Name: Harshitha Madhugiri Linganna

ID: 4179972

**API Documentation**

This API allows clients to manage and simulate production plans, consisting of formulas and sequences of formulas (plans).

When executing the GET requests for specific IDs in Postman please disable the request body, as it sometimes.

**Formulas API**

1. **Add a New Formula**

**Method:** POST

**Path:** /formulas

**Description:** Adds a new formula to the system.

**Request Body**: A JSON object representing the formula. A formula consists of a collection of inputs and a collection of outputs with resources and quantities.

**Example:**

{

"inputs": [

{

"resource": "iron ore",

"quantity": 2

}

],

"outputs": [

{

"resource": "iron bar",

"quantity": 1

}

]

}

**Response Codes and Body Contents:**

201 - Formula added successfully. The response body contains the formula ID

**Response Body**

/formulas/{formulaId}– Sample formulas/0

**Note:**

* The POST Method is appropriate for creating new resources on the server
* The /formulas path is clear and intuitive, indicating that this endpoint is for managing formulas.
* The request Body format makes it easy to understand the required data structure for creating a formula.
* The status code 201 is appropriate for indicating that a resource has been successfully created
* This formulaId will be mapped to the request body formula.
* The response body like /formulas/formulaId improves readability and helps quick mapping.

1. **Get All Formulas**

**Method**: GET

**Path**: /formulas

**Description**: Retrieves a list of all formulas.

**Parameters**: None.

**Response Codes and Body Contents**:

200 - Successful request. The response body contains an array of formulas.

{

"0": {

"inputs": [

{

"resource": "iron ore",

"quantity": 2

}

],

"outputs": [

{

"resource": "iron bar",

"quantity": 1

}

]

},

"1": {

"inputs": [

{

"resource": "iron ore",

"quantity": 3

},

{

"resource": "coal",

"quantity": 1

}

],

"outputs": [

{

"resource": "steel bar",

"quantity": 1

}

]

}

}

1. **See the inputs and outputs of a specific formula**.

**Method:** GET

**Path:** /formulas/:formulaId

**Description:** Retrieves details of a specific formula by its ID.

**Path parameter:**

**formulaId:** The ID of the formula to retrieve**.**

**Response Codes and Body Contents:**

200 - Successful request. The response body contains the specific formula details which consist of collections of input and output.

404 - Formula not found. The specified formula ID does not exist.

500 - Internal server error.

**Response Body**

{

"inputs": [

{

"resource": "iron ore",

"quantity": 3

},

{

"resource": "coal",

"quantity": 1

}

],

"outputs": [

{

"resource": "steel bar",

"quantity": 1

}

]

}

**Note:**

* Chosen the GET method, which is appropriate for retrieving existing resources from the server.
* The path includes a parameter placeholder:formulaId, which is a good practice for identifying the specific formula to retrieve.
* path parameter formulaId, which is the ID of the formula to retrieve is a suitable choice as it allows clients to specify which formula they want to access.
* 200 OK: This status code is appropriate for indicating that the request was successful.
* 404 Not Found: This status code is used when the specified formula ID does not exist, providing meaningful feedback to the client

1. **List which plans contain a specific formula**

**Method:** GET

**Path:** /formulas/:formulaId/plans

**Description:** Lists all plans that contain a specific formula.

**Parameters:**

**Path parameter:**

**formulaId:** The ID of the formula to search for in plans.

**Response Codes and Body Contents:**

200 - Successful request. The response body contains an array of plans containing the formula.

404 - Formula not found. The specified formula ID does not exist.

500 - Internal server error.

**Response Body**

[

"PlanA",

"PlanB"

]

**Note**:

* Since plans are an ordered sequence of formulas. We need to add a formulas and plans first to perform this API.
* The API lists all plans of a specific formula, thus, we need formulaId from list of formulas at list the plans hence going with this /formulas/:formulaId/plans is the right choice.

**Plans API**

1. **Add a New Plan**

**Method**: POST

**Path**: /plans

**Description:** Adds a new production plan.

**Parameters:**

**Request Body**: A JSON object representing the plan, which consists of an ordered sequence of the formulas.

{

"name": "PlanA",

"formulas": ["0", "1"]

}

**Response Codes and Body Contents**:

201 - Plan added successfully. The response body contains the plan ID

400 - Bad request. The request body is missing or invalid, or formula IDs are invalid.

500 - Internal server error.

The planId is mapped to the new Plan created.

**Response Body:**

plans/{planId } -> plans/0

Note:

* The POST Method is appropriate for creating new resources on the server
* The /formulas path is clear and intuitive, indicating that this endpoint is for managing formulas.
* The request Body format makes it easy to understand the required data structure for creating a formula.

1. **Get All Plans**

**Method**: GET

**Path**: /plans

**Description**: Retrieves a list of all production plans.

**Parameters**: None.

**Response Codes and Body Contents**:

200 - Successful request. The response body contains plans with a sequence of formulas.

500 - Internal server error.

**Response Body**:

{

"0": {

"name": "PlanA",

"formulas": [

"1",

"2"

]

},

"1": {

"name": "PlanB",

"formulas": [

"0"

]

}

}

1. **Append a formula to the end of a plan.**

**Method:** POST

**Path:** /plans/:planId/formula

**Description**: Appends a formula to the end of a specific plan's sequence.

**Parameters:**

**Path parameter**:

**planId:** non-empty number, The ID of the plan to which the formula should be appended.

**Request Body**: A JSON object containing the formulaId to append.

{ "formulaId" : 2}

**Required**: The formula must exist to append its ID to the plan

**Response Codes and Body Contents:**

200 - Formula appended successfully. The response body contains the plan with the updated formulas.

400 - Bad request. The plan or formula does not exist.

500 - Internal server error.

**Response body**: /plans/1/ formula

{

{

"0": {

"name": "PlanA",

"formulas": [

"1",

"2"

]

},

"1": {

"name": "PlanB",

"formulas": [

"0",

**“2”**

]

}

}

Note:

* We need to append the formula to the specific plan, hence we need planID from plans followed by formula - /plans/:planId/formula,
* Appends formula Id to the plan, this is convenient compared to adding formula as is because it improves accessibility.
* This method allows us to append existing formulas. if we want to add a new formula we must create the formula and then append it.

1. **Replace a formula anywhere in the sequence of formulas associated with a plan**

**Method**: PATCH

**Path**: /plans/:planId/formulas/:formulaId

**Description**: Replaces a formula anywhere in the sequence of formulas associated with a plan.

**Parameters**:

**Path parameters:**

planId: The ID of the plan to replace the formula in.

formulaId: The ID of the formula to be replaced.

**Request Body**: A JSON object containing the newFormulaId to replace the old formula. { "newFormulaId": 0}

**Required**: A new formula should already be created to do this operation.

**Response Codes and Body Contents**:

200 - Formula replaced successfully. The response body contains the updated plan.

404 - Bad request. The plan, formula, or new formula does not exist.

500 - Internal server error.

Response body: /plans/0/formulas/**1**

{

"0": {

"name": "PlanA",

"formulas": [

**"0",**

"2"

]

},

"1": {

"name": "PlanB",

"formulas": [

"0",

“1”

]

}

}

Note:

* plans/:planId/formulas/:formulaId – is appropriate and self-explanatory as we are trying to replace a formula id with the newformulaID from the specific plan.
* The request body has newformulaId, for the replacement.
* A new formula must be created before we use it for replacement.
* PATCH is used to apply partial updates to a resource, meaning that only the fields that need to be changed are sent in the request body.in this case formulaID

1. **List all the formulas contained in a plan**

**Method**: GET

**Path**: /plans/:planId/formulas

**Description**: Lists all formulas in a specific plan.

**Parameters**:

**Path parameter**:

planId: The ID of the plan to list formulas from.

**Response Codes and Body Contents:**

200 - Successful request. The response body contains an array of formulas of the plan.

404 - Plan not found. The specified plan ID does not exist.

500 - Internal server error.

Response Body:

[

{

"formulaId": "0",

"inputs": [

{

"resource": "iron ore",

"quantity": 2

}

],

"outputs": [

{

"resource": "iron bar",

"quantity": 1

}

]

}

]

Note:

* We will use /plans/:planId/formulas to list all formulas for a given planID.
* The request body has the list of formulas and not the ids for the output readability.

1. **Delete a Plan**

**Method**: DELETE

**Path**: /plans/:planId

**Description**: Deletes a specific production plan by its ID.

**Parameters:**

**Path parameter**:

**planId**: The ID of the plan to delete.

**Response Codes and Body Contents:**

204 - Plan deleted successfully. No response body.

404 - Plan not found. The specified plan ID does not exist.

500 - Internal server error.

Note:

* response with status code 204 (No Content) typically indicates that the request has been successfully processed, and there is no content to include in the response body. Therefore, it's expected that a response with status code 204 does not have a response body..
* /plans/:planId- deletes the specific plan from the plans.
* When we delete plans the IDs won’t be sequential, as it is expected when we are using the incrementing method.
* DELETE is used to delete the complete resource