



Assessment Diagnóstico

Nombre del alumno: Arroyo Gómez José Alfredo

Fecha: 15/septiembre/2023

Assessment Diagnostico

$$\{ M^p \}$$

BackEnd

Java

Desarrollar un api restFull que permita realizar operaciones CRUD que cumplan con los siguientes puntos:

- ✓ Manejo de excepciones
- ✓ Endpoints
- ✓ Pruebas
- ✓ Seguridad
- ✓ Git
- ✓ Base de datos

Considera que se deben tomar en cuenta las siguientes actividades:

1. Crear un microservicio de gestión de tareas (Java, creación de RestFull):

- El microservicio debe tener una API RESTful que permita realizar operaciones CRUD (*Create, Read, Update, Delete*) básicas para gestionar tareas.
- Cada tarea debe tener al menos los siguientes atributos: ID, descripción y estado (*pendiente, en progreso, completada*).
- Utiliza una base de datos relacional (*por ejemplo: MySQL, PostgreSQL*) para almacenar las tareas.

2. Manejo de excepciones:

- Implementa un manejo adecuado de excepciones en el microservicio para capturar y manejar los errores que puedan ocurrir durante las operaciones CRUD.
- Utiliza las funcionalidades proporcionadas por Spring Boot para gestionar las excepciones y devolver respuestas HTTP apropiadas con información clara sobre los errores ocurridos.

3. Endpoints: Diseña y define los endpoints de la API RESTful para cada una de las operaciones CRUD del microservicio de tareas

4. Pruebas:

- Escribe pruebas unitarias para las clases y métodos más importantes del microservicio utilizando el framework de pruebas JUnit.
- Utiliza el framework Mockito para simular dependencias externas y facilitar la realización de pruebas unitarias.
- Utiliza las anotaciones proporcionadas por Spring MVC para mapear los endpoints a los controladores correspondientes.

5. Seguridad:

- Agrega seguridad a la API RESTful utilizando Spring Security.
- Implementa autenticación basada en tokens JWT (*JSON Web Tokens*) para proteger los endpoints del microservicio.

- Los usuarios deben poder autenticarse y recibir un token JWT válido para acceder a los endpoints protegidos.

6. Git:

- Utiliza Git como sistema de control de versiones para gestionar el desarrollo del proyecto.
- Crea un repositorio de Git y realiza commits periódicos para documentar los cambios y el progreso del desarrollo.

7. Base de datos:

- Configura y utiliza una base de datos relacional (*por ejemplo: MySQL, PostgreSQL*) para almacenar las tareas.
- Asegúrate de que la configuración de la base de datos sea correcta y que las operaciones CRUD se realicen de manera adecuada.

Recuerda seguir las mejores prácticas de desarrollo de microservicios, como la separación de responsabilidades, la modularidad y la escalabilidad. Utiliza cualquier IDE o herramienta de desarrollo de tu elección y selecciona la base de datos relacional que prefieras.

Enlaces:

[Repositorio en Github](#)

[Carpeta en Google Drive](#)



JOSÉ ALFREDO, ARROYO GÓMEZ <20030029@itcelaya.edu.mx>

Arroyo Gómez José Alfredo - Booster Career

JOSÉ ALFREDO, ARROYO GÓMEZ <20030029@itcelaya.edu.mx>

5 de septiembre de 2023, 22:48

Para: adredsi26@gmail.com

Este es mi Assesment Diagnóstico del booster career.

[Repositorio en Github](#)

[Carpeta en Google Drive](#)

Capturas BD:

The screenshot shows a database management interface. On the left, the 'Navigator' pane displays the 'tasks' schema structure, including tables, views, stored procedures, and functions. The 'tasks' table is expanded, showing its columns, indexes, foreign keys, and triggers. The 'Administration' tab is selected, and the 'Schemas' section is active. The 'Schema: tasks' is highlighted in green.

On the right, the 'task' window shows a SQL query: `SELECT * FROM tasks.task;`. Below the query, the 'Result Grid' displays the following data:

	id_task	description	id_state
▶	1	Test Update	2
	3	Create the table task	3
	4	Task GET METHOD	3
	7	Task POST METHOD	3
	8	Task PUT METHOD	3
*	NULL	NULL	NULL

The screenshot shows a database management interface. On the left, the 'Navigator' pane displays a tree view of the database schema. The 'tasks' database is expanded, showing 'Tables' with 'state' and 'task'. The 'state' table is selected, and its structure is shown in the 'Information' pane below. The 'state' table has two columns: 'id_state' (int AI PK) and 'state' (varchar(40)).

On the right, the 'task' tab is active, showing a SQL query: `SELECT * FROM tasks.state;`. Below the query, the 'Result Grid' displays the data from the 'state' table:

	id_state	state
▶	1	PENDING
	2	IN PROGRESS
	3	COMPLETED
*	NULL	NULL

SQL:

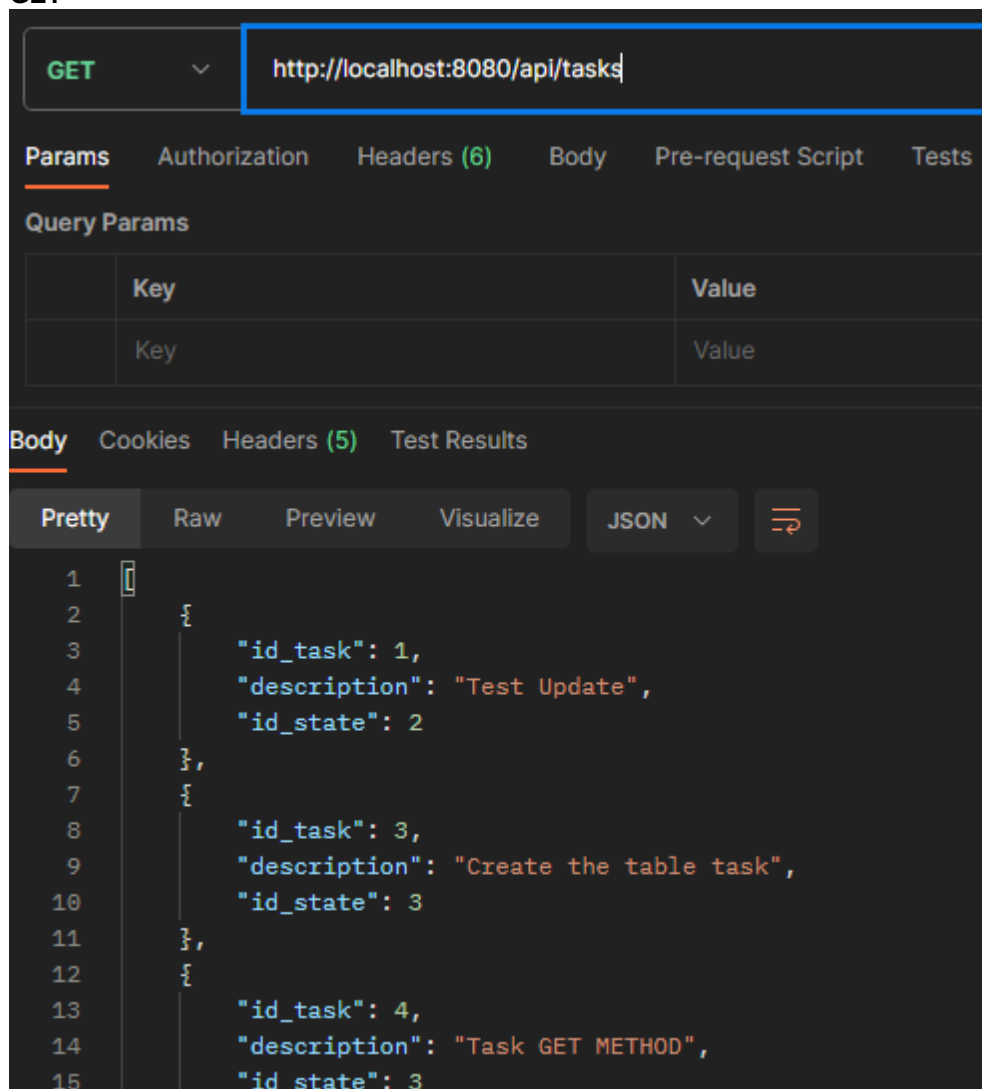
USE tasks;

CREATE TABLE state(id_state int primary key auto_increment, state varchar(40) not null);

CREATE TABLE task (id_task int primary key auto_increment, description varchar(255) not null, id_state int not null references state(id_state));

OPERACIONES CRUD:

GET



GET

Params Authorization Headers (6) Body Pre-request Script Tests

Query Params

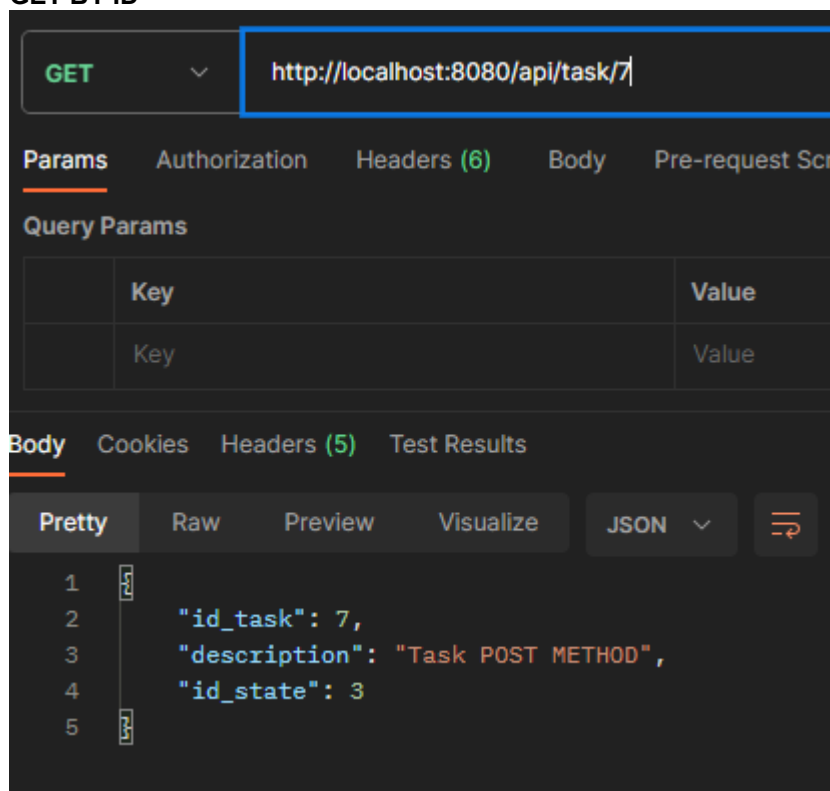
Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id_task": 1,
4     "description": "Test Update",
5     "id_state": 2
6   },
7   {
8     "id_task": 3,
9     "description": "Create the table task",
10    "id_state": 3
11  },
12  {
13    "id_task": 4,
14    "description": "Task GET METHOD",
15    "id_state": 3
16  }
17 ]
```

GET BY ID



GET

Params Authorization Headers (6) Body Pre-request Script Tests

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id_task": 7,
3   "description": "Task POST METHOD",
4   "id_state": 3
5 }
```


POST

POST

http://localhost:8080/api/task

Params

Authorization

Headers (8)

Body

Pre-request Script

T

none

form-data

x-www-form-urlencoded

raw

binary

1

2

3

4

....."description": "To Study",

....."id_state": 1

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

"id_task": 9,

"description": "To Study",

"id_state": 1

PUT

PUT ⌵ <http://localhost:8080/api/task/9>

Params Authorization Headers (8) **Body** ● Pre-request S

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐

```
1 {
2   .... "id_task": 9,
3   .... "description": "To Eat",
4   .... "id_state": 2
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ ↺

```
1 {
2   "id_task": 9,
3   "description": "To Eat",
4   "id_state": 2
5 }
```

DELETE

GET ⌵ <http://localhost:8080/api/task/9>

Params Authorization Headers (8) **Body** ● Pre-request

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐

```
1 {
2   .... "id_task": 9,
3   .... "description": "To Eat",
4   .... "id_state": 2
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ ↺

```
1 {
2   "id_task": 9,
3   "description": "To Eat",
4   "id_state": 2
5 }
```

DELETE ▼ `http://localhost:8080/api/task/9`

Params Authorization Headers (8) **Body** ● Pre-request Script Test

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ C

```
1 [
2   .... "id_task": 9,
3   .... "description": "To Eat",
4   .... "id_state": 2
5 ]
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text ▼

1

GET ▼ `http://localhost:8080/api/task/9`

Params Authorization Headers (8) **Body** ● Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```
1 [
2   .... "id_task": 9,
3   .... "description": "To Eat",
4   .... "id_state": 2
5 ]
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **JSON** ▼

1 null