**Documentation:**

**API Endpoints:**

1. GET /tasks:

- Description: Retrieve a list of all tasks.

- Method: GET

- Response:

- Status 200: OK

```json

[

{ "id": 1, "title": "Task 1", "description": "Description 1" },

{ "id": 2, "title": "Task 2", "description": "Description 2" }

]

```

2. GET /tasks/:id:

- Description: Retrieve a specific task by ID.

- Method: GET

- Parameters:

- `id` (integer): Task ID

- Response:

- Status 200: OK

```json

{ "id": 1, "title": "Task 1", "description": "Description 1" }

```

- Status 404: Not Found

```json

{ "error": "Task not found" }

```

3. POST /tasks:

- Description: Create a new task.

- Method: POST

- Request:

```json

{ "title": "New Task", "description": "New Description" }

```

- Response:

- Status 201: Created

```json

{ "id": 3, "title": "New Task", "description": "New Description" }

```

- Status 400: Bad Request

```json

{ "error": "Title and description are required" }

```

4. PUT /tasks/:id:

- Description: Update an existing task by ID.

- Method: PUT

- Parameters:

- `id` (integer): Task ID

- Request:

```json

{ "title": "Updated Task", "description": "Updated Description" }

```

- Response:

- Status 200: OK

```json

{ "id": 1, "title": "Updated Task", "description": "Updated Description" }

```

- Status 404: Not Found

```json

{ "error": "Task not found" }

```

- Status 400: Bad Request

```json

{ "error": "Title and description are required" }

```

5. DELETE /tasks/:id:

- Description: Delete a task by ID.

- Method:\* DELETE

- Parameters:

- `id` (integer): Task ID

- Response:

- Status 200: OK

```json

{ "message": "Task deleted successfully" }

```

- Status 404: Not Found

```json

{ "error": "Task not found" }

```

**Error Handling:**

- The API returns appropriate HTTP status codes along with error messages for invalid requests or not found resources.

**Middleware:**

- The API uses the `body-parser` middleware for JSON request parsing.

- A custom middleware function `validateTask` is implemented to ensure that task properties are required during task creation and update.

Report:

**Approach and Algorithm Choices:**

1. Express.js Framework:

- Chose Express.js for building the API due to its simplicity, flexibility, and widespread use in the Node.js ecosystem.

2. In-Memory Storage:

- Used a simple array (`tasks`) to store tasks in memory rather than incorporating a database for the sake of simplicity.

3.Middleware:

- Implemented middleware for JSON request parsing (`body-parser`) and a custom middleware (`validateTask`) for basic validation.

4.Error Handling:

- Implemented error handling to gracefully handle unexpected issues and provide meaningful error messages.

5. RESTful Principles:

- Followed RESTful principles by structuring endpoints for CRUD operations and using appropriate HTTP methods.

6. Documentation:

- Provided clear and concise documentation for each endpoint, including sample requests and responses.

7. Bonus Points (Optional):

- Pagination, sorting, and filtering were not implemented for simplicity. Authentication and authorization mechanisms were also not added for brevity.

**Conclusion:**

The chosen approach prioritizes simplicity and clarity for educational purposes. It provides a foundation for a basic RESTful API with essential CRUD operations, basic validation, and error handling. Further enhancements, such as database integration, authentication, and additional features, can be added based on specific project requirements.