

API CREATION

by Sohaan Sureshkumar

Submission date: 08-Dec-2025 02:15PM (UTC+0000)

Submission ID: 270030327

File name: 51186_Sohaen_Sureshkumar_API_CREATION_708807_1797727872.docx (1.91M)

Word count: 3207

Character count: 18799



ASSIGNMENT 1: API CREATION

Submitted By:

2221048

Sohaam Sureshkumar

Module: Cloud Based Web Services

Lecturer: Mark Huntly



Table of Contents

INTRODUCTION	2
1.1 What are Rest APIs	3
1.2 Alternative Protocols for Accessing Information	4
1.2.1 SOAP (Simple Object Access Protocol):	4
1.2.2 GraphQL	4
1.3 Advantages & Disadvantages Compared to Alternatives	5
1.3.1 REST vs SOAP	5
1.3.2 REST vs GraphQL	5
1.3.3 USE CASES	6
WEB API OVERVIEW	7
2.1 BASIC OPERATIONS	7
2.1.1 GET	7
2.1.2 POST	8
2.1.3 DELETE	10
2.2 Implementing a PATCH Operation	11
2.3 TWO STAGE WORKFLOW	14
2.4 FUNCTIONAL TESTS FOR API	16
CONCLUSION	19
REFERENCES	20

INTRODUCTION

The assignment consists of two essential sections which include research about web API styles and development of a RESTful service. The research investigates REST as a web API architectural style through a comparison with SOAP and GraphQL as alternative solutions. The analysis evaluates request-response structures, client-server binding strength, performance, complexity levels and application suitability for each style.

The practical section begins with creating a Task Management API through Node.js and Express which runs inside an Ubuntu virtual machine. The API system follows an e-commerce admin workflow which requires task tracking for product listing addition and order processing. The API enables users to create tasks and retrieve them and modify and delete them through endpoints which support partial updates via PATCH and basic workflow management that changes task status from pending to completed. The API endpoints undergo Postman manual testing followed by documentation through structured test tables.

REST APIs AND ALTERNATIVE PROTOCOLS

1.1 What are Rest APIs

A REST API (Representational State Transfer API) functions as a web service protocol which enables systems to exchange HTTP messages through established rules instead of using a specific network protocol. Roy Fielding established REST as an architectural framework for distributed hypermedia systems through his doctoral work although he did not create a new networking protocol [1].

A RESTful design structure represents data through resources which serve as the foundation of its architecture. Resources in RESTful design receive their identification through URLs that include /tasks and /tasks/1 and HTTP methods perform actions on these resources. The standard HTTP methods enable GET to retrieve data and POST to create new resources and PUT and PATCH to modify existing resources and DELETE to remove them [2]. The server provides resource representations through JSON data which serves as a lightweight format for web clients to process.

The design of REST depends on four essential restrictions. The client-server separation exists as a fundamental principle because web and mobile front-ends handle user interfaces but servers maintain responsibility for data storage and processing. The server operates without state maintenance because each request must contain complete information needed for processing. The stateless nature of REST services enables better scalability because any server instance can process requests independently [1][3]. The system enables response caching which allows intermediaries to save results for preventing duplicate processing.

Modern web and cloud APIs adopt REST as their standard choice because it uses established web standards and provides straightforward understanding and seamless integration with browsers and Postman and front-end JavaScript frameworks [3][4]. The Node.js and Express-based task management service implements REST principles through its /tasks endpoint which enables Create Read Update and Delete operations using HTTP verbs.

1.2 Alternative Protocols for Accessing Information

REST stands as a widely used web service design method but developers can use SOAP and GraphQL as alternative solutions for data exchange data. The two alternatives to REST use different methods for data exchange through their approaches.

1.2.1 SOAP (Simple Object Access Protocol):

The W3C standardized SOAP (Simple Object Access Protocol) as an XML-based messaging protocol [5]. A SOAP message contains an envelope that contains both a header section and a body section. A SOAP service provides its operations through WSDL (Web Services Description Language) files which define message formats and operations and bindings in a strongly typed machine-readable format [6]. The SOAP protocol supports multiple transport protocols including HTTP and SMTP and users can enable WS-* extensions for security and transaction management and reliable message delivery. The strict contract requirements and formal governance and complex integration needs of financial services and government platforms make SOAP their preferred choice.

1.2.2 GraphQL

The query-based system of GraphQL serves as a modern alternative to traditional data access methods. The open-source specification for GraphQL maintains its development at Facebook since its initial creation [7]. A GraphQL API provides one endpoint (e.g. /graphql) which clients use to send queries that determine their required data fields and relationships. The server executes the query against its strongly typed schema to deliver only the requested data in JSON format [8]. The system supports three main operations which include data retrieval through queries and data modification through mutations and real-time update reception through subscriptions [9].

The API designer must establish fixed endpoints for REST but GraphQL allows clients to request specific data fields which match their screen requirements. The current design of single-page applications and mobile applications benefits from GraphQL because it enables them to retrieve all necessary data through a single request instead of performing multiple REST API calls. The implementation of GraphQL servers requires additional complexity while HTTP-level caching becomes more difficult because all requests must go through the same URL [7][10].

1.3 Advantages & Disadvantages Compared to Alternatives

1.3.1 REST vs SOAP

The REST framework provides developers with a more lightweight and user-friendly experience than SOAP does. The use of HTTP methods and JSON data instead of XML in REST results in smaller messages that require less bandwidth for transmission which leads to better performance on mobile networks [3][4]. Developers can access REST APIs through common tools including curl and browsers and Postman without needing to create client stubs from WSDL files. The basic nature of REST provides substantial benefits to developers who need to build small to medium-sized CRUD applications.

SOAP provides its best value through its ability to create formal contracts and its enterprise-level functionality. The combination of WSDL and XML schemas creates exact message definitions which supports both strong data typing and automatic code production [6]. The security features of SOAP together with reliable messaging and transaction support make it suitable for banking and healthcare industries which require absolute message delivery and integrity assurance [5][11]. The main drawbacks of SOAP compared to REST include its wordy nature and its requirement for XML processing and its complex learning process. The additional complexity of SOAP makes it unsuitable for basic web and mobile development because it slows down the development process [3][15].

1.3.2 REST vs GraphQL

The main benefit of GraphQL compared to REST lies in its ability to let users retrieve specific data points from multiple connected resources through a single request. The system enables clients to obtain required data from multiple resources through a single request which results in fewer network requests and enhanced performance for complex front-end applications [7][9][22]. The flexibility of GraphQL makes it suitable for large consumer applications which need to support various UI views including social networks and dashboards.

The server-side implementation of GraphQL requires developers to handle additional complexity. Developers need to create and maintain schemas and write resolvers and implement query cost management and depth restrictions to stop expensive requests [8][10]. The process of HTTP-level caching becomes more complicated because

GraphQL responses do not maintain the same URL relationships as REST resources. The built-in HTTP headers of REST enable simple caching integration with CDNs and other infrastructure systems [3][26]. The advanced capabilities of GraphQL do not provide sufficient value for small APIs that perform basic CRUD operations.

1.3.3 USE CASES

The three approaches deliver solutions which meet different implementation requirements.

The REST approach produces its best results when developers use it to build public web APIs, microservices and educational projects that need fundamental design and tool support instead of strict contract enforcement [3].

SOAP is required for enterprise integration because it needs strict schema definitions and advanced security features including WS-Security and transactional support for payment processing and legacy system connections [5][11].

GraphQL is required for complex data-intensive clients which include large front-end applications that need to minimize network traffic and enable client-defined data structures [7][8].

WEB API OVERVIEW

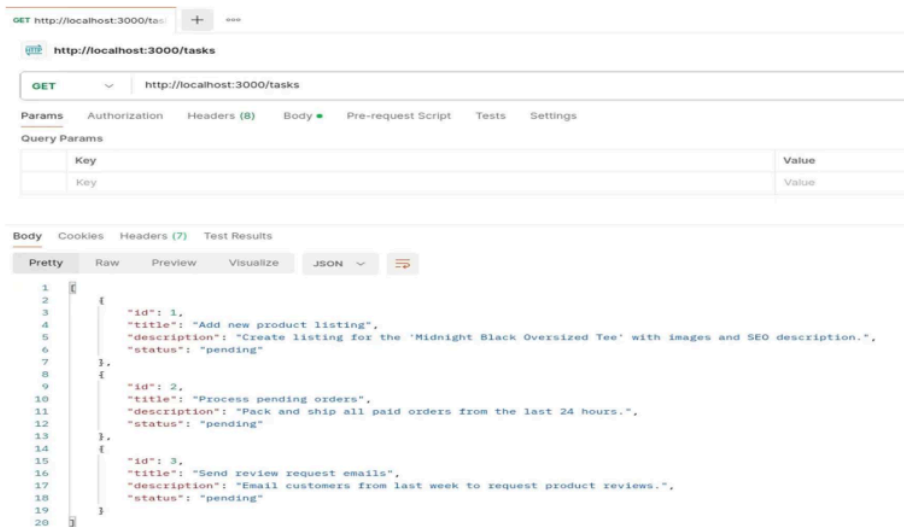
I created a Task Management REST API which supports basic e-commerce administrative operations. The system allows administrators to monitor their operational work through 2 main functions which include product listing addition and order processing. The project focuses on backend API development through Node.js with Express framework for Ubuntu virtual machine deployment.

The API operates within my Ubuntu virtual machine through Node.js implementation with Express framework. The application stores tasks in an array which maintains tasks as objects with id, title, description and status properties. The service enables multiple endpoints which use JSON for communication and Postman users can access them. The following sections describe how the API fulfills all requirements by explaining GET and POST and DELETE operations (Requirement I) and showing PATCH operation for editing (Requirement II) and using status field for two-stage workflow modelling (Requirement III) and showing Postman test results for software verification (Requirement IV).

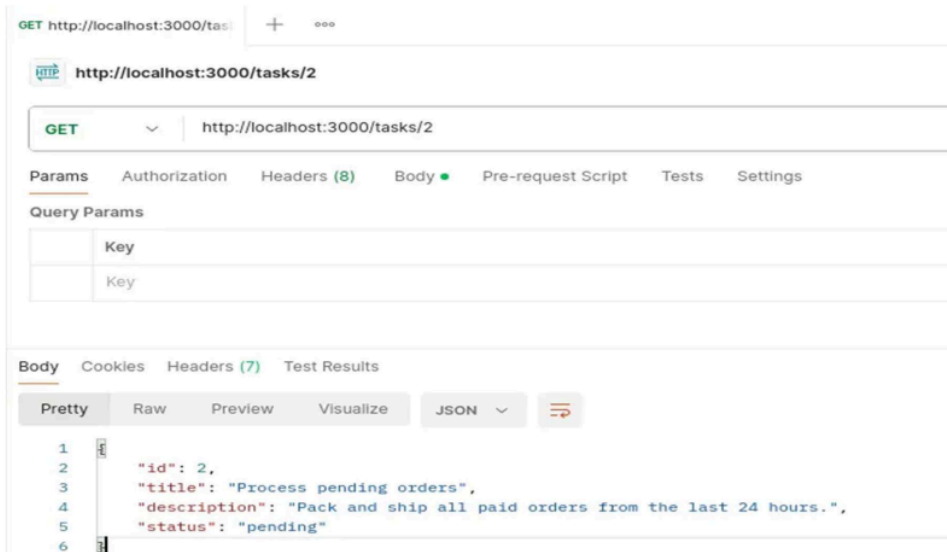
2.1 BASIC OPERATIONS

2.1.1 GET

The GET operations enable users to obtain data from the API. The first screenshot demonstrates a GET request to `http://localhost:3000/tasks` which lacks both parameters and body content. The API returns a JSON array that contains the three default tasks which exist in the in-memory list.



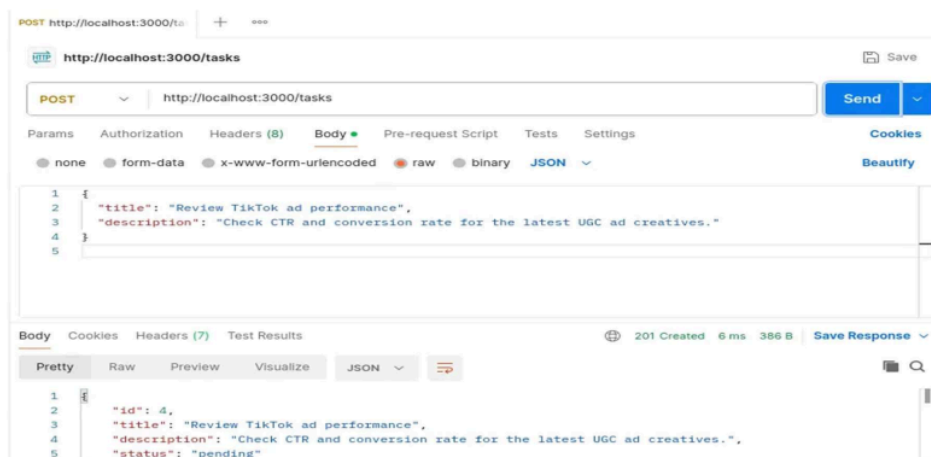
The API returns tasks with id and title and description and status set to "pending" for each task. The endpoint GET /tasks delivers all tasks to the e-commerce admin which includes "Add new product listing" and "Send review request emails".



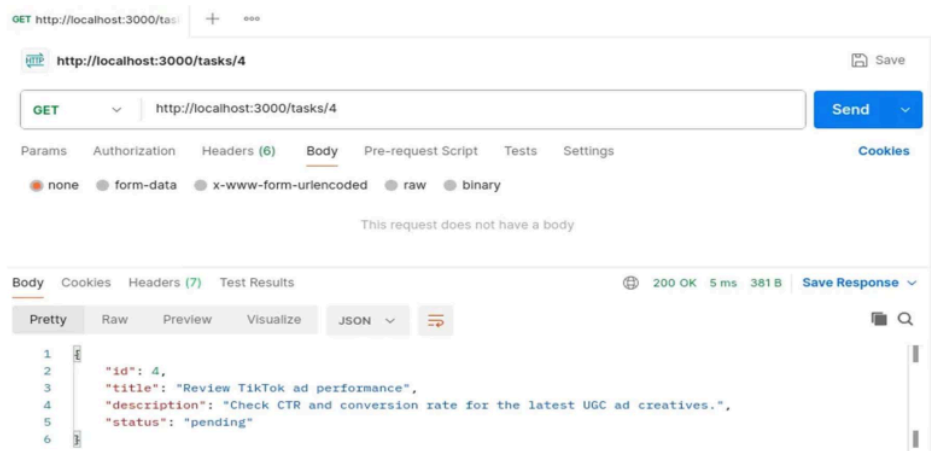
The second screenshot demonstrates how to access task number 2 by using GET http://localhost:3000/tasks/2. The API fetches the task with id: 2 through its URL path which proves the endpoint functions correctly for retrieving single records by their identifiers.

2.1.2 POST

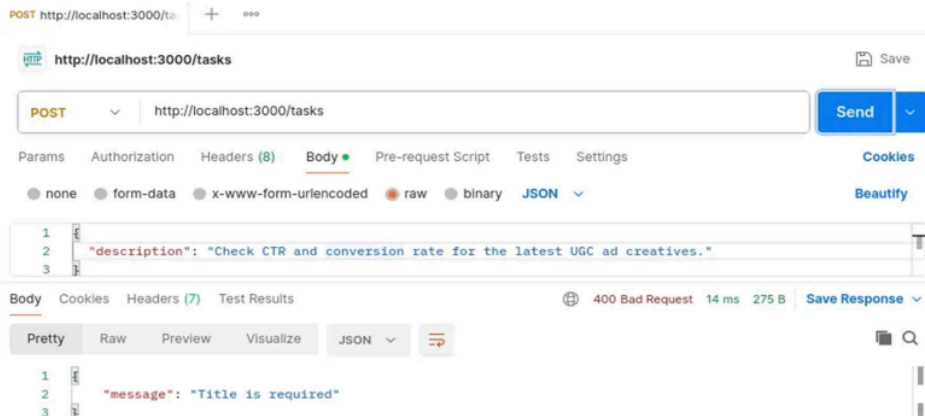
The POST operation enables users to establish new tasks. The following screenshot demonstrates a POST request to `http://localhost:3000/tasks` which includes a JSON body containing the new task details "Review TikTok ad performance". The API responds with status 201 Created and delivers the newly created task which includes `id: 4` and `status: "pending"`.



The following screenshot demonstrates GET `http://localhost:3000/tasks/4` which retrieves the exact task that proves its addition to the internal array.

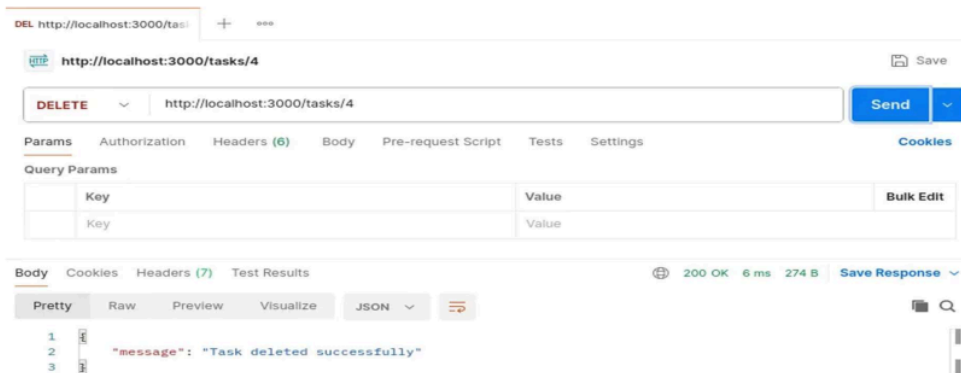


The API system rejects all requests that do not include title information because it returns a 400 Bad Request status with a JSON error message that states "Title is required". The API system refuses to process requests with missing information while delivering specific error feedback to clients.

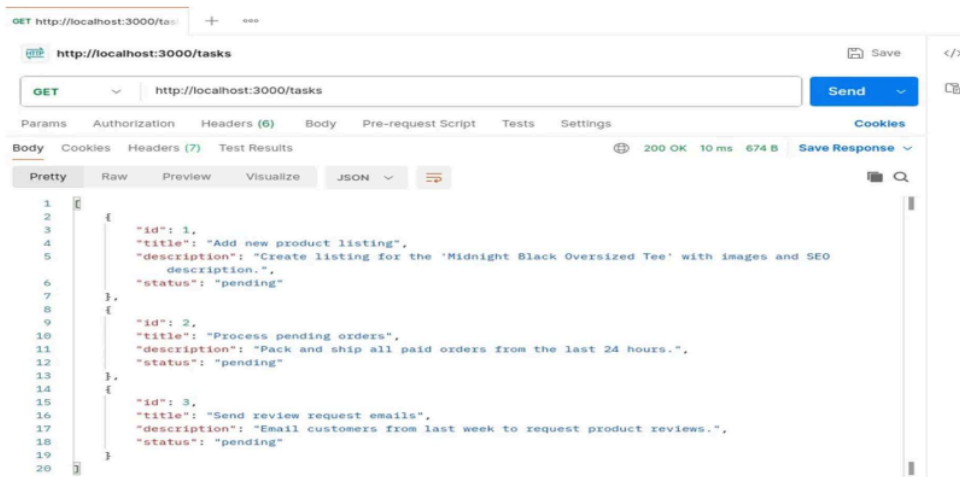


2.1.3 DELETE

The DELETE operation enables users to remove tasks from the system. The delete screenshot shows the process of deleting task number 4 through a DELETE request sent to `http://localhost:3000/tasks/4`. The server system finds task number 4 in the array before it removes the task and sends back a JSON response which states "Task deleted successfully" with a 200 OK status code.



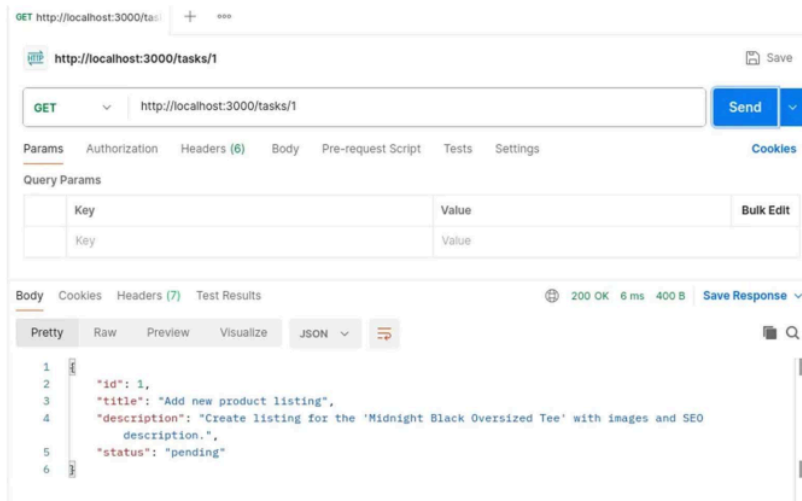
The final screenshot displays a GET `/tasks` request which returns only the three initial tasks with IDs 1, 2 and 3. The in-memory dataset shows no evidence of Task 4 because the delete operation successfully eliminated it.



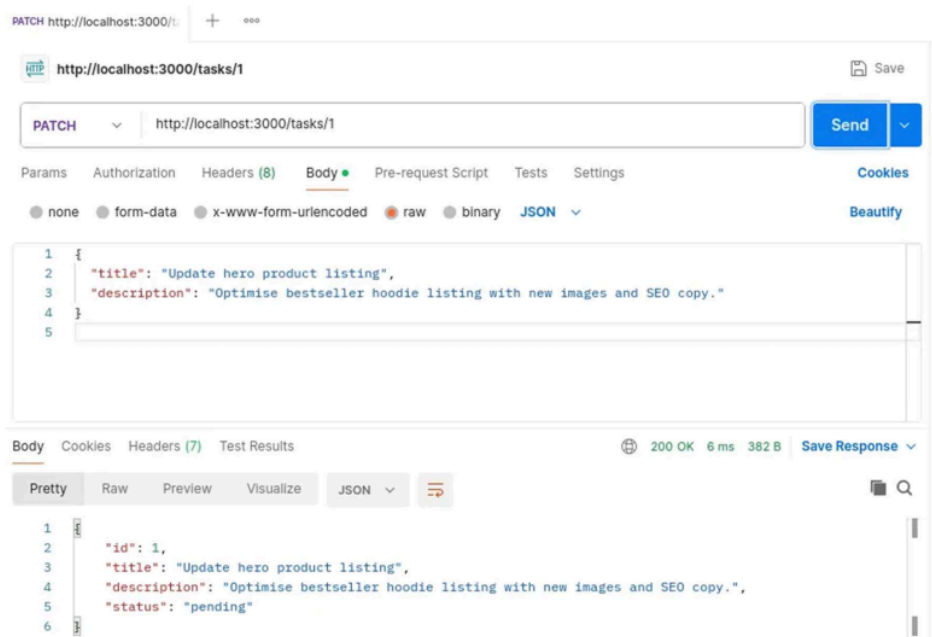
The API successfully performs all necessary REST operations including GET, POST and DELETE according to the assignment requirements.

2.2 Implementing a PATCH Operation

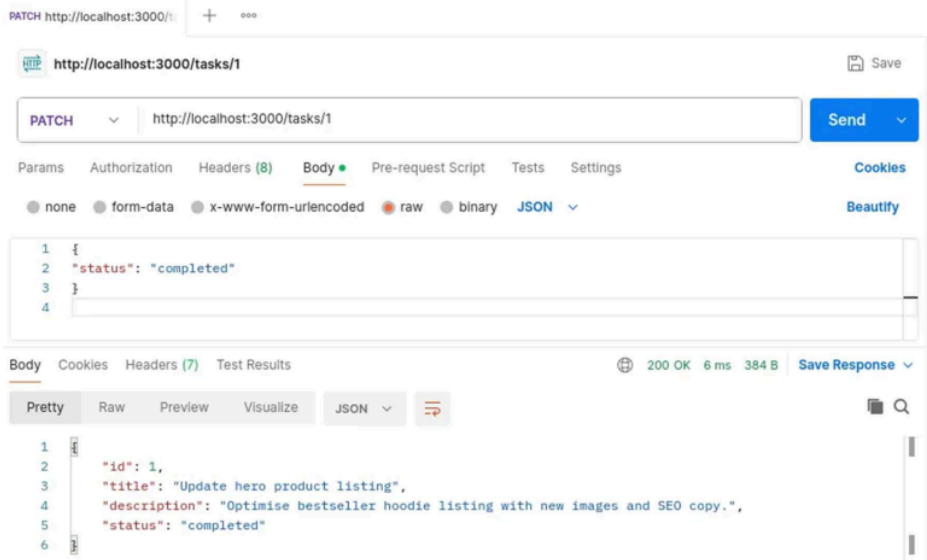
The implementation of editing required me to create a PATCH /tasks/:id endpoint which I tested on task 1. The first screenshot displays the original task details which can be accessed through GET http://localhost:3000/tasks/1. The API returns a JSON object containing id: 1 and the title "Add new product listing" and the original description about creating a listing for the "Midnight Black Oversized Tee" and status: "pending". The API confirms that the task exists in the in-memory array because it returns the task details before any modifications take place.



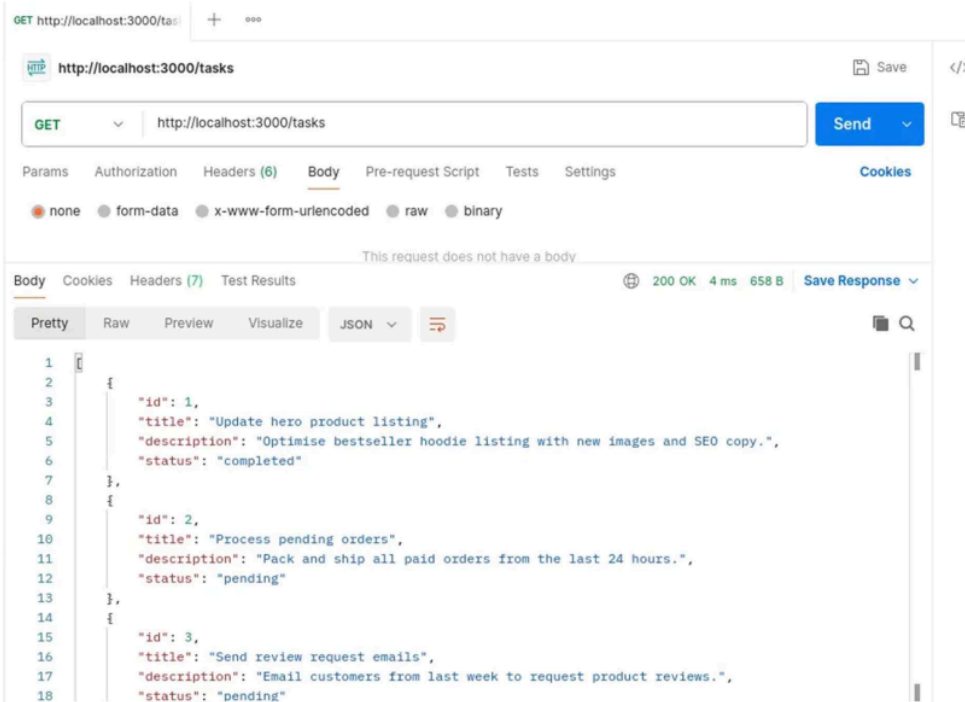
The first PATCH request appears in the second screenshot which sends a PATCH request to `http://localhost:3000/tasks/1` with a JSON body containing updated title and description information. The request body modifies the task to display "Update hero product listing" while it updates the description to improve the bestseller hoodie listing through new images and search engine optimization content. The server returns 200 OK along with the updated task information. The request body fields determine which fields will receive updates because the status field remains unchanged at "pending". The endpoint functions as a partial update which matches the expected behavior for PATCH requests.



The third screenshot demonstrates a second PATCH request to the same resource which sends only the status field with value "completed" in the request body. The API returns 200 OK with a JSON response which shows the title and description fields stay the same but the status field shows "completed" instead of "pending".

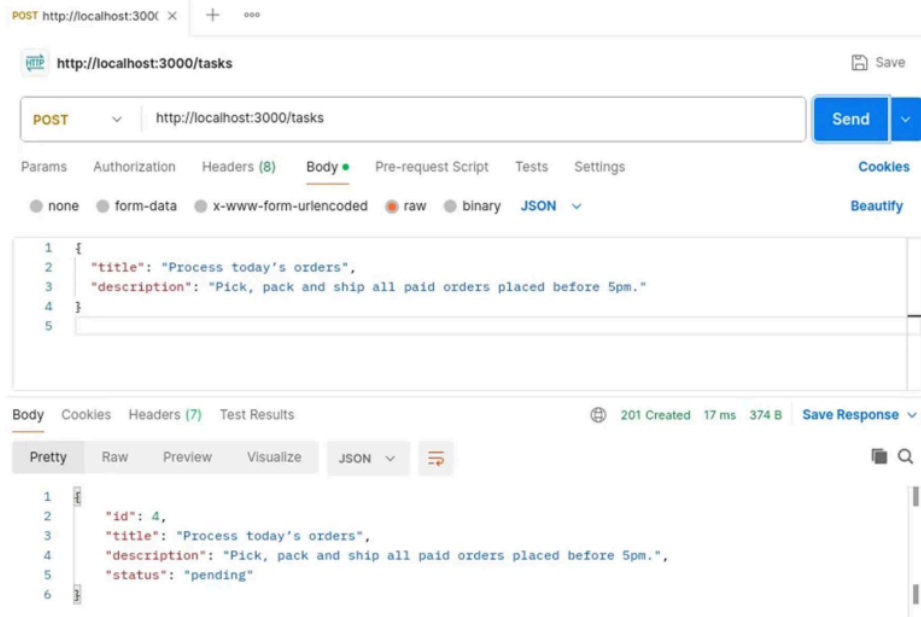


The final screenshot displays GET <http://localhost:3000/tasks> which retrieves the complete list of tasks. Task 1 displays its new title and description and completed status in the task list while all other tasks maintain their original state. The four API requests prove that the PATCH endpoint enables task editing and enables task status transitions from pending to completed without requiring task recreation.

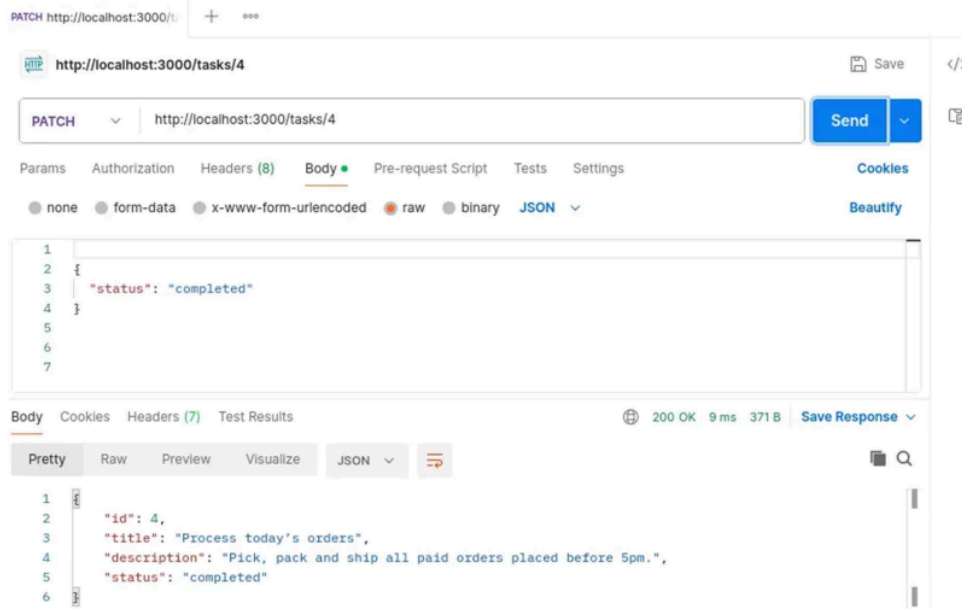


2.3 TWO STAGE WORKFLOW

The status field on each task serves as the solution to fulfill workflow requirements. The API requires tasks to begin in pending status before administrators can perform PATCH operations to finish tasks. The two screenshots demonstrate the two-step process for fulfilling "Process today's orders" tasks.



The admin begins task creation through a POST request which he sends to `http://localhost:3000/tasks`. The JSON body includes a title and description which explains that all paid orders from before 5pm need to be picked and then packed and shipped. The server checks the input data before creating a new task object with id: 4 and status: "pending" and returning it with 201 Created status. The system now shows the task but the actual work remains unfinished. The system now shows Stage 1 of the workflow which includes task creation and scheduling.



The admin proceeds to Stage 2 by performing a status update on the task after finishing order processing. The PATCH request to `http://localhost:3000/tasks/4` with `{ "status": "completed" }` JSON body demonstrates the process. The API system identifies task id: 4 while maintaining both title and description information before it changes the status from "pending" to "completed". The API returns a 200 OK status code together with the complete JSON object which shows the update.

The API implements a straightforward two-step workflow through these two API requests. The API uses POST requests to create new work items which remain in pending status until admins use PATCH requests to complete the work without modifying any other task fields.

2.4 FUNCTIONAL TESTS FOR API

Test Title	View All Tasks
Test Author	Sohaam
Method & Endpoint	GET /tasks
Preconditions	API running; default tasks loaded.
Test Action	Send a GET request to <code>http://localhost:3000/tasks</code> in Postman.
Expected Outcome	Response 200 OK with a JSON array of tasks, each containing id, title, description, status.

Test Title	View Task by ID
Test Author	Sohaam
Method & Endpoint	GET /tasks/1
Preconditions	Task with id = 1 exists.
Test Action	Send a GET request to <code>http://localhost:3000/tasks/1</code> .
Expected Outcome	Response 200 OK with a single JSON object for task id = 1.

Test Title	Create New Task
Test Author	Sohaam
Method & Endpoint	POST /tasks
Preconditions	API running.
Test Action	Send POST to /tasks with JSON: { "title": "Process today's orders", "description": "Pick, pack and ship all orders before 5pm." }
Expected Outcome	Response 201 Created with a new task JSON including a new id and status: "pending"

Test Title	Reject Task Without Title
Test Author	Sahaan
Method & Endpoint	POST /tasks
Preconditions	API running.
Test Action	Send POST to /tasks with JSON:{ "description": "This should fail – no title." }
Expected Outcome	Response 400 Bad Request with JSON error { "message": "Title is required" }.

Test Title	Update Task Status with PATCH
Test Author	Sahaan
Method & Endpoint	PATCH /tasks/4
Preconditions	Task with id = 4 exists with status = "pending".
Test Action	Send PATCH to /tasks/4 with JSON:{ "status": "completed" }
Expected Outcome	Response 200 OK with same task but status: "completed".

Test Title	Delete Existing Task
Test Author	Sahaan
Method & Endpoint	DELETE /tasks/2
Preconditions	Task with id = 2 exists.
Test Action	Send DELETE to /tasks/2, then GET /tasks to check.
Expected Outcome	DELETE returns 200 OK with { "message": "Task deleted successfully" }; subsequent GET no longer shows task id = 2.

Test Title	Workflow Stage 1 – Create Pending Task
Test Author	Sohaam
Method & Endpoint	POST /tasks
Preconditions	API running.
Test Action	Send POST to /tasks with JSON:{ "title": "Process today's orders", "description": "Pick, pack and ship all orders before 5pm." }
Expected Outcome	Response 201 Created with a new task JSON. The status field for the new task is automatically set to "pending", representing the first stage of the workflow

Test Title	Workflow Stage 2 – Complete Task
Test Author	Sohaam
Method & Endpoint	PATCH /tasks/{id} (e.g. PATCH /tasks/4)
Preconditions	Workflow Stage 1 has created a task (e.g. task id = 4) with status = "pending".
Test Action	Send PATCH to /tasks/4 with JSON:{ "status": "completed" }
Expected Outcome	Response 200 OK with the same task object, but status changed from "pending" to "completed", showing the second stage of the workflow.

CONCLUSION

The assignment shows how to create web APIs through REST by using a simple method which offers superior accessibility than SOAP and GraphQL. The REST API provides better performance and easier integration capabilities than SOAP and GraphQL because it offers a perfect mix of simplicity and speed which makes it suitable for small systems and educational projects.

The Task Management API implementation shows how RESTful APIs provide useful benefits when deployed in operational systems. The API provides simple endpoints for task management operations which include task creation and retrieval and modification and deletion functions and supports PATCH requests for updating existing data and follows a basic workflow from pending to completed status. The Postman testing process with test tables verifies that all endpoints function correctly for successful and error-based scenarios. The project demonstrates a complete process which starts with API style learning before moving to RESTful web service design and implementation and ends with service validation.

REFERENCES

- [1]
“Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST),” *Gbiv.com*, 2025. Available: https://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm
- [2]
GeeksforGeeks, “Different Types of APIs – REST API vs GraphQL vs SOAP,” *GeeksforGeeks*, Jan. 25, 2024. Available: <https://www.geeksforgeeks.org/blogs/rest-api-vs-graphql-vs-soap/>
- [3]
“Different Types of APIs – SOAP vs REST vs GraphQL,” *freeCodeCamp.org*, Mar. 07, 2023. Available: <https://www.freecodecamp.org/news/rest-vs-graphql-apis/>
- [4]
Steven, “Comparing API Protocols: REST vs. SOAP vs. GraphQL – Which Should Your Business Use?,” *Tribes*, Jun. 05, 2024. Available: <https://tribes.agency/insights/comparing-api-protocols-rest-vs-soap-vs-graphql-which-should-your-business-use>. [Accessed: Dec. 03, 2025]
- [5]
“SOAP Version 1.2 Part 1: Messaging Framework (Second Edition),” *W3.org*, 2025. Available: <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>. [Accessed: Dec. 03, 2025]
- [6]
“IBM Integration Bus,” *Ibm.com*, Jun. 21, 2023. Available: <https://www.ibm.com/docs/en/integration-bus/10.0.0?topic=standards-soap-11-12>. [Accessed: Dec. 03, 2025]
- [7]
The GraphQL Foundation, “GraphQL: A query language for APIs.,” *GraphQL.org*, 2012. Available: <https://graphql.org/>
- [8]
S. Buna, *GraphQL in action*. Shelter Island, Ny: Manning Publications Co, 2021.
- [9]
“GraphQL vs REST API - Difference Between API Design Architectures - AWS,” *Amazon Web Services, Inc.* Available: <https://aws.amazon.com/compare/the-difference-between-graphql-and-rest/>
- [10]

Vera, “GraphQL vs. REST API vs. SOAP: Comparison & Benefits,” *Knguru.de*, Sep. 13, 2025. Available: <https://www.knguru.de/en/blog/graphql-vs-rest-api-vs-soap>. [Accessed: Dec. 01, 2025]

[11]

“DigitalAPI,” *Digitalapi.ai*, 2022. Available: <https://www.digitalapi.ai/blogs/rest-vs-soap-key-differences>. [Accessed: Dec. 02, 2025]

API CREATION

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

79/100

Generic Assessment Criteria	Marks available	Marks awarded
1. Engagement with Literature Skills .	Assignment 1: 5	
Good number of sources, more of a range of credible and academic sources would have been desirable. References correctly used through the text.		3
2. Knowledge and Understanding Skills	Assignment 1: 10	
Investigation, use-cases and advantages / disadvantages. Good knowledge demonstrated of REST, SOAP and GraphQL with nicely detailed discussions of workings,		9

comparisons and use cases. A nicely detailed discussion of use-cases and comparison given. Some reasonable comparisons made within the narrative here.		
1. Cognitive and Intellectual Skills	Assignment 1: 30	
Design and documentation Test documentation and testing Good use of test tables, however a further box for PASS or FAIL would have been a desirable addition. Some good discussion of the api and the endpoints and nice use of both PUT and PATCH.		2 3. 5
4. Practical Application Skills	Assignment 1: 50	
AJAX implementation REST API A nice example of a functional REST API. Well commented and well packaged. A well packaged folder with package.json. A readme file may have been beneficial and would have cut down the word count in the report.		4 0

5. Transferable Skills for Life and Professional Practice	Assignment 1: 5	
Presentation A nicely presented report, clear introduction and conclusion. Images / figures should have been captioned. A high word count. This could have been avoided with a readme file in the code folder.		3.5

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22
