

The University of Nevada, Reno
Department of Computer Science and Engineering
CS 426: Senior Projects in Computer Science
Project Part #2: Revised Specification and Design



Team 05

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

Instructor: David Feil-Seifer, Vihn Le, and, Devin Lee

Advisor: Erin Keith

February 18th, 2022

| | |
|--|-----------|
| 1.0 Abstract | 2 |
| 2.0 Recent Project Changes | 2 |
| 3.0 Updated Specifications | 2 |
| 3.1 Summary of Changes in Project Specifications | 2 |
| 3.2 Updated Technical Requirements Specification | 3 |
| 3.3 Updated Use Case Modeling | 6 |
| 4.0 Updated Design | 11 |
| 4.1 Summary of Changes in Project Design | 11 |
| 4.2 Updated High-Level and Medium-Level Design | 11 |
| 4.2.1 Context Diagram | 11 |
| 4.2.2 Programming Units | 12 |
| 4.2.3 Database Structure | 24 |
| 4.3 Updated Hardware Design | 29 |
| 4.4 Updated User Interface Design | 29 |
| 5.0 Updated Glossary of Terms | 41 |
| 6.0 Engineering Standards and/or Technologies | 43 |
| 7.0 Project Impact and Context Considerations | 45 |
| 8.0 Updated List of References | 46 |
| Problem Related Websites | 46 |
| Problem Domain Articles | 47 |
| Problem Domain Books | 47 |
| 9.0 Contributions of Team Members | 48 |
| Araam Zaremehrijardi's Contribution | 48 |
| Joanna Lopez's Contribution | 48 |
| Melissa Flores' Contribution | 48 |
| Nasrin Juana's Contribution | 48 |
| Aisha Co's Contribution | 48 |

1.0 Abstract

The current dormitory maintenance requests system at the University of Nevada, Reno (UNR) provides a system that is flawed and susceptible to complications. Some issues the current system faces are paper trails, leading to lost reports, a lack of maintenance request progress tracking for the students, and deters students from submitting issues that occur within the property. Moreover, Optimum Property Fix (OPF) aims to revamp UNR's current system through a new and improved maintenance request system, accessible to dormitory students as well as facilities and service members. OPF's functionality fully eliminates the paper-based system and replaces it with a web application that retains all maintenance records in one place.

2.0 Recent Project Changes

The progress made towards OPF would be the implementation of the student pages and the chatbot based on the user interface (UI) design developed through Adobe XD. The design follows the initial layout of the UI design as well as the color scheme to make the web application clear and uniform. Correspondingly, this allows the users to intuitively navigate OPF.

Next, the features that are changed include the user sign-in page, creating a new maintenance request page, dashboard page, view maintenance requests page, and feedback page for the students and the chatbot. The sign-in page no longer uses AWS Cognito which had multi-factor authentication and encryption. The create maintenance requests now have updated features such as submission buttons and other text areas that are cohesive throughout the web application. The feedback page has also made significant changes to the application taking advantage of dropdowns for an error-prone user experience. The integration of the student webpage version and the chatbot creates the experience of navigating through the OPF web application. Currently, the web pages and the chatbot allow general interaction between the web application and the user.

Lastly, Team 05 has added an analytics section that encompasses the statistics about the request ticket and buildings by using ApexCharts charting library. The users will be able to view statistics regarding the health of each building in order to ensure preventive care for the buildings and monitor the conditions of the buildings.

3.0 Updated Specifications

3.1 Summary of Changes in Project Specifications

Optimum Property Fix has undergone changes since Fall 2021 consisting of removal of technologies, features, and re-organization. The goal of OPF is to digitize maintenance requests

to remove paper trails and allow better communication between students and administrators. One of the major changes was the removal of features such as the mailing system and submitting inspection reports. These changes were made because they were either unnecessary features towards the main goal, or they were not possible to be implemented due to technology changes and/or time constraints. Additionally, the maintenance ticket page was further enhanced while the administrator and the user's analytics page is an added feature to the web application. The maintenance ticket page is the key component for the web application thus the functionality of the tickets process was updated. Next, the analytics page displays important information regarding the dormitories that would be beneficial for future applications outside the web application.

Equally important, the removal of technology was also needed such as the AWS and DynamoDB for the back-end technologies and AI recognition for objects and images. AWS and DynamoDB will be replaced with Flask and SQLite. The reason for these changes is that DynamoDB is a non-relational database, which is not suitable for the OPF application. As such, SQLite, a relational database will be utilized for the database. Additionally, AWS, which was selected because of its useful functionalities with AI recognitions and available memory for the database, will be replaced with Flask. Unfortunately, AWS technologies are out of the scope of the project due to the technology's intricacy and could potentially be revisited beyond the CS 426 course. Again, Team 05's main focus is building the components for the maintenance requests and the analytics within the semester time frame. As such, Flask will be utilized for the back-end framework and to create the log-in page to make the implementation manageable for the project.

3.2 Updated Technical Requirements Specification

The technical requirement benefits the team in establishing a system of services and constraints for the system. The technical requirements are separated into functional requirements and non-functional requirements. Additionally, the functional requirements state the services that OPF should provide, how OPF behaves in certain situations, and how the users are able to interact with OPF. The non-functional requirements are the constraints on the services and the functions provided by OPF.

Table 3.2.1 displays the functional requirements for the OPF application while Table 3.2.2 displays the non-functional requirements for the OPF application. The tables include an identification number, level number, and a brief description of the requirements. Next, each of the requirements is given a priority level. Level 1 represents the requirements planned for implementation by the end of the Spring 2022 semester. Level 2 represents the requirements that might be implemented by the end of the Spring 2022 semester. Lastly, Level 3 represents the helpful and useful requirements that will most likely not be implemented by the end of the Spring 2022 semester.

Table 3.2.1: Functional requirements for Optimum Property Fix (OPF).

| ID | Level | Description |
|------|-------|---|
| FR01 | 1 | OPF shall allow the user to create an account. |
| FR02 | 1 | OPF shall allow the user to log in to their account. |
| FR03 | 1 | Users shall be able to submit a ticket. |
| FR04 | 1 | Administrator users shall be able to receive a ticket. |
| FR05 | 1 | Administrator users shall be able to edit a ticket status. |
| FR06 | 1 | The FAQ page shall display questions and answers pertaining to a user. |
| FR07 | 1 | Dashboard page shall include a calendar with an appointment. |
| FR08 | 1 | Dashboard page shall display one meaningful feedback statistic for administrator users. |
| FR09 | 1 | Student users shall be able to submit a feedback form. |
| FR10 | 1 | Student users shall be able to view the ticket progress status of a specific ticket. |
| FR11 | 1 | The Maintenance Ticket page shall allow tickets to be filtered. |
| FR12 | 1 | OPF shall allow users to be able to log out. |
| FR13 | 2 | OPF shall allow the user to register as a student, administrator, or facilities services user. |
| FR14 | 2 | Users shall verify their account via email. |
| FR15 | 2 | Student users shall be able to add tags to their tickets. |
| FR16 | 2 | Administrator users shall be able to add questions and answers to the FAQ page. |
| FR17 | 2 | Dashboard page shall display announcements. |
| FR18 | 2 | Dashboard page shall allow the calendar to display appointment information. |
| FR19 | 2 | Dashboard page shall display the most recently created tickets for administrator users. |
| FR20 | 2 | Administrator users can view all student feedback on the Feedback page. |
| FR21 | 2 | Chatbot shall be able to direct users to their requested page. |
| FR22 | 2 | OPF shall automatically direct the user to the login page when the user logs out. |
| FR23 | 3 | OPF shall allow further differentiation when creating an account; including student, administrator, resident assistant, and allow for different edge cases. |
| FR24 | 3 | OPF shall allow users to log in via Single Sign-On (SSO). |
| FR25 | 3 | Users shall be able to add a picture and/or recording of the damage to the ticket. |

| | | |
|------|---|---|
| FR26 | 3 | Administrator users shall be able to accept or reject a ticket before the ticket is sent to a Facilities Services user. |
| FR27 | 3 | Administrator users shall be able to add notes and descriptions to the ticket if the ticket was not resolved. |
| FR28 | 3 | Administrator users shall be able to edit the FAQ page. |
| FR29 | 3 | Dashboard page shall allow users to edit/delete appointments from the calendar. |
| FR30 | 3 | Users shall be able to see their feedback on the Feedback page. |
| FR31 | 3 | Feedback page shall allow administrators to print and/or export feedback. |
| FR32 | 3 | Facilities Services users shall be able to view notes added by other members on the Maintenance Ticket page. |
| FR33 | 3 | Chatbot shall be able to answer user questions based on information from a database. |
| FR34 | 3 | Administrator users shall be able to customize the Analytics and Dashboard page. |
| FR35 | 3 | Administrator users shall be able to generate health reports on the Analytics page. |
| FR36 | 3 | Users shall stay logged into their account if they exit the website without logging out. |
| FR37 | 3 | Student users shall be able to submit a two-week survey based on their tickets. |
| FR38 | 3 | OPF shall be able to schedule appointments for Facilities Services users. |

Table 3.2.2 displays the non-functional requirements for the OPF application. The non-functional requirements are the constraints on the services and the functions provided by OPF. The constraints include timing constraints and constraints on the developing process. The non-functional requirements apply to the whole application rather than the individual features.

Table 3.2.2: Non-Functional Requirements for Optimum Property Fix (OPF).

| ID | Level | Description |
|-------|-------|---|
| NFR01 | 1 | The user interface of OPF shall be implemented in JavaScript and utilize the React framework. |
| NFR02 | 1 | OPF shall provide an intuitive and elegant user interface. |
| NFR03 | 1 | OPF shall be compatible with Google Chrome, Firefox, and Safari web browsers. |
| NFR03 | 1 | OPF shall perform as a well-developed front-end website. |

| | | |
|-------|---|---|
| NFR04 | 1 | OPF shall facilitate the communication between student and administrator users. |
| NFR05 | 1 | OPF shall utilize SQLite relational database management system to manage its database. |
| NFR06 | 1 | OPF shall utilize the Flask framework in python and Flask-SQLAlchemy for its backend development. |
| NFR07 | 1 | OPF shall utilize REST API to communicate between the React and Flask frameworks. |
| NFR08 | 1 | Each of the maintenance requests shall be uniquely identified with a ticket number. |
| NFR09 | 2 | OPF shall utilize a database server to store information. |
| NFR10 | 3 | OPF shall utilize AR implementation. |
| NFR11 | 3 | OPF shall perform as multi-platform software and run on different operating systems such as IOS, Android, and web applications. |
| NFR12 | 3 | OPF shall send surveys two weeks after the maintenance request is resolved. |

3.3 Updated Use Case Modeling

The creation of OPF's use cases, and use case diagrams define the functions of the application from the perspective of the user and the system. Therefore, the use case diagram demonstrates the various use cases and the different types of users of the OPF system. Figure 3.3.1 outlines the use cases which are represented by ellipses and color-coded as purple for the user. The users (also known as actors) act as stick figures and represent the student and the administrator.

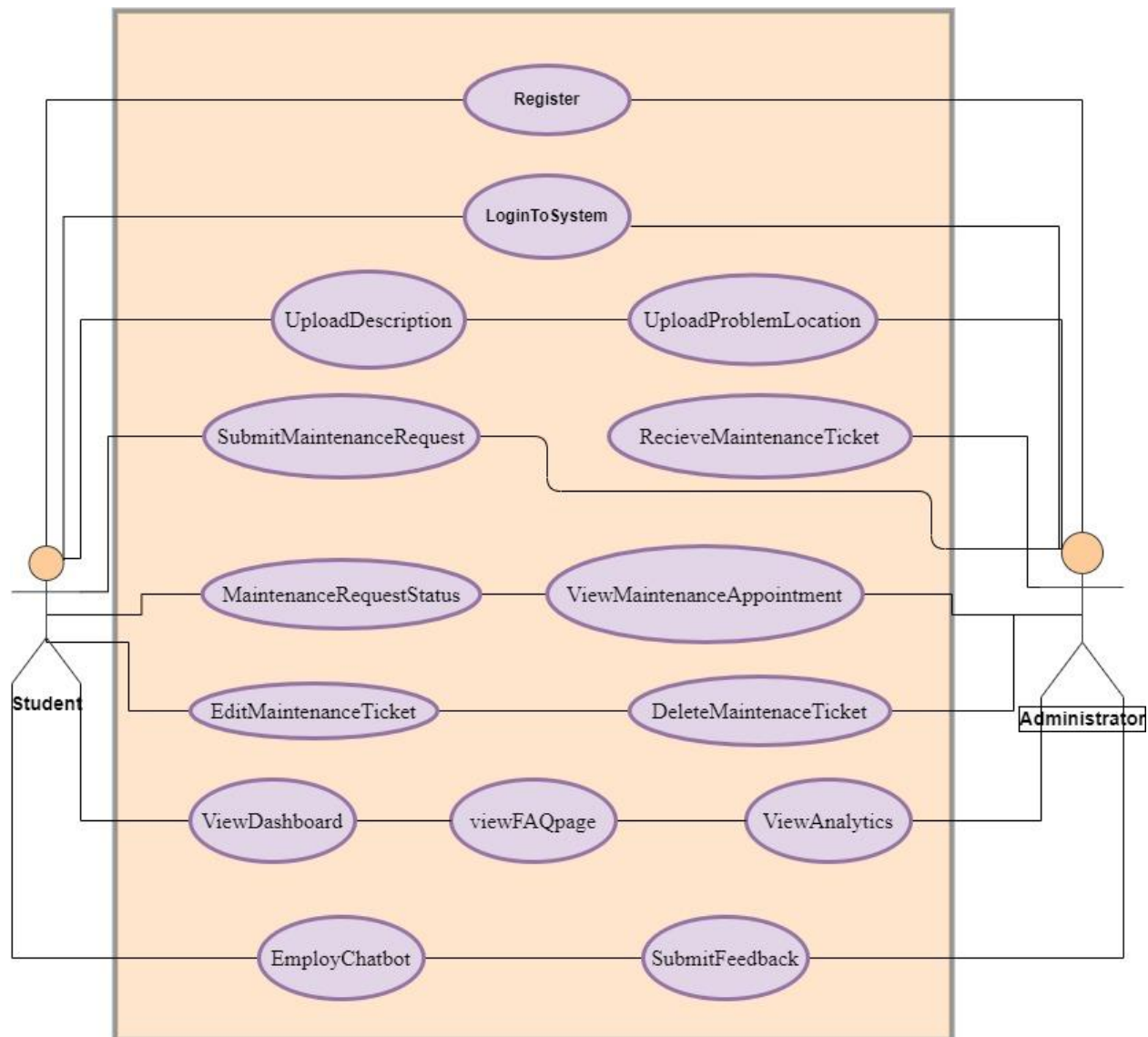


Fig. 3.3.1: Use case diagram outlining the various use cases and the different types of users of the OPF system.

A detailed use case description describes the narrative of the functionality of the web application OPF. The detailed use case diagram consists of detailed, step-by-step interactions between the user and the system, OPF. The description is the action taken to accomplish a specific goal set by Team 05. Additionally, the use cases are written from the point of view of the user of OPF. Table 3.3.1 describes the use cases by an identification number, a use case name, and a brief description of the functionality.

Table 3.3.1: Use cases describing the narrative of the functionality of the web application OPF.

| ID | Use Case | Description |
|------|----------------------------|---|
| UC01 | Register | Upon registration of the account, the user will enter information identifiable only to that specific user. The user will be able to register as a student or an administrator. |
| UC02 | LoginToSystem | The user will be able to log in to the OPF account using provided credentials. The user now has full access to the OPF website and its features. |
| UC03 | SubmitMaintenanceRequest | The user will be able to submit maintenance requests for their dormitory issue. The ticket will include information such as the problem location, description, title, and title. OPF will create a unique identifier once the request is submitted. |
| UC04 | EditMaintenanceTicket | The user will be able to edit information such as description, problem location, and status of a submitted ticket. The edited information will be saved to the maintenance ticket. |
| UC05 | UploadDescription | Upon the creation of a maintenance ticket, the user will have an option to upload a description of the issue. The user will also be able to edit the description once it is submitted. |
| UC06 | UploadProblemLocation | The user will be able to upload the location of their maintenance issue within their respective dormitory/facility. |
| UC07 | MaintenanceRequestStatus | The user will be able to view the status of their created maintenance request submitted on OPF. |
| UC08 | DeleteMaintenanceTicket | The user will be able to delete a submitted maintenance ticket. This will delete the ticket from the student and administrator side. |
| UC09 | ViewMaintenanceAppointment | The user will be able to view their maintenance appointment. The appointment will be located within the calendar on the dashboard page. |
| UC10 | ReceiveMaintenanceTicket | The administrator users will be able to receive the maintenance ticket from the student users. The administrator user will be able to accept or decline |

| | | |
|------|----------------|---|
| | | the maintenance request. |
| UC11 | ViewAnalytics | The user will be able to view meaningful statistics on the analytics page. The statistics can give insight about the data and building health. |
| UC12 | ViewDashboard | The user will be able to view their respective dashboard based on user type. The student and administrators user will view different layouts for their dashboard. |
| UC13 | EmployChatbot | The user will be able to use a chatbot function to guide them to a frequently asked question page. |
| UC14 | SubmitFeedback | The user will be able to provide feedback for the completed ticket. |
| UC15 | ViewFAQPage | The user will be able to view a page with frequently asked questions of users and proper authority. |

The Requirement Traceability Matrix relates both functional requirements with use test cases to ensure validation and that all requirements are checked. Functional requirements are statements to which the system should behave to certain inputs and also what the system should do. Additionally, the use cases identify the individual interactions between the system and users. Figure 3.3.2 provides the mapping of OPF's use cases and the functional requirements.

| | | USE CASES | | | | | | | | | | | | | | |
|-------------------------|------|-----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| | | UC 01 | UC 02 | UC 03 | UC 04 | UC 05 | UC 06 | UC 07 | UC 08 | UC 09 | UC 10 | UC 11 | UC 12 | UC 13 | UC 14 | UC 15 |
| FUNCTIONAL REQUIREMENTS | FR01 | | | | | | | | | | | | | | | |
| | FR02 | | | | | | | | | | | | | | | |
| | FR03 | | | | | | | | | | | | | | | |
| | FR04 | | | | | | | | | | | | | | | |
| | FR05 | | | | | | | | | | | | | | | |
| | FR06 | | | | | | | | | | | | | | | |
| | FR07 | | | | | | | | | | | | | | | |
| | FR08 | | | | | | | | | | | | | | | |
| | FR09 | | | | | | | | | | | | | | | |
| | FR10 | | | | | | | | | | | | | | | |
| | FR11 | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | |
|---|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| U I R E M E N T S | FR12 | | | | | | | | | | | | | | |
| | FR13 | | | | | | | | | | | | | | |
| | FR14 | | | | | | | | | | | | | | |
| | FR15 | | | | | | | | | | | | | | |
| | FR16 | | | | | | | | | | | | | | |
| | FR17 | | | | | | | | | | | | | | |
| | FR18 | | | | | | | | | | | | | | |
| | FR19 | | | | | | | | | | | | | | |
| | FR20 | | | | | | | | | | | | | | |
| | FR21 | | | | | | | | | | | | | | |
| | FR22 | | | | | | | | | | | | | | |
| | FR23 | | | | | | | | | | | | | | |
| | FR24 | | | | | | | | | | | | | | |
| | FR25 | | | | | | | | | | | | | | |
| | FR26 | | | | | | | | | | | | | | |
| | FR27 | | | | | | | | | | | | | | |
| | FR28 | | | | | | | | | | | | | | |
| | FR29 | | | | | | | | | | | | | | |
| | FR30 | | | | | | | | | | | | | | |
| | FR31 | | | | | | | | | | | | | | |
| | FR32 | | | | | | | | | | | | | | |
| | FR33 | | | | | | | | | | | | | | |
| | FR34 | | | | | | | | | | | | | | |
| | FR35 | | | | | | | | | | | | | | |
| | FR36 | | | | | | | | | | | | | | |
| | FR37 | | | | | | | | | | | | | | |
| | FR38 | | | | | | | | | | | | | | |

Fig. 3.3.2: The Requirement Traceability Matrix displaying the relationships between use cases, functional requirements, and non-functional requirements.

4.0 Updated Design

4.1 Summary of Changes in Project Design

The changes to the project design include pivoting the technology stack to better suit the scale of the application and capability of the team members. In addition, features were eliminated from OPF that were considered time-intensive. Moreover, Team 05 changed the focus of the website application to further prove programming concepts can expand after CS 426.

The current changes to the design document in comparison to fall for CS 425 include the context diagram, the programming units, and the database structure. The context diagram of the overall OPF web application included changes to the notification system. Unfortunately, the notification system is nonexistent in the project due to time restraints. Similarly, there have been changes to the database structure including identification of primary keys, foreign keys, and attributes. After further analysis and consultation with the advisor, Team 05 identified gaps between the database schema that needed immediate rectification to prevent faults within the system. Lastly, the programming units also need to be updated to reflect the current system by eliminating nonexistent relationships from the class diagram.

4.2 Updated High-Level and Medium-Level Design

4.2.1 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. The context diagram allows Team 05 to easily change and/or extend an entity without disarranging with the rest of the diagram. Figure 4.2.1.1, a context diagram, is a high-level view of the OPF web application system.

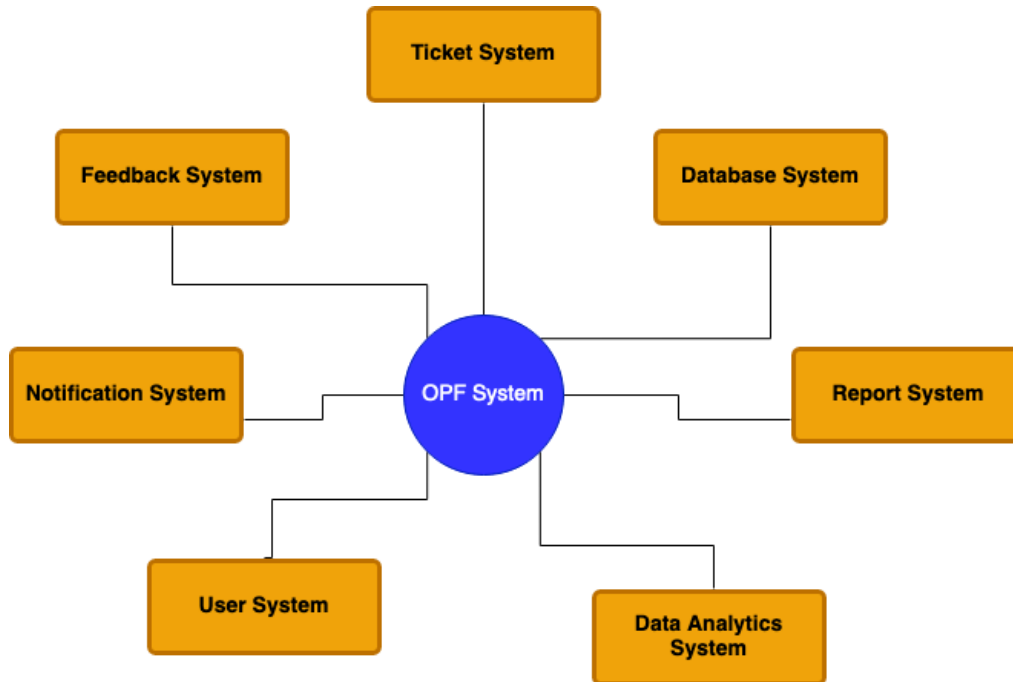


Figure 4.2.1.1: A high-level overview of a context diagram of the OPF web application describing the relationships between the system and its environment.

4.2.2 Programming 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. The table includes the class names, functions, and a brief description of its main attributes. Figure 4.2.2.1, is the overall visual representation of the class diagram for the OPF web application and its overall software architecture.

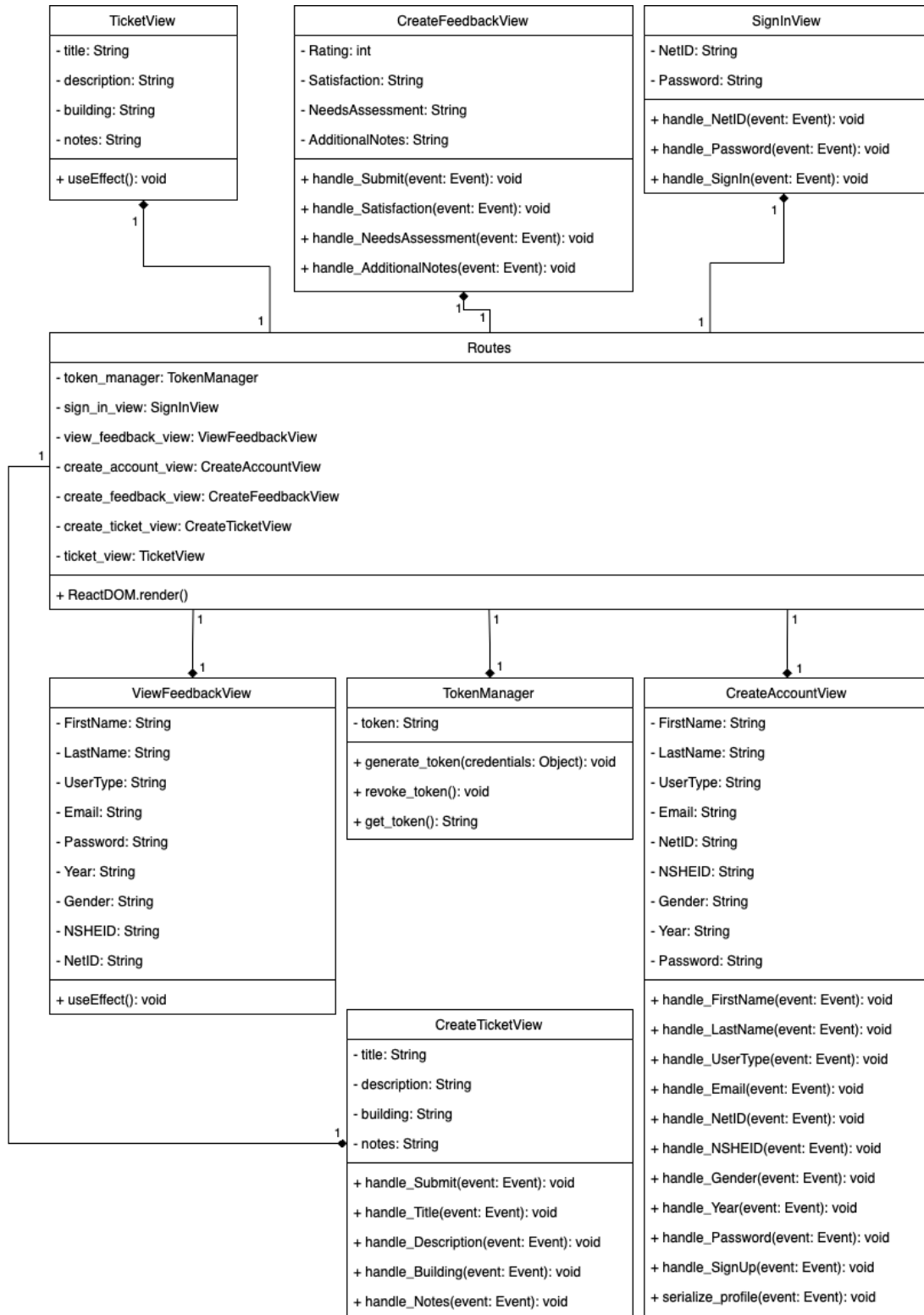


Figure 4.2.2.1: The design class diagram representing relationships, and multiplicity constraints for the OPF web application from the front-end.

Table 4.2.2.1: The class “CreateTicketView” contains all aspects of the methods that perform calls to create a ticket on behalf of the OPF users.

| | |
|----------------------------------|---|
| Class: CreateTicketView | The “CreateTicketView” class defines the overall layout and required fields and functionality required for processing user input to create a new ticket. The class accomplishes the task of getting user input, serializing data for transmission, and using the REST API to send the serialized data to the backend for further processing. |
| handle_Submit(event: Event) | The function is used to take-in a “onClick” event by the user and handle data the user submitted to create a ticket. The function handles the data by creating a JSON object of the attributes Title, Description, Building, and Notes into a text JSON object and then is serialized. Once serialized, the serialized object is sent to the backend with the REST API. |
| handle_Title(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Title” remains updated to information entered by the user. The semantic use for Title is to describe a title for the Ticket. |
| handle_Description(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Description” remains updated to information entered by the user. The semantic use for the Description is to describe in further detail what the dormitory student would like to report to facility services about their issue to the ticket. |
| handle_Building(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Building” remains updated to information entered by the user. The semantic use is to describe the building of origin of issue to the ticket. |

| | |
|----------------------------|--|
| handle_Notes(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Notes” remains updated to information entered by the user. The semantic use is to describe special instructions to facility services by the dormitory student and other things not related to the issue itself. |
|----------------------------|--|

Table 4.2.2.2: The class “TicketView” contains all aspects of the methods that perform calls to view a ticket on behalf of the OPF users.

| | |
|--------------------------|---|
| Class: TicketView | The “TicketView” class defines the overall layout and required fields and functionality required for interacting with and showing all information related to a single ticket. The class accomplishes the task of getting ticket data from the backend and rendering it to the frontend. |
| useEffect() | The function is used to make an API call to the backend to receive ticket information based upon the current ticket update class attributes with data from the backend. The attributes are then used to update variables rendered within the page. |

Table 4.2.2.3: The class “CreateFeedbackView” contains all aspects of the methods that perform calls to add feedback to a ticket behalf of OPF’s dormitory student users.

| | |
|----------------------------------|--|
| Class: CreateFeedbackView | The “CreateFeedbackView” class defines the overall layout and required fields and functionality required for processing user input to submit a feedback regarding a ticket made by the user. The class accomplishes the task of getting user input, serializing data for transmission, and using the REST API to send the serialized data to the backend for further processing. |
|----------------------------------|--|

| | |
|--------------------------------------|--|
| handle_Submit(event: Event) | The function is used to take-in a “onClick” event by the user and handle data the user submitted to send feedback about a ticket. The function handles the data by creating a JSON object of the attributes Satisfaction, NeedsAssessment, and AdditionalNotes into a text JSON object and then is serialized. Once serialized, the url parameter for the ticket is identified and is sent with the serialized object to backend with the REST API to be added to a specific ticket. |
| handle_Satisfaction(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Satisfaction” remains updated to information entered by the user. Satisfaction uses three values of “Yes” and “No” that are saved as strings to be sent to the backend. The semantic use for the Satisfaction is to report to facility services if the dormitory students issue was resolved. |
| handle_NeedsAssessment(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “NeedsAssessment” remains updated to information entered by the user. Satisfaction uses three values of “Yes, Maybe, and No” that are saved as strings to be sent to the backend. The semantic use for the assessment needs is to gauge to facility services how much of the issue is resolved. |
| handle_AdditionalNotes(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “AdditionalNotes” remains updated to information entered by the user. The semantic use for the additional notes is simply to add comments about the issue resolution to facility services. |

Table 4.2.2.4: The class “CreateAccountView” contains all aspects of the methods that perform calls to create an account on behalf of the OPF users.

| | |
|---------------------------------|--|
| Class: CreateAccountView | The “CreateAccountView” class defines the overall layout and required fields and functionality required for processing user input to create a new user. The class accomplishes the task of getting user input, serializing data for transmission, and using the REST API to send the serialized data to the backend for further processing. |
| handle_FirstName(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “FirstName” remains updated to information entered by the user. The semantic use for “FirstName” is to describe the first name of the user. |
| handle_LastName(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “LastName” remains updated to information entered by the user. The semantic use for “LastName” is to describe the last name of the user. |
| handle_UserType(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “UserType” remains updated to information entered by the user. The semantic use for “UserType” is to be able to identify the student as a student or an administrator (such as facility services). |
| handle_Email(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Email” remains updated to information entered by the user. The semantic use for “Email” is to be able to keep in-record of the user's email for communication purposes. |
| handle_NetID(event: Event) | The function is used to update a React state |

| | |
|-------------------------------|---|
| | variable based upon change from user input into an HTML form. The function ensures the class attribute “NetID” remains updated to information entered by the user. The semantic use for “NetID” is to be able to uniquely identify a user using their university assigned NetID. This value cannot be empty. |
| handle_NSHEID(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “NSHEID” remains updated to information entered by the user. The semantic use for “NSHEID” is to be able to identify students through their NSHE ID for various purposes such as bailing. |
| handle_Gender(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Gender” remains updated to information entered by the user. |
| handle_Year(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Year” remains updated to information entered by the user. The semantic use for “Year” is to be able to identify a student by their number of year(s) at the university. |
| handle_Password(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Password” remains updated to information entered by the user. |
| handle_SignUp(event: Event) | The function is used to take-in a “onClick” event by the user and handle data the user submitted to create a new user account. The function handles the data by calling the “serialize_profile()” function to return a JSON object of a user profile. Once serialized, the JSON object is sent to the backend to be added to the database. |

| | |
|---------------------|---|
| serialize_profile() | The function is used to serialize the class attributes into a JSON object. This function would be use to ready a message for the REST API to send to the backend. |
|---------------------|---|

Table 4.2.2.5: The class “SignInView” contains all aspects of the methods that perform calls to sign into the application on behalf of the OPF users.

| | |
|-------------------------------|--|
| Class: SignInView | |
| handle_NetID(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “NetID” remains updated to information entered by the user. |
| handle_Password(event: Event) | The function is used to update a React state variable based upon change from user input into an HTML form. The function ensures the class attribute “Password” remains updated to information entered by the user. |
| handle_SignIn(event: Event) | The function is used to take-in a “onClick” event by the user and handle data the user submitted to authenticate into the application. The function serializes credential information and sends it to the backend via the REST API to authenticate users. If an user is authenticated, the function saves an access token used by the application for an active session. |

Table 4.2.2.6: The class “ViewFeedbackView” contains all aspects of the methods that perform calls on behalf of the OPF users.

| | |
|--------------------------------|---|
| Class: ViewFeedbackView | The “ViewFeedbackView” class defines the overall layout and required fields and functionality required for interacting with and showing all information related to a single ticket. The class accomplishes the task of getting feedback data from the backend and rendering it to the frontend. |
| useEffect() | The function is used to make an API call to the backend to receive feedback information based upon the current ticket selected through |

| | |
|--|--|
| | the use of an url parameter for the current ticket. Class attributes are updated with returned information from the backend and rendered within the webpage. |
|--|--|

Table 4.2.2.7: The class “TokenManager” contains all aspects of the methods that perform calls to authenticate on behalf of the OPF users.

| | |
|-------------------------------------|---|
| Class: TokenManager | The “TokenManager” class is used to manage active session management by the user and assist in the generation and deletion of access tokens used by the frontend to access the backend. |
| generate_token(credentials: Object) | The function is used to make a request to the backend to generate a new access token based upon the credentials being correct and authenticated. The credentials object is a non-serialized object containing a user’s NetID and password. The generated token is stored within the browser’s local storage for continuous access by the application. |
| revoke_token() | The function is used to make a request to the backend to revoke the current access token in the backend. Once removed the function removes the access token from the browser’s local storage. |
| get_token() | The function is used to get the current access token stored within the browser’s local storage. |

Table 4.2.2.8: The class “Routes” contains all aspects of the methods that perform calls on behalf of the OPF users.

| | |
|-----------------------------------|---|
| Class: Routes | The “Routes” class is used to define the routing of the front-end and expected views of the application when the user goes to a certain url. These views are swapped depending upon the current url in the application. |
| ReactDOM.render(document: String) | This function is used to render the active of the application by taking in HTML markup to render to the browser. |

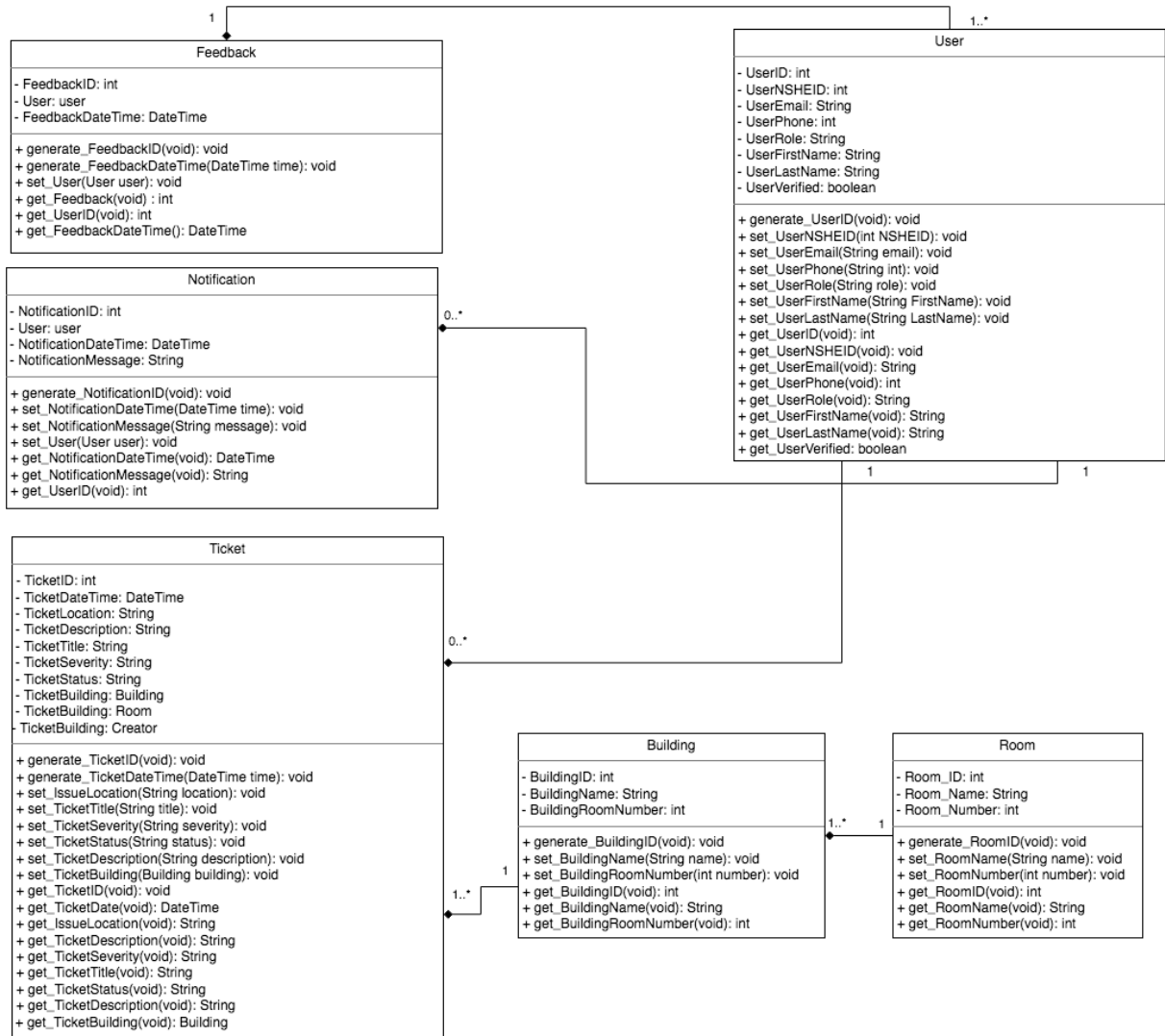


Figure 4.2.2.2: The design class diagram representing relationships, and multiplicity constraints for the OPF web application from the backend.

Table 4.2.2.9: 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 4.2.2.10: 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 4.2.2.11: 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_TicketSeverity(severity) | The function sets the ticket severity as either high, medium, and/or low. The severity 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. |

Table 4.2.2.12: 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. |

4.2.3 Database Structure

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 4.2.3.1, outlines the overall structure of the OPF web application's database and its relationships.

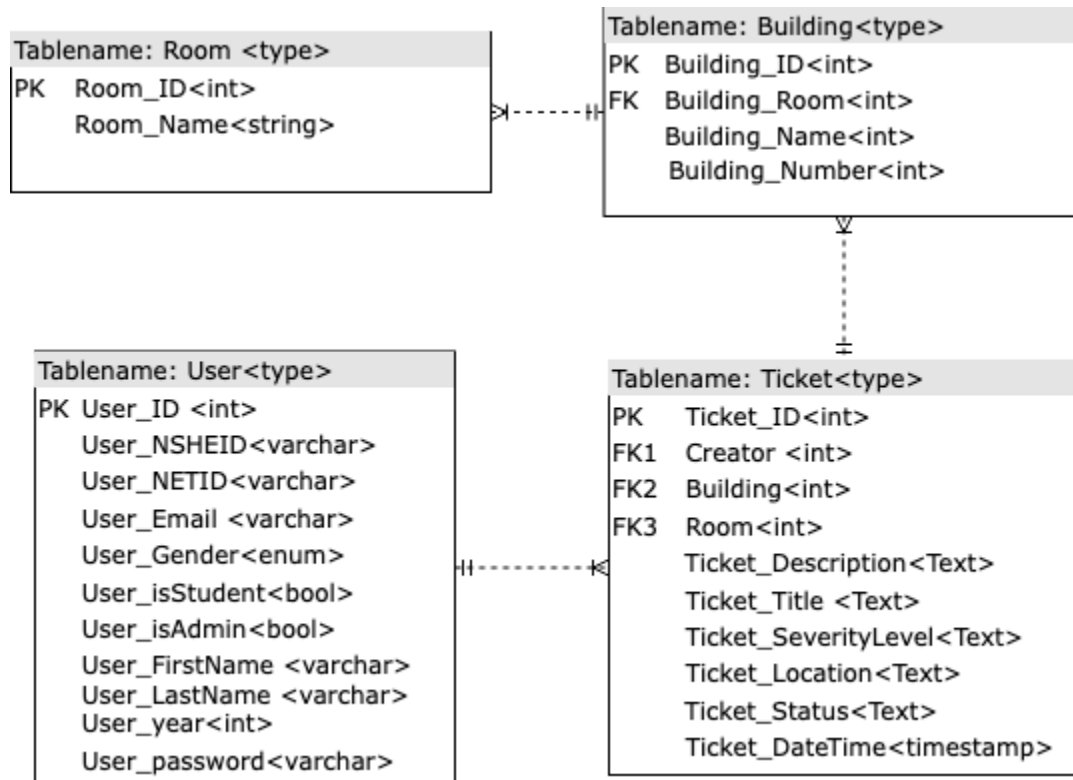


Figure 4.2.3.1: The overview of the OPF web application database management system and its interrelationships between the system.

All tables in Section 4.2.3 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 4.2.3.1: 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 a digit and identified as a primary key in the user table. | ‘1’ |
| | User_FirstName <varchar> | The first name of the user for the OPF web application. The first name of the user will be stored as type varchar in the user table. | ‘Joanna’ |
| | User_LastName <varchar> | The last name of the user for the OPF web application. The last name of the user will be stored as type varchar in the user table. | ‘Lopez’ |
| | User_NSHEID <varchar> | 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 varchar in the user table. | ‘0123456789’ |
| | User_NETID <varchar> | The NET ID of the user for the OPF website is a unique identifier stored as a variable character of variable length. | ‘joannalopez’ |
| | 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. | ‘joannalopez@nevada.unr.edu’ |

| | | | |
|--|----------------------------|--|-----------------|
| | User_Gender <enum> | The gender of the students is identified as an enum in the user table of 'M' or 'F'. | 'F' |
| | User_isStudent <bool> | The isStudent is a boolean and can either be True or False in the user table. It is used to identify if a user is a student through login to render specific student pages. | '1' |
| | User_isAdmin <bool> | The isAdmin is a boolean and can either be True or False in the user table. It is used to identify if a user is an administrator through login to render specific student pages. | '0' |
| | User_Year <int> | The year is an integer and is used in the user table to identify the student's enrollment year. | '4' |
| | User_Password <varchar> | The password is a variable character that is stored in the user table where the user can log into their account. | '\$Capstone123' |

Table 4.2.3.2: 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 tickets of the OPF web application. The type will be stored as an integer as an 8-digit and identified as a primary key in the ticket table. | '12349874' |
| <i>Foreign Key</i> | Building <int> | The building identifier from the building table. It's a foreign key and will be stored as a type integer in the ticket table. The number '13' identifies dormitory 'Argenta Hall'. | '13' |
| <i>Foreign Key</i> | Room <int> | The room identifier from the room table. | '205' |

| | | | |
|--------------------|------------------------------|--|---|
| | | It's a foreign key and will be stored as a type integer in the ticket table. | |
| <i>Foreign Key</i> | Creator <int> | The identifier from the user table that created the ticket on the OPF web application. It will be stored as a type integer in the ticket table. | '1' |
| | Ticket_Date<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. | '2022-02-15 08:02:08' |
| | Ticket_Description <Text> | The ticket description stores any information the user provides about their issue on the OPF web application. It is identified as text in the ticket table. | 'Help, I smell gas leaking from the kitchen stove!' |
| | Ticket_Title <Text> | The ticket title is a short description the user provides about their issue on the OPF web application. It is identified as text in the ticket table. | 'Gas Leak' |
| | Ticket_Severity <Text> | The ticket severity is used to identify the severity level the user provides about their issue. It is identified as text in the ticket table. | 'High' |
| | Ticket_Location <Text> | The ticket location is where the issue is located in the building and in the room number. It is identified as text in the ticket table. | 'Kitchen' |
| | Ticket_Status <Text> | The ticket status is the current information and can either be 'pending', 'completed', 'deleted'. It is identified as text in the ticket table. | 'Pending' |

Table 4.2.3.3: 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 analytics and the ticketing system.

| Building | | | |
|--------------------|--------------------------------|---|----------------|
| Key | Name <Type> | Description | Example |
| Primary Key | Building_ID <int> | The identification to differentiate between all users of the OPF web application. The type will be stored as an integer as a digit and identified as a primary key in the user table. | ‘1’ |
| <i>Foreign Key</i> | Building_Room <int> | The room is a foreign key from the room table and will be stored as a type integer in the building table. | ‘205’ |
| | Building_Number <int> | The number of the building table will be stored as a type integer in the building table. | ‘13’ |
| | Building_Name <string> | The name of the building will be stored as a type string in the building table. | ‘Argenta Hall’ |

Table 4.2.3.4: The “Room” table stores the relevant data for multiple rooms at each building. The room table will contain information that will be used in the ticketing system.

| Room | | | |
|--------------------|----------------------------|---|----------|
| Key | Name <Type> | Description | Example |
| Primary Key | Room_ID <int> | The identification to differentiate between all rooms of the OPF web application. The type will be stored as an integer as a digit and identified as a primary key in the room table. | ‘1’ |
| | Room_Name <string> | The name from the room table will be stored as a type string that will | ‘Double’ |

| | | | |
|--|--|--|--|
| | | differentiate between singles, doubles, triples, quad living arrangements. | |
|--|--|--|--|

4.3 Updated Hardware Design

Optimum Property Fix does not have any hardware components at this time.

4.4 Updated 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 4.4 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 Figma, a user interface creation and prototyping tool.

The image shows a user interface for 'Optimum Property Fix'. On the left side, there is a vertical sidebar with the company logo at the top, followed by the text 'Already Here?' and 'Log in to your existing account to access Dashboard'. Below this is a red 'Log In' button. At the bottom of the sidebar, it says 'Optimum Maintenance Service for your University's Housing'. The main area of the interface is a dark blue background with the title 'Create an Account' in white. Below the title is a form with several input fields: 'Who are you?' (a dropdown menu), 'Net ID', 'First Name', 'Last Name', 'Gender' (a dropdown menu), 'Date of Birth', 'Email Address', 'Password', and 'Confirm Password'. At the bottom of the form is a red 'Sign Up' button.

Fig. 4.4.1: This snapshot displays the page where users are able to create an account with Optimum Property Fix. The user is prompted to enter the appropriate credentials to sign up and on the left hand side, if a user already has an account, they may proceed to log in.

New Here?

Sign up now to report your first maintenance issue and receive Optimum service!

[Sign Up](#)

Optimum Maintenance Service
for your University's Housing

Welcome to Optimum Property Fix

Please sign in to continue.

[Forgot Password?](#)

[Log In](#)

Fig. 4.4.2: This snapshot displays the page where users are able to sign into their account with Optimum Property Fix. The user is prompted to enter the appropriate credentials to sign in and on the left hand side, if a user does not have an account, they may click the sign up button which will take them to the screen shown above.

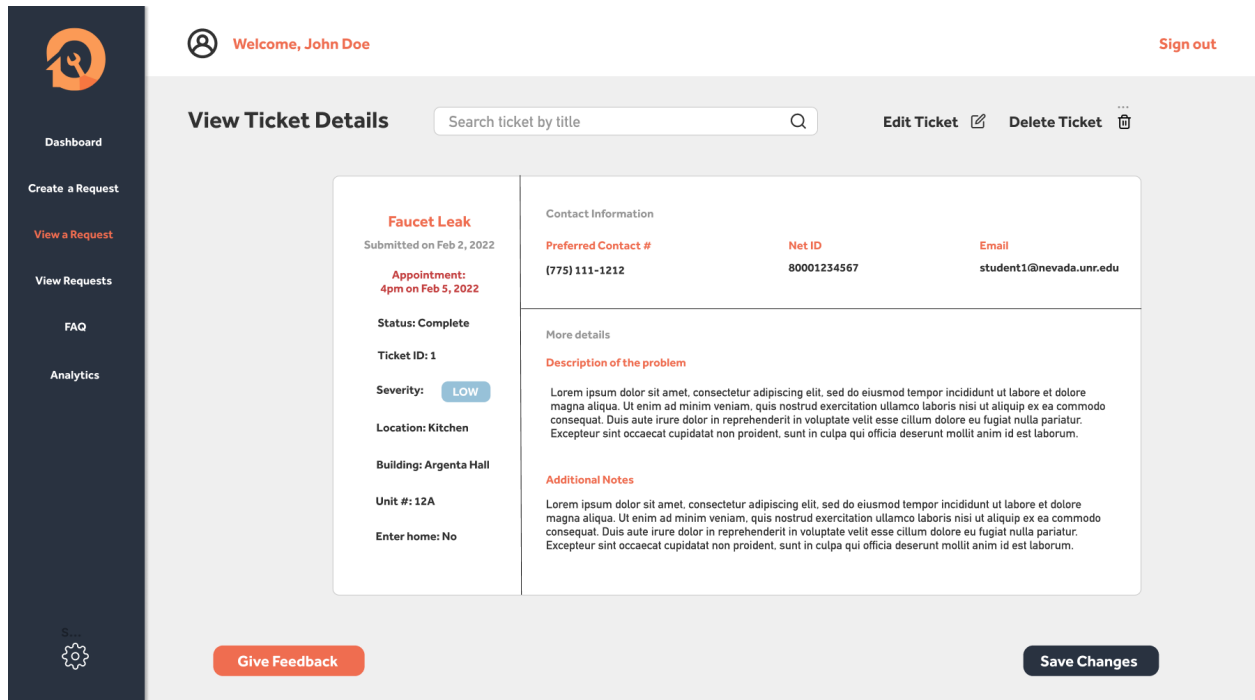


Fig. 4.4.3: This screen displays the student side of the web application where students may view an individual ticket’s details. The user may search for the ticket they want, by title, edit ticket and delete the ticket. The card includes the description, user’s contact details, and more details such as appointment time and date. The bottom left allows the user to give feedback and the bottom right allows the user to save any changes they make.

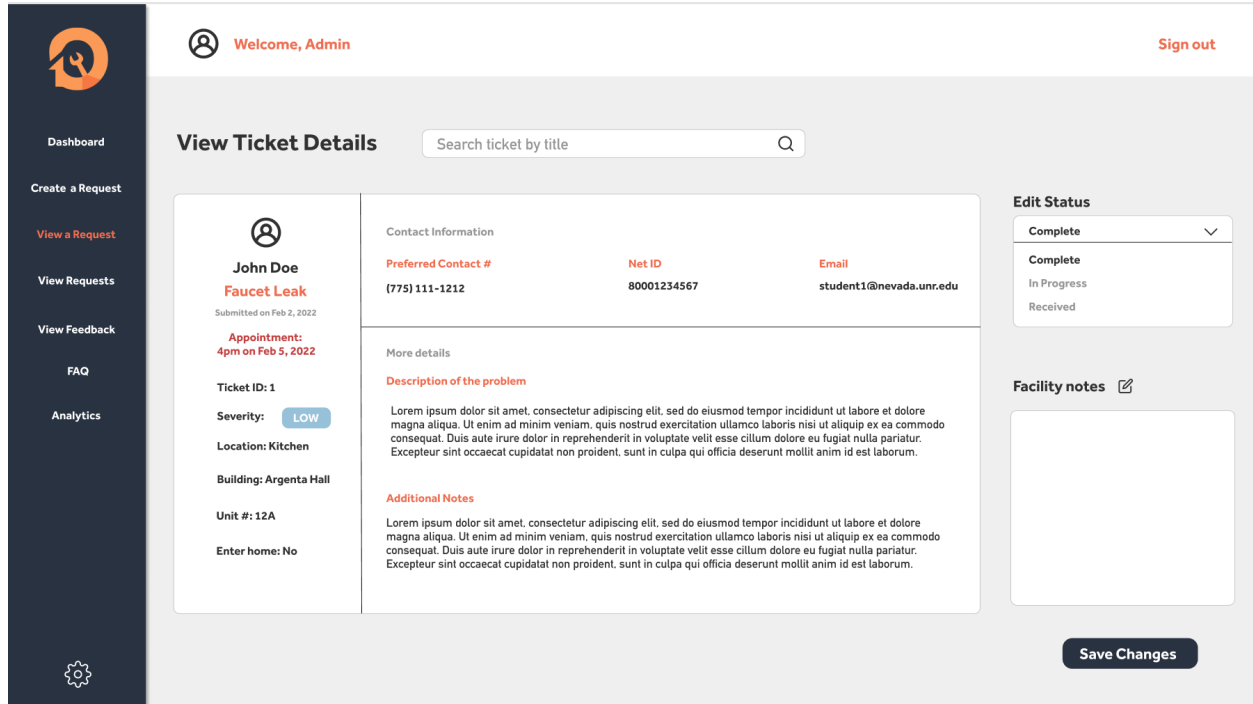





Fig. 4.4.4: This screen displays the admin side of the web application where admins may view an individual ticket's details. The admin may search for the ticket they want, by title and edit the status of the ticket which will render into the student's side. The card includes the description, user's contact details, and more details such as appointment time and date. The right hand side also has an option for the facility to write notes which they can view later on and the bottom right allows the user to save any changes they make.



Dashboard
Create a Request
View a Request
View Requests
FAQ
Analytics





Welcome, John Doe
Sign out

View all tickets

| Title | Status | ID | Severity | Date Requested | Location |
|------------------|-------------|----|----------|----------------|-------------|
| Broken Sink | Complete | 3 | High | 02-12-2022 | Bathroom |
| Faucet Leak | In Progress | 5 | Low | 02-2-2022 | Kitchen |
| Broken Heater | In Progress | 7 | Mild | 01-25-2022 | Living Room |
| Broken Sink | Complete | 8 | Low | 10-2-2022 | Kitchen |
| Clogged Toilet | Complete | 1 | Low | 01-18-2022 | Bedroom |
| Air Conditioning | Complete | 6 | Mild | 03-14-2022 | Bedroom |
| Drain Clogs | Complete | 4 | Low | 04-19-2022 | Shower |
| Leaky Faucet | Received | 2 | High | 01-12-2022 | Bathroom |
| Broken Door | Complete | 9 | High | 03-5-2022 | Bedroom |
| Shower Drain | Received | 10 | Mild | 01-5-2022 | Bathroom |

Fig. 4.4.5: This screen displays all of the tickets made by the student. It is displayed in a table and sorted by title, status, ID, severity, the date requested and location. The top left allows the user to search tickets and filter them by the categories.



Dashboard

Create a Request


View a Request


View Requests

Feedback

FAQ

Analytics




Welcome, Admin

Sign out

View all tickets











| Title | Status | | ID | Severity | Date Requested | Location |
|------------------|-------------|---|----|----------|----------------|-------------|
| Broken Sink | Complete |  | 3 | High | 02-12-2022 | Bathroom |
| Faucet Leak | In Progress |  | 5 | Low | 02-2-2022 | Kitchen |
| Broken Heater | In Progress |  | 7 | Mild | 01-25-2022 | Living Room |
| Broken Sink | Complete |  | 8 | Low | 10-2-2022 | Kitchen |
| Clogged Toilet | Complete |  | 1 | Low | 01-18-2022 | Bedroom |
| Air Conditioning | Complete |  | 6 | Mild | 03-14-2022 | Bedroom |
| Drain Clogs | Complete |  | 4 | Low | 04-19-2022 | Shower |
| Leaky Faucet | Received |  | 2 | High | 01-12-2022 | Bathroom |
| Broken Door | Complete |  | 9 | High | 03-5-2022 | Bedroom |
| Shower Drain | Received |  | 10 | Mild | 01-5-2022 | Bathroom |

Fig. 4.4.6: This screen displays the admin side of the tab of all tickets. It is displayed in a table and sorted by title, status, ID, severity, the date requested and location. The top left allows the user to search tickets and filter them by the categories. The admin also has the ability to edit the status of the ticket.

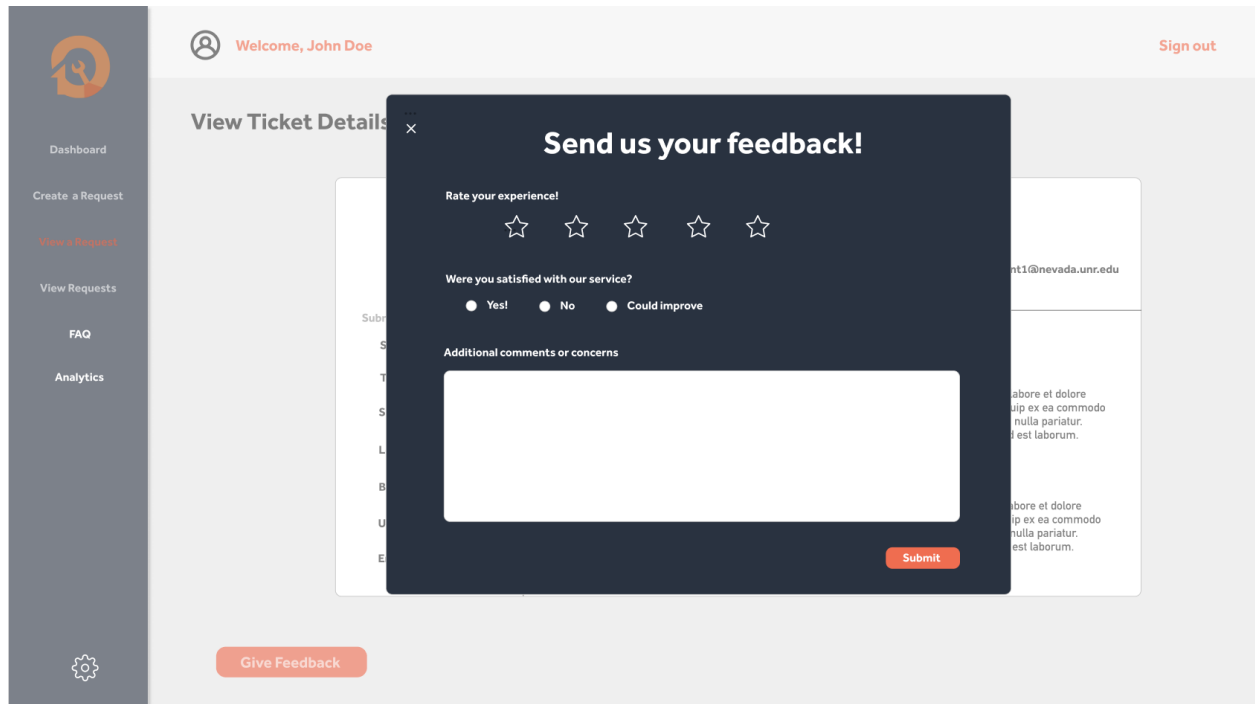


Fig. 4.4.7: As the student is on the view single ticket page and chooses the option to give feedback, this feedback alert box will pop up and prompt the user to answer three questions which will be sent back to the facility.

The screenshot displays a web application interface for creating a new ticket. On the left is a dark sidebar with a logo at the top and navigation links: Dashboard, Create a Request (highlighted in orange), View a Request, View Requests, FAQ, and Analytics. At the bottom of the sidebar is a gear icon for settings. The main content area has a header with a user profile icon, the text 'Welcome, John Doe', and a 'Sign out' link. The title 'Create New Ticket' is centered at the top of the form. The form is organized into several sections: 'Title' with a text input field and an example 'ex. "Faucet Leak"'; 'Please describe the problem' with a large text area; 'Select Appointment time' with 'Pick Date' and 'Pick Time' (with a clock icon) buttons; 'Does our maintenance team have permission to enter your residence without your permission?' with radio buttons for 'Yes' and 'No'; 'Set Severity' with three buttons: 'LOW' (blue), 'MILD' (orange), and 'HIGH' (red); 'Location of problem' with a dropdown menu showing 'Living room'; 'Your Building' with a dropdown menu showing 'Argenta Hall'; 'Your Unit #' with a text input field; 'Preferred Contact Number' with a text input field; and 'Additional Notes' with a large text area. A 'Submit Ticket' button is located at the bottom right of the form.

Fig. 4.4.8: This screenshot shows the create new ticket page where users may fill out a form that includes information needed to be collected in order for their maintenance issue to be fixed. There are a variety of questions such as drop downs, text boxes and button selection. The user can submit the ticket once it is done.

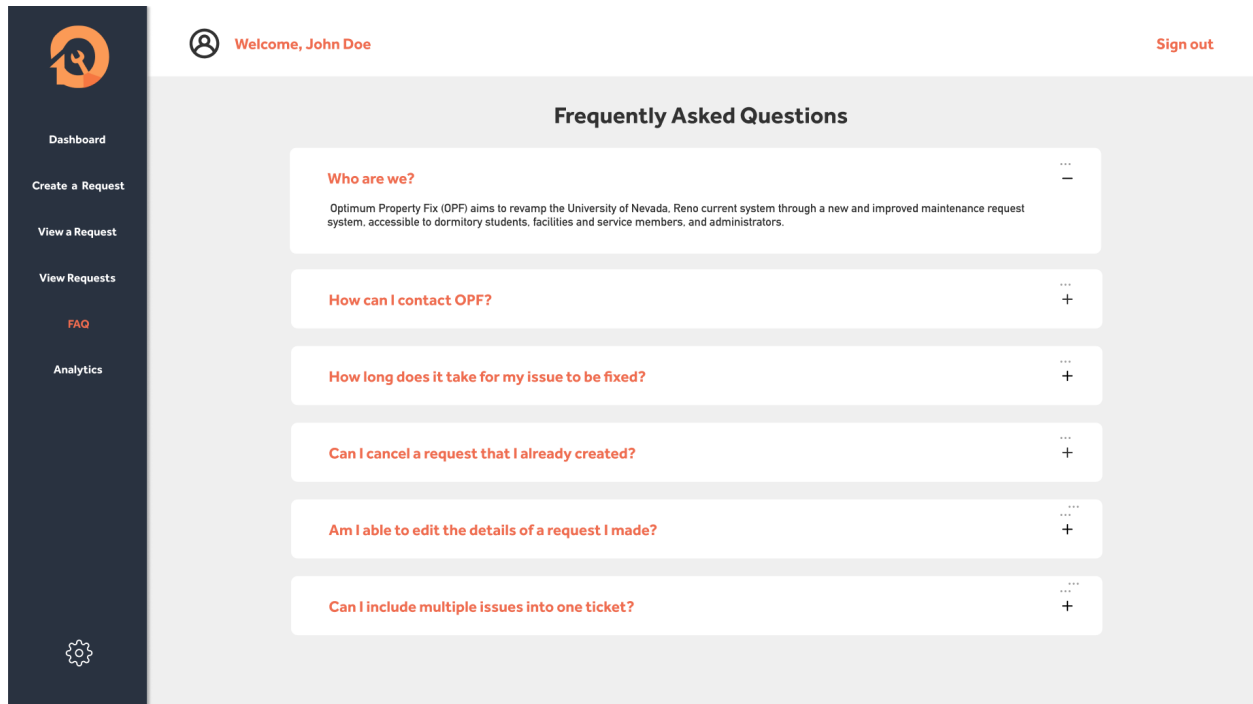


Fig 4.4.9: Here is a snapshot of the frequently asked questions page the student may visit. There is a collection of questions the student can expand and collapse in order to view the answer. Contact information is provided for any additional question the user may have.

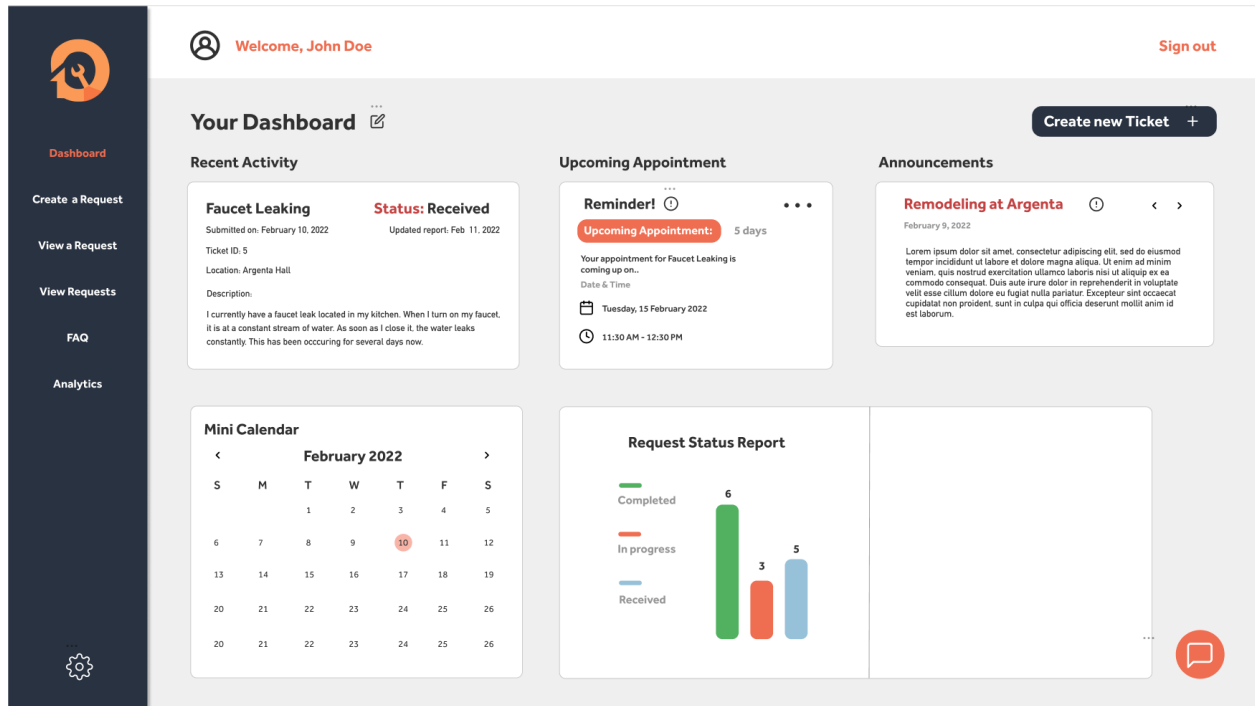


Fig 4.4.10: Here is a snapshot of the student dashboard page. Here the student is able to view all the essential data that they need such as their recent ticket, appointment reminders, important announcements made by the facility, a calendar and simple analytics. More analytics will be added that are of value for the user. There is also a chatbot that can answer questions and direct them to the FAQ page and an option to create a new ticket and edit the dashboard.

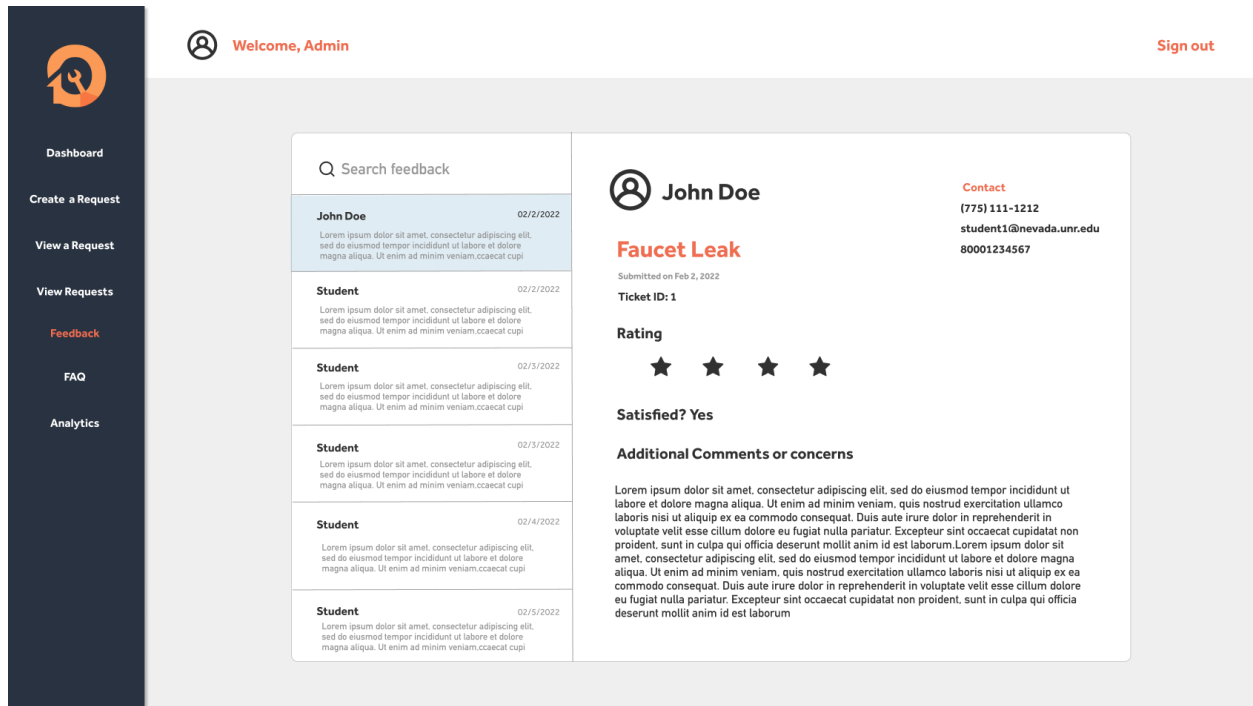


Fig 4.4.11: Here is a snapshot of the view feedback page for the admin. Here, the admin may search through all feedview and view a single one. All feedback sent to the admin will be able to be viewed on this page.

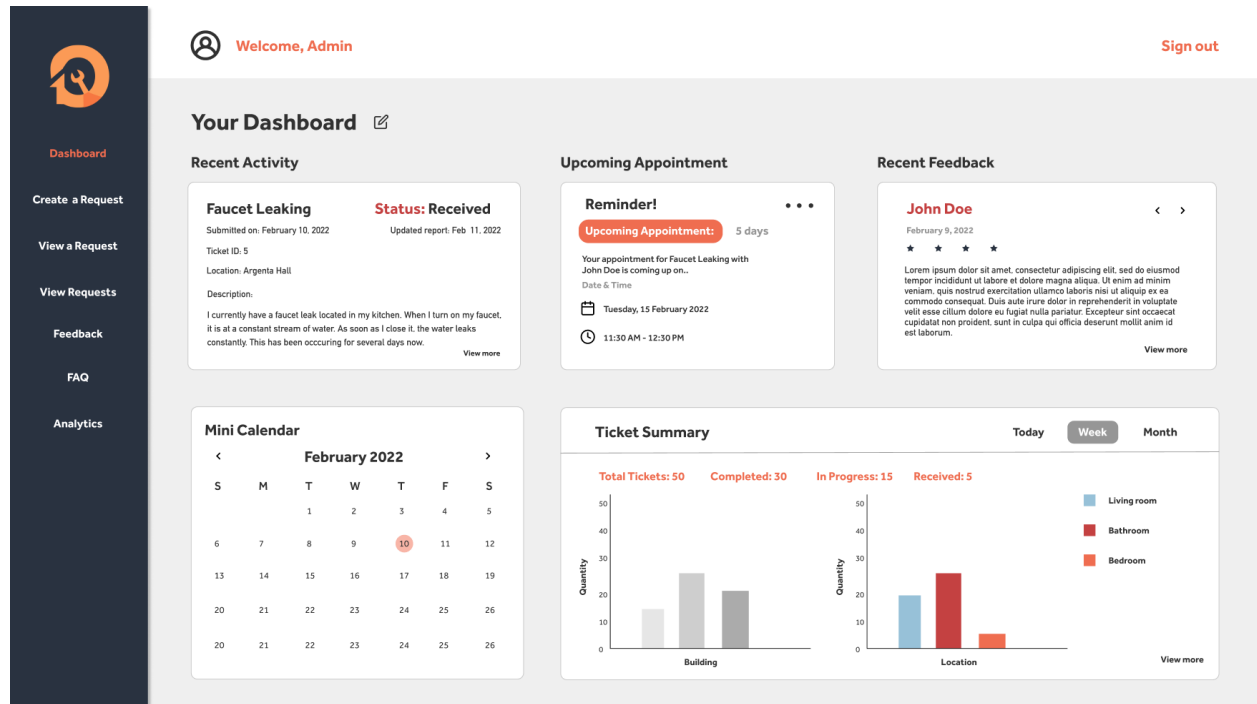


Fig 4.4.12: Here is a snapshot of the Admin dashboard page. Here the admin is able to view all the essential data that they need such as the recent ticket submitted, appointment reminders, recent feedback, a calendar and a preview of important analytics. Just like students, there is also an option to edit the dashboard layout.

5.0 Updated Glossary of Terms

Americans with Disabilities Act (ADA)

A law that requires that property that is open to the public include features access to the building. The ADA is designed to eliminate discrimination against individuals with disabilities by providing equal access to jobs, public accommodations, government services, public transportation, and telecommunications.

Appliance Repair

Used for people in the field that are knowledgeable and qualified to fix/repair appliances.

Building Standard

The specific set of amenities and alternations a landlord is willing to make free of chased for an incoming commercial tenant.

Commercial Property

Income-producing properties; public accommodations.

Corrective Maintenance

Maintenance is required when equipment or item has failed or been worn out, allowing an object to be restored and usable.

Deferred Maintenance

Physical depreciation or loss in value of a building resulting from postponed maintenance to the building.

Facilities Services

Professional management focused on the efficient and effective delivery of logistics and other services related to real property.

Fair Housing Act (Title VIII of the Civil Rights Act of 1968)

A federal law that prohibits discrimination in the sale, rental, or financing of housing based on race, color, religion, national origin, sex, familial status, and handicap.

Inspection Report

A document that is written by a home inspector after a thorough evaluation of the home's condition, including the electrical system, plumbing, furniture condition, and other structural features.

Maintenance Request

A formal way of asking the landlord for a repair or adjustment of a property or an establishment.

Operating Budget

A projection of income and expense for the operation of a property over a one-year period.

Operating Costs

A calculation of yearly costs of operation based on operating expenses from comparable properties and the maintenance needs of the subject property; used in preparing an annual operating budget

Paper Trail

A series of documents providing written evidence of a sequence of events or the activities of a person or organization

Preventive Maintenance

The regular and routine maintenance of equipment and assets in order to keep them running and prevent any costly unplanned interruption from unexpected equipment failure.

Property Evaluation

The estimation of one's property worth, which includes buildings, lands, and other properties.

Property Management

A branch of the real estate profession that seeks to preserve or increase the value of an investment property while generating income for its owners.

Reactive Maintenance

Maintenance is done to property or object when it breaks. The strategy follows that the person only fixes the property only when it breaks.

Regular Recurring Cost

A type of fixed property expense (eg., cleaning costs) that occurs consistently each month.

Rental History

A record of the prospect's previous patterns that can influence the manager's decision to rent or not rent. Considerations include frequent moves, expensive modifications, and future expansion requirements.

Residential Property

The type of property where people live. It includes privately-owned dwellings as well as government and instructional ownership and provides the greatest demand for professional property management.

Security Deposit

A payment by a tenant, held by the landlord during the lease term and kept (wholly or partially) on default or destruction of the premises by the tenant. Individual states set forth rules for holding, retaining, or returning security deposits.

Special Repairs

Repairs are done to a property by replacing existing parts that are deteriorated or deteriorating due to the property aging.

Tenant Emergency Procedures Manual

Printed booklet outlining emergency organization, workday procedures, telephone numbers, and after-hours procedures during an emergency.

Variable Expense

An expense item in a property's operating budget that increases or decreases with the occupancy level of the building.

Work Order Bids

Work estimates filed according to the property involved providing important information comparison data when considering future projects.

6.0 Engineering Standards and/or Technologies

ECMA 404

ECMA 404 defines the JavaScript Object Notation Standard and is a common data interchange format that uses a combination of key/value pairs that make up objects and arrays to store a set of objects. JSON is commonly used to define the communication formats used between clients and servers thus defining a RESTful API.

SQLite

SQLite is an open-source relational database management system that allows users to keep the existing data in a database organized. OPF will utilize SQLite to keep its data organized.

Javascript

JavaScript is a programming language used commonly for web applications inheriting several characteristics of being an interpreted language and dynamically typed language. JavaScript as technology is based upon the standards set forth by the ECMAScript language specification that web-browsers adopt to implement the browser interpreter for JavaScript.

React

React is a declarative and component-based library used to create user interfaces using core web technologies such as JavaScript, HTML, and CSS. React's one of two main features is creating interfaces in a procedural and formal way. React's second main feature is being component-based allowing for object-oriented-like code by encapsulating functionality and separating concerns in React's own JSX.

RFC 7519

RFC 7519 is an open standard used for defining the design, implementation, and functionality of JSON Web Tokens. The standard defines the transmission of data, algorithms used for data verification and authenticity, and the structure of JSON Web Tokens.

Flask

Flask is a micro web framework written in Python. It includes a simple and easy-to-extend core and supports extensions to add capabilities such as database support to the application. OPF will utilize the Flask framework to collect data from the SQLite database and for its backend development.

Python

Python is an interpreted high-level general-purpose programming language. Python is dynamically-types and garbage-collected and supports multiple programming paradigms, including structural, object-oriented, and functional programming. OPF will utilize Python for its Flask framework and its backend development.

HTML

HTML is a specification that defines the syntax, application programming interfaces (APIs), implementation, and usage of the language HTML to create web documents used by web applications. HTML is commonly characterized by its markup syntax to define the overall layouts of web pages.

ISO/IEC 9075

The ISO/IEC 9075 standard defines the structure, implementation, and expected functionality used by the Structured Query Language (SQL). The ISO/IEC 9075 standard is most commonly used by commercial and non-commercial database management systems to define queries made to a database.

7.0 Project Impact and Context Considerations

Optimum Property Fix (OPF) will correct the issues from UNR's current system while serving the same functionality of creating maintenance request tickets from the dormitory residences to the Facilities Services. Functionalities would be the ability to visually keep track of their request tickets, allowing students to identify the status of their tickets and Facilities Services to organize. Furthermore, students can provide feedback for the tickets, allowing communication between the two parties regarding the maintenance requests. There is a frequently asked questions (FAQ) page for residences to access and a chatbot for guided page navigation to that page. Specifically for the admin users, there is an analytics page containing data about each dormitory, allowing each building's health to be monitored. Additionally, administrators will be able to view preventive maintenance of buildings. These valuable functionalities as part of OPF will reflect the significant impact on the public's needs in terms of health, safety, or welfare in global, cultural, environmental, or economic contexts.

OPF's purpose and integration will allow for beneficial environmental impacts starting with replacing the current system of the university. Getting rid of paper trails and replacing them with OPF will not only solve lost reports but will also reduce paper waste, keeping the best interest in mind for the environment. In addition, creating and receiving maintenance requests will be easier to access for both students and the facility, leading to more awareness of any issues that arise in the buildings. As more problems are identified and reported, better and continuous maintenance of buildings means ensuring the safety of students. In a global context, if OPF displays success within the university, it may expand to other universities across the globe, ensuring maintenance within their own dormitories and offering optimum service. With better and constant maintenance, buildings may last longer and stronger, if kept consistently. With this, society will be motivated to take better care of their buildings through OPF because it is cost-effective. By eliminating the need to build new dormitories, expenses are cut from operational costs and there won't be much of a need to constantly invest and build new housing.

The environmental impacts mentioned, coincide with ensuring the safety of the public. For example, OPF introduces an efficient alternative to submitting maintenance requests, meeting the usage requirements. The user-centered design of OPF promotes ease of use and makes sure that the user's goals are met successfully. In this case, the goal would be solving the maintenance requests that are submitted from the user, maintaining the building's condition, and preventing

damages such as explosions or hazards. With OPF's easy-to-learn and effective design, maintenance requests are easily submitted, less likely to be lost, and attended to right away as facilities receive them. This prevents health and safety risks and keeps equipment, machines, and the environment safe and reliable.

8.0 Updated List of References

Problem Related Websites

1. "Facilities Services." *University of Nevada, Reno*, <https://www.unr.edu/facilities/>.

The facilities services section in the University of Nevada, Reno website demonstrates the current property management utilized by the facilities services. It describes the functionalities and resources, communication methods provided by the facilities services now. It is also responsible for the operation and maintenance of the University of Nevada, Reno, and campus buildings and provides guidance and understanding in the areas of building maintenance, moving furniture, housekeeping, and grounds landscaping.

2. "Free Maintenance Ticketing System Setup [Templates Included]." *Limble*, 14 May 2021, <https://limblecmms.com/blog/free-maintenance-ticketing-system/>.

Website describing the setup process of a maintenance ticketing system. It describes the lifecycle of the maintenance ticket, the requirements for implementing a digital maintenance ticketing system, and the basic functionalities utilized by a maintenance ticketing system. It also demonstrates a template to create and manage a maintenance ticket.

3. "Healthy Buildings, Healthy People - A Vision for the 21st Century." *EPA, Environmental Protection Agency*, <https://www.epa.gov/indoor-air-quality-iaq/healthy-buildings-healthy-people-vision-21st-century>.

The U.S. Environmental Protection Agency provides a guide to create a healthy indoor environment and expands on the importance of design, ventilation systems, materials, and products that can cut down on energy usage. The website outlines goals for existing buildings such as government, academia, and industry.

4. Sinha, Pritibhushan. "Towards Higher Maintenance Effectiveness: Integrating Maintenance Management with Reliability Engineering." *The International Journal of Quality & Reliability Management*, vol. 32, no. 7, Emerald Group Publishing, Ltd, 2015, pp. 754–62, <https://doi.org/10.1108/IJQRM-03-2013-0039>.

The article provides a guide for enhancing maintenance effectiveness. Moreover, the article discusses the management of maintenance and the approaches to reliability engineering and

maintenance enhancement. Lastly, the article reveals factors that can improve maintenance in a practical sense and decrease maintenance costs.

Problem Domain Articles

1. Dzulkifli, Nur'afini, et al. "Review on Maintenance Issues Toward Building Maintenance Management Best Practices." *Journal of Building Engineering*, vol. 44, Elsevier Ltd, 2021, p. 102985–, <https://doi.org/10.1016/j.jobbe.2021.102985>.

This article focuses on best practices concerning building maintenance and ISO standards. Additionally, the article highlights the importance of planning and management while addressing technology issues surrounding building maintenance. Lastly, the article discusses the problems surrounding deteriorating buildings, neglecting damages, and natural causes.

2. *Housing Manager's Procedures Manual - HUD* | *Hud.gov* / U ... Nov. 2005, https://www.hud.gov/sites/documents/DOC_9211.PDF.

This manual provides housing managers with procedural manuals based on management structure; consistent with norms in private housing. Housing managers are able to use the manual to decide what model best meets the needs of each property. It acts as a guide for the housing managers in administration to learn and perform their duties and responsibilities.

3. Iveta Pukite., "Different Approaches to Building Management and Maintenance Meaning Explanation." *Procedia Engineering*, Vol. 172. 2017.

Journal paper describing organizational approaches to property management overview economic operations, problems and solutions to repairs and maintenance, and key insights to effective property management. Building management systems are imperative to ensuring upkeep savings by being preemptive to issues that may arrive.

4. Rains, Harriet., "Federal Real Property: Management Issues of Structures, Historic Buildings, and Underutilized Property." New York: Nova Publishers, 2014. Print.

Journal article assessing federal government's efforts of property management with solutions to particular points of failure. Some comprehensive assessments are of government-provided housing and property management solutions that can be used in cost savings. The overarching goal to ensure solutions is proper tracking of issues.

Problem Domain Books

1. Haupt, Kathryn, et al. *Property Management*. 3rd ed., Rockwell Publishing, 2021.

This book is a reference for property managers for topics such as working with management clients, management planning, marketing rental properties, tenant selection, and risk

management. Moreover, the book is able to provide knowledge of maintenance requests and how to solve this problem in the industry as well as an internet-driven society.

2. Kyle, Robert C., et al. Property Management. Kaplan Real Estate Education, 2013.

This book provides an overview of the field for real estate practitioners and/or managers. The book covers topics such as professional property management, property management economics and planning, and environmental issues. Additionally, the book provides a guide regarding everyday issues with maintenance, administration, and accounting by stating how to solve these specific problems in regards to different properties such as apartment complexes, single-family homes, office buildings, retail properties, and government-related properties.

9.0 Contributions of Team Members

Araam Zaremehrijardi's Contribution

Araam Zaremehrijardi's total time worked on the revised specification and design totals 4 hours. The contributions include term definitions for Engineering Standards and Technologies and Programming Units for OPF's frontend.

Joanna Lopez's Contribution

Joanna Lopez's total time worked on the revised specification and design totals eight hours. The contributions include editing of the document, adding to problem domains, adding glossary terms, and all of section four in conjunction with Araam.

Melissa Flores' Contribution

Melissa Flores's total time worked on the revised specification and design totals ten hours. The contributions include the creation and redesign of the web application's user interface. Additionally, contributions include the project impact and context considerations section.

Nasrin Juana's Contribution

Nasrin Juana's total time worked on the revised specification and design totals seven hours. The contributions include updated specification in conjunction with Aisha Co, adding five terms in the updated glossary of terms, adding three references in the list of references, and part of the engineering standards and technologies.

Aisha Co's Contribution

Aisha Co's total time worked on the revised specification and design totals to nine hours. The contributions include updating specifications alongside Nasrin Juana, adding five terms to the glossary, and editing the document alongside Joanna Lopez.