University of Nevada, Reno

Department of Computer Science and Engineering

CS 425: Software Engineering

Optimum Property Fix

Team 05

Aisha Co, Melissa Flores, Joanna Lopez, Nasrin Juana and Araam Zaremenhjardi

Instructor: Sergiu Dascalu, Vihn Le, and Devrin Lee

Advisor: Erin Keith

November 17, 2021

Table of Contents

1. Abstract

The University of Nevada, Reno's (UNR) dormitory maintenance requests system presents itself as a problem in need of improvement ridden with inconveniences and flaws to dormitory students. The intended audience of Optimum Property Fix (OPF) is dormitory students at UNR and facilities services for dormitories. OPF is a planned approach to solving this problem through web technologies and advancements in Artificial Intelligence. OPF intends to be an overhaul of UNR's current dormitory upkeep service system through the elimination of paper trails while providing an accessible web application for both dormitory students and facilities and services members. While providing two different user-focused experiences, OPF intends to encase all records and management of facilities upkeep in one place while overall enhancing the original functionality of the previous paper-based system through artificial intelligence and content-rich data such as images and video.

2. Introduction

Optimum Property Fix (OPF) is a web application with the purpose of overhauling University of Nevada, Reno (UNR) current dormitory facility service systems by providing a suite of tools to both facility management members and dormitory students to manage living spaces. The current system has paper trails, is vulnerable to being lost, does not incentivize submission of minor issues within living spaces, and lacks progress tracking of maintenance requests to students. OPF will rectify these issues while providing analytical insights that allow Facilities and Management to preemptively manage dormitory buildings and assess dormitory buildings' health.

OPF has been updated with underlying technical aspects of the application including system design of the database data and behaviors that interact with the data. The database has been developed with abstractions of data defined within a database table. Definitions within the database tables include labels, types, and key types of the data (being equipped with a primary key and foreign key). These definitions within the database table serve as a basis to design the application's class abstractions with respective attributes to store data and methods. These relationships are defined through aggregations, compositions, and multiplicities. OPF has further specified the behavior of the system through descriptions of activity diagrams and state charts. The order of events such as the creation of maintenance requests, the creation of reports, and session management are described by activity diagrams. The general objects and interactions between other objects within the system are detailed by state charts.

OPF further tracked its progress and management changes to the software code via Github version control. The README file was updated to reflect the progress of the software engineering project. The text file has been updated with introductions and explanations of the overall project. OPF further developed user interfaces and user experience for the OPF web application. The newly added functionalities include data entry boxes, panels, and menu items. Additionally, OPF standardized a color scheme throughout the application for a more focused and consistent style. Standardizing OPF's user interface provides clear and intuitive user navigation while adopting a structure to OPF.

3. High-Level Design and Medium-Level Design

3.1. Context Diagram

A context diagram is used in software engineering that defines the boundaries between the system and its environment. The context diagram shows the entities that interact with each other and their relationships with the system. Additionally, the benefits of a context diagram include the scope and boundaries of the system at a glance including other systems that interface with it. As the scope of the project emerges, the context diagram allows Team 05 to easily change and/or extend an entity without disarranging with the rest of the diagram. Figure 3.3.1, a context diagram, is a high-level view of the OPF web application system.

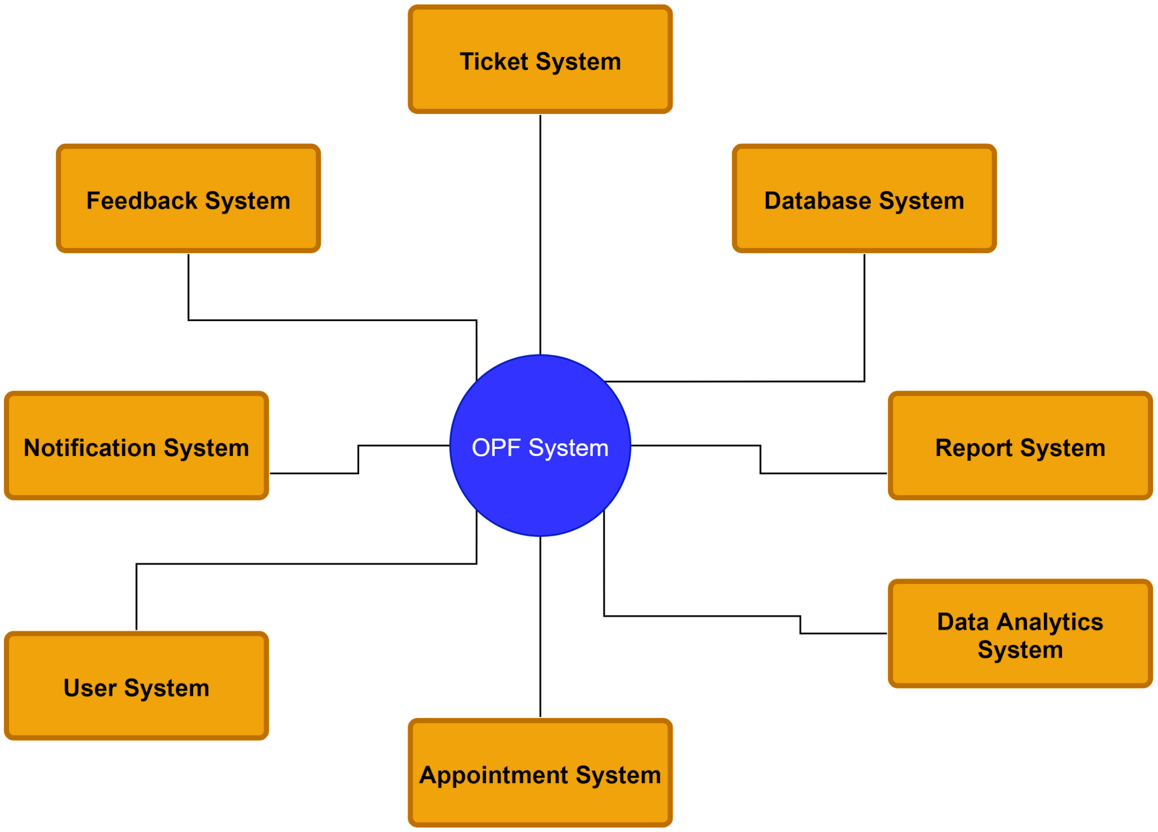


Fig. 3.1.1: A high-level overview of a context diagram of the OPF web application describing the relationships between the system and its environment.

3.2. Program Units

Program units in software engineering contain declarations, type definitions, procedures, and/ or interfaces. Program units provide initial values types for variables in named common blocks. OPF is using an object-oriented solution where the classes represent the program units. The design class diagram of the OPF web application details the attributes, operations, relationships, and multiplicity constraints. All tables in Section 3.2 detail the class diagrams that are used in the OPF web application. The tables include the class names, functions, and a brief description of its main attributes. Figure 3.2.1, is the overall visual representation of the class diagram for the OPF web application and its overall software architecture.

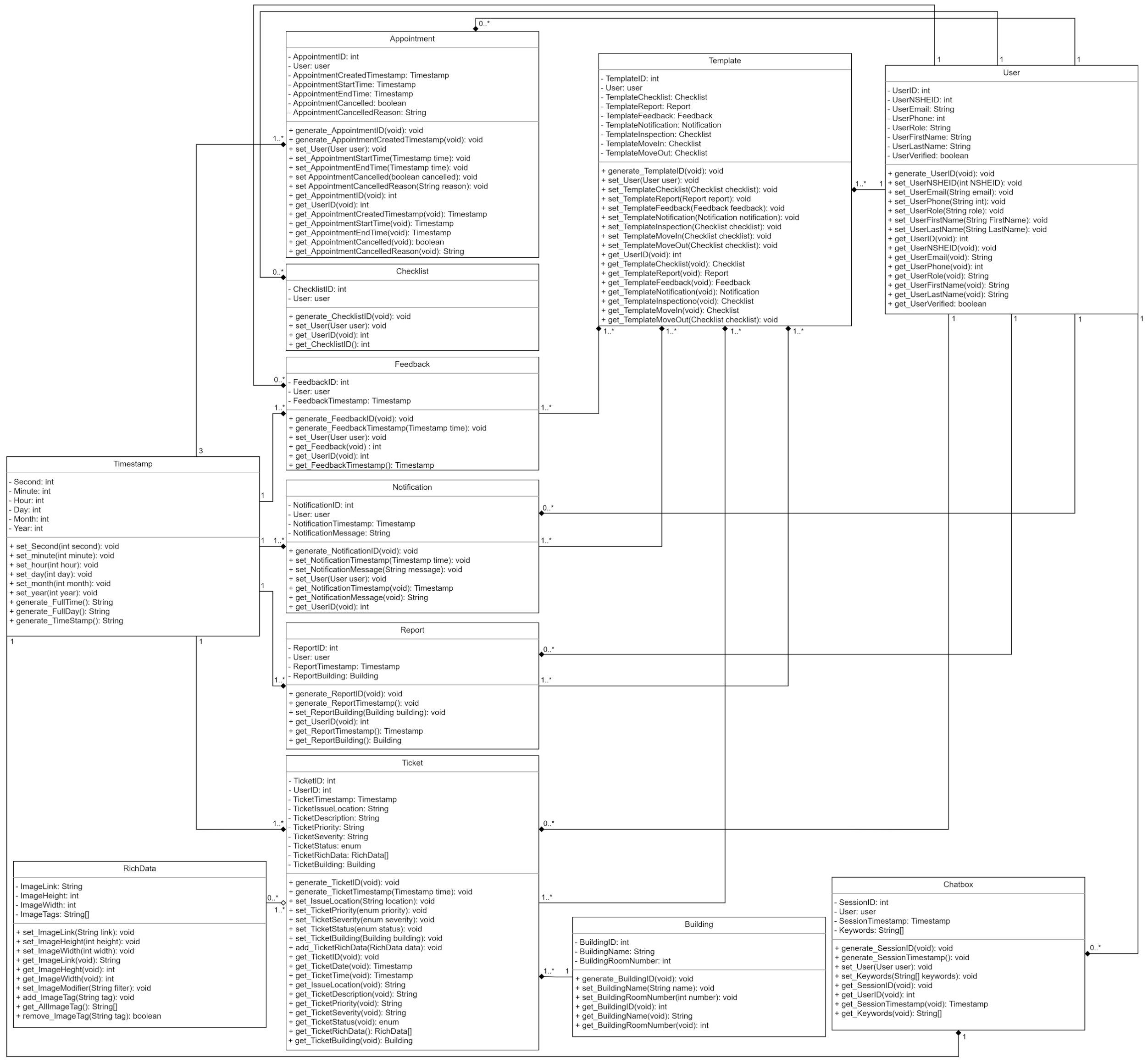

Appointment
- AppointmentID: int
- User: user
- AppointmentCreatedTimestamp: Timestamp
- AppointmentStartTime: Timestamp
- AppointmentEndTime: Timestamp
- AppointmentCancelled: boolean
- AppointmentCancelledReason: String
+ generate_AppointmentID(void): void
+ generate_AppointmentCreatedTimestamp(void): void
+ set_User(User user): void
+ set_AppointmentStartTime(Timestamp time): void
+ set_AppointmentEndTime(Timestamp time): void
+ set AppointmentCancelled(boolean cancelled): void
+ set AppointmentCancelledReason(String reason): void
+ get_AppointmentID(void): int
+ get_UserID(void): int
+ get_AppointmentCreatedTimestamp(void): Timestamp
+ get_AppointmentStartTime(void): Timestamp
+ get_AppointmentEndTime(void): Timestamp
+ get_AppointmentCancelled(void): boolean
+ get_AppointmentCancelledReason(void): String
Template
- TemplateID: int
- User: user
- TemplateChecklist: Checklist
- TemplateReport: Report
- TemplateFeedback: Feedback
- TemplateNotification: Notification
- TemplateInspection: Checklist
- TemplateMoveIn: Checklist
- TemplateMoveOut: Checklist
+ generate_TemplateID(void): void
+ set_User(User user): void
+ set_TemplateChecklist(Checklist checklist): void
+ set_TemplateReport(Report report): void
+ set_TemplateFeedback(Feedback feedback): void
+ set_TemplateNotification(Notification notification): void
+ set_TemplateInspection(Checklist checklist): void
+ set_TemplateMoveIn(Checklist checklist): void
+ set_TemplateMoveOut(Checklist checklist): void
+ get_UserID(void): int
+ get_TemplateChecklist(void): Checklist
+ get_TemplateReport(void): Report
+ get_TemplateFeedback(void): Feedback
+ get_TemplateNotification(void): Notification
+ get_TemplateInspectiono(void): Checklist
+ get_TemplateMoveIn(void): Checklist
+ get_TemplateMoveOut(Checklist checklist): void
User
- UserID: int
- UserNSHEID: int
- UserEmail: String
- UserPhone: int
- UserRole: String
- UserFirstName: String
- UserLastName: String
- UserVerified: boolean
+ generate_UserID(void): void
+ set_UserNSHEID(int NSHEID): void
+ set_UserEmail(String email): void
+ set_UserPhone(String int): void
+ set_UserRole(String role): void
+ set_UserFirstName(String FirstName): void
+ set_UserLastName(String LastName): void
+ get_UserID(void): int
+ get_UserNSHEID(void): void
+ get_UserEmail(void): String
+ get_UserPhone(void): int
+ get_UserRole(void): String
+ get_UserFirstName(void): String
+ get_UserLastName(void): String
+ get_UserVerified: boolean
Checklist
- ChecklistID: int
- User: user
+ generate_ChecklistID(void): void
+ set_User(User user): void
+ get_UserID(void): int
+ get_ChecklistID(): int
Feedback
- FeedbackID: int
- User: user
- FeedbackTimestamp: Timestamp
+ generate_FeedbackID(void): void
+ generate_FeedbackTimestamp(Timestamp time): void
+ set_User(User user): void
+ get_Feedback(void) : int
+ get_UserID(void): int
+ get_FeedbackTimestamp(): Timestamp
Timestamp
- Second: int
- Minute: int
- Hour: int
- Day: int
- Month: int
- Year: int
+ set_Second(int second): void
+ set_minute(int minute): void
+ set_hour(int hour): void
+ set_day(int day): void
+ set_month(int month): void
+ set_year(int year): void
+ generate_FullTime(): String
+ generate_FullDay(): String
+ generate_TimeStamp(): String
Notification
- NotificationID: int
- User: user
- NotificationTimestamp: Timestamp
- NotificationMessage: String
+ generate_NotificationID(void): void
+ set_NotificationTimestamp(Timestamp time): void
+ set_NotificationMessage(String message): void
+ set_User(User user): void
+ get_NotificationTimestamp(void): Timestamp
+ get_NotificationMessage(void): String
+ get_UserID(void): int
Report
- ReportID: int
- User: user
- ReportTimestamp: Timestamp
- ReportBuilding: Building
+ generate_ReportID(void): void
+ generate_ReportTimestamp(): void
+ set_ReportBuilding(Building building): void
+ get_UserID(void): int
+ get_ReportTimestamp(): Timestamp
+ get_ReportBuilding(): Building
Ticket
- TicketID: int
- UserID: int
- TicketTimestamp: Timestamp
- TicketIssueLocation: String
- TicketDescription: String
- TicketPriority: String
- TicketSeverity: String
- TicketStatus: enum
- TicketRichData: RichData[]
- TicketBuilding: Building
+ generate_TicketID(void): void
+ generate_TicketTimestamp(Timestamp time): void
+ set_IssueLocation(String location): void
+ set_TicketPriority(enum priority): void
+ set_TicketSeverity(enum severity): void
+ set_TicketStatus(enum status): void
+ set_TicketBuilding(Building building): void
+ add_TicketRichData(RichData data): void
+ get_TicketID(void): void
+ get_TicketDate(void): Timestamp
+ get_TicketTime(void): Timestamp
+ get_IssueLocation(void): String
+ get_TicketDescription(void): String
+ get_TicketPriority(void): String
+ get_TicketSeverity(void): String
+ get_TicketStatus(void): enum
+ get_TicketRichData(): RichData[]
+ get_TicketBuilding(void): Building
RichData
- ImageLink: String
- ImageHeight: int
- ImageWidth: int
- ImageTags: String[]
+ set_ImageLink(String link): void
+ set_ImageHeight(int height): void
+ set_ImageWidth(int width): void
+ get_ImageLink(void): String
+ get_ImageHeght(void): int
+ get_ImageWidth(void): int
+ set_ImageModifier(String filter): void
+ add_ImageTag(String tag): void
+ get_AllImageTag(): String[]
+ remove_ImageTag(String tag): boolean
Building
- BuildingID: int
- BuildingName: String
- BuildingRoomNumber: int
+ generate_BuildingID(void): void
+ set_BuildingName(String name): void
+ set_BuildingRoomNumber(int number): void
+ get_BuildingID(void): int
+ get_BuildingName(void): String
+ get_BuildingRoomNumber(void): int
Chatbox
- SessionID: int
- User: user
- SessionTimestamp: Timestamp
- Keywords: String[]
+ generate_SessionID(void): void
+ generate_SessionTimestamp(): void
+ set_User(User user): void
+ set_Keywords(String[] keywords): void
+ get_SessionID(void): void
+ get_UserID(void): int
+ get_SessionTimestamp(void): Timestamp
+ get_Keywords(void): String[]

Fig. 3.2.1: The design class diagram representing relationships, and multiplicity constraints for the OPF web application.

Table 3.2.1: The class “User” contains all aspects of the methods that perform calls on behalf of the user and the values based on the “User” table

Class: User	The “User” class stores essential primitives with appropriate methods to interact with data relating to each user of the application. The class allows for a standardized interface to use dormitory student data to keep a record of dormitory students' details for identification and record-keeping.
get_UserID( )	The function gets the user identification number for the OPF web application. The user identification is important to specify between each user of the application. Additionally, if the user identification number is not found, the method will throw an error.
get_UserEmail( )	The function gets the user email for communication purposes between the user and OPF. User email may also be used to verify the account on the website to create maintenance tickets.
get_UserRole( )	The function gets the user role for application operations. This function is used to change the application functionality to match the role of the user. This allows dormitory students’ and administrators’ user experience to be changed with additional unique features.

Table 3.2.2: The class “Building” contains all aspects of the methods that perform calls based on the “Building” table.

Class: Building	The “Building” class stores primitives with appropriate methods to interact with data by abstracting Building values. The class is used to abstract buildings and information pertaining to the University of Nevada, Reno campus to be used in conjunction with tickets to identify the location of a ticket issue.
generate_BuildingID( )	This function generates the building identification number. The building identification number is used to give a specific numeric identification code that makes each building unique. This numeric identification code eliminates the possibility of name errors associated with String values.
set_BuildingName(name)	This function takes in a String “name” parameter to name a building name in the Building class. The class is used to map an identification code to a dormitory building on the University of Nevada, Reno campus. The function provides tying the identification code to a human-readable String for

	dormitory students.
set_BuildingRoomNumber (number)	This function takes an integer “number” to apply to a room in a building. This function assists in adding additional data to building information. This is used in conjunction with tickets for identifying the room used for maintenance tickets.

Table 3.2.3: The class “Notification” contains all aspects of notifying users on the OPF web application based on the “Notification” table.

Class: Notification	The role of the “Notification” class stores appropriate methods that interact with data by abstracting the creation and content of notifications. This class is critical to the users of the application and plays a large role in notifying users of appointments and maintenance tickets.
generate_NotificationID( )	This function sets the notification identification for differentiating different types for different users. The notification identification number is used to give a specific numeric identification code that makes each notification unique. This numeric identification code eliminates the possibility of confusion between different types of notifications for the users.
get_NotificationTimestamp( )	This function gets the notification timestamp for the date and times of different notifications. The timestamp will be used to visually show the users when a notification was delivered to them which will allow them to organize their notifications and take actions based on the notification and the time they were delivered to the users.
get_NotificationMessage( )	This function gets the notification message to display messages to different users by phone or email. The notification message includes a simple and brief description of what needs the users’ attention.

Table 3.2.4: The class “Ticket” contains all aspects of methods and functions based on the “Notification” table.

Class: Ticket	The role of the “Ticket” class is the core concept of the OPF web application where maintenance tickets are created and tracked. The class is expected to have components such as priorities, a description of the issue, and time associated with each ticket.
set_TicketPriority(priority)	The function sets the ticket priority as either high, medium,

	and/or low. The priority level is set based on the criticality of the issue reported by the dormitory student and is used by OPF to determine the assignment time for the maintenance appointment.
set_TicketStatus(status)	The function sets the ticket status as either completed, canceled, in-progress, submitted, or reviewed. The ticket status will allow the users to know how their ticket is being processed at the moment.
get_TicketRichData( )	This function gets the ticket-rich data such as image tags. This function allows OPF to obtain the details of the tickets in visual representations, including images and videos.

Table 3.2.5: The class "Appointment" contains all aspects of methods and functions based on the "Appointment" table.

Class: Appointment	The "Appointment" class stores abstract data types, primitives, and methods used to model users' appointments and appointment operations. The class is used to describe information associated with the scheduling of appointments for service requests, inspections, and other services.
generate_AppointmentID( )	This function generates the appointment identification number to differentiate between different appointments made by users. The appointment identification number is used to give a specific numeric identification code that makes each appointment unique. This numeric identification code eliminates the possibility of confusion between different appointments for the users.
get_AppointmentID( )	This function gets the appointment identification number to differentiate between different appointments made by users. This function allows OPF to obtain the unique appointment ID and then use that ID to obtain information about the specified appointment.
get_AppointmentStartTime( )	This function gets the specific appointment start time for the user. The appointment start time will be obtained and recorded by OPF so that the user knows when the appointment will start. This function notifies the user of when their appointment is coming up, more specifically when it is getting close to the start time recorded.
get_AppointmentEndTime( )	This function gets the specific appointment end time for the user. The appointment end time will be obtained and recorded

	by OPF so that the user knows when the appointment will end, from the start time. Once the start time has been obtained, OPF will determine the end time according to the length the appointment was set to. This function saves the end time and notifies the user once their appointment has ended.
get_AppointmentCancelled( )	This function gets the appointment if a user cancels an appointment. This function allows OPF to save the canceled appointment and its information and notify everyone involved in the appointment of the appointment status.

Table 3.2.6: The class “Template” contains aspects of methods and function.

Class: Template	The class “Template” is a document that has the basic formatting for OPF documents. The templates used on the OPF web application will consist of reports, notifications, feedback, and move-in/out templates.
generate_TemplateID( )	This function generates a template identification number for different templates on the OPF web application. The template identification number is used to give a unique numeric identification code. This numeric identification code will aid in differentiating between the different types of templates.

Table 3.2.7: The class “Chatbox” contains aspects of methods and functions based on the “Chatbox” table.

Class: Chatbox	The role of class “Chatbox” is used on the OPF web application as a real-time online chat and online interaction with users. The class contains session identifiers as well as keywords. The sessions will be stored as log files in the database.
get_SessionID( )	This function gets the session identification number for each chatbox session launched by a user. The session identification number is used to give a specific numeric identification code that makes each session unique. The session ID will be used by OPF and its users to get information about each of the sessions.
get_Keywords( )	This function gets keyword identifications triggered by users for text queries on the OPF web application. The database already contains keywords and associated information to print for the words. This function will be called when a user mentions certain words or phrases, matching the keywords stored in the database, in the chatbox. Once the keyword

	identification is obtained, OPF will be able to use it and print the associated information with the keyword.

Table 3.2.8: The class "Checklist" contains aspects of methods and functions based on the "Checklist" table.

Class: Checklist	The class "Checklist" aids as a list of items needed to be completed or considered in the OPF web application. The class contains the user information as well as a checklist identifier.
generate_ChecklistID( )	The function generates the checklist identification number to differentiate between different checklist questions. The checklist identification number is used to give a specific numeric identification code that makes each checklist unique. The checklist ID will be used by OPF and its users to get user information associated with each of the checklists.

Table 3.2.9: The class "Feedback" contains aspects of methods and functions based on the "Feedback" table.

Class: Feedback	The class "Feedback" contains all methods and functions needed to deliver feedback to appropriate users on the OPF web application. The class contains the user identification, timestamp, and description via a visual representation of the application.
generate_FeedbackID( )	The function generates the feedback identification number to differentiate between the given feedback questions. The feedback identification number is used to give a specific numeric identification code that makes each feedback unique. The feedback ID will be used by OPF and its users to get user information associated with the feedback.
get_Feedback( )	The function gets feedback from the user for the completed maintenance job. The feedback will be obtained once the user has chosen which maintenance request it is being made for and once it is submitted. The feedback will be generated as a report and recorded in the database. It will then be sent to the facility.

Table 3.2.10: The class “Report” contains aspects of methods and functions based on the “Report” table.

Class: Report	The role of class “Report” aids in generating different report types for users of the OPF website. This class is critical to users such as an administrator user role and can be used as a tool for building health reports.
generate_ReportTimestamp( )	The function generates a formatted timestamp for each report item. Once the user has submitted a report, OPF will obtain the exact time that it was submitted and it will be recorded in the database. The timestamp will be displayed for the facilities to help sort reports from oldest to newest.
generate_ReportBuilding( )	The function generates the building report for distinguishing different buildings. The dormitory students live in different buildings, and each building has a different health condition. The building reports, as a result, create a way to summarize the maintenance requests provided by the students per building. This would allow administrators to determine actions to take towards the dormitory buildings based on the building’s health condition.

Table 3.2.11: The class “Timestamp” contains aspects of methods and functions based on the “Timestamp” table.

Class: Timestamp	The class “Timestamp” provides the formatting of an object using second-time values. The class provides formatted options of clock services such as formatted date, formatted time, and combined formatted datetime. These timestamps are used to aggregate and create time data.
generate_TimeStamp( )	This function generates a timestamp composed of the previous data. The timestamp would be created with the use of the Date object class to generate a timestamp, based in real-time. This function would be the root for all the timestamp-related functions as this generates the timestamps that would be used to help organize data and display the time certain data was created.

Table 3.2.12: The class “Rich Data” contains aspects of methods and functions based on the “Rich Data” table.

Class: Rich Data	The class “Rich Data” is used for formatting images uploaded by the user including height and width. The class provides functions used to modify and read image data to be used for

	visual processing.
set_ImageModifier(filter)	This function sets the image modifier for an image uploaded by a user. The filter contains the data about the image from when the user created it. Using the filter, the function will add the filter to the image, so the image will be similar to the image's original state (a.k.a. when the user created the image). This image (with the filter) would then be used for visual processing in the backend.

3.3. Database Tables

The database consists of very large amounts of data that will need to store records efficiently for the OPF web application. A structured database will allow Team 05 to facilitate the storage, retrieval, modifications, and deletion of data in conjunction with various data processing operations. Additionally, the benefits of a well-structured database account for reduced data redundancy, increased consistency, and overall reduced retrieval costs of data. Figure 3.3.1, LucidChart was used to create cardinality notation and was used to convey relationships between database tables. Lastly, Figure 3.3.2, outlines the overall structure of the OPF web application's database and its relationships.

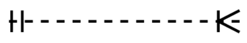


Figure 3.3.1: The left-symbol of the figure conveys a one (and only one) relationship and the right-symbol of the figure conveys a one or many relationship.

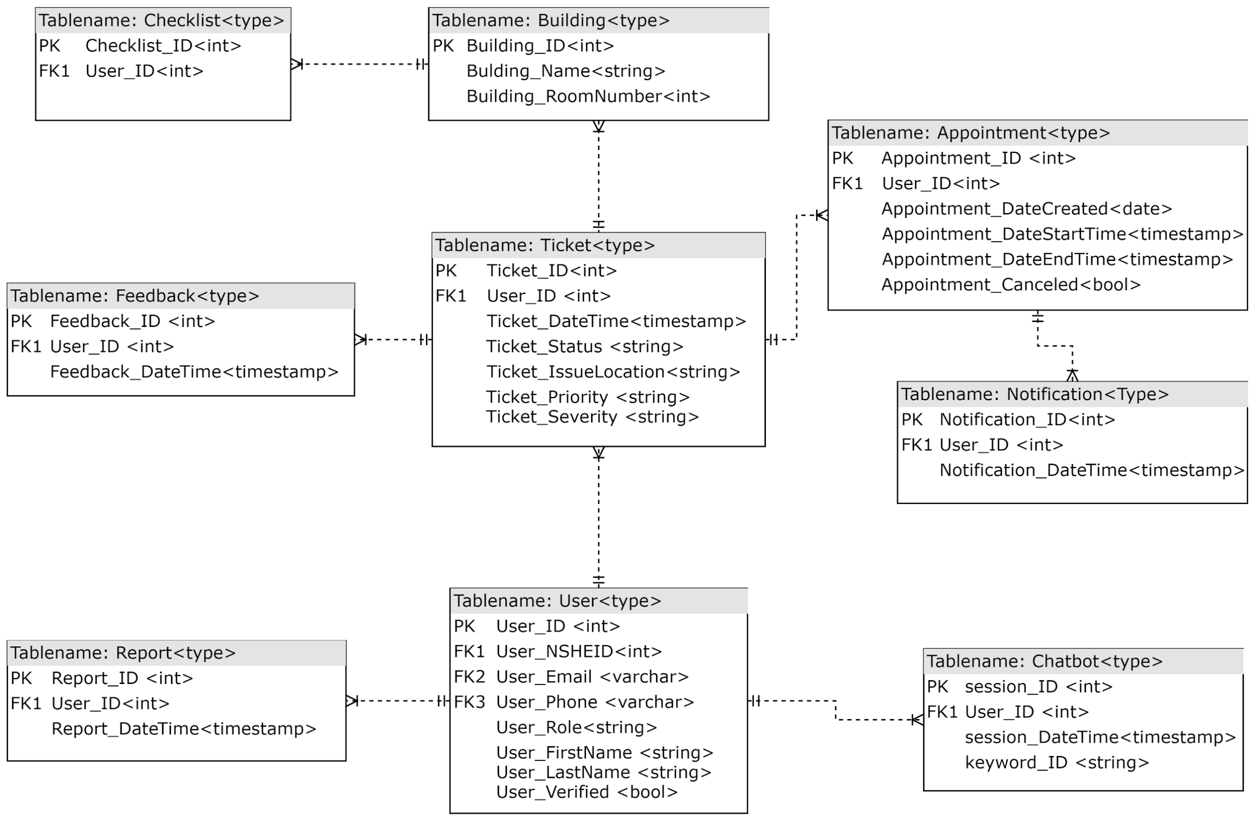


Fig. 3.3.2: The overview of the OPF web application database management system and its interrelationships between the system.

All tables in Section 3.3 detail the database structures that are used in the OPF web application. The database tables consist of attributes i.e. columns and a primary key. Moreover, a primary key is a specific choice of a minimal set of attributes that uniquely specifies a row in the table. The use of keys in the database allows Team 05 to compare, sort, store, and create relationships between the OPF data.

Table 3.3.1: The "User" table stores the login credentials into the OPF database. The user table consists of information that will be used to authenticate and verify users on the OPF web application.

User			
Key	Name <Type>	Description	Example
Primary Key	**User_ID <int>**	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a primary key in the user table.	'01553212'

▨	User_FirstName <string>	The first name of the user for the OPF web application. The first name of the user will be stored as a type string in the user table.	'Joan'
▨	User_LastName <string>	The last name of the user for the OPF web application. The last name of the user will be stored as a type string in the user table.	'Lopez'
Foreign Key	User_NSHEID <integer>	The NSHE ID number of the user for the OPF web application. The number is a 10-digit unique identifier type integer and is identified as a foreign key in the user table.	'8000123456'
Foreign Key	User_Email <varchar>	The email of the user's choice for the OPF web application can be used to authenticate an account and various communications. The email will be stored as a variable character of variable length and is identified as a foreign key in the user table.	'joanlopez@ nevada.edu'
Foreign Key	User_Phone <varchar>	The user phone number for the OPF web application to contact and authenticate accounts. The number will be stored as a variable character of variable length and is identified as a foreign key in the user table.	'1231231234'
▨	User_Role <string>	The user role for the OPF web application. The user role will be stored in the user database table as a type string.	'student'
▨	User_Verified <bool>	The user has verified their email address to use their account on OPF. The type is a boolean with either True/ False values in the user table.	'true'

Table 3.3.2: The “Ticket” table stores the relevant data when a user creates a maintenance ticket. The ticket table consists of information that will be used to view statuses of maintenance tickets and information pertaining to the maintenance ticket on the OPF web application.

Ticket			
Key	Name <Type>	Description	Example
Primary Key	**Ticket_ID <int>**	The ticket identification number to differentiate between all maintenance tickets created on the OPF website. The type will be stored as an integer as an 8-digit and is identified as a primary key in the ticket table.	‘34068239’
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in the ticket table.	‘01553212’
	Ticket_DateTime<timestamp>	The date and time the ticket was created for maintenance tickets. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	‘2021-10-16 12:35:08’
	Ticket_IssueLocation <string>	The maintenance issue is located in the dormitory of the user. The issue location will be stored in the ticket database table as a type string.	‘kitchen’
	Ticket_Priority <string>	A priority label of the maintenance issue for maintenance jobs. The ticket priority will be stored in the ticket database table as a type string.	‘high’
	Ticket_Severity <string>	A severity label of the maintenance issue for fixing maintenance jobs. The ticket severity will be stored in the ticket database table as a type string.	‘moderate’
	Ticket_Status <string>	The status of the ticket created by the user. The ticket status will be stored in the ticket database table as a type string.	‘closed’

Table 3.3.3: The "Appointment" table stores the relevant data when a user creates an appointment. The appointment table consists of information that will be used to create appointments and information pertaining to an appointment on the OPF web application.

Appointment			
Key	Name <Type>	Description	Example
Primary Key	**Appoinment_ID <int>**	The appointment identification number to differentiate between all appointments created on the OPF website. The type will be stored as an integer as an 8-digit and is identified as a primary key in the appointment table.	'15413724'
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in the appointment table.	'01553212'
	Appointment_DateCreated <Date>	The date the appointment was created for the user. The type will be stored as a date data type in the table in YYYY-MM-DD format.	'2021-11-16'
	Appointment_DateStartTime <timestamp>	The start time and date of the appointment are scheduled for the user. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	'2021-11-16 1:00:00'
	Appointment_DateEndTime <timestamp>	The date and end time of the appointment is scheduled for the user. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	'2021-11-16 2:00:00'
	Appointment_Canceled <bool>	The appointment was canceled by the user. The type is a boolean with either True/ False values in the appointment table.	'false'

Table 3.3.4: The "Notification" table stores the relevant data when a user is notified. The notification table consists of information that will be used to notify specific users and information on the OPF web application.

Notification			
Key	Name <Type>	Description	Example
Primary Key	**Notification_ID<int>**	The notification identification number to differentiate between the notifications sent to the user. The type will be stored as an integer as an 8-digit and is identified as a primary key in the notification table.	'72452539'
Foreign Key	User_ID<int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in our database table.	'01553212'
▨	Notification_DateTime <timestamp>	The notification date is the date the user was notified. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	'2021-10-16 11:55:10'

Table 3.3.5: The "Chatbox" table stores the relevant data when a user uses the chatbox on the OPF website.

Chatbox			
Key	Name <Type>	Description	Example
Primary Key	**session_ID<int>**	The identification to differentiate the chatbox sessions of users. The type will be stored as an integer as an 8-digit and is identified as a primary key in the chatbox table.	'42415280'
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is	'01553212'

		identified as a foreign key in our database table.	
▨	session_DateTime<timestamp>	The chatbox session date of the user and the system on the OPF application. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	'2021-11-12 10:32:07'
▨	keyword_ID<string>	The chatbox keywords are used to store specific words that the user may use in a session such as a specific word or a phrase. The keyword identifier will be stored as a type string.	'status'

Table 3.3.6: The "Checklist" table stores the relevant data for multiple templates on the OPF website. These templates use specific questions that are then stored in the database.

Checklist			
Key	Name <Type>	Description	Example
Primary Key	**Checklist_ID <int>**	The identification to differentiate between all checklists of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a primary key in the checklist table.	'99083896'
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in our database table.	'01553212'

Table 3.3.7: The “Building" table stores the relevant data for multiple buildings at the University of Nevada, Reno. The building table will contain information that will be used in the appointment and ticketing system.

Building			
Key	Name <Type>	Description	Example
Primary Key	**Building_ID <int>**	The identification to differentiate between all buildings at the University of Nevada, Reno. The type will be stored as an integer as an 8-digit and is identified as a primary key in the building table.	‘10000011’
▨	Building_Name <string>	The different building names at the University of Nevada, Reno. The name will be stored in the building table as a type string.	‘Uncommon’
▨	Building_RoomNumber <int>	The different building names at the University of Nevada, Reno. The room number will be stored in the building table as an integer type.	‘12’

Table 3.3.8: The “Feedback" table stores the relevant data for feedback on the OPF web application.

Feedback			
Key	Name <Type>	Description	Example
Primary Key	**Feedback_ID <int>**	The identification to differentiate between the feedback of users on the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a primary key in the feedback table.	‘57666344’
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in our database table.	‘01553212’
▨	Feedback_DateTime	The feedback date is the date and the	‘2021-11-16

▨	<timestamp>	time the feedback was given by the user. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	11:14:07’

Table 3.3.9: The “Report”' table stores the relevant data for reports on the OPF web application.

Report			
Key	Name <Type>	Description	Example
Primary Key	**Report_ID <int>**	The identification to differentiate between reports on the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a primary key in the report table.	‘10610249’
Foreign Key	User_ID <int>	The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as an 8-digit and is identified as a foreign key in our database table.	‘01553212’
▨	Report_DateTime <timestamp>	The report time is the date and time the report was given by the user. The type will be stored as a timestamp data type in the table as YYYY-MM-DD 00:00:00 format.	‘2021-11-16 09:30:01’

4. Detailed Design

The detailed design focuses on using behavioral diagrams to explain the processes of specific functionalities for OPF. There are four activity diagrams and one state diagram. The activity diagrams demonstrate the actions being performed, and the state charts demonstrate the states the object may have.

4.1. State Charts

Figure 4.1.1 demonstrates a state chart for the appointment set-up process. The initial state would be when the dormitory student has successfully submitted their maintenance request. The request would then follow a cycle until it is accepted by a member in Facilities and Management. When the request is accepted, the dormitory student would be notified of an appointment time.

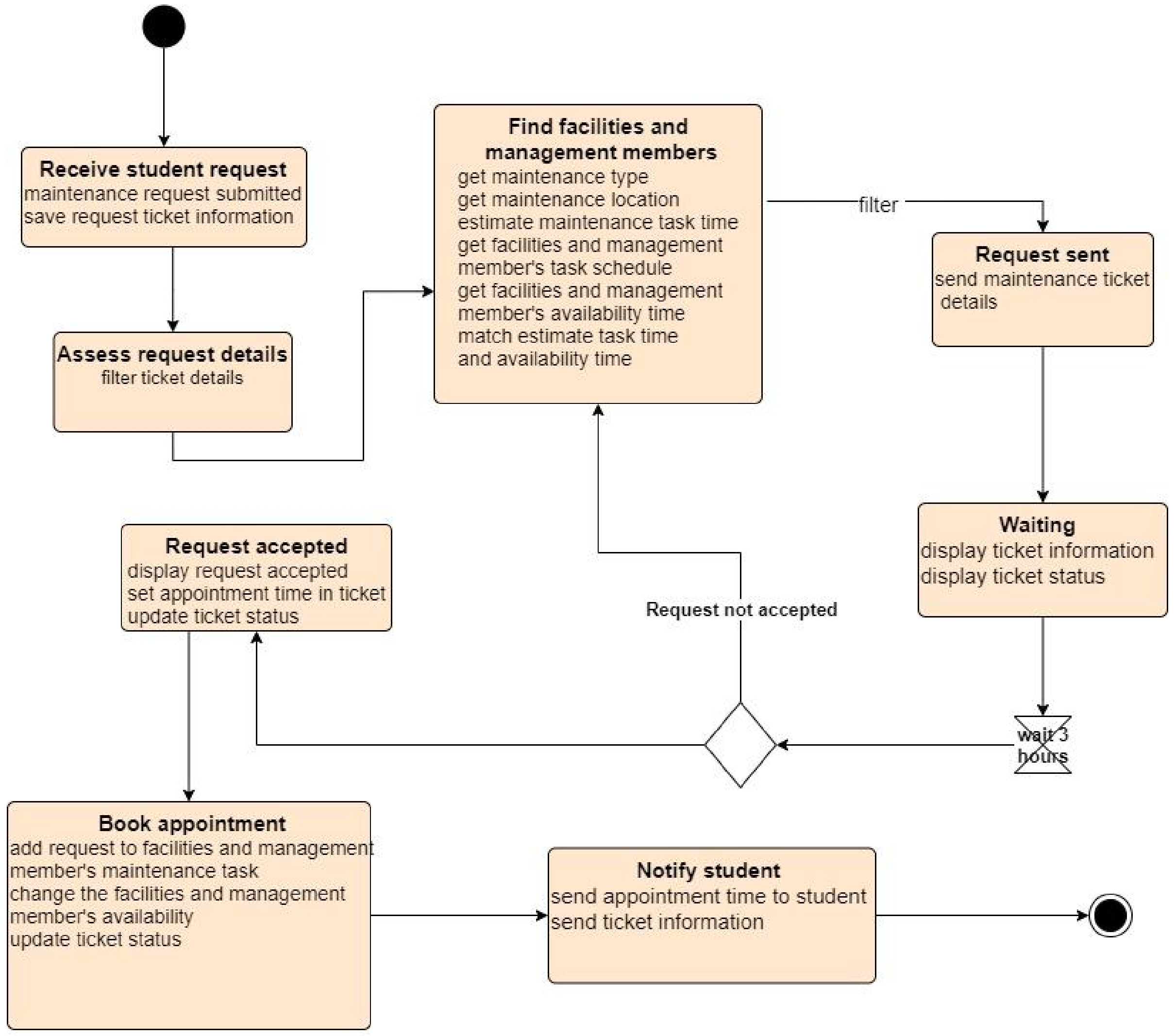
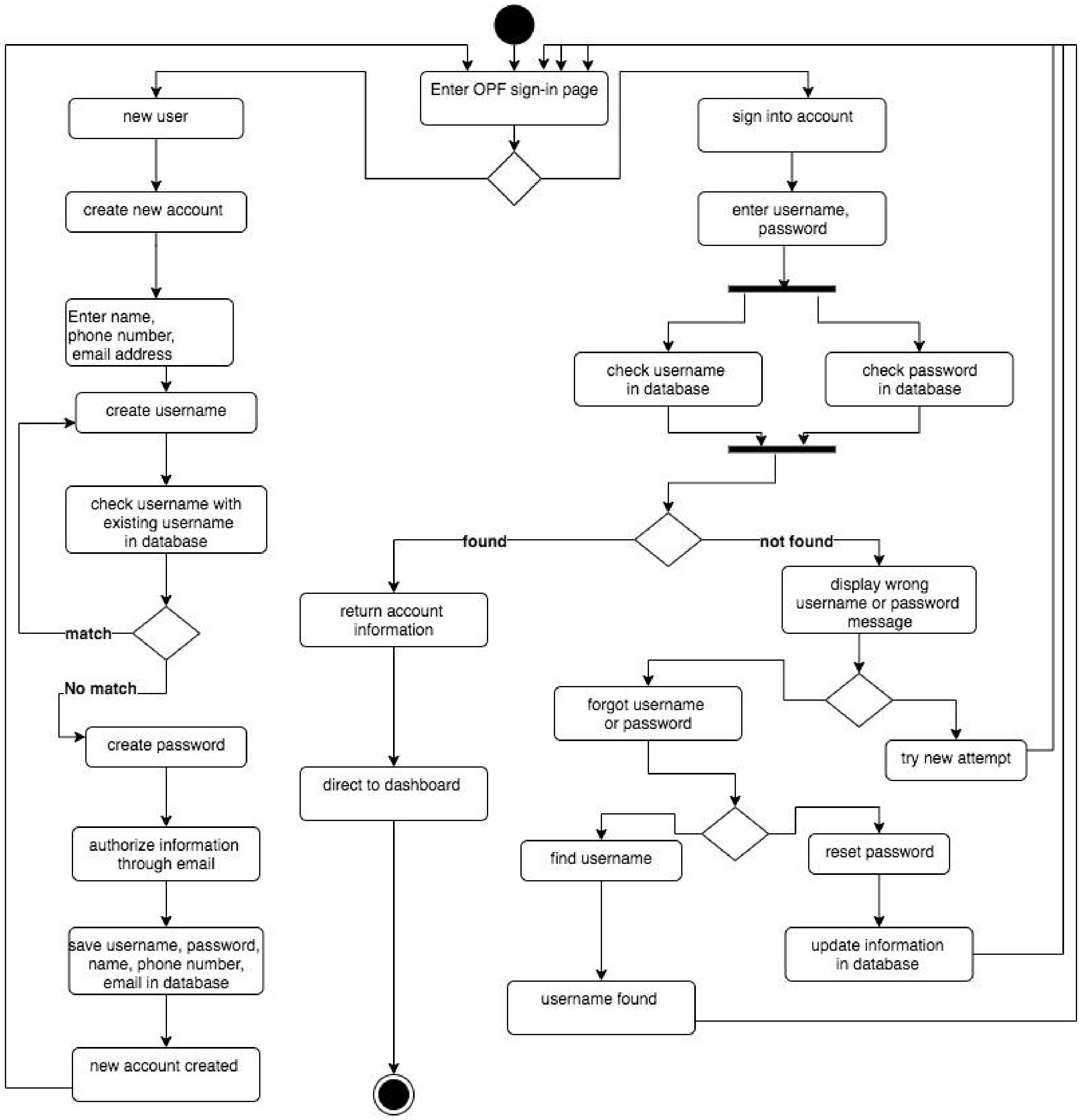

Receive student request
maintenance request submitted
save request ticket information
Assess request details
filter ticket details
Find facilities and management members
get maintenance type
get maintenance location
estimate maintenance task time
get facilities and management member's task schedule
get facilities and management member's availability time
match estimate task time and availability time
filter
Request sent
send maintenance ticket details
Waiting
display ticket information
display ticket status
wait 3 hours
Request not accepted
Request accepted
display request accepted
set appointment time in ticket
update ticket status
Book appointment
add request to facilities and management member's maintenance task
change the facilities and management member's availability
update ticket status
Notify student
send appointment time to student
send ticket information

Fig. 4.1.1: The appointment state chart for the appointment setup process for the members of the facilities and management.

4.2. Activity Diagrams

Figure 4.2.1 displays an activity diagram for the sign-in and authorization process. The diagram represents a user attempting to sign into their OPF account.



Enter OPF sign-in page
new user
sign into account
create new account
enter username, password
Enter name, phone number, email address
check username in database
check password in database
create username
check username with existing username in database
found
not found
return account information
display wrong username or password message
match
No match
forgot username or password
create password
try new attempt
direct to dashboard
authorize information through email
find username
reset password
save username, password, name, phone number, email in database
update information in database
username found
new account created

Fig. 4.2.1: The user sign-in and authorization activity diagram demonstrating how OPF processes the sign-in procedure.

Figure 4.2.2 contains an activity diagram for a dormitory student issuing a maintenance request. The dormitory student would have to enter the maintenance tab and select “Make new request”. Afterward, the dormitory student would have to fill out information regarding their issue and then submit the request. Through the usage of AI, the request would be categorized by priority, and the accumulated information would be sent to Facilities and Management. Similarly, the dormitory student would be able to see the status of their request.

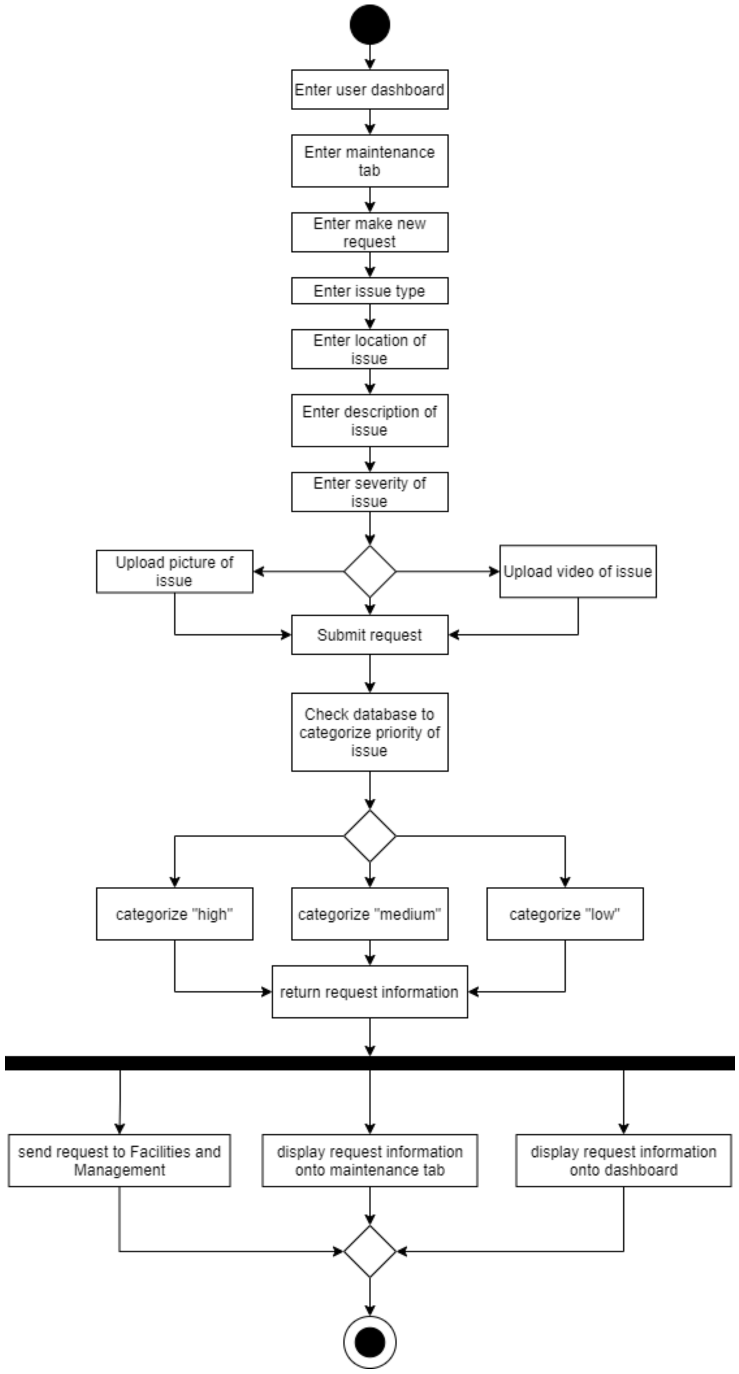
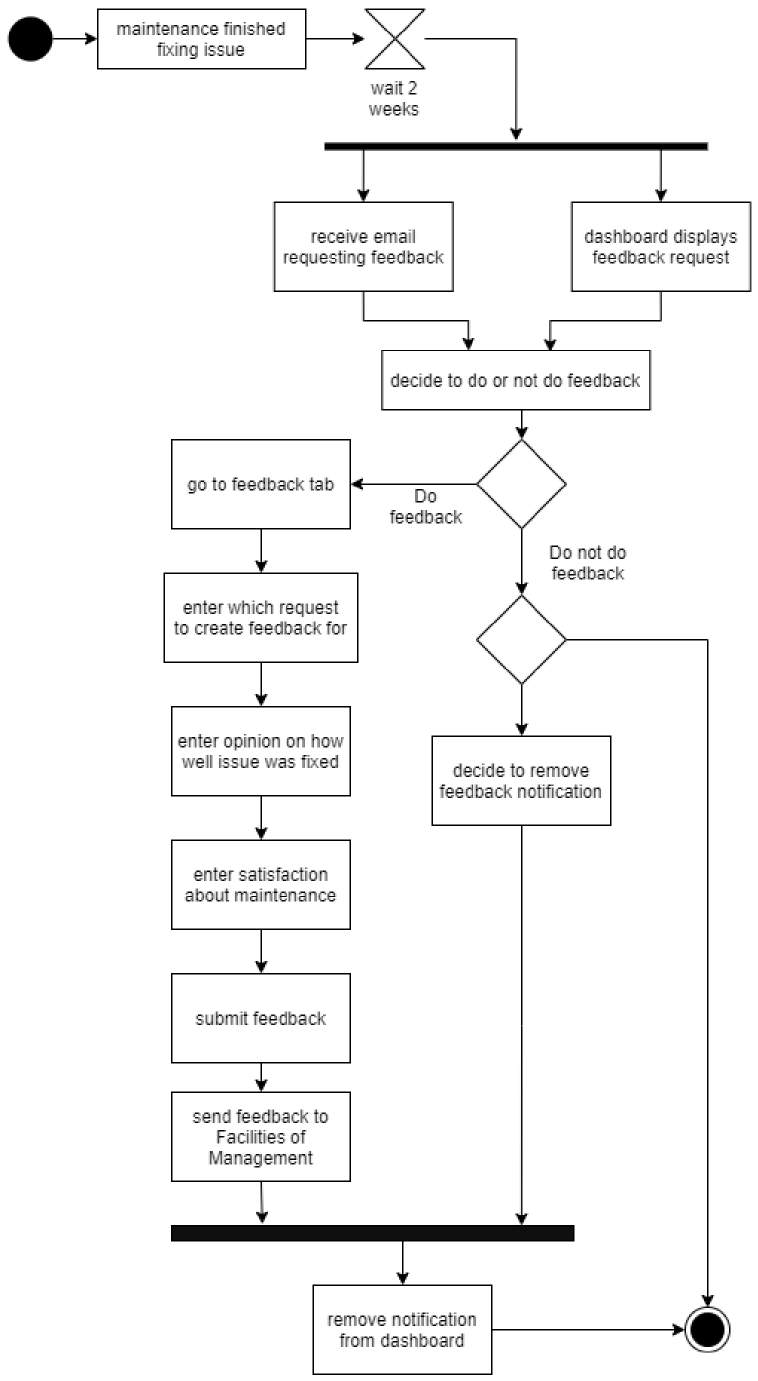


Fig. 4.2.2: The maintenance request activity diagram demonstrates the process of submitting a maintenance request through the OPF web application.

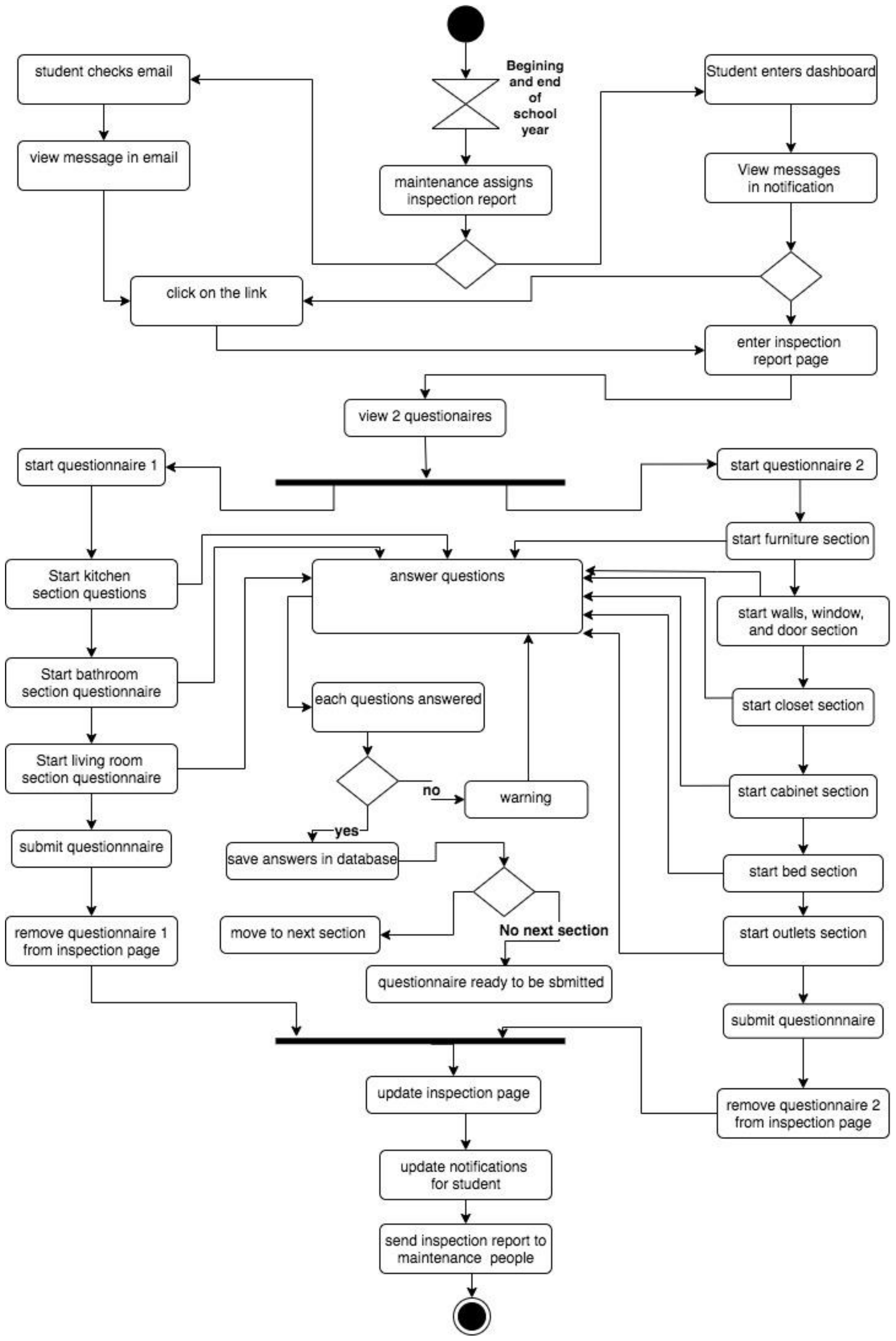
Figure 4.2.3 contains the activity diagram for creating feedback for a completed maintenance request. The dormitory student that issued a maintenance request would be able to answer a feedback form two weeks after the request was completed. If the user decides to fill the feedback form, the user would answer three questions and submit the form. If not, the dormitory student has the option to ignore the request or remove the notification.



maintenance finished fixing issue
wait 2 weeks
receive email requesting feedback
dashboard displays feedback request
decide to do or not do feedback
go to feedback tab
Do feedback
Do not do feedback
enter which request to create feedback for
enter opinion on how well issue was fixed
decide to remove feedback notification
enter satisfaction about maintenance
submit feedback
send feedback to Facilities of Management
remove notification from dashboard

Fig. 4.2.3: The feedback activity diagram demonstrating the process of creating the feedback through OPF.

Figure 4.2.4 demonstrates an activity diagram for filling an inspection report. The inspection report has to be completed when the dormitory student moves into the dormitory and when the dormitory student leaves the dormitory. The dormitory student has to fill out two forms and answer questions regarding the topic of the form.



Begining and end of school year
maintenance assigns inspection report
student checks email
view message in email
Student enters dashboard
View messages in notification
click on the link
enter inspection report page
view 2 questionaires
start questionnaire 1
start questionnaire 2
Start kitchen section questions
Start bathroom section questionnaire
Start living room section questionnaire
submit questionnaire
remove questionnaire 1 from inspection page
answer questions
each questions answered
no
warning
yes
save answers in database
move to next section
No next section
questionnaire ready to be sbmitted
start furniture section
start walls, window, and door section
start closet section
start cabinet section
start bed section
start outlets section
submit questionnnaire
remove questionnaire 2 from inspection page
update inspection page
update notifications for student
send inspection report to maintenance people

Fig. 4.2.4: The inspection report activity diagram displays the process for filing the inspection reports for the dormitory.

5. Initial Hardware Design

The OPF web application doesn't have any hardware components at this time.

6. User Interface Design

The user interface design demonstrates the user interfaces which include maintenance reports, alert boxes, chat boxes, and data diagrams. All figures in Section 6 display 12 snapshots of the OPF user interface with brief descriptions regarding each figure. They show how the users will be able to interact with OPF and the user interfaces are designed to be simple, interactive, useful, elegant, and accessible. All 12 snapshots are created in AdobeXD.

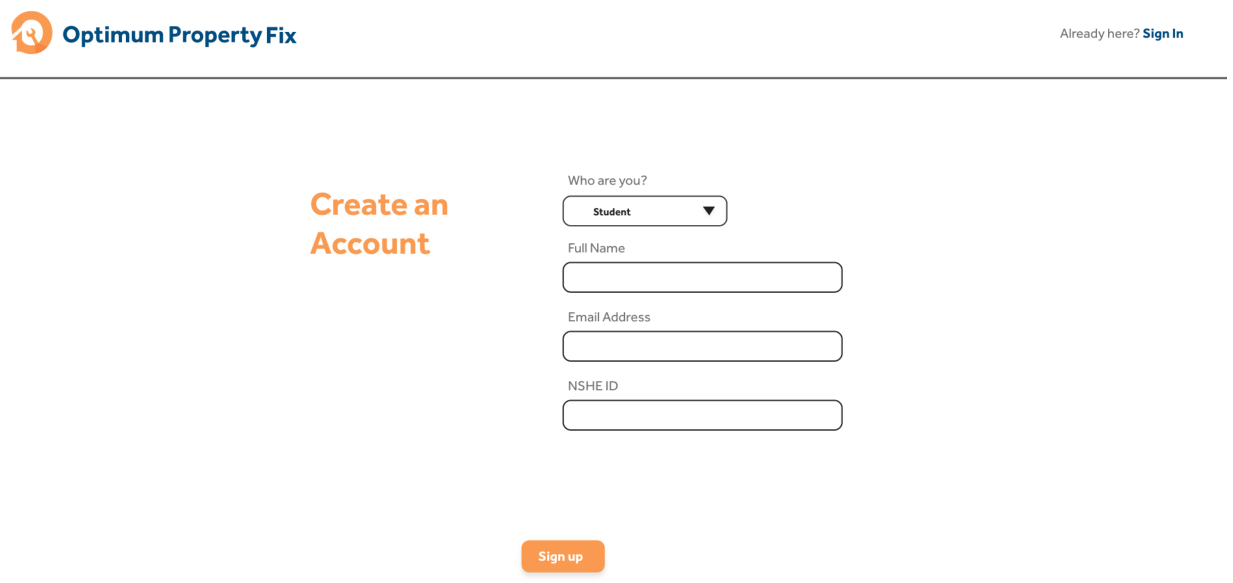


Fig. 6.1: The snapshot shows the page where the user can select who they are such as student or admin and then proceed to make an account. They will input information such as their full name, their email address, and their NSHE ID. On the top right corner, if the user already has an account, they may proceed to sign in

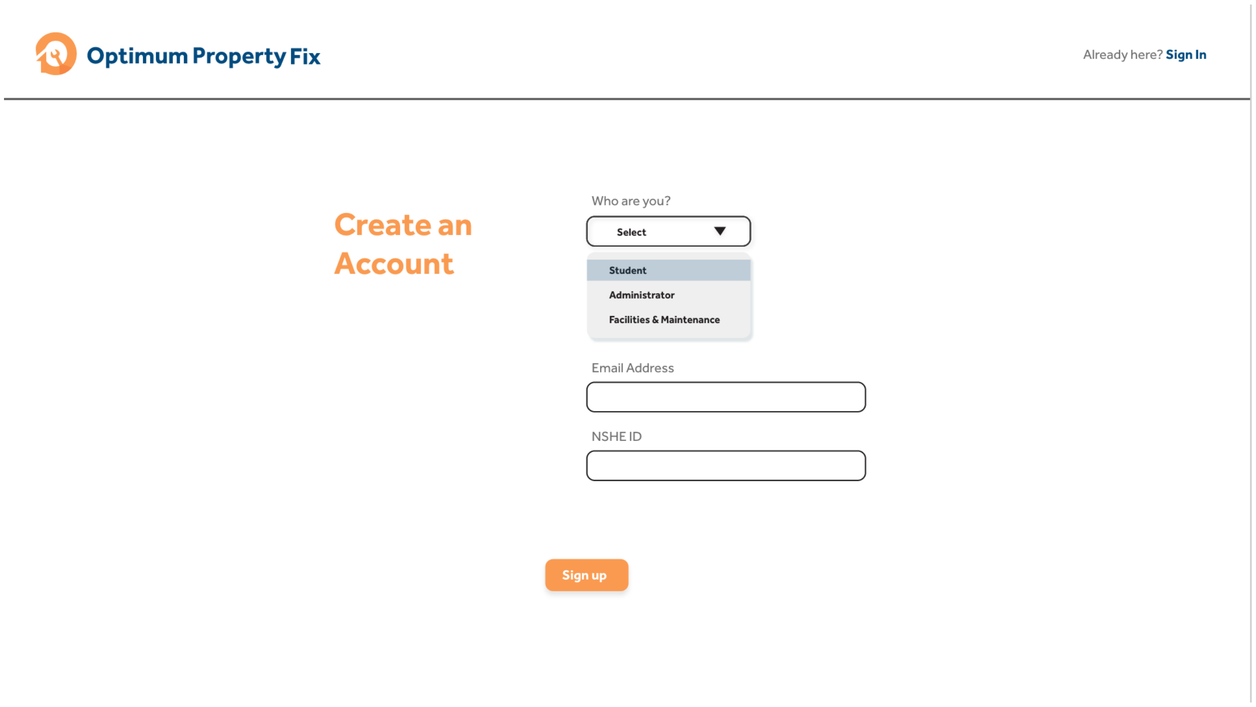


Fig. 6.2: The login page of OPF displays a drop-down menu for the user to select their role. Their role can be either a Student, Administrator, or Facilities and Management. Furthermore, the role will assign a different dashboard view for the different users.

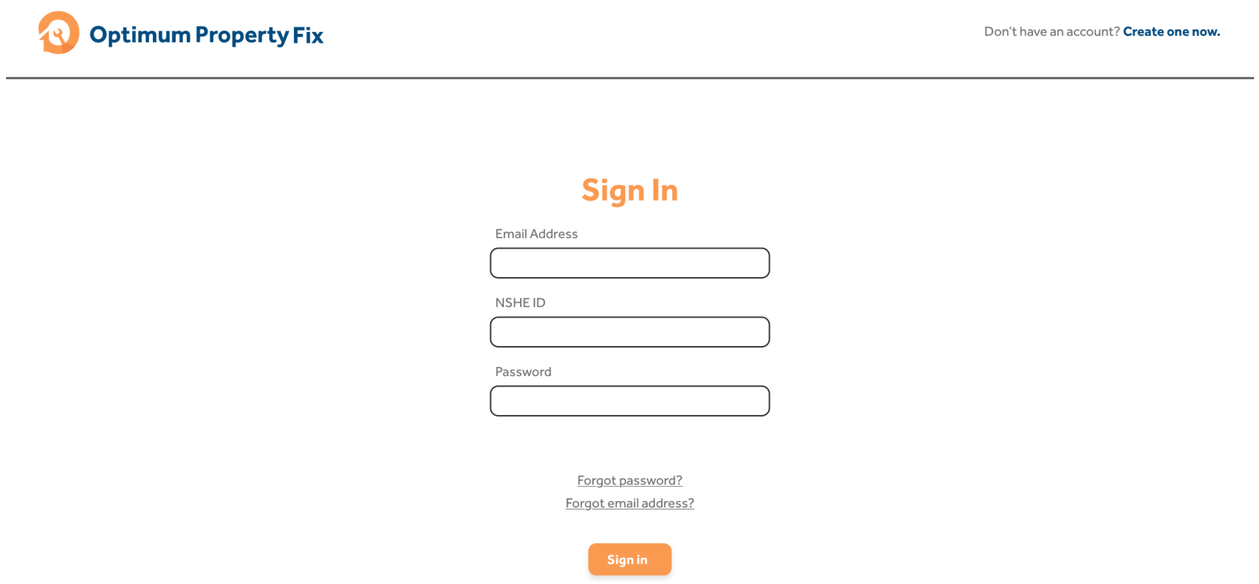
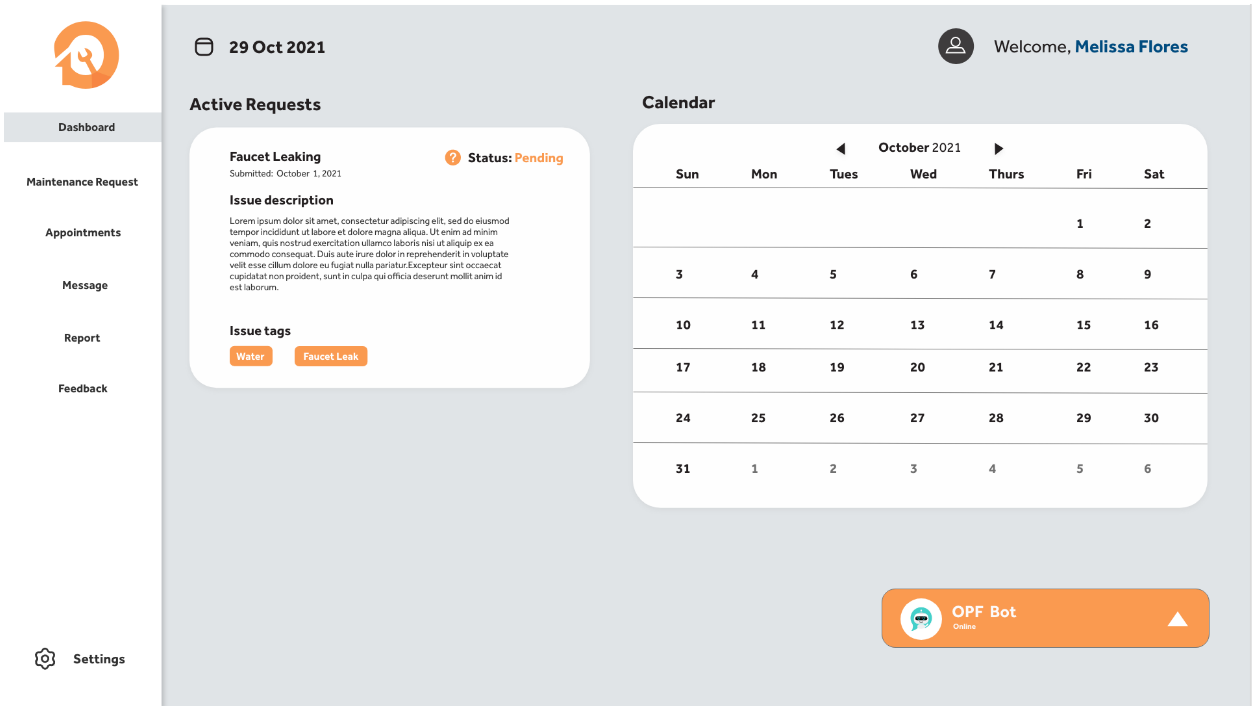


Fig. 6.3: The sign-in page of OPF displays where the user is able to sign in to the web application. Here, the user inputs their email address, NSHE ID, and password. There is also an option that provides the user an option to recover their password or email address if forgotten. On the top right, it shows a suggestion if a user does not already have an account with OPF.



29 Oct 2021

Welcome, Melissa Flores

Active Requests

Faucet Leaking
Submitted: October 1, 2021
Status: Pending

Issue description

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Issue tags

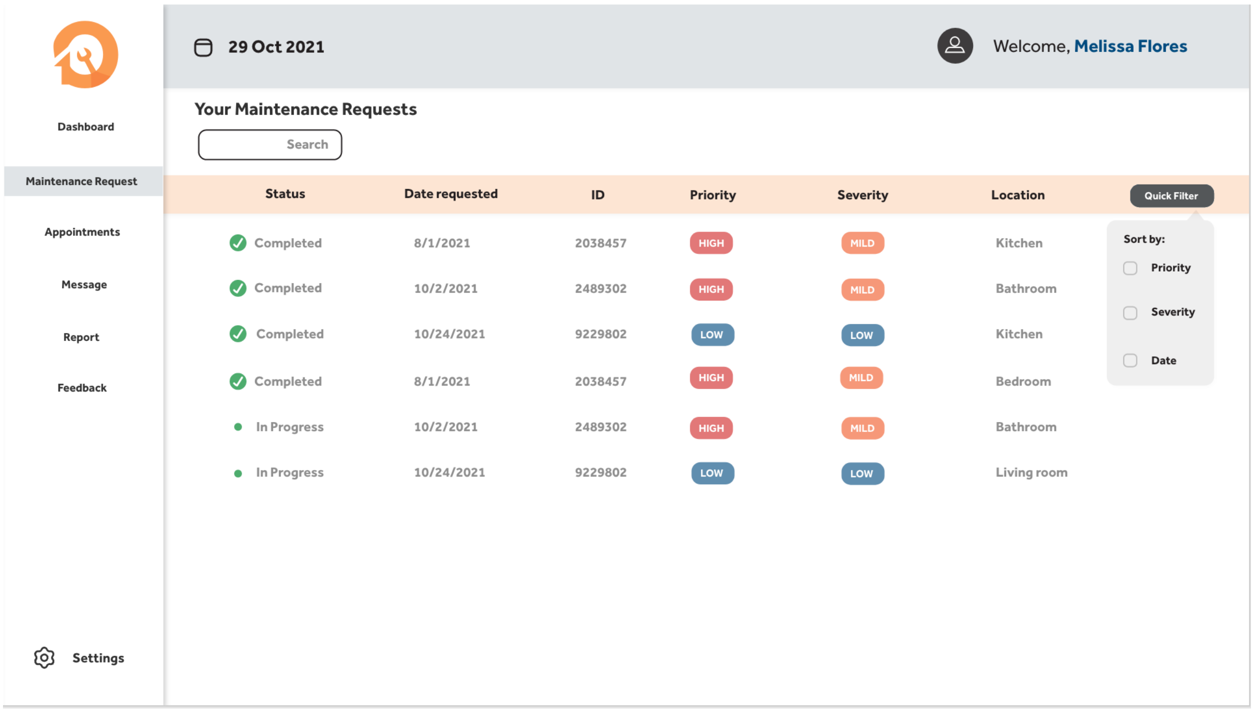
Water Faucet Leak

Calendar

October 2021

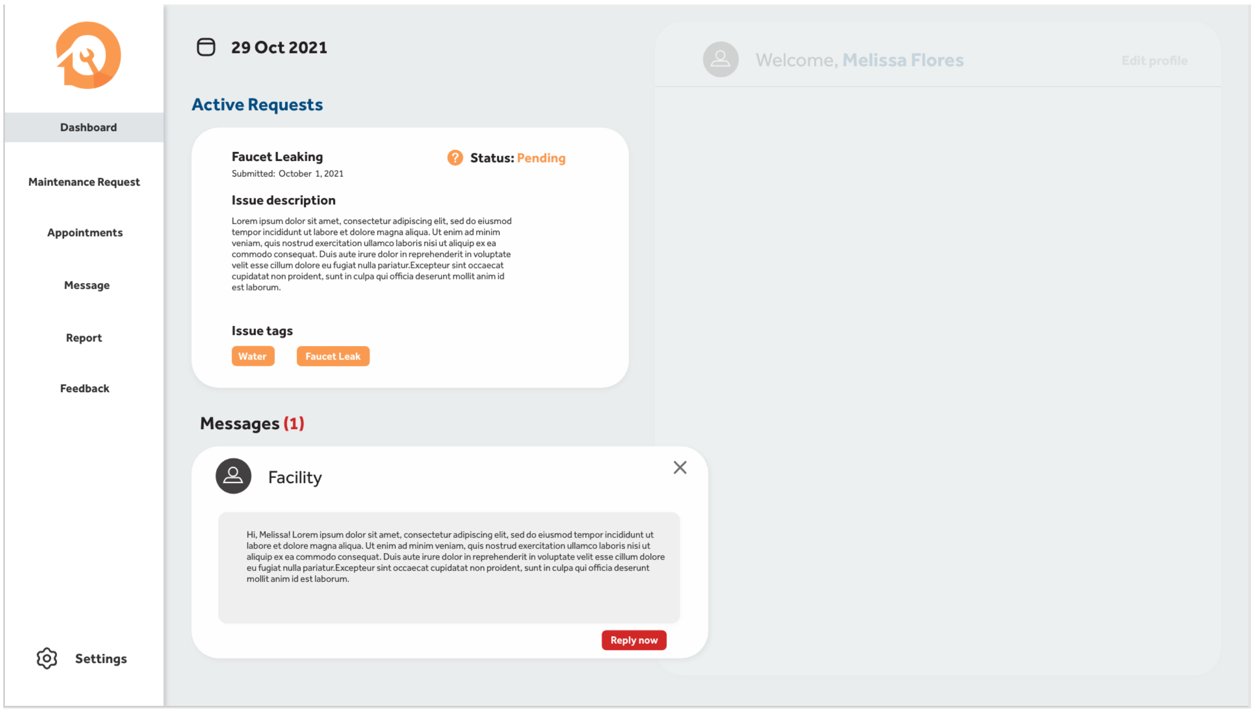
Sun	Mon	Tues	Wed	Thurs	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Fig. 6.4: The dashboard page of the student user shows the calendar in which you can keep track of appointments and maintenance reports. The bottom right shows the chatbox where you will be able to continuously access communication with the OPF bot.



Status	Date requested	ID	Priority	Severity	Location
✓ Completed	8/1/2021	2038457	HIGH	MILD	Kitchen
✓ Completed	10/2/2021	2489302	HIGH	MILD	Bathroom
✓ Completed	10/24/2021	9229802	LOW	LOW	Kitchen
✓ Completed	8/1/2021	2038457	HIGH	MILD	Bedroom
● In Progress	10/2/2021	2489302	HIGH	MILD	Bathroom
● In Progress	10/24/2021	9229802	LOW	LOW	Living room

Fig. 6.5: The maintenance request page for the student shows a list of completed and in-progress maintenance reports. The student is able to click on the quick filter button which will open a drop-down menu allowing the user to sort and filter the reports according to their priority, severity, and date.



29 Oct 2021

Welcome, Melissa Flores

Active Requests

Faucet Leaking

Submitted: October 1, 2021

Status: Pending

Issue description

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Issue tags

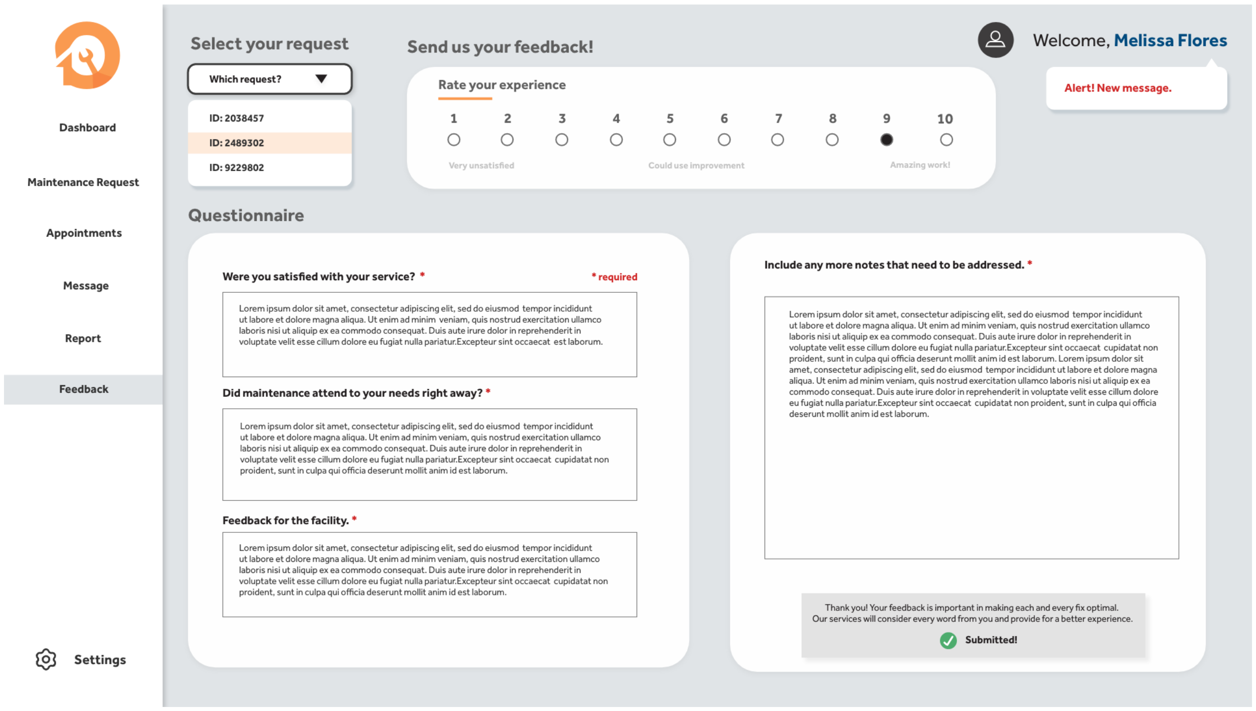
Water Faucet Leak

Messages (1)

Facility

Hi, Melissa! Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Fig. 6.6: The dashboard page shows a pop-up of an important message alert. When this happens, the majority of the page will fall back so that it will place more of an emphasis on the message that needs attention.



Select your request

Which request?

ID: 2038457

ID: 2489302

ID: 9229802

Send us your feedback!

Rate your experience

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	●	○

Very unsatisfied — Could use improvement — Amazing work!

Questionnaire

Were you satisfied with your service? * — * required

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat est laborum.

Did maintenance attend to your needs right away? *

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Feedback for the facility. *

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

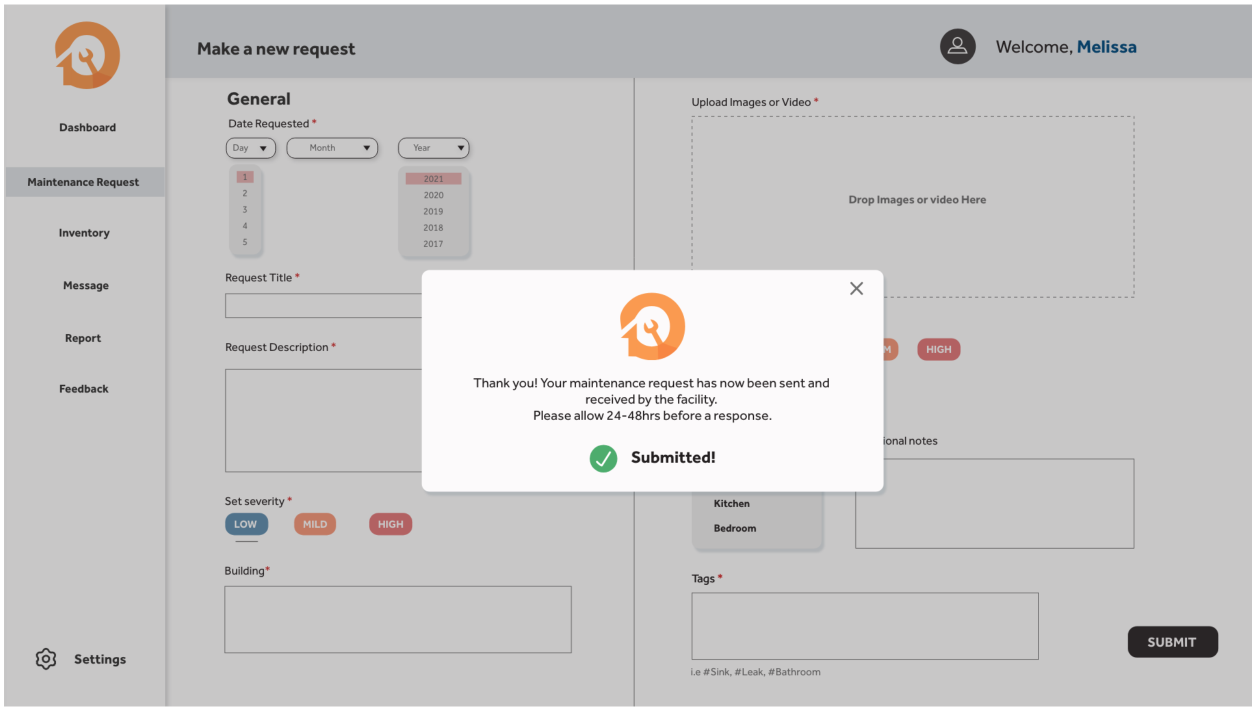
Include any more notes that need to be addressed. *

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Thank you! Your feedback is important in making each and every fix optimal.
Our services will consider every word from you and provide for a better experience.

✓ Submitted!

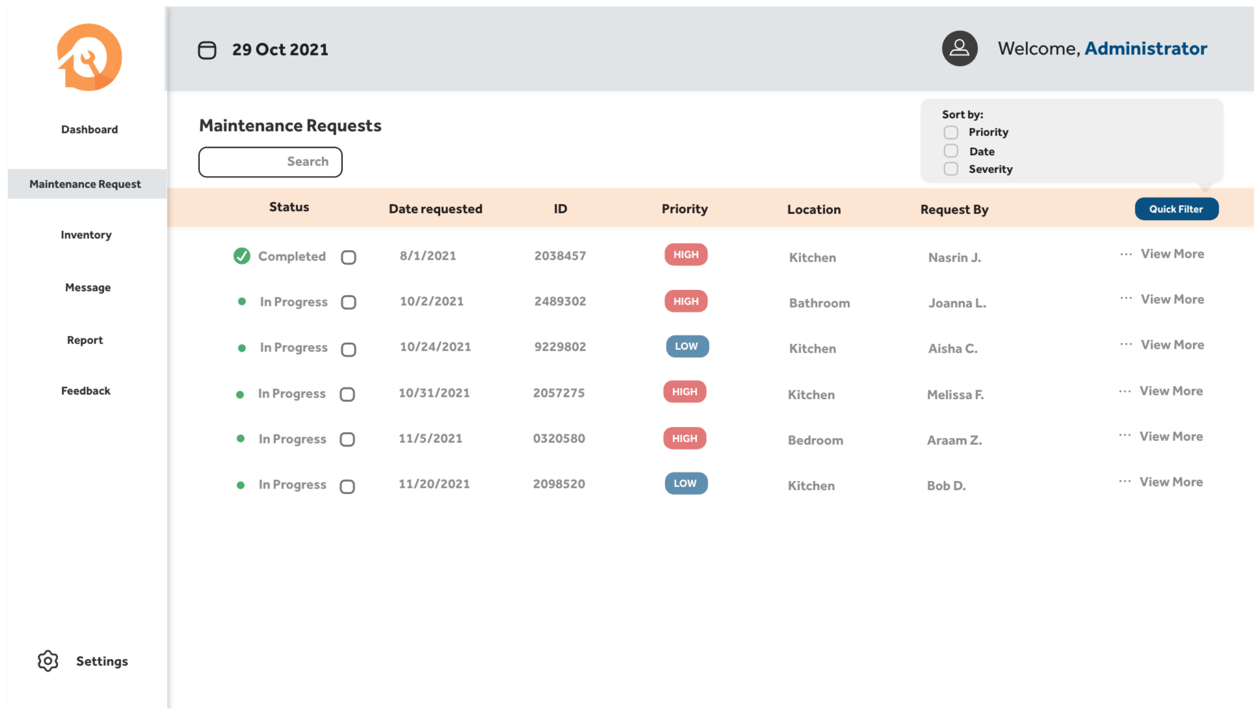
Fig. 6.7: The feedback page of the student user shows the feedback form they will fill out after a maintenance request has been completed. The student is able to select which request they'd like to provide feedback for, with the drop-down selection. Next, they are able to rate their experience from 1 to 10, fill out the questionnaire, and be notified that the form has been submitted.



Thank you! Your maintenance request has now been sent and received by the facility.
Please allow 24-48hrs before a response.

Submitted!

Fig. 6.8: The maintenance request page after a user has submitted their request. A pop-up alert box will block out the rest of the screen. It will tell the user that the request has been successfully submitted and is sent to the facility.



29 Oct 2021

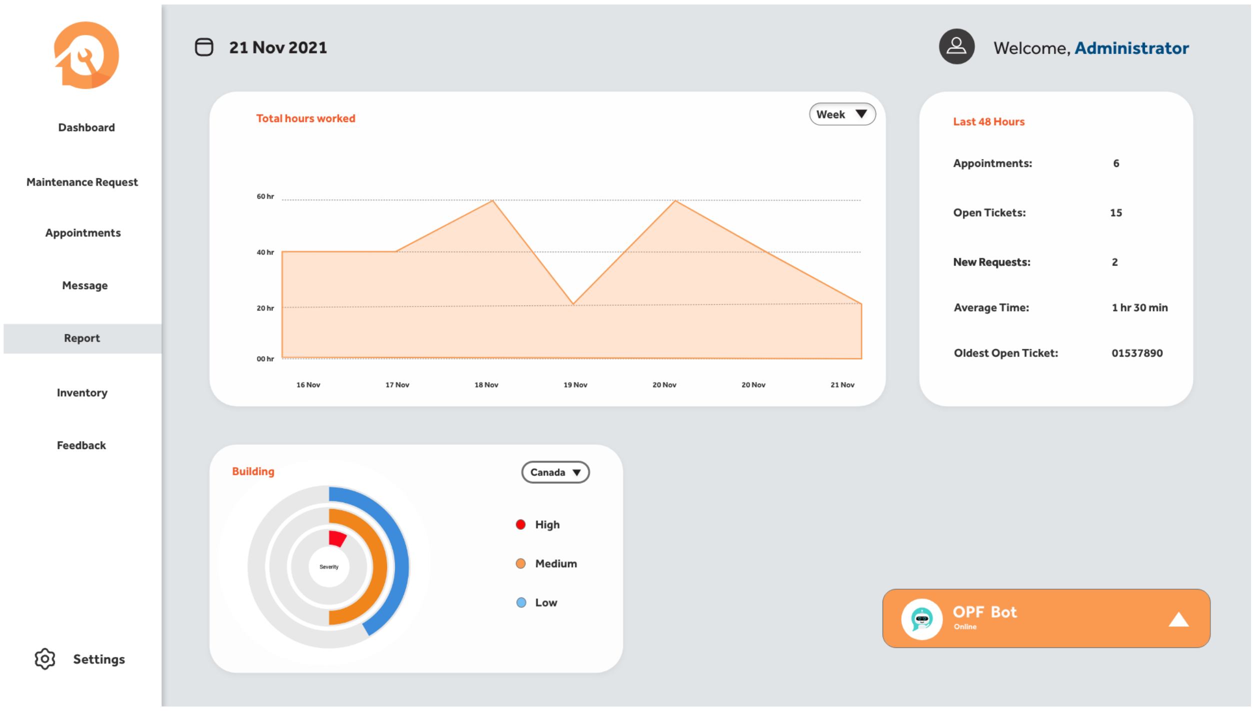
Welcome, Administrator

Maintenance Requests

Sort by:
- Priority
- Date
- Severity

Status	Date requested	ID	Priority	Location	Request By	
✓ Completed	8/1/2021	2038457	HIGH	Kitchen	Nasrin J.	··· View More
● In Progress	10/2/2021	2489302	HIGH	Bathroom	Joanna L.	··· View More
● In Progress	10/24/2021	9229802	LOW	Kitchen	Aisha C.	··· View More
● In Progress	10/31/2021	2057275	HIGH	Kitchen	Melissa F.	··· View More
● In Progress	11/5/2021	0320580	HIGH	Bedroom	Araam Z.	··· View More
● In Progress	11/20/2021	2098520	LOW	Kitchen	Bob D.	··· View More

Fig. 6.9: The maintenance request page for the administrator shows a list of completed and in-progress maintenance reports. The administrator is able to click on the quick filter button and a drop-down menu will appear. This will allow the user to sort and filter the reports according to their priority, severity, and date. Here, the administrator is also able to see who requested the report and to also view more details.



Last 48 Hours

Appointments:	6
Open Tickets:	15
New Requests:	2
Average Time:	1 hr 30 min
Oldest Open Ticket:	01537890

Fig. 6.10: The report page of the administrator user shows a summary of the data collected from reports. The graph displays hours worked daily. On the right, data is collected from the past 48 hours, showing appointments, tickets, requests, time, and oldest tickets. The radial graph displays severity according to the building.

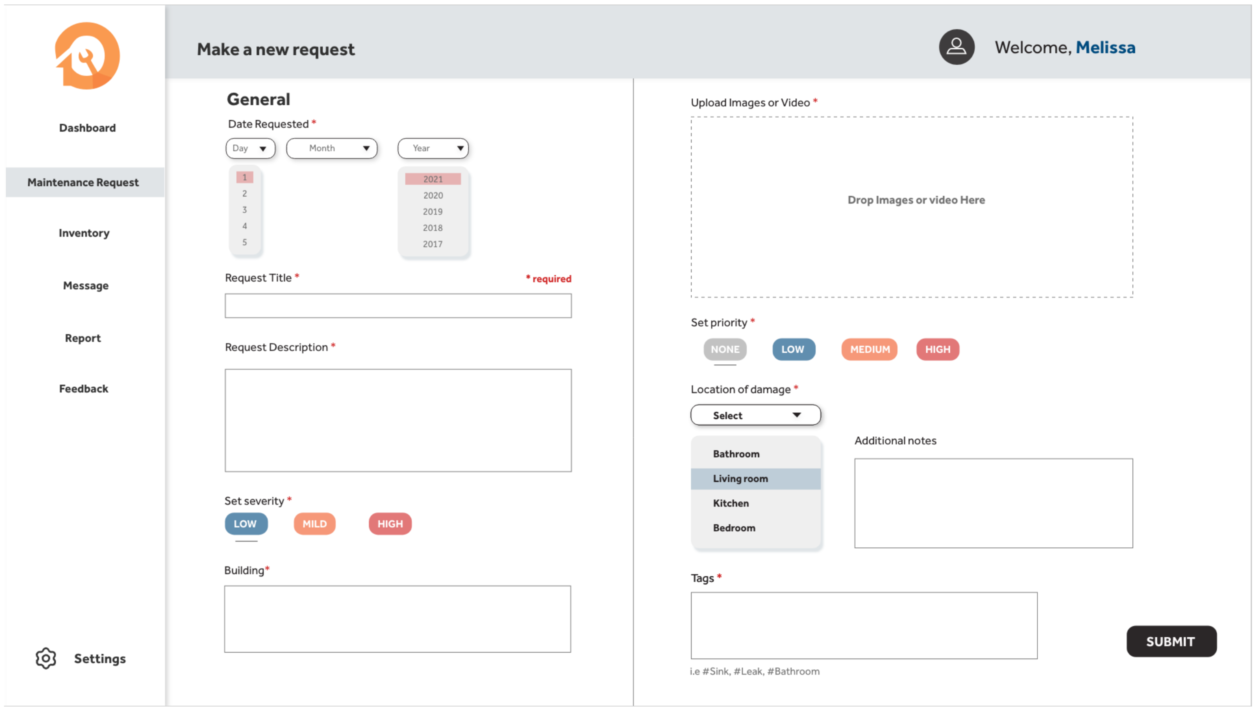
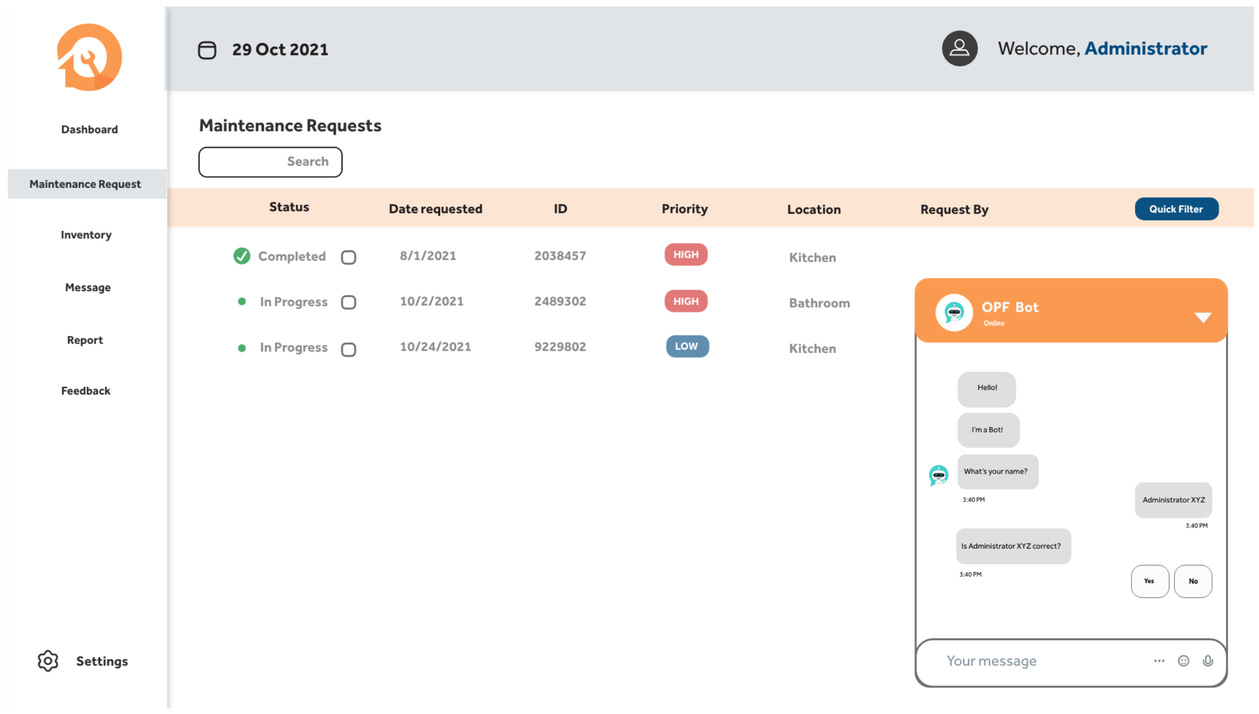


Fig. 6.11: The OPF web page is the desktop version of the maintenance request creation page. Users would be able to assign general information to a maintenance request ticket such as a description, severity level, and image.


29 Oct 2021

Welcome, Administrator

Dashboard

Maintenance Request

Inventory

Message

Report

Feedback

Settings

Maintenance Requests

Search

Quick Filter

Status	Date requested	ID	Priority	Location	Request By
✓ Completed	8/1/2021	2038457	HIGH	Kitchen	
● In Progress	10/2/2021	2489302	HIGH	Bathroom	
● In Progress	10/24/2021	9229802	LOW	Kitchen	

OPF Bot

Online

Hello!

I'm a Bot!

What's your name?

3:40 PM

Administrator XYZ

3:40 PM

Is Administrator XYZ correct?

3:40 PM

Yes

No

Your message

Fig. 6.12: The OPF web page displays a chatbox function for a user on the bottom right of the page. The user will be able to interact with the chatbox for multiple purposes including creating tickets, answering frequently asked questions, and checking the status of a maintenance request.

7. Version Control and Software Management System

The CS 425 senior capstone project, a public repository, on GitHub was created and the link is included here https://github.com/joannalopez223/UNR-Capstone.

8. Contributions of Team Members

8.1. Araam Zaremehrjardi's Contribution

Araam Zaremehrjardi's total time worked on the project design totals fifteen hours. Contribution entails writing the introduction, creating the program unit diagram, and ensuring consistent formatting throughout the document.

8.2. Joanna Lopez's Contribution

Joanna Lopez's total time worked on the project design totals fifteen hours. The contributions include aiding in the writing of the context diagram, database tables/figures, and program units.

8.3. Melissa Flores' Contribution

Melissa Flores's total time worked on the project design totals eight hours. The contributions include the creation of the user interface using Adobe XD and writing detailed descriptions for the program units.

8.4. Nasrin Juana's Contribution

Nasrin Juana's total time worked on the project design totals nine hours. The contribution includes the detailed design alongside Aisha Co and the writing of some of the program units.

8.5. Aisha Co's Contribution

Aisha Co's total time worked on the project design totals to five hours. The contribution includes the detailed design alongside Nasrin Juana, a couple of data tables in program units, and minor edits on the introduction.