**Software Design Specification**

**for**

**Vacation Tracking System**

**Version 1.0**

**Prepared by: Yuki Lieu, Aimee Valladares, Ramnik Bharath**

**CEN 4065 - Software Design and Architecture**

**Florida Polytechnic University**

**December 7, 2021**

**Table of Contents**

[**1. Introduction**](#_etpcncggidd2) **2**

[1.1 Purpose](#_ftit76pkzyem) 2

[1.2 System Overview](#_5eo634w7pmw7) 2

[**2. Design Considerations**](#_8795mxb2fzt2) **3**

[2.1 Assumptions](#_7r5zqv8ak1la) 3

[2.2 General Constraints](#_22re8nxr844) 3

[**3. System Architecture**](#_546cok1xt0gj) **4**

[**4. Detailed System Design**](#_tuot6q88apkm) **7**

[4.1 Class Diagram](#_ay38gub9abeg) 7

[4.2 Activity Diagram](#_pyxb4ocyza92) 8

[4.3 UML Message Sequence Diagram](#_jaxf1k2bb5vb) 19

[4.4 UML Statechart Diagram](#_4c94u4qyvn5n) 21

[4.5 Algorithms for Components/Methods](#_ufdxht1xuv80) 23

[4.6 Database Design](#_aozsrxmxtx48) 23

[**5. User Interface Design**](#_105xozio2vcf) **33**

[**Appendix A: Glossary**](#_5cthzfs4g8m2) **38**

# **1.** **Introduction**

## **1.1** **Purpose**

The purpose of the Vacation Tracking System (VST) system is to allow employees to manage their own vacation time, sick leave, and personal time off while abiding company policy or local facility’s leave policies. The application improves the process to manage time off requests for multiple departments, making the process more efficient. The application will include a rules-based validation system, a notification system, and have multiple interfaces for the various personnel, departments, and systems. The system design will utilize and apply client-server, publish-subscribe, and repository architectural styles. The system will include in-depth activity, message sequence, statechart, class, and entity-relationship diagrams to demonstrate how the system will operate.

## **1.2** **System Overview**

The system design will be in-depth as it utilizes several diagrams and prototypes to display the requirements, key features, and workflows. The system design will utilize and apply client-server, publish-subscribe, and repository architectural styles. The system has several use cases and scenarios to sufficiently describe how the system is meant to work. The system design has a detailed class diagram that defines the classes and objects in the system, defines the attributes and operations of the classes, and depicts the relationships between classes. The system design utilizes several in-depth activity diagrams to help demonstrate the flow of control from activity to activity for the use cases. The system design also utilizes several in-depth sequence diagrams to help demonstrate how objects interact with each other to support the use cases. The system design also utilizes statechart diagrams to help model the state of a single object, what causes transitions, and the actions resulting from change. The system also utilizes an entity-relationship diagram for the database design to help model the data stored in the databases. Finally, the system utilizes wireframes and prototypes to describe and directly support the construction of the user interface screens.

# 

# 

# 

# **2.** **Design Considerations**

## **2.1** **Assumptions**

Some assumptions we had with the design process is that we wanted the application to be a simple process for not just the employees, but also the managers and H.R. The program should be designed in a way where it would not require any additional background knowledge to know how to use the application and should be straightforward for the user.

Some other assumptions with the design process is that some days may need to be blocked off in the system if too many employees requested time off, so it’s important to create a system that is not only functional, but also realistic with giving the requested time off as needed.

The PTO should also be linked to some sort of employee calendar that is able to track which employees have which day off. It’ll be helpful to know not just from H.R’s perspective, but also for all the other employees as well.

A flowchart hierarchy can also be assumed for the design process, as the vacation tracking system has to go through a set of people before being able to put in the request. Depending on the status of the employee, it would be then able to determine who to send the request to.

## **2.2** **General Constraints**

Some general constraints present with the design consideration would be that the functionalities of the application would be limited to that of the VTS system which would mean that to what extent the application can run would be limited to that particular software/hardware.

The application could also have setbacks where it would need to be able to run through an employer’s network or VPN depending on the application, as it may not run on a private network such as at home without company authentication through an employee portal. This is where verifying the employee is crucial and needs to be done by allowing them to login first before a request can be sent.

For some other constraints, as a developer one has to consider how much PTO a user has already asked for before and then adjust that accordingly. This is where following the employee hierarchy in the company is important.

# **3.** **System Architecture**

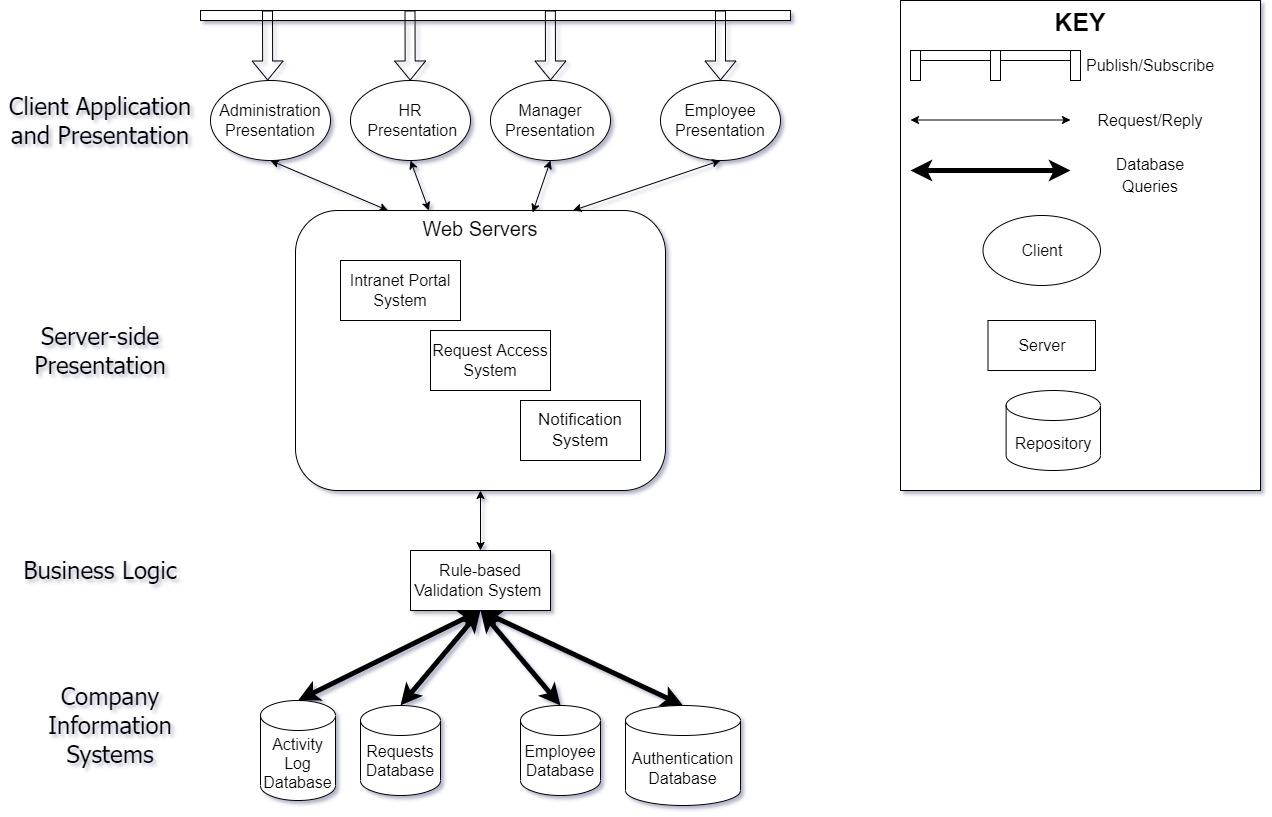
For the high-level architecture design, we applied the client-server, publish-subscribe, and repository architectural styles. Client-server architectural style is used to illustrate the interaction between the client and server components that is meant to improve the system’s modularity.

The client presentation would represent the application for specific personnel namely administration, HR, managers, and employees. Each of the client presentations and applications represent each type of user interface and the specific actions and functions that each user can accomplish. For instance, the employee user interface will not be the same as the HR user interface as HR personnel have the ability to override actions whereas employees do not.

While the server components are used for the server-side presentation and business logic. Specifically the web servers would involve the authentication system, request access system, and the notification system. For the authentication system it would handle connecting to the existing intranet portal system and utilize the portal’s single-sign-on mechanisms for user authentication. The request access system that handles the process of accessing previous requests, creating future requests, and creating forms for certain requests. The notification system that handles request approval and status changes. Additionally, the application would have a rules-based system that is used for validating and verifying leave time requests.

Meanwhile, the system would utilize the publish-subscribe architectural style specifically for the system design of the notification system as it involves interacting with the employee and managers. Additionally, employees, managers, and HR should receive request status changes in real-time. This will help connect the notification system server and all of the client applications.

Finally, the system design also encompasses the repository architectural style because of the several databases utilized by the application. One database is the activity log database that would keep track of the transactions. Another database is the requests database which handles all the information relating to any requests made. Another database is the Employee database that has employee information. Finally, there is the Authentication database that contains the personnel login-in information that allows access through the portal system’s single sign-on mechanism.



**Figure 1. High-Level System Architecture Diagram**

Reliability is the probability of the software executing without failure for a specific period of time. Reliability issues can occur because of improper inputs, errors in the code itself, components that are not available when needed, and hardware failures. This is a major design issue relevant to the project as the system must be able to run with little to no failure as it is a system used by countless personnel (employees and managers) and must be able to deal with risk management. For instance, minimizing data loss in case of failure and limited communication occurring in the workplace due to workload.

Reusability is the indicator of the relative effort required to convert a software component to other applications. Reusability issues can occur due to modularity, documentation, the environment, and capability.

Maintainability is the ease at which a software’s code is understood, repaired, or enhanced. Maintainability issues can occur in the system due to incorrect documentation and no refactoring. This is another major design issue relevant to the project as the system should be able to be understood, enhanced, and repaired as it is used to help streamline the functions of the HR department and must be understood as the desired functionality must be considered in the context of any existing or proposed systems.

Testability is the ease at which a system or system unit can be tested and verified. The system can experience testability issues due to the requirements, subsystem, or other components not being verified as acceptable or not resulting in many unresolved faults and malfunctions.

Performance represents the responsiveness of the system to various inquiries and actions. For this system, performance would involve the response time for the queries to display results. Additionally, the system performance aspects include data capacity (Maximum amount of data stored in a database), dynamic capacity (Maximum number of concurrent users of the system), and latency (the amount of time it takes for the data to be processed). Performance issues the system can experience are imprecise response time and latency. This is another major design issue as the system must be able to meet requirements in a timely fashion. The system response time should not take longer than 10 seconds. People using this system are busy people and they can get frustrated.

Portability is the measure of effort needed to migrate software from one operating environment to another. Portability issues can occur due to what portions of the product are movable and localized as well as what platforms need support which affects the environment.

Security deals with blocking unauthorized access to system functions or data, ensuring that the software is protected from malware attacks. Security issues can arise due to the internet opening issues like malware, phishing, and denial of service. This is another major design issue as the system must be able to protect against unauthorized access to the application and its data. For instance, the system should not compromise or hinder the security of the existing authentication portal system. Security is important due to the several servers and databases.

Safety deals with the need to prevent a system from doing any injury to people or damage to the property. Safety issues can occur due to the system not following business rules.

# 

# 

# 

# 

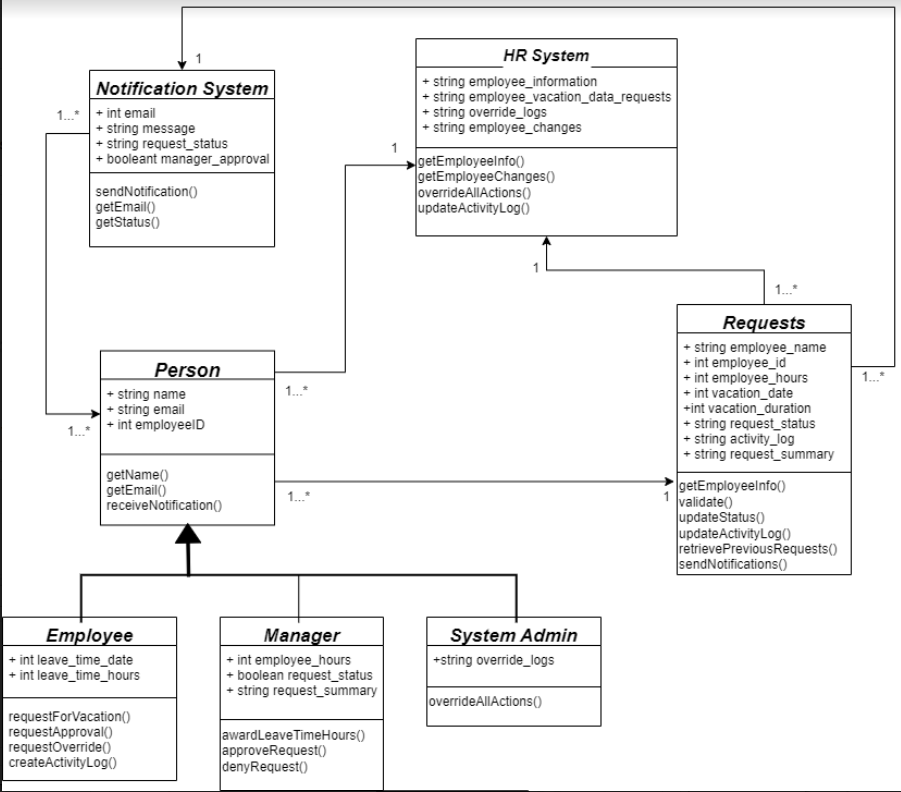
# **4.** **Detailed System Design**

*This section provides detailed system design.*

## **4.1** **Class Diagram**

The class diagram has 7 classes in total: Notification System, HR System, Requests, Person, Employee, Manager, and System Admin. There is inheritance within this system; the base class is Person, while the derived classes are Employee, Manager, and System Admin. In the base class, Person, the classes commonly share a name, an email, and an employee ID. The shared methods include getName(), getEmail() and receiveNotification().

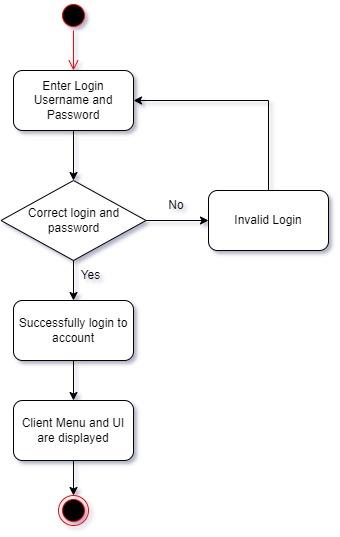
One person or many people, employees, can call to Requests by requestForVacation() function. Requests will gather employee information through getEmployeeInfo() and update the activity log to keep track of any transactions through updateActivityLog() and retrievePreviousRequests(). Requests will then update the status of the transaction through updateStatus() and verify the request through validate(). To confirm the status of the transaction, Requests uses sendNotification() to send a notification to the Notification System. The Notification System then takes the email (getEmail()) and status (getStatus()) and combines the two into a message to send to the Manager. The Manager can approve or deny the request and sends the response back to the Notification System. The Manager can also award personal leave time hours through awardLeaveTimeHours(), which updates the int employee\_hours. The hours are included when getEmployeeInfo() is called. The Notification System gets the status of the request through getStatus() and sends it back to the Requests class for this class to updateStatus() and updateActivityLog(). The Notification System makes a new message to notify the employee of their approved or denied status. The HR System is able to access and update the Activity Log and employee info. Through the HR System, an HR employee can override any actions. Additionally, System Admins also have power to override any actions through overrideAllActions().



**Figure 2. Class Diagram**

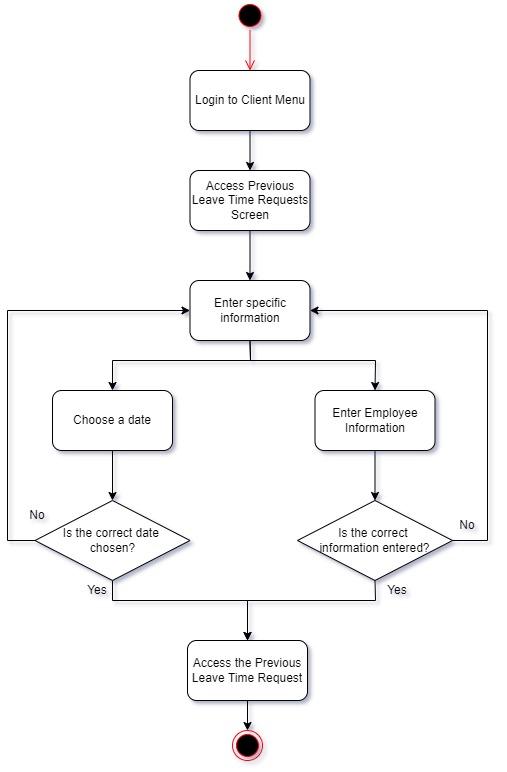
## **4.2** **Activity Diagram**

The Vacation Tracking System (VTS) design involves seven detailed activity diagrams for the essential use cases that address several key features and actions that can occur in the system. The first activity diagram handles the workflow for logging into the system. The activity diagram demonstrates a use case that applies to all types of users as everyone accessing the system must log in to the existing intranet portal system using their credentials. When the user enters their login username and password, the system will check if the information entered is correct and in the system. If the information was correct then the screen will demonstrate that the user logged in successfully and the client user interface will be displayed. If the information entered is incorrect, then the screen will display the information that was not correct and invalid. The system should then bring the user back to the login screen so they may be able to enter their information again.



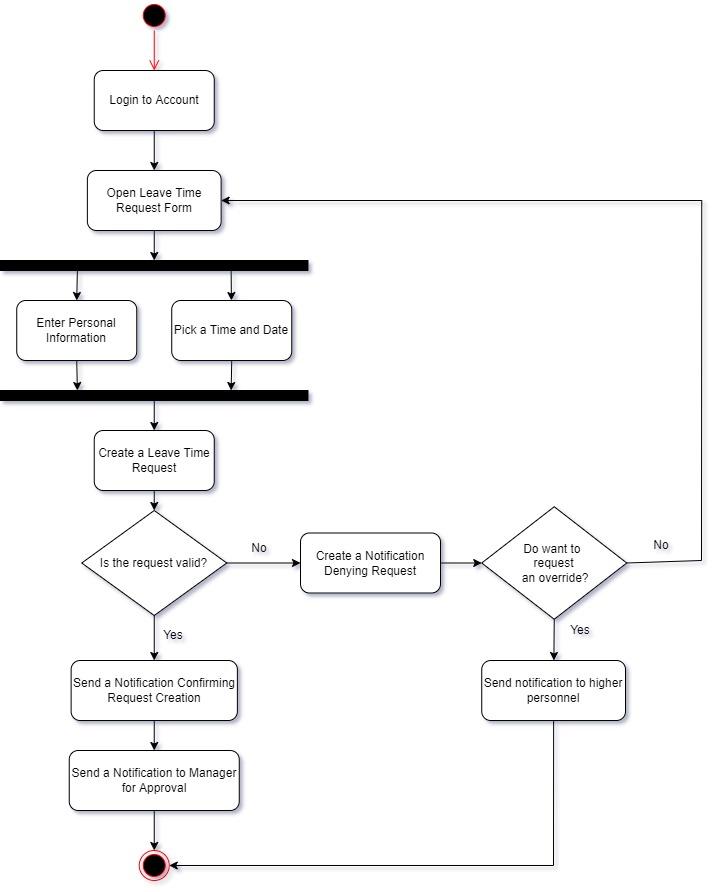
**Figure 3. Activity Diagram for the Login Use Case**

The second activity diagram handles the workflow for access to previous requests made in the previous calendar year. The activity diagram demonstrates a use case that applies to multiple types of users as users with a certain level of approval can access previous requests. After the user logins into the system, the user can access previous leave time requests. Upon clicking the button the user may choose a date or enter an employee’s information (like their name or ID number). The system will then check if the information is correct. If the correct information was entered or chosen then the user will access the specified request. If the correct information was not entered or chosen, the user will be taken back to the previous screen. The user can then try to enter the correct information again.



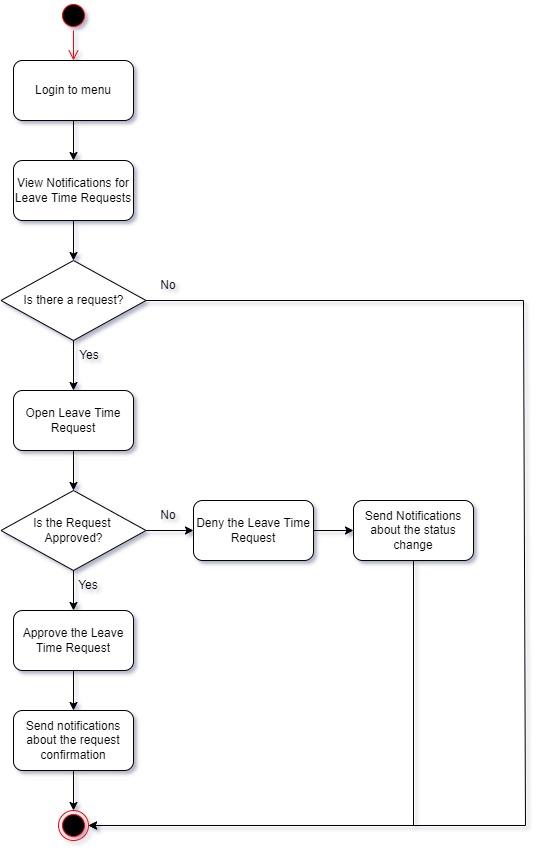
**Figure 4. Activity Diagram for Accessing Previous Leave Time Requests**

The third activity diagram handles the workflow for creating a leave time request. The activity diagram demonstrates the most common use case that primarily applies to employees as they usually just send the request as the first option to get time off for any reason. After the user logs into the system, the user can then open a leave time request form. The form will prompt the user to enter their information and pick the time and date for the amount of leave time. All of the entered information must be entered to proceed. Afterward, the system will then try to create the leave time request but the system checks if the request is valid. Specifically, the system will check if the request is valid and verified. If the request is valid then the system will send a notification to the user that the request was successfully created and submitted. The system will then send a notification to the manager to state that a request needs approval. If the request is not valid then the system will create and send a notification of the request denial. The system will prompt the user and ask if they need an override to create their request. If the user does want an override, then the system will send a notification to higher personnel like HR and the system administrator asking for an override. If not, the user will be taken back to the screen asking for the information.



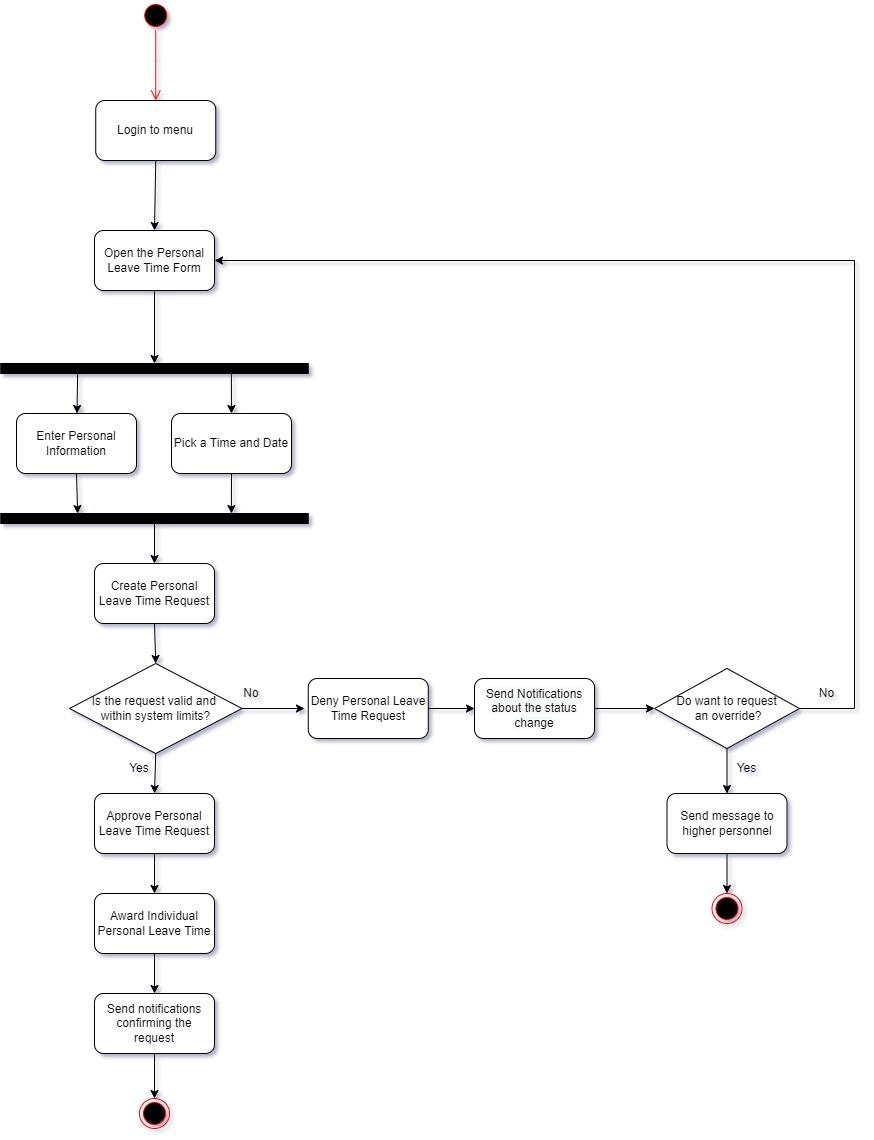
**Figure 5. Activity Diagram for Creating a Leave Time Request**

The fourth activity diagram handles the workflow for manager approval. The activity diagram demonstrates a common use case that applies to managers as they are usually the people that approve the most requests. After the user logs into the system, the manager may receive and view notifications for leave time requests that need approval. If there is not a request then the manager can continue with what they were doing. If there is a request, then the user can open the leave time request. The user can then read and evaluate the request and choose to approve it. If the user chooses to approve the request, then they can approve it and the system sends notifications about the request confirmation. If the user does not approve the request, then the user can deny the leave time request and the system then sends notifications about the status change.



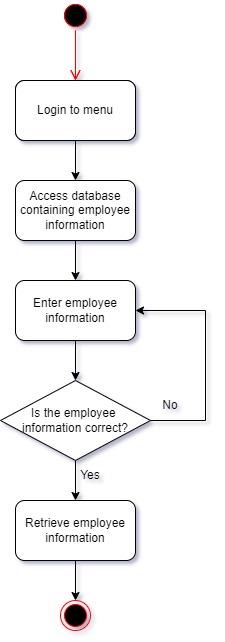
**Figure 6. Activity Diagram for Manager Approval**

The fifth activity diagram handles the workflow for awarding personal leave time. The activity diagram demonstrates a use case that applies to managers as they have the ability to directly award personal leave time (within system-set limits). Once the user logs in to the system, they can open a personal leave time form. The user then needs to enter the personal information of the employee who is being awarded the personal leave time and pick a time a date for how long the leave time is. All of the information must be entered before continuing the process. Once the user enters the necessary information, they may try to create the personal leave time request. The system will then check if the request is valid and within system-set limits. If the request is valid, then the system approves the personal leave time request and then awards the individual their personal leave time (which should be seen in their user interface). Finally, the system will send notifications confirming the request. However, if the request is not valid then the system will deny the personal leave time request and send notifications about the request’s status change. The system will prompt the user and ask if they request an override for the leave time request. If the user does request an override, then the system will send a message to higher personnel like HR or a system administrator. If the user does not want to request an override. Then the system will take the user back to the screen that prompts the user to enter the necessary information again.



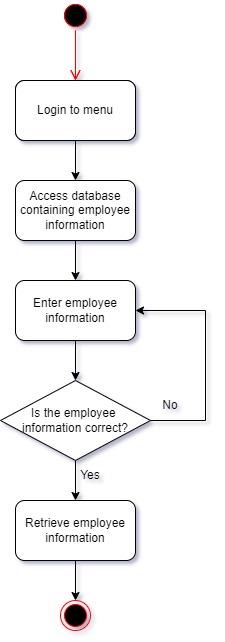
**Figure 7. Activity Diagram for Awarding Personal Leave Time**

The sixth activity diagram handles the workflow for overriding actions that are restricted by rules. The activity diagram demonstrates a use case that applies to personnel like HR and system administrators as they have the ability to override actions restricted by the rules-based system and allow the leave time request to be awarded. Once the user logs in to the system, the user may receive and view notifications for override requests that need approval. If there is not a request then the administrators can continue with what they were doing. If there is a request, then the user can open the override request. The user can then read and evaluate the request and choose to approve it. If the user chooses to approve the request, then they can approve it and the system sends notifications about the request confirmation. If the user does not approve the request, then the user can deny the leave time request and the system then sends notifications about the status change.



**Figure 8. Activity Diagram for Overriding Actions**

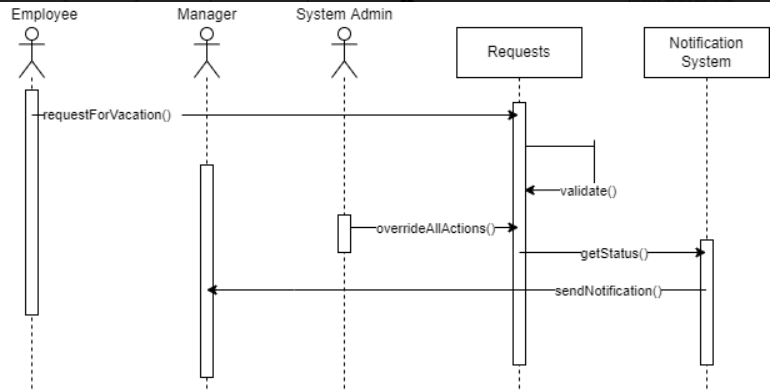
Finally, the seventh diagram handles the workflow for accessing employee information and changes. The activity diagram demonstrates a use case that applies to HR as their interface is integrated with the HR department legacy systems and they have the ability to retrieve the required employee information and changes. Once the user logs in to the system, they can then access the database containing employee information. The user can then enter the employee’s information like their employee ID or name. The system will then check if the employee information is in the system or if the correct employee was picked. If the information is correct, then the user can retrieve the employee information and the changes that have occurred. However, if the information is incorrect then the user can go back to the screen and try to enter the employee information again.



**Figure 9. Activity Diagram for Accessing Employee Information and Changes**

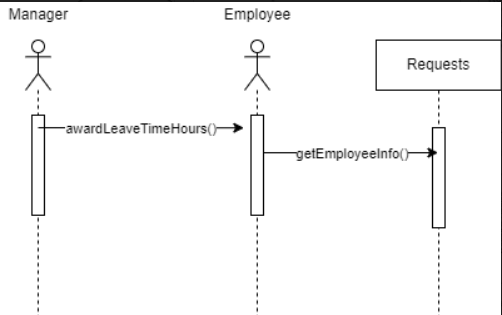
## **4.3** **UML Message Sequence Diagram**

The first UML Message Sequence Diagram depicts a scenario where an employee requests for leave time through requestForVacation(). This is sent to Requests where it will validate() the request before proceeding; the validate function will determine if the employee has gone over the hours limit. The System Admin is able to override any actions with overrideAllActions(). Assuming the request is verified, Requests getStatus() to Notification System. The Notification System sends a notification of an employee’s leave time request to the Manager through sendNotification().



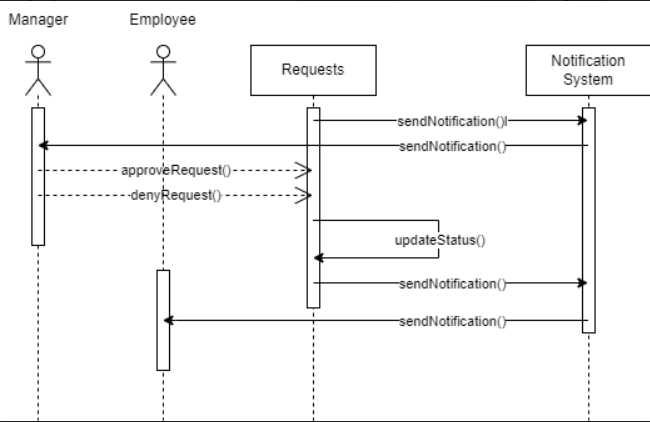
**Figure 10. UML Message Sequence Diagram for Requesting Leave Time.**

The second UML Message Sequence Diagram is for awarding personal leave time. The manager is able to award personal leave time hours through awardLeaveTimeHours() to an employee. The employee’s hours are retrieved when Requests is given the employee information through getEmployeeInfo() in order to validate a request.



**Figure 11. UML Message Sequence Diagram for Awarding Personal Leave Time Hours.**

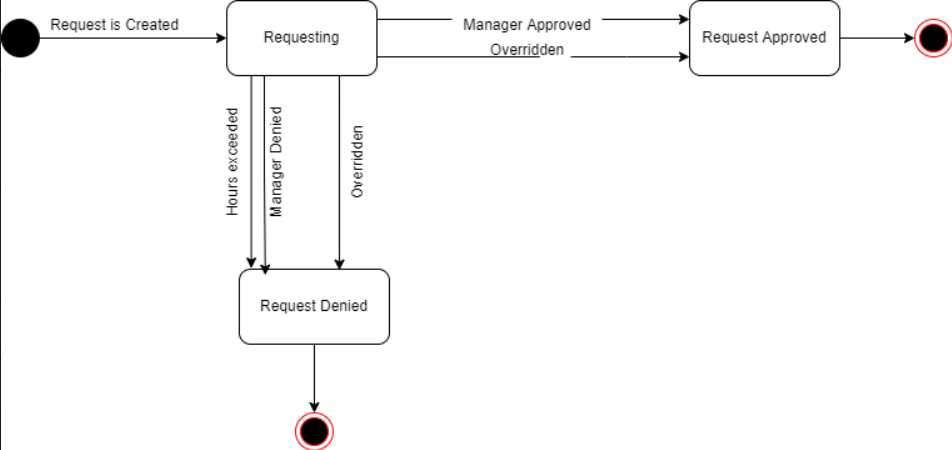
The third, and final, UML Message Sequence Diagram is for manager approval. Managers must approve or deny any leave time requests from the employees. The Requests class will send its completed request information and status to the Notification System through sendNotification(). The Notification System then sends a notification to the manager of the request through sendNotification(). The manager takes this request and either approves the request with approveRequest(), or denies the request with denyRequest(). The Requests class self-updates itself with updateStatus() to then send the status over to the Notification System with sendNotification(). The Notification System then finally sends a notification of the request result from the manager to the Employee with sendNotification().



**Figure 12. UML Message Sequence Diagram for Manager Approval.**

## **4.4** **UML Statechart Diagram**

When a request is created, the system will move into a requesting state. Here the requesting state is determining whether the state should move forward to “Request Approved” or “Request Denied.” If the manager approves or there’s an overridden exception, the request can be moved into an approved state. If the manager declines, there’s hours exceeded, or there’s an override, the request can move into a denied state.



**Figure 13. UML Statechart Diagram for Requests**

## 

## 

## 

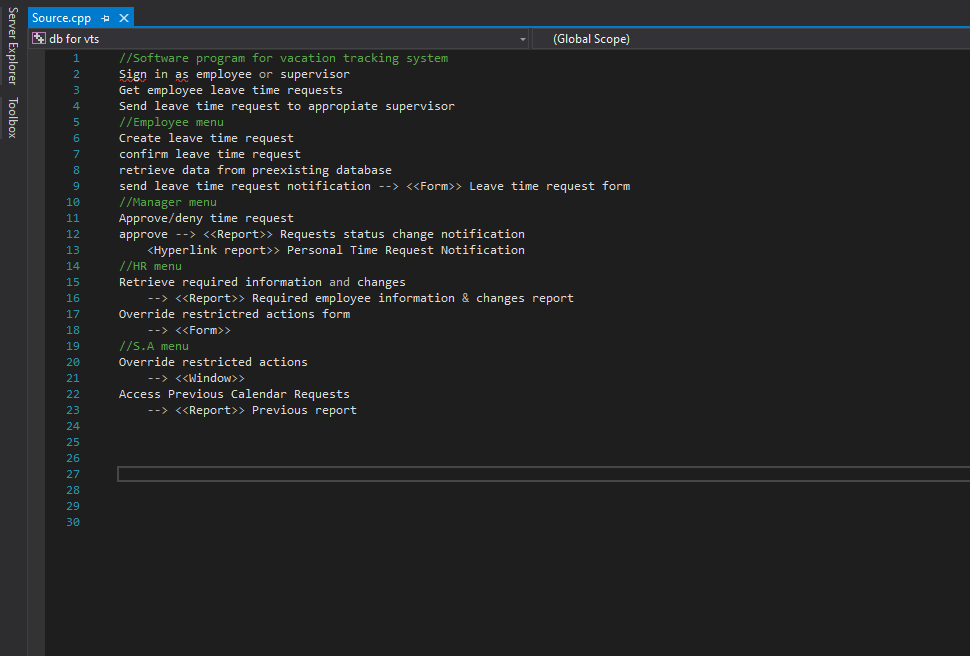
## 

## 

## **4.5** **Algorithms for Components/Methods**

Each section is a container method for the pseudo code below. The first one is the general software program that gives the option for the user to sign in as an employee or supervisor.

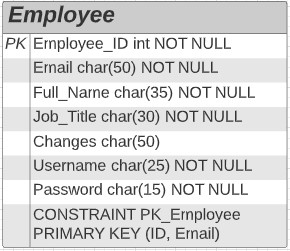
The algorithm that was used was a get-and-retrieve method that will create a new entry when prompted from the employee or manager and then retrieve the essential data that would be connected to a database. This information is then displayed through a U.I for the user to be able to interact with the application and modify or delete changes and reports as needed.



**Figure 14. Pseudocode for Algorithms for Components/Methods**

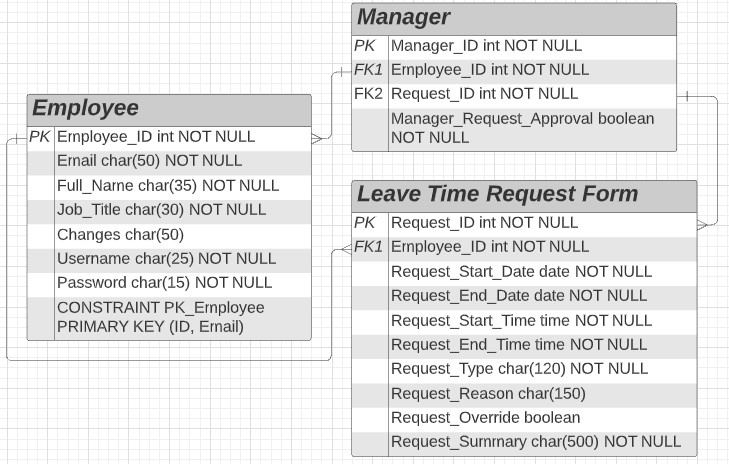
## **4.6** **Database Design**

The database design for the VTS system is extensive as it has 10 tables containing information regarding the different users (employee, manager, HR, and system admin), the various forms regarding leave time, and notifications. The *Employee* table is meant to represent an employee and contains several attributes that are meant to behold information for each employee. Specifically, the table contains attributes about the employee’s ID, name, job title, possible changes, email, and login information. There is only one primary key for the table as it acts as a unique identifier. However, there is a primary key constraint (PK\_Employee) that states the primary key value is made up of the ID and email attributes.



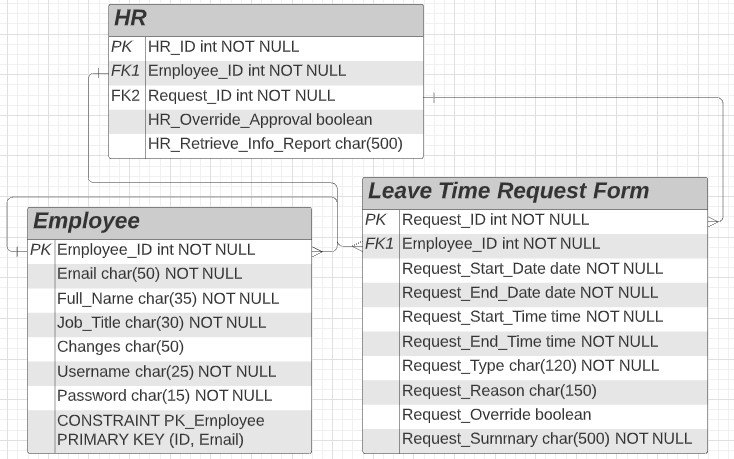
**Figure 15. Employee Table and Relationship(s)**

The *Manager* table is meant to represent a manager and contains a few attributes specifically for actions that a manager can do and may need a record of. The table contains attributes about the manager’s ID, references to the employee ID and request ID, and the approval value. The manager ID is the primary key for the table as it acts as a unique identifier. The employee ID is the foreign key for the table and refers to the primary key of the *Employee* table. The *Manager* table and the *Employee* table have a one-to-many (1:M) relationship as one manager has many employees reporting to them. The request ID is another foreign key for the table and refers to the primary key of the *Leave Time Request Form* table. The *Manager* table and the *Leave Time Request Form* table have a one-to-many (1:M) relationship as one manager is handling multiple leave time request forms.



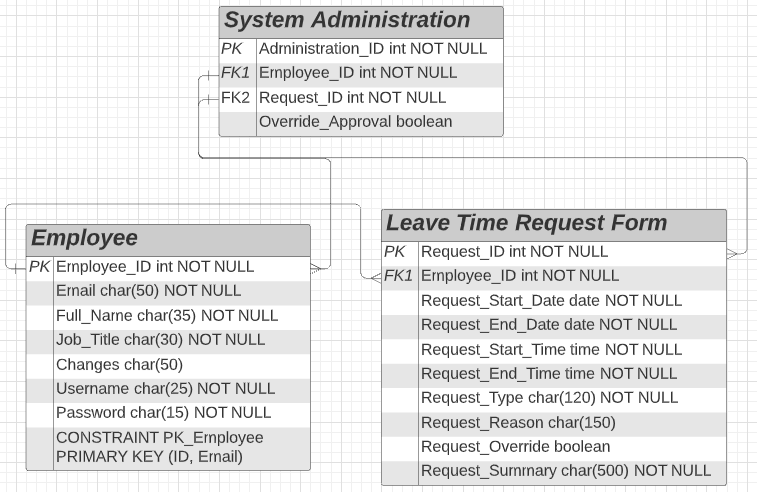
**Figure 16. Manager Table and Relationship(s)**

The *HR* table is meant to represent an HR representative and contains a few attributes specifically addressing the actions that an HR representative can do and may need a record of. The table contains attributes about a representative’s ID number, references to the employee ID and request ID, override status on a request, and a report containing employee information and changes. The HR ID is the primary key for the table and acts as the table’s unique identifier. The employee ID is the foreign key for the table and refers to the primary key of the *Employee* table. The *HR* table and the *Employee* table have a one-to-many (1:M) relationship as one HR representative is responsible for multiple employees and one employee. The request ID is another foreign key for the table and refers to the primary key of the *Leave Time Request Form* table. The *HR* table and the *Leave Time Request Form* table have a one-to-many (1:M) relationship as one HR representative is responsible for handling multiple leave time request forms.



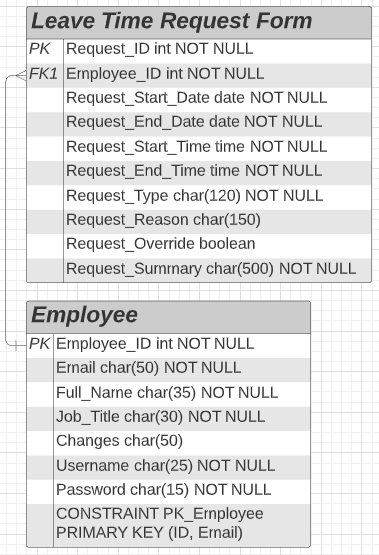
**Figure 17. HR Table and Relationship(s)**

The *System Administration* table is meant to represent a system administrator and contains a few attributes specifically addressing the actions that an HR representative can do and may need a record of. The table contains the administrator’s ID number, the override status on a request, and references to the employee ID and request ID. The administration ID is the primary key for the table and acts as the table’s unique identifier. The employee ID is the foreign key for the table and refers to the primary key of the *Employee* table. The *System Administration* table and the *Employee* table have a one-to-many (1:M) relationship as one system administrator is responsible for handling multiple employees (like a system issue). The request ID is the foreign key for the table and refers to the primary key of the *Leave Time Request Form* table. The *System Administration* table and the *Leave Time Request Form* table have a one-to-many (1:M) relationship as one system administrator will typically handle multiple leave time request forms.



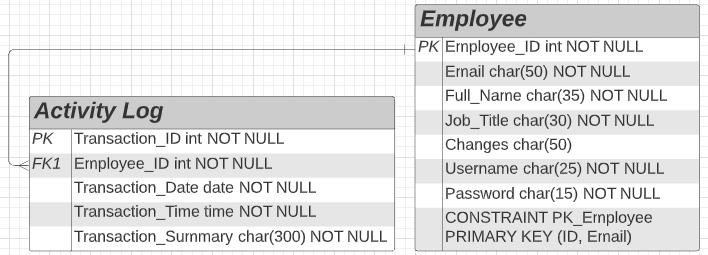
**Figure 18. System Administration Table and Relationship(s)**

The *Leave Time request form* table is meant to represent the form an employee needs to submit in order to request leave time. The table contains several attributes that represent the information involved in submitting the form. The table contains a request ID number, the request start/end date and time to state how long the leave time is, the type of leave time (i.e. personal time off or sick leave), a small description for the leave time reason, if the request required an override, a summary for the request, and a reference to the employee ID. The request ID is the primary key for the table and is the unique identifier. The employee ID is the foreign key for the table and refers to the primary key of the *Employee* table. The *Leave Time Request* form table and *Employee* table have a one-to-many (1:M) relationship as one employee can submit multiple leave time request forms.



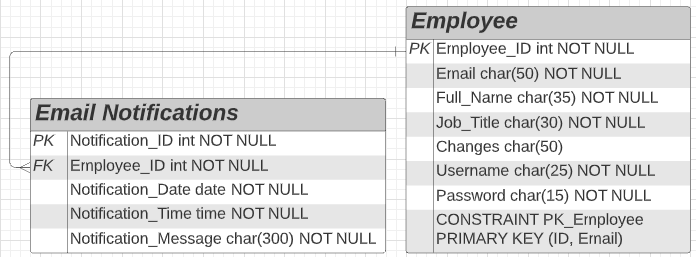
**Figure 19. Leave Time Request Form Table and Relationship(s)**

The *Activity Log* table is meant to represent the activity logs that are for all of the transactions that can occur. The table contains a few attributes that represent the information involved in each transaction. The table contains a transaction ID, the date of the transaction, the time of the transaction, the transaction summary, and a reference to the employee ID. The transaction ID is the primary key and unique identifier for the table. The employee ID is the foreign key for the table and refers to the primary key of the *Employee* table. The *Activity Log* and *Employee* tables have a one-to-many (1:M) relationship as an employee can have multiple activity logs as they complete several transactions.



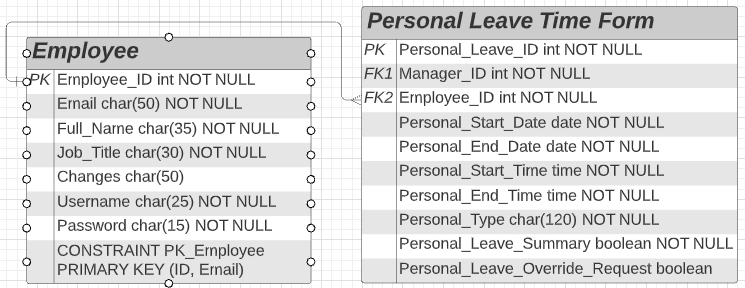
**Figure 20. Activity Log Table and Relationship(s)**

The *Email Notifications* table is meant to represent the email notifications that result from the various requests that can occur. The table contains a few attributes that represent the information involved in each notification. The table contains a notification ID, the date of when the notification was sent, the when the time of the notification was sent, the notification message, and a reference to the employee ID. The notification ID is the primary key and unique identifier for the table. The employee ID is the foreign key for the table and refers to the *Employee* table primary key. The *Email Notifications* and *Employee* tables have a one-to-many (1:M) relationship as an employee can have and/or send multiple email notifications.



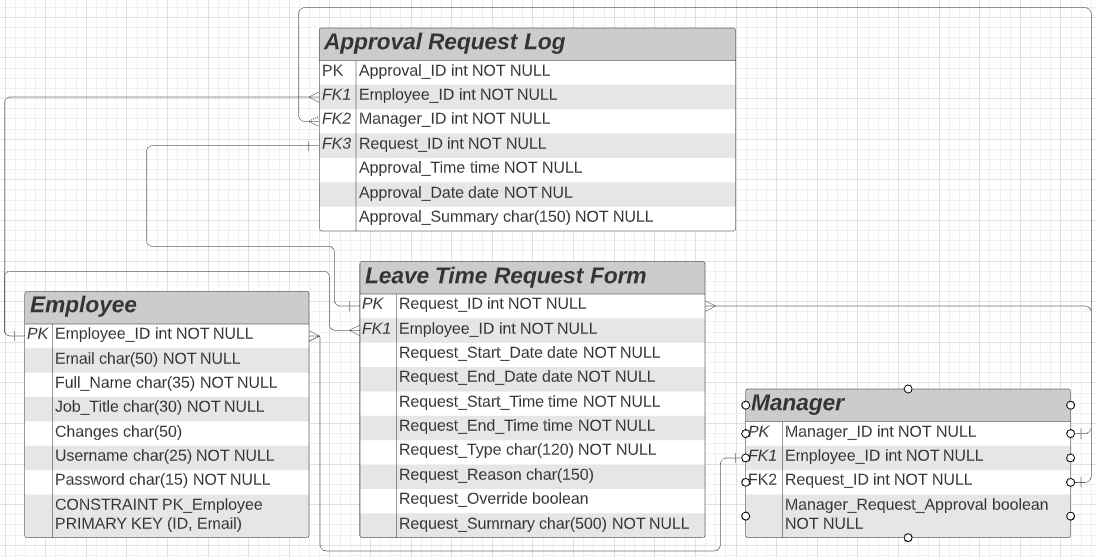
**Figure 21. Email Notifications Table and Relationship(s)**

The *Personal Leave Time Form* table is meant to represent the form that directly awards personal leave time to the employee. The table contains the personal leave time ID, the start/end date and time, the type of leave, a summary of the personal leave time, if the request asked for an override, and a reference to the employee ID. The personal leave ID is the primary key and unique identifier for the table. The manager ID is a foreign key for the table and refers to the *Manger* table primary key. The *Personal Leave Time Form* and *Manager* tables have a one-to-many (1:M) relationship as one manager can award multiple employees with personal leave time. The employee ID is a foreign key for the table and refers to the *Employee* table primary key. The *Personal Leave Time Form* and *Employee* tables have a one-to-many (1:M) relationship as an employee can be awarded personal leave time multiple times.



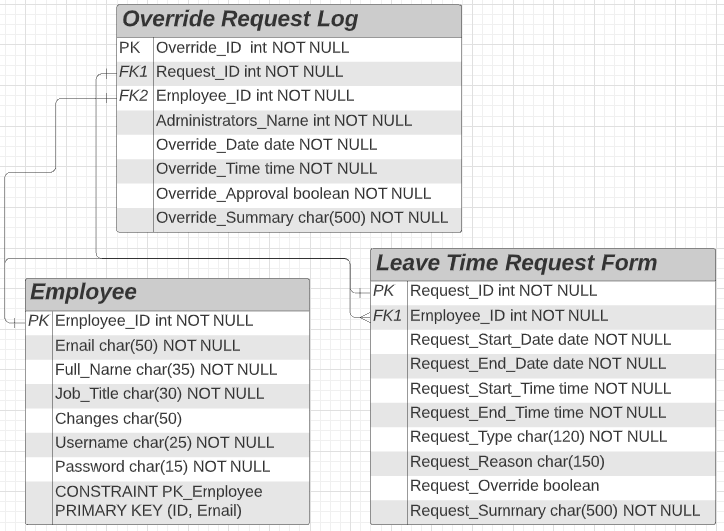
**Figure 22. Personal Leave Time Form Table and Relationship(s)**

The *Approval Request Log* table is meant to represent the logs created from an approval request by the manager. The table contains the approval ID, the approval time, the approval date, a summary of the approval request, and references to the employee, manager, and leave time request form tables. The approval ID is the primary key and unique identifier for the table. The employee ID is a foreign key for the table and refers to the primary key of the *Employee* table. The *Approval Request Log* and *Employee* tables have a one-to-one (1:M) relationship as an employee can receive one approval for each leave time request. The manager ID is a foreign key for the table and refers to the *Manager* table primary key. The *Approval Request Log* and *Manager* tables have a one-to-one (1:1) relationship as a manager can approve one leave time request at a time. The *Approval Request Log* and *Leave Time Request Form* tables have a one-to-one (1:1) relationship as one leave time request only needs one approval request.

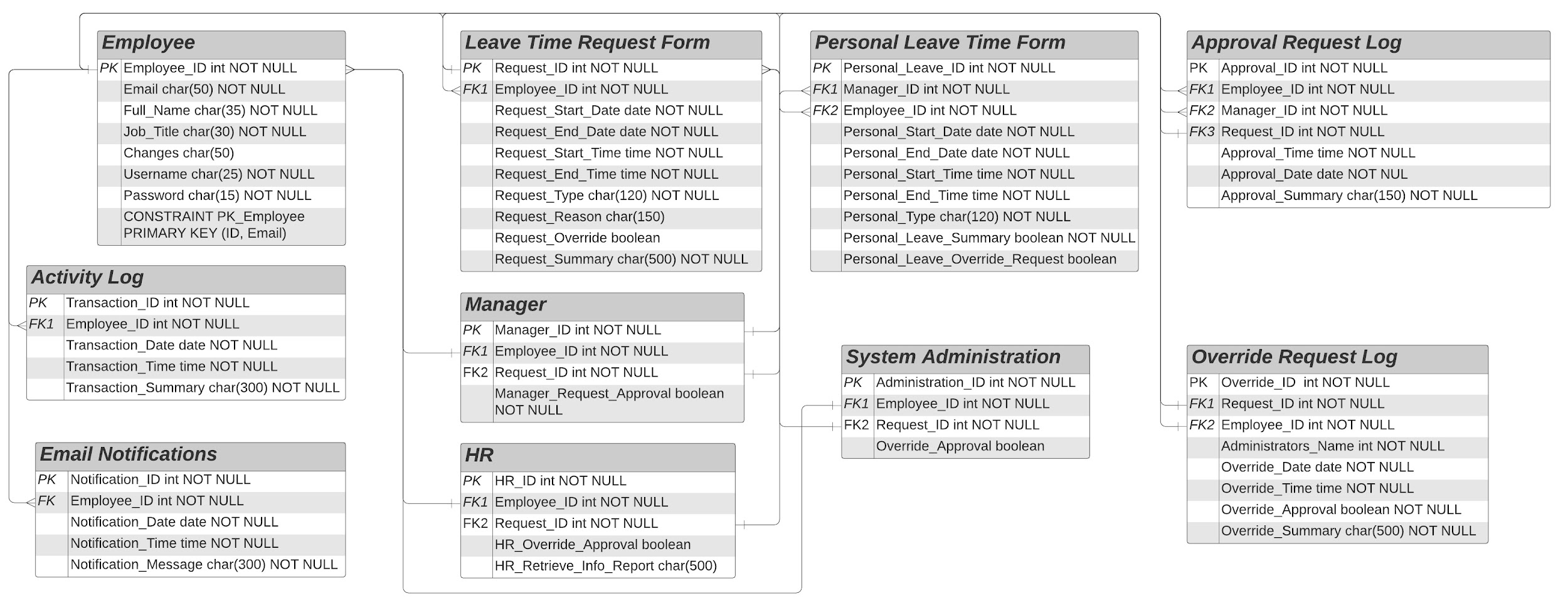


**Figure 23. Approval Request Log Table and Relationship(s)**

The *Override Request Log* table is meant to represent the logs created from an override request by administration personnel like HR and system admins. The table contains the override ID, the administrator looking over the request’s name, the date and time of when the override request was evaluated, whether the override request was approved, a summary of the override request, and references to the leave time request form and employee tables. The override ID is the primary key for the table. The request ID is a foreign key for the table and refers to the primary key of the *Leave Time Request Form* table. The *Override Request Log* and *Leave Time Request Form* tables have a one-to-one (1:1) relationship as one leave time request can have the rules be overridden once. The employee ID is a foreign key for the table and refers to the primary key of the *Employee* table. The *Override Request Log* and *Employee* tables have a one-to-many (1:M) relationship as an employee can try to ask for and/or receive multiple overrides for their leave time requests.



**Figure 24. Override Request Log Table and Relationship(s)**

****

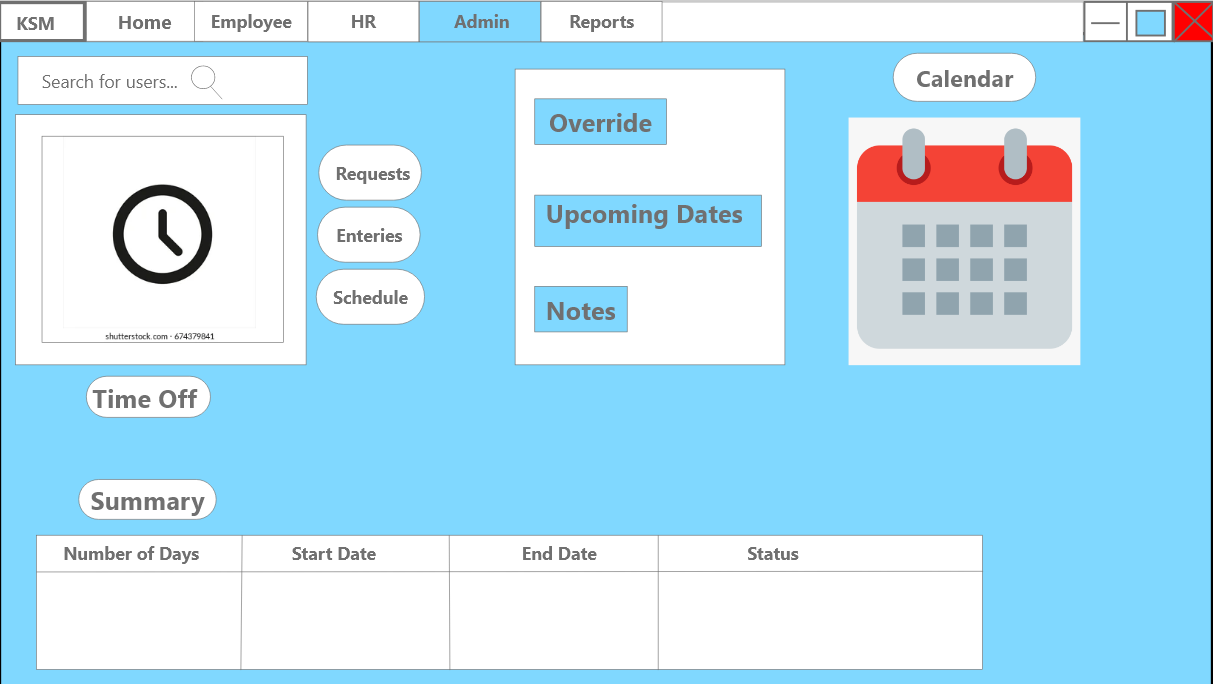
**Figure 25. Overall Database Design**

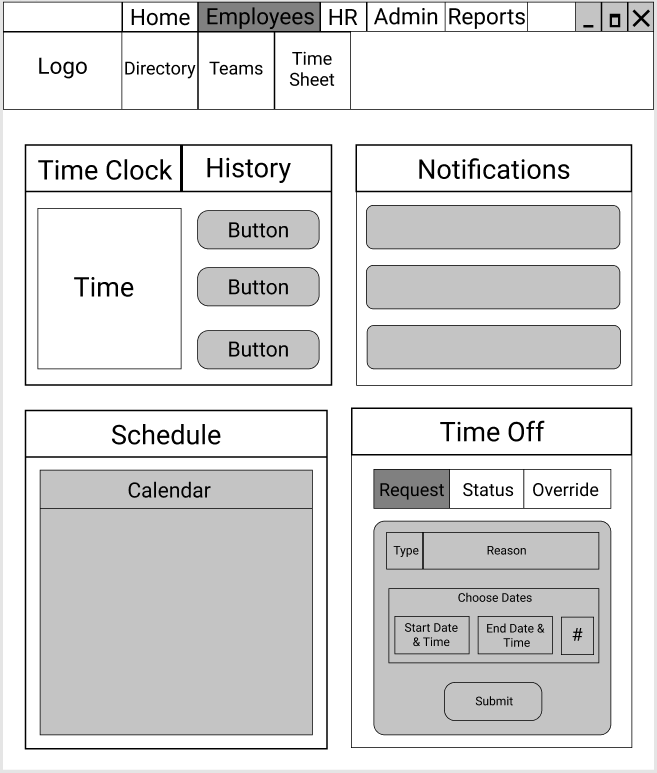
# 

# **5.** **User Interface Design**

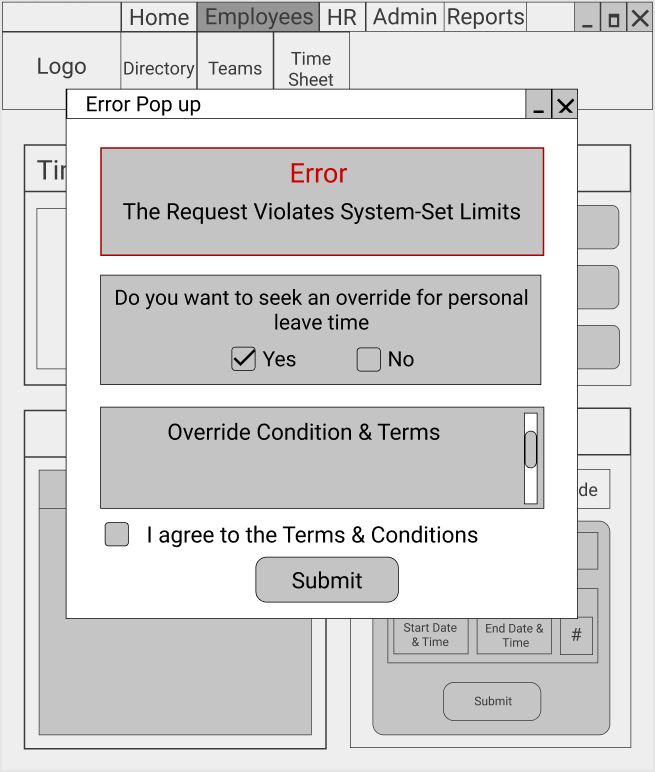
All of the user interfaces will have the same feel and design standards of other systems that the users may use since they follow the same design rules and guidelines that the company sets. Additionally, the user interfaces should be easy-to-use and easy-to-understand as different kinds of personnel are using the system. The different user interfaces do have some common behavior. First, all the screens have the same navigation bar as it is meant to navigate through the different kinds of level of access and common functionalities that all the users have. For instance, all of the user interfaces can have access to the company directory, a list of what teams they are working with, and their own time sheet of what hours they have worked. Another common behaviour all the screens share are certain menus they can use. For example, all of the users will have the same login screen as it utilizes a single sign-on mechanism for all of the authentication. Additionally, any user can access previous leave time requests made in the last calendar year.

Moreover, all the user screens should display the same pop-ups when entering certain information and error messages whenever there is a mistake in processing any kind of request. For instance, entering the date and time on a form should follow the same format for all forms and all users. Specifically, when entering the date on any form, the menu should bring up a calendar with certain days shaded out due to them being invalid. While entering the time in any form should bring up a clock. Additionally, when certain invalid information is entered, like entering a date that has already passed, then the system should have a pop-up error message that states what the issue was.

**Figure 26. High-Level Sketch Design of Admin UI**

****

**Figure 27. Screen Sketch and Wireframe of Employee UI**

****

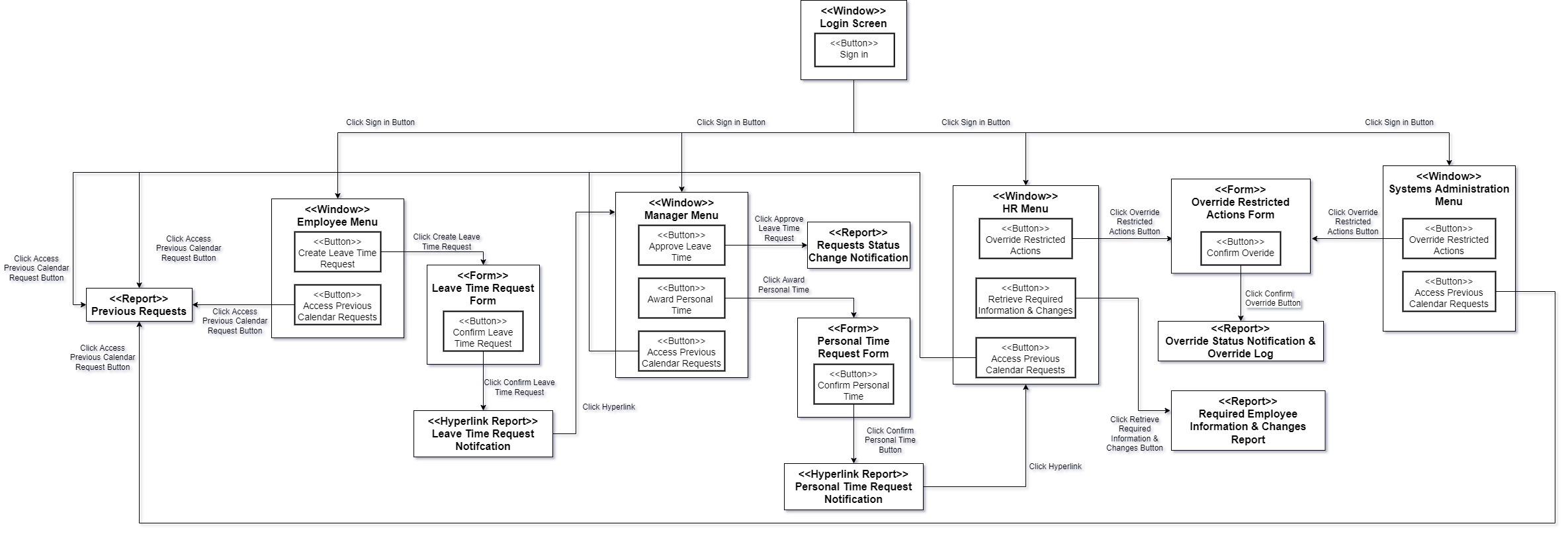
**Figure 28. Screen Sketch and Wireframe of Possible Error that can Occur When Request Violates Rule-Based System**

To illustrate the flow of control through the various screens a Windows Navigation Diagram (WND) is made. The diagram helps show the relationship between all screens, forms, and reports that are used by the Vacation Tracking System (VTS). The first window that any user must access is the same, which is the *Login Screen*. The *Login Screen can* then transition into the different user windows once the user successfully logs in by clicking on the *Sign in* button. Once the user logs in, depending on the user they go to their specific menu. If an employee logs in then they are taken to the *Employee Menu* window, which consists of 2 buttons. One button is the *Create the Leave Time* button which when pressed transitions to the *Leave Time Request Form*. Once the user enters the correct information, they can submit the form by pressing the *Confirm Leave Time Request* button. Clicking the button will then lead to a report on the *Leave Time Request Notification*. The other button is the *Access Previous Calendar Requests* which brings up a report of previous leave time requests made in the last calendar year.

If a manager logs in then they are taken to the *Manager Menu* window, which consists of 3 buttons. One button is the *Approve Leave Time* button which is used to approve or deny a leave time request made by an employee under them. Once the user clicks on the *Approve Leave Time* button then they receive a report of the *Request Status Change Notifications*. Another button is the *Award Personal Time* button which is used to directly award an employee’s personal leave time. Once the user clicks on the *Award Personal Time* button, they are taken to the *Personal Time Request Form*. Once the user enters the necessary information in the form they can submit the form by clicking on the *Confirm Personal Time*. Once clicking on this button, then they receive a report of the *Personal Time Request Notification*. The final button the user can click on in the *Manager Menu* is the *Access Previous Calendar Requests* button which has the same functionality across all screens.

If an HR representative logs in then they are taken to the *HR Menu* window, which consists of 3 buttons. One button is the *Override Restricted Actions* button which begins the process of overriding an action restricted by the rules. By clicking on the *Override Restricted Actions* button, the user is taken to the *Override Restricted Actions Form*. Once the user enters the correct information, they can submit the form by clicking on the *Confirm Override* button. Upon clicking the *Confirm Override* button and submitting the form, the user then receives a report for the *Override Status Notification & Override Log* as the system logs in every override. Another button is the *Retrieve Required Information & Changes* button which is used to allow HR to retrieve required employee information and changes. Clicking this button will take the user to the *Required Employee Information & Changes Report*. The final button the user can click on in the *HR Menu* is the *Access Previous Calendar Requests* button which has the same functionality across all screens.

If a systems administrator logs in then they are taken to the System Administration Menu window, which consists of 2 buttons. One button is the *Override Restricted Actions* button and the other button is the *Access Previous Calendar Requests* button. These buttons have the same functionality across all screens.

****

**Figure 29 Windows Navigation Diagram for Overall System**

# **Appendix A: Glossary**

* WND - Windows Navigation Diagram
* VTS - Vacation Tracking Software