**By: Rahul Barolia**

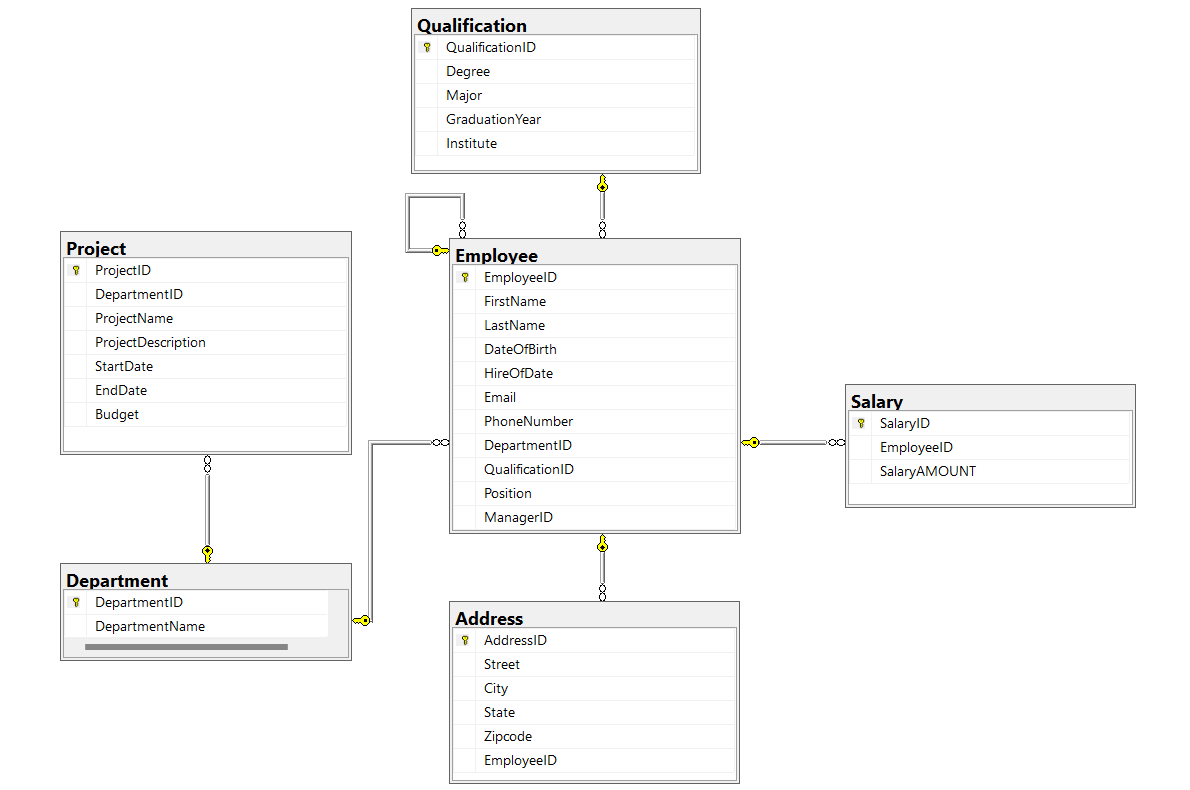
Employee Management System

**Data:**

In this project, I have used employee data and how to manage this data in company.

So, According to its corresponding entities are Employee, Department, Project, Salary, Qualification, Address. All these entities have relationships among themselves. So, we can easily get all the information about the employee.

**ER diagram of Employee Information**

****

**Relationships:**

* One-to- many relationship between Employee and address.
* One -to-many relationship between Employee and Qualification.
* Many-to-many relationship between Employee and Department.
* Many-to-many relationships between Employee and project.
* One-to-many relationship between Employee and Salary.

**API:**

Application Programming interface is employed for communication between different software systems.

In this project, APIs are used define the endpoints that allow external systems or services to interact with the Employee Management System.

**Endpoints:**

* Endpoints are specific URLs that an API exposes for performing various operations.
* The project defines endpoints for CRUD operations on employee records.

**Postman:**

* Postman is a popular API development and testing tool.
* It is used to test the API endpoints of the Employee Management System, ensuring that they respond correctly to different requests and handle various scenarios.

**HTTP Request methods to perform CRUD operations:**

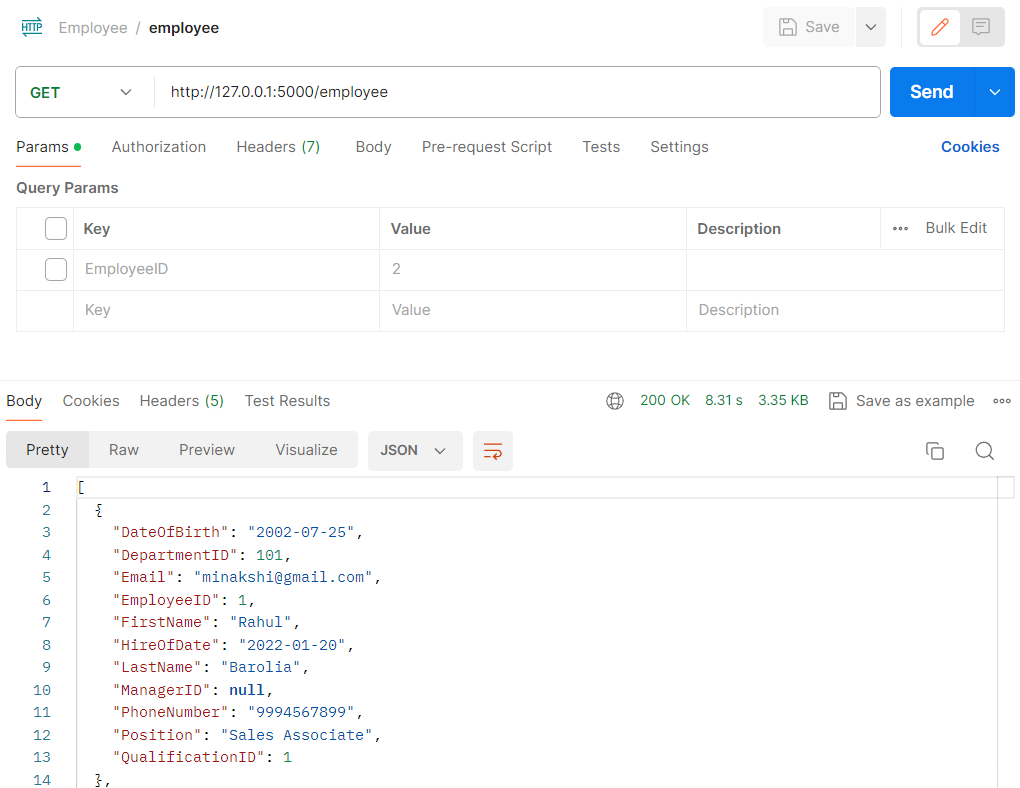
* **GET Request:**To retrieve or fetch data
* **POST Request:**To add data
* **PUT Request:**To update data
* **DELETE Request:**For deleting data
* **PATCH Request:** PATCH is a method of modifying resources where the client sends partial data that is to be updated without modifying the entire data.

**Endpoints used in project:**

* [/employee](http://127.0.0.1:5000/employee)

* In this endpoint I have managed employee data like employee name, phone number etc.
* From this endpoint we can fetch employee data from the server and easily perform operations like GET, POST, etc.

**For example, to retrieve of employee data using GET Request:**

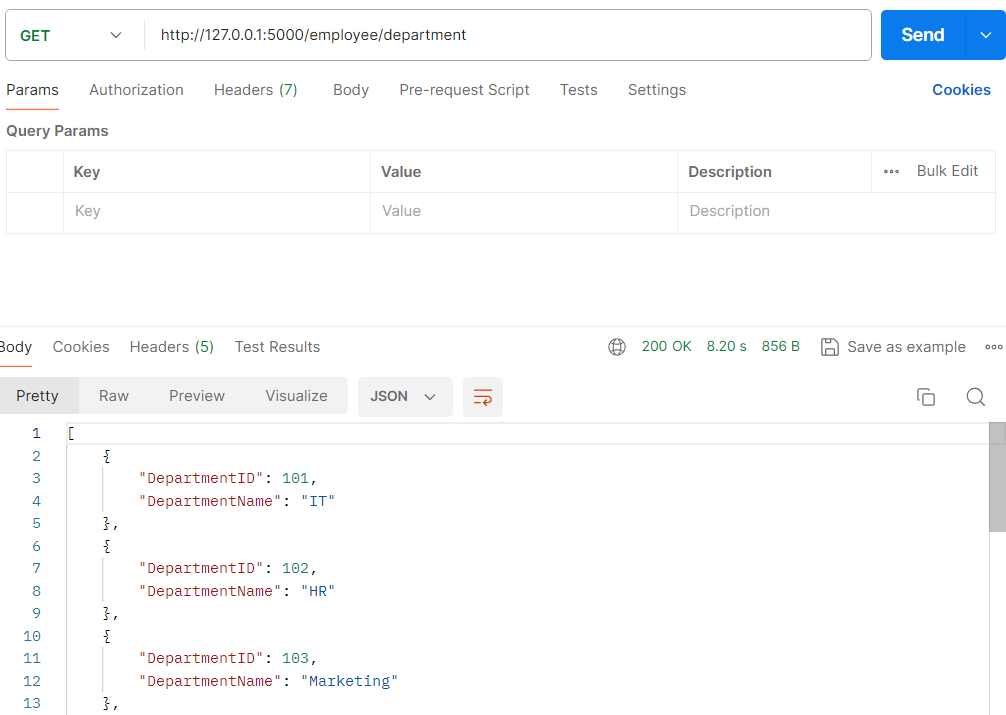


* [/employee/address](http://127.0.0.1:5000/employee/address)
  + It is also important to maintain the address of the employee in an organization.
  + So, the employee's address is maintained in this endpoint and this endpoint can be used according to the employee's address.

**For example, to retrieve a list of all address of employee using GET Request:**

* [/employee/department](http://127.0.0.1:5000/employee/department)
  + The employee's department is managed in this endpoint.
  + With this API endpoint, we can easily manage the department data like get department, add department, update department, delete any department.

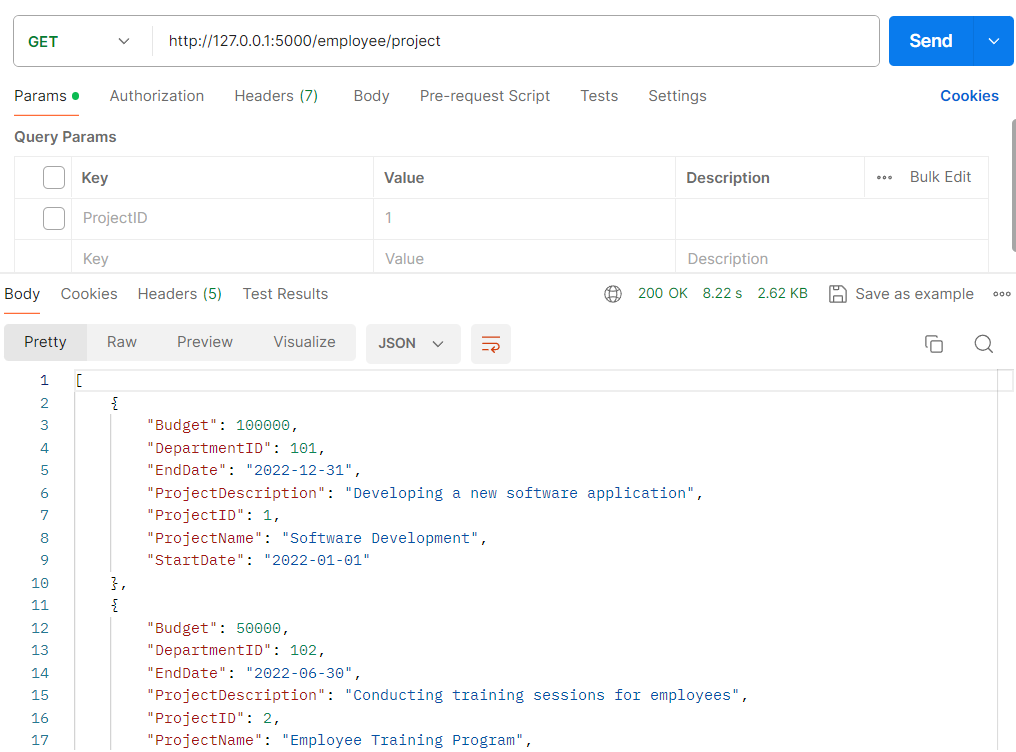
**For example, to retrieve department of employee using GET Request:**

****

* [/employee/project](http://127.0.0.1:5000/employee/project)

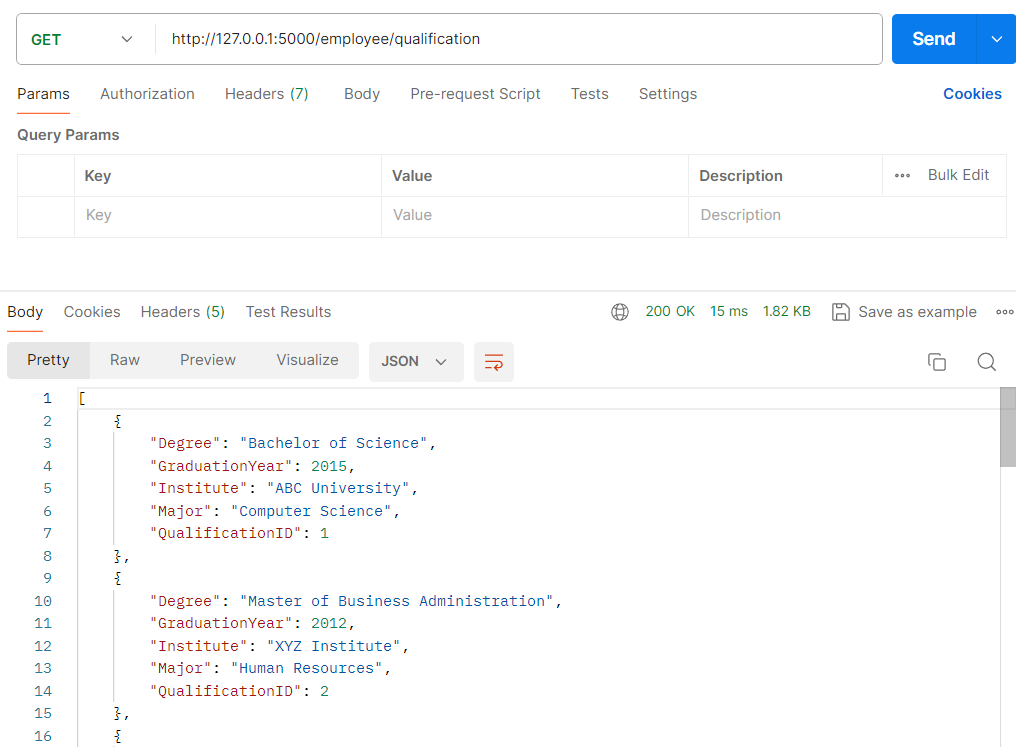
* + With this API endpoint, we can easily manage the project data like get project, add project, update project, delete any project.
  + From this endpoint, we can check the progress of the employee and which department has done good projects for the organization and in less budget so that the growth of the company can be greater.

**For example, I want to retrieve all projects record:**

****

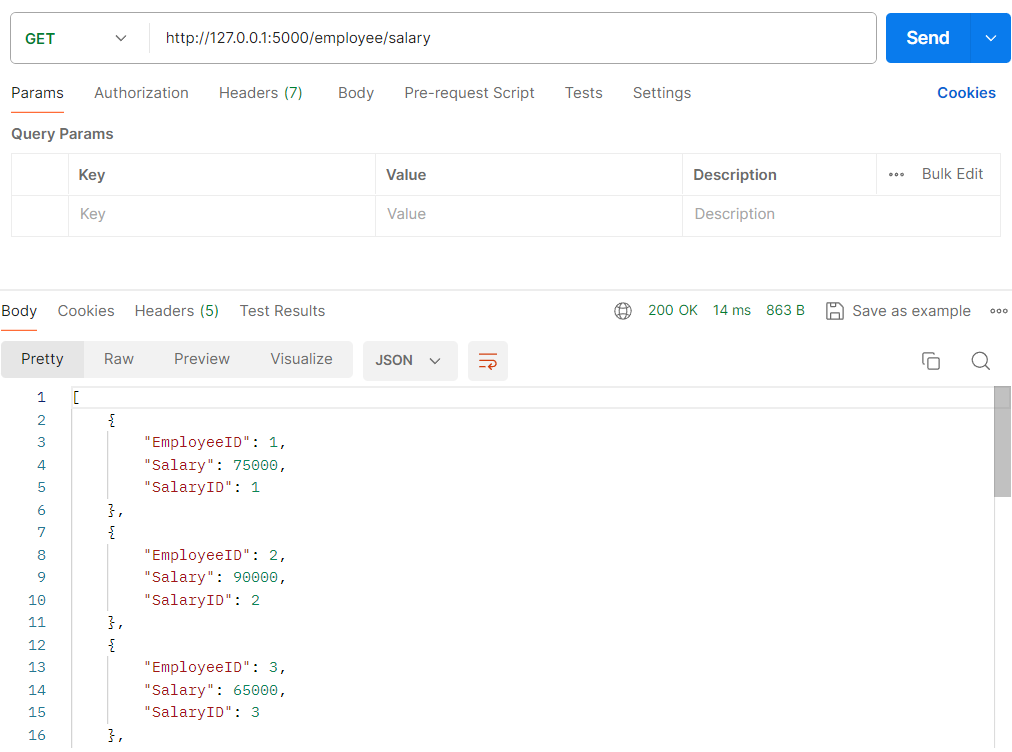
* [/employee/qualification](http://127.0.0.1:5000/employee/qualification)
  + In this endpoint we can see the record of employee's qualification, which employee has what qualification.
  + And whenever new employees come to the organization, the data of their qualifications can be easily managed by using this endpoint.
  + Or if you have to make some changes in the data of present employees, you can easily manage it.

**For example, I want to see the qualifications of the employee.**

****

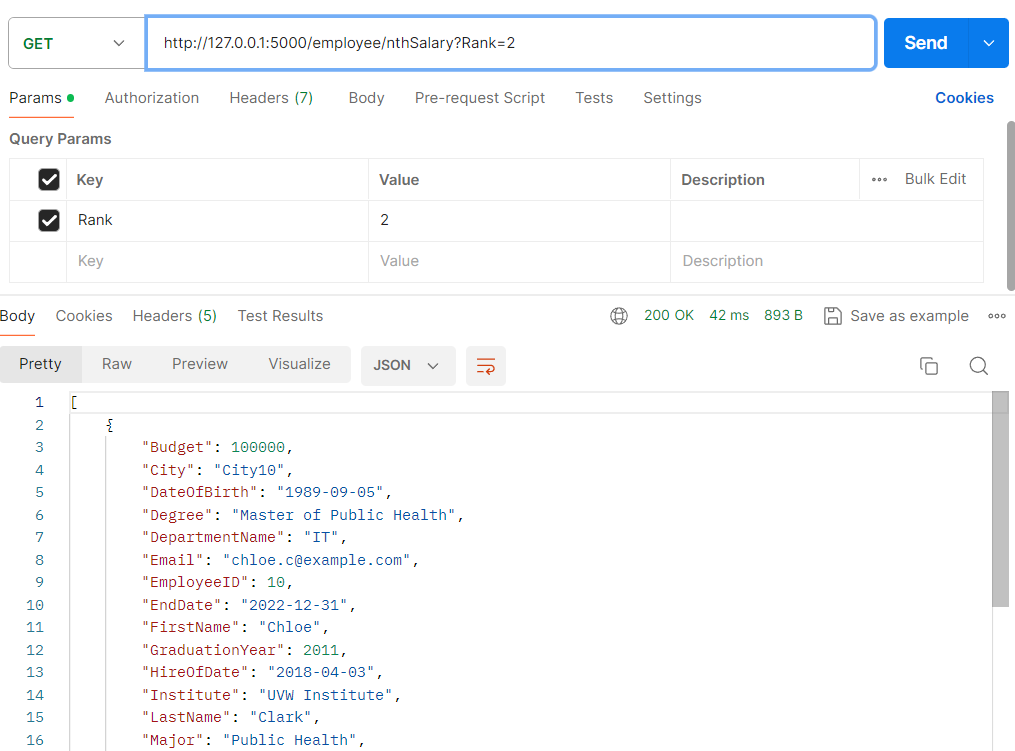
* [/employee/salary](http://127.0.0.1:5000/employee/salary)
  + In this endpoint we can see the employee's salary records.
  + And you can easily perform some operations in the employee's salary like add salary, update salary, delete salary.

**For example, I want to see the employee's salary record.**

****

* [/employee/nthSalary?Rank=3](http://127.0.0.1:5000/employee/nthSalary?Rank=3)
  + In this endpoint, to retrieve a list of employee whose salary is nth salary.
  + We can find out which employee's salary is more or less by using this endpoint.

**For example, we must find out the second highest salary in the organization.**



**Authentication and Authorization mechanisms:**

**Authentication:**

* + Authentication is the process of verifying the identity of users attempting to access the employee management system. It ensures that only authenticated users with valid credentials (such as username and password) are granted access to the system. Authentication helps in preventing unauthorized access and maintaining the security of sensitive employee data.

**Authorization:**

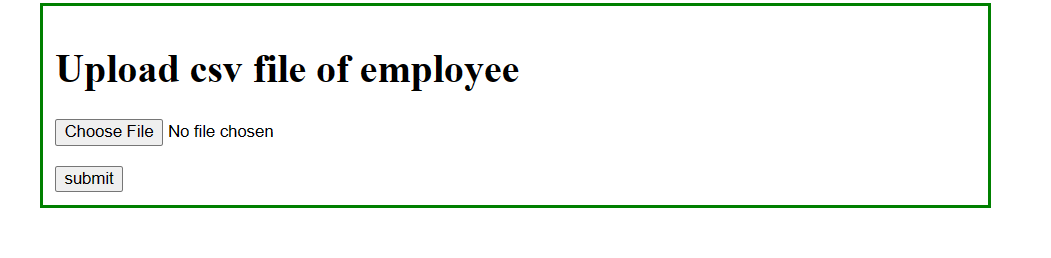
* + Authorization is the process of determining the level of access and permissions granted to authenticated users within the employee management system. It defines what actions users are allowed to perform based on their roles, responsibilities, and privileges. Authorization helps in enforcing access control policies and ensuring that users can only access the data and functionalities relevant to their roles within the organization.

**Bulk upload csv file:**

* + Bulk uploading CSV files for employee data offers a convenient and efficient way to manage large volumes of employee information.

**Endpoint used for bulk upload csv file:**

* + - **/employee/csvdata**

****

**Unit Testing:**

* Unit testing is an essential part of software development to verify that individual units of code work as intended.
* In this project, unit testing is implemented using the built-in testing framework in Flask to ensure the reliability of the application.

**token\_get:**

* + This method retrives token from a file and set it as the authorization header.
  + This token is used in all testing api.

**Testing methods for employee:**

**test\_1\_get\_all\_employee:**

* + This method tests the GET request to retrieve all employees from the employee table. It verifies that the server responds with a status code of 200 (OK) upon successful retrieval.

**test\_2\_get\_particular\_employee:**

* + this method tests the GET request to retrieve a specific employee by ID from the employee table. It ensures that the server responds with a status code of 200 (OK) upon successful retrieval of the employee.

**test\_3\_add\_employee:**

* + This method tests the POST request to add a new employee to the employee table. It verifies that the server responds with a status code of 201 (Created) upon successful addition of the employee.

**test\_4\_update\_employee:**

* + This method tests the PUT request to update an existing employee in the employee table. It ensures that the server responds with a status code of 200 (OK) upon successful update of the employee information.

**test\_5\_patch\_method\_employee:**

* + This method tests the PATCH request to partially update an existing employee in the employee table. It verifies that the server responds with a status code of 200 (OK) upon successful partial update of the employee information.

**test\_6\_delete\_employee:**

* + This method tests the DELETE request to delete an existing employee from the employee table. It ensures that the server responds with a status code of 200 (OK) upon successful deletion of the employee.

**Testing methods for department:**

**test\_1\_get\_all\_department:**

* + Tests the GET request to retrieve all departments from the department table. It verifies that the server responds with a status code of 200 (OK) upon successful retrieval.

**test\_2\_get\_particular\_department:**

* + Tests the GET request to retrieve a specific department by ID from the department table. It ensures that the server responds with a status code of 200 (OK) upon successful retrieval of the department.

**test\_3\_add\_department:**

* + Tests the POST request to add a new department to the department table. It verifies that the server responds with a status code of 201 (Created) upon successful addition of the department.

**test\_4\_update\_department:**

* + Tests the PUT request to update an existing department in the department table. It ensures that the server responds with a status code of 200 (OK) upon successful update of the department information.

**test\_5\_delete\_department:**

* + Tests the DELETE request to delete an existing department from the department management system. It verifies that the server responds with a status code of 200 (OK) upon successful deletion of the department.

**Testing methods for address:**

**test\_1\_get\_all\_address:**

* + This test verifies the functionality of retrieving all addresses from the address table. It sends a GET request to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_2\_get\_particular\_address:**

* + This test verifies the functionality of retrieving a particular address by its ID from the address table. It sends a GET request with a specific AddressID parameter and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_3\_add\_address:**

* + This test verifies the functionality of adding a new address to the address table. It sends a POST request with address data to the API endpoint and asserts that the server responds with a status code of 201 (Created), indicating that the address was successfully added.

**test\_4\_update\_address:**

* + This test verifies the functionality of updating an existing address in the address table. It sends a PUT request with updated address data and the AddressID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the address was successfully updated.

**test\_5\_delete\_ address:**

* + This test verifies the functionality of deleting an existing address from the address table. It sends a DELETE request with the AddressID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the address was successfully deleted.

**Testing methods for salary:**

**test\_1\_get\_all\_salary:**

* + This test verifies the functionality of retrieving all salaries from the salary table. It sends a GET request to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_2\_get\_particular\_salary:**

* + This test verifies the functionality of retrieving a particular salary by its ID from the salary table. It sends a GET request with a specific SalaryID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_3\_add\_salary:**

* + This test verifies the functionality of adding a new salary to the salary table. It sends a POST request with salary data to the API endpoint and asserts that the server responds with a status code of 201 (Created), indicating that the salary was successfully added.

**test\_4\_update\_salary:**

* + This test verifies the functionality of updating an existing salary in the salary table. It sends a PUT request with updated salary data and the SalaryID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the salary was successfully updated.

**Testing methods for qualification:**

**test\_1\_get\_all\_qualification:**

* + This test verifies the functionality of retrieving all qualifications from the qualification table. It sends a GET request to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_2\_get\_particular\_qualification:**

* + This test verifies the functionality of retrieving a particular qualification by its ID from the qualification table. It sends a GET request with a specific QualificationID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_3\_add\_qualification:**

* + This test verifies the functionality of adding a new qualification to the qualification table. It sends a POST request with qualification data to the API endpoint and asserts that the server responds with a status code of 201 (Created), indicating that the qualification was successfully added.

**test\_4\_update\_qualification:**

* + This test verifies the functionality of updating an existing qualification in the qualification table. It sends a PUT request with updated qualification data and the QualificationID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the qualification was successfully updated.

**test\_5\_delete\_qualification:**

* + This test verifies the functionality of deleting an existing qualification from the qualification table. It sends a DELETE request with the QualificationID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the qualification was successfully deleted.

**Testing methods for project:**

**test\_1\_get\_all\_project:**

* + This test verifies the functionality of retrieving all projects from the project table. It sends a GET request to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_2\_get\_particular\_project:**

* + This test verifies the functionality of retrieving a particular project by its ID from the project table. It sends a GET request with a specific SalaryID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the retrieval was successful.

**test\_3\_add\_project:**

* + This test verifies the functionality of adding a new project to the project table. It sends a POST request with project data to the API endpoint and asserts that the server responds with a status code of 201 (Created), indicating that the project was successfully added.

**test\_4\_update\_project:**

* + This test verifies the functionality of updating an existing project in the project table. It sends a PUT request with updated project data and the SalaryID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the project was successfully updated.

**test\_5\_delete\_project:**

* + This test verifies the functionality of deleting an existing project from the project table. It sends a DELETE request with the ProjectID parameter to the API endpoint and asserts that the server responds with a status code of 200 (OK), indicating that the project was successfully deleted.

**Conclusion:**The Employee Management System provides a robust solution for managing employee information within an organization. The use of Flask, API, endpoints, unit testing, and Postman contributes to the development of a reliable and scalable system.