protected String position_Name;
8
9     public Employee() {
10 }
```

```
1 public void set_Info(String employeeID, String name, char gender, String contact_Number, int salary, String position) {
2     this.employeeID = employeeID;
3     this.name = name;
4     this.gender = gender;
5     this.contact_Number = contact_Number;
6     this.position_Name = position;
7 }
```

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)

```
1  public String get_Auto_InvoiceID() throws FileNotFoundException {
2      FileReader fileReader = new FileReader(this.invoice_txt);
3      Scanner scanner = new Scanner(fileReader);
4      scanner.nextLine();
5
6      Integer num;
7      for(num = 0; scanner.hasNextLine(); num = num + 1) {
8          String[] data = scanner.nextLine().split(":", 8);
9          String number = data[0].substring(3);
10         num = Integer.parseInt(number);
11     }
12
13     String str = "INV" + num.toString();
14     System.out.println(str);
15     return str;
16 }
17
```

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.



The screenshot shows a code editor window with a dark theme. At the top left are three colored window control buttons: red, yellow, and green. The code editor displays the following Java code:

```
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.

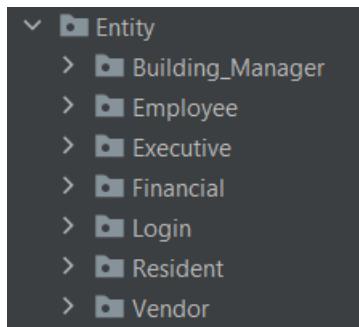
The setExecutiveID method sets the executive's ID, while the getName method returns the executive's name. Similarly, setContact\_Number sets the executive's contact number, and getGender returns the executive's gender. These operations allow for accessing and modifying the attributes of an executive 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.

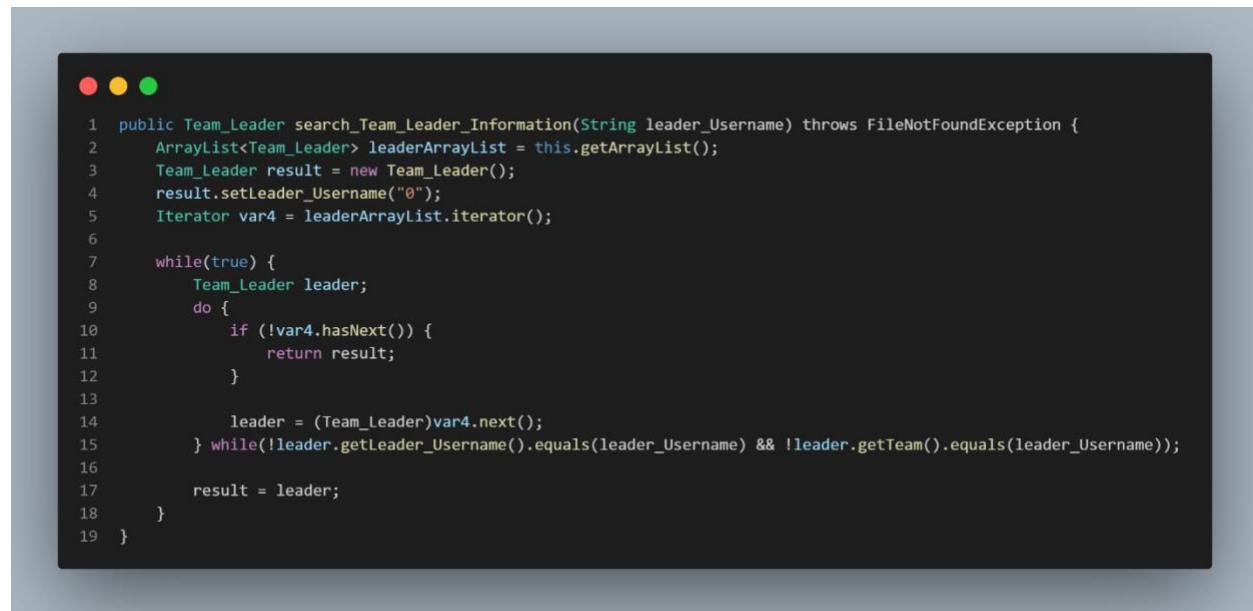
In summary, the package statement is a fundamental aspect of OOP that enables developers to organize their code more structurally, making it easier to manage and maintain large-scale software systems. Furthermore, by using packages, developers can create a logical grouping of classes, functions, and 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



```
1 public Team_Leader search_Team_Leader_Information(String leader_Username) throws FileNotFoundException {
2     ArrayList<Team_Leader> leaderArrayList = this.getArrayList();
3     Team_Leader result = new Team_Leader();
4     result.setLeader_Username("0");
5     Iterator var4 = leaderArrayList.iterator();
6
7     while(true) {
8         Team_Leader leader;
9         do {
10             if (!var4.hasNext()) {
11                 return result;
12             }
13
14             leader = (Team_Leader)var4.next();
15         } while(!leader.getLeader_Username().equals(leader_Username) && !leader.getTeam().equals(leader_Username));
16
17         result = leader;
18     }
19 }
```



```
1 public boolean check_Team_Leader_Availability(String team, String position) throws FileNotFoundException {
2     boolean result = false;
3     ArrayList<Team_Leader> leaderArrayList = this.getArrayList();
4     Iterator var5 = leaderArrayList.iterator();
5
6     while(var5.hasNext()) {
7         Team_Leader leader1 = (Team_Leader)var5.next();
8         if (leader1.getTeam().equals(team) && leader1.getPosition().equals(position)) {
9             result = true;
10        }
11    }
12 }
```

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:

```
1 public ArrayList<Statement> getArrayList() throws FileNotFoundException {
2     ArrayList<Statement> statementArrayList = new ArrayList();
3     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM.dd.yyyy");
4     Scanner scanner = new Scanner(this.statement_txt);
5     scanner.nextLine();
6
7     while(scanner.hasNextLine()) {
8         String[] data = scanner.nextLine().split(":", 5);
9         Statement statement = new Statement(data[0], data[1], LocalDate.parse(data[2], formatter), data[3], data[4]);
10        statementArrayList.add(statement);
11    }
12
13    scanner.close();
14    return statementArrayList;
15 }
```

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.

```
● ● ●  
1 public String getInvoiceID() { return this.invoiceID; }  
2  
3 public void setInvoiceID(String invoiceID) { this.invoiceID = invoiceID; }  
4  
5 public String getIssuerID() { return this.issuerID; }  
6  
7 public void setIssuerID(String issuerID) { this.issuerID = issuerID; }  
8  
9 public String getUnitID() { return this.unitID; }  
10  
11 public void setUnitID(String unitID) { this.unitID = unitID; }  
12  
13 public int getAmount() { return this.amount; }  
14  
15 public void setAmount(int amount) { this.amount = amount; }  
16  
17 public String getInvoiceID() { return this.invoiceID; }  
18  
19 public void setInvoiceID(String invoiceID) { this.invoiceID = invoiceID; }  
20  
21 public String getIssuerID() { return this.issuerID; }  
22  
23 public void setIssuerID(String issuerID) { this.issuerID = issuerID; }  
24  
25 public String getUnitID() { return this.unitID; }  
26  
27 public void setUnitID(String unitID) { this.unitID = unitID; }  
28  
29 public int getAmount() { return this.amount; }  
30  
31 public void setAmount(int amount) { this.amount = amount; }  
32  
33 public LocalDate getDueDate() { return this.dueDate; }  
34  
35 public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }  
36  
37 public LocalDate getDueDate() { return this.dueDate; }  
38  
39 public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }  
40
```

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



```
 1  public class Building_Executive_Function {
 2      public Building_Executive_Function() {
 3      }
 4
 5      public static class Building_Executive extends Executive {
 6          private String position = "Building Executive";
 7          private File building_Executive_Info_txt = new File("src/Database/Building_Executive_Information.txt");
 8
 9          public Building_Executive() {
10          }
11
12          public Building_Executive(String executiveID, String name, char gender, String contact_number) {
13              super.set_Info(executiveID, name, gender, contact_number, this.position);
14          }
15      }
16  }
```

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).

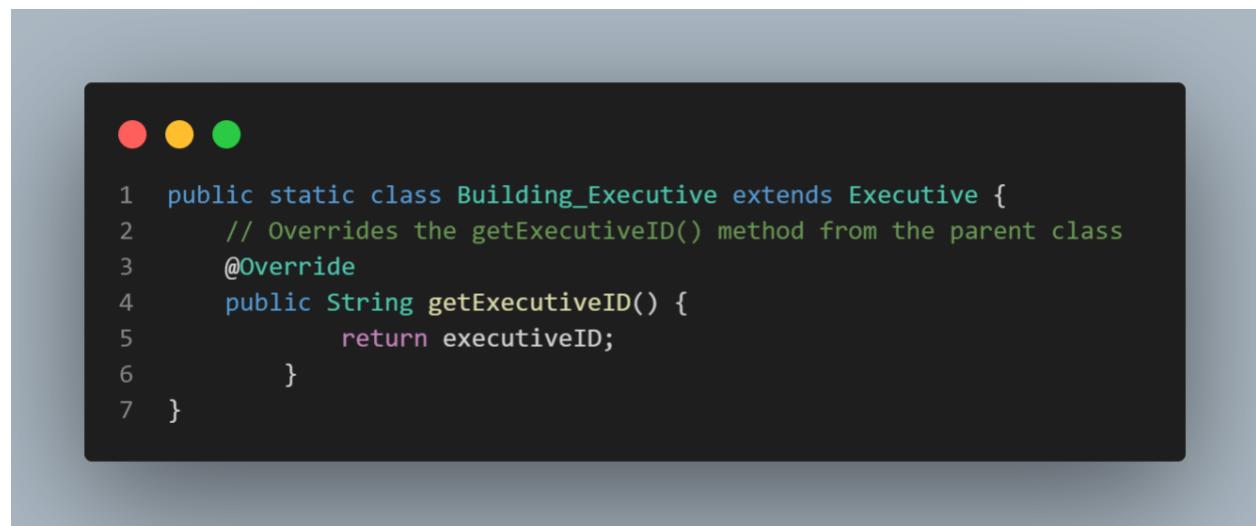
In conclusion, the implementation of inheritance in the provided code exemplifies the fundamental OOP concept of class inheritance, which is a powerful tool for developers to create hierarchical relationships between classes, reuse existing code, and enhance the organization and structure of software. 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 inheritance 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:



A screenshot of a Java code editor showing a code snippet. The code is a public static class named Building\_Executive that extends the Executive class. It overrides the getExecutiveID() method, which returns the executiveID attribute. The code is color-coded with syntax highlighting.

```
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  }
```

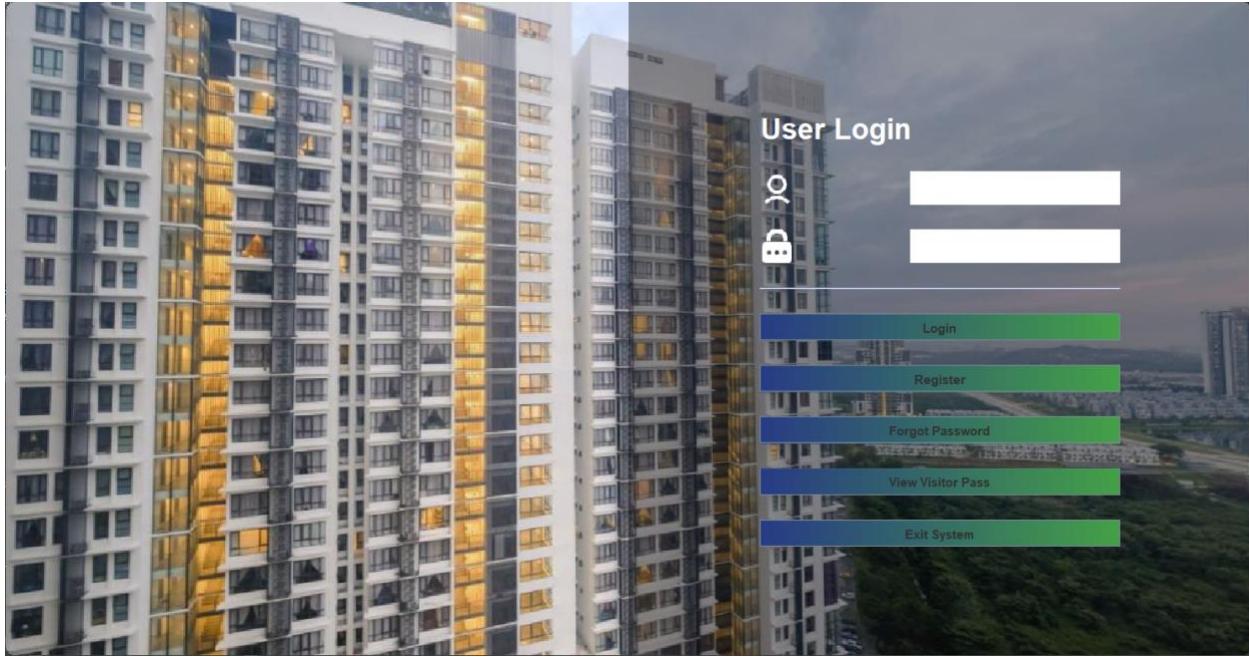
To further explain the implementation of polymorphism in the provided Java code, the Building\_Executive class is used to create objects of both its type and the parent class Executive type. This is because the Building\_Executive class inherits all the methods of the parent class Executive, including 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:

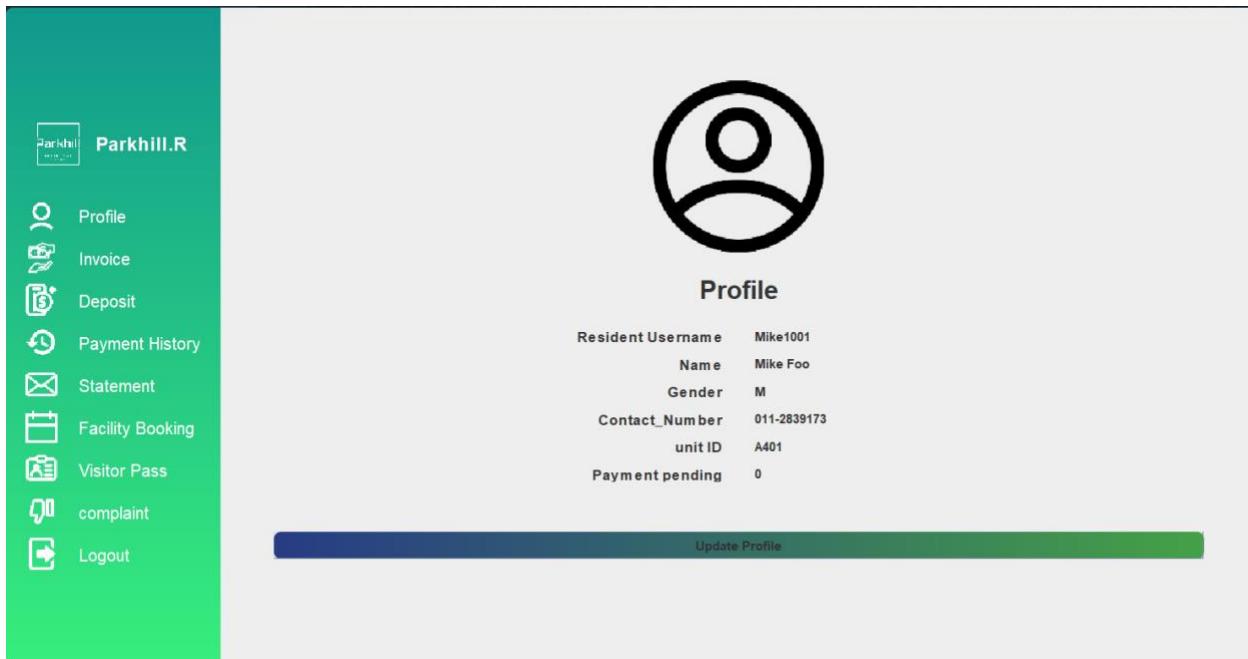
```
1 public Building_Executive(String executiveID, String name, char gender, String contact_number) {  
2     super.set_Info(executiveID, name, gender, contact_number, this.position);  
3 }
```

## 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.



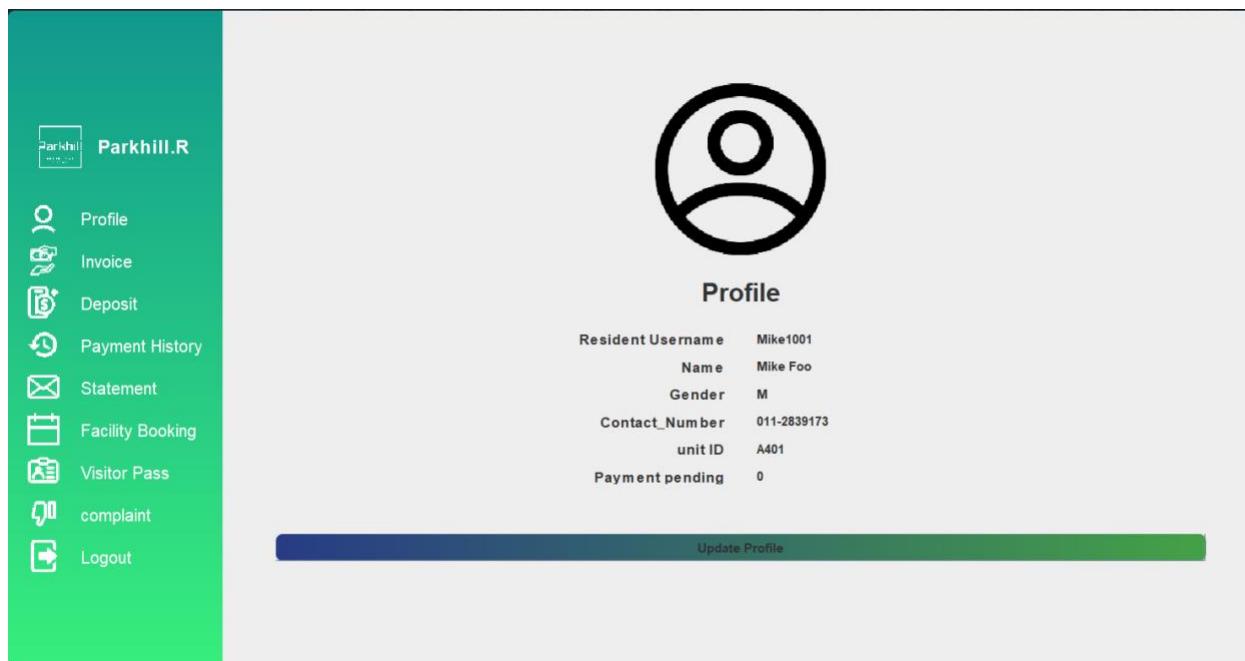
After logging into the system, all user interfaces have a "Logout" button in the bottom left corner. By clicking this button, each user can return to the login page.

## 7.2 Resident / Tenant

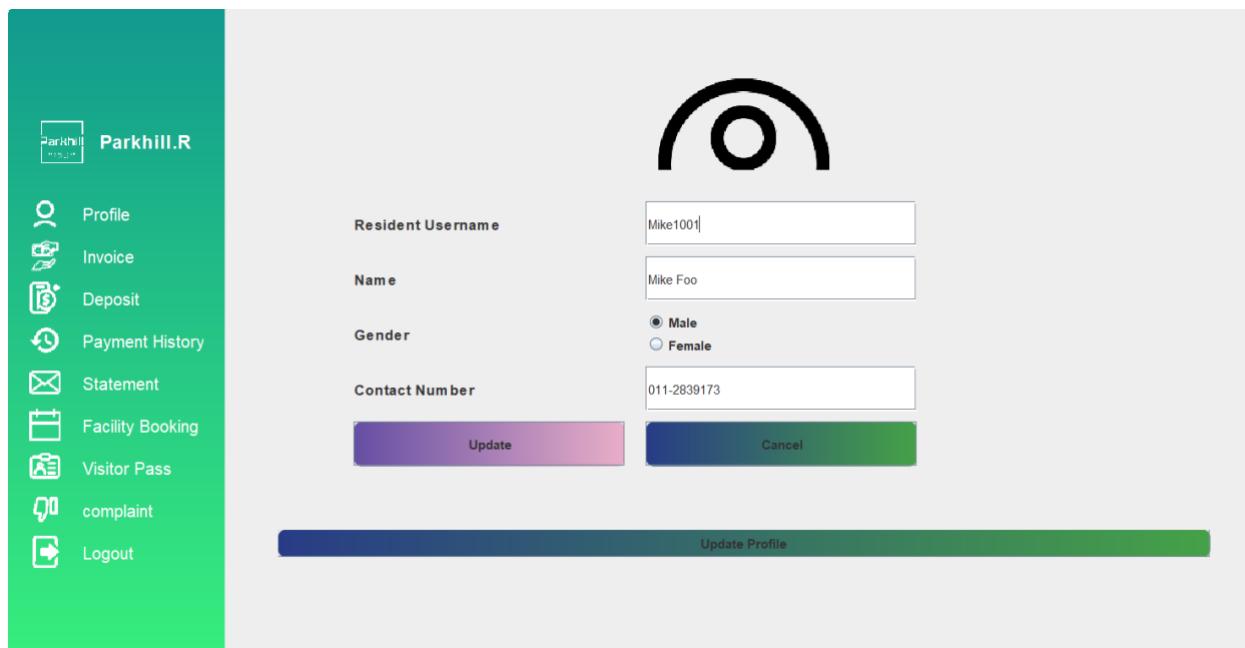
### 7.2.1 Functional Requirement

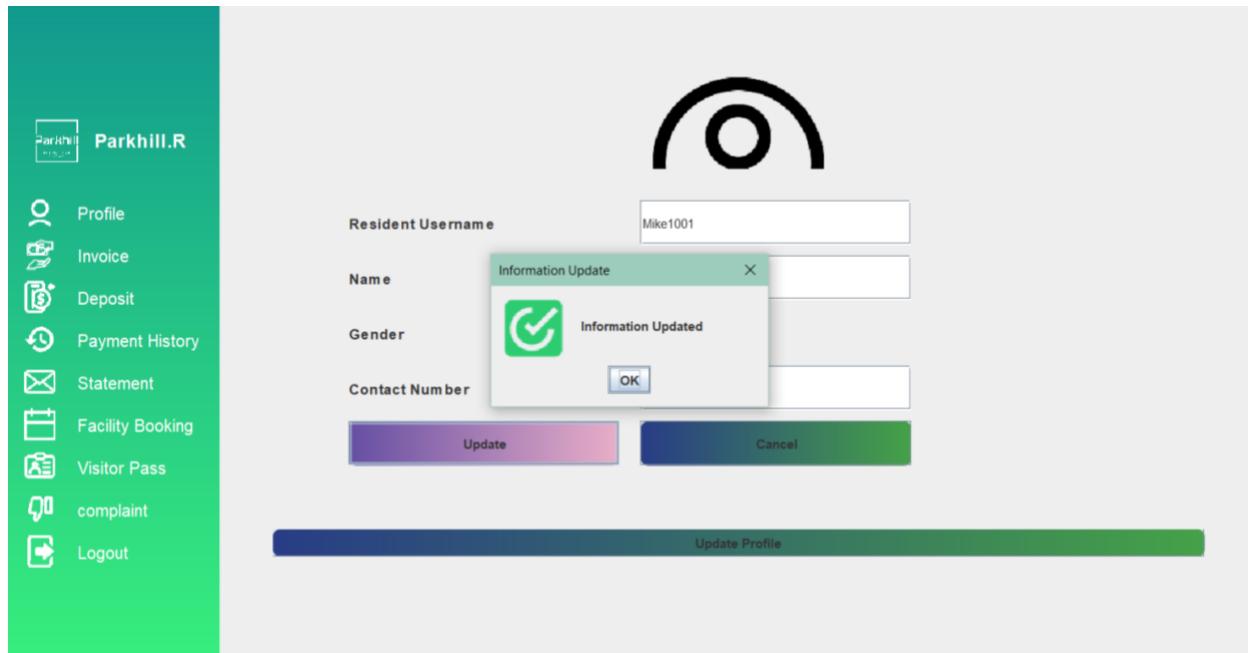
1. User profile interface
2. Invoice function
3. Deposit function
4. Payment history interface
5. Statement interface
6. Facility booking function
7. Visitor pass interface
8. Complaint interface

### 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

The screenshot shows the Parkhill.R mobile application's invoice tab. The sidebar on the left is identical to the profile update screen. The main area features a search bar and a user ID (Mike1001). Below is a table of invoices:

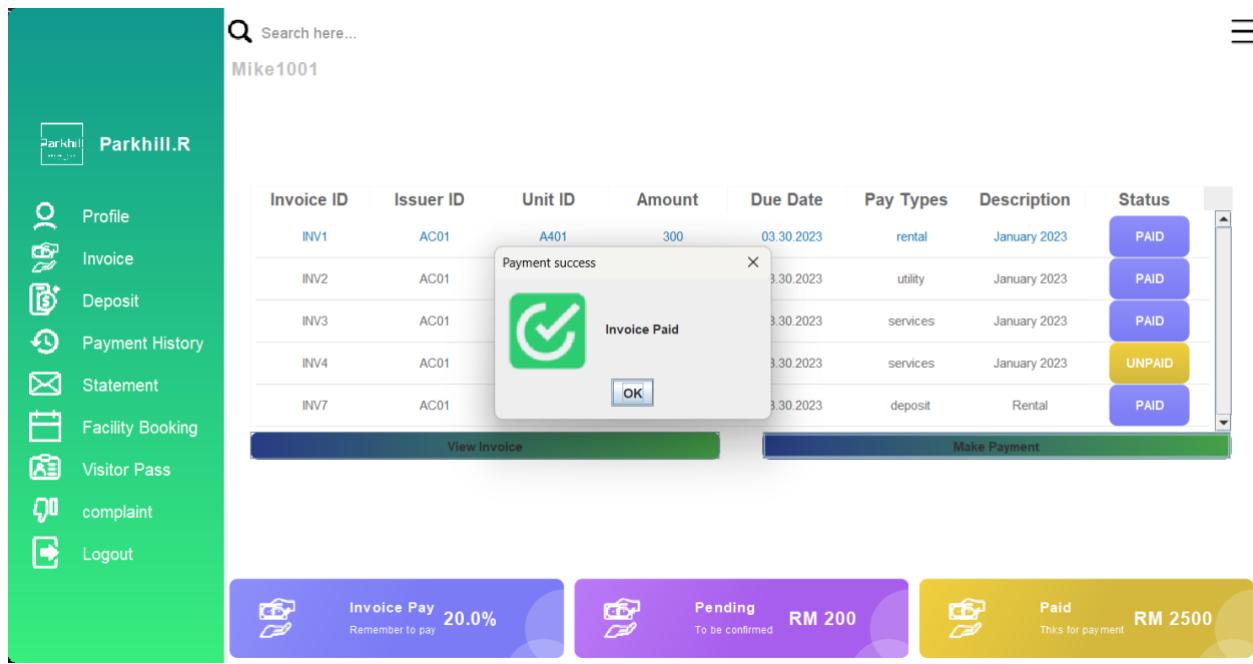
Invoice ID	Issuer ID	Unit ID	Amount	Due Date	Pay Types	Description	Status
INV1	AC01	A401	300	03.30.2023	rental	January 2023	PAID
INV2	AC01	A401	200	03.30.2023	utility	January 2023	PAID
INV3	AC01	A401	100	03.30.2023	services	January 2023	PAID
INV4	AC01	A401	100	03.30.2023	services	January 2023	UNPAID
INV7	AC01	A401	1200	03.30.2023	deposit	Rental	PAID

At the bottom are "View Invoice" and "Make Payment" buttons. Below the table are three status boxes: "Invoice Pay 20.0% Remember to pay", "Pending To be confirmed RM 200", and "Paid This for payment RM 1900".

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



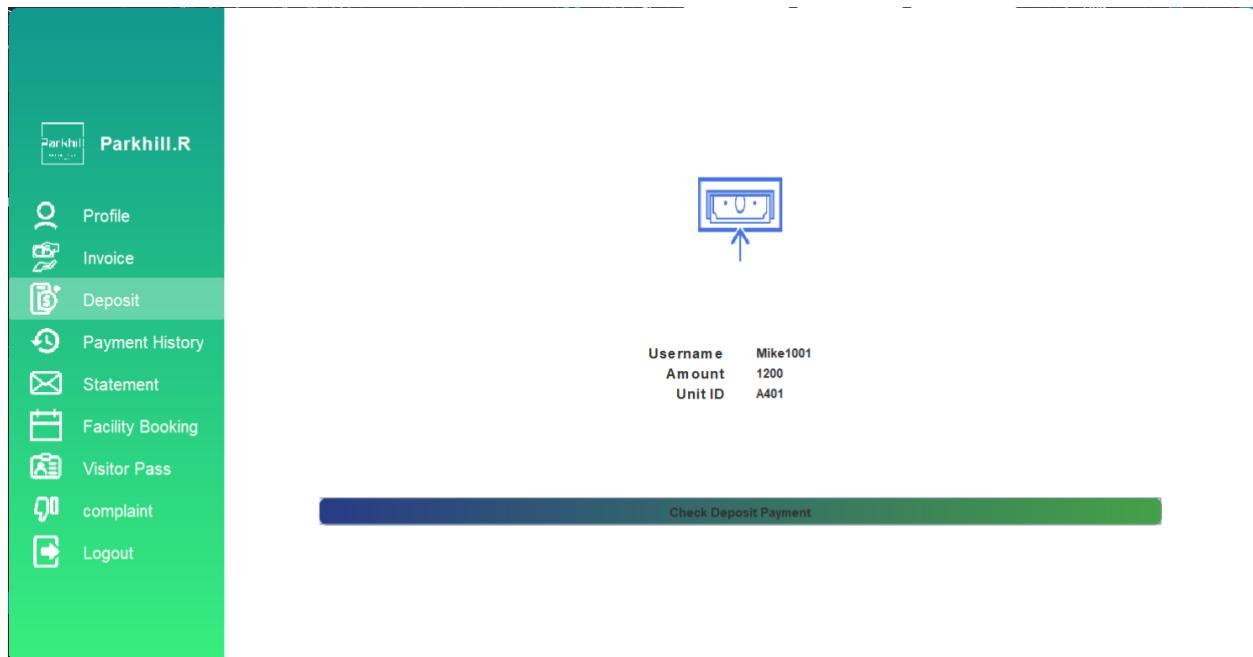
If the user clicks on the "Make Payment" button, they will be alerted once the payment has been made.

### 7.2.3.2 View Invoice



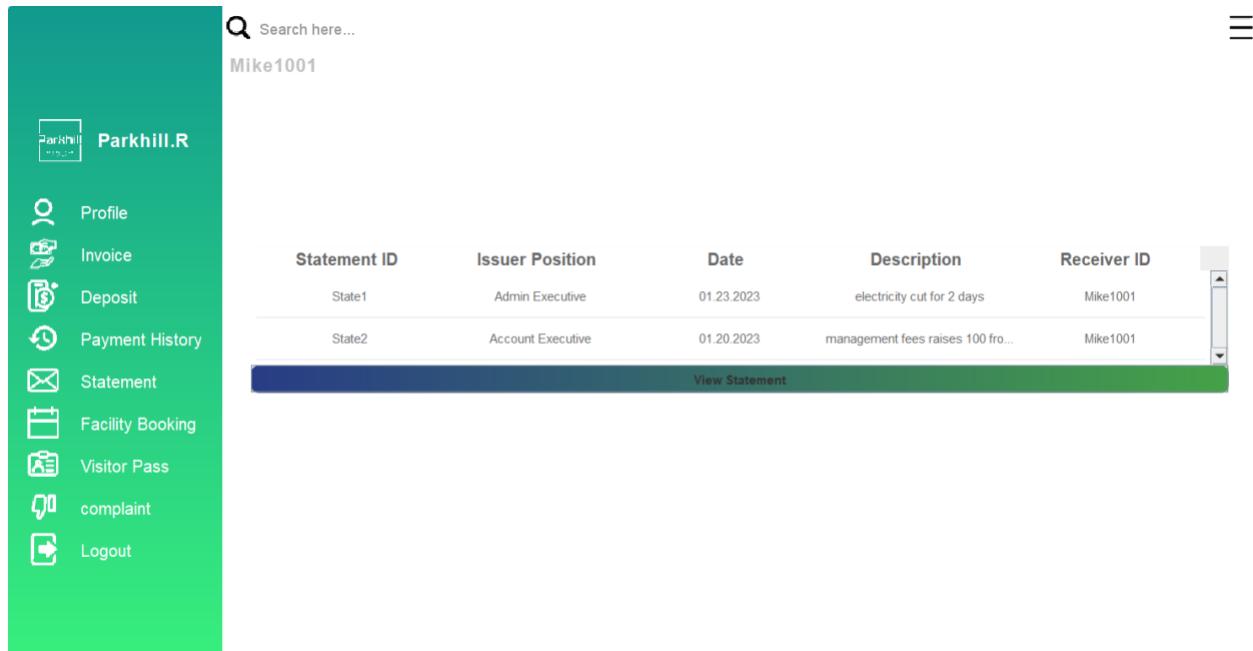
If the users choose to click on one of the payment histories displayed and then view it, they can click the "View Invoice" button, showing the user the invoice for the particular invoice they chose.

### 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



The screenshot shows the Parkhill.R mobile application interface. On the left is a vertical navigation bar with icons and labels: Profile, Invoice, Deposit, Payment History (which is highlighted in blue), Statement, Facility Booking, Visitor Pass, complaint, and Logout. At the top right is a search bar with placeholder text "Search here..." and a user profile section showing "Mike1001". The main content area displays a table of payment history entries:

Statement ID	Issuer Position	Date	Description	Receiver ID
State1	Admin Executive	01.23.2023	electricity cut for 2 days	Mike1001
State2	Account Executive	01.20.2023	management fees raises 100 fro...	Mike1001

A blue button at the bottom of the table says "View Statement".

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

Payment ID	Invoice ID	Pay Usern...	Unit ID	Amount	Payment d...	Payment T...	Issuer ID	Issued date
Pay1	INV1	Mike1001	A401	300	01.22.2023	rental	AC01	01.23.2023
Pay2	INV2	Mike1001	A401	200	01.23.2023	utility	AC01	01.24.2023
Pay3	INV3	Mike1001	A401	200	01.23.2023	services	AC01	01.24.2023
Pay6	INV7	Mike1001	A401	1200	01.23.2023	deposit	AC01	01.24.2023

View Receipt

**Statement**

Issuer Position	Account Executive
Date	01.20.2023
Receiver ID	Mike1001
Description	management fees raises 100 from next year onwards

Issued by Parkhill Residence

**Close**

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

The screenshot shows the 'Facility Booking' section of the Parkhill.R application. On the left is a sidebar with icons and labels: Profile, Invoice, Deposit, Payment History, Statement, Facility Booking (which is highlighted in green), Visitor Pass, complaint, and Logout. At the top right is a search bar with placeholder text 'Search here...' and a user profile for 'Mike1001'. The main area displays a table of bookings:

Booking ID	Facility ID	Resident Username	Date	Start time	End time
Book1	BH1	Mike1001	01.22.2023	110000	120000
Book2	BH1	Mike1001	01.24.2023	090000	100000

Below the table are four buttons: 'Book facility' (green), 'Update booking' (blue), 'Cancel Booking' (blue), and 'View Booking details' (blue).

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

**FACILITY BOOKING FORM**

Booking ID	Book3
Facility	Badminton Hall 2
Resident Username	Mike1001
Date (MM.dd.yyyy)	03.03.2023
Start time (HHmmss)	150000
End time (HHmmss)	160000
<b>Book Facility</b>	
<b>Cancel Booking</b>	

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

**FACILITY BOOKING FORM**

Booking ID	Book2
Facility	Badminton Hall 1
Resident Username	Mike1001
Date (MM.dd.yyyy)	01.24.2023
Start time (HHmmss)	130000
End time (HHmmss)	140000

**Buttons:** Update Booking details, Cancel Update

**FACILITY BOOKING FORM**

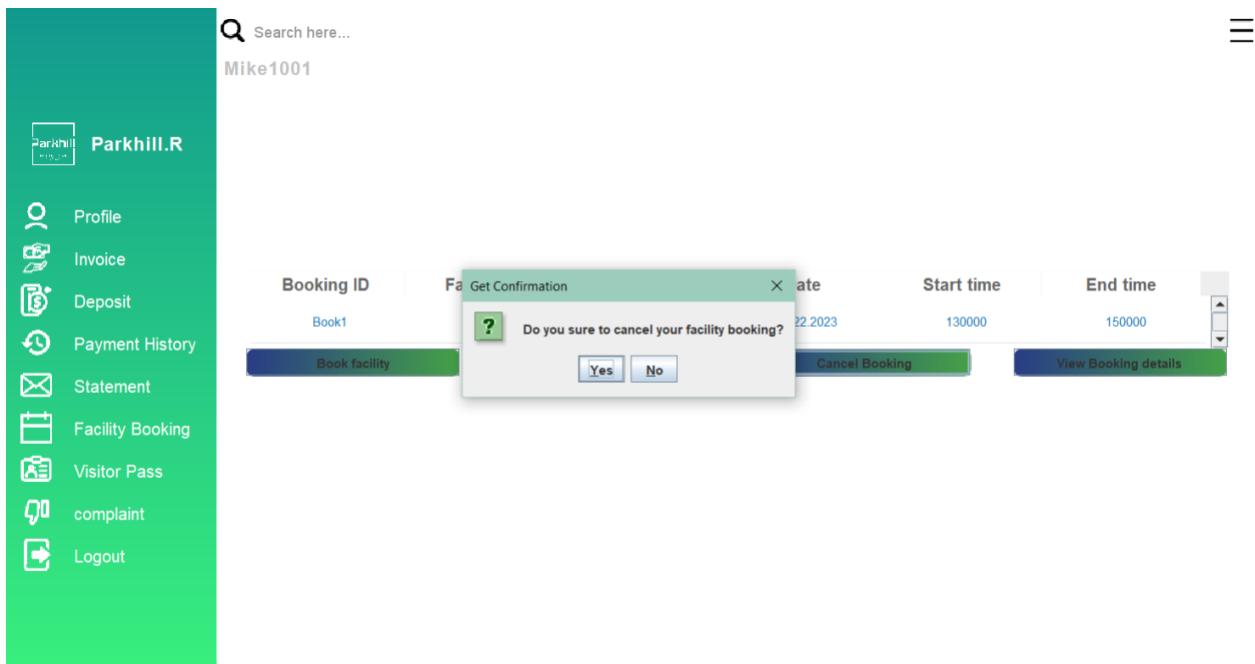
Booking ID	Book2
Facility	Badminton Hall 1
Resident Username	Mike1001
Date (MM.dd.yyyy)	<div style="background-color: #c8e6c9; padding: 5px; width: fit-content;">Time slot not available</div>
Start time (HHmmss)	130000
End time (HHmmss)	140000

**Alert Dialog:** Time slot not available (OK button)

**Buttons:** Update Booking details, Cancel Update

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

BOOKING DETAILS	
Booking ID	Book1
Facility ID	BH1
Resident Username	Mike1001
Date (MM.dd.yyyy)	01.22.2023
Start time (HHmmss)	130000
End time (HHmmss)	150000
<small>Issued by Parhill Residence</small>	
<input type="button" value="Close"/>	

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

The screenshot shows the Parkhill.R mobile application interface. The left sidebar contains a navigation menu with icons and labels: Profile, Invoice, Deposit, Payment History, Statement, Facility Booking, Visitor Pass (which is highlighted in green), complaint, and Logout. The main content area has a search bar at the top with placeholder text "Search here..." and a user profile "Mike1001". Below the search bar is a table showing visitor pass details:

Visitor Pas...	Visitor Name	Resident U...	Unit ID	Gender	Contact N...	Date Start	Date End	Status
VI1	Edmund	Mike1001	A401	M	012-3456789	12.26.2022	12.27.2022	APPROVED
VI3	Ed sheerann	Mike1001	A401	M	012-2346789	12.26.2022	12.27.2022	PENDING APPROVAL

At the bottom of the table are four buttons: "Apply Visitor Pass", "Update Visitor Pass", "Cancel Visitor Pass", and "View 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

**Visitor Pass Apply Form**

Visitor Pass ID	V14
Visitor name	Jennifer
Resident Username	Mike1001
Unit ID	A401
Gender	Female
Contact Number	010-4061997
Date start (MM.dd.yyyy)	03.02.2023
Date end (MM.dd.yyyy)	03.04.2023
<b>Apply Visitor Pass</b>	
<b>Cancel Application</b>	

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

**Visitor Pass Apply Form**

Visitor Pass ID	Vi3
Visitor name	Ed sheerann
Resident Username	Mike1001
Unit ID	A401
Gender	Male
Contact Number	012-2346789
Date start (MM.dd.yyyy)	12.26.2022
Date end (MM.dd.yyyy)	12.27.2022
<input type="button" value="Update Visitor Pass"/>	
<input type="button" value="Cancel Update"/>	

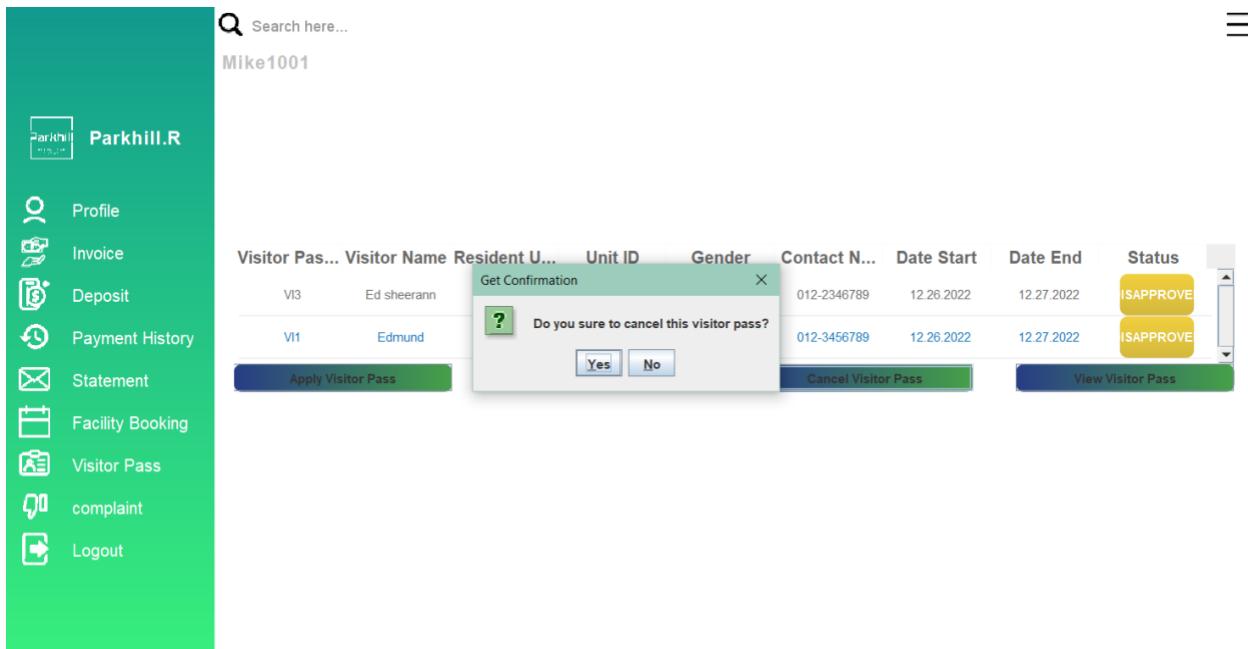
To update a visitor pass, the users must click on the visitor pass they wish to update and then proceed by clicking the "update visitor pass" button. The users will then be brought to this interface which allows them to update their visitor pass details. Afterward, the users can click the "update visitor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

### 7.2.8.3 View Visitor Pass

<b>VISITOR PASS</b>	
Visitor Pass ID	V13
Visitor name	Ed sheerann
Resident Username	Mike1001
Unit ID	A401
Gender	M
Contact Number	012-2346789
Date Start (MM.dd.yyyy)	12.26.2022
Date End(MM.dd.yyyy)	12.27.2022
Status	Disapproved
Issued by Parhill Residence	
<a href="#">Close</a>	

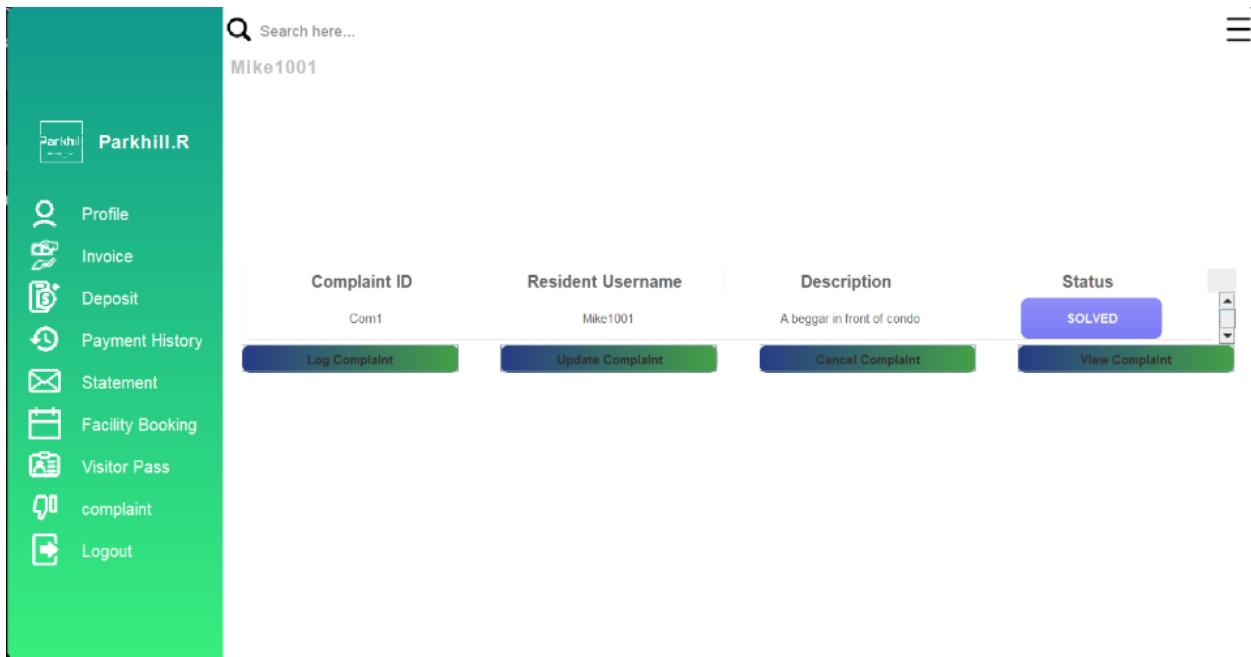
Users must select the visitor pass they want to cancel and click the “cancel visitor pass” button to cancel a visitor pass. This action will lead them to a new interface where they can confirm the cancellation of their pass details. Users will be notified to verify their decision before the visitor pass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

The screenshot shows a user interface titled "Complaint logging Form". It contains three input fields: "Complaint ID" with value "Com9", "Resident Username" with value "Mike1001", and a large "Description" text area which is currently empty. Below these fields are two buttons: "Log Complaint" (highlighted in red) and "Cancel Logging".

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

**Complaint logging Form**

Complaint ID	Com9
Resident Username	Mike1001
Description	the cat dancing in my house

**Buttons:**

- Log Complaint
- Cancel Logging

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

The screenshot shows a user interface titled "Complaint". It displays the following information:

Complaint ID	Com1
Resident Username	Mike1001
Description	A beggar in front of condo

Below the table, it says "Issued by Parhill Residence". At the bottom, there are two buttons: a blue "Cancel Complaint" button and a green "Back" button.

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.

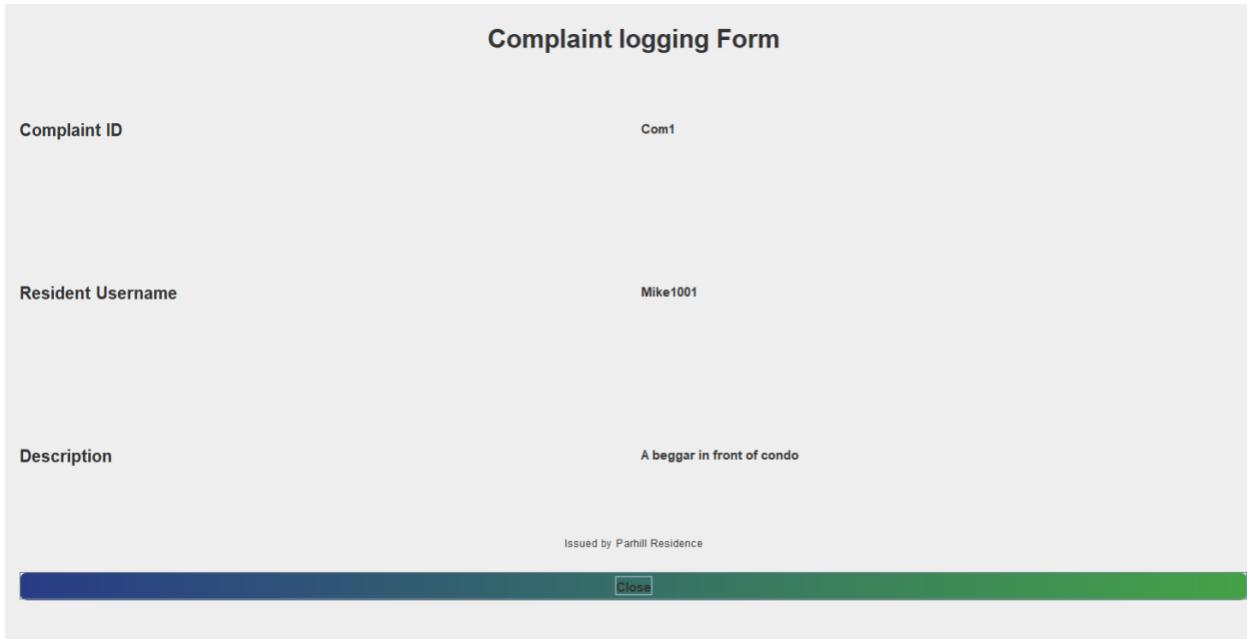
#### 7.2.9.4 View Complaint

**Complaint logging Form**

Complaint ID	Com1
Resident Username	Mike1001
Description	A beggar in front of condo

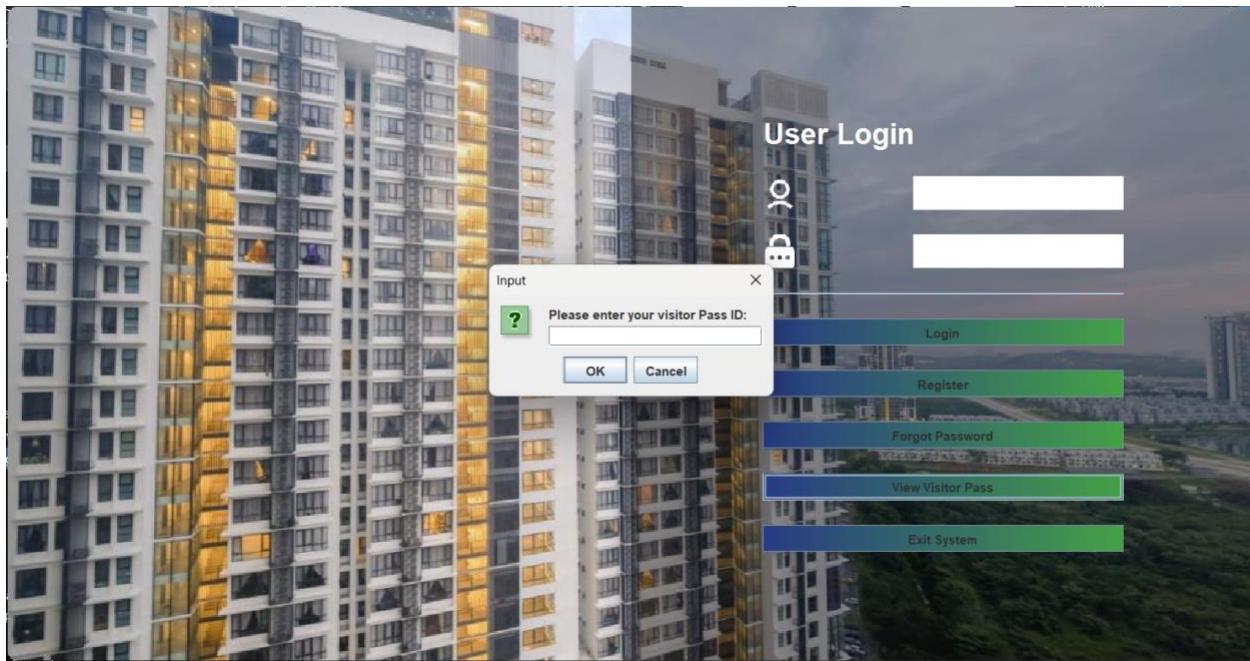
Issued by Parhill Residence

[Close](#)



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.

VISITOR PASS	
Visitor Pass ID	V13
Visitor name	Ed sheerann
Resident Username	Mike1001
Unit ID	A401
Gender	M
Contact Number	012-2346789
Date Start (MM.dd.yyyy)	12.26.2022
Date End(MM.dd.yyyy)	12.27.2022
Status	Disapproved

Issued by Parkhill Residence

[Close](#)

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.

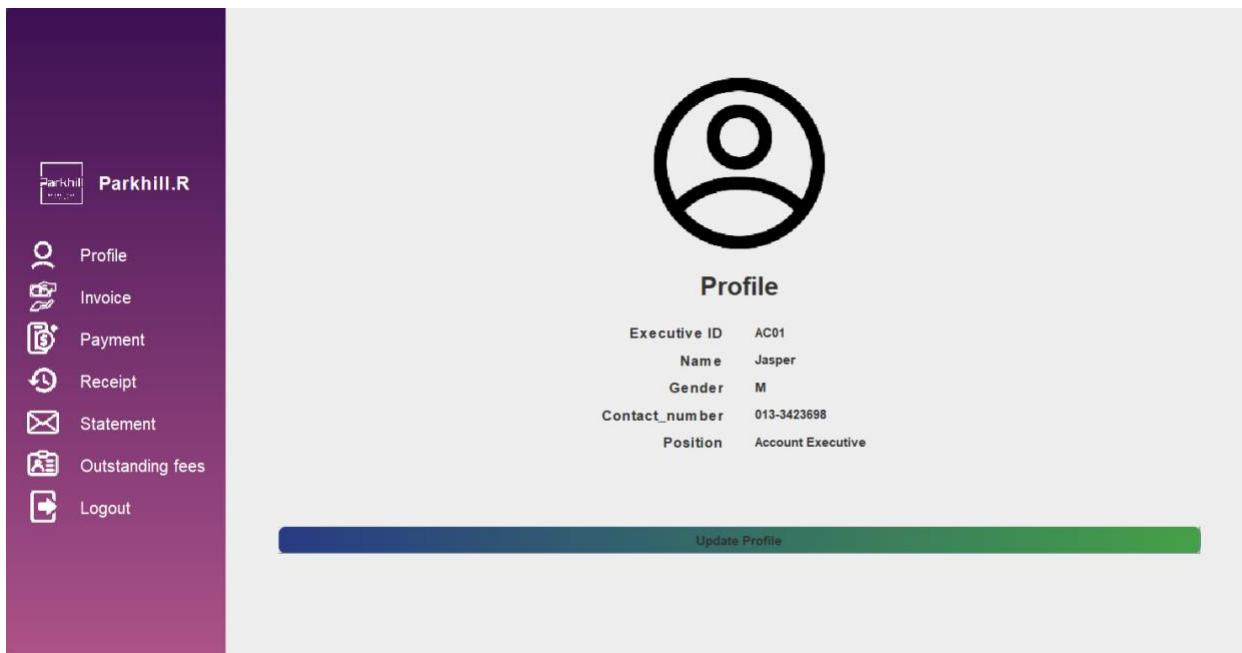
## 8 TP065116 Neaw Aik Ka

### 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.

The screenshot shows the Parkhill.R mobile application's profile update screen. On the left is a sidebar with a logo and various navigation icons: Profile, Invoice, Payment, Receipt, Statement, Outstanding fees, and Logout. The main area features a large circular logo at the top. Below it is a form with fields for Executive ID (AC01), Name (Jasper), Gender (Male selected), Contact Number (013-3423698), and Position (Account Executive). At the bottom are two buttons: "Update" (purple) and "Cancel" (green). A green bar at the very bottom contains the text "Update Profile".

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

The screenshot shows the Parkhill.R mobile application's invoice tab. The sidebar on the left includes a search bar, the user ID "AC01", and the same set of navigation icons as the profile screen. The main area displays a table of invoices with the following columns: Invoice ID, Issuer ID, Unit ID, Amount, Due Date, Pay Types, Description, and Status. The table lists eight invoices, each with a status indicator (PAID or UNPAID). At the bottom are two buttons: "View Invoice" (green) and "Issue Invoice" (blue). Below these buttons are three summary boxes: "Unpaid Inv... 50%" (blue), "Pending To be confirmed RM 200" (purple), and "Paid Payment get RM 2300" (yellow).

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

The screenshot shows a user interface for issuing an invoice. The title 'INVOICE' is centered at the top. Below it are several input fields:

- Invoice ID:** INV9
- Issuer ID:** AC01
- Unit ID:** A403 (in a dropdown menu)
- Amount (RM):** .00
- Due Date (MM.dd.yyyy):** ..
- Payment Types:** (empty input field)
- Description:** (empty input field with scroll bars)

At the bottom of the form are two buttons: 'Issue Invoice' (highlighted in blue) and 'Cancel'.

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

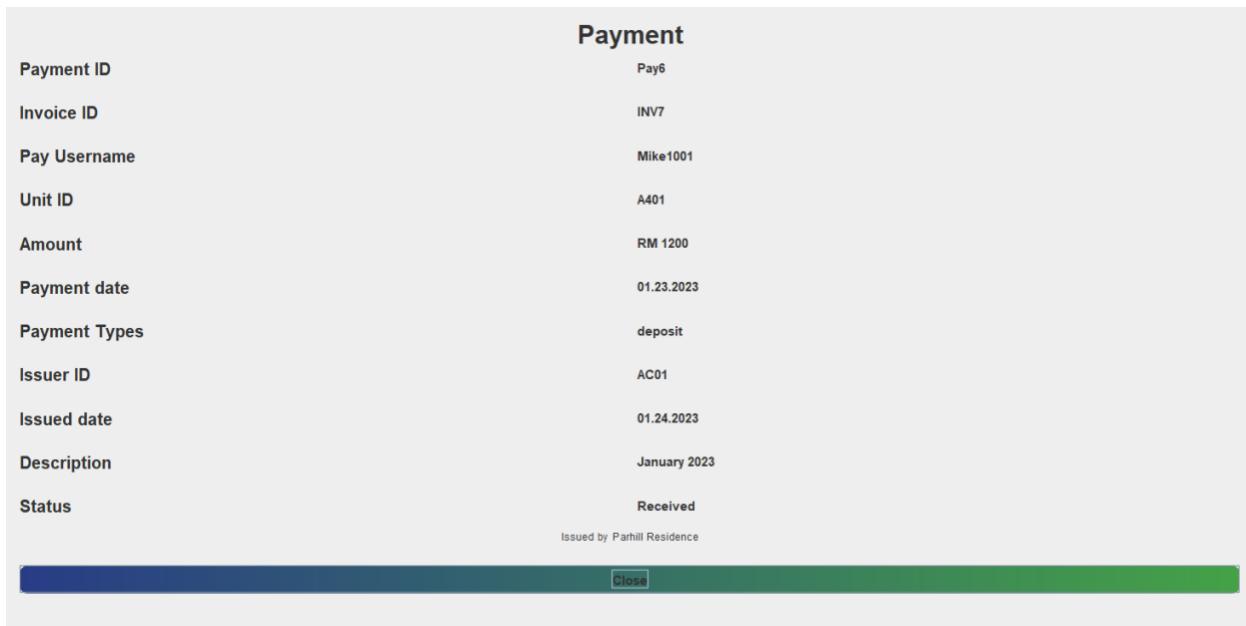
The screenshot shows the Parkhill.R user interface with a dark purple sidebar on the left containing navigation icons and a logo. The main area has a light gray header with a search bar and the identifier 'AC01'. Below the header is a table displaying payment records. The table columns are: Payment ID, Invoice ID, Pay User..., Unit ID, Amount, Payment ... (partially visible), Payment ... (partially visible), Issuer ID, Issued date, and Status. The data in the table is as follows:

Payment ID	Invoice ID	Pay User...	Unit ID	Amount	Payment ...	Payment ...	Issuer ID	Issued date	Status
Pay1	INV1	Mike1001	A401	300	01.22.2023	rental	AC01	01.23.2023	RECEIVED
Pay2	INV2	Mike1001	A401	200	01.23.2023	utility	AC01	01.24.2023	RECEIVED
Pay3	INV3	Mike1001	A401	200	01.23.2023	services	AC01	01.24.2023	RECEIVED
Pay4	INV5	VE001	floor 1 canteen	200	01.23.2023	services			PENDING
Pay5	INV6	VE001	floor 1 canteen	200	01.23.2023	services	AC01	01.24.2023	RECEIVED
Pay6	INV7	Mike1001	A401	1200	01.23.2023	deposit	AC01	01.24.2023	RECEIVED

At the bottom of the table are two green buttons: 'View Payment' and 'Confirm payment'.

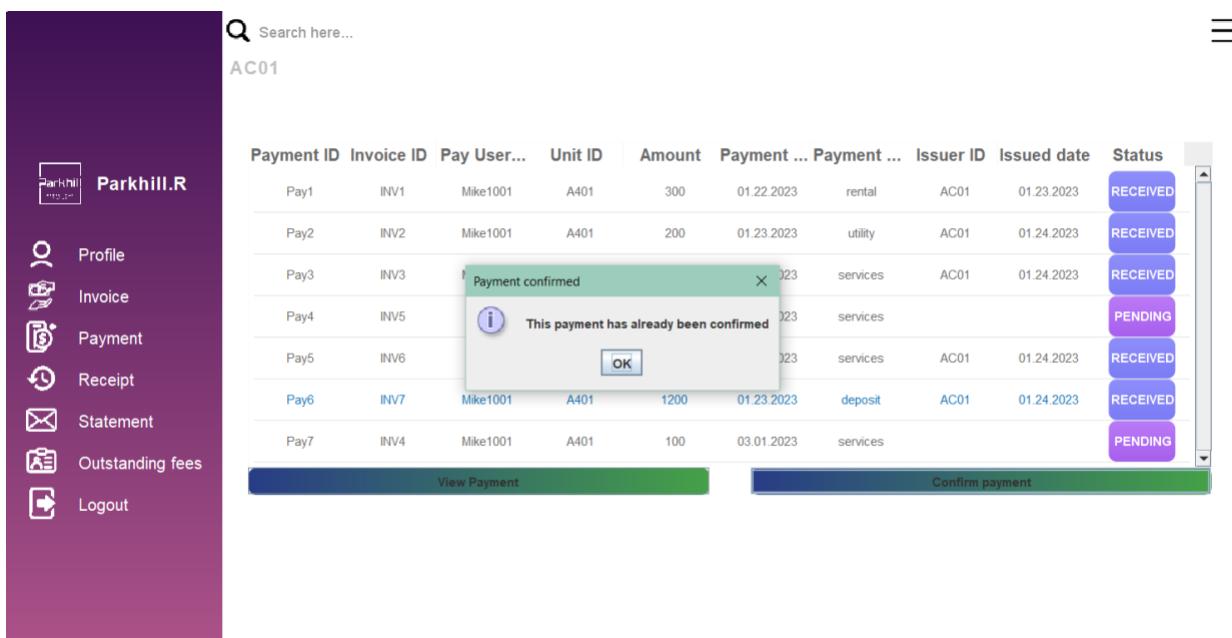
When users access the payment tab, they will be directed to a user interface that displays all payment that is waiting to be confirmed by the user. The interface also features button functions allowing users to search through everyone's payment status, view payment, and confirm payment.

#### 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

Payment ID	Invoice ID	Pay User...	Unit ID	Amount	Payment ...	Payment ...	Issuer ID	Issued date	Status
Pay1	INV1	Mike1001	A401	300	01.22.2023	rental	AC01	01.23.2023	RECEIVED
Pay2	INV2	Mike1001	A401	200	01.23.2023	utility	AC01	01.24.2023	RECEIVED
Pay3	INV3	Mike1001	A401	200	01.23.2023	services	AC01	01.24.2023	RECEIVED
Pay5	INV6	VE001	floor 1 canteen	200	01.23.2023	services	AC01	01.24.2023	RECEIVED
Pay6	INV7	Mike1001	A401	1200	01.23.2023	deposit	AC01	01.24.2023	RECEIVED

View 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.

RECEIPT	
Payment ID	Pay6
Invoice ID	INV7
Pay Username	Mike1001
Unit ID	A401
Amount	RM 1200
Payment date	01.23.2023
Payment Types	deposit
Issuer ID	AC01
Issued date	01.24.2023
Description	January 2023
Status	Received

Issued by Parhill Residence

Close

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

The screenshot shows a user interface for managing financial statements. On the left is a sidebar with a purple header containing the Parkhill.R logo and a search bar. Below the header are several menu items: Profile, Invoice, Payment, Receipt, Statement (which is highlighted in blue), Outstanding fees, and Logout. The main content area has a header 'AC01' and a search bar. Below this is a table with columns: Statement ID, Issuer Position, Date, Description, and Receiver ID. The table contains four rows of data. At the bottom of the page are two buttons: 'View Statement' and 'Issue Statement'.

Statement ID	Issuer Position	Date	Description	Receiver ID
State1	Admin Executive	01.23.2023	electricity cut for 2 days	Mike1001
State2	Account Executive	01.20.2023	management fees raises 100 fro...	Mike1001
State1	Admin Executive	01.23.2023	electricity cut for 2 days	VE001
State2	Account Executive	01.20.2023	management fees raises 100 fro...	VE001

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.

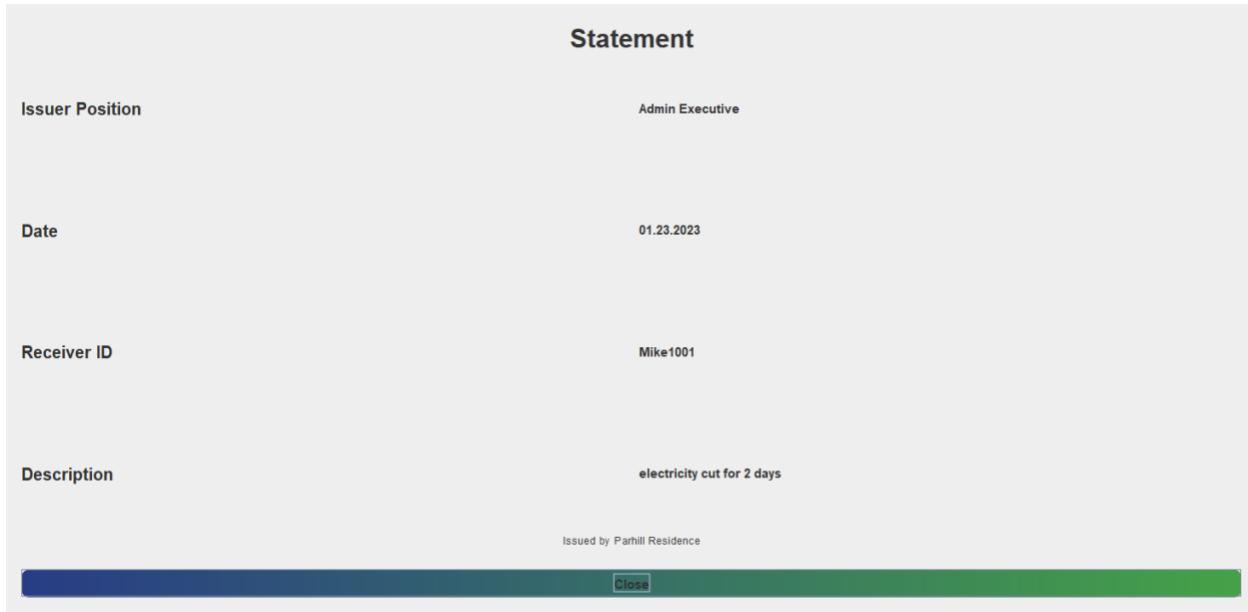
### 8.1.6.1 View Statement

**Statement**

Issuer Position	Admin Executive
Date	01.23.2023
Receiver ID	Mike1001
Description	electricity cut for 2 days

Issued by Parhill Residence

**Close**



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.

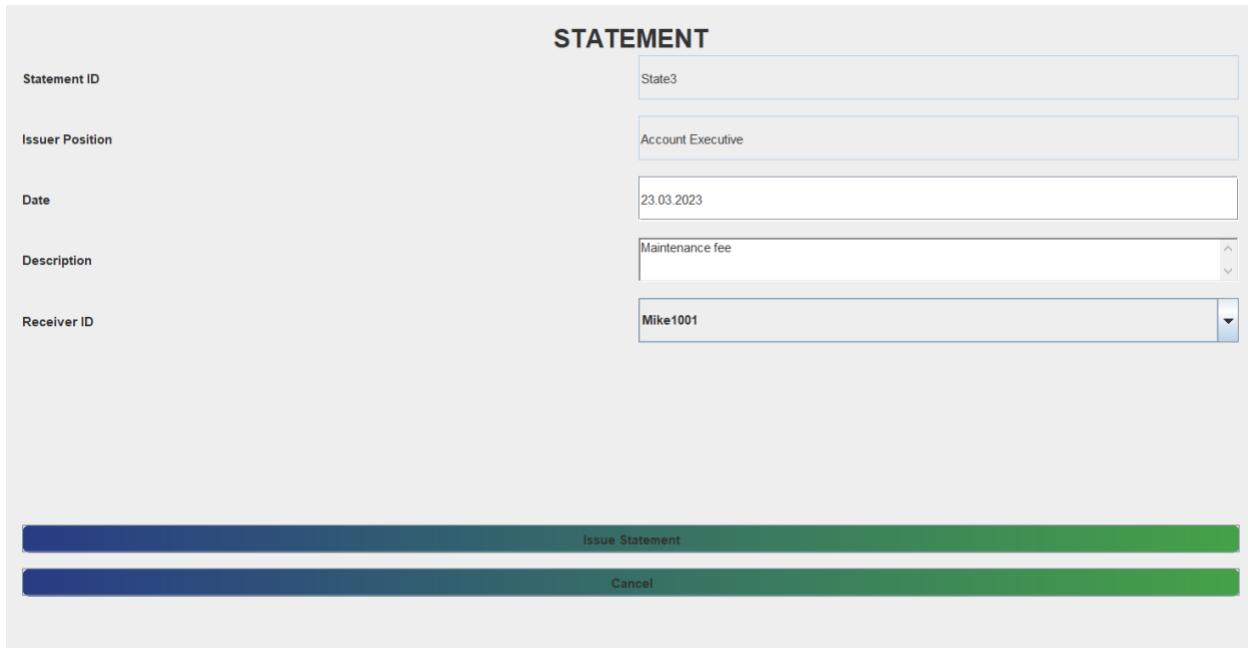
### 8.1.6.2 Issue Statement

**STATEMENT**

Statement ID	State3
Issuer Position	Account Executive
Date	23.03.2023
Description	Maintenance fee
Receiver ID	Mike1001

**Issue Statement**

**Cancel**



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 screenshot shows the Parkhill.R mobile application interface. On the left is a sidebar with a purple gradient background containing the Parkhill logo and navigation options: Profile, Invoice, Payment, Receipt, Statement, Outstanding fees, and Logout. The main content area has a white background with a search bar at the top. Below it, a section labeled 'AC01' displays a table of user data:

Username	Name	Gender	Contact Number	Unit ID	Payment pending
Andrew123	Andrew Wiggins	M	012-7564382	A402	0
Jay1030	Jay Chou	M	013-78433298	A403	0
Ahmad4403	Ahmad Wakil	F	015-6723783	A404	0
Joy101	Joy Cheng	F	012-2143673	A405	0
Foo1202	Foo Hei	F	019-6453783	A406	-1
Mike1001	Mike Foo	F	011-2839173	A401	0
VE002	MalaiPo	F	013-1234152	floor 3 mamak	600
VE001	Abunene	M	011-1743475	floor 1 canteen	500

Below the table are two green buttons: 'View Pending fees' and 'View Deposit'. At the bottom, there are three colored boxes showing pending amounts: a blue box for 'Total Pending RM 99 To be paid', a purple box for 'Vendor Pending RM 100 To be paid', and a yellow box for 'Resident Pending RM -1 To be paid'.

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

This screenshot shows a modal window titled 'PENDING DETAILS' with the following user information:

Username	VE001
Name	Abunene
Gender	M
Contact Number	011-1743475
Unit ID	floor 1 canteen
Payment Pending	RM 500

At the bottom of the modal, it says 'Issued by Parkhill Residence' and has a 'Close' button.

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

The screenshot shows the Parkhill.R mobile application interface. On the left is a sidebar with navigation icons and labels: Profile, Invoice, Payment, Receipt, Statement, Outstanding fees, and Logout. The main area has a header 'DEPOSIT'. Below it is a table with columns: Username, Name, Gender, Contact Number, Unit ID, and Payment pending. A modal dialog box is overlaid on the table, displaying the message: 'The user you have selected has no deposit with management.' with an 'OK' button. At the bottom of the screen are three summary cards: 'Total Pend... RM 99' (To be paid), 'Vendor Pe... RM 100' (To be paid), and 'Resident ... RM -1' (To be paid).

Username	Name	Gender	Contact Number	Unit ID	Payment pending
Mike1001	Mike Foo	M	011-2839173	A401	RM 1200
Andrew1123	Andrew Wiggins	M	012-7564382	A402	0
Jay1030	Jay Chou	M	013-78433298	A403	0
Ahmad4403	Ahmad Wakil	F	015-6723783	A404	0
Joy101				A405	0
Foo1202				A406	-1
Mike1001				A401	0
VE002	MalaiPo	F	013-1234152	floor 3 mamak	600
VE001	Abunene	M	011-1743475	floor 1 canteen	500

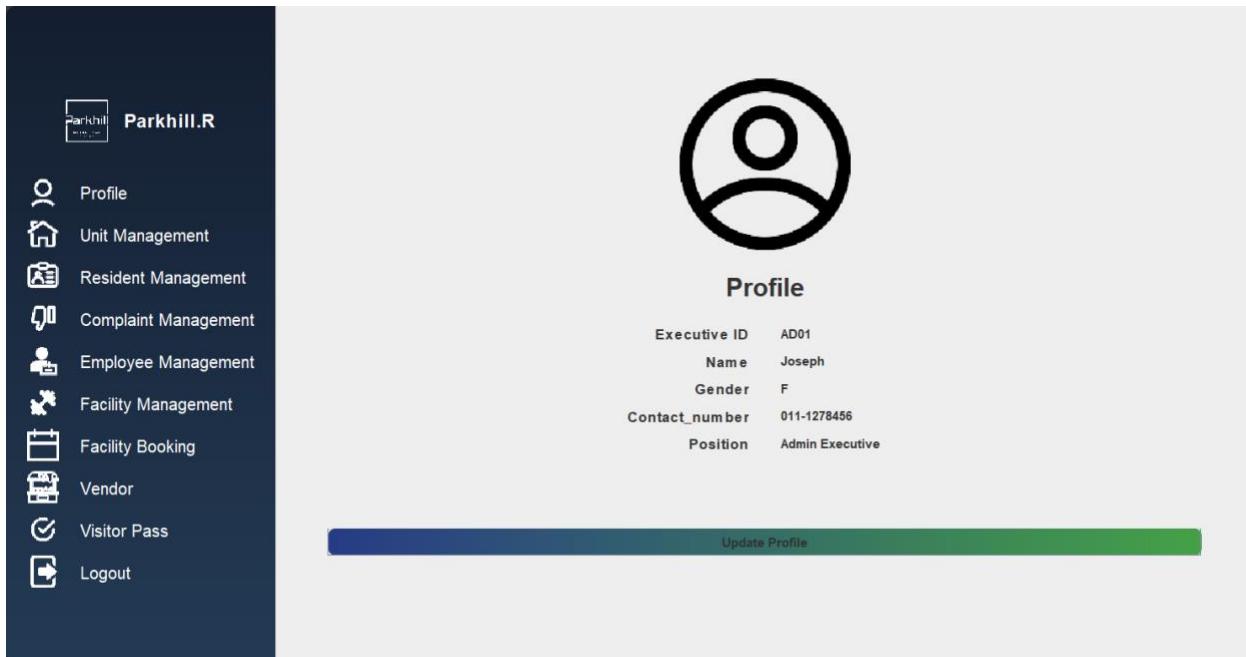
To view a deposit, users must choose the one they want to view and click the "view deposit" button. This will direct them to a new interface that displays the deposit details if there is one. Otherwise, the system will notify the users about the lack of deposit info.

## 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.

The screenshot shows a user interface for managing a profile. On the left is a sidebar with a dark blue background containing icons and text for various management functions: Profile, Unit Management, Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main area has a light gray background. At the top center is a large, stylized circular logo consisting of concentric arcs. Below the logo is a form with fields for Executive ID (AD01), Name (Joseph), Gender (Female selected), Contact Number (011-1278456), and Position (Admin Executive). There are two buttons at the bottom: a purple "Update" button and a green "Cancel" button. A long, thin green bar at the bottom contains the text "Update Profile".

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

The screenshot shows a web-based application interface for unit management. On the left is a sidebar menu with icons and labels: Profile, Unit Management (selected), Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main area has a search bar at the top labeled "Search here...". Below it, the text "AD01" is displayed. The central part of the screen is a table listing units:

Floor	Unit ID	Complete Year	Furnish	Parking Unit	Owner Userna...	Resident User...
4	A401	2020	Fully	P01-001	Mike1001	Mike1001
4	A402	2020	Fully	P01-002	Andrew1123	Andrew1123
4	A403	2020	Fully	P01-003	Jay1030	Jay1030
4	A404	2020	Not	P01-004	Ahmad4403	Ahmad4403
4	A405	2020	Fully	P01-005	Joy101	Rain1031
5	A506	2020	Half	P01-006	Foo1201	Foo1201

At the bottom of the table are four buttons: "Add New Unit" (green), "Modify Unit Info" (blue), "Delete Unit" (green), and "View Unit details" (blue). A vertical scroll bar is visible on the right side of the table.

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

**UNIT ADDING**

Floor	11
Unit ID	A115
Completer Year	2023
Furnish situation	Fully
Parking Unit	1
Owner Username	min9yu_k
Resident Username	MinGyu

Add New Unit

Cancel

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

**UNIT DETAILS MODIFYING**

Floor	4
Unit ID	A404
Completer Year	2020
Furnish situation	Not
Parking Unit	P01-004
Owner Username	<input type="text" value="Ahmad4403"/> <input type="text" value="Ahmad4403"/>
<a href="#">Add New Unit</a> <a href="#">Cancel</a>	

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

The screenshot shows the Parkhill.R application's Unit Management page. On the left is a sidebar with various management options: Profile, Unit Management (selected), Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main area displays a table of unit details with columns: Floor, Unit ID, Complete Year, Furnish, Parking Unit, Owner Username, and Resident User... (partially visible). One row in the table has a highlighted background. A delete confirmation dialog box is overlaid on the table, asking "Are you sure to delete this unit?" with "Yes" and "No" buttons. At the bottom of the page are four green buttons: "Add New Unit", "Modify Unit Info", "Delete Unit" (disabled), and "View Unit details".

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

UNIT DETAILS	
Floor	4
Unit ID	A404
Completer Year	2020
Furnish situation	Not
Parking Unit	P01-004
Owner Username	Ahmad4403
Resident Username	Ahmad4403
Issued by Parkhill Residence	
<a href="#">Close</a>	

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

The screenshot shows a web-based application interface for resident management. At the top left is the logo "Parkhill.R". To its right is a search bar with the placeholder "Search here...". Below the search bar is the identifier "AD01". On the far right is a vertical menu icon.

The left sidebar contains a vertical navigation menu with the following items:

- Profile
- Unit Management
- Resident Management** (highlighted in blue)
- Complaint Management
- Employee Management
- Facility Management
- Facility Booking
- Vendor
- Visitor Pass
- Logout

The main content area displays a table of resident records:

Resident Username	Name	Gender	Contact Number	Unit ID
Andrew1123	Andrew Wiggins	M	012-7564382	A402
Jay1030	Jay Chou	M	013-78433298	A403
Ahmad4403	Ahmad Wakil	F	015-6723783	A404
Joy101	Joy Cheng	F	012-2143673	A405
Foo1202	Foo Hei	F	019-6453783	A406
Mike1001	Mike Foo	M	011-2839173	A401

Below the table are four buttons:

- Add New Resident
- Modify Resident Info
- Delete Resident
- View Resident Info

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

The screenshot shows a form titled "RESIDENT ADDING". The form fields and their values are:

- Resident Username: keita999
- Name: Keita
- Gender:
  - Male
  - Female
- Contact Number: 010-4072001
- Unit ID: G999

At the bottom of the form are two buttons:

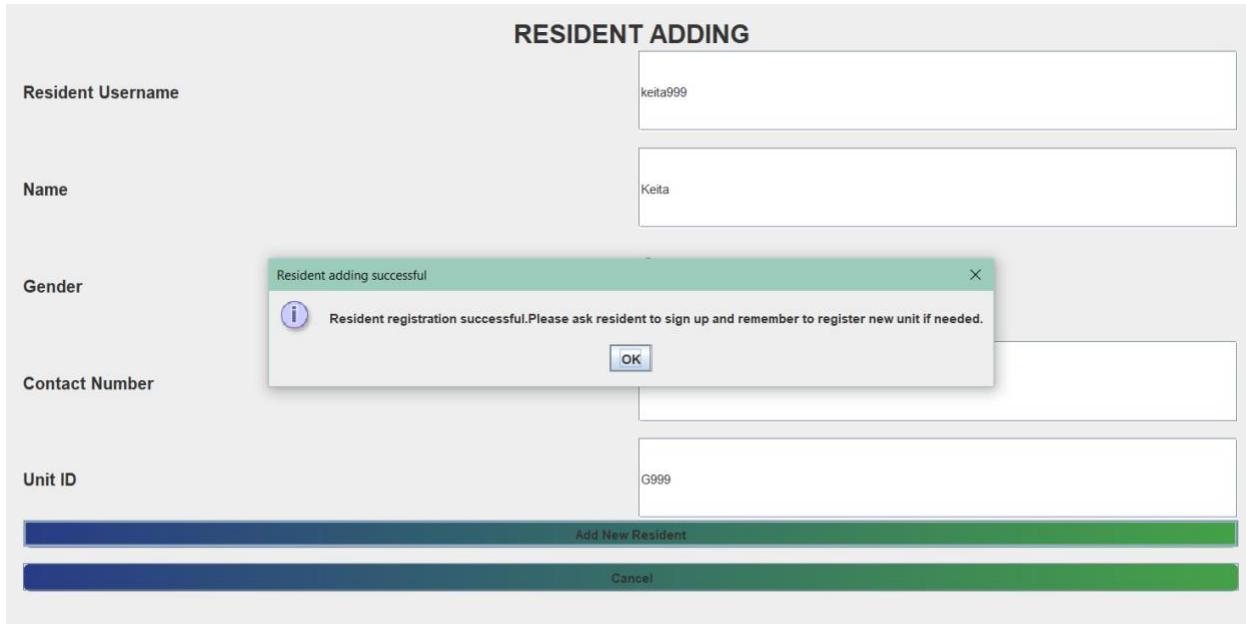
- Add New Resident (highlighted in green)
- Cancel

**RESIDENT ADDING**

Resident Username	keita999
Name	Keita
Gender	<div style="background-color: #2e6b2e; color: white; padding: 5px; text-align: center;">Resident adding successful</div> <div style="background-color: #f0f0f0; border: 1px solid #ccc; padding: 10px; margin-top: -10px;"><span style="color: #0070C0;">i</span> Resident registration successful. Please ask resident to sign up and remember to register new unit if needed. <button style="border: none; background-color: inherit; color: inherit; font-size: inherit; padding: 0; margin: 10px 0;">OK</button></div>
Contact Number	
Unit ID	G999

**Add New Resident**

**Cancel**

A screenshot of a web-based application titled "RESIDENT ADDING". The form contains fields for Resident Username (keita999), Name (Keita), Gender (a modal dialog box is open, showing a success message: "Resident adding successful" and "Resident registration successful. Please ask resident to sign up and remember to register new unit if needed.", with an "OK" button), Contact Number (empty), Unit ID (G999), and two buttons at the bottom: "Add New Resident" and "Cancel".

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.

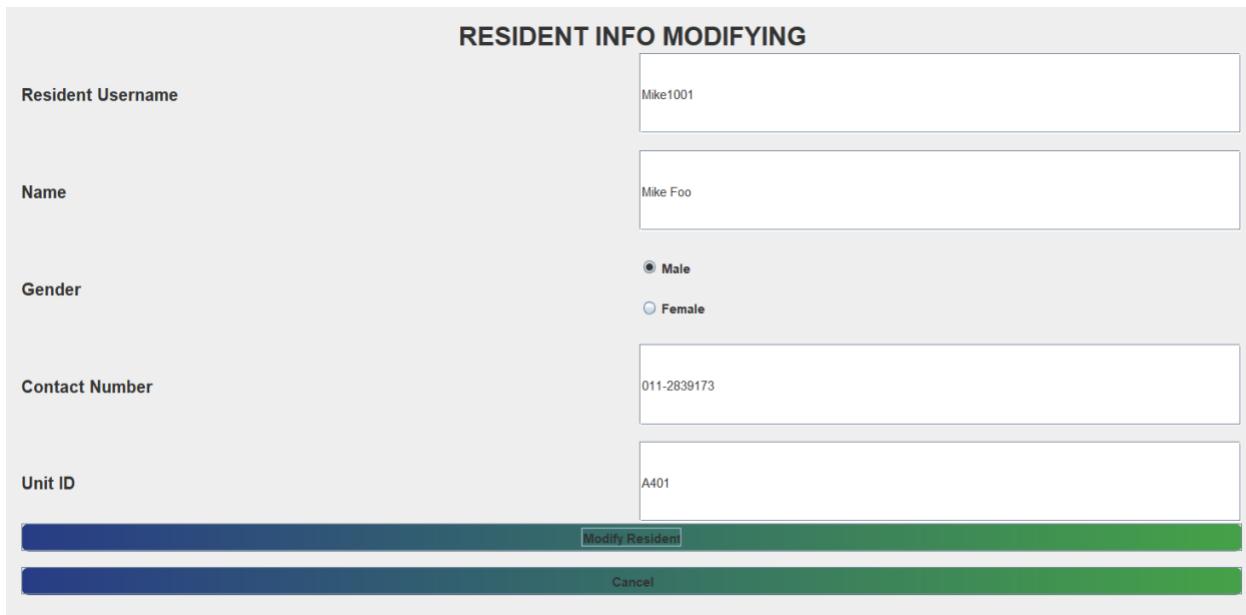
#### 8.2.4.2 Modify Resident Info

**RESIDENT INFO MODIFYING**

Resident Username	Mike1001
Name	Mike Foo
Gender	<input checked="" type="radio"/> Male <input type="radio"/> Female
Contact Number	011-2839173
Unit ID	A401

**Modify Resident**

**Cancel**



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

The screenshot shows a web-based application interface for managing residents. On the left is a dark sidebar menu with various icons and labels: Profile, Unit Management, Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main content area has a header with a search bar and the identifier 'AD01'. Below this is a table listing resident information. A modal dialog box titled 'Delete confirmation' is overlaid on the table, asking 'Are you sure to delete this resident?' with 'Yes' and 'No' buttons. The table columns are Resident Username, Name, Gender, Contact Number, and Unit ID. The rows contain data for Andrew Wiggins, Jay Chou, Ahmad Wakil, Joy101, Foo1202, Mike1001, and Keita. At the bottom of the table are four buttons: Add New Resident, Modify Resident Info, Delete Resident (which triggered the dialog), and View Resident Info.

Resident Username	Name	Gender	Contact Number	Unit ID
Andrew1123	Andrew Wiggins	M	012-7564382	A402
Jay1030	Jay Chou	M	013-78433298	A403
Ahmad4403	Ahmad Wakil	F	015-6723783	A404
Joy101	Delete confirmation		012-2143673	A405
Foo1202	Are you sure to delete this resident?		019-6453783	A406
Mike1001	<input type="button" value="Yes"/> <input type="button" value="No"/>		011-2839173	A401
keita999	Keita	M	010-4072001	G999

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

**RESIDENT DETAILS**

Resident Username	Mike1001
Name	Mike Foo
Gender	M
Contact Number	011-2839173
Unit ID	A401

Issued by Parkhill Residence

[Close](#)

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

The screenshot shows the Parkhill.R application interface. On the left is a dark sidebar menu with icons and labels: Profile, Unit Management, Resident Management, Complaint Management (which is highlighted in blue), Employee Management, Facility Management, Facility Booking, Vendor, and Logout. Above the sidebar is the text "Parkhill.R". To the right of the sidebar is a search bar with the placeholder "Search here...". Below the search bar is the user identifier "AD01". The main content area displays a table of complaints:

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOLVED
Com3	Joy101	Earthquake	UNSOLVED
Com4	VE001	Earthquake	UNSOLVED
Com5	VE001	swimming pool has pee	UNSOLVED

Below the table are three green buttons: "New Complaint", "Delete Complaint", and "Complaint solved". To the right of the table are two more green buttons: "Modify Complaint details" and "View Complaint details".

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

The screenshots illustrate the process of adding a new complaint. In the first screenshot, the user has filled out the required fields: Complaint ID (Com6), Resident Username (keita999), and Description (Prices at XX vendor is too expensive). The 'New Complaint' button is highlighted in green, indicating it is the active or intended action. In the second screenshot, a confirmation dialog box titled 'Complaint adding successful' appears, containing the message 'Complaint adding successful' and an 'OK' button. This dialog obscures the bottom portion of the form and the 'Cancel' button. The background of the second screenshot shows the completed form fields.

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.

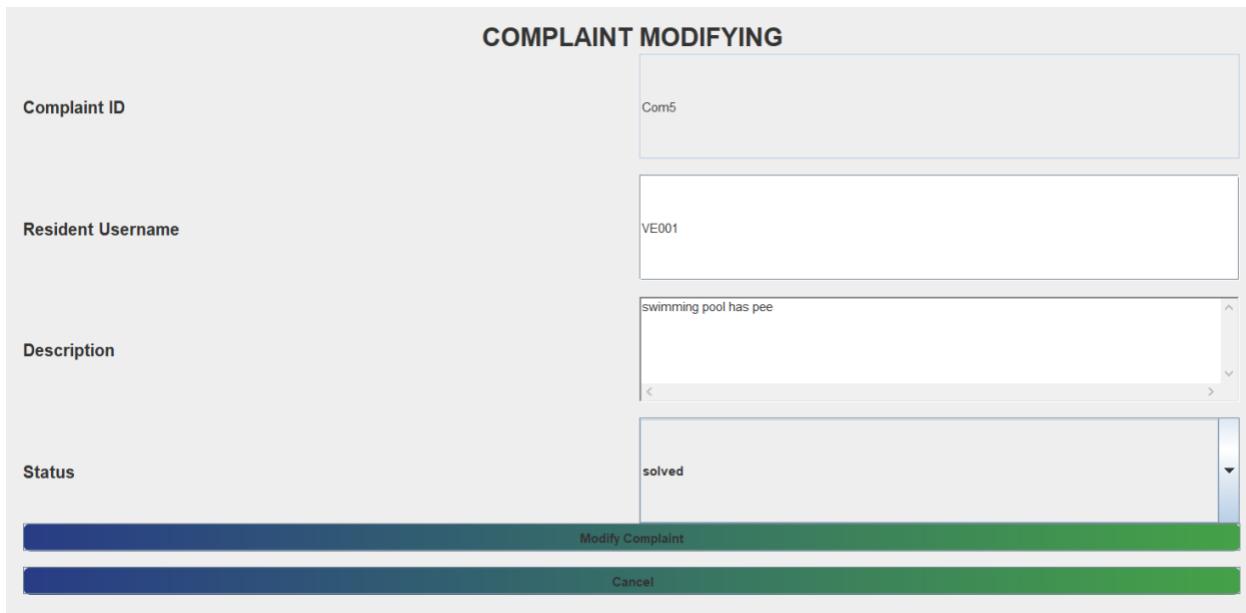
### 8.2.5.2 Modify Complaint details

**COMPLAINT MODIFYING**

Complaint ID	Com5
Resident Username	VE001
Description	swimming pool has pee
Status	solved

**Modify Complaint**

**Cancel**



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

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOVED
Com3	Joy101	Earthquake	UNSOVED
Com4		Earthquake	UNSOVED
Com5		swimming pool has pee	UNSOVED
Com6		prices at XX vendor is too expensive	UNSOVED

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

The screenshot shows a list of complaints on the right side of the screen. A modal window titled 'Confirmation message' is open over the list, asking 'Change complaint status to solved?'. The modal has 'Yes' and 'No' buttons. The complaints listed are:

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOLVED
Com3	Joy101	Earthquake	UNSOLVED
Com4		Earthquake	UNSOLVED
Com5		swimming pool has pee	UNSOLVED
Com6		Prices at XX vendor is too expensive	UNSOLVED

On the left, there is a sidebar with various management options: Profile, Unit Management, Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, and Logout. There is also a search bar at the top.

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

The screenshot shows a user interface for managing employees. On the left is a dark sidebar menu with icons and labels: Profile, Unit Management, Employee Management (highlighted), Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, and Logout. At the top right is a search bar with placeholder text "Search here..." and a user ID "AD01". The main area contains a table with columns: Employee ID, Name, Gender, Contact number, Salary, and Position. The table lists six employees: Alex (TN001), Brayden (TN002), JoJo (CN001), JiaYi (CN002), Alan (SG001), and Jojo (SG002). Below the table are four buttons: "Add New Employee" (green), "Delete Employee" (green), "Modify Employee Info" (blue), and "View Employee Info" (blue).

Employee ID	Name	Gender	Contact number	Salary	Position
TN001	Alex	F	011-12679348	1200	Technician
TN002	Brayden	F	012-22341627	1200	Technician
CN001	JoJo	F	011-72921342	1200	Cleaner
CN002	JiaYi	M	012-12025647	1200	Cleaner
SG001	Alan	F	011-1456327	1200	Security Guard
SG002	Jojo	F	019-3453321	1200	Security Guard

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

The screenshot shows the Parkhill.R application's main dashboard. On the left is a sidebar with various icons and links: Profile, Unit Management, Employee Management, Complaint Management, Facility Management, Facility Booking, Vendor, and Logout. The main area displays a table of employees with columns: Employee ID, Name, Gender, Contact number, Salary, and Position. A modal dialog titled "Position of Employee" is open, asking "Which employee do you want to add?" with a dropdown menu showing "Security Guard". At the bottom of the screen are four buttons: Add New Employee, Modify Employee Info, Delete Employee, and View Employee Info.

Employee ID	Name	Gender	Contact number	Salary	Position
TN001	Alex	F	011-12679348	1200	Technician
TN002	Brayden	F	012-22341627	1200	Technician
CN001			011-72921342	1200	Cleaner
CN002			012-12025647	1200	Cleaner
SG001			011-1456327	1200	Security Guard
SG002	Jojo	F	019-3453321	1200	Security Guard

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

The screenshot shows the "Employee ADDING" form. It includes fields for Employee ID (SG), Name (Ali), Gender (Male selected), Contact Number (011-11111111), Salary (RM) (2000), and Position Name (Security Guard). At the bottom are "Add New Employee" and "Cancel" buttons. A modal dialog titled "Position of Employee" is open, asking "Which employee do you want to add?" with a dropdown menu showing "Security Guard".

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.

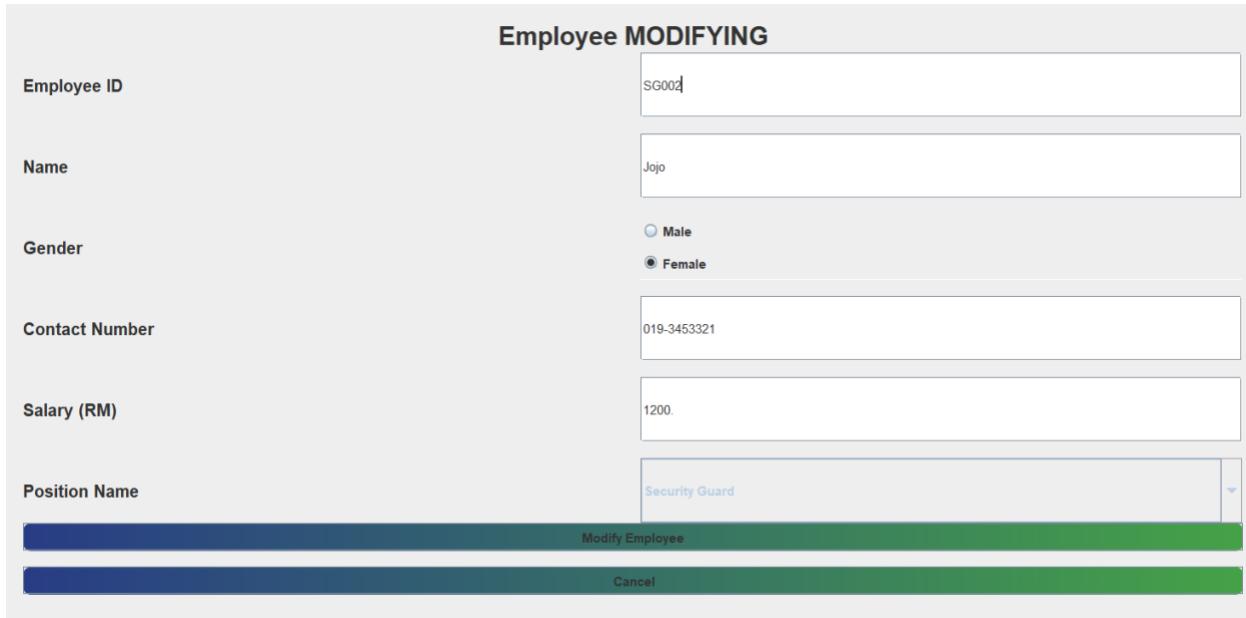
### 8.2.6.2 Modify Employee Info

**Employee MODIFYING**

Employee ID	SG002
Name	Jojo
Gender	<input type="radio"/> Male <input checked="" type="radio"/> Female
Contact Number	019-3453321
Salary (RM)	1200.
Position Name	Security Guard

[Modify Employee](#)

[Cancel](#)



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

The screenshot shows a user interface for managing employees. On the left is a sidebar with various icons and links: Profile, Unit Management, Employee Management (with two entries), Complaint Management, Facility Management, Facility Booking, Vendor, and Logout. The main area has a search bar at the top and the text 'AD01' below it. A table lists employees with columns: Employee ID, Name, Gender, Contact number, Salary, and Position. An alert dialog box is overlaid on the table, asking 'Are you sure to delete this Employee?' with 'Yes' and 'No' buttons. At the bottom of the screen are four green buttons: 'Add New Employee', 'Delete Employee' (highlighted in yellow), 'Modify Employee Info', and 'View Employee Info'.

Employee ID	Name	Gender	Contact number	Salary	Position
TN001	Alex	F	011-12679348	1200	Technician
TN002	Brayden	F	012-22341627	1200	Technician
CN001	CN002	F	011-72921342	1200	Cleaner
CN002			012-12025647	1200	Cleaner
SG001	Jojo	F	011-1456327	1200	Security Guard
SG002			019-3453321	1200	Security Guard

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

BOOKING DETAILS	
Booking ID	Book2
Facility ID	BH1
Resident Username	Mike1001
Date (MM.dd.yyyy)	01.24.2023
Start time (HHmmss)	090000
End time (HHmmss)	110000
Issued by Parkhill Residence	
<a href="#">Close</a>	

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

The screenshot shows a user interface for facility management. On the left is a dark sidebar menu with icons and labels: Profile, Unit Management, Employee Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, and Logout. The main area has a search bar at the top labeled "Search here...". Below it, the user ID "AD01" is displayed. A table lists facilities with columns for Facility ID and Name. The facilities listed are BH1 (Badminton Hall 1), BH2 (Badminton Hall 2), GR1 (Gymnasium Room 1), GR2 (Gymnasium Room 2), FR1 (Function Room 1), and FR2 (Function Room 2). At the bottom of the main area are four buttons: "Add New Facility", "Delete Facility", "Modify Facility Details", and "View Facility Details".

Facility ID	Name
BH1	Badminton Hall 1
BH2	Badminton Hall 2
GR1	Gymnasium Room 1
GR2	Gymnasium Room 2
FR1	Function Room 1
FR2	Function Room 2

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

The screenshot shows a user interface titled "FACILITY ADDING". It contains two input fields: "Facility ID" with the value "BH3" and "Name" with the value "Badminton Hall 3". At the bottom of the form are two buttons: "Add New Facility" (highlighted in green) and "Cancel".

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.

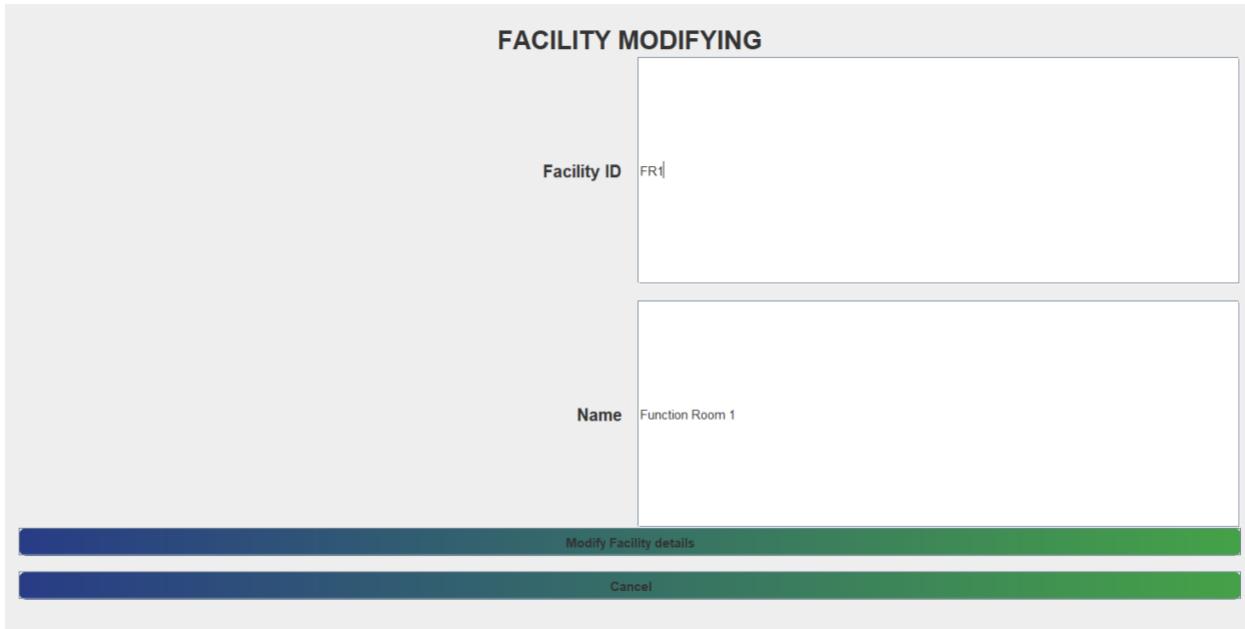
#### 8.2.7.2 Modify Facility Details

**FACILITY MODIFYING**

Facility ID	FR1
Name	Function Room 1

**Modify Facility details**

**Cancel**



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.

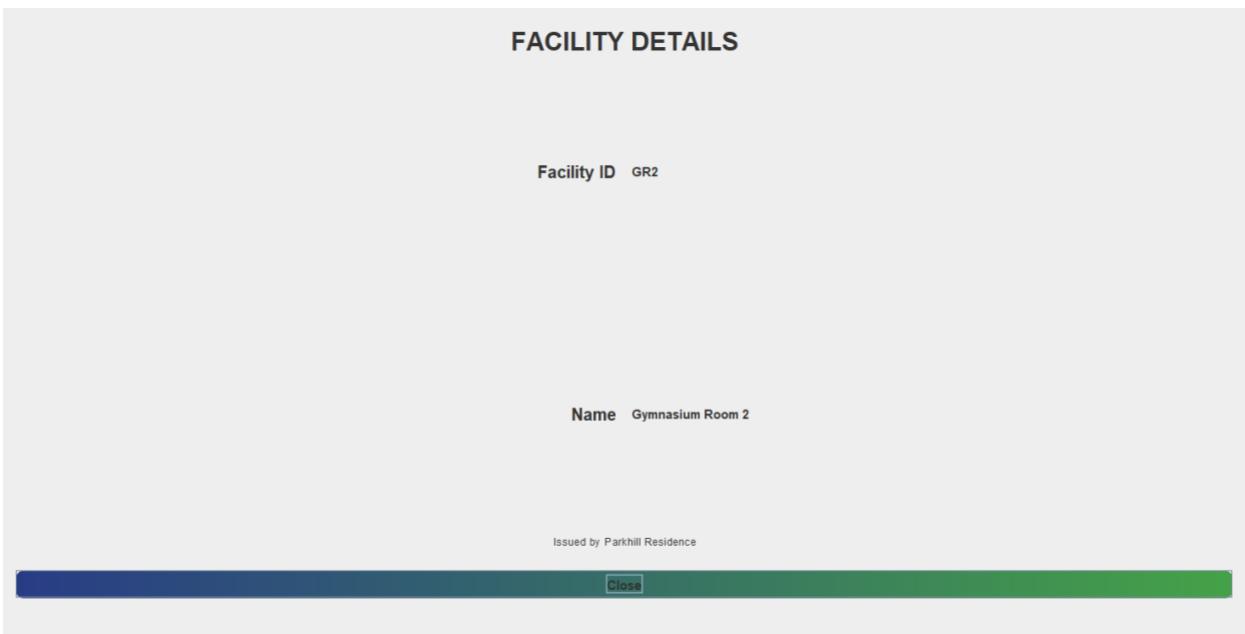
### 8.2.7.3 Delete Facility

The screenshot shows the Parkhill.R application interface. On the left is a sidebar with various management options: Profile, Unit Management, Employee Management, Complaint Management, Facility Management, Facility Booking, Vendor, and Logout. The main area displays a list of facilities with columns for Facility ID and Name. A modal dialog box titled "Delete confirmation" is overlaid on the page, asking "Are you sure to delete this Facility?" with "Yes" and "No" buttons. Below the modal are four buttons: "Add New Facility", "Delete Facility", "Modify Facility Details", and "View Facility Details". The "Delete Facility" button is highlighted with a red border.

Facility ID	Name
BH1	Badminton Hall 1
BH2	Badminton Hall 2
FR1	Gymnasium Room 1
FR2	Gymnasium Room 2
	Function Room 1
	Function Room 2

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

Booking ID	Facility ID	Resident Username	Date	Start time	End time
Book1	BH1	Mike1001	01.22.2023	130000	150000
Book2	BH1	Mike1001	01.24.2023	090000	110000

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

**FACILITY BOOKING FORM**

Booking ID	Book3
Facility	Gymnasium Room 1
Resident Username	Keita
Date (MM.dd.yyyy)	
Start time (HHmmss)	160000
End time (HHmmss)	170000

A modal dialog box is displayed in the center of the form, showing an error message: "Facility not found" with a red X icon, and "Facility not existed" with an OK button.

At the bottom of the form, there are two buttons: "Make new booking" (highlighted in green) and "Cancel".

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.

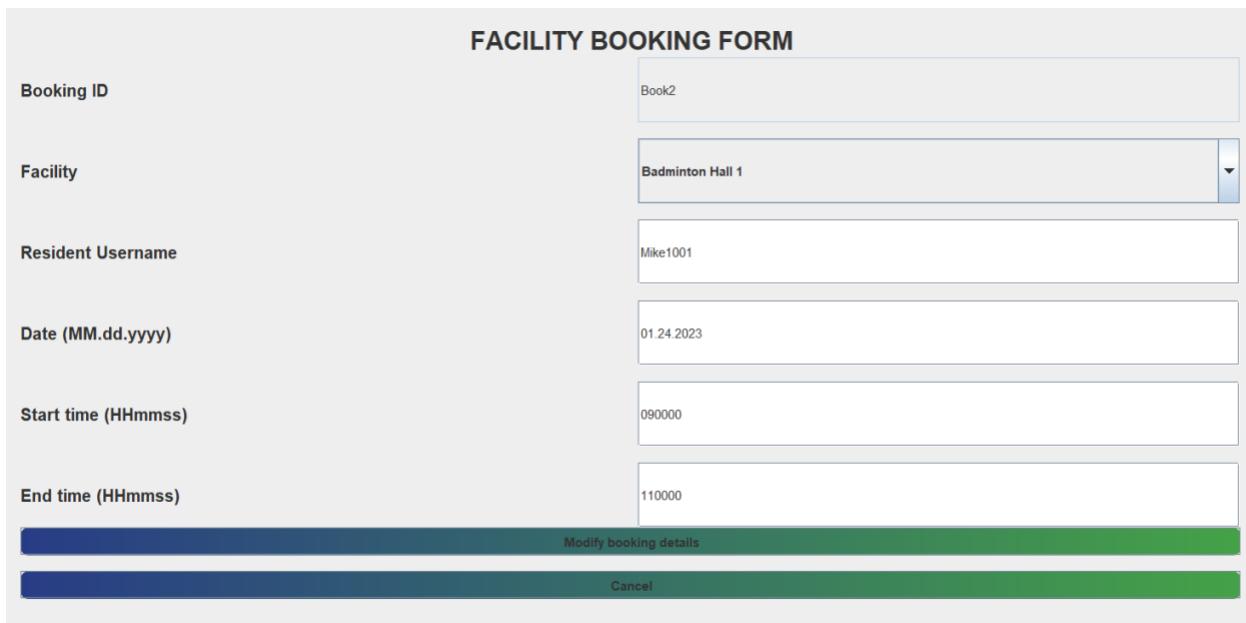
### 8.2.8.2 Modify Booking Info

**FACILITY BOOKING FORM**

Booking ID	Book2
Facility	Badminton Hall 1
Resident Username	Mike1001
Date (MM.dd.yyyy)	01.24.2023
Start time (HHmmss)	090000
End time (HHmmss)	110000

[Modify booking details](#)

[Cancel](#)

The image shows a 'Facility Booking Form' interface. It consists of a grid of input fields and labels. The fields are: Booking ID (Book2), Facility (Badminton Hall 1), Resident Username (Mike1001), Date (01.24.2023), Start time (090000), and End time (110000). Below the form is a green horizontal bar with two buttons: 'Modify booking details' and 'Cancel'. The entire interface is contained within a light gray box.

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

The screenshot shows the Parkhill.R software interface. On the left is a dark sidebar with various management options: Profile, Unit Management, Resident Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The Resident Management option is highlighted with a green bar. At the top right, there's a search bar with placeholder text "Search here..." and a user ID "AD01". The main area displays a booking list with columns: Booking ID, Facility ID, Resident Usern..., Date, Start time, and End time. Two bookings are listed: Book1 (Date 01.22.2023, Start 130000, End 150000) and Book2 (Date 01.24.2023, Start 090000, End 110000). A modal window titled "Delete confirmation" is open over the list, asking "Are you sure to delete this booking?". It has "Yes" and "No" buttons. Below the modal are two buttons: "Delete Booking" and "View Booking details".

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

**FACILITY BOOKING FORM**

<b>Booking ID</b>	Book2
<b>Facility</b>	Badminton Hall 1
<b>Resident Username</b>	Mike1001
<b>Date (MM.dd.yyyy)</b>	01.24.2023
<b>Start time (HHmmss)</b>	090000
<b>End time (HHmmss)</b>	110000

Issued by Parkhill Residence

**Close**

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

The screenshot shows a dark-themed user interface for vendor management. On the left is a sidebar with icons and labels for Profile, Unit Management, Vendor Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main area has a search bar at the top. Below it, the text 'AD01' is displayed. A table lists two vendors: VE002 (MalaiPo, F, 013-1234152, floor 3 mamak, 600, 0) and VE001 (Abunene, M, 011-1743475, floor 1 canteen, 500, 100). Below the table are four buttons: 'Add New Vendor' (green), 'Modify Vendor Info' (blue), 'Delete Vendor' (green), and 'View Vendor Info' (blue).

Vendor User...	Name	Gender	Contact Num...	Vendor Unit	Monthly pay...	Unpaid pay...
VE002	MalaiPo	F	013-1234152	floor 3 mamak	600	0
VE001	Abunene	M	011-1743475	floor 1 canteen	500	100

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

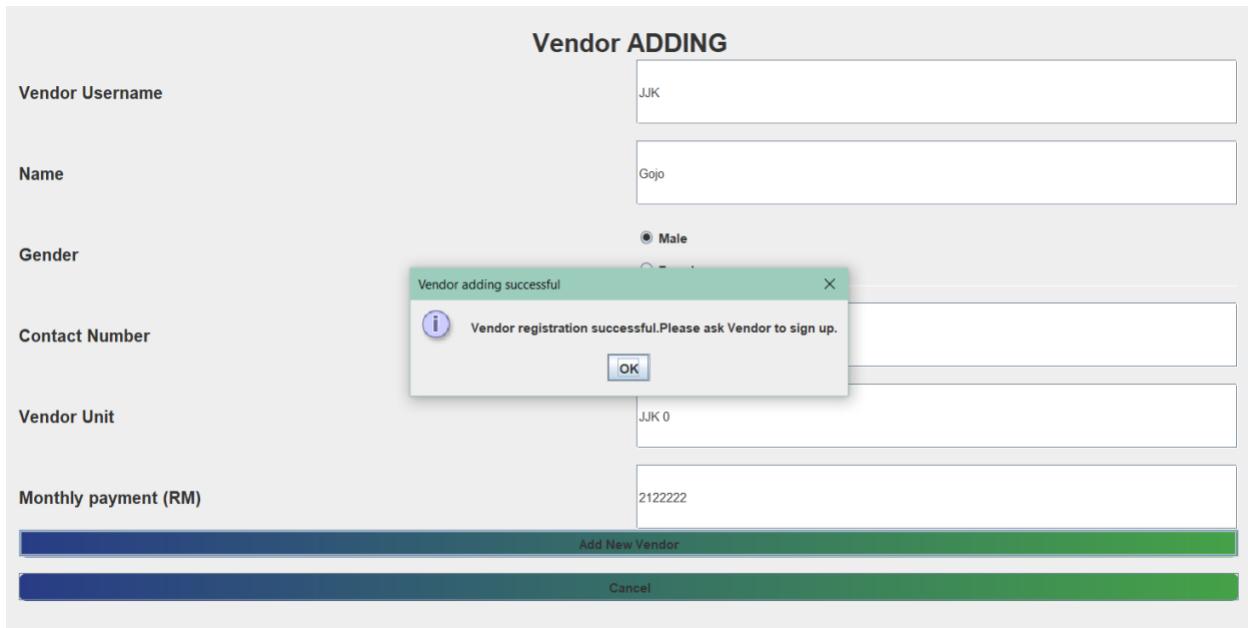
vendors, add a new vendor, modify vendor details, delete vendors, view vendor details, and approve vendor registration.

#### 8.2.9.1 Add New Vendor

**Vendor ADDING**

Vendor Username	JJK
Name	Gojo
Gender	<input checked="" type="radio"/> Male <input type="radio"/> -
Contact Number	<div style="background-color: #007bff; color: white; padding: 5px; text-align: center;">Vendor adding successful</div> <div style="background-color: #f8d7da; border: 1px solid #ccc; border-radius: 10px; padding: 10px; width: fit-content; margin: auto;"><p><span style="color: #007bff; font-size: 1.5em;">i</span> Vendor registration successful. Please ask Vendor to sign up.</p><p style="text-align: center;"><span style="border: 1px solid #ccc; padding: 2px;">OK</span></p></div>
Vendor Unit	JJK 0
Monthly payment (RM)	2122222

Add New VendorCancel



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

**Vendor INFO MODIFYING**

Vendor Username	JJK
Name	Gojo
Gender	<input checked="" type="radio"/> Male <input type="radio"/> Female
Contact Number	012-3456785
Vendor Unit	JJK 0
Monthly payment (RM)	511.0
Unpaid payment (RM)	0.0

[Modify Vendor Info](#)

[Cancel](#)

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

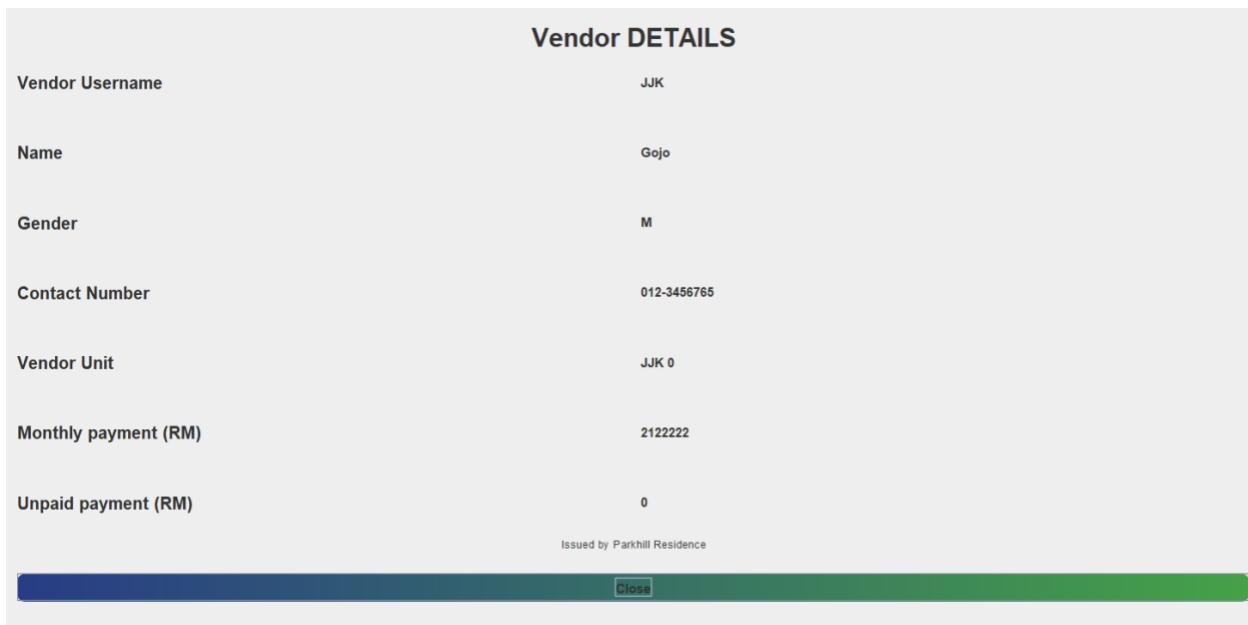
Vendor User...	Name	Gender	Contact Num...	Vendor Unit	Monthly pay...	Unpaid pay...
VE002	M	Male	152	floor 3 mamak	600	0
VE003	JJK	Male	785	JJK 0	212222	0

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

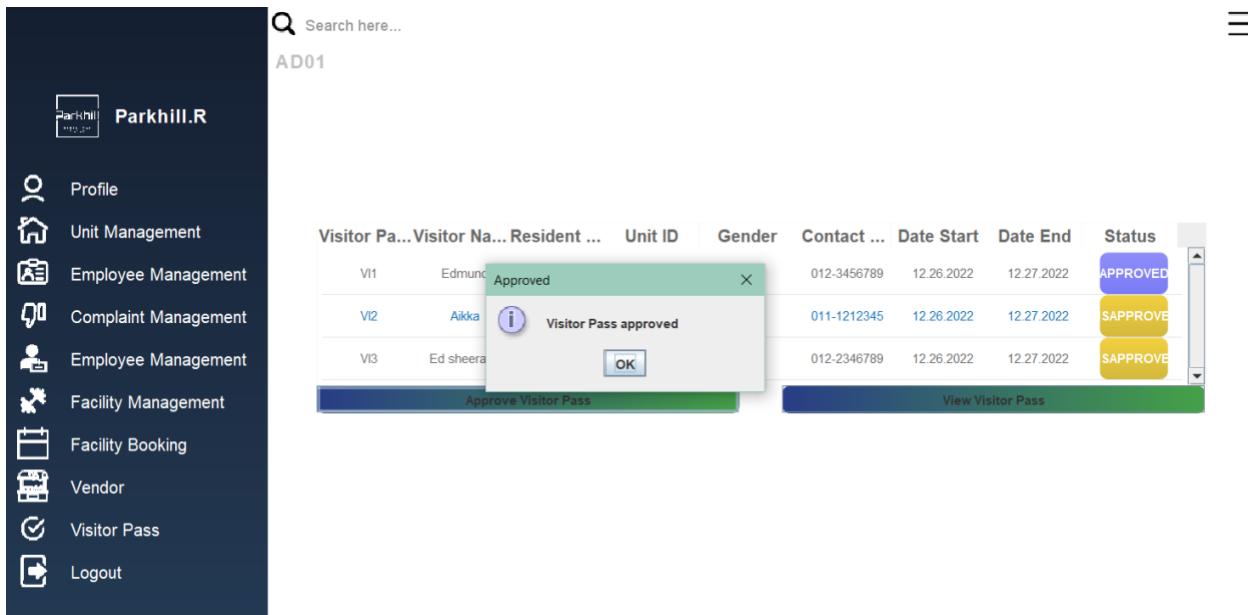
The screenshot shows a web-based application interface for managing visitor passes. The left sidebar contains a navigation menu with icons and labels: Profile, Unit Management, Employee Management, Complaint Management, Employee Management, Facility Management, Facility Booking, Vendor, Visitor Pass, and Logout. The main content area has a search bar at the top labeled "Search here...". Below it, the text "AD01" is displayed. The central part of the screen shows a table of visitor pass records:

Visitor Pa...	Visitor Na...	Resident ...	Unit ID	Gender	Contact ...	Date Start	Date End	Status
VI1	Edmund	Mike1001	A401	M	012-3456789	12.26.2022	12.27.2022	APPROVED
VI2	Aikka	Andrew123	A402	M	011-1212345	12.26.2022	12.27.2022	SAPPROVE
VI3	Ed sheerann	Mike1001	A401	M	012-2346789	12.26.2022	12.27.2022	SAPPROVE

At the bottom of the table are two buttons: "Approve Visitor Pass" and "View Visitor Pass".

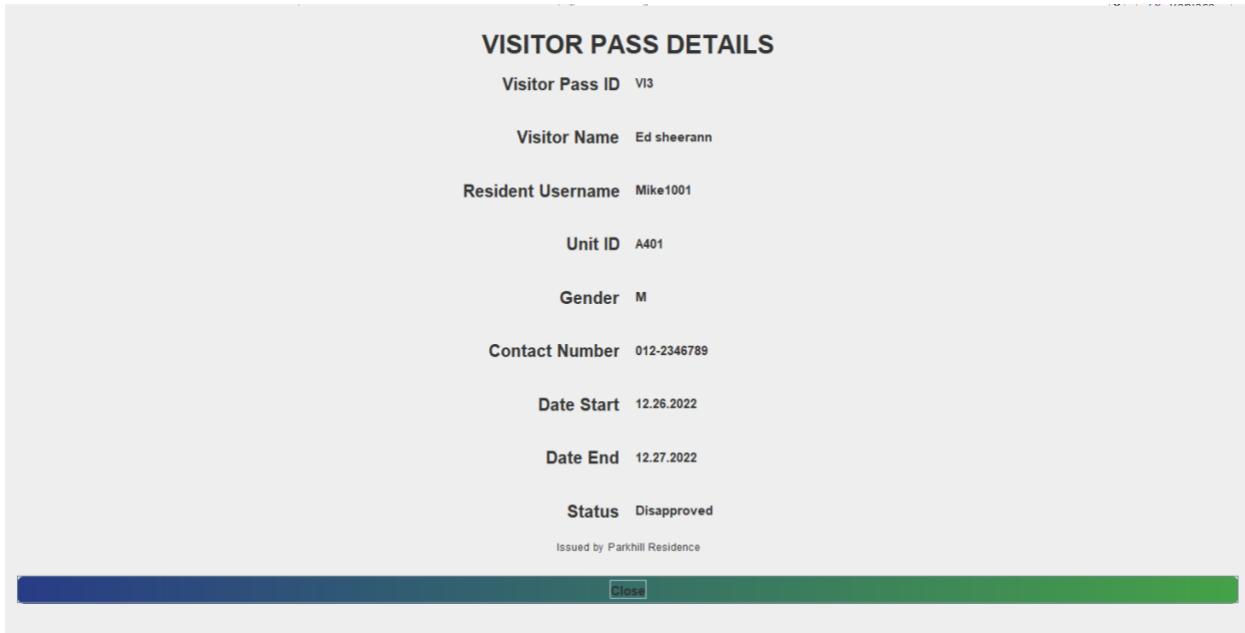
Users who navigate the visitor pass management section in the menu will be taken to this interface to access and view the list of visitor pass records. The interface also allows users to search for a specific visitor pass, approve visitor pass and view visitor pass details.

### 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



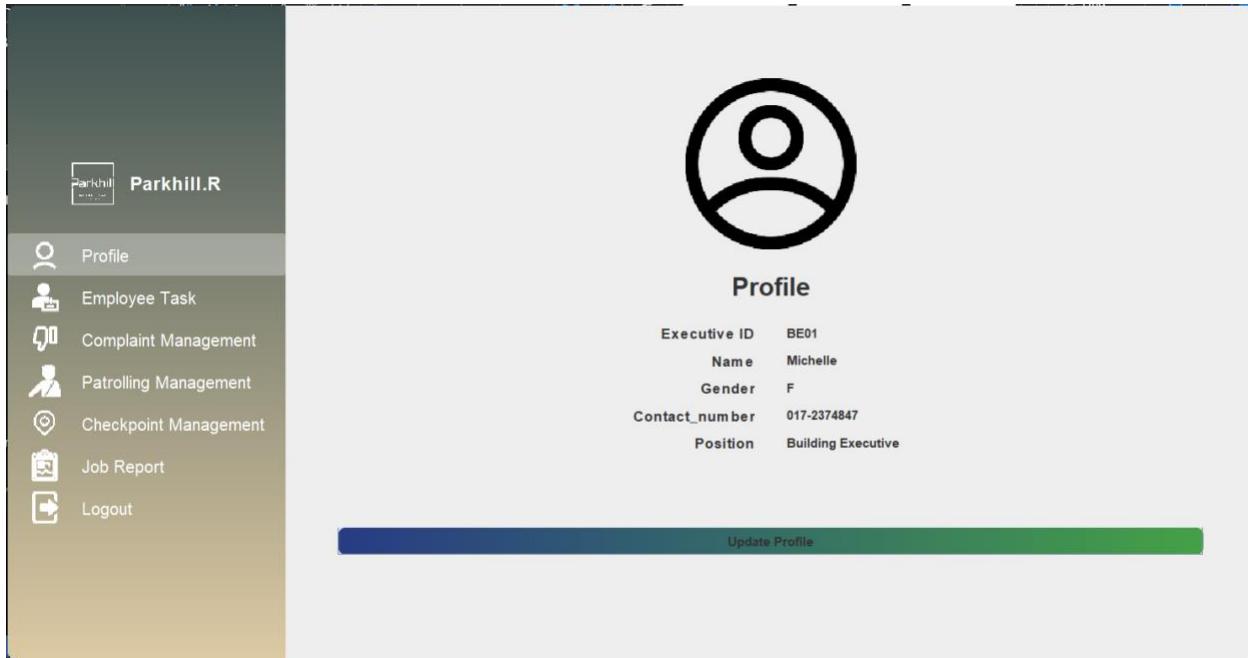
To view a visitor pass, users must choose the one they want to view and click the "view visitor pass " button. This will direct them to a new interface displaying visitor pass details.

## 8.3 Building Executive

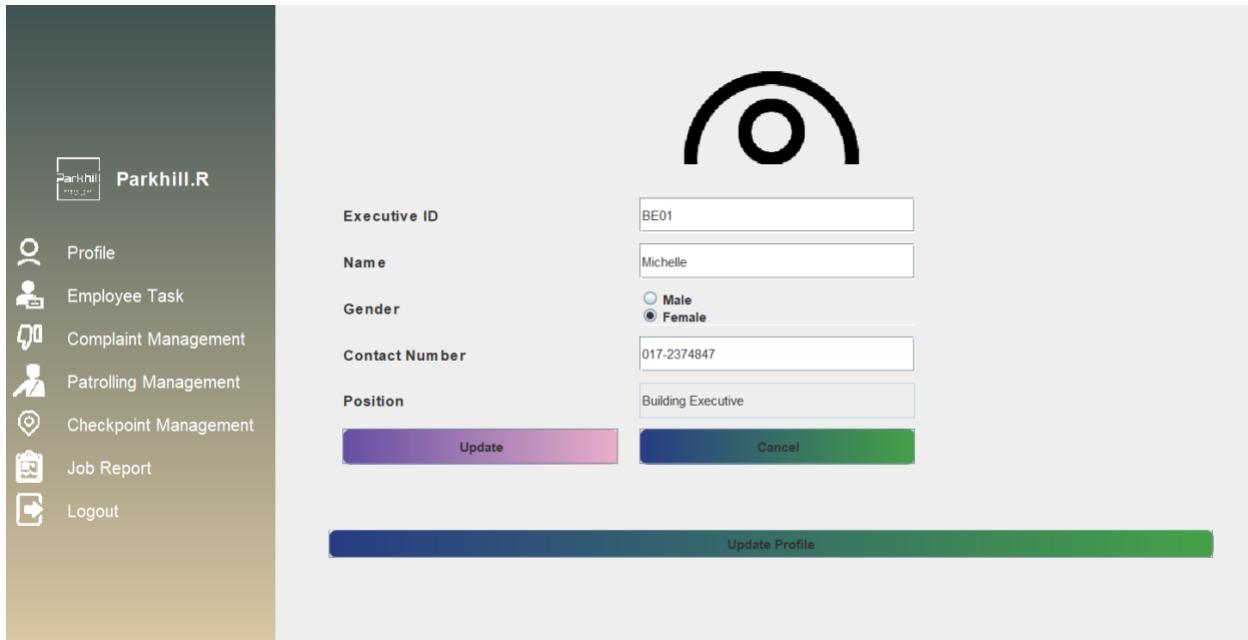
### 8.3.1 Functional requirement

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

Task ID	Employee ID	Description	Status
Task1	SG0001	check gym	done
Task2	CN0001	check floor 1 washroom	undone

**Buttons:**

- Assign New Employee Task
- Modify Employee Task Details
- Delete Employee Task
- View Employee Task Details

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

**Employee Task ADDING**

Employee Task ID	Task3
Employee ID	SG002
Description	Needs detailed report of incident to be submitted to the police station latest 02/03/2023

**Buttons:**

- Assign Employee\_Task
- Cancel

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.

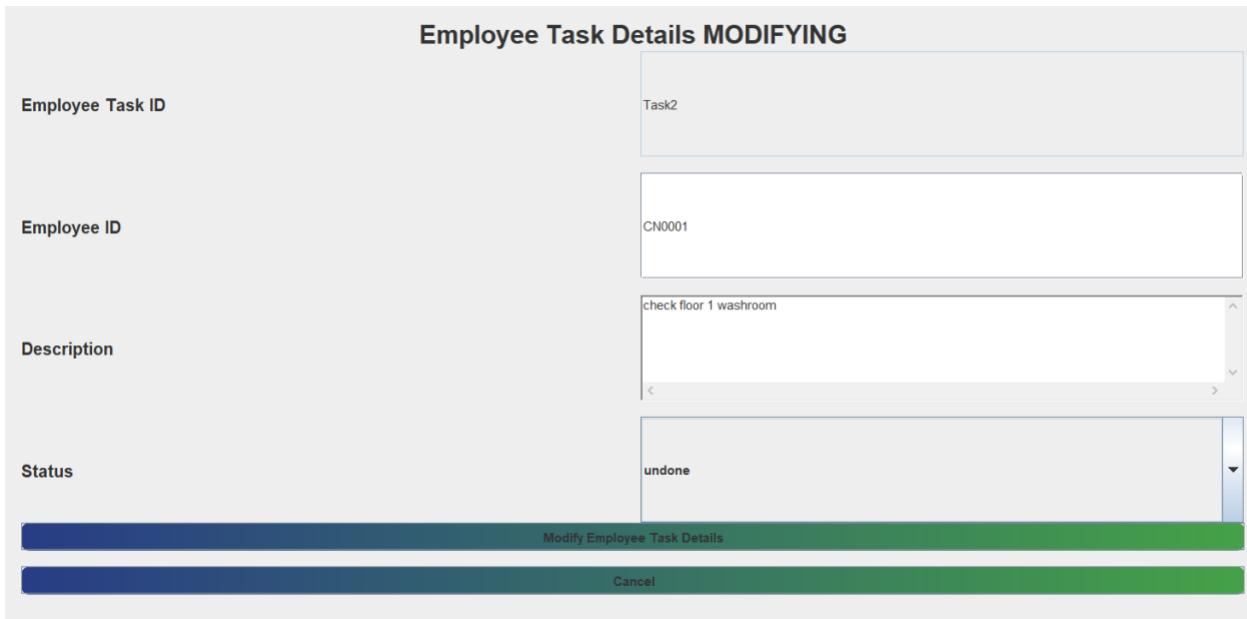
### 8.3.3.2 Modify Employee Task Details

**Employee Task Details MODIFYING**

Employee Task ID	Task2
Employee ID	CN0001
Description	check floor 1 washroom
Status	undone

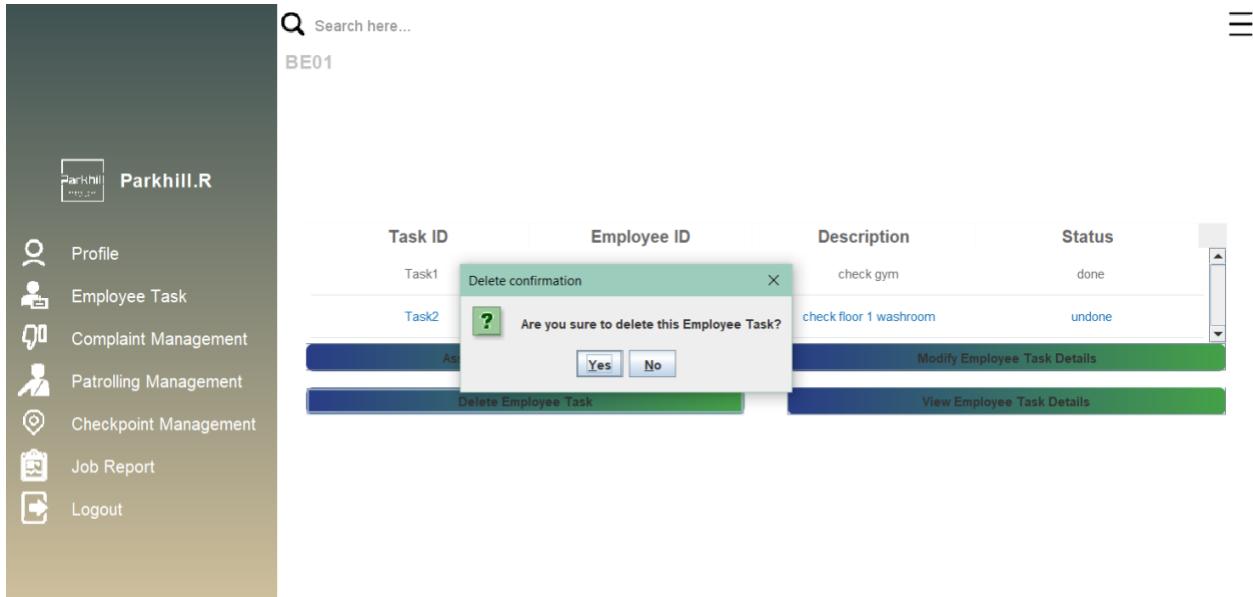
**Modify Employee Task Details**

**Cancel**



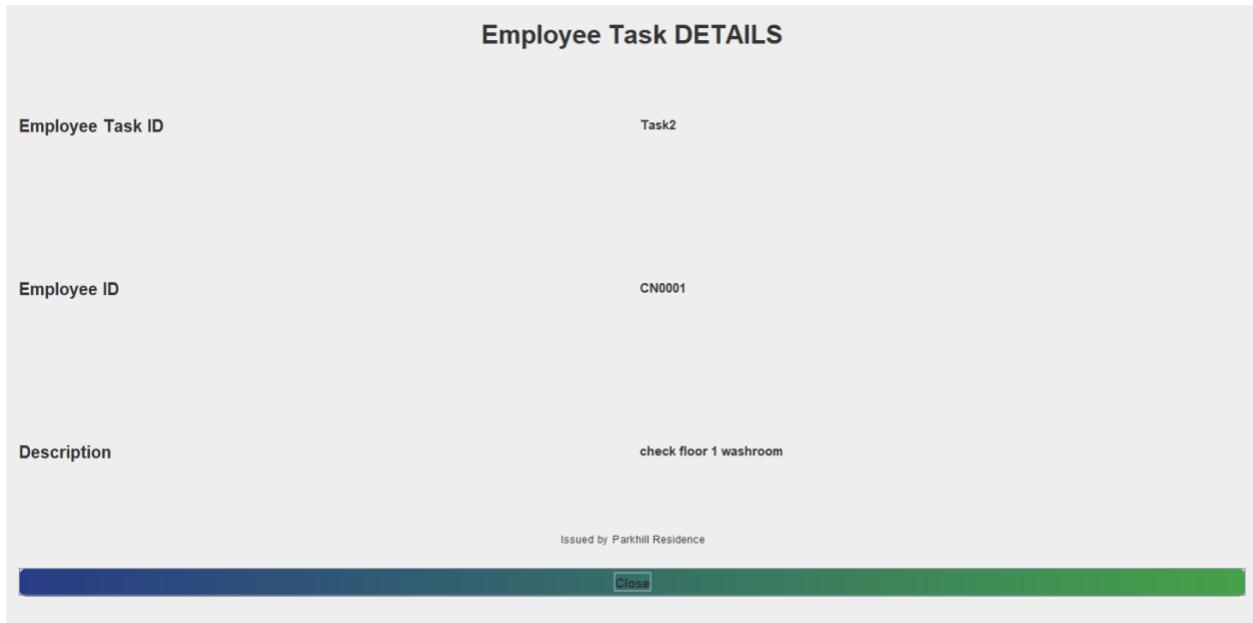
To modify a task, users must click on the task they wish to modify and then proceed by clicking the "modify employee task" button. The users will then be directed to an interface to modify task details. Afterward, users can click the "modify employee task" button to submit their modified task details.

### 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

The screenshot shows a user interface for managing complaints. On the left is a sidebar with a logo for 'Parkhill.R' and a search bar. The main area displays a table of complaints with columns for Complaint ID, Resident Username, Description, and Status. The status column uses colored buttons: blue for solved and yellow for unsolved. Buttons at the bottom allow viewing details or marking a complaint as solved.

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOLVED
Com3	Joy101	Earthquake	UNSOLVED
Com4	VE001	Earthquake	UNSOLVED
Com5	VE001	swimming pool has pee	UNSOLVED

**View Complaint details**  
**Complaint solved**

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

The screenshot shows a "Complaint Management" interface with a sidebar and a main content area. The sidebar includes links for Profile, Employee Task, Complaint Management, Patrolling Management, Checkpoint Management, Job Report, and Logout. The main content area displays a table of complaints:

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOLVED
Com3		Earthquake	UNSOLVED
Com4		Earthquake	UNSOLVED
Com5		swimming pool has pee	UNSOLVED

A confirmation dialog box is overlaid on the table, asking "Change complaint status to solved?". It contains "Yes" and "No" buttons. Below the table, two buttons are visible: "View Complaint details" and "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 screenshot shows the "Patrolling Management" section of the Building Executive page. On the left, there is a sidebar with the "Parkhill.R" logo and a search bar labeled "Search here...". Below the search bar, the identifier "BE01" is displayed. The main content area features a table with two rows of patrol data:

Patrol ID	Security Employee ID	Day	Time Start	Time End
Patrol1	SG001	Monday	130000	170000
Patrol2	SG002	Tuesday	120000	150000

Below the table, there are four buttons arranged in a 2x2 grid:

- Top-left: Setup new patrolling
- Top-right: Modify patrolling details
- Bottom-left: Delete patrolling
- Bottom-right: View patrolling details

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.

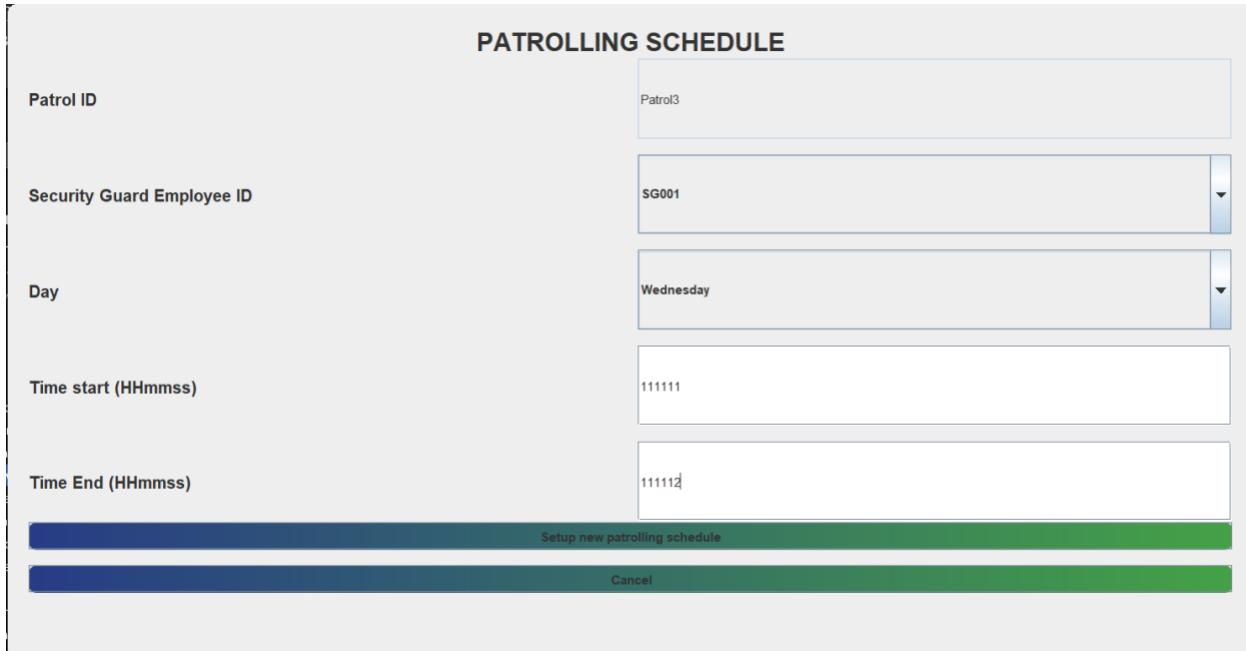
### 8.3.5.1 Setup new patrolling

**PATROLLING SCHEDULE**

<b>Patrol ID</b>	Patrol3
<b>Security Guard Employee ID</b>	SG001
<b>Day</b>	Wednesday
<b>Time start (HHmmss)</b>	111111
<b>Time End (HHmmss)</b>	111112

**Setup new patrolling schedule**

**Cancel**



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.

**PATROLLING SCHEDULE**

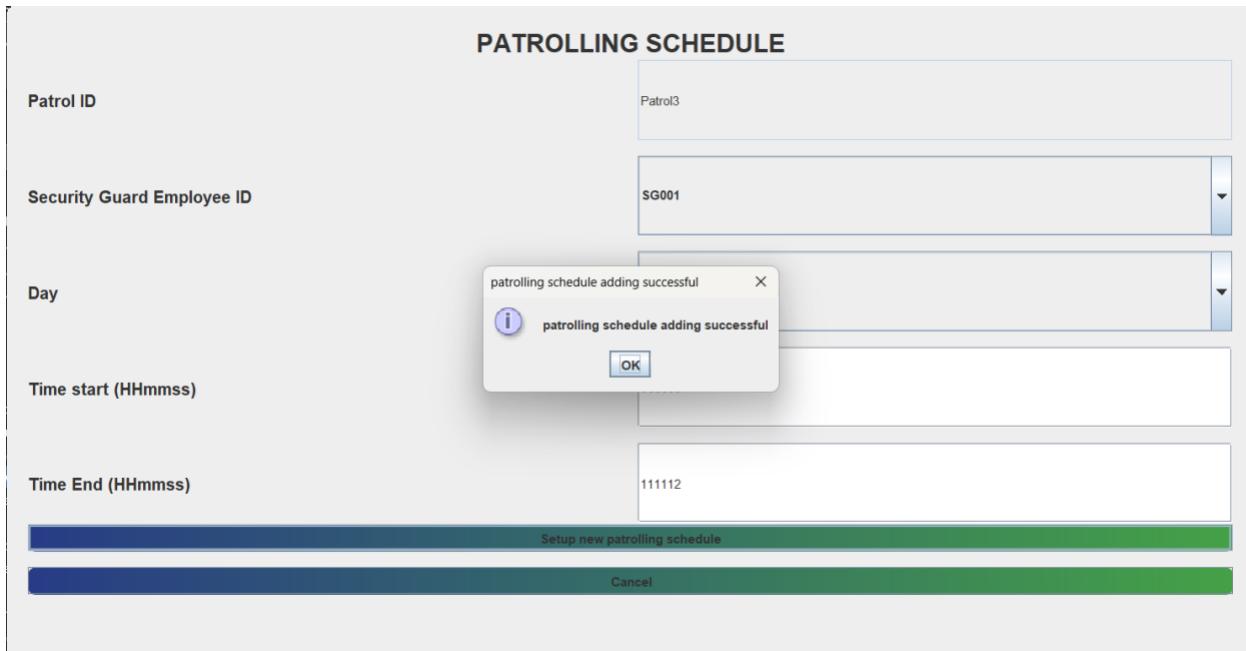
<b>Patrol ID</b>	Patrol3
<b>Security Guard Employee ID</b>	SG001
<b>Day</b>	Wednesday
<b>Time start (HHmmss)</b>	111111
<b>Time End (HHmmss)</b>	111112

**Setup new patrolling schedule**

**Cancel**

patrolling schedule adding successful

OK



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

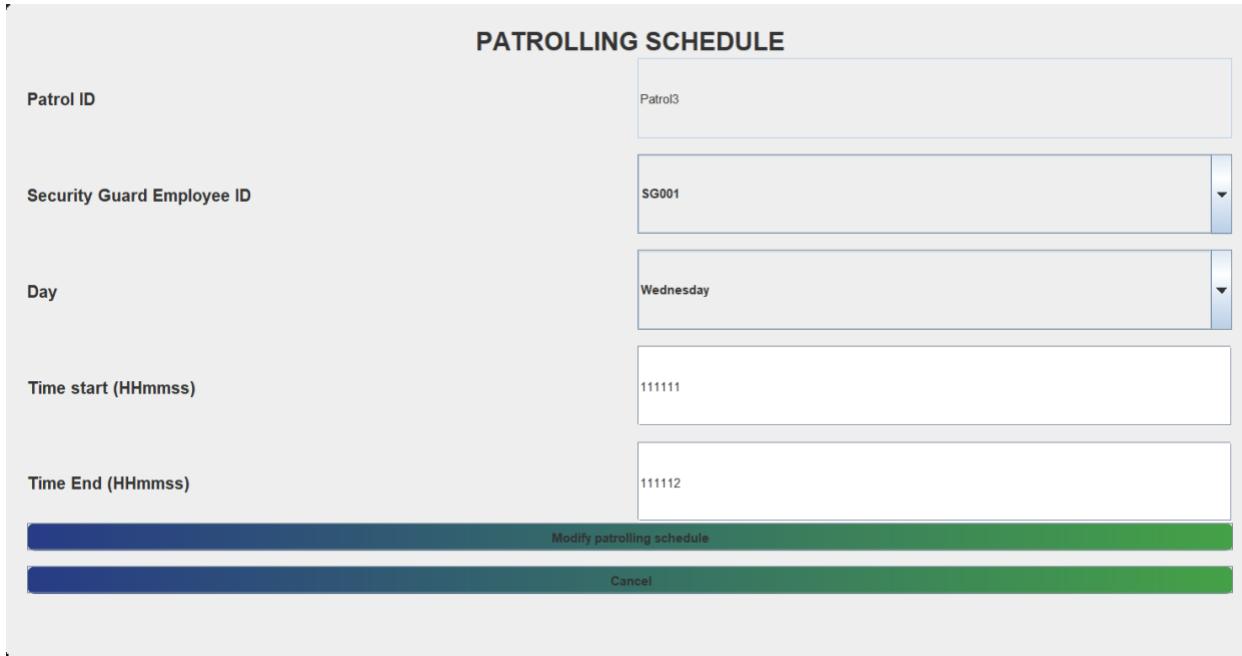
### 8.3.5.2 Modify patrolling details

**PATROLLING SCHEDULE**

Patrol ID	Patrol3
Security Guard Employee ID	SG001
Day	Wednesday
Time start (HHmmss)	111111
Time End (HHmmss)	111112

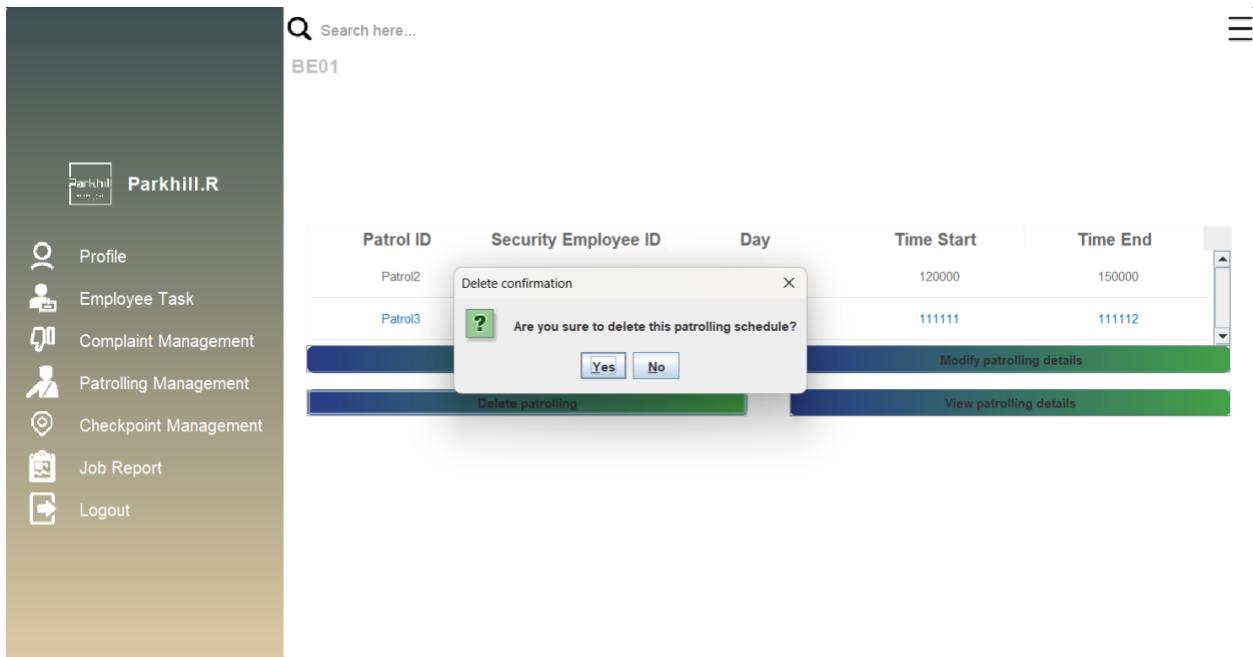
[Modify patrolling schedule](#)

[Cancel](#)



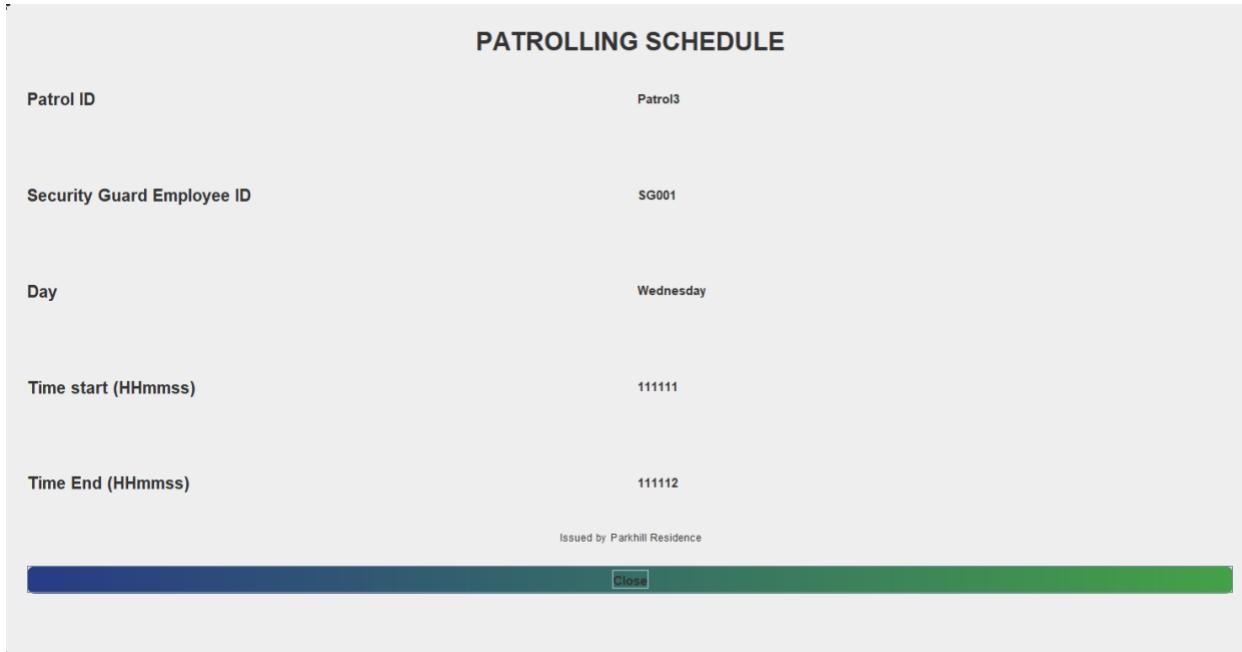
Users can modify specific patrolling schedules by clicking "Modify patrolling details". This will take them to an interface where they can edit the relevant information. Once the modifications are complete, the user can click "Modify patrolling details" to save the changes.

### 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

The screenshot shows a user interface for managing security checkpoints. On the left is a sidebar with a logo and several menu options: Profile, CheckPoint, Complaint Management, Patrolling Management, Checkpoint Management, Job Report, and Logout. The main area has a search bar at the top and displays the text "BE01". Below this, there is a table with columns for CheckPoint ID, Name, and Location. Two rows are visible: one for "CP001" located in "CarPark1" at "Block A floor 2 Car Park", and another for "CP002" located in "Washroom1" at "Block A floor 3 Washroom". At the bottom of the table are four buttons: "Add New CheckPoint" (green), "Delete CheckPoint" (green), "Modify CheckPoint Info" (green), and "View CheckPoint Info" (green).

CheckPoint ID	Name	Location
CP001	CarPark1	Block A floor 2 Car Park
CP002	Washroom1	Block A floor 3 Washroom

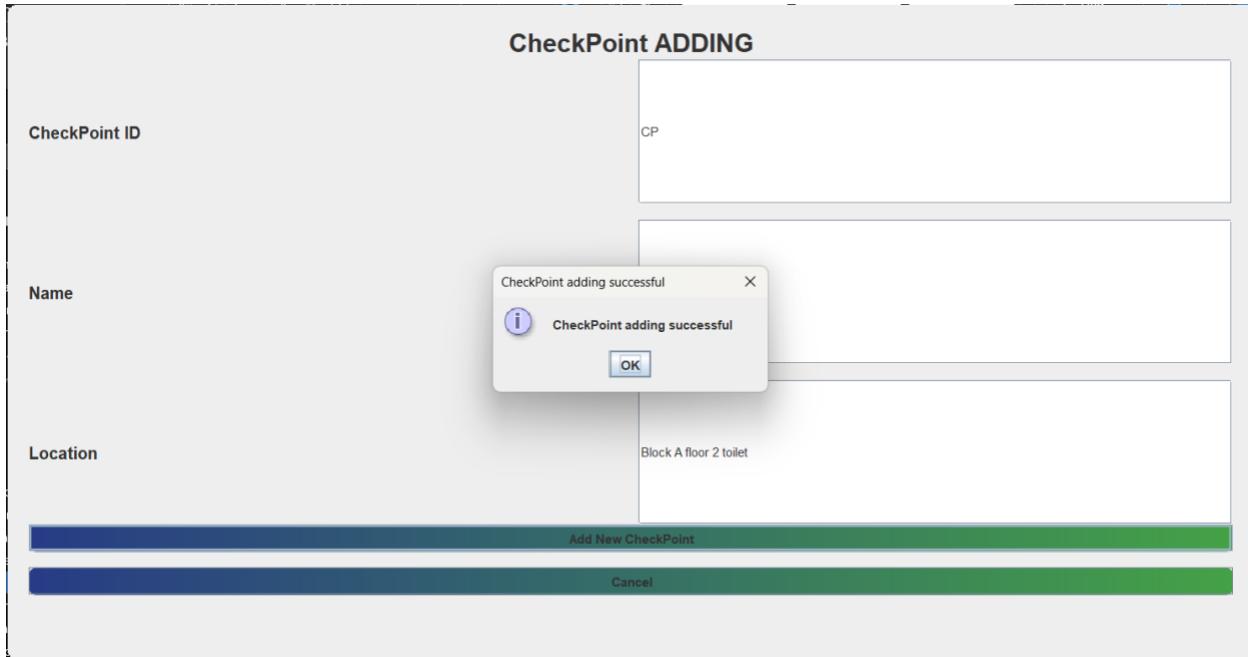
Building executives can navigate to the checkpoint management page by clicking on the corresponding button. On this page, they can view all information related to security check points, such as the checkpoint ID and its specific location.

### 8.3.6.1 Add New CheckPoint

The screenshot shows a user interface for adding a new checkpoint. The title 'CheckPoint ADDING' is at the top. There are three input fields: 'CheckPoint ID' with value 'CP', 'Name' with value 'Toile', and 'Location' with value 'Block A floor 2 toilet'. At the bottom are two buttons: a green 'Add New CheckPoint' button and a blue 'Cancel' button.

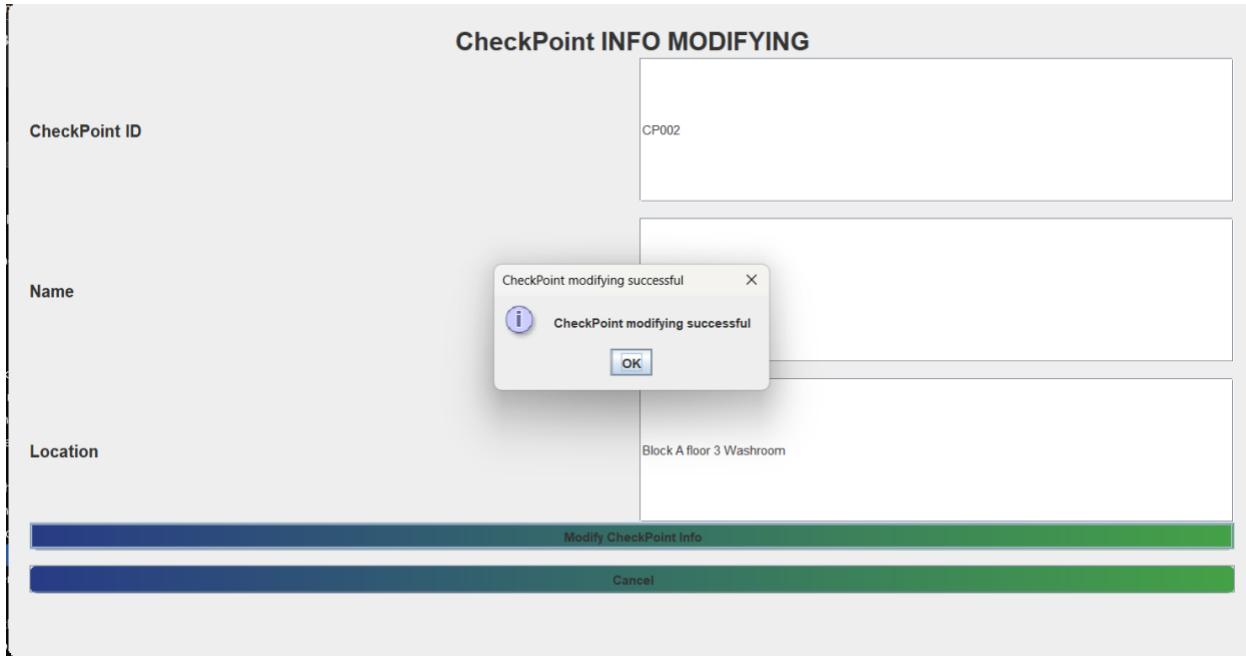
CheckPoint ADDING	
CheckPoint ID	CP
Name	Toile
Location	Block A floor 2 toilet
<input type="button" value="Add New CheckPoint"/>	
<input type="button" value="Cancel"/>	

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.

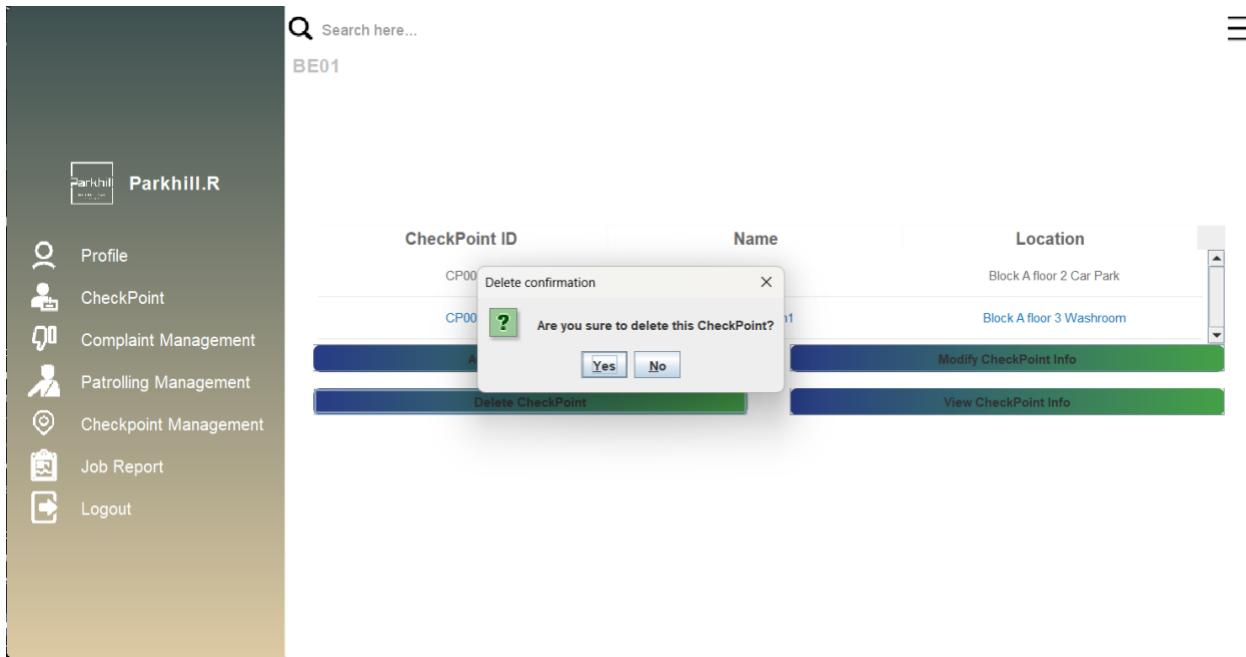


After adding a new checkpoint using the "Add New CheckPoint" button, the system will automatically display a prompt confirming the successful addition of the checkpoint.

### 8.3.6.2 Modify CheckPoint Info

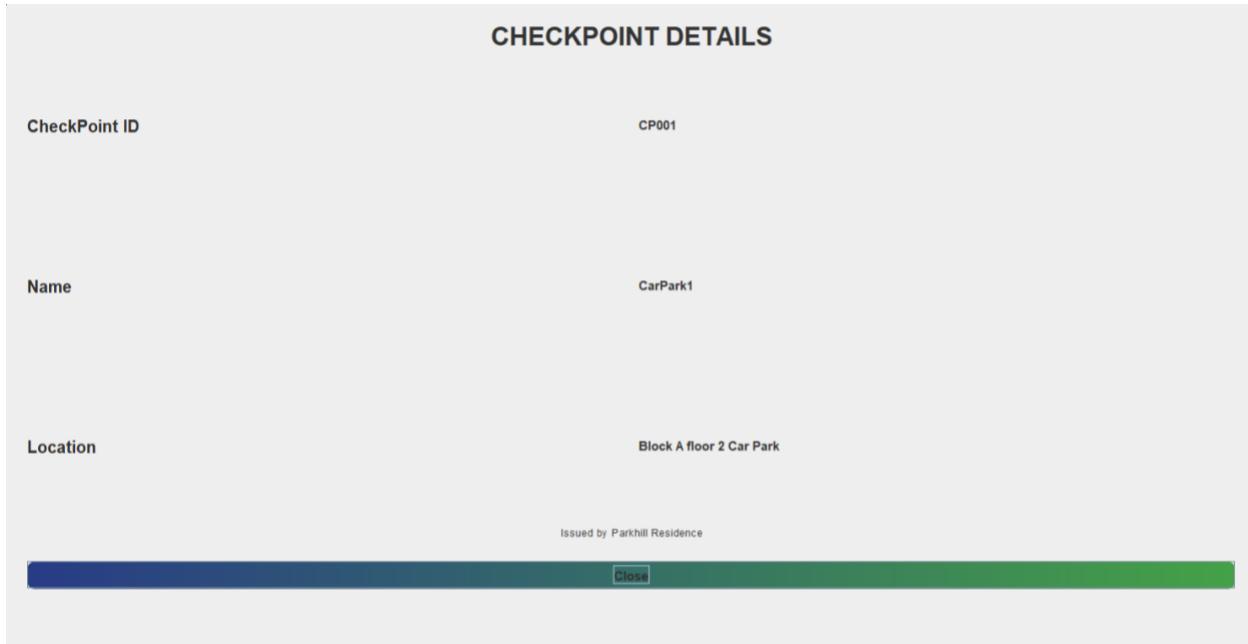


### 8.3.6.3 Delete CheckPoint



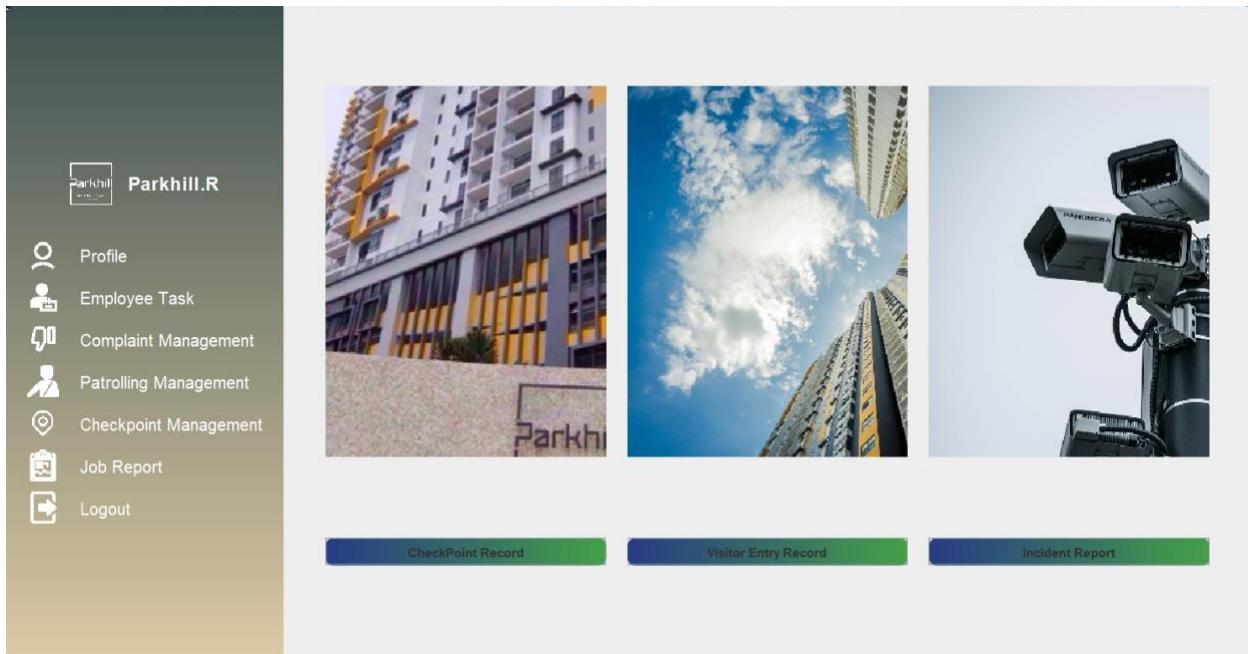
The user can select the row or column they want to delete and then click "Delete Check Point" to remove it from the list.

#### 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.

Security Guard Employee ID	CheckPoint ID	Date	Time
SG001	CP001	01.12.2023	140000
SG001	CP001	01.12.2023	140000
SG001	CP001	01.12.2023	140000
SG001	CP001	01.12.2023	140000
SG001	CP001	01.12.2023	140000

Close

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

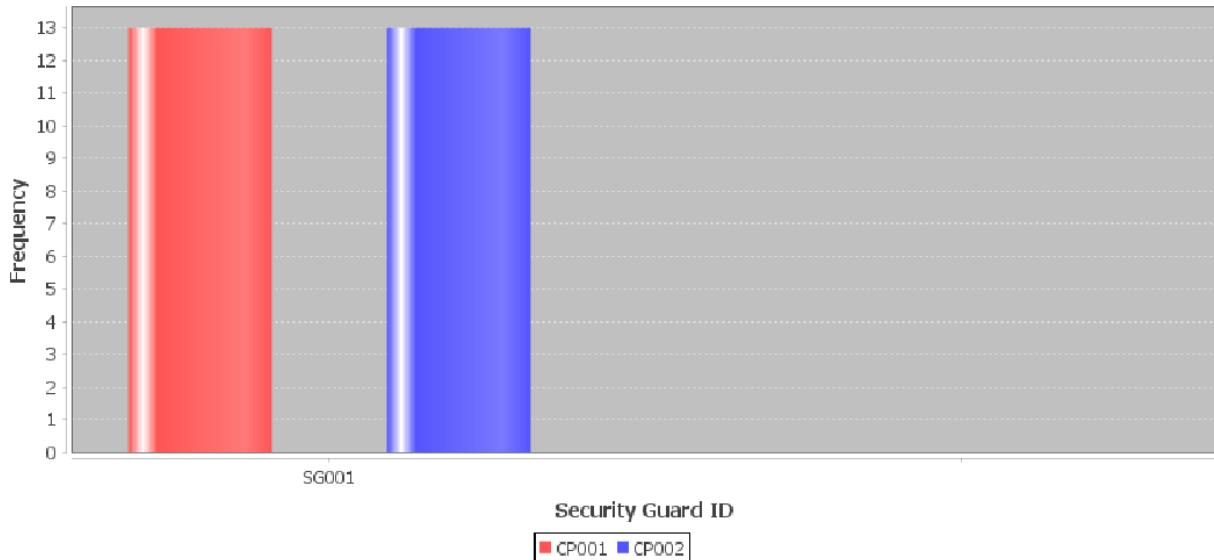
### 8.4.1 System Report Panel

```

 1 public reportPanel() throws IOException, ClassNotFoundException {
 2     JFreeChart checkPointRecordChart = ChartFactory.createBarChart("Security Guard CheckPoint Record", "Security Guard ID", "Frequency", getCheckPointDataset(), PlotOrientation.VERTICAL, true, true, false);
 3     JFreeChart executivePieChart = ChartFactory.createPieChart("Executive User Number", getExecutiveDataset(), true, true, false);
 4     JFreeChart complaintPieChart = ChartFactory.createPieChart("User Complaint Frequency", getUserComplaintDataset(), true, true, false);
 5     JFreeChart taskPieChart = ChartFactory.createPieChart("Employee Task Amount", getEmployeeTaskDataset(), true, true, false);
 6     JFreeChart bookingChart = ChartFactory.createBarChart("Resident Booking Frequency", "Resident Username", "Frequency", getBookingDataset(), PlotOrientation.VERTICAL, true, true, false);
 7     JFreeChart visitorPieChart = ChartFactory.createPieChart("Visitor Pass Visiting Frequency", getVisitorDataset(), true, true, false);
 8
 9     ChartPanel checkPointPanel = new ChartPanel(checkPointRecordChart);
10     ChartPanel executivePanel = new ChartPanel(executivePieChart);
11     ChartPanel bookingPanel = new ChartPanel(bookingChart);
12     ChartPanel complainPanel = new ChartPanel(complaintPieChart);
13     ChartPanel visitorPanel = new ChartPanel(visitorPieChart);
14     ChartPanel taskPanel = new ChartPanel(taskPieChart);
15
16     JPanel contentPane = new JPanel();
17     contentPane.setLayout(new GridLayout(6, 1, 10, 10));
18     contentPane.add(checkPointPanel);
19     contentPane.add(executivePanel);
20     contentPane.add(bookingPanel);
21     contentPane.add(complainPanel);
22     contentPane.add(visitorPanel);
23     contentPane.add(taskPanel);
24
25     setViewportView(contentPane);
26
27     OutputStream out = Files.newOutputStream(Paths.get("src/ReportPNG/checkPoint.png"));
28     writeAsPNG(checkPointRecordChart, out, 1280, 720);
29     out = Files.newOutputStream(Paths.get("src/ReportPNG/executive.png"));
30     writeAsPNG(executivePieChart, out, 1280, 720);
31     out = Files.newOutputStream(Paths.get("src/ReportPNG/complaint.png"));
32     writeAsPNG(complainPieChart, out, 1280, 720);
33     out = Files.newOutputStream(Paths.get("src/ReportPNG/task.png"));
34     writeAsPNG(taskPieChart, out, 1280, 720);
35     out = Files.newOutputStream(Paths.get("src/ReportPNG/booking.png"));
36     writeAsPNG(bookingChart, out, 1280, 720);
37     out = Files.newOutputStream(Paths.get("src/ReportPNG/visitor pass.png"));
38     writeAsPNG(visitorPieChart, out, 1280, 720);
39 }

```

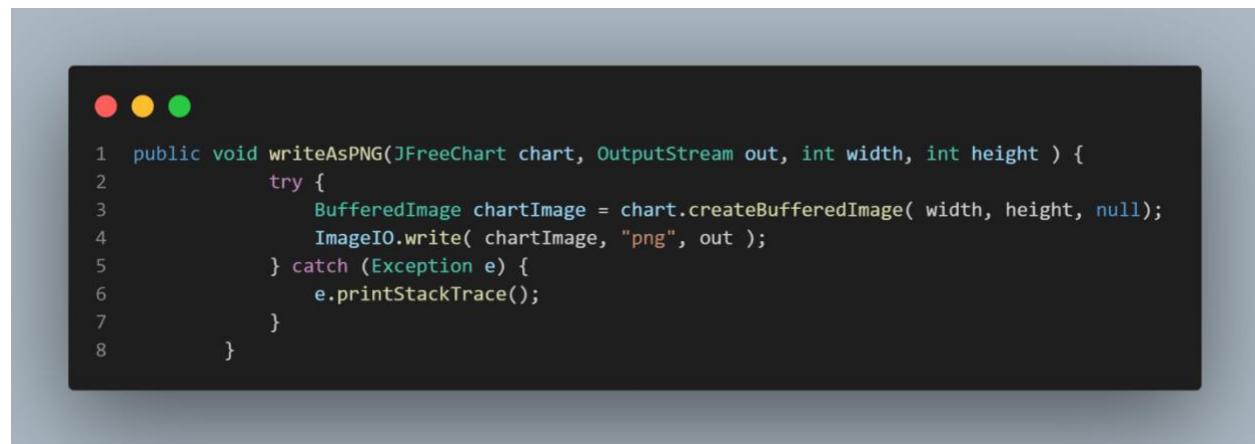
**Security Guard CheckPoint Record**



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



A screenshot of a terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) at the top left. The code shown is:

```
1 public void writeAsPNG(JFreeChart chart, OutputStream out, int width, int height ) {  
2     try {  
3         BufferedImage chartImage = chart.createBufferedImage( width, height, null);  
4         ImageIO.write( chartImage, "png", out );  
5     } catch (Exception e) {  
6         e.printStackTrace();  
7     }  
8 }
```

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

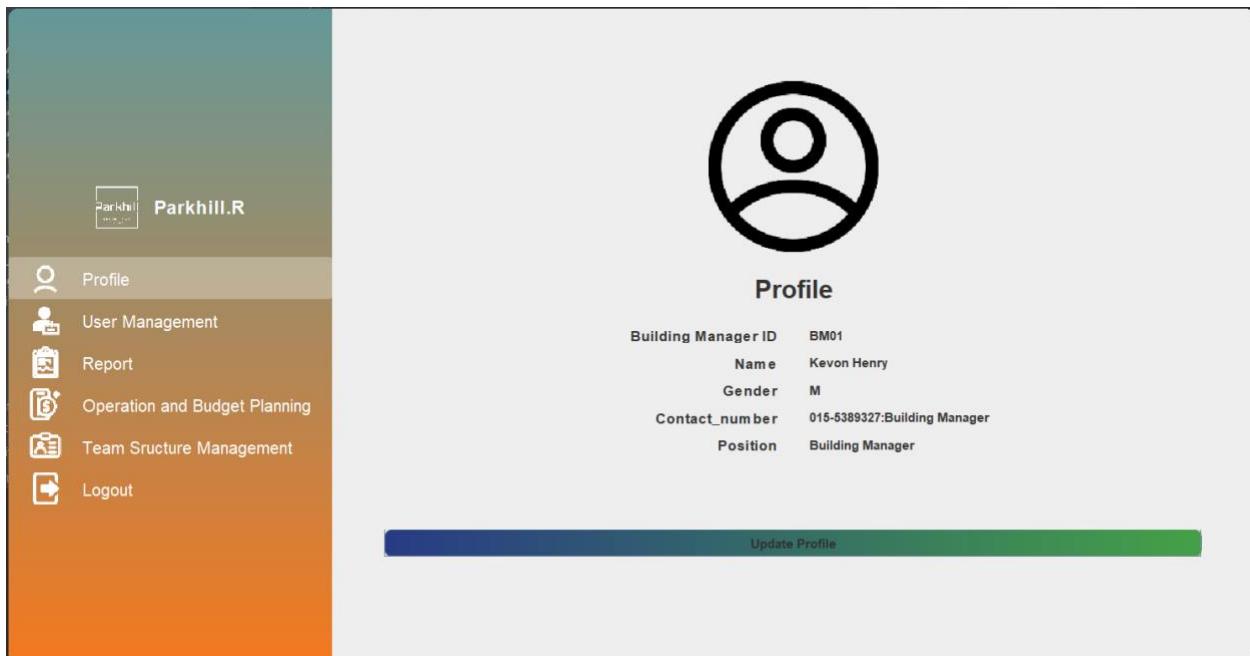
## 9 TP067435 Edmund Chen Siang Zuan

### 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 screenshot shows a user management interface for a building manager. The left sidebar contains navigation links: Profile, User Management, Report, Operation and Budget Planning, Team Structure Management, and Logout. The main area has a search bar and a table showing user executive information. The table includes columns for User Executive ID, Name, Gender, Contact Number, and Position Name. The data in the table is as follows:

User Executive ID	Name	Gender	Contact Number	Position Name
AC02	James Bond	M	014-234789	Account Executive
AC01	Jasper	M	013-3423698	Account Executive
BE01	Michelle	F	017-2374847	Building Executive
BE02	Lucy	F	018-2143726	Building Executive
AD02	Alan	M	013-1267483	Admin Executive
AD01	Joseph	F	011-1278456	Admin Executive

At the bottom, there are four buttons: Add New User (green), Modify User Info (dark blue), Delete User (green), and View User Info (dark blue).

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

User Executive ID	Name	Gender	Contact Number	Position Name
AC02	James Bond	M	014-234789	Account Executive
AC01	Jasper	M	013-3423698	Account Executive
BE01	Position of Executive	F	017-2374847	Building Executive
BE02	Which executive do you want to add?	F	018-2143726	Building Executive
AD02	Account Executive	M	013-1267483	Admin Executive
AD01	Joseph	F	011-1278456	Admin Executive

Buttons at the bottom: Add New User, Modify User Info, Delete User, View User Info.

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.

**User ADDING**

User ExecutiveID	BE13
Name	Nadila
Gender	<input type="radio"/> Male <input checked="" type="radio"/> Female
Contact Number	01 - 23456
Position Name	Building Executive

Add New Executive      Cancel

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.

The screenshot shows a user interface titled "User ADDING". It contains several input fields: "User ExecutiveID" (BE13), "Name" (Nadila), "Gender" (unchecked), "Contact Number" (empty), "Position Name" (Building Executive). A central modal window displays the message: "User adding successful" and "User registration successful. Please ask User to sign up and remember to register new unit if needed." with an "OK" button. At the bottom, there is a navigation bar with "Add New Executive" and "EDMUND CHEN SIANG ZUAN (TP067435@mail.apu.edu.my) is signed in" (highlighted in green), and a "Cancel" button.

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

**User MODIFYING**

User ExecutiveID	AC01
Name	Jasper
Gender	<input checked="" type="radio"/> Male <input type="radio"/> Female
Contact Number	013-3423698
Position Name	Account Executive

[Add New Executive](#)

[Cancel](#)

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

The screenshot shows a user management interface for Parkhill.R. On the left is a sidebar with icons for Profile, User Management, Report, Operation and Budget Planning, Team Structure Management, and Logout. The main area has a search bar at the top and a table titled 'BM01' displaying user data. A modal dialog box titled 'Delete confirmation' is overlaid on the table, asking 'Are you sure to delete this User?' with 'Yes' and 'No' buttons. The table data includes columns for User Executive ID, Name, Gender, Contact Number, and Position Name. The users listed are AC02 (James Bond), AC01 (Jasper), BE01 (Michelle), BE02 (F), BE13 (F), AD02 (M), AD01 (Joseph), and AD01 (F). The 'Delete User' button is highlighted in the bottom navigation bar.

User Executive ID	Name	Gender	Contact Number	Position Name
AC02	James Bond	M	014-234789	Account Executive
AC01	Jasper	M	013-3423698	Account Executive
BE01	Michelle	F	017-2374847	Building Executive
BE02		F	018-2143726	Building Executive
BE13		F	01 -	Building Executive
AD02		M	013-1267483	Admin Executive
AD01	Joseph	F	011-1278456	Admin Executive

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

The screenshot shows a detailed view of an executive's information. The title is 'EXECUTIVE DETAILS' and the executive ID is 'User ExecutiveID BE01'. The details shown are: Name (Michelle), Gender (F), Contact Number (017-2374847), and Position Name (Building Executive). At the bottom, there is a note 'Issued by Parkhill Residence' and a 'Close' button.

User ExecutiveID	BE01
Name	Michelle
Gender	F
Contact Number	017-2374847
Position Name	Building Executive

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

The screenshot shows the "Operation and Budget Planning" section for Building Manager BM01. The interface includes a sidebar with links for Profile, User Management, Report, Operation and Budget Planning (selected), Team Structure Management, and Logout. A search bar at the top right contains the placeholder "Search here...". Below the search bar, the Building Manager ID "BM01" is displayed. The main content area features a table with columns: Operation ID, Building Manager ID, Operation Title, Description, and Budget amount. One row is visible with OP1, BM01, Fix swimming pool, swimming pool renovation, and 100. Below the table are four buttons: "Add Operation Budgetting" (green), "Delete Operation Budgetting" (green), "Pay for operation" (green), "Update Operation Info" (green), "View Operation Info" (green), and "Fund planning..." (purple). At the bottom are three summary cards: "Total fund... 5.5555...", "Fund in ... RM 100", and "Paid Exp... Expenses for... RM 0".

Operation ID	Building Manager ID	Operation Title	Description	Budget amount
OP1	BM01	Fix swimming pool	swimming pool renovation	100

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

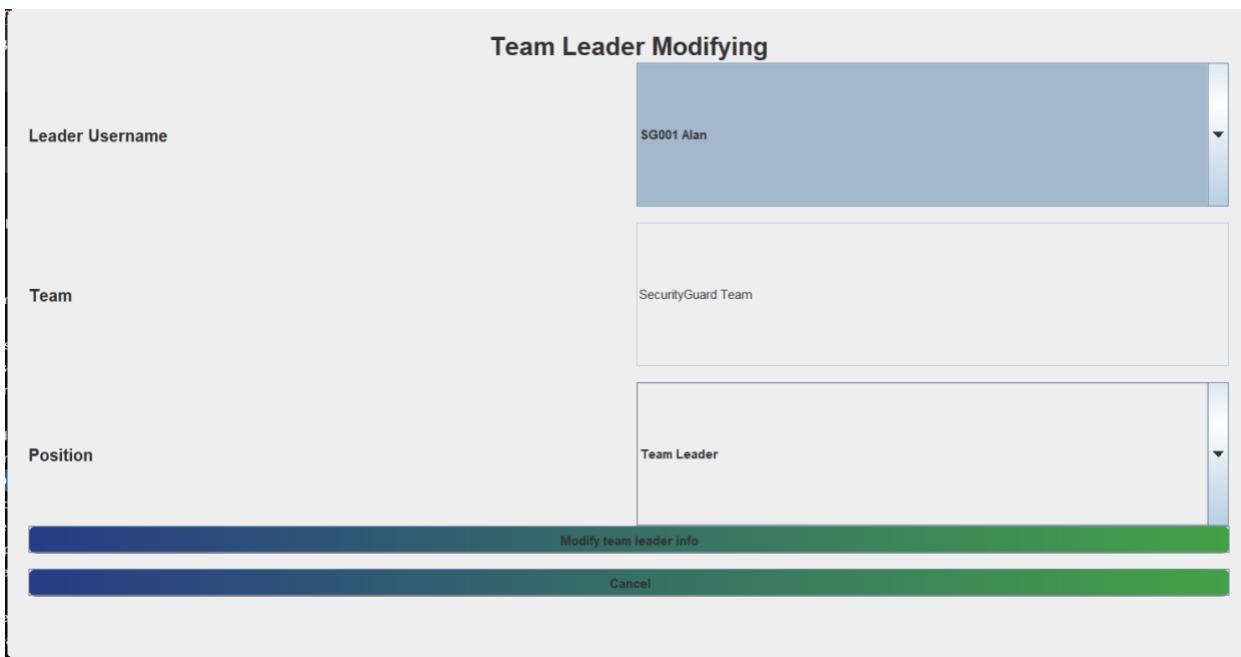
The screenshot shows the "OPERATION ADDING" form. It includes fields for Operation ID (OP2), Building Manager ID (BM01), Operation Title (Fix gym facility), Description (Gym facility renovation), and Budget Amount (RM) (2000). At the bottom are two buttons: "Add Operation Budget" (green) and "Cancel Budgetting" (blue).

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.

The screenshot shows a user interface titled "OPERATION ADDING". The form contains fields for "Operation ID" (OP2), "Building Manager ID" (BM01), "Operation Title" (empty), "Description" (empty), and "Budget Amount (RM)" (20). At the bottom are buttons for "Add Operation Budget" and "Cancel Budgetting". A pop-up window titled "Operation added" with an "OK" button is centered over the form, indicating a successful addition.

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

The screenshot shows the "Operation and Budget Planning" section of the application. A table lists operations with columns: Operation ID, Building Manager ID, Operation Title, Description, and Budget amount. Two rows are visible: OP1 (BM01, Fix swimming pool, swimming pool renovation, 100) and OP2 (BM01, see you, see you, 100). A "Delete confirmation" dialog box is overlaid on the table, asking "Are you sure to delete this Operation?". Below the table are buttons for "Update Operation Info" and "View Operation Info". At the bottom, there are summary boxes for "Total fun...", "Fund in ... RM 200", and "Paid Exp... RM 0".

Operation ID	Building Manager ID	Operation Title	Description	Budget amount
OP1	BM01	Fix swimming pool	swimming pool renovation	100
OP2	BM01	see you	see you	100

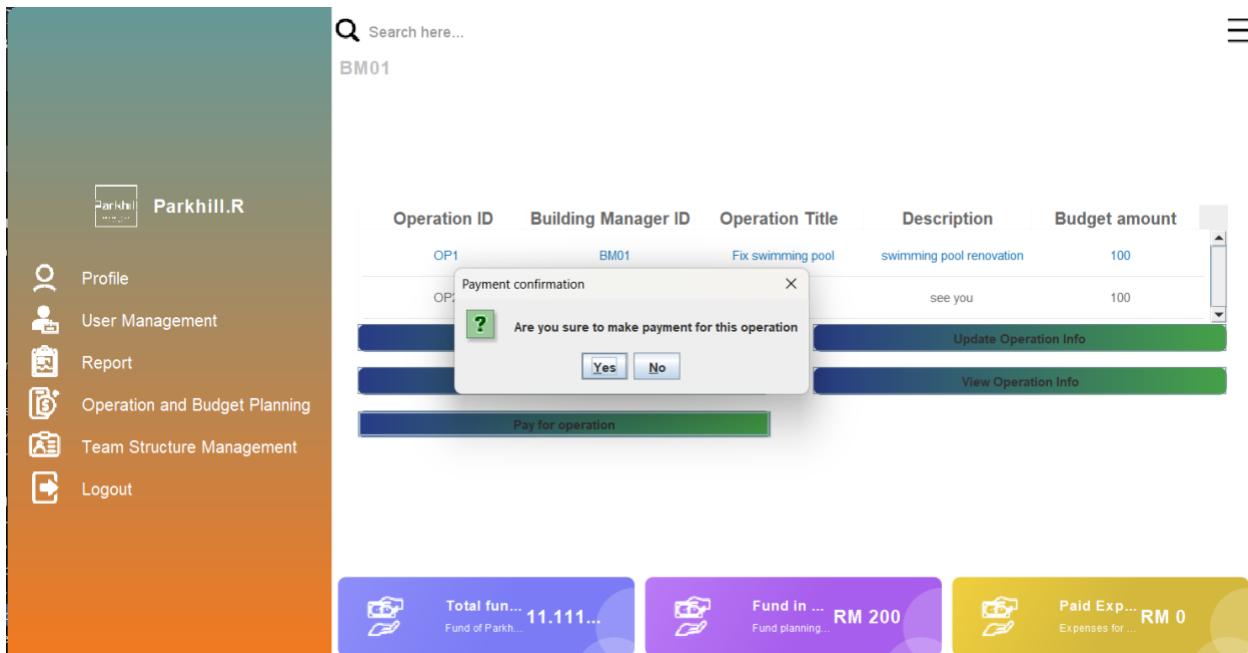
Building Manager can delete a row by clicking on "Delete Operation Budgeting". After clicking, the system will prompt for confirmation of the deletion.

#### 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

The screenshot shows the Parkhill.R application's Team Structure Management page. On the left is a sidebar with icons for Profile, User Management, Report, Operation and Budget Planning, Team Structure Management, and Logout. The main area has a search bar at the top and a header "BM01". Below is a table with columns: Team Leader Username, Team Name, and Position Name. Two rows are visible: one for "AC01" (Account Executive Team, Team Leader) and another for "AC02" (Account Executive Team, Team Vice Leader). At the bottom are four buttons: "Add New Team Leader" (highlighted in green), "Delete Team Leader" (highlighted in green), "Modify Team Leader Info" (highlighted in green), and "View Team Leader Info".

Team Leader Username	Team Name	Position Name
AC01	Account Executive Team	Team Leader
AC02	Account Executive Team	Team Vice Leader

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

The screenshot shows the Parkhill.R application's Team Structure Management page with the "Add New Team Leader" button highlighted. A modal dialog box titled "Team selection" is open, asking "Which team leader do you want to add?". It lists "SecurityGuard Team" as the selected option. There are "OK" and "Cancel" buttons at the bottom of the dialog.

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.

**Team Leader ADDING**

Leader Username	SG001 Alan
	SG001 Alan SG002 Jojo
Team	SecurityGuard Team
Position	Team Leader

[Add New Team Leader](#)

[Cancel](#)

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.

**Team Leader ADDING**

Leader Username	SG001 Alan
Team	<p style="text-align: center;">Team Leader adding success</p> <div style="border: 1px solid #ccc; padding: 10px; width: fit-content; margin: auto;"><p><i>Team Leader added</i></p><p><a href="#">OK</a></p></div>
Position	Team Leader

[Add New Team Leader](#)

[Cancel](#)

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

**Team Leader Modifying**

Leader Username	AC01 Jasper
Team	Account Executive Team
Position	Team Leader

[Modify team leader info](#)  
[Cancel](#)

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

Parkhill.R

Search here...

BM01

Team Leader Username	Team Name	Position Name
AC01	Account Executive Team	Team Leader
	Executive Team	Team Vice Leader
	Board Team	Team Leader

**Delete confirmation**

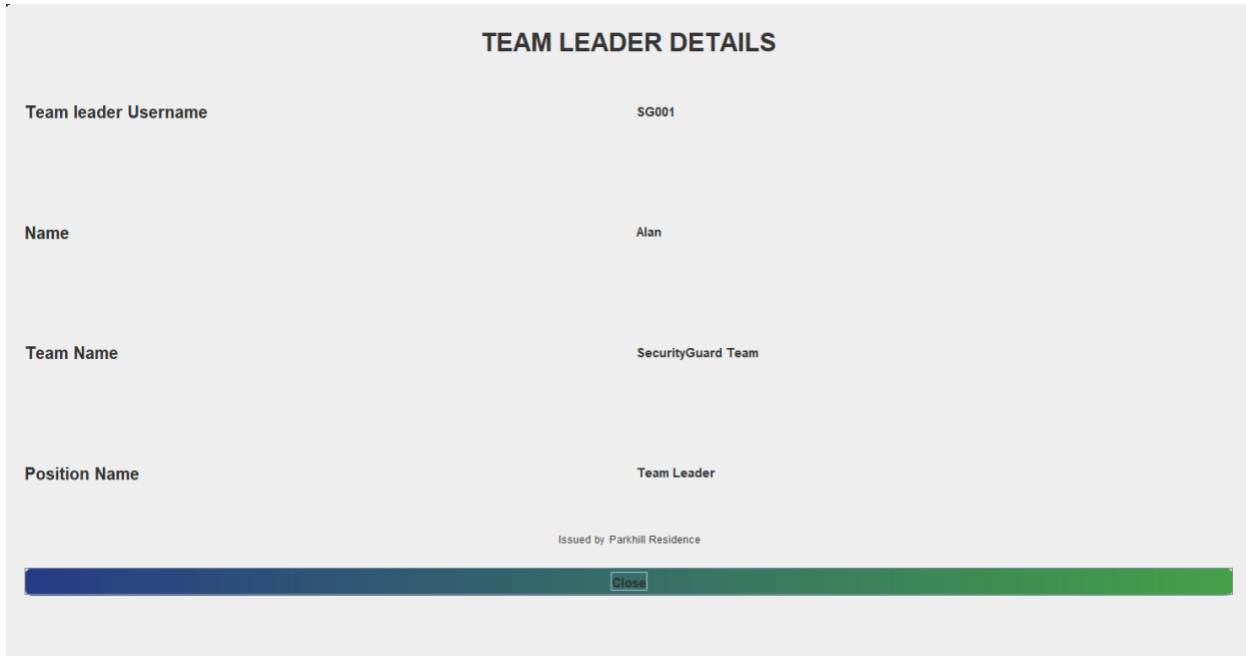
Are you sure to delete this Team leader?

[Yes](#) [No](#)

[Delete Team Leader](#) [View Team Leader Info](#)

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

#### 9.1.6.4 View Team Leader



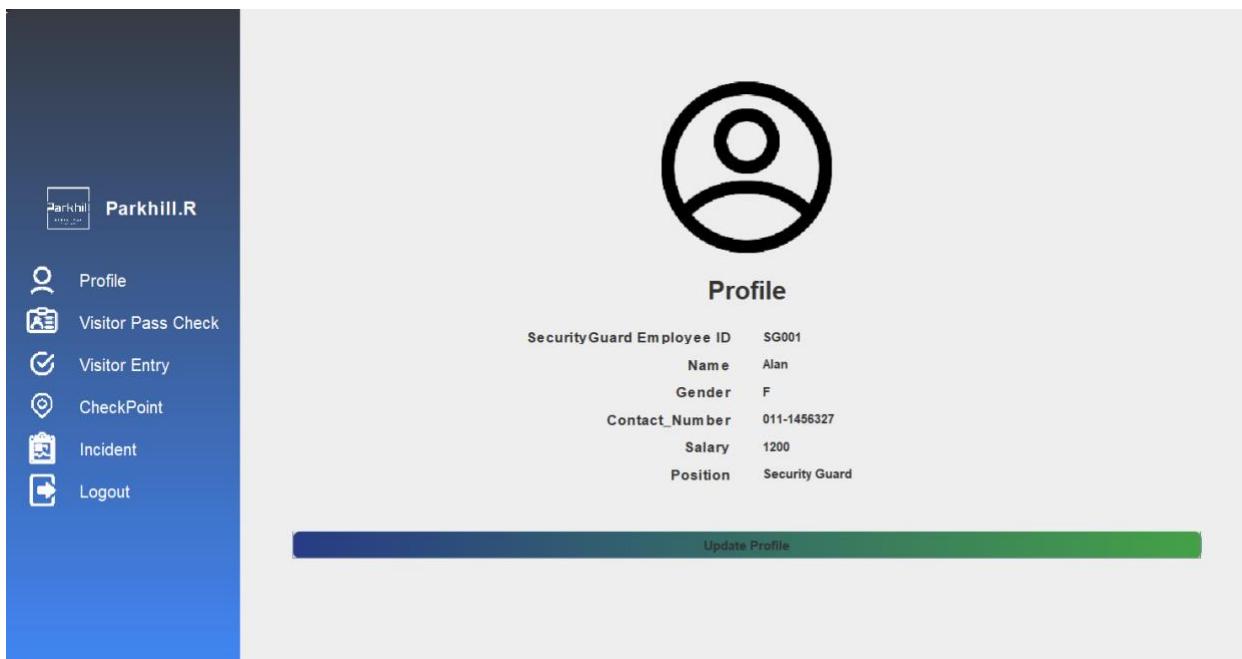
The building manager can click "View Team Leader" to see the relevant information for the selected row.

## 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

The screenshot shows the Parkhill.R mobile application interface. On the left is a vertical navigation bar with icons for Profile, Visitor Pass Check, Visitor Entry, CheckPoint, Incident, and Logout. The main area has a search bar at the top with placeholder text 'Search here...'. Below it is a header 'SG001'. The central part of the screen displays a table of visitor pass information:

Visitor Pas...	Visitor Name	Resident U...	Unit ID	Gender	Contact N...	Date Start	Date End	Status
VI1	Edmund	Mike1001	A401	M	012-3456789	12.26.2022	12.27.2022	APPROVED
VI2	Aikka	Andrew1123	A402	M	011-1212345	12.26.2022	12.27.2022	PENDING
VI3	Ed sheerann	Mike1001	A401	M	012-2346789	12.26.2022	12.27.2022	PENDING

At the bottom of the table is a green button labeled 'View Visitor Pass'.

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

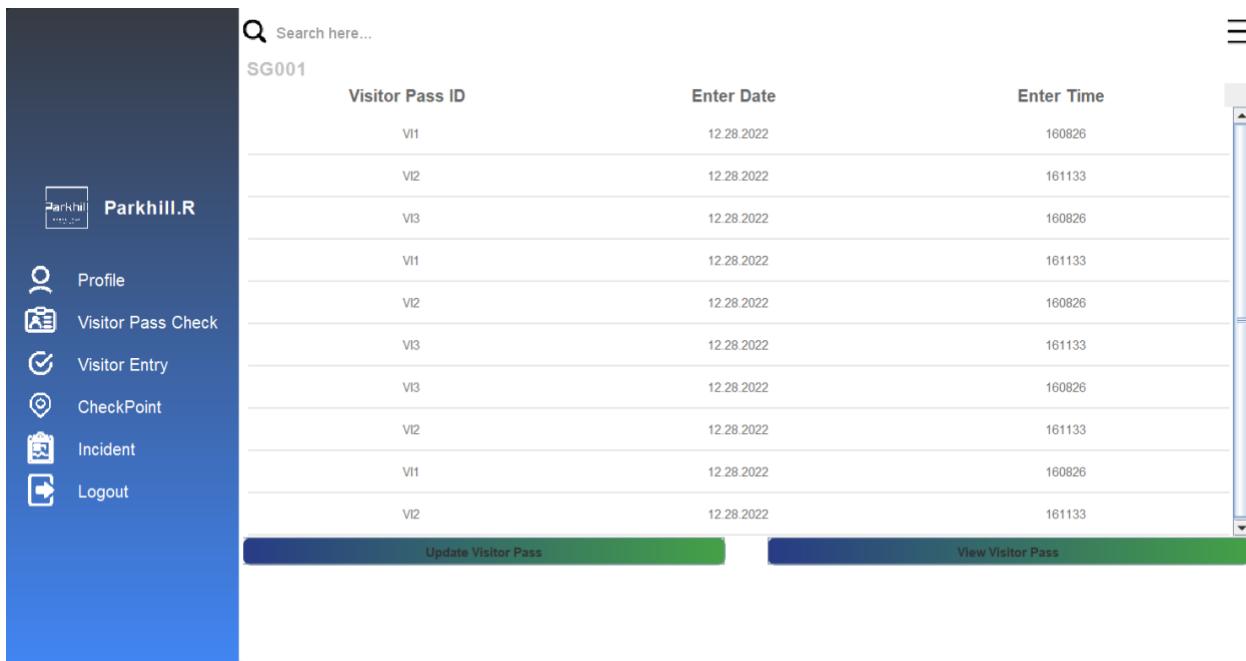
This screenshot shows a detailed view of a visitor pass. The title 'VISITOR PASS' is at the top. The details are listed in pairs:

Visitor Pass ID	VI2
Visitor name	Aikka
Resident Username	Andrew1123
Unit ID	A402
Gender	M
Contact Number	011-1212345
Date Start (MM.dd.yyyy)	12.26.2022
Date End(MM.dd.yyyy)	12.27.2022
Status	Disapproved

Below the details, a small note says 'Issued by Parkhill Residence'. At the bottom is a green button labeled 'Close'.

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



The screenshot shows a software interface for 'Parkhill.R' with a dark blue header and sidebar. The sidebar on the left includes icons for Profile, Visitor Pass Check, Visitor Entry (which is highlighted in yellow), CheckPoint, Incident, and Logout. The main area displays a table titled 'SG001' with columns for 'Visitor Pass ID', 'Enter Date', and 'Enter Time'. The table lists several entries:

Visitor Pass ID	Enter Date	Enter Time
VI1	12.28.2022	160826
VI2	12.28.2022	161133
VI3	12.28.2022	160826
VI1	12.28.2022	161133
VI2	12.28.2022	160826
VI3	12.28.2022	161133
VI3	12.28.2022	160826
VI2	12.28.2022	161133
VI1	12.28.2022	160826
VI2	12.28.2022	161133

At the bottom of the main area are two green buttons: 'Update Visitor Pass' on the left and 'View Visitor Pass' on the right. A search bar at the top says 'Search here...'. On the far right, there is a vertical scroll bar and a three-line menu icon.

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.

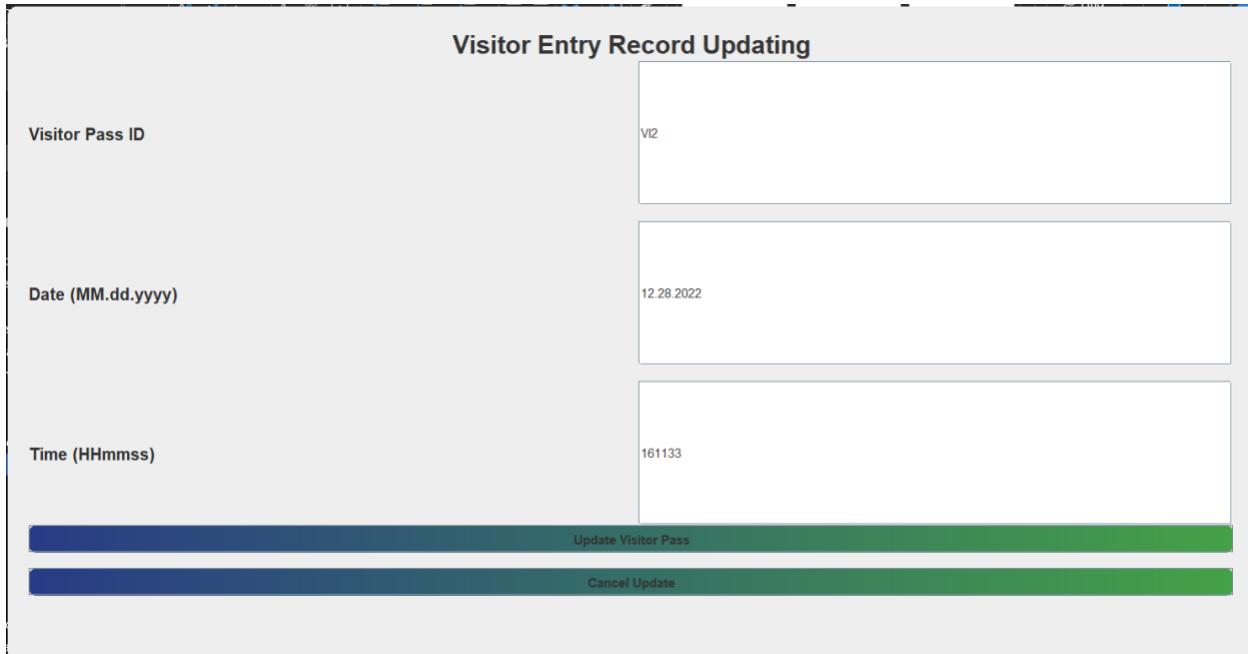
#### 9.2.4.1 Update Visitor Pass

**Visitor Entry Record Updating**

Visitor Pass ID	VI2
Date (MM.dd.yyyy)	12.28.2022
Time (HHmmss)	161133

[Update Visitor Pass](#)

[Cancel Update](#)



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

**VISITOR ENTRY RECORD**

Visitor Pass ID	VI2
Date	12.28.2022
Time	161133

[Close](#)



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.

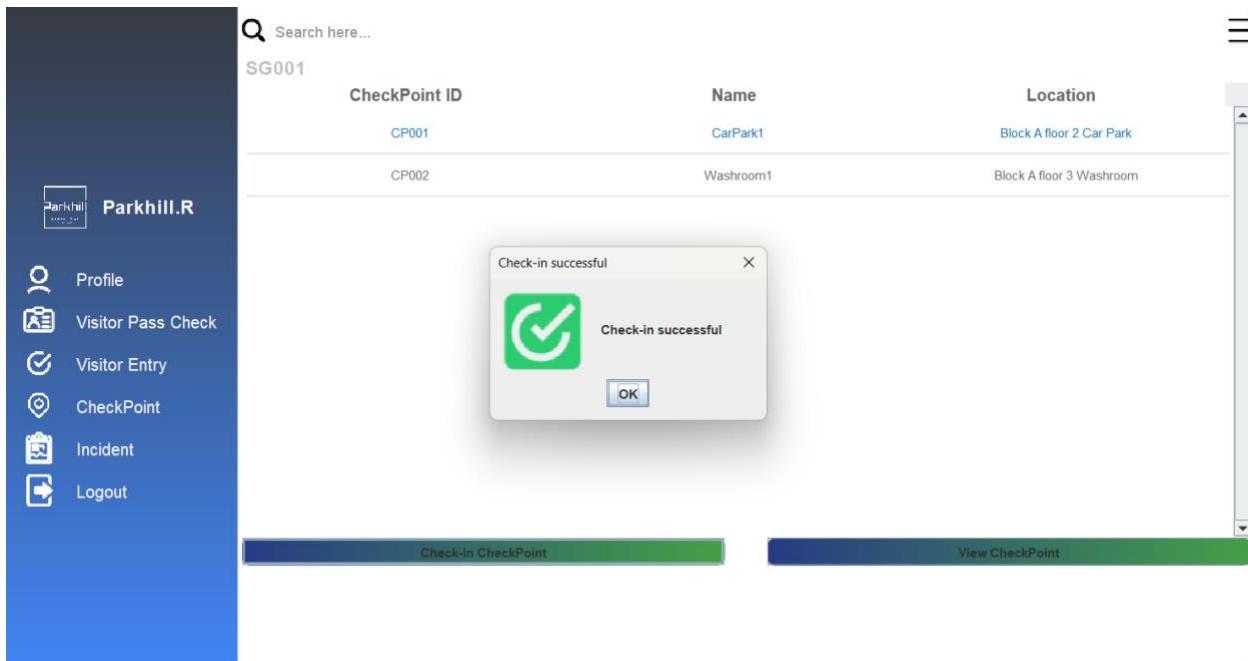
### 9.2.5 Checkpoint

The screenshot shows the Parkhill.R software interface. On the left is a vertical sidebar with a dark blue header containing the Parkhill.R logo and a search bar labeled 'Search here...'. Below the header are several menu items with icons: Profile (person), Visitor Pass Check (document), Visitor Entry (checkmark), CheckPoint (location pin), Incident (camera), and Logout (arrow). The 'CheckPoint' item is highlighted with a light blue background. The main content area has a white header 'SG001'. Below it is a table with three columns: 'CheckPoint ID', 'Name', and 'Location'. Two rows are visible: one for CP001 (CarPark1, Block A floor 2 Car Park) and one for CP002 (Washroom1, Block A floor 3 Washroom). At the bottom of the main area are two green buttons: 'Check-In CheckPoint' on the left and 'View CheckPoint' on the right. The entire interface is framed by a thick black border.

CheckPoint ID	Name	Location
CP001	CarPark1	Block A floor 2 Car Park
CP002	Washroom1	Block A floor 3 Washroom

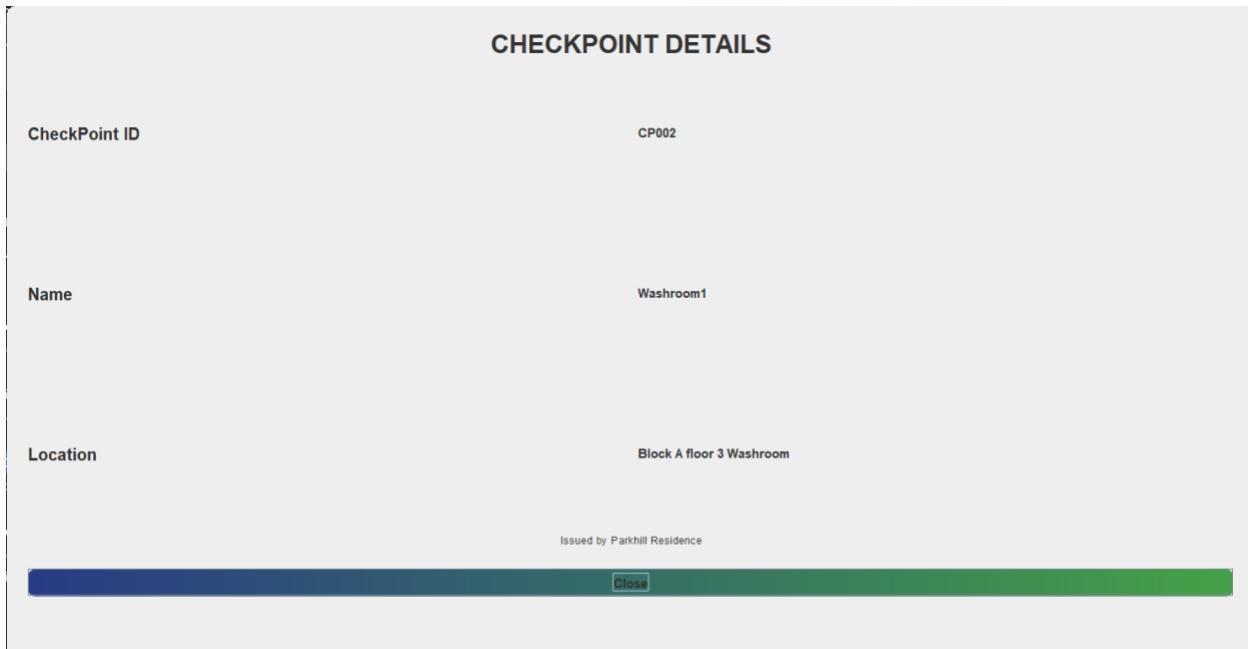
When the user wants to access a checkpoint, they can navigate to the checkpoint section where they will see a list of all checkpoints with their specific information, such as location, ID, and status.

### 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

The screenshot shows the 'Incident' page of the Parkhill.R application. The page has a dark blue header with the logo 'Parkhill.R'. On the left, there is a sidebar with icons for Profile, Visitor Pass Check, Visitor Entry, CheckPoint, Incident, and Logout. At the top right is a search bar with placeholder text 'Search here...'. Below the search bar is a table titled 'SG001' showing ten incident entries. The table has columns for Incident ID, Date, Time, SecurityGuard ID, and Description. The incidents listed are: Inc1 (Date: 01.02.2023, Time: 150318, SG001, Description: 'a robber drink alcohol with me'); Inc2 (Date: 01.22.2923, Time: 142321, SG002, Description: 'malay boy is so handsome'); Inc3 (Date: 01.02.2023, Time: 150318, SG001, Description: 'a robber drink alcohol with me'); Inc4 (Date: 01.22.2923, Time: 142321, SG002, Description: 'malay boy is so handsome'); Inc5 (Date: 01.02.2023, Time: 150318, SG001, Description: 'a robber drink alcohol with me'); Inc6 (Date: 01.22.2923, Time: 142321, SG002, Description: 'malay boy is so handsome'); Inc7 (Date: 01.02.2023, Time: 150318, SG001, Description: 'a robber drink alcohol with me'); Inc8 (Date: 01.22.2923, Time: 142321, SG002, Description: 'malay boy is so handsome'); Inc9 (Date: 01.02.2023, Time: 150318, SG001, Description: 'a robber drink alcohol with me'); Inc10 (Date: 01.22.2923, Time: 142321, SG002, Description: 'malay boy is so handsome'). At the bottom of the table are three buttons: 'Record Incident' (green), 'Update Incident' (blue), and 'Check Incident' (green).

Incident ID	Date	Time	SecurityGuard ID	Description
Inc1	01.02.2023	150318	SG001	a robber drink alcohol with me
Inc2	01.22.2923	142321	SG002	malay boy is so handsome
Inc3	01.02.2023	150318	SG001	a robber drink alcohol with me
Inc4	01.22.2923	142321	SG002	malay boy is so handsome
Inc5	01.02.2023	150318	SG001	a robber drink alcohol with me
Inc6	01.22.2923	142321	SG002	malay boy is so handsome
Inc7	01.02.2023	150318	SG001	a robber drink alcohol with me
Inc8	01.22.2923	142321	SG002	malay boy is so handsome
Inc9	01.02.2023	150318	SG001	a robber drink alcohol with me
Inc10	01.22.2923	142321	SG002	malay boy is so handsome

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.

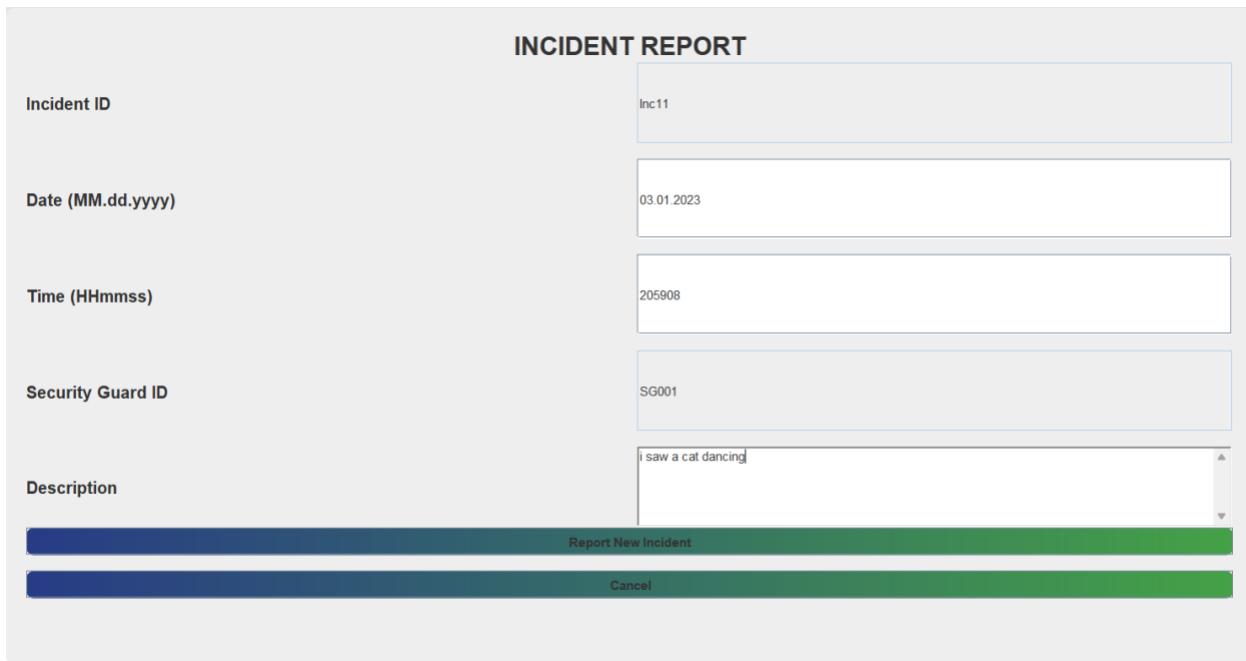
### 9.2.6.1 Record incident

**INCIDENT REPORT**

Incident ID	Inc11
Date (MM.dd.yyyy)	03.01.2023
Time (HHmmss)	205908
Security Guard ID	SG001
Description	i saw a cat dancing

**Report New Incident**

**Cancel**



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.

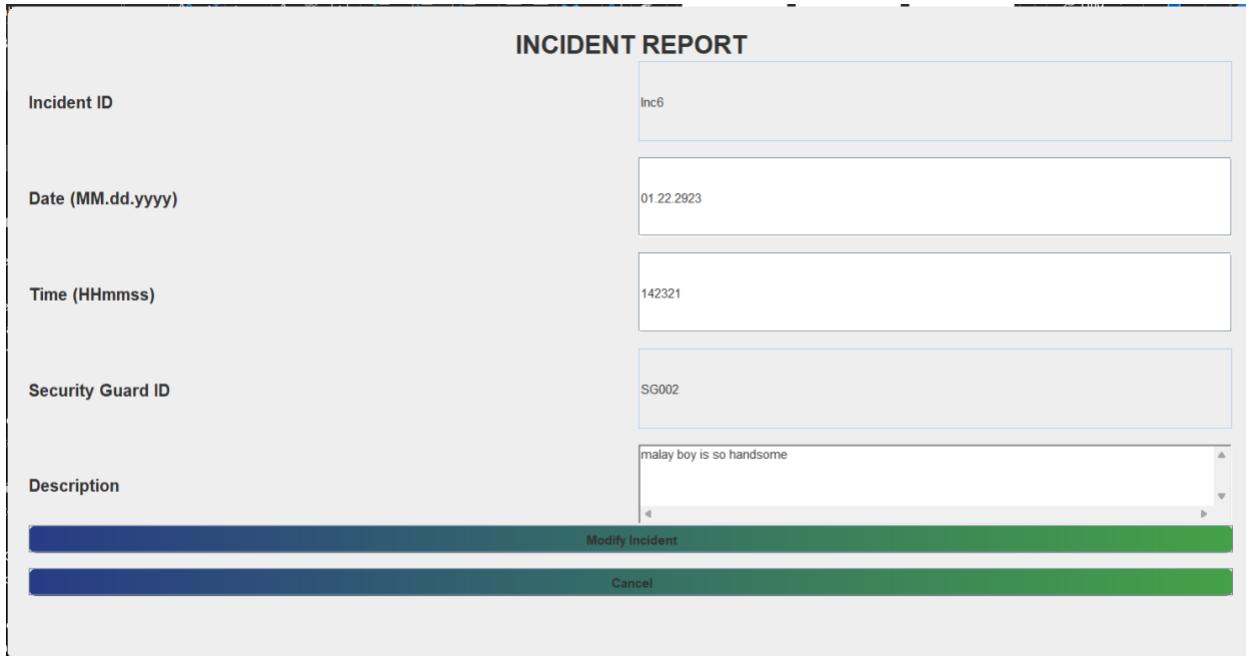
### 9.2.6.2 Update Incident

**INCIDENT REPORT**

Incident ID	Inc6
Date (MM.dd.yyyy)	01.22.2023
Time (HHmmss)	142321
Security Guard ID	SG002
Description	malay boy is so handsome

**Modify Incident**

**Cancel**



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.

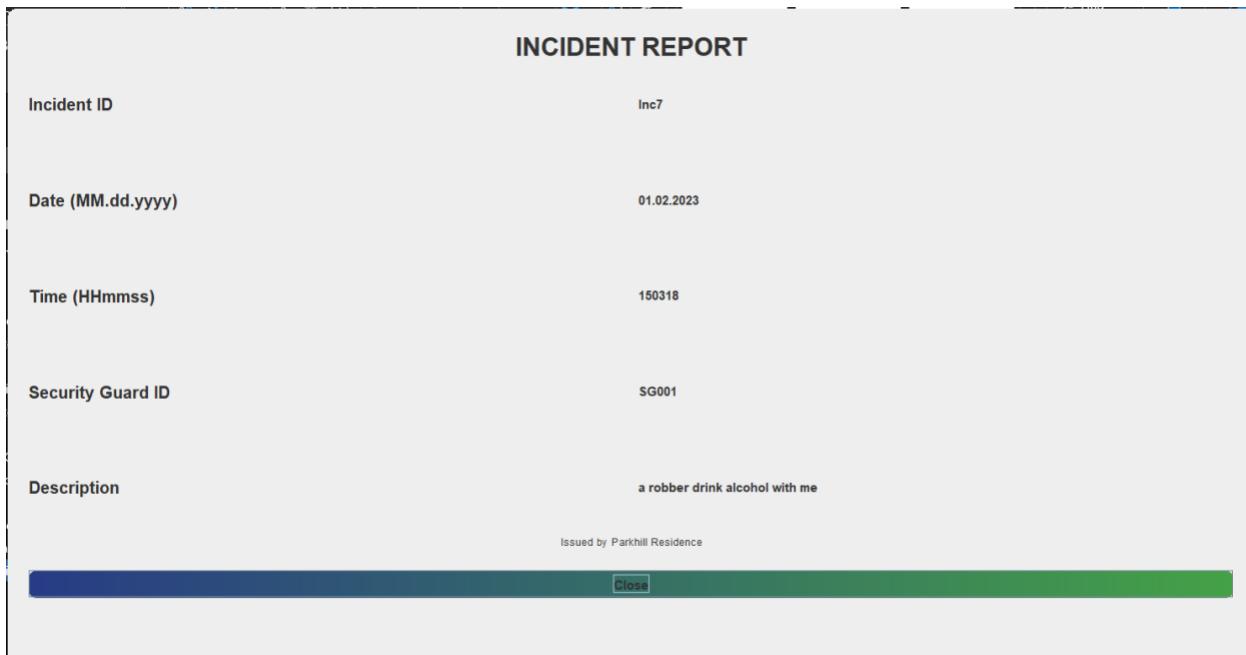
### 9.2.6.3 Check Incident

**INCIDENT REPORT**

Incident ID	Inc7
Date (MM.dd.yyyy)	01.02.2023
Time (HHmmss)	150318
Security Guard ID	SG001
Description	a robber drink alcohol with me

Issued by Parkhill Residence

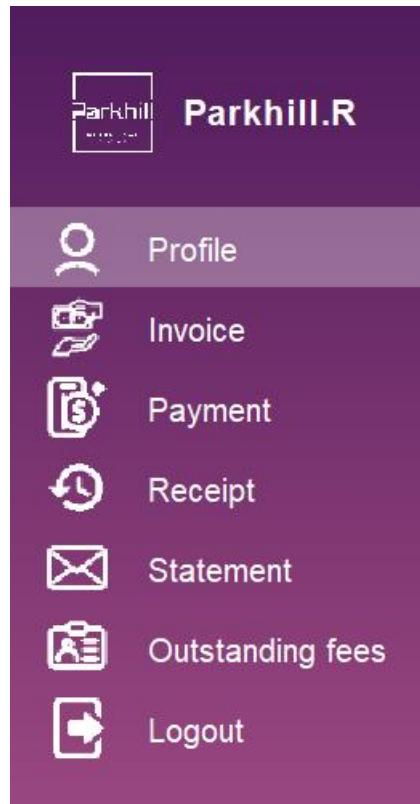
**Close**



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.

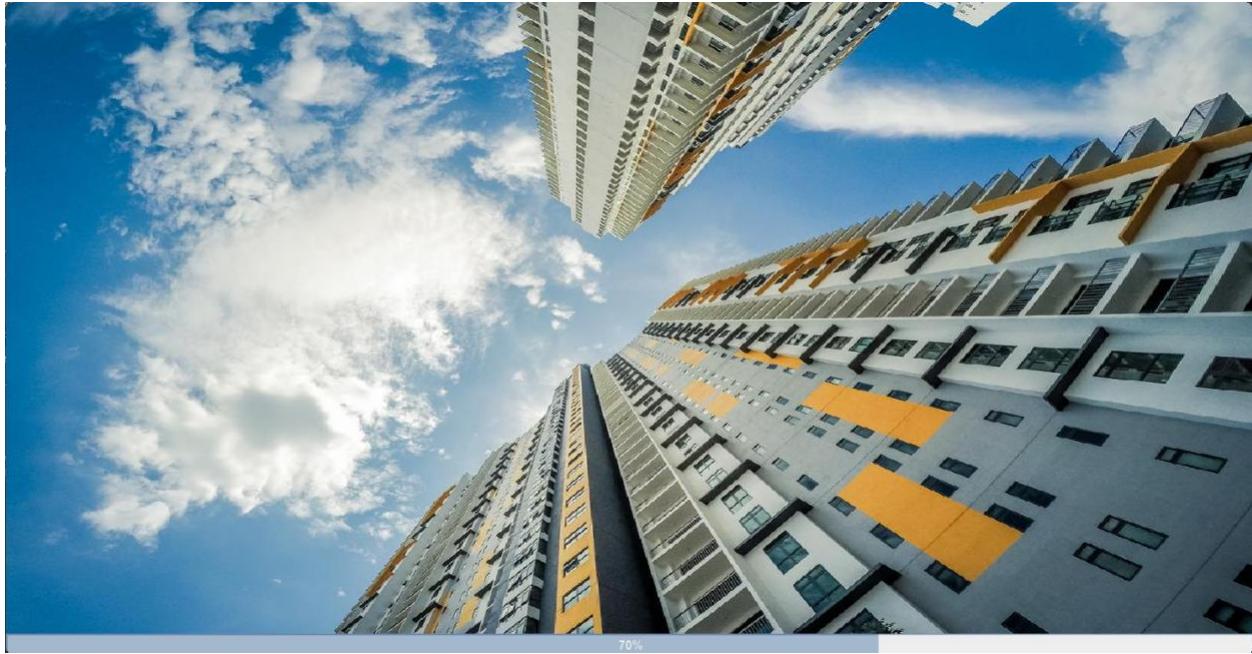
### 9.3 Other features

#### 9.3.1 Menu



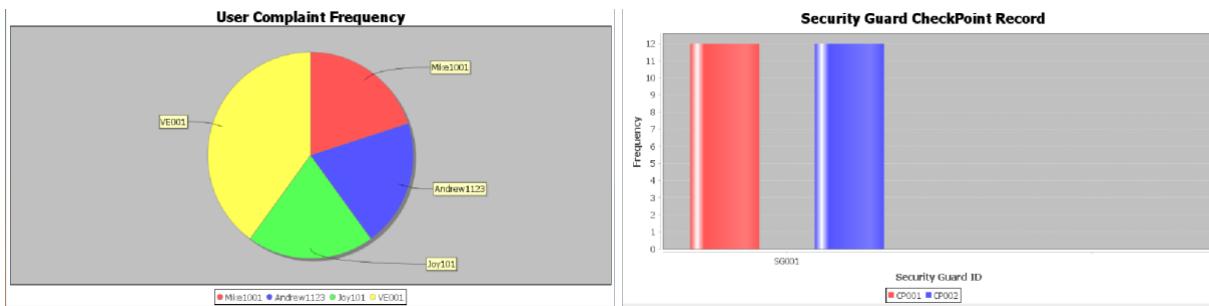
The user interface features a dynamic visual cue to assist users in navigating between sections. When a user selects a different section, the background color of that section will change to a crisp white, providing a clear and intuitive indication of the user's current location within the interface.

#### 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

Facility ID	Name
BH1	Badminton Hall 1
BH2	Badminton Hall 2
GR1	Gymnasium Room 1
GR2	Gymnasium Room 2
FR1	Function Room 1
FR2	Function Room 2

[Add New Facility](#)
[Modify Facility Details](#)

[Delete Facility](#)
[View Facility Details](#)

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

Complaint ID	Resident Username	Description	Status
Com1	Mike1001	A beggar in front of condo	SOLVED
Com2	Andrew1123	Rooftop water leakage	UNSOLVED
Com3	Joy101	Earthquake	UNSOLVED
Com4	VE001	Earthquake	UNSOLVED
Com5	VE001	swimming pool has pee	UNSOLVED

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

```
1 public class Building_Manager_Function{
2     public static class Building_Manager{
3         private String buildingManagerID;
4         private String name;
5         private char gender;
6         private String contact_Number;
7         private String position = "Building Manager";
8         private final File building_Manager_Info_txt = new File("src/Database/Building_Manager_Information.txt");
9         public Building_Manager(){}
10        public Building_Manager(String buildingManagerID, String name, char gender, String contact_Number) {
11            this.buildingManagerID = buildingManagerID;
12            this.name = name;
13            this.gender = gender;
14            this.contact_Number = contact_Number;
15        }
}
```

```
1 public ArrayList<Building_Manager> getArrayList() throws FileNotFoundException {
2     ArrayList<Building_Manager> building_ManagerArrayList = new ArrayList<>();
3     FileReader reader = new FileReader(building_Manager_Info_txt);
4     Scanner scanner = new Scanner(reader);
5     scanner.nextLine();
6     while (scanner.hasNextLine()){
7         String[] data = scanner.nextLine().split(":", 4);
8         building_ManagerArrayList.add(new Building_Manager(data[0], data[1], data[2].charAt(0), data[3]));
9     }
10    return building_ManagerArrayList;
11 }
```

#### 10.1.2 Save all building manager

```
1 public void save_All_Building_Manager(ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList) throws IOException {
2     FileWriter fileWriter = new FileWriter(building_Manager_Info_txt, false);
3     fileWriter.write("Building Manager ID:Name:Gender:contact_number:position\n");
4     for (Building_Manager building_Manager : building_ManagerArrayList){
5         String dataLine = getDataString(building_Manager);
6         fileWriter.write(dataLine);
7     }
8     fileWriter.close();
9 }
```

### 10.1.3 getarraylist

```
1 public ArrayList<Building_Manager> getArrayList() throws FileNotFoundException {
2     ArrayList<Building_Manager> building_ManagerArrayList = new ArrayList<>();
3     FileReader reader = new FileReader(building_Manager_Info_txt);
4     Scanner scanner = new Scanner(reader);
5     scanner.nextLine();
6     while (scanner.hasNextLine()){
7         String[] data = scanner.nextLine().split(":", 4);
8         building_ManagerArrayList.add(new Building_Manager(data[0], data[1], data[2].charAt(0), data[3]));
9     }
10    return building_ManagerArrayList;
11 }
```

### 10.1.4 Search executive information

```
1 public Building_Manager search_Executive_Information(String executiveID) throws FileNotFoundException {
2     ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList = getArrayList();
3     Building_Manager_Function.Building_Manager result = new Building_Manager_Function.Building_Manager();
4     result.setBuildingManagerID("0");
5     for (Building_Manager_Function.Building_Manager accountExecutive : building_ManagerArrayList) {
6         if (accountExecutive.getBuildingManagerID().equals(executiveID)) {
7             result = accountExecutive;
8         }
9     }
10    return result;
11 }
```

### 10.1.5 Check building manager availability

```
1 public boolean check_Building_Manager_Availability(String buildingManagerID) throws FileNotFoundException {
2     boolean result = false;
3     Building_Manager buildingManager = new Building_Manager();
4     ArrayList<Building_Manager_Function.Building_Manager> building_ManagerArrayList = buildingManager.getArrayList();
5     for (Building_Manager buildingManager1 : building_ManagerArrayList) {
6         if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
7             result = true;
8     }
9     return result;
10 }
```

### 10.1.6 Update building manager info

```
1 public void update_Building_Manager_Info(Building_Manager buildingManager, String buildingManagerID) throws IOException {
2     ArrayList<Building_Manager> buildingManagerArrayList = buildingManager.getArrayList();
3     for (Building_Manager buildingManager1 : buildingManagerArrayList){
4         if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
5             buildingManagerArrayList.remove(buildingManager1);
6     }
7     buildingManagerArrayList.add(buildingManager);
8     buildingManager.save_All_Building_Manager(buildingManagerArrayList);
9 }
```

### 10.1.7 Add account executive

```
1 public void add_Account_Executive(Account_Executive_Function.Account_Executive accountExecutive) throws IOException {
2     ArrayList<Account_Executive_Function.Account_Executive> account_executiveArrayList = accountExecutive.getArrayList();
3     if (!accountExecutive.check_Account_Executive_Availability(accountExecutive.getExecutiveID()))
4         account_executiveArrayList.add(accountExecutive);
5     accountExecutive.save_All_Account_Executive(account_executiveArrayList);
6 }
```

### 10.1.8 Delete account executive

```
1 public void delete_Account_Executive(String executiveID) throws IOException {
2     Account_Executive_Function.Account_Executive accountExecutive = new Account_Executive_Function.Account_Executive();
3     ArrayList<Account_Executive_Function.Account_Executive> account_executiveArrayList = accountExecutive.getArrayList();
4     if (accountExecutive.check_Account_Executive_Availability(executiveID)){
5         for (Account_Executive_Function.Account_Executive accountExecutive1 : account_executiveArrayList){
6             if (accountExecutive1.getExecutiveID().equals(executiveID))
7                 account_executiveArrayList.remove(accountExecutive1);
8         }
9     }
10    accountExecutive.save_All_Account_Executive(account_executiveArrayList);
11 }
```

### 10.1.9 Modify account executive

```
1 public void modify_Account_Executive(Account_Executive_Function.Account_Executive accountExecutive, String executiveID) throws IOException {
2     delete_Account_Executive(executiveID);
3     add_Account_Executive(accountExecutive);
4 }
```

### 10.1.10Delete building executive

```
● ● ●
1 public void delete_Building_Executive(String executiveID) throws IOException {
2     Building_Executive_Function.Building_Executive buildingExecutive = new Building_Executive_Function.Building_Executive();
3     ArrayList<Building_Executive_Function.Building_Executive> buildingExecutiveArrayList = buildingExecutive.getArrayList();
4     if (buildingExecutive.check_Building_Executive_Availability(executiveID)){
5         for (Building_Executive_Function.Building_Executive buildingExecutive1 : buildingExecutiveArrayList){
6             if (buildingExecutive1.getExecutiveID().equals(executiveID))
7                 buildingExecutiveArrayList.remove(buildingExecutive1);
8         }
9     }
10    buildingExecutive.save_All_Building_Executive(buildingExecutiveArrayList);
11 }
```

### 10.1.11Modify building executive

```
● ● ●
1 public void modify_Building_Executive(Building_Executive_Function.Building_Executive buildingExecutive, String executiveID) throws IOException {
2     delete_Account_Executive(executiveID);
3     add_Building_Executive(buildingExecutive);
4 }
```

### 10.1.12Add\_admin\_executive

```
● ● ●
1 public void add_Admin_Executive(Admin_Executive_Function.Admin_Executive adminExecutive) throws IOException {
2     ArrayList<Admin_Executive_Function.Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
3     if (!(adminExecutive.check_Admin_Executive_Availability(adminExecutive.getExecutiveID())))
4         adminExecutiveArrayList.add(adminExecutive);
5     adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList);
6 }
```

### 10.1.13Delete admin executive

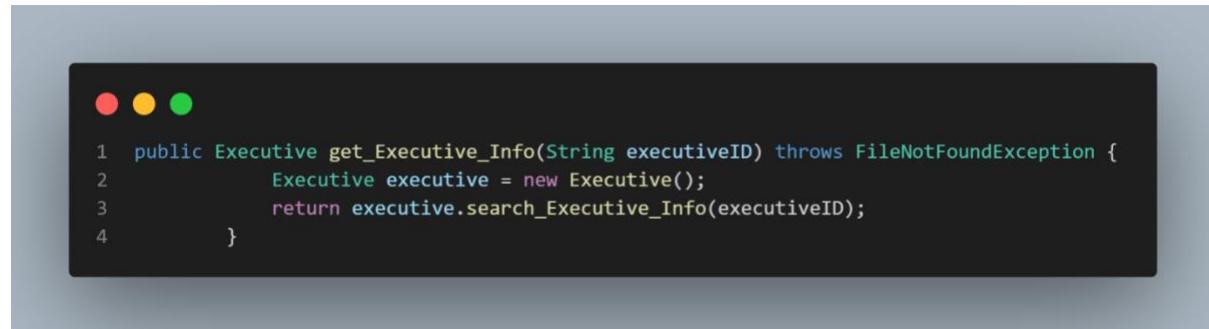
```
● ● ●
1 public void delete_Admin_Executive(String executiveID) throws IOException {
2     Admin_Executive_Function.Admin_Executive adminExecutive = new Admin_Executive_Function.Admin_Executive();
3     ArrayList<Admin_Executive_Function.Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
4     if (adminExecutive.check_Admin_Executive_Availability(executiveID)){
5         for (Admin_Executive_Function.Admin_Executive adminExecutive1 : adminExecutiveArrayList){
6             if (adminExecutive1.getExecutiveID().equals(executiveID))
7                 adminExecutiveArrayList.remove(adminExecutive1);
8         }
9     }
10    adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList);
11 }
```

### 10.1.14modify\_Admin\_Executive



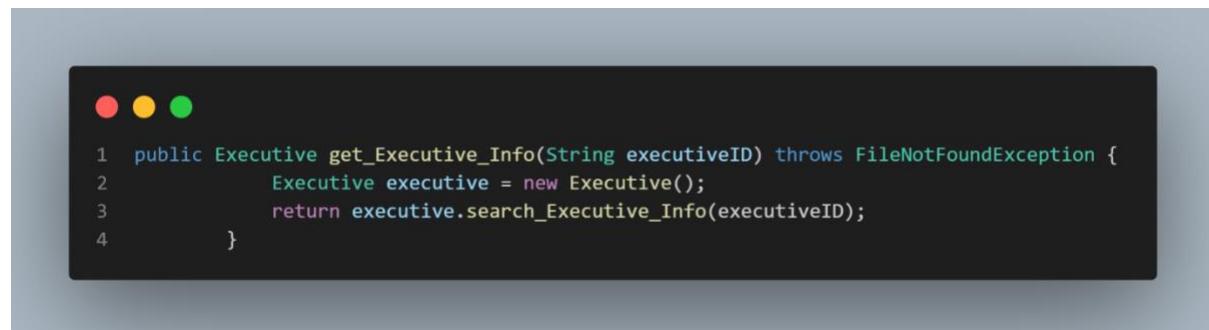
```
1 public void modify_Admin_Employee/Admin_Employee_Function.Admin_Employee adminExecutive, String executiveID) throws IOException {
2     delete_Admin_Employee(executiveID);
3     add_Admin_Employee(adminExecutive);
4 }
5 }
```

### 10.1.15get\_Employee\_Info



```
1 public Executive get_Employee_Info(String executiveID) throws FileNotFoundException {
2     Executive executive = new Executive();
3     return executive.search_Employee_Info(executiveID);
4 }
```

### 10.1.16get\_Building\_Manager\_Info



```
1 public Executive get_Employee_Info(String executiveID) throws FileNotFoundException {
2     Executive executive = new Executive();
3     return executive.search_Employee_Info(executiveID);
4 }
```

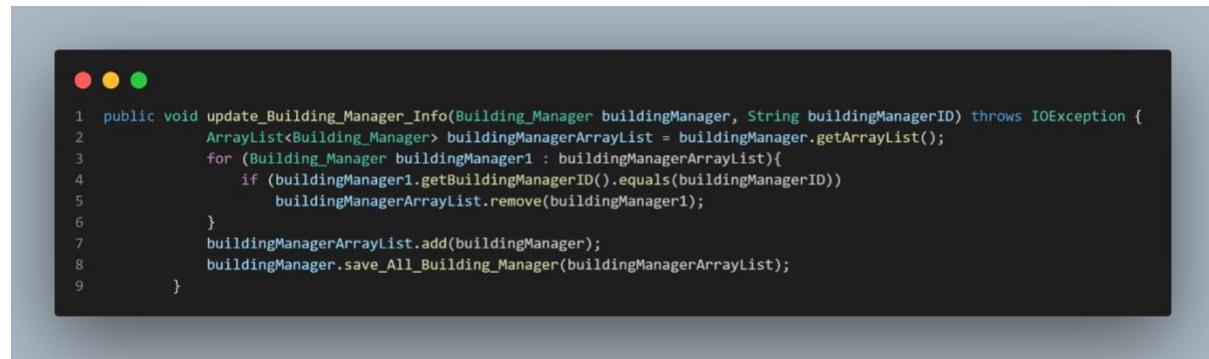
## 10.2 Account Executive

### 10.2.1 Account\_Executive extends Executive



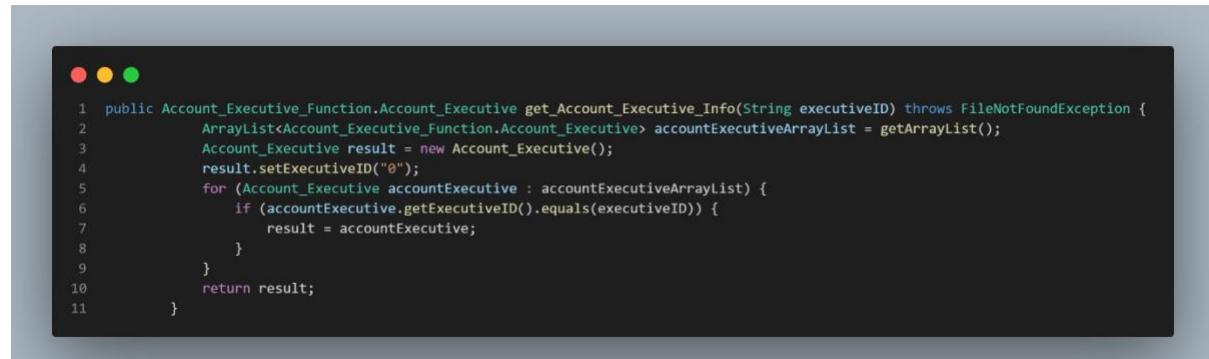
```
1 public static class Account_Executive extends Executive {
2     private String position = "Account Executive";
3     private File account_Executive_Info_txt = new File("src/Database/Account_Executive_Information.txt");
4     public Account_Executive(){}
5     public Account_Executive(String account_Executive_ID, String name, char gender, String contact_Number){
6         super();
7         super.set_Info(account_Executive_ID, name, gender, contact_Number, position);
8     }
}
```

### 10.2.2 update\_Account\_Executive\_Info



```
1 public void update_Building_Manager_Info(Building_Manager buildingManager, String buildingManagerID) throws IOException {
2     ArrayList<Building_Manager> buildingManagerArrayList = buildingManager.getArrayList();
3     for (Building_Manager buildingManager1 : buildingManagerArrayList){
4         if (buildingManager1.getBuildingManagerID().equals(buildingManagerID))
5             buildingManagerArrayList.remove(buildingManager1);
6     }
7     buildingManagerArrayList.add(buildingManager);
8     buildingManager.save_All_Building_Manager(buildingManagerArrayList);
9 }
```

### 10.2.3 get\_Account\_Executive\_Info



```
1 public Account_Executive_Function.Account_Executive get_Account_Executive_Info(String executiveID) throws FileNotFoundException {
2     ArrayList<Account_Executive_Function.Account_Executive> accountExecutiveArrayList = getArrayList();
3     Account_Executive result = new Account_Executive();
4     result.setExecutiveID("");
5     for (Account_Executive accountExecutive : accountExecutiveArrayList) {
6         if (accountExecutive.getExecutiveID().equals(executiveID)) {
7             result = accountExecutive;
8         }
9     }
10    return result;
11 }
```

#### 10.2.4 issue\_All\_Unit\_Invoice

```
● ● ●  
1 public void issue_All_Unit_Invoice(String issuerID, int amount, LocalDate dueDate, String paymentTypes, String description) throws IOException {  
2     ArrayList<Unit> unitArrayList = new Unit().getArrayList();  
3     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();  
4     String status = "unpaid";  
5     for (Unit unit : unitArrayList){  
6         Invoice invoice = new Invoice(null, issuerID, unit.getUnitID(), amount, dueDate, paymentTypes, description, status);  
7         invoice.setInvoiceID(invoice.get_Auto_InvoiceID());  
8         invoiceArrayList.add(invoice);  
9     }  
10    new Invoice().save_All_Invoice(invoiceArrayList);  
11 }
```

#### 10.2.5 get\_All\_pending\_Payment

```
● ● ●  
1 public ArrayList<Payment> get_All_pending_Payment() throws FileNotFoundException {  
2     Payment payment = new Payment();  
3     ArrayList<Payment> paymentArrayList = payment.getArrayList();  
4     ArrayList<Payment> paymentArrayList1 = new ArrayList<>();  
5     for (Payment payment1 : paymentArrayList){  
6         if (payment1.getIssuerID().equals(""))  
7             paymentArrayList1.add(payment1);  
8     }  
9     return paymentArrayList1;  
10 }
```

### 10.2.6 get\_All\_Unit\_List

```
1 public ArrayList<String> get_All_Unit_List() throws FileNotFoundException {
2     ArrayList<Resident> residentArrayList = new Resident().getArrayList();
3     ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
4     ArrayList<String> all_Unit = new ArrayList<>();
5     for (Resident resident : residentArrayList){
6         all_Unit.add(resident.getUnitID());
7     }
8     for (Vendor vendor : vendorArrayList){
9         all_Unit.add(vendor.getVendor_Unit());
10    }
11    for (int i = 0;i < all_Unit.size();i++){
12        for (int j = 0;j < all_Unit.size();j++){
13            if (all_Unit.get(i).equals(all_Unit.get(j))){
14                all_Unit.remove(j);
15            }
16        }
17    }
18 }
```

### 10.2.7 issue\_Unit\_Invoice

```
1 public void issue_Unit_Invoice(Invoice invoice) throws IOException {
2     ArrayList<Invoice> invoiceArrayList = invoice.getArrayList();
3     invoiceArrayList.add(invoice);
4     invoice.save_All_Invoice(invoiceArrayList);
5 }
```

### 10.2.8 issue\_Receipt

```
1 public void issue_Receipt(String paymentID, String issuerID) throws IOException {
2     Payment payment = new Payment();
3     ArrayList<Payment> paymentArrayList = payment.getArrayList();
4     for (Payment payment1 : paymentArrayList){
5         if (payment1.getPaymentID().equals(paymentID) && check_Account_Executive_Availability(issuerID)){
6             payment1.setIssuerID(issuerID);
7             payment1.setIssuedDate(LocalDate.now());
8         }
9     }
10    new Payment().save_All_Payment(paymentArrayList);
11 }
```

### 10.2.9 issue\_Statement

```
1 public void issue_Statement(Statement statement) throws IOException {
2     ArrayList<Statement> statementArrayList = new Statement().getArrayList();
3     statementArrayList.add(statement);
4     new Statement().save_All_Statement(statementArrayList);
5 }
```

### 10.2.10 get\_Unit\_All\_Invoice

```
1 public ArrayList<Invoice> get_Unit_All_Invoice(String unitID) throws FileNotFoundException {
2     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
3     for (Invoice invoice : invoiceArrayList){
4         if (!(invoice.getUnitID().equals(unitID)))
5             invoiceArrayList.remove(invoice);
6     }
7     return invoiceArrayList;
8 }
```

### 10.2.11get\_All\_Unpaid\_Invoice

```
1 public ArrayList<Invoice> get_All_Unpaid_Invoice() throws FileNotFoundException {
2     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
3     ArrayList<Invoice> invoicesReturn = new ArrayList<>();
4     for (Invoice invoice : invoiceArrayList){
5         if (invoice.getStatus().equals("unpaid"))
6             invoicesReturn.add(invoice);
7     }
8     return invoicesReturn;
9 }
```

### 10.2.12get\_Unit\_Unpaid\_Amount

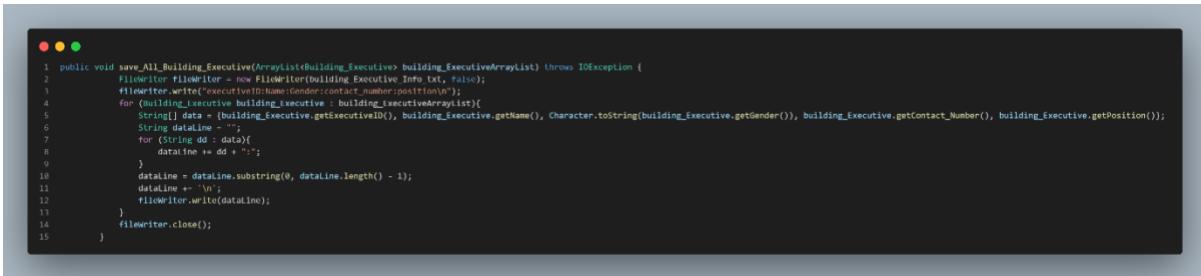
```
1 public int get_Unit_Unpaid_Amount(String unitID) throws FileNotFoundException {
2     int total = 0;
3     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
4     for (Invoice invoice : invoiceArrayList){
5         if (invoice.getUnitID().equals(unitID) && invoice.getStatus().equals("unpaid"))
6             total += invoice.getAmount();
7     }
8     return total;
9 }
```

## 10.3 Building Executive

### 10.3.1 Account\_Executive extends Executive

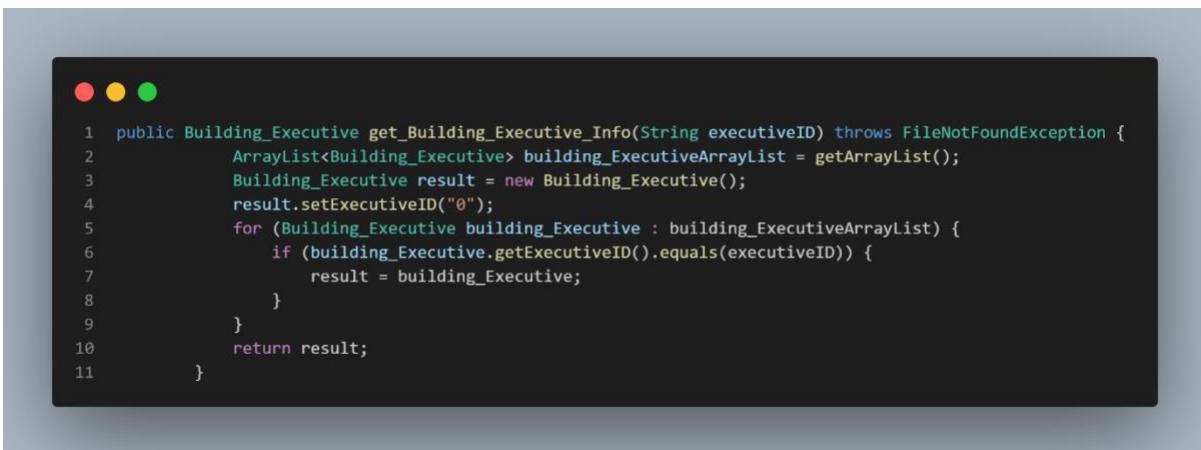
```
1 public static class Building_Executive extends Executive{
2     private String position = "Building Executive";
3     private File building_Executive_Info_txt = new File("src/Database/Building_Executive_Information.txt");
4     public Building_Executive(){}
5     public Building_Executive(String executiveID, String name, char gender, String contact_number){
6         super();
7         super.set_Info(executiveID, name, gender, contact_number, position);
8     }
```

### 10.3.2 save\_All\_Building\_Executive



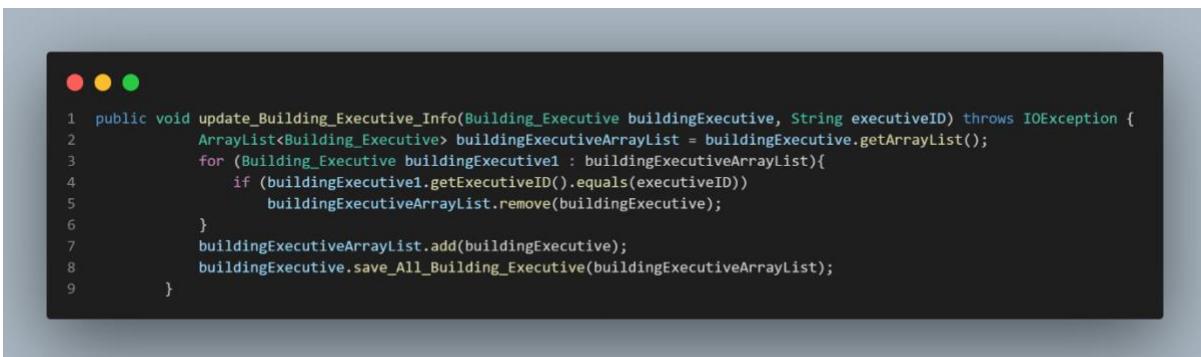
```
1 public void save_All_Building_Executive(ArrayList<Building_Executive> building_ExecutiveArrayList) throws IOException {
2     FileWriter fileWriter = new FileWriter(building_Executive_Info.txt, false);
3     fileWriter.write("executiveID:name:gender:contact_number:position\n");
4     for (Building_Executive building_Executive : building_ExecutiveArrayList){
5         String[] data = {building_Executive.getExecutiveID(), building_Executive.getName(), Character.toString(building_Executive.getGender()), building_Executive.getContact_Number(), building_Executive.getPosition()};
6         String dataLine = "";
7         for (String dd : data){
8             dataLine += dd + ":";
9         }
10        dataLine = dataLine.substring(0, dataLine.length() - 1);
11        dataLine += "\n";
12        fileWriter.write(dataLine);
13    }
14    fileWriter.close();
15 }
```

### 10.3.3 get\_Building\_Executive\_Info



```
1 public Building_Executive get_Building_Executive_Info(String executiveID) throws FileNotFoundException {
2     ArrayList<Building_Executive> building_ExecutiveArrayList = getArrayList();
3     Building_Executive result = new Building_Executive();
4     result.setExecutiveID("0");
5     for (Building_Executive building_Executive : building_ExecutiveArrayList) {
6         if (building_Executive.getExecutiveID().equals(executiveID)) {
7             result = building_Executive;
8         }
9     }
10    return result;
11 }
```

### 10.3.4 update\_Building\_Executive\_Info



```
1 public void update_Building_Executive_Info(Building_Executive buildingExecutive, String executiveID) throws IOException {
2     ArrayList<Building_Executive> buildingExecutiveArrayList = buildingExecutive.getArrayList();
3     for (Building_Executive buildingExecutive1 : buildingExecutiveArrayList){
4         if (buildingExecutive1.getExecutiveID().equals(executiveID))
5             buildingExecutiveArrayList.remove(buildingExecutive);
6     }
7     buildingExecutiveArrayList.add(buildingExecutive);
8     buildingExecutive.save_All_Building_Executive(buildingExecutiveArrayList);
9 }
```

### 10.3.5 add\_Employee\_Task

```
1 public void add_Employee_Task(Employee_Task employeeTask) throws IOException {
2     String status = "undone";
3     employeeTask.setStatus(status);
4     ArrayList<Employee_Task> employeeTaskArrayList = employeeTask.getArrayList();
5     employeeTaskArrayList.add(employeeTask);
6     employeeTask.save_All_Employee_Task(employeeTaskArrayList);
7 }
8 }
```

### 10.3.6 delete\_Employee\_Task

```
1 public void delete_Employee_Task(String taskID) throws IOException {
2     Employee_Task employeeTask = new Employee_Task();
3     ArrayList<Employee_Task> employeeTaskArrayList = employeeTask.getArrayList();
4     for (Employee_Task employeeTask1 : employeeTaskArrayList){
5         if (employeeTask1.getTaskID().equals(taskID))
6             employeeTaskArrayList.remove(employeeTask1);
7     }
8     employeeTask.save_All_Employee_Task(employeeTaskArrayList);
9 }
```

### 10.3.7 update\_Employee\_Task

```
1 public void update_Employee_Task(Employee_Task employeeTask, String taskID) throws IOException {
2     delete_Employee_Task(taskID);
3     add_Employee_Task(employeeTask);
4 }
```

### 10.3.8 view\_All\_Complaint

```
1 public ArrayList<Complaint> view_All_Complaint() throws FileNotFoundException {
2     ArrayList<Complaint> complaintArrayList = new Complaint().getArrayList();
3     return complaintArrayList;
4 }
```

### 10.3.9 add\_Patrolling\_Schedule

```
1 public void add_Patrolling_Schedule(Patrolling patrolling) throws IOException, ClassNotFoundException {
2     SecurityGuard securityGuard = new SecurityGuard();
3     boolean check = securityGuard.check_SecurityGuard_Availability(patrolling.getEmployeeID());
4     if (check){
5         ArrayList<Patrolling> patrollingArrayList = patrolling.getArrayList();
6         patrollingArrayList.add(patrolling);
7         patrolling.save_All_Patrolling(patrollingArrayList);
8     }
9 }
```

### 10.3.10 delete\_Patrolling\_Schedule

```
1 public void modify_Patrolling_Schedule(Patrolling patrolling, String patrolID) throws IOException, ClassNotFoundException {
2     delete_Patrolling_Schedule(patrolID);
3     add_Patrolling_Schedule(patrolling);
4 }
5
```

### 10.3.11add\_New\_CheckPoint

```
1 public void add_New_CheckPoint(CheckPoint checkPoint) throws IOException, ClassNotFoundException {
2     ArrayList<CheckPoint> checkPointArrayList = checkPoint.getArrayList();
3     checkPointArrayList.add(checkPoint);
4     checkPoint.save_All_CheckPoint(checkPointArrayList);
5 }
```

### 10.3.12delete\_CheckPoint

```
1 public void delete_CheckPoint(String checkPointID) throws IOException, ClassNotFoundException {
2     CheckPoint checkPoint = new CheckPoint();
3     ArrayList<CheckPoint> checkPointArrayList = checkPoint.getArrayList();
4     for (CheckPoint checkPoint1 : checkPointArrayList){
5         if (checkPoint1.getCheckPointID().equals(checkPointID))
6             checkPointArrayList.remove(checkPoint1);
7     }
8     checkPoint.save_All_CheckPoint(checkPointArrayList);
9 }
```

### 10.3.13modify\_CheckPoint

```
1 public void modify_CheckPoint(CheckPoint checkPoint, String checkPointID) throws IOException, ClassNotFoundException {
2     delete_CheckPoint(checkPointID);
3     add_New_CheckPoint(checkPoint);
4 }
```

## 10.4 Admin Executive

### 10.4.1 Admin\_Executive extends Executive

```
● ● ●
1 public static class Admin_Executive extends Executive {
2     private String position = "Admin Executive";
3     private File unit_Information_txt = new File("src/Database/Unit_Information.txt");
4     private File admin_Executive_Info_txt = new File("src/Database/Admin_Executive_Information.txt");
5
6     public Admin_Executive(){}
7     public Admin_Executive(String admin_Executive_ID, String name, char gender, String contact_Number){
8         super();
9         super.set_Info(admin_Executive_ID, name, gender, contact_Number, position);
10    }
```

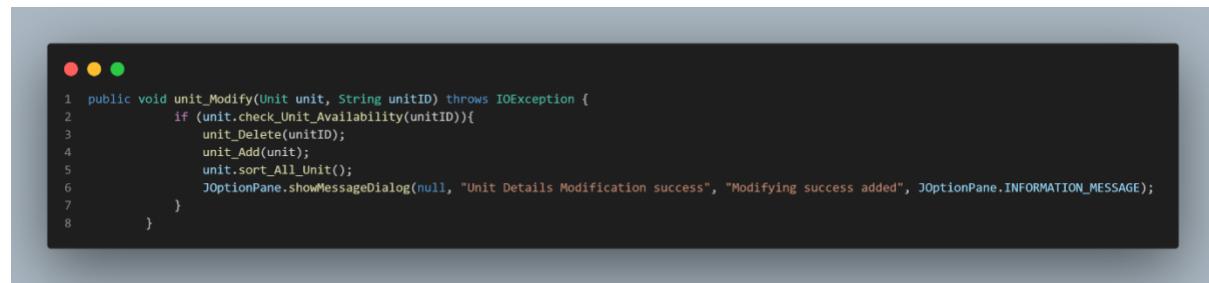
### 10.4.2 update\_Admin\_Executive\_Info

```
● ● ●
1 public void update_Admin_Executive_Info(Admin_Executive adminExecutive, String executiveID) throws IOException {
2     ArrayList<Admin_Executive> adminExecutiveArrayList = adminExecutive.getArrayList();
3     ArrayList<Admin_Executive> adminExecutiveArrayList1 = new ArrayList<>();
4     for (Admin_Executive adminExecutive1 : adminExecutiveArrayList){
5         if (!(adminExecutive1.getExecutiveID().equals(executiveID)))
6             adminExecutiveArrayList1.add(adminExecutive1);
7     }
8     adminExecutiveArrayList1.add(adminExecutive);
9     adminExecutive.save_All_Admin_Executive(adminExecutiveArrayList1);
10 }
```

### 10.4.3 unit\_Add

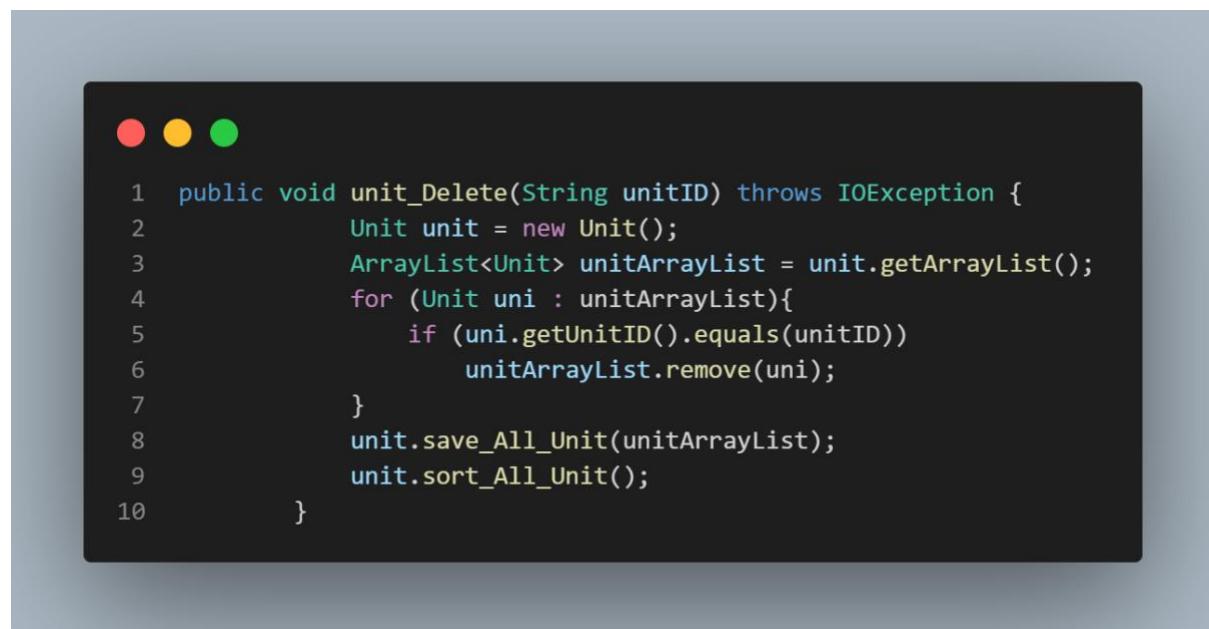
```
● ● ●
1 public void unit_Add(Unit unit) throws IOException {
2     ArrayList<Unit> unitArrayList = unit.getArrayList();
3     unitArrayList.add(unit);
4     unit.save_All_Unit(unitArrayList);
5 }
```

#### 10.4.4 unit\_Modify



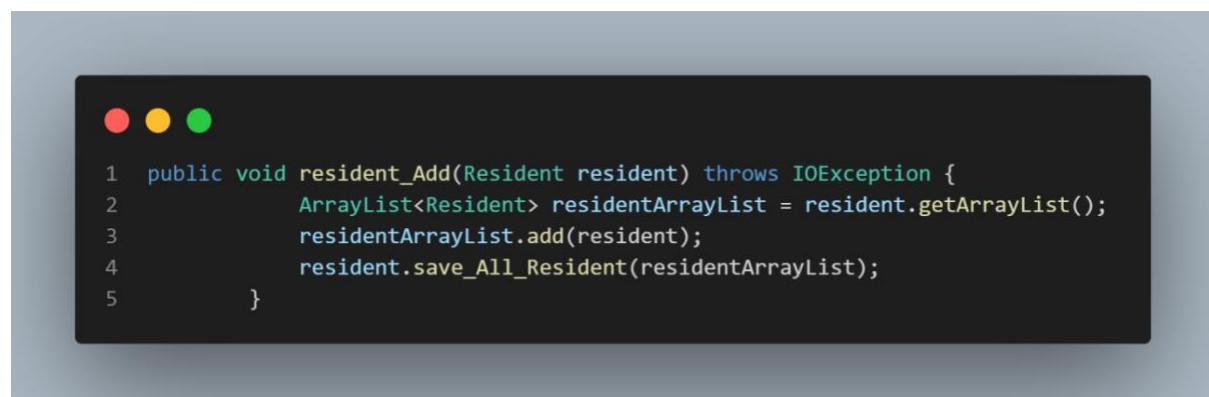
```
1 public void unit_Modify(Unit unit, String unitID) throws IOException {
2     if (unit.check_Unit_Availability(unitID)){
3         unit_Delete(unitID);
4         unit_Add(unit);
5         unit.sort_All_Unit();
6         JOptionPane.showMessageDialog(null, "Unit Details Modification success", "Modifying success added", JOptionPane.INFORMATION_MESSAGE);
7     }
8 }
```

#### 10.4.5 unit\_Delete



```
1 public void unit_Delete(String unitID) throws IOException {
2     Unit unit = new Unit();
3     ArrayList<Unit> unitArrayList = unit.getArrayList();
4     for (Unit uni : unitArrayList){
5         if (uni.getUnitID().equals(unitID))
6             unitArrayList.remove(uni);
7     }
8     unit.save_All_Unit(unitArrayList);
9     unit.sort_All_Unit();
10 }
```

#### 10.4.6 resident\_Add



```
1 public void resident_Add(Resident resident) throws IOException {
2     ArrayList<Resident> residentArrayList = resident.getArrayList();
3     residentArrayList.add(resident);
4     resident.save_All_Resident(residentArrayList);
5 }
```

#### 10.4.7 resident\_Modify



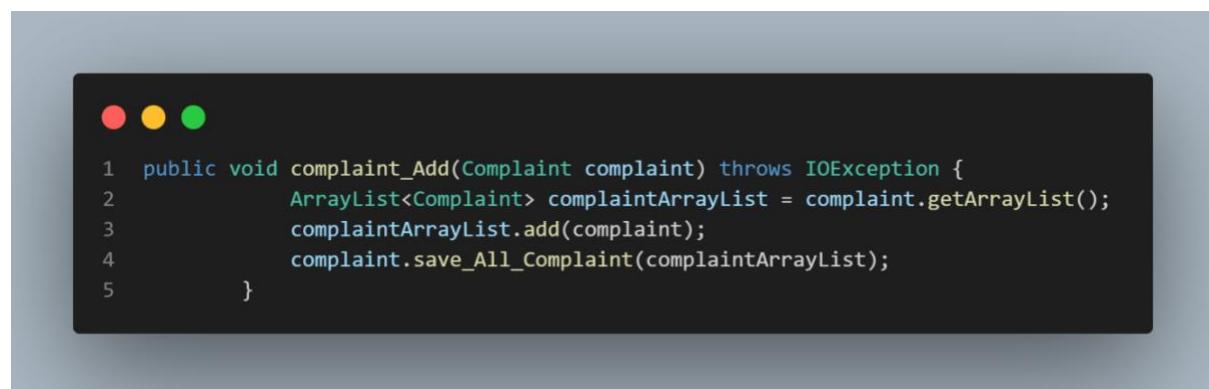
```
1 public void resident_Modify(Resident resident, String resident_Username) throws IOException {
2     String dataLine = resident.getDataString(resident);
3     if (resident.check_Resident_Availability(resident_Username)){
4         resident_Delete(resident_Username);
5         resident_Add(resident);
6         JOptionPane.showMessageDialog(null, "Resident Information modifying success", "Resident information modifying success", JOptionPane.INFORMATION_MESSAGE);
7     }
8 }
```

#### 10.4.8 resident\_Delete



```
1 public void resident_Delete(String resident_Username) throws IOException {
2     Resident resident = new Resident();
3     ArrayList<Resident> residentArrayList = resident.getArrayList();
4     for (Resident re : residentArrayList){
5         if (re.getResident_Username().equals(resident_Username))
6             residentArrayList.remove(re);
7     }
8     resident.save_All_Resident(residentArrayList);
9 }
```

#### 10.4.9 complaint\_Add



```
1 public void complaint_Add(Complaint complaint) throws IOException {
2     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
3     complaintArrayList.add(complaint);
4     complaint.save_All_Complaint(complaintArrayList);
5 }
```

#### 10.4.10 complaint\_Update

```
1 public void complaint_Update(Complaint complaint, String complaintID) throws IOException {
2     if (complaint.check_Complaint_Availability(complaintID)){
3         complaint_Delete(complaintID);
4         complaint_Add(complaint);
5     }
6 }
```

#### 10.4.11 complaint\_Delete

```
1 public void complaint_Delete(String complaintID) throws IOException {
2     Complaint complaint = new Complaint();
3     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
4     for (Complaint com : complaintArrayList){
5         if (com.getComplaintID().equals(complaintID))
6             complaintArrayList.remove(com);
7     }
8     complaint.save_All_Complaint(complaintArrayList);
9 }
```

#### 10.4.12 add\_Employee

```
1 public void add_Employee(Employee employee) throws IOException, ClassNotFoundException {
2     String condition = employee.getEmployeeID().substring(0, 2);
3     if (condition.equals("SG")){
4         SecurityGuard securityGuard = new SecurityGuard();
5         securityGuard.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
6         ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
7         securityGuardArrayList.add(securityGuard);
8         securityGuard.save_All_SecurityGuard(securityGuardArrayList);
9     }else if (condition.equals("CN")){
10         Cleaner cleaner = new Cleaner();
11         cleaner.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
12         ArrayList<Cleaner> cleanerArrayList = cleaner.getArrayList();
13         cleanerArrayList.add(cleaner);
14         cleaner.save_All_Cleaner(cleanerArrayList);
15     } else if (condition.equals("TN")){
16         Technician technician = new Technician();
17         technician.set_Info(employee.getEmployeeID(), employee.getName(), employee.getGender(), employee.getContact_Number(), employee.getSalary(), employee.getPosition_Name());
18         ArrayList<Technician> technicianArrayList = technician.getArrayList();
19         technicianArrayList.add(technician);
20         technician.save_All_Technician(technicianArrayList);
21     }
22 }
```

#### 10.4.13view\_All\_Employee

```
1 public ArrayList<Employee> view_All_Employee() throws IOException, ClassNotFoundException {
2     ArrayList<Employee> employeeArrayList = new ArrayList<>();
3     ArrayList<Technician> technicianArrayList = new Technician().getArrayList();
4     ArrayList<Cleaner> cleanerArrayList = new Cleaner().getArrayList();
5     ArrayList<SecurityGuard> securityGuardArrayList = new SecurityGuard().getArrayList();
6     employeeArrayList.addAll(technicianArrayList);
7     employeeArrayList.addAll(cleanerArrayList);
8     employeeArrayList.addAll(securityGuardArrayList);
9     return employeeArrayList;
10 }
```

#### 10.4.14delete\_Employee

```
1 public void delete_Employee(String employeeID) throws IOException, ClassNotFoundException {
2     String condition = employeeID.substring(0, 2);
3     if (condition.equals("SG")){
4         SecurityGuard securityGuard = new SecurityGuard();
5         ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();
6         for (SecurityGuard securityGuard1 : securityGuardArrayList){
7             if (securityGuard1.getEmployeeID().equals(employeeID))
8                 securityGuardArrayList.remove(securityGuard1);
9         }
10        securityGuard.save_All_SecurityGuard(securityGuardArrayList);
11    }else if (condition.equals("CN")){
12        Cleaner cleaner = new Cleaner();
13        ArrayList<Cleaner> cleanerArrayList = cleaner.getArrayList();
14        for (Cleaner cleaner1 : cleanerArrayList){
15            if (cleaner1.getEmployeeID().equals(employeeID))
16                cleanerArrayList.remove(cleaner1);
17        }
18        cleaner.save_All_Cleaner(cleanerArrayList);
19    } else if (condition.equals("TN")){
20        Technician technician = new Technician();
21        ArrayList<Technician> technicianArrayList = technician.getArrayList();
22        for (Technician technician1 : technicianArrayList){
23            if (technician1.getEmployeeID().equals(employeeID))
24                technicianArrayList.remove(technician1);
25        }
26        technician.save_All_Technician(technicianArrayList);
27    }
28 }
```

#### 10.4.15update\_Employee

```
● ● ●  
1  public void update_Employee(Employee employee, String employeeID) throws IOException, ClassNotFoundException {  
2      delete_Employee(employeeID);  
3      add_Employee(employee);  
4  }
```

#### 10.4.16add\_Facility

```
● ● ●  
1  public void add_Facility(Facility facility) throws IOException, ClassNotFoundException {  
2      ArrayList<Facility> facilityArrayList = facility.getArrayList();  
3      if (!facility.check_Facility_Availability(facility.getFacilityID()))  
4          facilityArrayList.add(facility);  
5      facility.save_All_Facility(facilityArrayList);  
6  }
```

#### 10.4.17delete\_Facility

```
● ● ●  
1  public void delete_Facility(String facilityID) throws IOException, ClassNotFoundException {  
2      Facility facility = new Facility();  
3      ArrayList<Facility> facilityArrayList = facility.getArrayList();  
4      for (Facility facility1 : facilityArrayList){  
5          if (facility1.getFacilityID().equals(facilityID))  
6              facilityArrayList.remove(facility1);  
7      }  
8      facility.save_All_Facility(facilityArrayList);  
9  }
```

#### 10.4.18update\_Facility

```
● ● ●  
1 public void update_Facility(Facility facility, String facilityID) throws IOException, ClassNotFoundException {  
2     delete_Facility(facilityID);  
3     add_Facility(facility);  
4 }
```

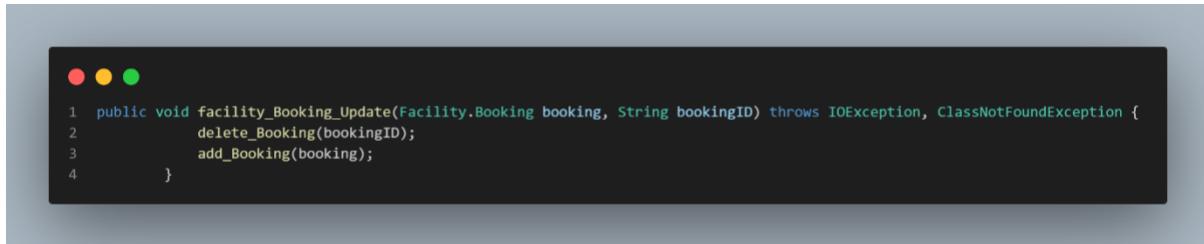
#### 10.4.19add\_Booking

```
● ● ●  
1 public void add_Booking(Facility.Booking booking) throws IOException, ClassNotFoundException {  
2     boolean check = booking.check_TimeSlot_Availability(booking);  
3     if (check) {  
4         ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();  
5         bookingArrayList.add(booking);  
6         booking.save_All_Facility_Booking(bookingArrayList);  
7     }  
8 }
```

#### 10.4.20delete\_Booking

```
● ● ●  
1 public void delete_Booking(String bookingID) throws IOException, ClassNotFoundException {  
2     Facility.Booking booking = new Facility.Booking();  
3     ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();  
4     for (Facility.Booking booking1 : bookingArrayList){  
5         if (booking1.getBookingID().equals(bookingID))  
6             bookingArrayList.remove(booking1);  
7     }  
8     booking.save_All_Facility_Booking(bookingArrayList);  
9 }
```

#### 10.4.21facility\_Booking\_Update



```
1 public void facility_Booking_Update(Facility.Booking booking, String bookingID) throws IOException, ClassNotFoundException {
2     delete_Booking(bookingID);
3     add_Booking(booking);
4 }
```

#### 10.4.22save\_All\_Admin\_Executive



```
1 public void save_All_Admin_Executive(ArrayList<Admin_Executive> adminExecutiveArrayList) throws IOException {
2     FileWriter fileWriter = new FileWriter(admin_Executive_Info.txt, false);
3     fileWriter.write("executiveID:Name:Gender:contact_number:position\n");
4     for (Admin_Executive adminExecutive : adminExecutiveArrayList){
5         String[] data = {adminExecutive.getExecutiveID(), adminExecutive.getName(), Character.toString(adminExecutive.getGender()), adminExecutive.getContact_Number(), adminExecutive.getPosition()};
6         String dataLine = "";
7         for (String dd : data){
8             dataLine += dd + ",";
9         }
10        dataLine = dataLine.substring(0, dataLine.length() - 1);
11        dataLine += '\n';
12        fileWriter.write(dataLine);
13    }
14    fileWriter.close();
15 }
```

#### 10.4.23get\_Admin\_Executive\_Info



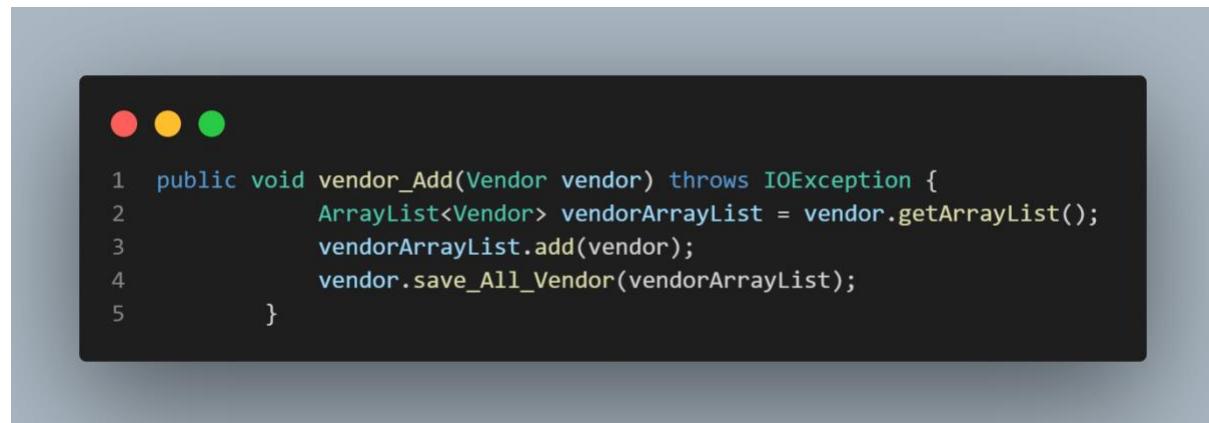
```
1 public Admin_Executive get_Admin_Executive_Info(String executiveID) throws FileNotFoundException {
2     ArrayList<Admin_Executive> adminExecutiveArrayList = getArrayList();
3     Admin_Executive result = new Admin_Executive();
4     result.setExecutiveID("0");
5     for (Admin_Executive adminExecutive : adminExecutiveArrayList) {
6         if (adminExecutive.getExecutiveID().equals(executiveID)) {
7             result = adminExecutive;
8         }
9     }
10    return result;
11 }
```

#### 10.4.24check\_Admin\_Executive\_Availability



```
1 public boolean check_Admin_Employee_Availability(String executiveID) throws FileNotFoundException {
2     boolean result = false;
3     Admin_Employee adminEmployee = new Admin_Employee();
4     ArrayList<Admin_Employee> adminEmployeeArrayList = adminEmployee.getArrayList();
5     for (Admin_Employee adminEmployee1 : adminEmployeeArrayList) {
6         if (adminEmployee1.getExecutiveID().equals(executiveID)) {
7             result = true;
8             break;
9         }
10    }
11    return result;
12 }
```

#### 10.4.25vendor\_Add



```
1 public void vendor_Add(Vendor vendor) throws IOException {
2     ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
3     vendorArrayList.add(vendor);
4     vendor.save_All_Vendor(vendorArrayList);
5 }
```

#### 10.4.26vendor\_Modify



```
1 public void vendor_Modify(Vendor vendor, String vendor_Username) throws IOException {
2     String dataLine = vendor.getDataString(vendor);
3     if (vendor.check_Vendor_Availability(vendor_Username)){
4         vendor.Delete(vendor_Username);
5         vendor.Add(vendor);
6         JOptionPane.showMessageDialog(null, "Vendor Information modifying success", "Vendor information modifying success", JOptionPane.INFORMATION_MESSAGE);
7     }
8 }
```

#### 10.4.27vendor\_Delete

```
1 public void vendor_Delete(String vendor_Username) throws IOException {
2     Vendor vendor = new Vendor();
3     ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
4     for (Vendor vendor1 : vendorArrayList){
5         if (vendor1.getVendor_Username().equals(vendor_Username))
6             vendorArrayList.remove(vendor1);
7     }
8     vendor.save_All_Vendor(vendorArrayList);
9 }
```

## 10.5 Resident

### 10.5.1 Resident

```
1 public Resident(String resident_Username, String name, char gender, String contact_Number, String unitID, int payment) {
2     this.resident_Username = resident_Username;
3     this.name = name;
4     this.gender = gender;
5     this.contact_Number = contact_Number;
6     this.unitID = unitID;
7     this.payment = payment;
8 }
```

### 10.5.2 get\_Resident\_Info

```
1 public Resident get_Resident_Info(String resident_Username) throws FileNotFoundException {
2     Resident resident = new Resident();
3     resident.setResident_Username("0");
4     ArrayList<Resident> residentArrayList = resident.getArrayList();
5     for (Resident resident1 : residentArrayList){
6         if (resident1.getResident_Username().equals(resident_Username))
7             return resident1;
8     }
9     return resident;
10 }
```

### 10.5.3 update\_Resident\_Info

```
1 public void update_Resident_Info(Resident resident, String resident_Username) throws IOException {
2     ArrayList<Resident> residentArrayList = resident.getArrayList();
3     ArrayList<Resident> residentArrayList1 = new ArrayList<>();
4     for (Resident resident1 : residentArrayList){
5         if (!(resident1.getResident_Username().equals(resident_Username)))
6             residentArrayList1.add(resident1);
7     }
8     residentArrayList1.add(resident);
9     resident.save_All_Resident(residentArrayList1);
10 }
11 }
```

### 10.5.4 make\_Payment

```
1 public void make_Payment(Invoice invoice, String pay_Username) throws IOException {
2     Payment payment1 = new Payment();
3     payment1.make_Payment(invoice, pay_Username);
4
5     Resident resident = new Resident();
6     ArrayList<Resident> residentArrayList = resident.getArrayList();
7     for (Resident resident1 : residentArrayList){
8         resident1.setPayment(resident1.get_Unpaid_Amount(resident1.getResident_Username(), resident1.getUnitID()));
9     }
10    resident.save_All_Resident(residentArrayList);
11 }
```

### 10.5.5 get\_All\_Receipt

```
1 public ArrayList<Payment> get_All_Receipt(String unitID) throws FileNotFoundException {
2     Payment payment = new Payment();
3     ArrayList<Payment> paymentArrayList = payment.get_All_Receipt(unitID);
4     return paymentArrayList;
5 }
```

### 10.5.6 get\_All\_pending\_Payment

```
1 public ArrayList<Payment> get_All_pending_Payment(String unitID) throws FileNotFoundException {
2     Payment payment = new Payment();
3     ArrayList<Payment> paymentArrayList = payment.getArrayList();
4     ArrayList<Payment> paymentArrayList1 = new ArrayList<>();
5     for (Payment payment1 : paymentArrayList)
6     {
7         if (payment1.getIssuerID().equals("") && !payment1.getUnitID().equals(unitID)) {
8             paymentArrayList1.add(payment1);
9         }
10    }
11    return paymentArrayList1;
12 }
```

### 10.5.7 get\_Statement\_for\_Resident

```
1 public ArrayList<Statement> get_Statement_for_Resident(String resident_Username) throws FileNotFoundException {
2     Statement statement = new Statement();
3     return statement.get_Statement_for_Receiver(resident_Username);
4 }
```

### 10.5.8 add\_Facility\_Booking

```
1 public void add_Facility_Booking(Facility.Booking booking) throws IOException, ClassNotFoundException {
2     ArrayList<Facility.Booking> bookingArrayList = booking.getArrayList();
3     if (new Facility().check_Facility_Availability(booking.getFacilityID()) && booking.check_TimeSlot_Availability(booking)){
4         bookingArrayList.add(booking);
5         System.out.println("pass");
6         booking.save_All_Facility_Booking(bookingArrayList);
7     }
8 }
```

### 10.5.9 view\_Resident\_Booking

```
1 public ArrayList<Facility.Booking> view_Resident_Booking(String resident_Username) throws IOException, ClassNotFoundException {
2     ArrayList<Facility.Booking> bookingArrayList = new Facility.Booking().getArrayList();
3     ArrayList<Facility.Booking> bookingArrayList1 = new ArrayList<>();
4     for (Facility.Booking booking : bookingArrayList) {
5         if (booking.getResident_Username().equals(resident_Username))
6             bookingArrayList1.add(booking);
7     }
8     return bookingArrayList1;
9 }
```

### 10.5.10cancel\_Facility\_Booking

```
1 public void cancel_Facility_Booking(String bookingID) throws IOException, ClassNotFoundException {
2     ArrayList<Facility.Booking> bookingArrayList = new Facility.Booking().getArrayList();
3     ArrayList<Facility.Booking> bookingArrayList1 = new ArrayList<>();
4     for (Facility.Booking booking : bookingArrayList){
5         if (!(booking.getBookingID().equals(bookingID)))
6             bookingArrayList1.add(booking);
7     }
8     new Facility.Booking().save_All_Facility_Booking(bookingArrayList1);
9 }
```

### 10.5.11update\_Facility\_Booking

```
1 public void update_Facility_Booking(Facility.Booking booking, String bookingID) throws IOException, ClassNotFoundException {
2     cancel_Facility_Booking(bookingID);
3     add_Facility_Booking(booking);
4 }
```

### 10.5.12apply\_Visitor\_Pass

```
1 public void apply_Visitor_Pass(Visitor_Pass visitorPass) throws IOException {
2     visitorPass.setStatus("Disapproved");
3     ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
4     visitorPassArrayList.add(visitorPass);
5     visitorPass.save_All_Visitor(visitorPassArrayList);
6 }
```

### 10.5.13view\_All\_Visitor\_Pass\_Apply

```
1 public ArrayList<Visitor_Pass> view_All_Visitor_Pass_Apply(String resident_Username) throws FileNotFoundException {
2     Visitor_Pass visitorPass = new Visitor_Pass();
3     ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
4     ArrayList<Visitor_Pass> visitorPassesReturn = new ArrayList<>();
5     for (Visitor_Pass visitorPass1 : visitorPassArrayList){
6         if (visitorPass1.getResident_Username().equals(resident_Username))
7             visitorPassesReturn.add(visitorPass1);
8     }
9     return visitorPassesReturn;
10 }
```

### 10.5.14cancel\_Visitor\_Pass

```
1 public void cancel_Visitor_Pass(String visitor_Pass_ID) throws IOException {
2     Visitor_Pass visitorPass = new Visitor_Pass();
3     ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
4     ArrayList<Visitor_Pass> visitorPassArrayList1 = new ArrayList<>();
5     for (Visitor_Pass visitorPass1 : visitorPassArrayList){
6         if (!(visitorPass1.getVisitor_Pass_ID().equals(visitor_Pass_ID)))
7             visitorPassArrayList1.add(visitorPass1);
8     }
9     visitorPass.save_All_Visitor(visitorPassArrayList1);
10 }
```

### 10.5.15update\_Visitor\_Pass

```
1 public void update_Visitor_Pass(Visitor_Pass visitorPass) throws IOException {
2     cancel_Visitor_Pass(visitorPass.getVisitor_Pass_ID());
3     apply_Visitor_Pass(visitorPass);
4 }
```

### 10.5.16log\_Complaint

```
1 public void log_Complaint(Complaint complaint) throws IOException {
2     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
3     complaintArrayList.add(complaint);
4     complaint.save_All_Complaint(complaintArrayList);
5 }
```

### 10.5.17view\_Complaint

```
1 public ArrayList<Complaint> view_Complaint(String resident_Username) throws FileNotFoundException {
2     Complaint complaint = new Complaint();
3     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
4     ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
5     for (Complaint complaint1 : complaintArrayList){
6         if (complaint1.getResident_Username().equals(resident_Username)){
7             complaintArrayList1.add(complaint1);
8         }
9     }
10    return complaintArrayList1;
11 }
```

### 10.5.18update\_Complaint

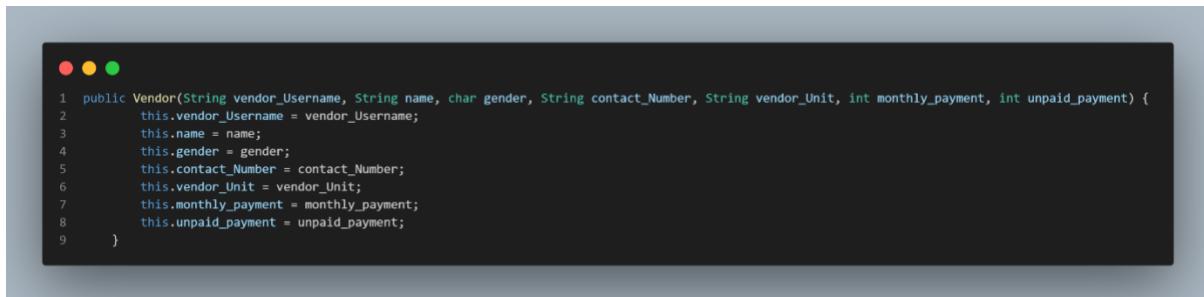
```
1 public void update_Complaint(Complaint complaint) throws IOException {
2     if (complaint.check_Complaint_Availability(complaint.getComplaintID())){
3         cancel_Complaint(complaint.getComplaintID());
4         log_Complaint(complaint);
5     }
6 }
```

### 10.5.19cancel\_Complaint

```
1 public void cancel_Complaint(String complaintID) throws IOException {
2     Complaint complaint = new Complaint();
3     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
4     for (Complaint com : complaintArrayList){
5         if (com.getComplaintID().equals(complaintID))
6             complaintArrayList.remove(com);
7     }
8     complaint.save_All_Complaint(complaintArrayList);
9 }
```

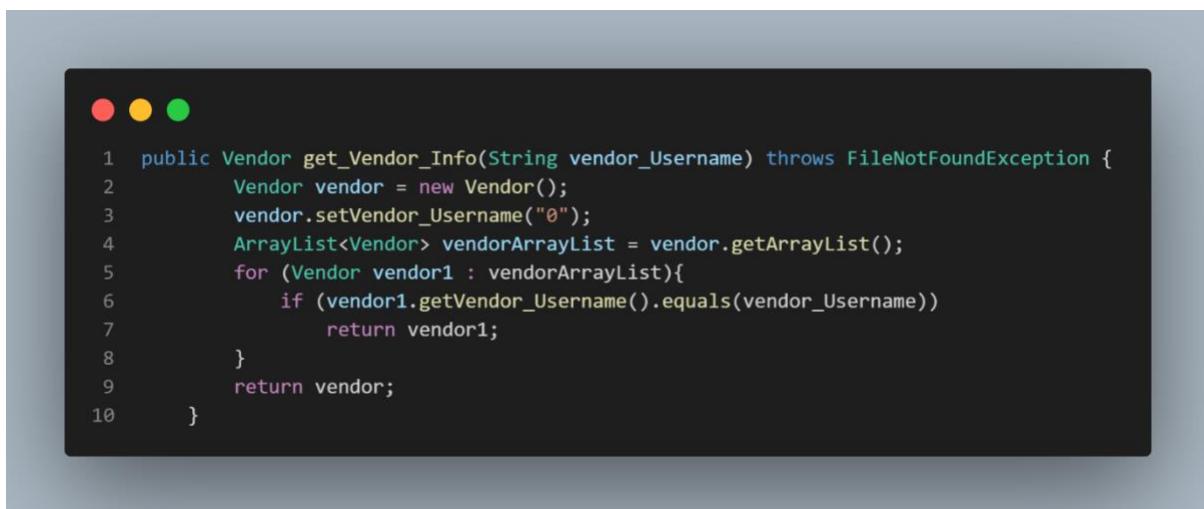
## 10.6 Vendor

### 10.6.1 Vendor



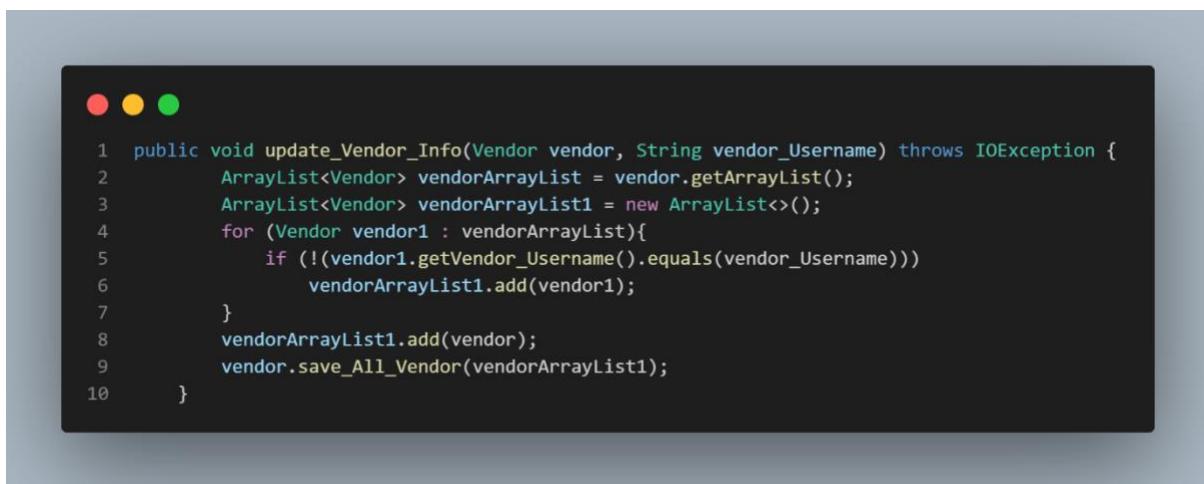
```
1 public Vendor(String vendor_Username, String name, char gender, String contact_Number, String vendor_Unit, int monthly_payment, int unpaid_payment) {
2     this.vendor_Username = vendor_Username;
3     this.name = name;
4     this.gender = gender;
5     this.contact_Number = contact_Number;
6     this.vendor_Unit = vendor_Unit;
7     this.monthly_payment = monthly_payment;
8     this.unpaid_payment = unpaid_payment;
9 }
```

### 10.6.2 get\_Vendor\_Info



```
1 public Vendor get_Vendor_Info(String vendor_Username) throws FileNotFoundException {
2     Vendor vendor = new Vendor();
3     vendor.setVendor_Username("0");
4     ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
5     for (Vendor vendor1 : vendorArrayList){
6         if (vendor1.getVendor_Username().equals(vendor_Username))
7             return vendor1;
8     }
9     return vendor;
10 }
```

### 10.6.3 update\_Vendor\_Info



```
1 public void update_Vendor_Info(Vendor vendor, String vendor_Username) throws IOException {
2     ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
3     ArrayList<Vendor> vendorArrayList1 = new ArrayList<>();
4     for (Vendor vendor1 : vendorArrayList){
5         if (!(vendor1.getVendor_Username().equals(vendor_Username)))
6             vendorArrayList1.add(vendor1);
7     }
8     vendorArrayList1.add(vendor);
9     vendor.save_All_Vendor(vendorArrayList1);
10 }
```

#### 10.6.4 make\_Payment

```
1 public void make_Payment(Invoice invoice, String pay_Username) throws IOException {
2     Payment payment1 = new Payment();
3     payment1.make_Payment(invoice, pay_Username);
4
5     Vendor vendor = new Vendor();
6     ArrayList<Vendor> vendorArrayList = vendor.getArrayList();
7     for (Vendor vendor1 : vendorArrayList){
8         vendor1.setUnpaid_payment(vendor1.get_Unpaid_Amount(vendor1.getVendor_Username(), vendor1.getVendor_Unit()));
9     }
10    vendor.save_All_Vendor(vendorArrayList);
11 }
```

#### 10.6.5 change\_Monthly\_Payment\_Rental

```
1 public void change_Monthly_Payment_Rental(String vendor_Unit, int rental_Amount) throws IOException {
2     ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
3     for (Vendor vendor : vendorArrayList){
4         if (vendor.getVendor_Unit().equals(vendor_Unit)){
5             vendor.setMonthly_payment(rental_Amount);
6         }
7     }
8     save_All_Vendor(vendorArrayList);
9 }
```

#### 10.6.6 get\_All\_Receipt

```
1 public ArrayList<Payment> get_All_Receipt(String vendor_Unit) throws FileNotFoundException {
2     Payment payment = new Payment();
3     ArrayList<Payment> paymentArrayList = payment.get_All_Receipt(vendor_Unit);
4     return paymentArrayList;
5 }
```

### 10.6.7 get\_All\_pending\_Payment

```
1 public ArrayList<Payment> get_All_pending_Payment(String vendor_Unit) throws FileNotFoundException {
2     Payment payment = new Payment();
3     ArrayList<Payment> paymentArrayList = payment.getArrayList();
4     ArrayList<Payment> paymentArrayList1 = new ArrayList<>();
5     for (Payment payment1 : paymentArrayList)
6     {
7         if (payment1.getIssuerID().equals("") && payment1.getUnitID().equals(vendor_Unit))
8             paymentArrayList1.add(payment1);
9     }
10    return paymentArrayList1;
11 }
```

### 10.6.8 get\_Statement\_for\_Vendor

```
1 public ArrayList<Statement> get_Statement_for_Vendor(String vendor_Username) throws FileNotFoundException {
2     Statement statement = new Statement();
3     return statement.get_Statement_for_Receiver(vendor_Username);
4 }
5
```

### 10.6.9 get\_Unit\_All\_Unpaid\_Invoice

```
1 public ArrayList<Invoice> get_Unit_All_Unpaid_Invoice(String vendor_Unit) throws FileNotFoundException {
2     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
3     ArrayList<Invoice> invoiceArrayList1 = new ArrayList<>();
4     for (Invoice invoice : invoiceArrayList){
5         if (invoice.getUnitID().equals(vendor_Unit) && invoice.getStatus().equals("unpaid"))
6             invoiceArrayList1.add(invoice);
7     }
8     return invoiceArrayList1;
9 }
```

### 10.6.10get\_Unit\_All\_Invoice

```
1 public ArrayList<Invoice> get_Unit_All_Invoice(String vendor_Unit) throws FileNotFoundException {
2     ArrayList<Invoice> invoiceArrayList = new Invoice().getArrayList();
3     ArrayList<Invoice> invoiceArrayList1 = new ArrayList<>();
4     for (Invoice invoice : invoiceArrayList){
5         if (invoice.getUnitID().equals(vendor_Unit))
6             invoiceArrayList1.add(invoice);
7     }
8     return invoiceArrayList1;
9 }
```

### 10.6.11vendor\_Unit\_rent

```
1 public ArrayList<String> vendor_Unit_rent(String vendor_Username) throws FileNotFoundException {
2     ArrayList<Vendor> vendorArrayList = new Vendor().getArrayList();
3     ArrayList<String> data = new ArrayList<>();
4     for (Vendor vendor1 : vendorArrayList){
5         if (vendor1.getVendor_Username().equals(vendor_Username))
6             data.add(vendor1.getVendor_Unit());
7     }
8     return data;
9 }
```

### 10.6.12log\_Complaint

```
1 public void log_Complaint(Complaint complaint) throws IOException {
2     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
3     complaintArrayList.add(complaint);
4     complaint.save_All_Complaint(complaintArrayList);
5 }
```

### 10.6.13view\_Complaint

```
1 public ArrayList<Complaint> view_Complaint(String resident_Username) throws FileNotFoundException {
2     Complaint complaint = new Complaint();
3     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
4     ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
5     for (Complaint complaint1 : complaintArrayList){
6         if (complaint1.getResident_Username().equals(resident_Username))
7             complaintArrayList1.add(complaint1);
8     }
9     return complaintArrayList1;
10 }
```

### 10.6.14update\_Complaint

```
1 public void update_Complaint(Complaint complaint, String complaintID) throws IOException {
2     if (complaint.check_Complaint_Availability(complaintID)){
3         cancel_Complaint(complaintID);
4         log_Complaint(complaint);
5     }
6 }
```

### 10.6.15cancel\_Complaint

```
1 public void cancel_Complaint(String complaintID) throws IOException {
2     Complaint complaint = new Complaint();
3     ArrayList<Complaint> complaintArrayList = complaint.getArrayList();
4     ArrayList<Complaint> complaintArrayList1 = new ArrayList<>();
5     for (Complaint com : complaintArrayList){
6         if (!(com.getComplaintID().equals(complaintID)))
7             complaintArrayList1.add(com);
8     }
9     complaint.save_All_Complaint(complaintArrayList1);
10 }
```

## 10.7 Security Guard

### 10.7.1 SecurityGuard extends Employee

```
1 public class SecurityGuard extends Employee implements Serializable{  
2     protected String position = "Security Guard";  
3     private File security_Guard_Info_txt = new File("src/Database/SecurityGuard_Information.txt");  
4     private File visitor_Entry_Record_txt = new File("src/Database/Visitors_Entry_Record.txt");  
5     public SecurityGuard(){  
6         public SecurityGuard(String employeeID){  
7             super();  
8             super.set_Info(employeeID, name, gender, contact_Number, salary);  
9         }  
10    }
```

### 10.7.2 update\_SecurityGuard\_Info

```
1 public void update_SecurityGuard_Info(SecurityGuard securityGuard, String employeeID) throws IOException, ClassNotFoundException {  
2     ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();  
3     for (SecurityGuard securityGuard1 : securityGuardArrayList){  
4         if (securityGuard1.getEmployeeID().equals(employeeID)){  
5             securityGuardArrayList.remove(securityGuard1);  
6         }  
7     securityGuardArrayList.add(securityGuard);  
8     securityGuard.save_All_SecurityGuard(securityGuardArrayList);  
9 }
```

### 10.7.3 get\_Security\_Guard\_Info

```
1 public SecurityGuard get_Security_Guard_Info(String employeeID) throws IOException, ClassNotFoundException {  
2     SecurityGuard securityGuard = new SecurityGuard();  
3     securityGuard.setEmployeeID("0");  
4     ArrayList<SecurityGuard> securityGuardArrayList = securityGuard.getArrayList();  
5     for (SecurityGuard securityGuard1 : securityGuardArrayList){  
6         if (securityGuard1.getEmployeeID().equals(employeeID)){  
7             return securityGuard1;  
8         }  
9     }  
10    return securityGuard;  
11 }
```

#### 10.7.4 add\_Visitor\_Entry\_Record

```
● ● ●
1 public void add_Visitor_Entry_Record(String visitor_Pass_ID) throws IOException {
2     LocalDateTime myDateObj = LocalDateTime.now();
3     DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("MM.dd.yyyy:HHmmss");
4     String data = visitor_Pass_ID + ":" + myDateObj.format(myFormatObj) + '\n';
5     FileWriter fileWriter = new FileWriter(visitor_Entry_Record_txt, true);
6     fileWriter.write(data);
7     fileWriter.close();
8     JOptionPane.showMessageDialog(null, "Visitor Entry Record add success", "Record success added", JOptionPane.INFORMATION_MESSAGE);
9 }
```

#### 10.7.5 delete\_Visitor\_Entry\_Record

```
● ● ●
1 public void delete_Visitor_Entry_Record(String visitor_Pass_ID) throws IOException {
2     ArrayList<String[]> record = get_all_Visitor_Entry_Record();
3     for (String[] data : record){
4         if (data[0].equals(visitor_Pass_ID)){
5             record.remove(data);
6         }
7     }
8     save_All_Visitor_Entry_Record(record);
9 }
10
```

#### 10.7.6 update\_Visitor\_Entry\_Record

```
● ● ●
1 public void update_Visitor_Entry_Record(String visitor_Pass_ID, String dateTIme) throws IOException {
2     delete_Visitor_Entry_Record(visitor_Pass_ID);
3     add_Visitor_Entry_Record(visitor_Pass_ID, dateTIme);
4 }
5
```

### 10.7.7 checkIn\_CheckPoint

```
1 public void checkIn_CheckPoint(String securityGuardID, String checkPointID) throws IOException, ClassNotFoundException {
2     SecurityGuard securityGuard = new SecurityGuard();
3     CheckPoint checkPoint = new CheckPoint();
4     if (securityGuard.check_SecurityGuard_Availability(securityGuardID) && checkPoint.check_CheckPoint_Existence(checkPointID)){
5         CheckPoint.Record record = new CheckPoint.Record();
6         ArrayList<CheckPoint.Record> recordArrayList = record.getArrayList();
7         LocalDate date = LocalDate.now();
8         LocalTime time = LocalTime.now();
9         recordArrayList.add(new CheckPoint.Record(securityGuardID, checkPointID, date, time));
10        record.save_All_Record(recordArrayList);
11    }
12 }
```

## 10.8 Visitor

### 10.8.1 search\_Visitor\_Pass\_Info

```
1 public Visitor_Pass search_Visitor_Pass_Info(String visitor_Pass_ID) throws FileNotFoundException {
2     Visitor_Pass visitorPass = new Visitor_Pass();
3     visitorPass.setVisitor_Pass_ID("0");
4     ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
5     for (Visitor_Pass visitorPass1 : visitorPassArrayList){
6         if (visitorPass1.getVisitor_Pass_ID().equals(visitor_Pass_ID)){
7             return visitorPass1;
8         }
9     }
10    return visitorPass;
11 }
```

### 10.8.2 search\_Visitor\_Pass\_Info\_by\_Name

```
1 public Visitor_Pass search_Visitor_Pass_Info_by_Name(String visitor_Name) throws FileNotFoundException {
2     Visitor_Pass visitorPass = new Visitor_Pass();
3     visitorPass.setVisitor_Pass_ID("0");
4     ArrayList<Visitor_Pass> visitorPassArrayList = visitorPass.getArrayList();
5     for (Visitor_Pass visitorPass1 : visitorPassArrayList){
6         if (visitorPass1.getVisitor_Name().equals(visitor_Name)){
7             return visitorPass1;
8         }
9     }
10    return visitorPass;
11 }
```

### 10.8.3 get\_Auto\_Visitor\_Pass\_ID

```
1 public String get_Auto_Visitor_Pass_ID() throws FileNotFoundException {
2     FileReader fileReader = new FileReader(visitor_Pass_Info_txt);
3     Scanner scanner = new Scanner(fileReader);
4     scanner.nextLine();
5     Integer num = 0;
6     while (scanner.hasNextLine()){
7         String[] data = scanner.nextLine().split(":", 9);
8         String number = data[0].substring(2);
9         num = Integer.parseInt(number);
10        num += 1;
11    }
12    String str = "VI" + num.toString();
13    return str;
14 }
```

## 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>