API Documentation: Point of Contact Lookup for Collaboration Framework

the organization

November 2024

1 Introduction

The API described in this document is designed to facilitate collaboration within the organization, which has experienced rapid growth. This growth has led to a challenge: finding the right point of contact across different teams and locations has become increasingly difficult. This API addresses that challenge by providing an efficient way to query for a point of contact for any given product, thereby enhancing collaboration and productivity.

2 Purpose

The main purpose of this API is to provide the organization employees with a quick and efficient way to find the relevant point of contact for a product. Employees can use the product name or repository name to obtain contact details, which can help overcome issues related to collaboration across teams in different geographical locations.

3 API Overview

The Point of Contact Lookup API is designed to provide information regarding the individual responsible for a particular product. Users of this API will be able to retrieve contact information using either the product name or the GitHub repository name. This is especially useful in scenarios where employees from one department need to reach out to individuals who own, support, or develop a particular product.

3.1 Use Case

Consider an employee working in one office location. She needs support from a team working on a security API based in another office location. She has not worked with anyone from that team before, and she requires a quick way to identify a point of contact for her questions. The Point of Contact Lookup API helps her query for the details of the person she needs to collaborate with, based on the product name.

4 Endpoints and Usage

The API exposes the following key endpoints:

4.1 GET /

This endpoint allows users to query for a point of contact by providing either the product name, GitHub repository name, or location.

- **Request Parameters:**

  - `product_name`: (Optional) The name of the product for which contact information is needed.
  - `repository_name`: (Optional) The name of the GitHub repository for the product.
  - `location`: (Optional) The location to filter employees by geographical region.

  At least one of these parameters must be provided.

- **Response:** The response will be an object containing an array of employees matching the specified criteria. Each employee object will include the following details:
  - `first_name`: The first name of the point of contact.
  - `last_name`: The last name of the point of contact.
  - `email`: The email address of the point of contact.
  - `chat_username`: The chat username (e.g., Slack handle) of the point of contact.
  - `location`: The geographical location of the point of contact.
  - `title`: The title or role of the point of contact within the company.
- **Response Format:** The response will be in JSON format.
- **Error Handling:**
  - If no product or repository name is provided, the API will return a `400 Bad Request` response.
  - If no employees are found, the API will return a `404 Not Found` response with an appropriate error message.
  - If there is an internal server error, the API will return a `500 Internal Server Error` response with error details for debugging.

4.2 POST /

This endpoint allows users to add a product along with its associated employees to the database.

- **Request Body:** The request body should be in JSON format and must include the following keys:
  - `product`: An object containing the product details.
    - `product_name`: The name of the product. This must be a non-empty string.
    - `repo_name`: (Optional) The repository name associated with the product.
  - `employees`: A list of employee objects associated with the product.
    - Each employee object must include:
      - `first_name`, `last_name`, `email`, `chat_username`, `location`, `title`
- **Response:** If successful, the API will return a `201 Created` response, including a message indicating the data was added successfully.
- **Error Handling:**
  - If the request payload is empty or if required keys are missing, the API will return a `400 Bad Request` response with an error message specifying the issue.
  - If an employee or product already exists, a `409 Conflict` response will be returned.
  - If there is an internal server error, the API will return a `500 Internal Server Error` response.

4.3 DELETE /email/{email}

This endpoint allows users to delete an employee and their associated assignments by providing the employee's email.

- **URL Parameter:**
  - `email`: The email address of the employee to be deleted.
- **Response:** The API will return a `200 OK` response if the employee is successfully deleted.
- **Error Handling:**
  - If the employee does not exist, a `404 Not Found` response will be returned.
  - If there is an internal server error, the API will return a `500 Internal Server Error` response with error details.

5 Data Model

The data model used by the API includes three main entities:

5.1 Employees

This table stores information about employees, including their unique identifiers, first and last names, email addresses, chat usernames, locations, and job titles.

- **Fields:**
  - `employee_id`: A unique identifier for the employee.
  - `first_name`: The first name of the employee.
  - `last_name`: The last name of the employee.
  - `email`: The email address of the employee.
  - `chat_username`: The chat username of the employee.
  - `location`: The geographical location of the employee.
  - `title`: The job title or role of the employee.

5.2 Products

This table contains information about the products, such as the product ID, product name, and associated repository name.

- **Fields:**
  - `product_id`: A unique identifier for the product.
  - `product_name`: The name of the product.
  - `repository_name`: The name of the GitHub repository associated with the product.

5.3 Employee Assignments

This table tracks the assignment of employees to specific products, using references to both the employee and product IDs.

- **Fields:**
  - `employee_id`: A reference to the unique identifier of an employee.
  - `product_id`: A reference to the unique identifier of a product.

5.4 Entity-Relationship Analysis

The relationships between these tables are crucial for establishing which employees are responsible for which products. Each employee can be assigned to multiple products, and each product can have multiple employees working on it. This many-to-many relationship is managed by the extttemp_assignment table.

The extttemployees and extttproducts tables represent entities, while the extttemp_assignment table represents the association between these entities. By maintaining foreign key constraints, the database enforces referential integrity, ensuring that an employee or product cannot be deleted without also updating their relationships.

6 Detailed Flow of API Usage

The Point of Contact Lookup API is designed to be simple to use, yet effective in providing the necessary information quickly. Below is a more in-depth explanation of how to use the API effectively.

6.1 Step 1: Authentication

Before making any requests to the API, ensure you have a valid API key. The API key is used for authentication to verify that the request is coming from an authorized user.

6.2 Step 2: Making a Request

To find the point of contact for a product, make a GET request to the appropriate endpoint. You must include at least one of the following parameters: extttproduct_name, extttrepository_name, or extttlocation.

6.3 Step 3: Database Lookup

Upon receiving a valid request, the API performs the following operations:

- **Product Identification**: The API first searches the extttproducts table to find the extttproduct_id associated with the provided extttproduct_name or extttrepository_name. This ensures that the product exists in the system.
- **Employee Assignment Retrieval**: Once the product is identified, the API queries the extttemp_assignment table to find all extttemployee_id values associated with the extttproduct_id. This step determines which employees are assigned to work on the given product.
- **Employee Information Fetching**: Using the extttemployee_id values retrieved from the previous step, the API then queries the extttemployees table to gather detailed information about each employee assigned to the product.

6.4 Step 4: Handling the Response

The response will be in JSON format, containing all the necessary details of the point of contact. If the product or repository name provided is invalid, appropriate error messages will be returned, which should be handled by the calling application to provide meaningful feedback to the user.

6.5 Step 5: Error Handling and Logging

Error responses such as `400 Bad Request`, `404 Not Found`, and `500 Internal Server Error` should be appropriately handled. Logging mechanisms should be put in place to track errors for further debugging and improvement of the API. For example, every failed API request should generate a log entry containing the time of the request, the input parameters, and the error details. This can be very useful for diagnosing issues and improving system reliability.

7 API Functions Explanation

This section provides detailed explanations for each of the primary functions used in the API.

7.1 Function: get_employees

The `get_employees()` function is responsible for handling GET requests to the API. This function allows users to retrieve information about employees based on either a product name, repository name, or location.

- **Parameters**:
  - `product_name` (Optional)
  - `repository_name` (Optional)
  - `location` (Optional)
- **Functionality**: The function first validates the incoming request to ensure that at least one of the parameters is provided. It then searches the `products` table to find the product details. Once the product is identified, it retrieves employee assignments from the `emp_assignment` table. Finally, it uses the `employee_id` values to gather detailed employee information.
- **Error Handling**: `get_employees()` handles cases where no parameters are provided by returning a `400 Bad Request` response. If no employees are found, it returns a `404 Not Found` response.

7.2 Function: delete_employee_by_email(email)

The `delete_employee_by_email(email)` function handles DELETE requests for removing an employee from the database based on their email address.

- **Parameters**:
  - `email` (Required): The email address of the employee to be deleted.
- **Functionality**: This function accepts the employee email as input and verifies if the employee exists in the database. If the employee is found, it deletes the employee record as well as any associated assignments from the `emp_assignment` table.
- **Error Handling**: If the employee does not exist, the function returns a `404 Not Found` response. In case of database issues or other server-side errors, a `500 Internal Server Error` response is returned.

7.3 Function: add_product_with_employees_api()

The `add_product_with_employees_api()` function processes POST requests for adding a product and its associated employees to the database.

- **Parameters**:
  - `product`: A dictionary containing the product details such as `product_name` and `repo_name`.
  - `employees`: A list of dictionaries, each containing employee information.
- **Functionality**: The function validates the input to ensure all required fields are provided. It then creates a new product entry in the `products` table and assigns employees to the product using the `emp_assignment` table. This function ensures that existing employees or products are not duplicated.
- **Error Handling**: If any required field is missing or if a product/employee already exists, the function returns a `400 Bad Request` or `409 Conflict` response, respectively. In case of any other issues, a `500 Internal Server Error` response is returned.

8 Conclusion

In conclusion, the Point of Contact Lookup API offers an efficient solution for managing employee-product relationships. By providing quick and accurate information about the point of contact for each product, it helps distributed teams collaborate more effectively and ensures that everyone knows exactly who to reach out to for support.

Database Explanation

1 Introduction

The provided database consists of three main collections: `emp_assignment`, `employees`, and `products`. This structure is designed to facilitate interactions and queries, particularly in scenarios where engineers need to track project assignments and manage collaboration across different teams. Below is a detailed explanation of each collection and its relationships.

2 Collections

2.1 `emp_assignment` Collection

This collection serves as a mapping table linking employees with products. Each entry includes:

- `employee_id`: A unique identifier for each employee.
- `product_id`: A reference to the product that the employee is assigned to.

The purpose of this collection is to track employee assignments for various products. For instance, an engineer can determine all employees assigned to a product by filtering on `product_id`.

Example:

```
{
  "employee_id": "550e8400-e29b-41d4-a716-446655440000",
  "product_id": 1
}
```

This entry links an employee (with ID `"550e8400-e29b-41d4-a716-446655440000"`) to product ID 1.

2.2 `employees` Collection

The `employees` collection provides detailed information about each employee, such as:

- `employee_id`: A unique identifier used to link to the `emp_assignment` collection.
- `first_name`, `last_name`: Full name of the employee.
- `email`: Contact email.
- `chat_username`: Chat handle for quick internal communication.
- `location`: The city and state of the employee.
- `title`: The employee's position within the company.

Example:

```
{
  "employee_id": "550e8400-e29b-41d4-a716-446655440000",
  "first_name": "Michael",
  "last_name": "Anderson",
  "email": "michael.anderson@corpglobal.com",
  "chat_username": "michael.a",
  "location": "New York, NY",
  "title": "Senior Vice President, Product Development"
}
```

This record provides details about Michael Anderson, including his title, email, and chat handle.

2.3 `products` Collection

The `products` collection provides information about each product being developed:

- `product_id`: Unique identifier for each product, linked to `emp_assignment`.
- `product_name`: The name of the product.
- `repository_name`: The name of the code repository associated with the product.

Example:

```
{
  "product_id": 1,
  "product_name": "Global CRM Platform",
  "repository_name": "crm-platform"
}
```

This record details the product named "Global CRM Platform" and specifies the related repository `crm-platform`.

3 Relationships Between Collections

The database structure follows a relational model that can be explained as follows:

1. **Employee Assignments to Products**: The `emp_assignment` collection uses foreign keys (`employee_id` and `product_id`) to link employees to products, establishing a many-to-many relationship between employees and products.
2. **Finding Point of Contact**: By joining `employees` with `emp_assignment`, you can find the point of contact for a given product. For instance, to find employees assigned to "Global CRM Platform":
   - Query `products` to get `product_id`.
   - Use `product_id` to find relevant `employee_id` entries in `emp_assignment`.
   - Retrieve employee details from the `employees` collection using the `employee_id`.

4 API Use Case

The accompanying API is designed to facilitate collaboration by allowing users to find a point of contact for a product. The inputs for the API can be either the product name or repository name. The output includes the employee's first and last name, email, chat username, location, and role/title, which are all fields available in the `employees` collection.

5 Example Query Workflow

1. **Query Input**: Product name = “AI-powered Analytics Suite”.
2. **Step 1**: Query `products` to get `product_id` for “AI-powered Analytics Suite”.
3. **Step 2**: Use the `product_id` to find associated `employee_id` entries in `emp_assignment`.
4. **Step 3**: Retrieve detailed employee information from `employees` using the `employee_id`.

6 Suggested Enhancements

- **Indexing**: Adding indexes to fields like `employee_id`, `product_id`, and `repository_name` could improve query performance as the dataset grows.
- **Normalization**: The current structure is normalized, but denormalizing some fields may reduce the number of joins needed for read-heavy operations.

This detailed breakdown should assist engineers in leveraging the provided database to manage project assignments and enhance collaboration effectively.