**Endpoint (/register)**

{

"username": "newuser",

"password": "password123"

}  
{

"message": "User registered successfully"

}  
  
  
**/profile Endpoint:**

HTTP/1.1 200 OK

Content-Type: application/json

{

"id": 1,

"username": "existinguser",

"email": "user@example.com",

"roles": ["ROLE\_USER"]

}  
Unauthorized Response (HTTP 401 Unauthorized):

HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"error": "Unauthorized",

"message": "Authentication failed. Invalid or expired token."

}  
Not Found Response (HTTP 404 Not Found):

HTTP/1.1 404 Not Found

Content-Type: application/json

{

"error": "Not Found",

"message": "User profile not found."

}  
  
  
**/login Endpoint**Successful Login Response (HTTP 200 OK):

HTTP/1.1 200 OK

Content-Type: application/json

{

"token": "your\_jwt\_token\_here"

}

HTTP/1.1 200 OK

Content-Type: application/json

{

"token": "your\_jwt\_token\_here"

}  
  
Unauthorized Login Response (HTTP 401 Unauthorized):  
HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"error": "Unauthorized",

"message": "Authentication failed. Invalid username or password."

}  
Bad Request Response (HTTP 400 Bad Request):  
HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"error": "Bad Request",

"message": "Request body is malformed or incomplete."

}

**/register Endpoint:**

**S**uccessful Registration Response (HTTP 201 Created)  
HTTP/1.1 201 Created

Content-Type: application/json

{

"message": "User registered successfully"

}

Conflict Response (HTTP 409 Conflict):

HTTP/1.1 409 Conflict

Content-Type: application/json

{

"error": "Conflict",

"message": "Username 'existinguser' already exists."

}

Bad Request Response (HTTP 400 Bad Request):

HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"error": "Bad Request",

"message": "Request body is malformed or incomplete."

}  
  
1. \*\*File Upload Endpoint:\*\*

- \*\*Endpoint:\*\* `/upload`

- \*\*HTTP Method:\*\* POST

- \*\*Request Body:\*\*

```json

{

"file": "<binary data of the file>"

}

```

- \*\*Response Body:\*\*

```json

{

"message": "File uploaded successfully",

"fileId": "<unique file identifier>"

}

```

2. \*\*File Download Endpoint:\*\*

- \*\*Endpoint:\*\* `/download/{fileId}`

- \*\*HTTP Method:\*\* GET

- \*\*Request Body:\*\* None

- \*\*Response Body:\*\*

- The binary data of the requested file.

- Content-Disposition header should be set to suggest the filename.

3. \*\*List All Files Endpoint:\*\*

- \*\*Endpoint:\*\* `/files`

- \*\*HTTP Method:\*\* GET

- \*\*Request Body:\*\* None

- \*\*Response Body:\*\*

```json

{

"files": [

{

"fileId": "<unique file identifier>",

"fileName": "example.txt",

"fileSize": "1024 KB",

"contentType": "text/plain",

"uploadDate": "2023-09-10T12:00:00Z"

},

{

"fileId": "<unique file identifier>",

"fileName": "image.jpg",

"fileSize": "2048 KB",

"contentType": "image/jpeg",

"uploadDate": "2023-09-11T14:30:00Z"

}

]

}

```

4. \*\*Delete File Endpoint:\*\*

- \*\*Endpoint:\*\* `/delete/{fileId}`

- \*\*HTTP Method:\*\* DELETE

- \*\*Request Body:\*\* None

- \*\*Response Body:\*\*

```json

{

"message": "File deleted successfully"

}

```

5. \*\*File Metadata Endpoint:\*\*

- \*\*Endpoint:\*\* `/metadata/{fileId}`

- \*\*HTTP Method:\*\* GET

- \*\*Request Body:\*\* None

- \*\*Response Body:\*\*

```json

{

"fileId": "<unique file identifier>",

"fileName": "example.txt",

"fileSize": "1024 KB",

"contentType": "text/plain",

"uploadDate": "2023-09-10T12:00:00Z"

}

```

6. \*\*Update File Metadata Endpoint:\*\*

- \*\*Endpoint:\*\* `/metadata/{fileId}`

- \*\*HTTP Method:\*\* PUT

- \*\*Request Body:\*\*

```json

{

"fileName": "new\_filename.txt",

"contentType": "application/octet-stream"

}

```

- \*\*Response Body:\*\*

{

"message": "Metadata updated successfully"

}

Certainly! When creating a Monitoring and Metrics API using Spring Boot, you can define various endpoints to provide insights into system health and performance. Below, I'll suggest some additional endpoints you can include along with their expected request and response formats in JSON. Keep in mind that the specific details may vary depending on your application's requirements and the monitoring tools you're using.

1. `/metrics`

- Purpose: To provide various metrics about your application's performance.

- Request Body: N/A (GET request)

- Response Body (JSON):

```json

{

"cpu\_usage": 0.85,

"memory\_usage": 0.45,

"request\_count": 1000,

"error\_count": 5

// Add more custom metrics as needed

}

```

- Headers: N/A

2. `/health`

- Purpose: To check the overall health of your application.

- Request Body: N/A (GET request)

- Response Body (JSON):

```json

{

"status": "UP",

"details": {

"database": "UP",

"external\_service": "UP"

// Add more health checks for different components

}

}

```

- Headers: N/A

3. `/logs`

- Purpose: To retrieve application logs.

- Request Body: N/A (GET request)

- Response Body (JSON or plain text):

- You can provide the log entries as an array of objects or as plain text, depending on your preference.

- JSON example:

```json

[

{

"timestamp": "2023-09-10T14:30:00Z",

"level": "INFO",

"message": "Application started"

},

{

"timestamp": "2023-09-10T15:00:00Z",

"level": "ERROR",

"message": "An error occurred"

}

]

```

- Plain text example:

```

2023-09-10T14:30:00Z INFO Application started

2023-09-10T15:00:00Z ERROR An error occurred

```

- Headers: N/A

4. `/custom-metrics`

- Purpose: To expose custom application-specific metrics.

- Request Body: N/A (GET request)

- Response Body (JSON):

```json

{

"custom\_metric\_1": 123,

"custom\_metric\_2": "some\_value"

// Add more custom metrics as needed

}

```

- Headers: N/A

5. `/config`

- Purpose: To provide configuration details about your application.

- Request Body: N/A (GET request)

- Response Body (JSON):

```json

{

"database\_url": "jdbc:mysql://localhost:3306/mydb",

"cache\_enabled": true,

"log\_level": "INFO"

// Add more configuration parameters as needed

}

```

- Headers: N/A

6. `/metrics/reset`

- Purpose: To reset custom metrics or counters.

- Request Body: N/A (POST request)

- Response Body (JSON):

```json

{

"message": "Metrics reset successfully"

}

```

- Headers: N/A

Certainly! When creating a Bug Tracking and Issue Management API using Spring Boot, you can define various endpoints to perform CRUD (Create, Read, Update, Delete) operations on issues. Additionally, you might need endpoints for comments, status updates, and more. Here are some endpoints you can consider, along with their request bodies, response bodies, and headers in JSON format:

1. \*\*Create Issue\*\*

- Endpoint: `/issues`

- Method: POST

- Request Body (JSON):

```json

{

"title": "Bug Title",

"description": "Bug Description",

"createdBy": "User ID",

"status": "Open"

}

```

- Response Body (JSON):

```json

{

"id": "Issue ID",

"title": "Bug Title",

"description": "Bug Description",

"createdBy": "User ID",

"status": "Open"

}

```

2. \*\*Retrieve Issue\*\*

- Endpoint: `/issues/{id}`

- Method: GET

- Response Body (JSON):

```json

{

"id": "Issue ID",

"title": "Bug Title",

"description": "Bug Description",

"createdBy": "User ID",

"status": "Open",

"comments": [List of Comments],

"updatedAt": "Date and Time"

}

```

3. \*\*Update Issue\*\*

- Endpoint: `/issues/{id}`

- Method: PUT

- Request Body (JSON):

```json

{

"title": "New Bug Title",

"description": "New Bug Description",

"status": "In Progress"

}

```

- Response Body (JSON):

```json

{

"id": "Issue ID",

"title": "New Bug Title",

"description": "New Bug Description",

"createdBy": "User ID",

"status": "In Progress",

"updatedAt": "Date and Time"

}

```

4. \*\*Close Issue\*\*

- Endpoint: `/issues/{id}/close`

- Method: POST (or PUT)

- No Request Body

- Response Body (JSON):

```json

{

"id": "Issue ID",

"title": "Bug Title",

"description": "Bug Description",

"createdBy": "User ID",

"status": "Closed",

"updatedAt": "Date and Time"

}

```

5. \*\*Add Comment to Issue\*\*

- Endpoint: `/issues/{id}/comments`

- Method: POST

- Request Body (JSON):

```json

{

"text": "Comment Text",

"createdBy": "User ID"

}

```

- Response Body (JSON):

```json

{

"id": "Comment ID",

"text": "Comment Text",

"createdBy": "User ID",

"createdAt": "Date and Time"

}

```

6. \*\*Update Comment\*\*

- Endpoint: `/comments/{id}`

- Method: PUT

- Request Body (JSON):

```json

{

"text": "Updated Comment Text"

}

```

- Response Body (JSON):

```json

{

"id": "Comment ID",

"text": "Updated Comment Text",

"createdBy": "User ID",

"createdAt": "Date and Time",

"updatedAt": "Date and Time"

}

```

7. \*\*Retrieve Issue Status\*\*

- Endpoint: `/status`

- Method: GET

- Response Body (JSON):

```json

{

"statuses": ["Open", "In Progress", "Closed"]

}

```

Certainly! When creating a Configuration Management API using Spring Boot, you'll want to consider various endpoints that allow you to effectively manage configuration settings across different environments. Here are some additional endpoints you might consider, along with their expected request bodies, response bodies, and headers in JSON format:

1. \*\*GET /configurations/{id}\*\*

- Description: Retrieve a specific configuration by its ID.

- Request Headers: None required.

- Response Body (JSON):

```json

{

"id": 1,

"name": "example-config",

"value": "config-value",

"environment": "production",

"created\_at": "2023-09-10T12:00:00Z",

"updated\_at": "2023-09-10T12:30:00Z"

}

```

2. \*\*POST /configurations\*\*

- Description: Create a new configuration setting.

- Request Body (JSON):

```json

{

"name": "new-config",

"value": "new-config-value",

"environment": "staging"

}

```

- Response Body (JSON):

```json

{

"id": 2,

"name": "new-config",

"value": "new-config-value",

"environment": "staging",

"created\_at": "2023-09-10T13:00:00Z",

"updated\_at": null

}

```

3. \*\*PUT /configurations/{id}\*\*

- Description: Update an existing configuration setting.

- Request Body (JSON):

```json

{

"value": "updated-config-value"

}

```

- Response Body (JSON):

```json

{

"id": 2,

"name": "new-config",

"value": "updated-config-value",

"environment": "staging",

"created\_at": "2023-09-10T13:00:00Z",

"updated\_at": "2023-09-10T13:30:00Z"

}

```

4. \*\*DELETE /configurations/{id}\*\*

- Description: Delete a configuration setting by its ID.

- Request Headers: None required.

- Response Body (JSON):

```json

{

"message": "Configuration with ID 2 has been deleted successfully."

}

```

5. \*\*GET /environments\*\*

- Description: Retrieve a list of available environments.

- Request Headers: None required.

- Response Body (JSON):

```json

{

"environments": ["development", "staging", "production"]

}

```

6. \*\*GET /settings\*\*

- Description: Retrieve global configuration settings that apply to all environments.

- Request Headers: None required.

- Response Body (JSON):

```json

{

"settings": [

{

"name": "global-setting-1",

"value": "value-1"

},

{

"name": "global-setting-2",

"value": "value-2"

}

]

}

```

To create an API endpoint for changing environment variables, you can use a PATCH or PUT request to update the values of specific environment variables. Here's an example of how you can design this endpoint:

\*\*Endpoint:\*\*

```

PATCH /environments/{environment}/variables/{variableName}

```

- `{environment}` should be the name of the target environment (e.g., "staging" or "production").

- `{variableName}` should be the name of the environment variable you want to update.

\*\*Request Body (JSON):\*\*

```json

{

"value": "new-variable-value"

}

```

- The request body should include the new value that you want to set for the environment variable.

\*\*Request Headers:\*\*

- You may require authentication headers or specific permissions to make changes to environment variables, depending on your security requirements.

\*\*Response Body (JSON):\*\*

If the update is successful, you can respond with a success message or the updated variable information:

```json

{

"message": "Environment variable updated successfully.",

"environment": "staging",

"variableName": "example-variable",

"newValue": "new-variable-value"

}

```

\*\*HTTP Status Codes:\*\*

- Use appropriate HTTP status codes to indicate the success or failure of the operation (e.g., 200 OK for success, 400 Bad Request for validation errors, 401 Unauthorized for authentication issues, 404 Not Found if the environment or variable doesn't exist, etc.).

Make sure to implement proper validation and error handling in your API to handle cases where the environment or variable name is invalid or the update operation fails for any reason.

4. \*\*Update Book Details\*\*

- \*\*Endpoint\*\*: `/api/books/{id}`

- \*\*Method\*\*: PUT

- \*\*RequestBody\*\*: JSON object containing updated book details (e.g., title, author, year).

- \*\*ResponseBody\*\*: JSON object confirming the successful update or an error message.

- \*\*Headers\*\*: None or authentication headers if required.

5. \*\*Delete Book\*\*

- \*\*Endpoint\*\*: `/api/books/{id}`

- \*\*Method\*\*: DELETE

- \*\*RequestBody\*\*: None (path parameter used for specifying the book ID to be deleted).

- \*\*ResponseBody\*\*: JSON object confirming the successful deletion or an error message.

- \*\*Headers\*\*: None or authentication headers if required.

6. \*\*Search Books by Title\*\*

- \*\*Endpoint\*\*: `/api/books/search`

- \*\*Method\*\*: POST

- \*\*RequestBody\*\*: JSON object containing a `title` field with the search keyword.

- \*\*ResponseBody\*\*: JSON array of books matching the search criteria.

- \*\*Headers\*\*: None or authentication headers if required.

7. \*\*Search Books by Author\*\*

- \*\*Endpoint\*\*: `/api/books/search`

- \*\*Method\*\*: POST

- \*\*RequestBody\*\*: JSON object containing an `author` field with the search keyword.

- \*\*ResponseBody\*\*: JSON array of books matching the search criteria.

- \*\*Headers\*\*: None or authentication headers if required.

8. \*\*Filter Books by Year\*\*

- \*\*Endpoint\*\*: `/api/books/filter/year`

- \*\*Method\*\*: POST

- \*\*RequestBody\*\*: JSON object containing a `year` field with the filter value.

- \*\*ResponseBody\*\*: JSON array of books filtered by the specified year.

- \*\*Headers\*\*: None or authentication headers if required.

Here's a summary of the new endpoints and their JSON structures:

\*\*Update Book Details\*\*

- Endpoint: `/api/books/{id}`

- Method: PUT

- RequestBody: JSON object containing updated book details (e.g., title, author, year).

```json

{

"title": "Updated Title",

"author": "Updated Author",

"year": 2022

}

```

- ResponseBody: JSON object confirming the successful update or an error message.

```json

{

"message": "Book updated successfully"

}

```

\*\*Delete Book\*\*

- Endpoint: `/api/books/{id}`

- Method: DELETE

- RequestBody: None (path parameter used for specifying the book ID to be deleted).

- ResponseBody: JSON object confirming the successful deletion or an error message.

```json

{

"message": "Book deleted successfully"

}

```

\*\*Search Books by Title\*\*

- Endpoint: `/api/books/search`

- Method: POST

- RequestBody: JSON object containing a `title` field with the search keyword.

```json

{

"title": "Great"

}

```

- ResponseBody: JSON array of books matching the search criteria.

```json

[

{

"id": 1,

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"year": 1925

}

]

```

\*\*Search Books by Author\*\*

- Endpoint: `/api/books/search`

- Method: POST

- RequestBody: JSON object containing an `author` field with the search keyword.

```json

{

"author": "Harper Lee"

}

```

- ResponseBody: JSON array of books matching the search criteria.

```json

[

{

"id": 2,

"title": "To Kill a Mockingbird",

"author": "Harper Lee",

"year": 1960

}

]

```

\*\*Filter Books by Year\*\*

- Endpoint: `/api/books/filter/year`

- Method: POST

- RequestBody: JSON object containing a `year` field with the filter value.

```json

{

"year": 1960

}

```

- ResponseBody: JSON array of books filtered by the specified year.

```json

[

{

"id": 2,

"title": "To Kill a Mockingbird",

"author": "Harper Lee",

"year": 1960

}

]

```

You can add these endpoints and their corresponding functionality to your Spring Boot application. Make sure to implement proper error handling, validation, and any necessary database interactions based on your project's requirements.

Certainly, here are the Request and Response examples for the additional endpoints I mentioned earlier in JSON format:

\*\*1. Update Book Details (PUT Request and Response)\*\*

- \*\*PUT Request:\*\*

Endpoint: `/api/books/1` (Updating book with ID 1)

Request JSON:

```json

{

"title": "Updated Title",

"author": "Updated Author",

"year": 2022

}

```

- \*\*PUT Response:\*\*

Response JSON (Success):

```json

{

"message": "Book updated successfully"

}

```

Response JSON (Error, if book with ID 1 is not found):

```json

{

"error": "Book not found with ID 1"

}

```

\*\*2. Delete Book (DELETE Request and Response)\*\*

- \*\*DELETE Request:\*\*

Endpoint: `/api/books/2` (Deleting book with ID 2)

- \*\*DELETE Response:\*\*

Response JSON (Success):

```json

{

"message": "Book deleted successfully"

}

```

Response JSON (Error, if book with ID 2 is not found):

```json

{

"error": "Book not found with ID 2"

}

```

\*\*3. Search Books by Title (POST Request and Response)\*\*

- \*\*POST Request:\*\*

Endpoint: `/api/books/search`

Request JSON:

```json

{

"title": "Great"

}

```

- \*\*POST Response:\*\*

Response JSON:

```json

[

{

"id": 1,

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"year": 1925

}

]

```

\*\*4. Search Books by Author (POST Request and Response)\*\*

- \*\*POST Request:\*\*

Endpoint: `/api/books/search`

Request JSON:

```json

{

"author": "Harper Lee"

}

```

- \*\*POST Response:\*\*

Response JSON:

```json

[

{

"id": 2,

"title": "To Kill a Mockingbird",

"author": "Harper Lee",

"year": 1960

}

]

```

\*\*5. Filter Books by Year (POST Request and Response)\*\*

- \*\*POST Request:\*\*

Endpoint: `/api/books/filter/year`

Request JSON:

```json

{

"year": 1960

}

```

- \*\*POST Response:\*\*

Response JSON:

```json

[

{

"id": 2,

"title": "To Kill a Mockingbird",

"author": "Harper Lee",

"year": 1960

}

]

```

These examples demonstrate the JSON format for both the request and response for each of the mentioned endpoints. The actual format and structure may vary based on your application's implementation and specific use cases.