What went well:

1. Designing a RESTful API enables the ability to create an intuitional API. Adhering to the REST principles while using Proper HTTP methods (GET, POST, PUT, DELETE) and Response Codes ensured that the API operated as required. Clean, Easy to use Resource URL Structure (/tasks, /tasks/:id)

2. The process of manual testing using Postman was very thorough and organised. I performed validation for all CRUD operations, as well as for some error scenarios (404 and 400) and success scenarios. In addition, since I had a record of my test cases, it was easy to do re runs once changes were applied.

What could have been done better:

1. Standardizing APIs to Make it Easier for Clients to Consume
   Current: Different formats for responses for each endpoint
   Improved: Should have created a standard wrapper for all API responses.
   Importance: Will help to establish a more professional-looking API and also will make it easier for customers to read the responses of the API in a uniform way.

2. Pagination of API Response Data.
   Current: A request for GET /tasks will return everything for the task endpoint, this could contain thousands of individual task records.

   Improved: Should have implemented Pagination as follows:

   Accepts page and limit query string parameters
   Returns the total number of records and current page.
   Limits default records to 10-20

   GET /tasks?page=1&limit=10
   Importance: Using Pagination will significantly increase the performance of large sets of data and will lower the amount of bandwidth used to deliver that data.

   Screenshots of your database schema:

Table: inventory

Columns:

inventory_id	int AI PK
product_id	int
supplier_id	int
quantity	int
last_updated	timestamp

Table: product

Columns:

product_id	int AI PK
product_name	varchar(100)
description	text
price	decimal(10,2)
created_at	timestamp

Table: supplier

Columns:

supplier_id	int AI PK
supplier_name	varchar(100)
contact_info	varchar(255)
created_at	timestamp