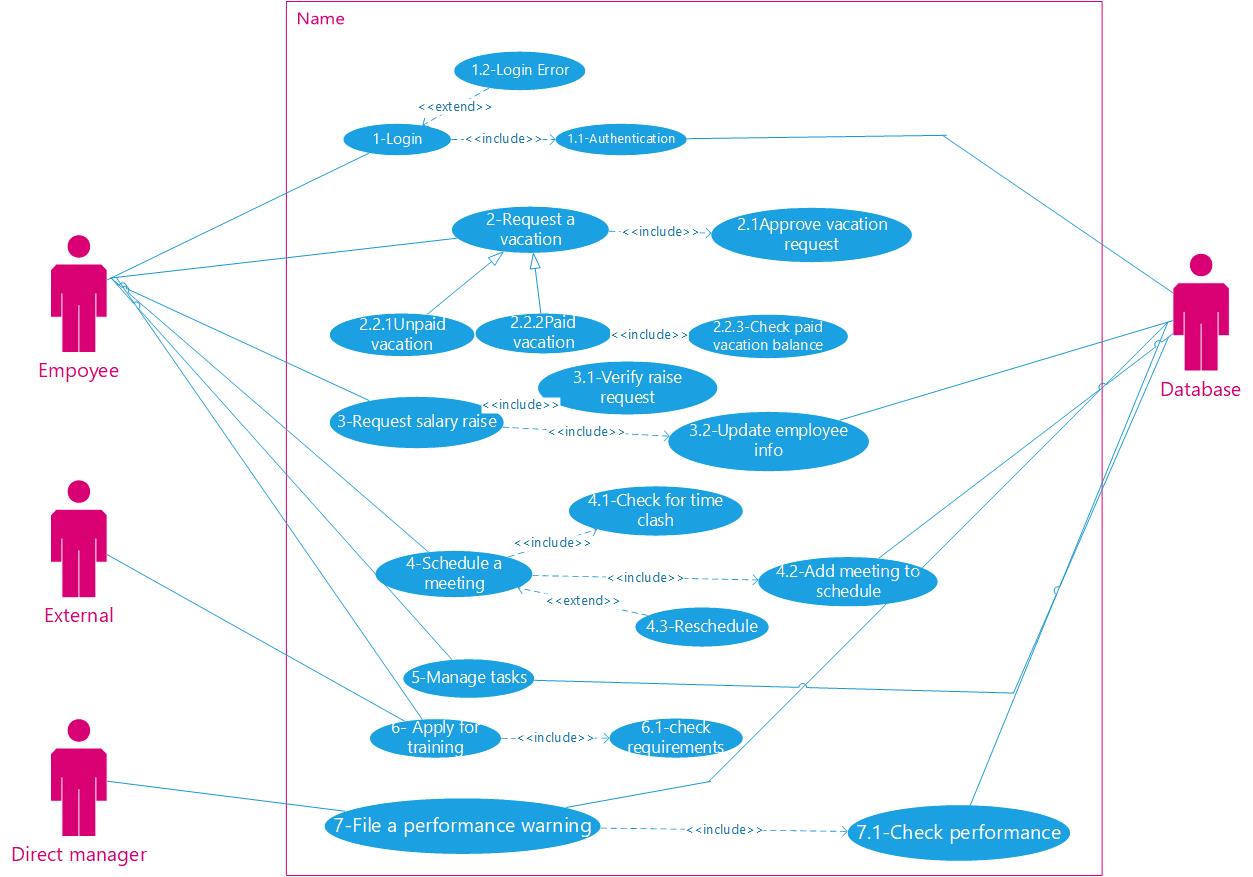
|  |  |
| --- | --- |
| **AIN SHAMS UNIVERSITY**  **FACULTY OF ENGINEERING**  **International Credit Hours Engineering Programs (i.CHEP)** |  |
|  | |
| **Design Document**  **Software Engineering**  **Participants:**  **Omar Ashraf Mabrouk 19P8102**  **Mahmoud Mohamed Omar 19P5803**  **Hussein Ahmed Hassan Selim 19P9614**  **Abdelrahman Mohamed Salah 19P9131**  **Zakaria Sobhy Abd El-Salam 19P2676**  **Mahmoud Mohamed Seddik 19P3374** | |
|  | |

|  |  |  |
| --- | --- | --- |
| Course Code  CSE 334 | Course Name  Software Engineering | |
|  | **Semester**  Spring 2021 | **Date of Submission**  11-06-2021 |

|  |  |
| --- | --- |
|  |  |

**Employee Use-Case Diagram**



Use-Case Descriptive Narative

|  |  |
| --- | --- |
| Use-Case Name | 1-Login |
| Related Requirements |  |
| Goal in Context | Log the user into his account |
| Preconditions | Program is installed and the user got his username and password |
| Successful end condition | Logged into the users account |
| Failed end condition | Can’t login either due to wrong username, or password, or username/password mismatch |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Trigger | Pressing login button |
| Included use-case | Authentication |
| Main Flow  Extension | 1. The user enters the username and password.   Include: Authentication  2.1) the username is checked if it exists in the database  2.2) if the username is found it is checked with the password registered  2.3) then the input data by the user are checked to be matching with that in the database  3) The user is then taken to the main screen after logging in  4) The user can’t log in due to mismatching username and password |

|  |  |
| --- | --- |
| Use-Case Name | 1.2- Login Error |
| Related Requirements | 1-login |
| Goal in Context | Define the error in the logging in process to the user |
| Preconditions | Login is tried |
| Successful end condition | The user logs into the system |
| Failed end condition | The user can’t log in |
| Primary Actor | Employee |
| Secondary Actor | ------------------- |
| Trigger | Failed to log in |
| Main Flow | 1. The username and password aren’t matching. 2. The user is asked to re-enter the username and password. 3. If the process fails, more than twice the user is asked to go back to the program’s admin |

|  |  |
| --- | --- |
| Use-Case Name | 2-Request a Vacation |
| Related Requirements | Paid/Unpaid Vacation |
| Goal in Context | Apply for a vacation so that it could be verified |
| Preconditions | Be logged into your account |
| Successful end condition | Vacation request is sent successfully to whom it may concern |
| Failed end condition | Failed to send vacation request |
| Primary Actor | Employee |
| Secondary Actor | Database, HR |
| Included use-case | Approve vacation request |
| Main Flow | 1. The user inputs the start/end date of the vacation. 2. The start and end dates are checked to be upcoming, and the end date is after the start date. 3. The request is then sent to the database to notify the HR employee responsible for approving the vacation request. 4. The respond of the HR employee is sent to the database and a notification is sent to the requestor. |

|  |  |
| --- | --- |
| Use-Case Name | 2.1- Approve Vacation Request |
| Related Requirements | 2-Request a Vacation |
| Goal in Context | Give a respond to a vacation request |
| Preconditions | A vacation request is filed by an employee |
| Successful end condition | A respond is delivered to the requestor |
| Failed end condition | Not respond is sent to the request |
| Primary Actor | HR Employee |
| Secondary Actor | Database, Employee |
| Trigger | A vacation is requested, and a notification is sent to the concerned HR employee |
| Main Flow | 1. The notification of an Employee that requested a vacation is sent from the database to the concerned HR employee. 2. The employee responds to the vacation request either by approval or denial. 3. The respond is then registered in the database. 4. A notification with the respond is then sent to the vacation requestor |

|  |  |
| --- | --- |
| Use-Case Name | 2.2.2- Paid vacation |
| Related Requirements | 2- Request a vacation |
| Goal in Context | Request a vacation that is fully paid |
| Preconditions | A vacation is requested |
| Successful end condition | A paid vacation request is sent |
| Failed end condition | Failed to send the request |
| Primary Actor | Employee |
| Secondary Actor | Database, HR employee |
| Base use-case (inherited from) | 2- Request a vacation |
| Included use-case | 2.1) approve vacation request |
| Main Flow | 1. The user requests a vacation as in the base use case by entering the start and end date. 2. The user is asked to enter the type of vacation requested which is paid. 3. The start and end date are sent to the database and the duration is calculated. 4. The requested duration is then compared with the paid vacation balance if there is enough balance a request is sent to the HR employee.   Include: approve vacation request   1. After the vacation is reviewed and a respond is registered in the database the respond is sent to the user |

|  |  |
| --- | --- |
| Use-Case Name | 3- Request Salary Raise |
| Related Requirements | Update user info |
| Goal in Context | Send a request to the concerned person about a salary raise |
| Preconditions | Be logged into the system |
| Successful end condition | Salary raise request is sent successful |
| Failed end condition | Failed to send salary raise request |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Included use-case | 3.1-Verify raise request  3.2-update employee info |
| Main Flow | 1. The user enters the requested salary to be raised. 2. The requested salary is saved to the database. 3. The salary request is sent from the database to the concerned HR employee.   Include: Verify raise request   1. The HR employee either approve or deny the request. 2. The respond of the HR employee is then saved to the database.   Include: update employee info   1. If the request is approved, then the salary of the employee is updated to the new value in the database |

|  |  |
| --- | --- |
| Use-Case Name | 4- schedule a meeting |
| Related Requirements | ---------------------------------------- |
| Goal in Context | Add a meeting to the Schedule |
| Preconditions | Be logged into your account and have the authorization to schedule a meeting |
| Successful end condition | Meeting scheduled successfully |
| Failed end condition | Can’t schedule a meeting |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Included use-case | 4.1- check for time clash  4.2- add meeting to schedule |
| Main Flow | 1. The employee schedule a meeting by entering the date of the meeting.   Include: check for time clash   1. Save the meeting time to the database.   Include: add meeting to schedule   1. Send a notification to all concerned employees about the meeting |

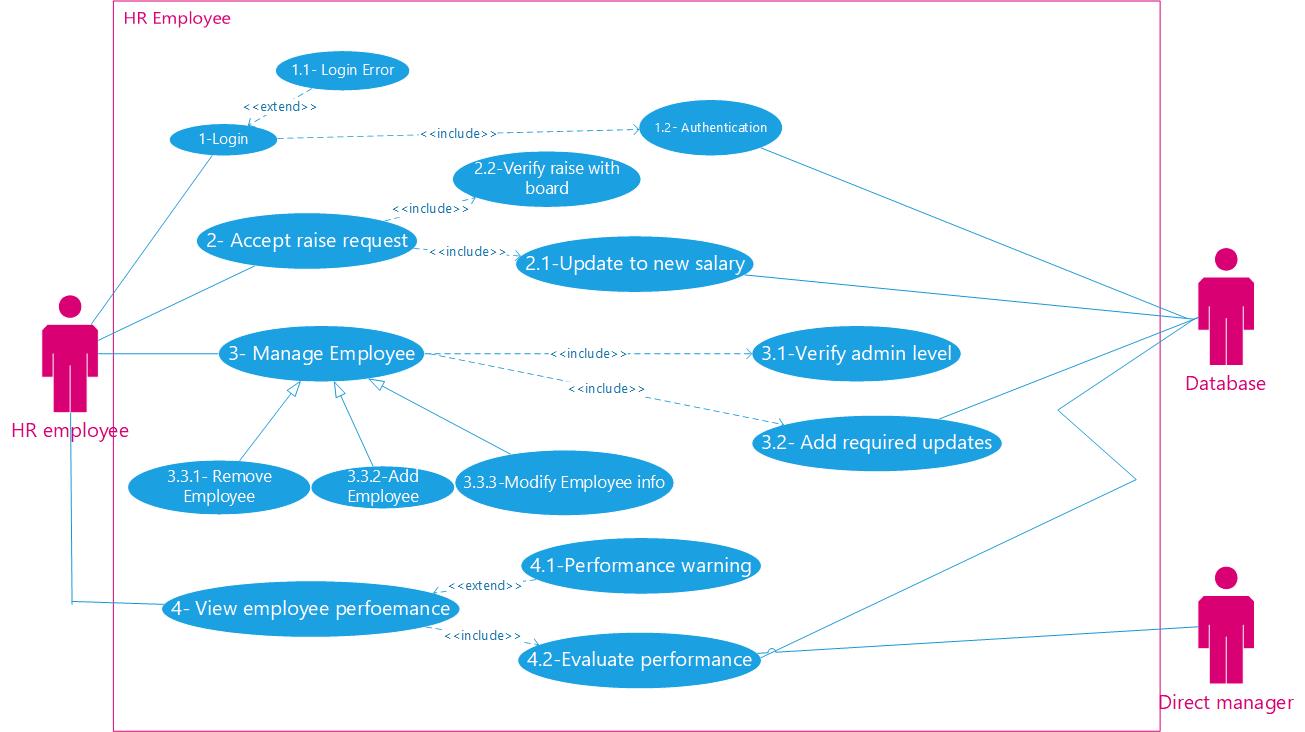
|  |  |
| --- | --- |
| Use-Case Name | 4.1- Check for time clash |
| Related Requirements | 4- Scheduling a meeting |
| Goal in Context | Check if there is a clash between a new meeting which is to be registered and an already scheduled meeting |
| Preconditions | A date for a meeting is entered by a user |
| Successful end condition | There is no clash you can schedule the meeting |
| Failed end condition | There is a clash with a list of meetings maybe you would consider rescheduling the meeting |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Trigger | Schedule a meeting |
| Main Flow | 1. Check if there is an already scheduled meeting having the same time as that which is to be rescheduled. 2. If there is a clash suggest other time to schedule the new meeting, if there is not sent a message that there is no clash. 3. Ask the user to choose a new date for the meeting and repeat the steps 1 and 2 again till you reach a valid date for the meeting |

|  |  |
| --- | --- |
| Use-Case Name | 6- Apply for training |
| Related Requirements | ------------------------------ |
| Goal in Context | The employee wants to file a request for training |
| Preconditions | Meet the training requirements |
| Successful end condition | Request sent |
| Failed end condition | Failed to send request |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Included use-case | 6.1- Check Requirements |
| Main Flow | 1. The user inputs the required data for applying for a certain training. 2. The data input by the user are checked with the requirements.   Include: Check Requirements   1. If the employee meets the required qualities, then the system should accept his request and save it to the database. 2. If the user does not meet the requirements send a notification with the fields that doesn’t meet the requirements 3. In both cases send a notification of the status for the request. |

|  |  |
| --- | --- |
| Use-Case Name | 6.1- Check Requirements |
| Related Requirements | 6- Apply for training |
| Goal in Context | Check the requirements for applying into a certain training program to see if the applicant meets the requirements or not |
| Preconditions | An employee has applied for a training and provided the required entry data. |
| Successful end condition | Applicants’ data checked against required criteria |
| Failed end condition | Failed to check applicant’s data |
| Primary Actor | Employee |
| Secondary Actor | Database |
| Trigger | An employee has applied for training |
| Main Flow | 1. Fetch the applicants’ qualifications from the database. 2. Fetch the minimum qualifications required to enter the training program from the database. 3. Compare between the applicants’ qualifications and minimum entry qualifications required. 4. If the applicants’ qualifications are as or more than required accept his application 5. If the applicants’ qualifications are less than required in some fields send a notification to the user with the fields need to be improved 6. Save the systems respond in the database. 7. Send the respond to the applicant |

|  |  |
| --- | --- |
| Use-Case Name | 7- File a performance warning |
| Related Requirements | ----------------------------------------- |
| Goal in Context | Send a warning to an employee due to his bad performance |
| Preconditions | Performance is calculated and a certain standard is provided for the system |
| Successful end condition | Warning is sent successfully to the employee |
| Failed end condition | Failed to send the warning |
| Primary Actor | Direct Manager |
| Secondary Actor | Database, Employee |
| Included use-case | 7.1- Check Performance |
| Main Flow | 1. Employee’s performance is feed to the system.   Include: Check Performance   1. If the performance is less than the standard the direct manager is notified 2. A warning should be sent to the employee. 3. A history of the warnings sent to the employee is saved in the database as a reference. |

|  |  |
| --- | --- |
| Use-Case Name | 7.1- Check Performance |
| Related Requirements | 7- File a performance warning |
| Goal in Context | Assess the performance of an employee |
| Preconditions | The employee’s performance is provided to the system and a given standard is saved in the database |
| Successful end condition | Performance is checked successfully, and the result of the check is saved into the database |
| Failed end condition | Failed to conduct performance check |
| Primary Actor | Direct Manager |
| Secondary Actor | Database |
| Trigger | The system is asked to file a warning |
| Main Flow | 1. The employee’s performance is fetched from the database. 2. The standard performance is either entered by a manager or from a stored value in the database. 3. The employee’s performance is compared to standard performance. 4. The result of the check is then stored into the database |

HR and Manager Use-Case Diagram

Use-Case Descriptive Narative

|  |  |
| --- | --- |
| Use-Case Name | 2- Accept Raise request |
| Related Requirements |  |
| Goal in Context | The HR employee can accept a salary raise request made by an employee |
| Preconditions | A raise request is made by an employee |
| Successful end condition | The HR employee has made a clear decision in whether to accept or refuse the salary raise request and upon which the salary is updated |
| Failed end condition |  |
| Primary Actor | HR employee |
| Secondary Actor | Database |
| Trigger | An employee has requested a salary raise |
| Included use-case | 2.1-Update to new salary/2.2-verify raise with board |
| Main Flow | 1. The system prompts the HR Employee that a certain employee has requested for a salary raise. 2. The HR will review the employee’s request. 3. The HR will have meetings with the board to verify the salary raise. 4. Once the change in salary is agreed upon ,the system will send the new salary to Database   Include::Update to new salary |

|  |  |
| --- | --- |
| Use-Case Name | 2.1- Update to new salary |
| Related Requirements | ---------------------------------------------------------------------------- |
| Goal in Context | Change the value of the salary for an employee in Database |
| Preconditions | The salary raise is accepted by HR |
| Successful end condition | The value of salary is changed successfully |
| Failed end condition | There is an error in connection with the database |
| Primary Actor | Database |
| Secondary Actor | ---------------------------------------------------------------------------- |
| Trigger | The employee salary is changed |
| Main Flow | 1) Search the database for the employee by his username.  Update the Employee salary to match the updated salary |

|  |  |
| --- | --- |
| Use-Case Name | 3- Manage Employee |
| Related Requirements | 2.2.2 Employee information management |
| Goal in Context | The HR employee can add/remove or modify employee info |
| Preconditions | The HR employee should have an admin level privilege |
| Successful end condition | Data is updated/created or removed successfully, and those modifications is accepted by the database. |
| Failed end condition | Data is not updated/created or removed properly, or the updating employee has no admin level privilege. |
| Primary Actor | HR employee |
| Secondary Actor | Database |
| Trigger | HR has opened the employee editing screen |
| Included use-case | Verify Admin level, add required updates |
| Main Flow | 1) The HR opens the employee editing screen.  Include::VerifyAdminlevel.  Include::addrequiredupdates. |

|  |  |
| --- | --- |
| Use-Case Name | 3.3.1- Remove Employee |
| Related Requirements | 2.2.2 Employee information management |
| Goal in Context | The HR employee can remove an employee after he is efforts is no longer needed |
| Preconditions | The HR employee should have an admin level privilege |
| Successful end condition | Employee Data is removed successfully from the system |
| Failed end condition | ------------------------------------------------------------------------------------\* |
| Primary Actor | HR Employee |
| Secondary Actor | Database |
| Trigger | HR employee chooses to remove an employee from the system |
| Base use-case (inherited from) | 3- Manage Employee |
| Main Flow | The HR selects the username of the employee he wishes to remove from the database. |

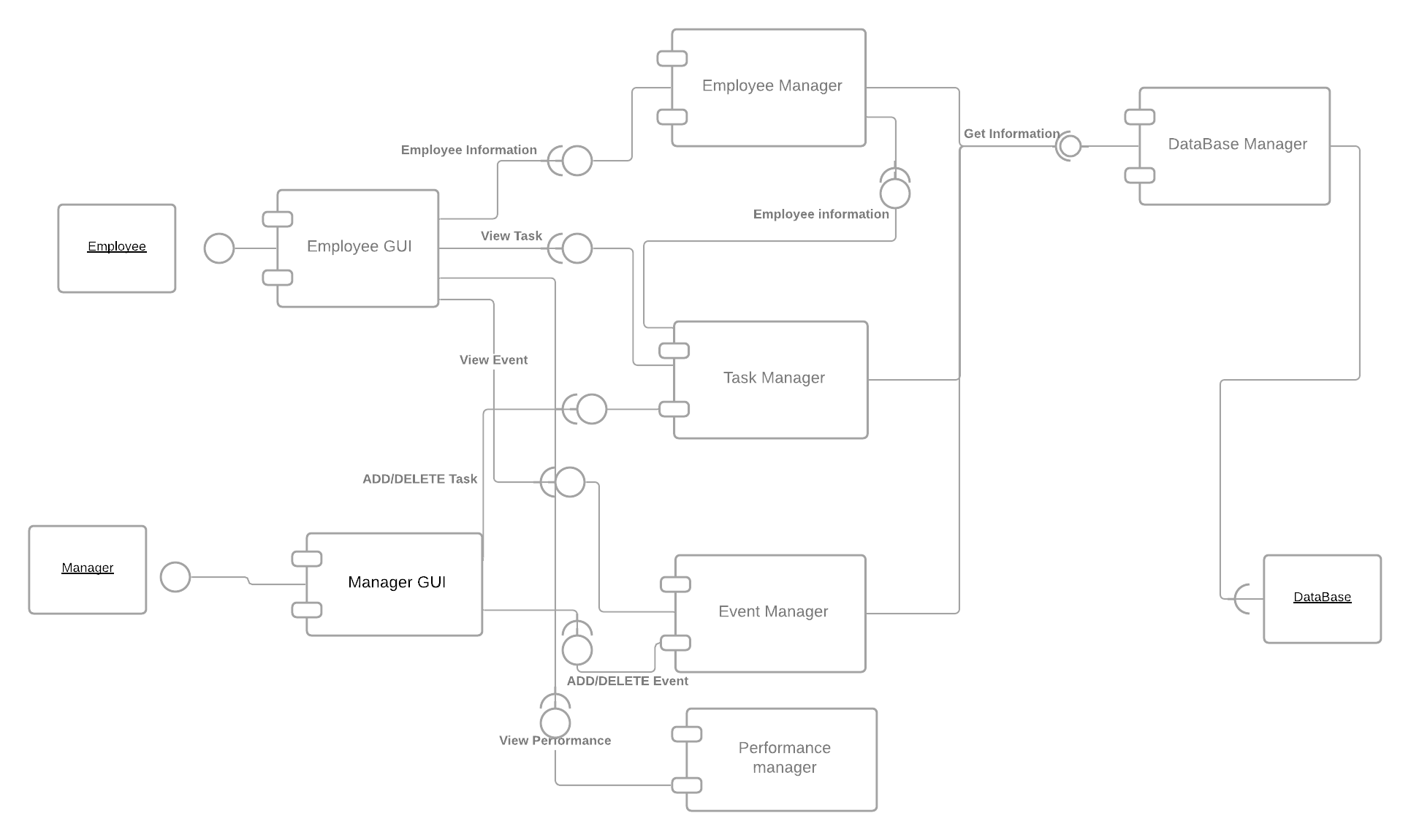
|  |  |
| --- | --- |
| Use-Case Name | 3.3.2- Add Employee |
| Related Requirements | 3- Manage Employee |
| Goal in Context | Adding a new employee to the system |
| Preconditions | The list of employees in the system is accessed |
| Successful end condition | The information of the employee is added to the list of employees and is stored in the database successfully |
| Failed end condition | Failure in adding the employee to the system |
| Primary Actor | HR-employee |
| Secondary Actor | Database |
| Trigger | The system is asked to add a new employee |
| Main Flow | 1) the HR-employee opens the list of all employees.  2) he then chooses to add new employee.  3) the information of the employee is then entered.  4) the HR-employee then saves this information in the list and the database. |

|  |  |
| --- | --- |
| Use-Case Name | 4- View Employee performance |
| Related Requirements | ----------------------------------------- |
| Goal in Context | Assess the performance of an employee |
| Preconditions | The employee’s performance is provided to the system and a given standard is saved in the database |
| Successful end condition | Performance is checked successfully, and the result of the check is saved into the database |
| Failed end condition | Failed to conduct performance check |
| Primary Actor | HR Employee |
| Secondary Actor | Database |
| Trigger | The system is asked to view employee performance |
| Main Flow | 1. The employee’s performance is fetched from the database. 2. The standard performance is either entered by a manager or from a stored value in the database. 3. The employee’s performance is compared to standard performance.   4) The result of the check is then stored into the database |

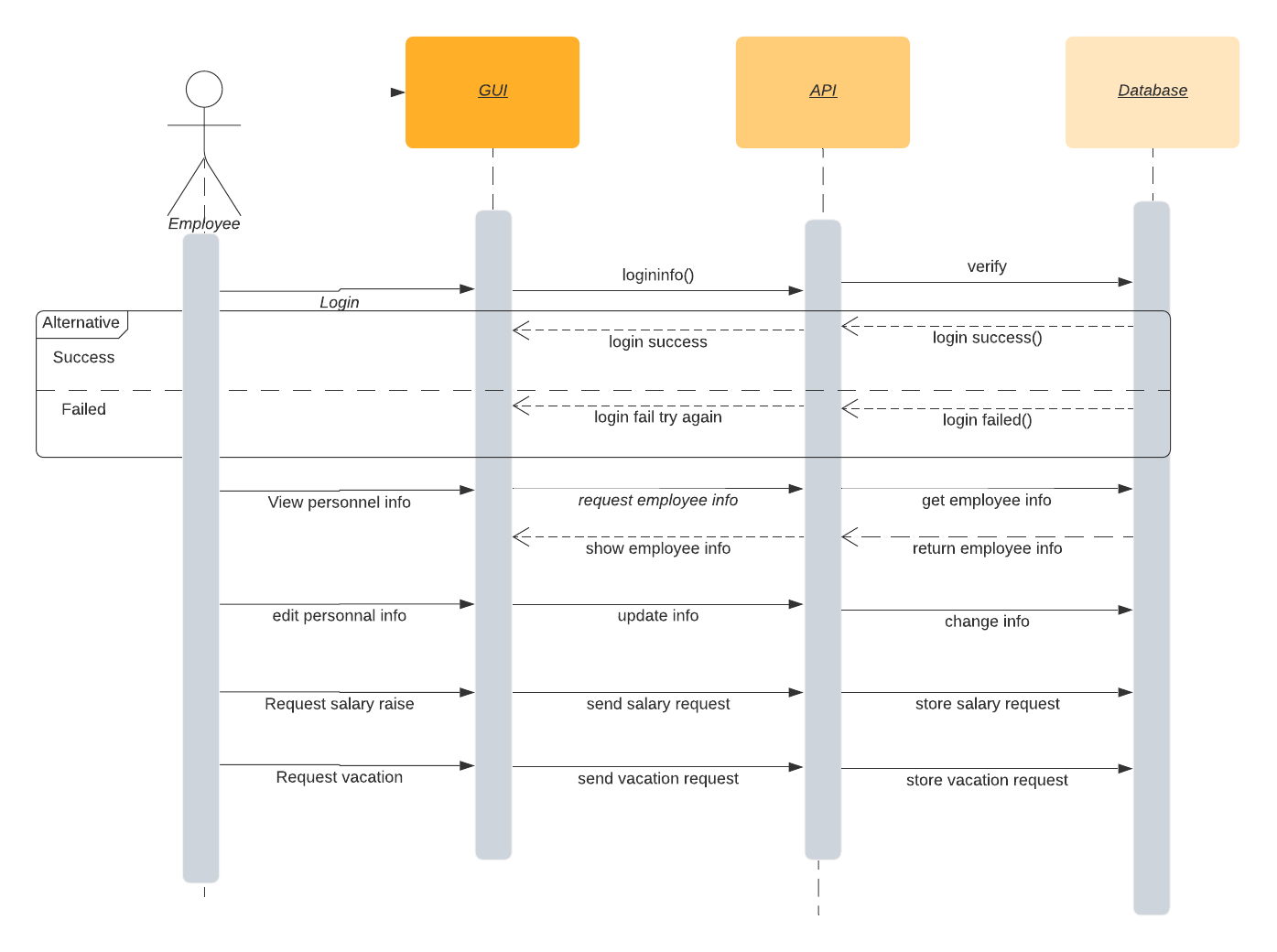
|  |  |
| --- | --- |
| Use-Case Name | 4.1- Performance warning |
| Related Requirements | 4- View Employee performance |
| Goal in Context | Issue a performance warning to an employee whose performance is lacking |
| Preconditions | 1) The employee’s performance is provided to the system and a given standard is saved in the database  2) The system finds the employee performance to be poor |
| Successful end condition | The warning is issued correctly, and the result of the issue is saved into the database and sent to the employee |
| Failed end condition | Failed to issue the warning |
| Primary Actor | HR employee |
| Secondary Actor | Database – employee |
| Trigger | The system is asked to file a warning |
| Main Flow | 1) the employee performance is provided to the system  2) the HR employee checks the performance of the employee and compares it to his peers  3) if his performance is found to be lacking the warning is issued |

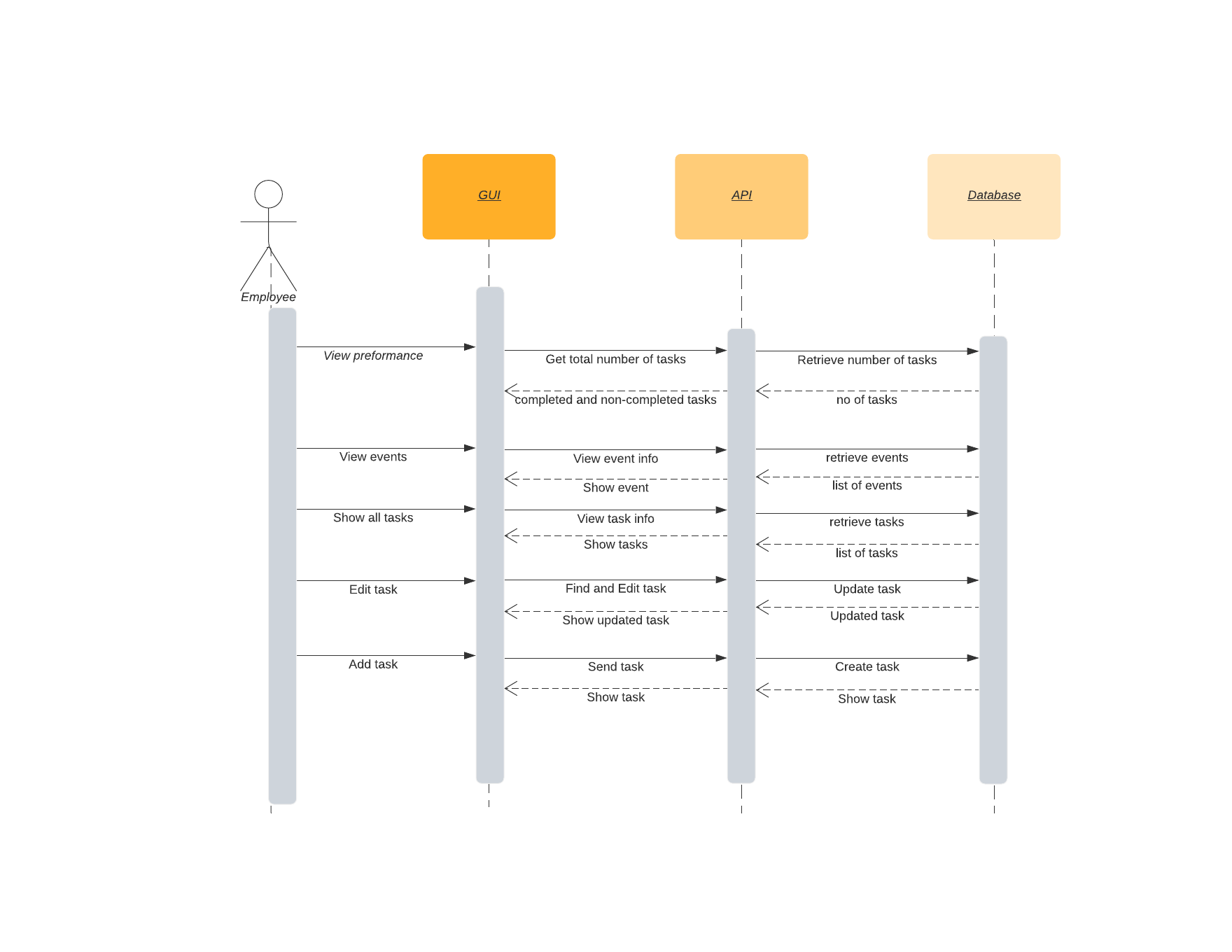
|  |  |
| --- | --- |
| Use-Case Name | 4.2- Evaluate Performance |
| Related Requirements | 4- View Employee performance |
| Goal in Context | Adding evaluation, the performance of the employee |
| Preconditions | The employee’s information is stored in the system. The manager puts an evaluation of the employee |
| Successful end condition | Manager manages to update the employee performance and it is saved on the database |
| Failed end condition | Performance failed to be updated |
| Primary Actor | Direct Manager |
| Secondary Actor | Database |
| Trigger | The system is asked to evaluate a certain employee performance |
| Included use-case | 4-view performance |
| Main Flow | 1) The manager views the performance of a certain employee  Include: view performance  2) the manager chooses to update the performance of the employee |

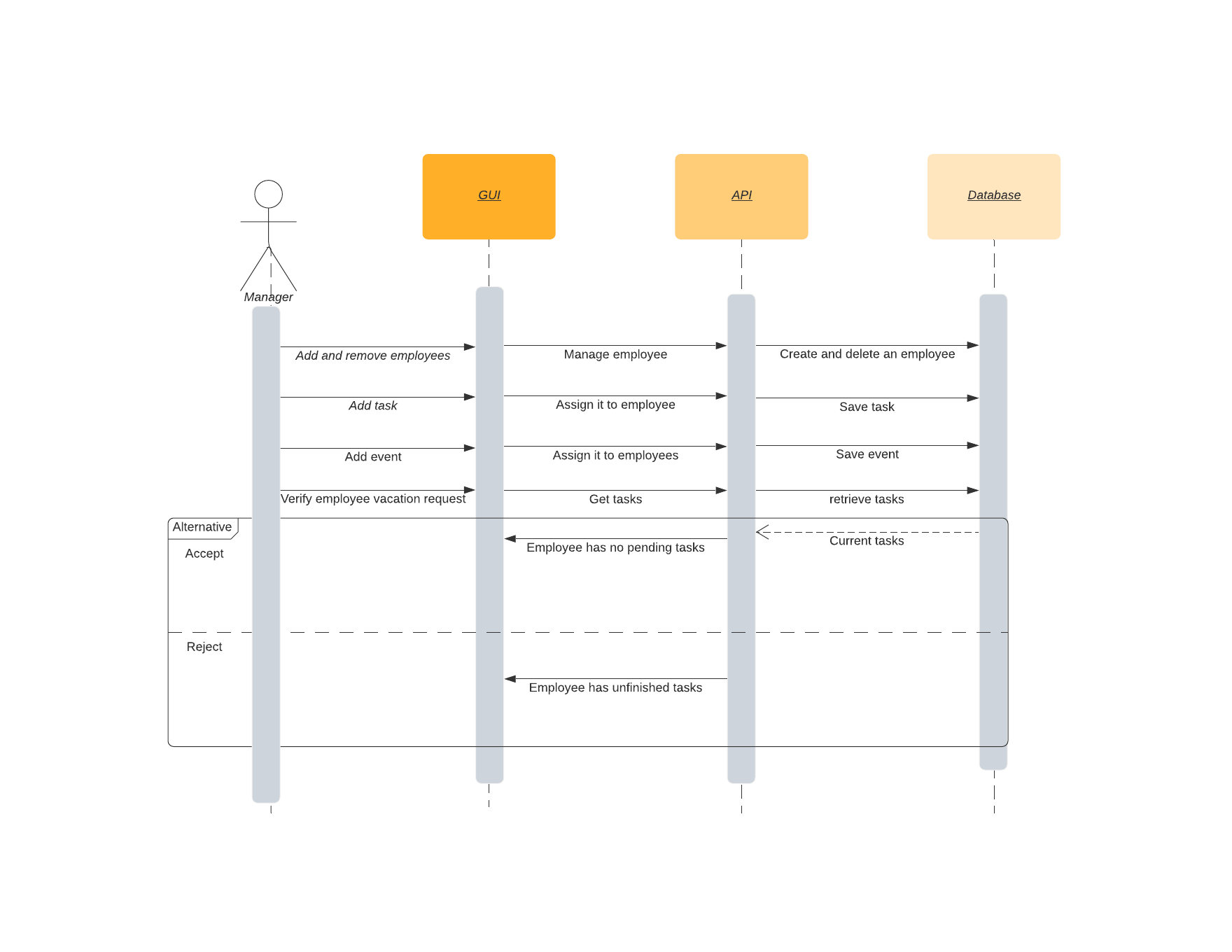
**Component Diagram**

****

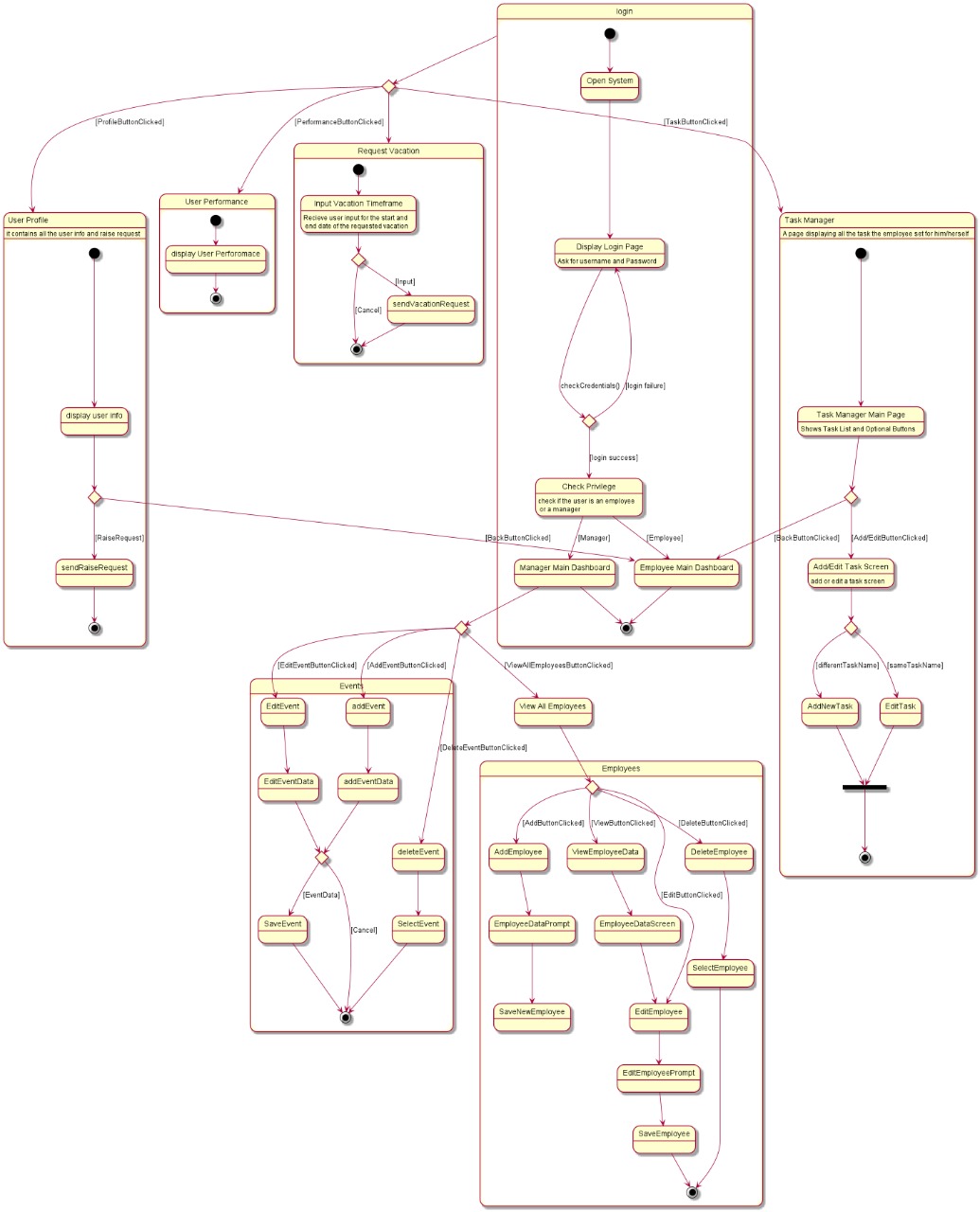
**Sequence Diagram**

****

****

****

**State Diagram**

****

# Login Frame:

By running the application the system opens {Open System} and displays login page

{Display Login Page}.

{Display Login Page} then has [login failure] which goes to the login page by {Display Login Page} or in case of success, the process of {Check Privilege}which is done to check if this was an employee or a manager; therefore, it is called to switch either to {Manager Main Dashboard} or {Employee Main Dashboard}.

# Events Frame:

From the Main Screen: In case the Manager clicks on {Edit Event} or {add Event} both process execute {Edit Event Data} and {add Event Data} respectively; which pops a screen prompting for new data, however, a choice is then given to either cancel or add/edit data.

In case the Manager clicks on {delete Event} after selecting {Select Event}, the process is executed deleting this selected event.

# Employees Frame:

From the Main Screen: In case of Manager clicking on {View All Employee} a screen is shown with:

{Add Employee}: Upon clicking shows input screen for user by process {Employee Data prompt} and saves new employee to database by {Save New Employee}.

{View Employee Data}: Upon clicking navigates to Employee data screen by {Employee Data Screen} which can be used to edit employee to prompt user for input {Edit Employee Prompt} and then saves employee by {Save Employee}.

{Delete Event}: Upon clicking Delete event {Delete Event} process is executed by selecting the employee wanted for deletion {Select Employee}

# Task Manager Frame:

From the Main Screen: Navigate to Task Manager by clicking go to Task Manager; this show the tasks and functions by the process {Task Manager Main Page}. Add, edit or delete will be used by adding new in prompt screen using {Add New Task} and editing existing data by a prompt screen using {Edit Task} this then ends the current state. An option exists to return back to the main dashboard {Employee Main Dashboard}; upon clicking back to main.

# Request Vacation Frame:

From the Main Screen: Navigate to request vacation screen by clicking on Request vacation button which will then prompt by {Input vacation TimeFrame}, and has 2 choices either to cancel and return back to main or {Send Vacation Request} for state to end.

# User Performance Frame:

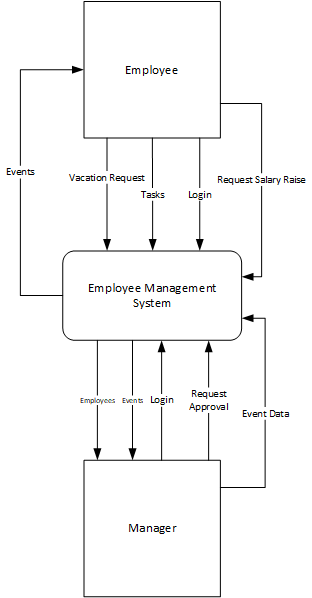
From the Main Screen: Navigate to performance of employee by clicking on the my performance button; which then displays user performance by {Display User Performance} and ends the state.

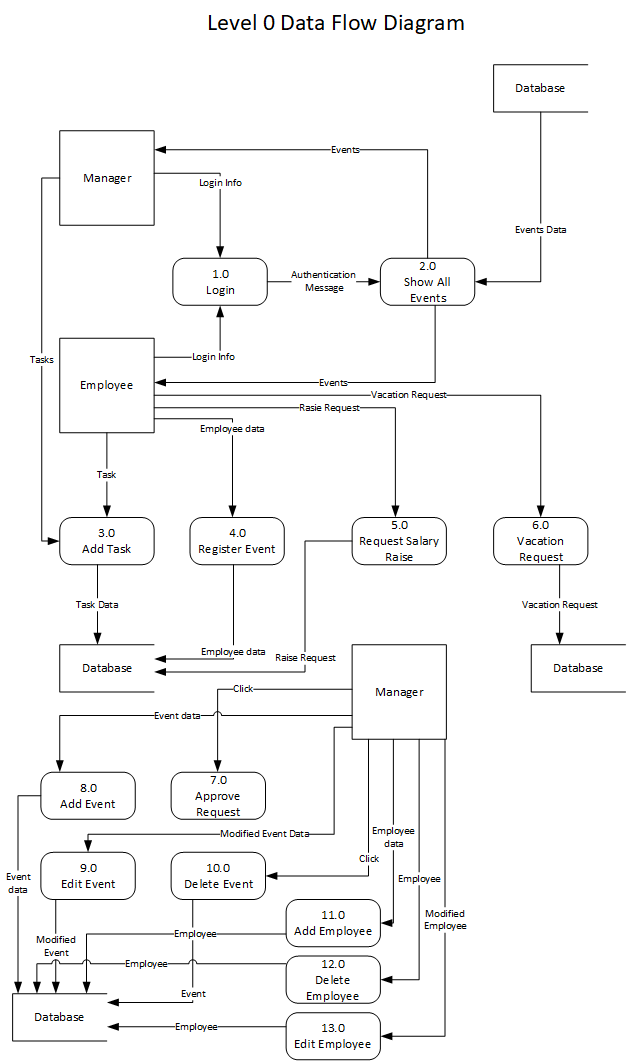
It can also navigate back to main screen by button back to main.

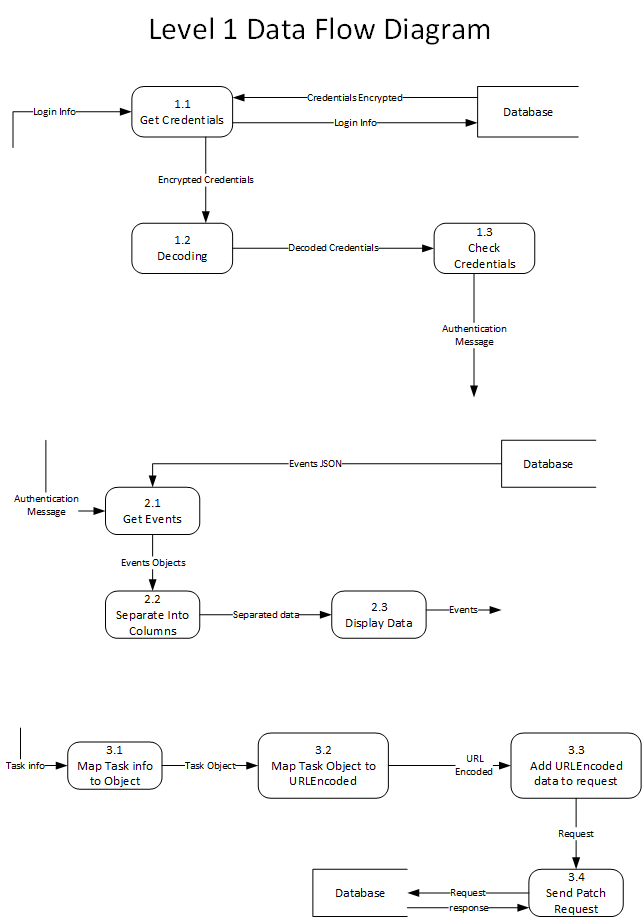
# User Profile Frame:

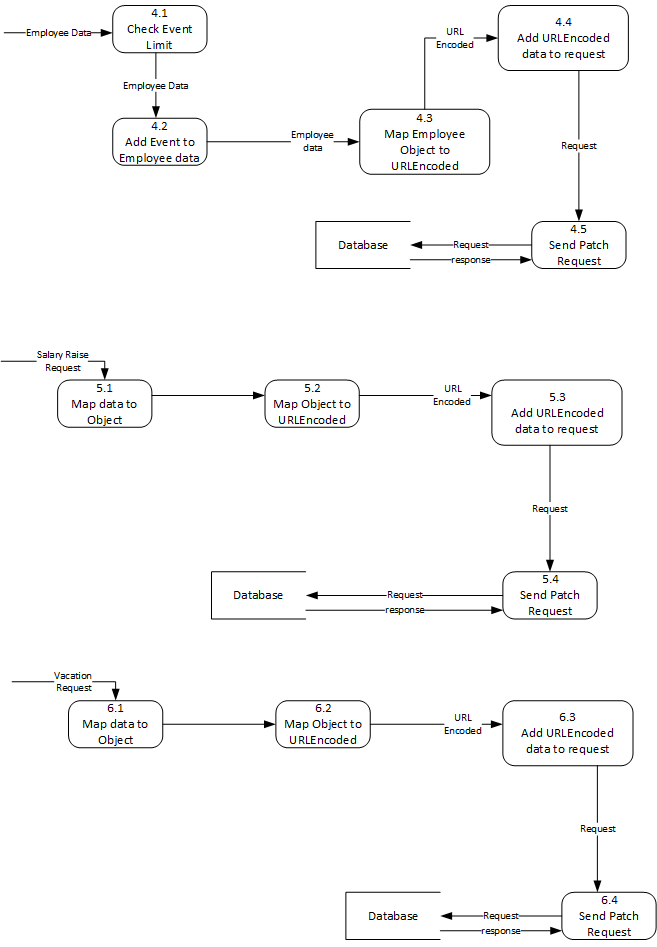
From the Main Screen: Navigate to User Profile of the employee by clicking on my profile which displays information using {Display Info}, then gives you 2 choices either to go back to main by clicking on it or send salary raise request by {Send Raise Request} and ends the state.

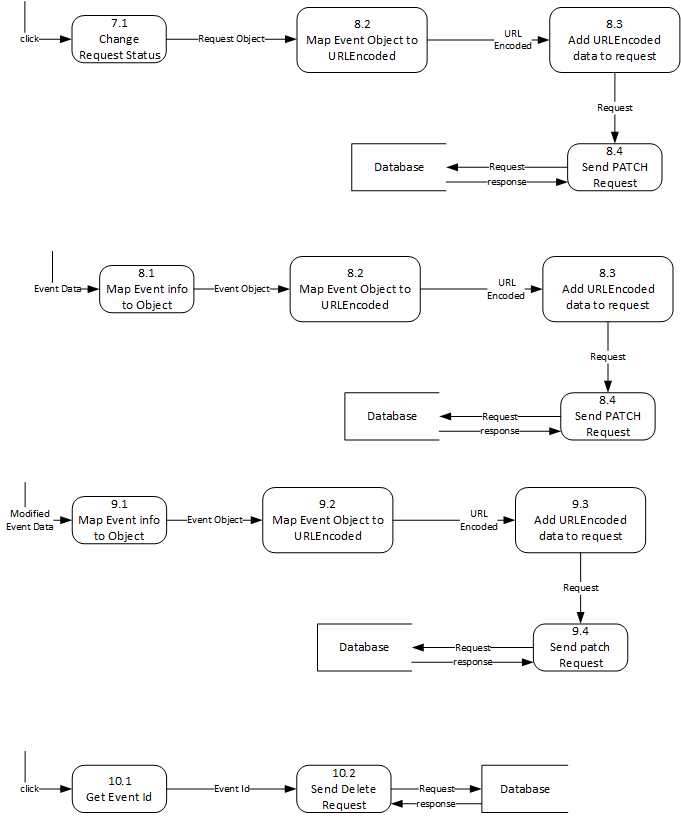
**DFD Diagram**

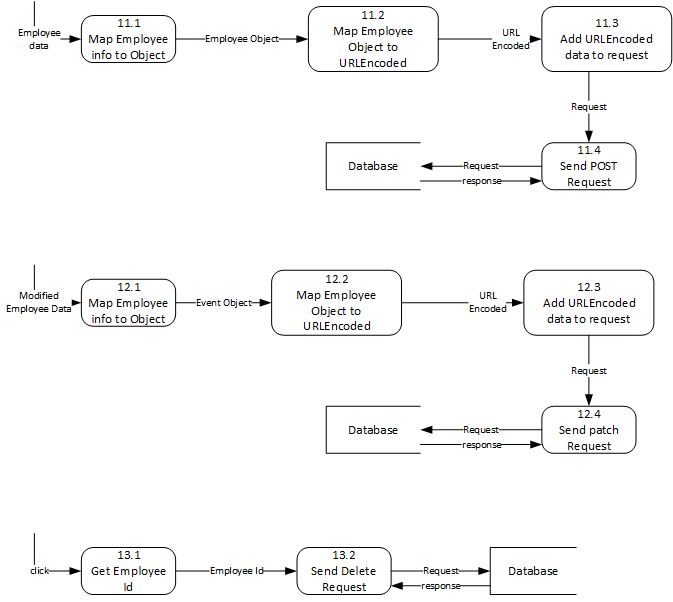
****

****

****

****

****

****

# E:\download\MicrosoftTeams-image (4).png E:\download\MicrosoftTeams-image (5).png

# Context Description:

**Employee Management System** is communicating between 2 external entities Employee and Manager.

**Employee:** sends login information to enter the main screen, also he has the ability to send tasks information, and vacation /salary raise requests and view events details.

**Manager:** sends login information to enter the main screen, also he has the ability to send and view tasks /events information, vacation requests, and approve employee’s requests.

# Level-0 DFD:

Contains an additional internal database communicating with the other external entities and the EMS.

1. **Login:** Both Employee and Manager send login info which will be authenticated and sent as a message as an output
2. **Show All Events**: This is the Main screen for both the employee and Manager, in which all events will be shown by receiving the authenticated message from 1.0, then getting events data from the database to be sent/shown to the Employee and the Manager
3. **Add Task:** This process takes tasks info from the Employee or the Manager to be added by sending information to the database
4. **Register Event:** Employee will register the event shown on screen to be saved and sent to the database
5. **Request Salary Raise:** Takes salary request information from the Employee and sends it to the database to be saved
6. **Vacation Request**: Takes vacation request information from the Employee and sends it to the database to be saved
7. **Approve Request:** Upon clicking the Employee’s request is approved and modified in the database
8. **Add Event**: The Manager sends the event information to be added, then this data is sent to the database to be saved
9. **Edit Event:** The Manager sends the modified event information, then this data is sent to the database to be saved
10. **Delete Event**: Upon clicking on an event this event object is deleted from the database
11. **Add Employee:** Takes new Employee Data and sends it as an object to the database to saved
12. **Delete Employee:** Upon clicking this Employee will be deleted from the database
13. **Edit Employee:** The Manager sends the modified employee information, then this data is sent to the database to be saved

# Level-1 DFD:

1. **Login:** 
   1. **Get Credentials:** Takes login information from the Manager or the Employee then sends it to the database to be encrypted and returned; encrypted data is then sent to as an output
   2. **Decoding:** Takes encrypted data and decodes it to be sent as an output
   3. **Check Credentials:** This then takes the decoded credentials and checks them send an authentication message as output to be used to show the main screen
2. **Show All Events:**
   1. **Get Events:** Extracts Events as a JSON object from the database, and takes the authentication message as a flag to show the main screen and its events
   2. **Separate into columns:** This takes the Events object and separates it into logical columns to be sent as an output for display
   3. **Display Data:** takes separated data and shows them on screen as Events data
3. **Add Task:**
   1. **Map Task info to Object:** Takes new Tasks info inputted from the Employee or the Manager and converts it to an object to be passed
   2. **Map Task Object to URL-Encoded:** takes the Task object and converts it to URL-Encoded to be passed
   3. **Add URL-Encoded Data to Reques**t: The URL-Encoded is then passed to be added and a request is generated to be sent as an output
   4. **Send Patch Request:** Receives the request from the URL-Encoded and sends the request to the database to receive a response of confirmation
4. **Register Event:** 
   1. **Check Event Limit:** Takes Employee Data who took the action of clicking and sends it as an output after checking the limit of Employees for this event
   2. **Add Event to Employee Data:** Takes this employee data and adds the event to its data to be sent forward
   3. **Map Employee Object to URL-Encoded:** Takes the final employee data to convert it to URL-Encoded to be passed
   4. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   5. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
5. **Request Salary Raise:** 
   1. **Map Data to Object:** Takes the salary raise request inputted details and send them forward after converting them to an object
   2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
   3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
6. **Vacation Request:**
   1. **Map Data to Object:** Takes the vacation request inputted details and send them forward after converting them to an object
   2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
   3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
7. **Approve Request:** 
   1. **Change Request Status:** takes the action of clicking and sends a request object to be passed
   2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
   3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
8. **Add Event:** 
   1. **Map Event Info to Object:** Takes the Event inputted details and send them forward after converting them to an object
   2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
   3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
9. **Edit Event:** 
   1. **Map Event Info to Object:** Takes the Modified Event inputted details and send them forward after converting them to an object
   2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
   3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
   4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
10. **Delete Event**
    1. **Get Event ID:** Upon clicking get ID method is called to extract the events ID to be passed
    2. **Send Delete Request:** Takes the ID of the Event and sends a request containing the ID to the database and receives a confirmation of deletion
11. **Add Employee:** 
    1. **Map Event Info to Object:** Takes the new Employee inputted details and send them forward after converting them to an object
    2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
    3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
    4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
12. **Edit Employee:**
    1. **Map Event Info to Object:** Takes the modified Employee inputted details and send them forward after converting them to an object
    2. **Map Object to URL-Encoded:** Takes the object and converts it to URL-Encoded to be passed forward
    3. **Add URL-Encoded Data to Request:** Takes the URL-Encoded and adds it to a request (generated) to be sent
    4. **Send Patch Request:** This is responsible to deliver the request to the database and receives a response of confirmation
13. **Delete Employee**
    1. **Get Employee ID:** Upon clicking get ID method is called to extract the events ID to be passed
    2. **Send Delete Request:** Takes the ID of the Event and sends a request containing the ID to the database and receives a confirmation of deletion

# Level-2 DFD:

1. Login:
   * 1. **Decrypt Data:** Takes the Encrypted Credentials and the Salt to decrypt the credentials received and output it as decrypted Data
     2. **Get Salt:** Gets salt from the database to be passed to Decrypt Data (1.2.1)
2. **Show All Events:**

2.2.1 **Request Separation:** Takes the Events Object and extracts each column separately to be passed

2.2.2 **Extracting Name:** Gets the required Name of the Event

2.2.3 **Extracting Description:** Gets the required Description of the Event

2.2.4 **Extracting Date:** Gets the required Name of the Event

2.2.5 **Format Data:** takes all required data (Name, Description and Date) of the required Event to be formatted for final separation and send as separated data of event

2.1.1 **Check Authentication Message:** Takes the Authentication Message and sends checks if it is valid returning a Boolean answer

2.1.2 **Events JSON to JSON Array Object:** takes the Boolean answer from 2.1.1 and takes the JSON Events object to convert it to an array object of JSON type to be passed

2.1.3 **Map JSON Array to Event Objects:** Takes the JSON Array Object and converts it to Event object to be passed forward and used to be shown

**System Architecture**

View

NETWORK

Controller

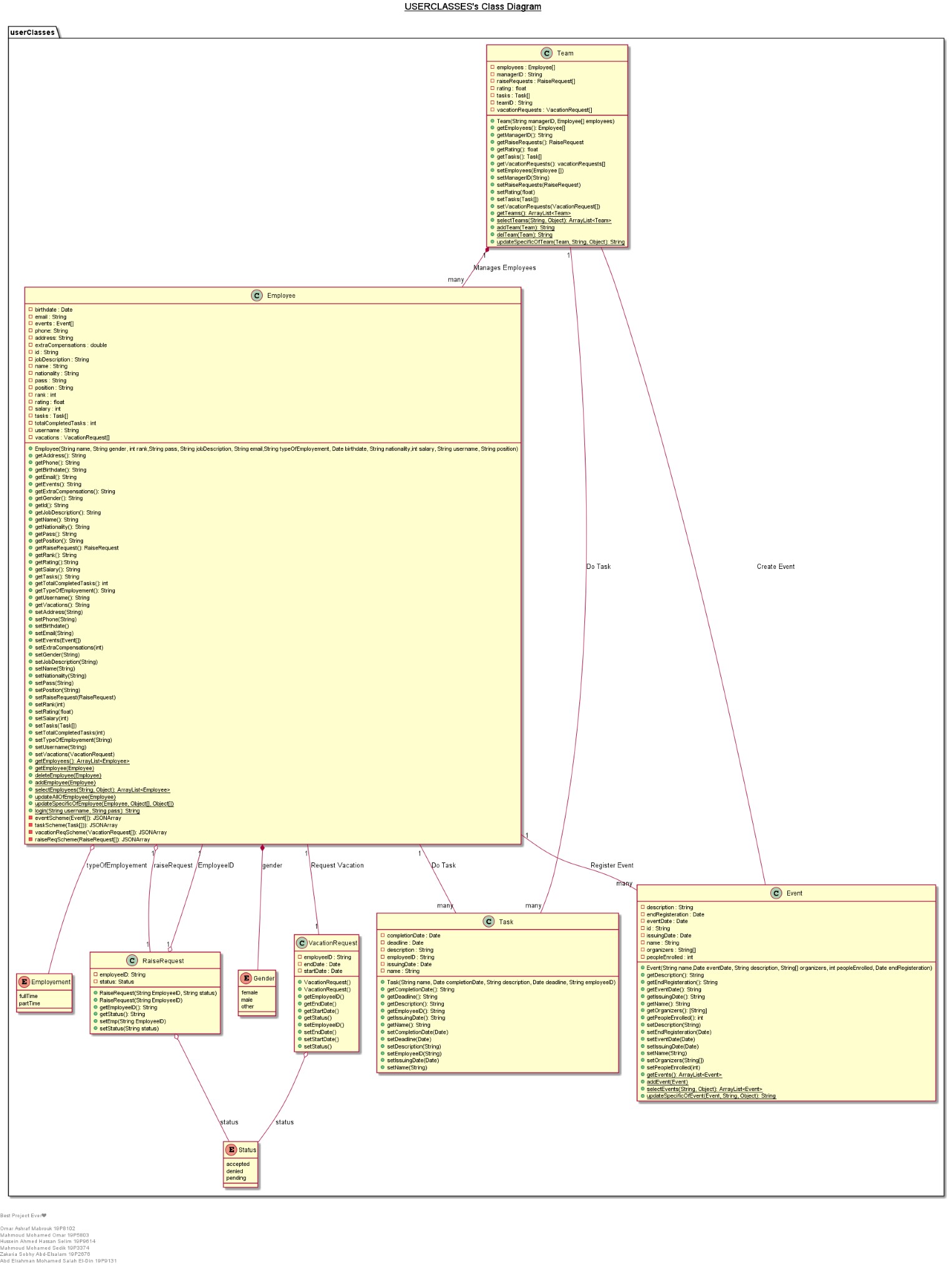
NETWORK

Model

**View:**

It is the part of the program the user can see and interact with. It includes the graphical user interface, which includes the windows presented to users when the software starts.

**Controller:**



These are all the classes used to create objects, this object oriented architecture is used to represent the controller which is the part responsible of manipulating the data in a program. The controller is represented in terms of classes since these classes are responsible of using and manipulating the program’s data, and controls the interaction between the model and the view if the program.

**Model:**

The model is represented in a data centered architecture since the clients all share the same database. The model is the data used by the program, all the program’s data is stored in the database and can be accessed by the company’s staff.

Database

Manager

Employee

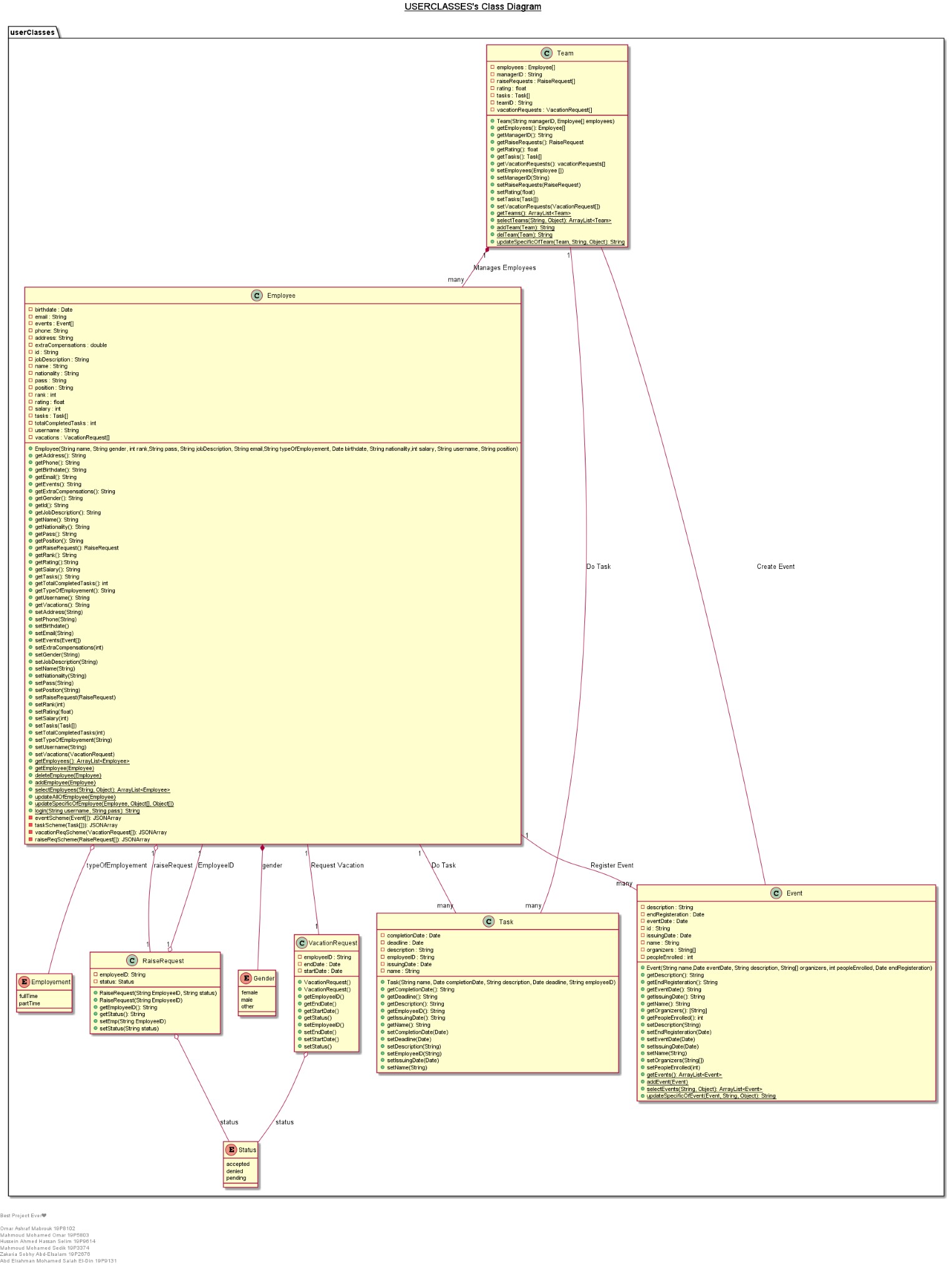
**Merged Architecture:**

GUI

(Software Frontend)

View

NETWORK



Controller

NETWORK

Model

Employee

Manager

Database

The merged architecture divides the program structure into three parts represented the MVC structure, while the connection between the parts of the program is showed in a client server structure to show the interconnection of the program parts.