

Assignment 1: API Creation

CLOUD BASED WEB SERVICE, ACCA6024

RYAN HOPKINS (2221956)

Table of Contents

1. REST APIS AND ALTERNATIVE PROTOCOLS	2
1.1 WHAT IS A REST API?	2
1.2 ALTERNATIVE PROTOCOL GRAPHQL	3
1.3 ALTERNATIVE PROTOCOL GRPC	4
1.4 ADVANTAGES AND DISADVANTAGES OF REST	4
1.5 USE CASES FOR EACH PROTOCOL	5
2. API DESIGN AND IMPLEMENTATION	5
2.1 OVERVIEW AND DESIGN RATIONALE	5
2.2 ENDPOINTS	5
2.2.1 GET	6
2.2.2 POST	7
2.2.3 DELETE	8
2.3 EDIT FUNCTION	8
2.3.1 PUT	8
2.3.2 PATCH	9
2.4 MULTI-STAGE WORKFLOW	10
2.4.1 BASKET DATA TABLE	10
2.4.2 BASKET INSERT	11
2.4.3 BASKET CHECKOUT	12
3. SOFTWARE TESTING	12
3.1 TESTING APPROACH	12
3.2 TEST CASES	13
3.3 TEST ANALYSIS	14
4. CONCLUSION	14
5. BIBLIOGRAPHY	14

Introduction

This report will aim to explore what REST is, key architectural design differences between different protocols, and discussing advantages and disadvantages to using REST in unique situations. The second part of this report documents the development of a RESTful api and specifically demonstrates each HTTP method of GET, POST, PUT, PATCH and DELETE, the type of api chosen was an order processing system, which included products, a basket and a checkout. Overall, this report aims to show an understanding of the concept of REST and the ability to create and test a RESTful API

1. REST APIs and alternative protocols

1.1 What is a REST API?

REST stands for Representational State Transfer, created by Roy Fielding in his doctoral dissertation in 2000 [1]. The core motivation behind REST was to improve how systems communicate with one another over the web, with Fielding stating that the architectural style was designed to induce properties of "performance, scalability, simplicity, modifiability, visibility, portability, and reliability" [1]. Prior to REST, systems often used entirely different languages and syntaxes to communicate with one another, which frequently caused compatibility errors and made integration between systems complex.

RESTful APIs interact with data using five different HTTP methods. The first of which is GET this is used to retrieve data from the server without changing it. The second is POST, this is used to send new data to the server. The third is DELETE this is used to remove data from the server. Finally the fourth and fifth are PUT and PATCH, both are used to modify data however PUT replaces the source entirely and PATCH only updates specified fields. See figure 1 for a visual demonstration of REST.

Fielding defined six core controls that a RESTful API must follow to be achieved [2]. The first is client-server separation, meaning the user interface and the data storage are kept independent of one another, allowing each to develop without affecting the other. The second is statelessness, which means that every request sent from a client to a server must contain all the information needed to process that request, as the server does not store any memory of previous interactions [3]. The third constraint is cacheability, allowing responses to be stored temporarily to improve performance. The fourth is a uniform interface, ensuring all systems communicate using standardised HTTP methods such as GET, POST, PUT, PATCH and DELETE. The fifth is a layered system, meaning a client does not need to know whether it is communicating directly with the server or through a proxy server. Finally, the sixth constraint is code on demand, which is optional and allows servers to send executable code to clients when required [3]. Another key part of REST is resources this is pieces of data that can be accessed using a web address and interacted with using HTTP methods

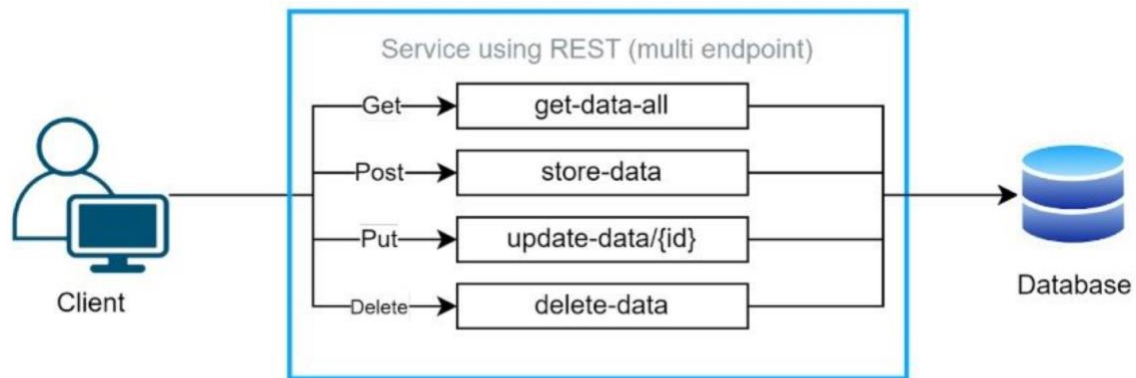


Figure 1 Rest API example [4]

1.2 Alternative protocol GraphQL

One alternative to REST is GraphQL, originally developed by Facebook in 2012, GraphQL went open source in 2015, this protocol was directly created to address over fetching data, on mobile devices, they needed to fix this as too much data can degrade the user performance. [4] Rather than having lots of different web addresses for different types of data like REST does, GraphQL uses just one single address for everything, and the app simply describes what information it needs each time [5]. The main downsides of GraphQL are that it is more complicated to set up than REST and can be excessive for simpler projects. It is also harder to speed up through caching, which is something REST handles very naturally. See figure 2 for a visual demonstration

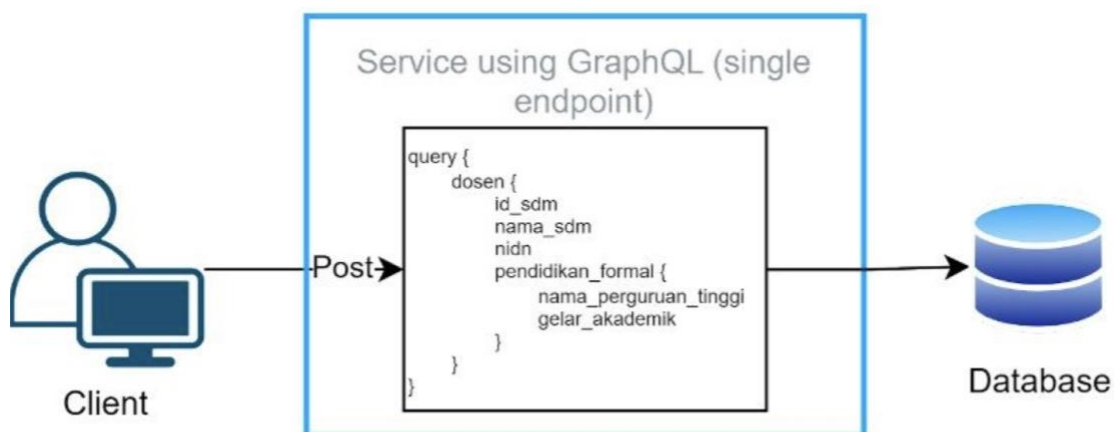


Figure 2 GraphQL example [4]

1.3 Alternative protocol gRPC

Another alternative protocol is gRPC, this is an open-source protocol developed by google and now maintained by Cloud Native Computing Foundation or CNCF [6]. gRPC is a service-based protocol, a client can call from the server as if they are locally hosted. This is achieved by using protocol buffers instead of JSON. Protocol buffers or protobuf is data which has been compressed into a binary format, which makes it more efficient in transport to the client [1]. Large companies such as Netflix and Uber use these systems, but for background communications between systems, as protobufs are incredibly strong at transmitting communication data but struggle when used for websites or applications due to the number of transmissions needed, this is directly quoted [7]. The main disadvantage of gRPC is that it does not directly work in a web browser without a complex setup, this explains why Netflix and Uber both use it for their background communication [1]. See figure 3 for a visual demonstration

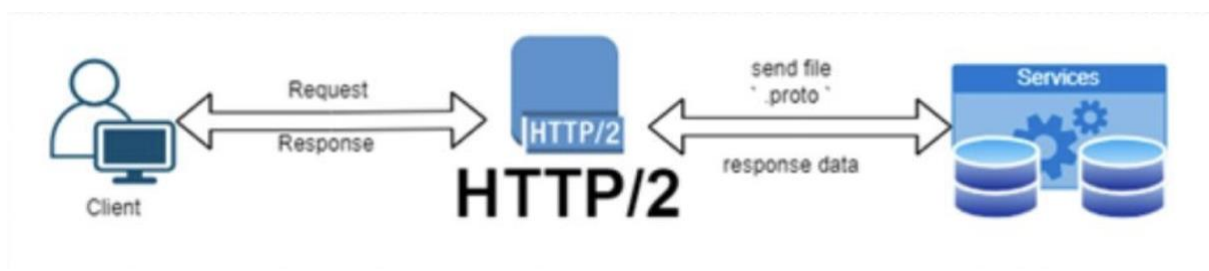


Figure 3 gRPC API example [4]

1.4 Advantages and disadvantages of REST

REST is commonly referred to as the easiest and most accessible ways to create APIs, mainly due to the use of HTTP methods that most developers already understand. A key advantage to REST is caching, because of the standardised http methods being used responses can be stored in browsers or proxy servers, which will reduce the overall requests made to the server improving the performance [1], this is something that alternative protocols like GraphQL struggle with, as it's a single endpoint approach which HTTP caching more challenging to implement.

Another advantage that REST benefits from is statelessness, meaning the server does not need to store information about previous requests, this makes RESTful APIs highly scalable as they can be distributed without loss of context. This is primarily useful in large scale systems where they might handle thousands of users simultaneously, as it allows the infrastructure to grow without significant changes to the core architecture.

However RESTful APIs have major drawbacks one of the most common of which is over-fetching and under-fetching data. As RESTful endpoints return a fixed value, the user might receive too much information, this could become a major problem on devices where bandwidth is limited such as mobile phones, which was the reason that Facebook developed GraphQL in 2012.

Also REST does not natively support real time communication, as it's a request model the server will only send data when the client requests it. This means its not well suited for applications which require live updates. In these situations alternatives like gRPC is better suited due as these capabilities are built into the protocol. Despite limitations REST remains one of the most used API styles due to its simplicity and compatibility

1.5 Use cases for each protocol

Each protocol previously discussed in this report is best suited to different situations depending on the needs of the application. REST is most commonly used for public-facing APIs where simplicity and wide compatibility are important. A well known example of this is Stripe, the online payment platform, which uses a REST API to allow developers to integrate payment processing into their applications easily [8]. Another example is Twitter, which also provides a REST API allowing developers to access tweets, user data and trends. [9]

GraphQL is best suited to applications where different clients need different subsets of the same data. GitHub is a prime example of this, offering a GraphQL API alongside its original REST API, allowing developers to request only the specific repository information, issues or pull requests they need in a single query rather than making multiple separate requests [5]. This makes it particularly useful for mobile applications where minimising the amount of data sent back and forth is important, which was in fact the original reason Facebook developed GraphQL in the first place [4].

gRPC is best suited to situations where speed and efficiency are the priority, particularly for communication happening behind the scenes between different parts of a system. Netflix and Uber both utilise gRPC internally to allow their services to communicate with each other as quickly as possible [6]. Unlike REST and GraphQL, gRPC is not typically used for public-facing APIs but instead excels in large-scale systems where performance is critical.

It is worth noting that many large organisations do not rely on just one protocol. A business might utilise REST for its public API, GraphQL for its own frontend application, and gRPC for internal communication between its services, choosing the most appropriate tool depending on the situation rather than committing to a single approach [1]

2. API design and implementation

2.1 Overview and design rationale

The development that's been implemented is an order processing system, using NodeJS with Express and JS files. The order processing system has been chosen as it displays all the necessary modifications. The data that is stored for the system is the name of the product, the price of the product and the stock level of that product. These fields could be expanded upon in the future by including a location finder, or a next delivery date, however the basic fields as previously mentioned will work the deliver an adequate

2.2 Endpoints

This section documents the core RESTful endpoints of the order processing API, demonstrating how data can be retrieved, created, and removed

2.2.1 Get

The GET endpoint allows the user to retrieve product data from the API, either returning all products in the inventory or a single product when a specific ID is provided

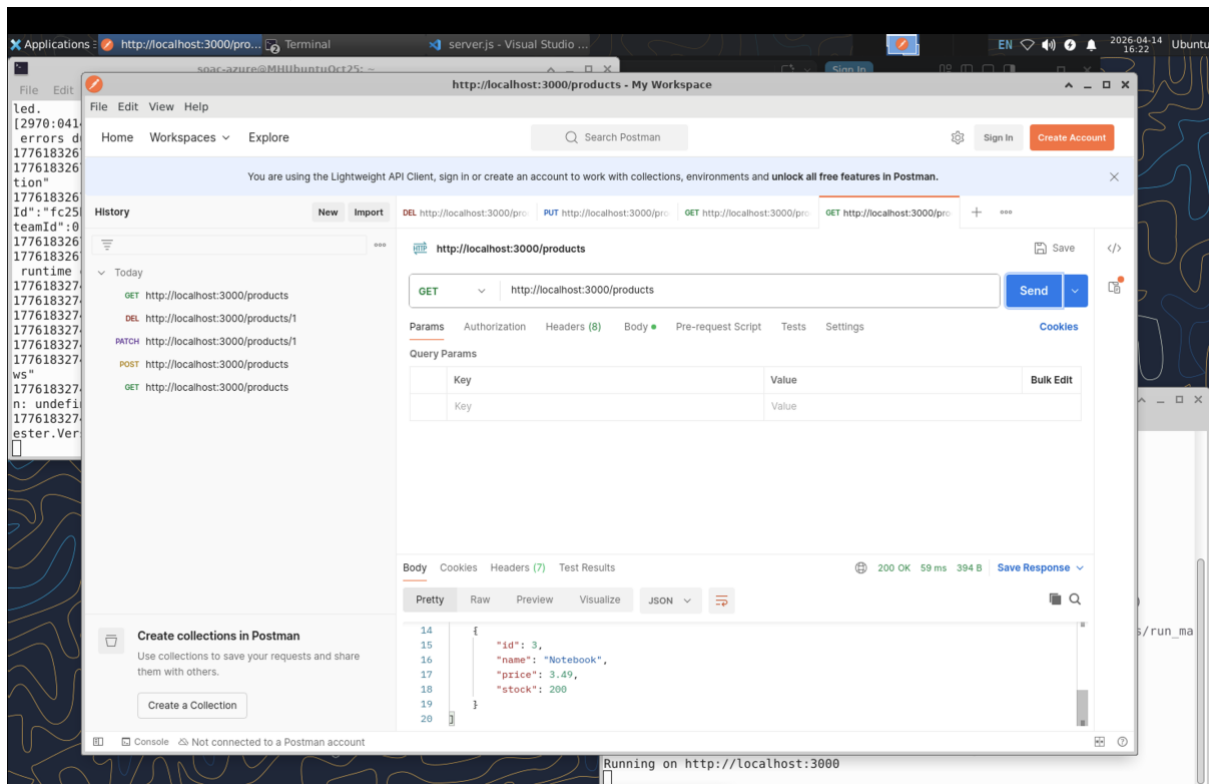


Figure 4 GET request

2.2.2 Post

The POST endpoint allows the user to add a new product to the inventory by sending the product details in the request body, which the API then stores and returns with an automatically assigned ID.

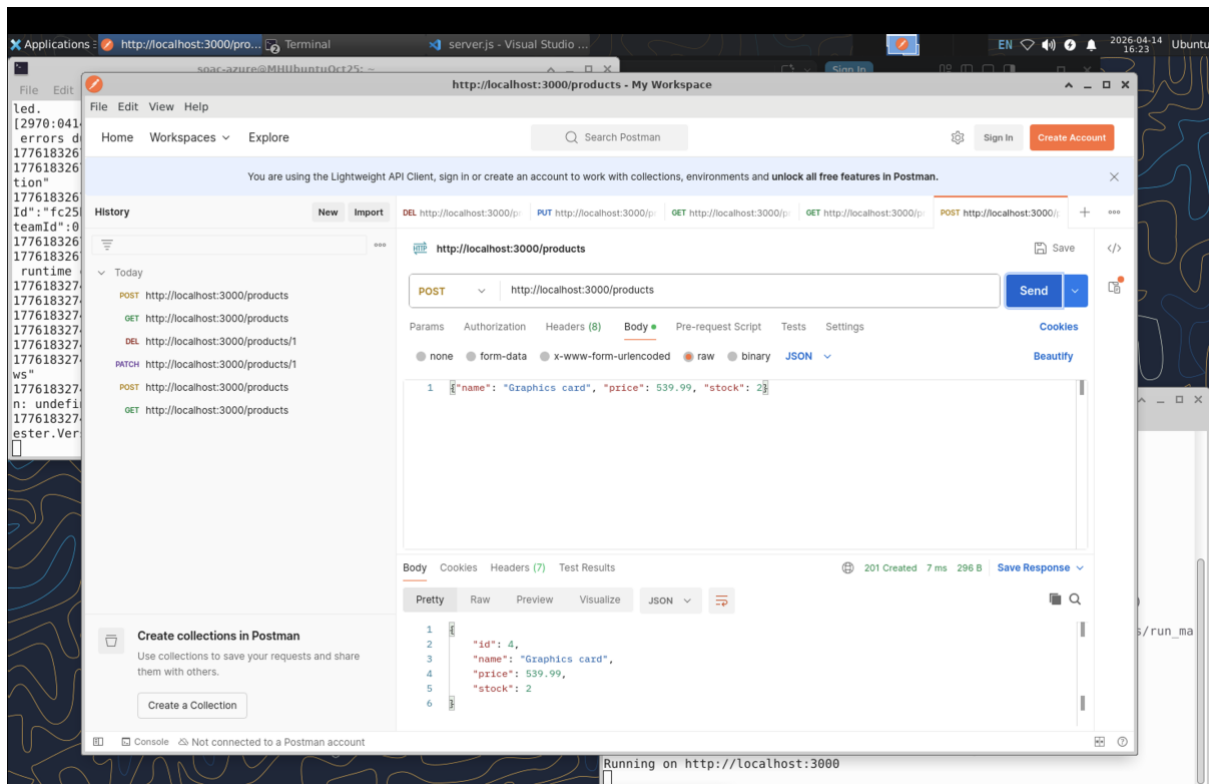


Figure 5 POST request

2.2.3 Delete

The DELETE endpoint allows the user to remove a product from the inventory by specifying its ID in the URL, with the API confirming the deletion or returning an error if the product does not exist.

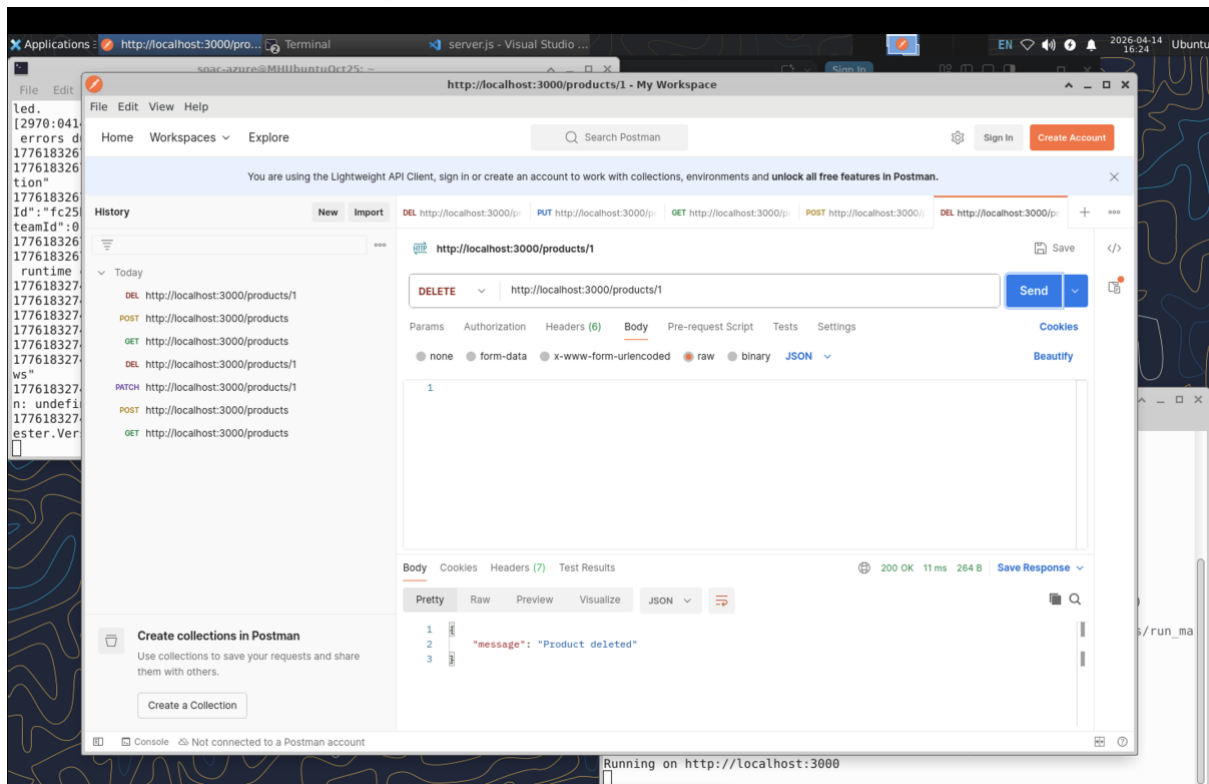


Figure 6 DELETE request

2.3 Edit function

This section demonstrates how existing products in the inventory can be edited, covering both the PUT and PATCH methods and the key differences between them.

2.3.1 Put

The PUT endpoint allows the user to completely replace an existing product by sending all of the product fields in the request body, meaning any fields that are not included will be overwritten.

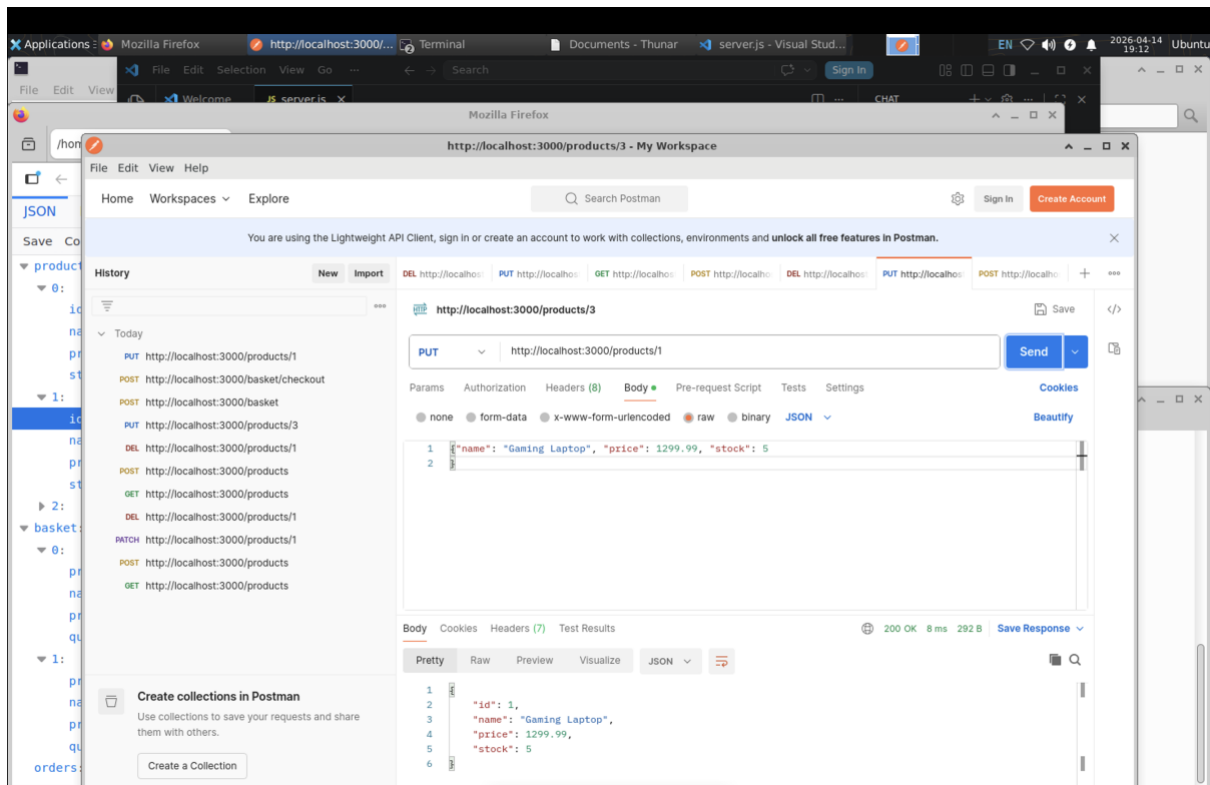


Figure 7 PUT request

2.3.2 Patch

The PATCH endpoint allows the user to update only specific fields of an existing product without affecting the rest of the data, making it a more flexible and precise method of editing compared to PUT

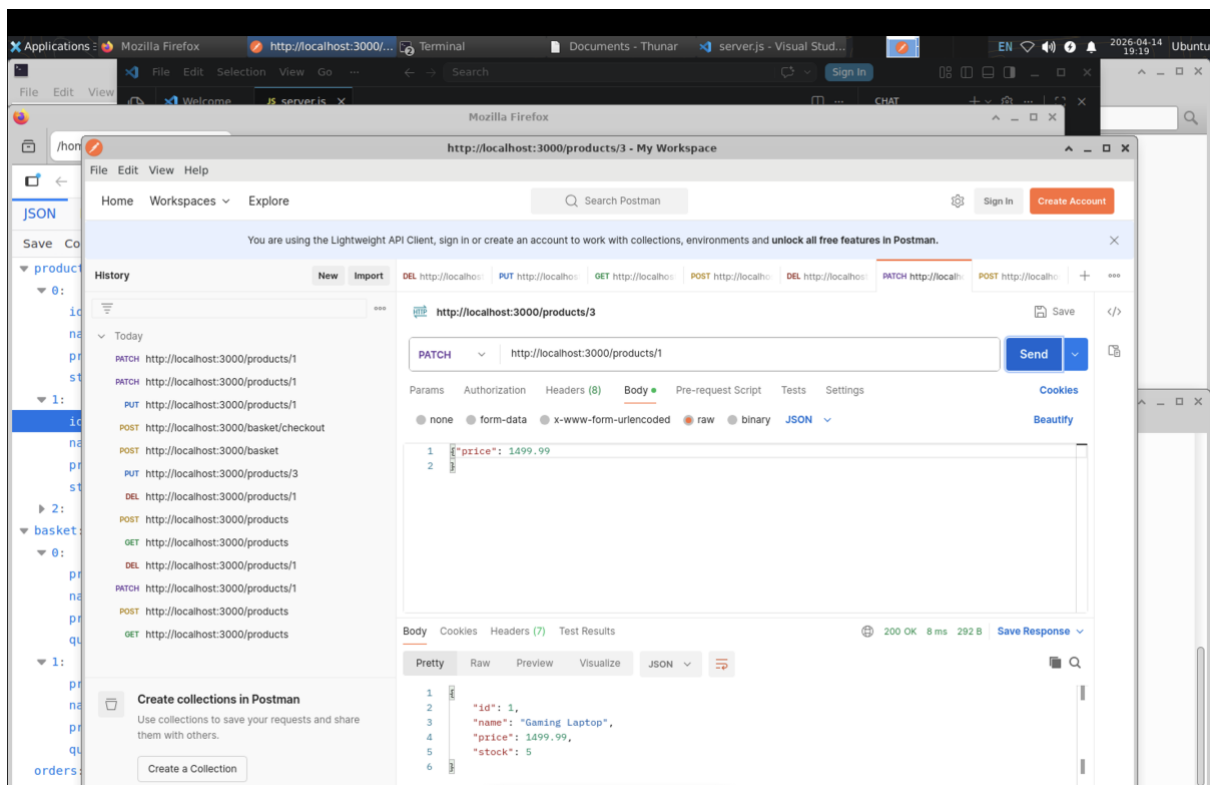


Figure 8 PATCH request

2.4 Basket and checkout workflow

This section demonstrates the two-stage order processing workflow implemented within the API, showing how a user can add products to their basket and proceed through to checkout, with stock levels reducing and an order record being created upon confirmation.

2.4.1 Basket data

This table displays the current contents of the basket, showing each product that has been added, the quantity selected, and the total price calculated by the API.

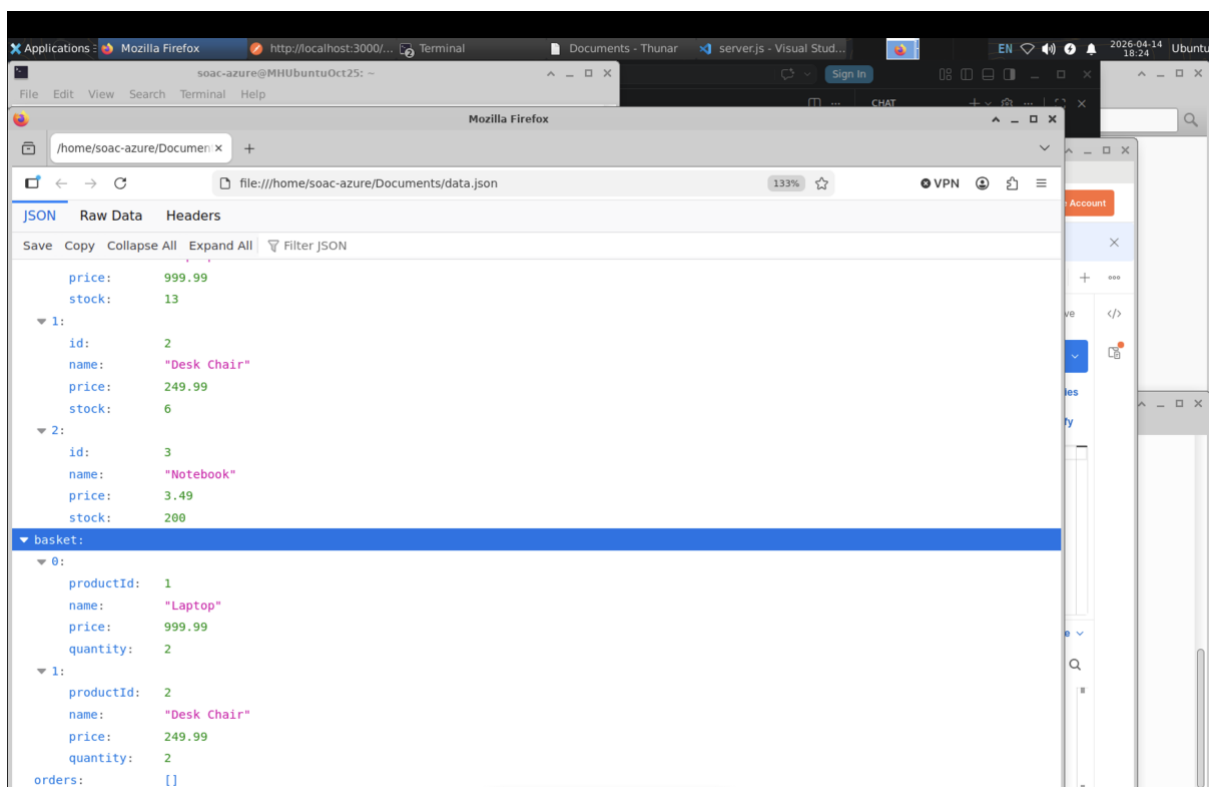


Figure 9 Website with shows the data inside the basket

2.4.2 Basket insert

This section demonstrates the first stage of the order processing workflow, showing how a product is added to the basket via the API along with the corresponding reduction in stock levels

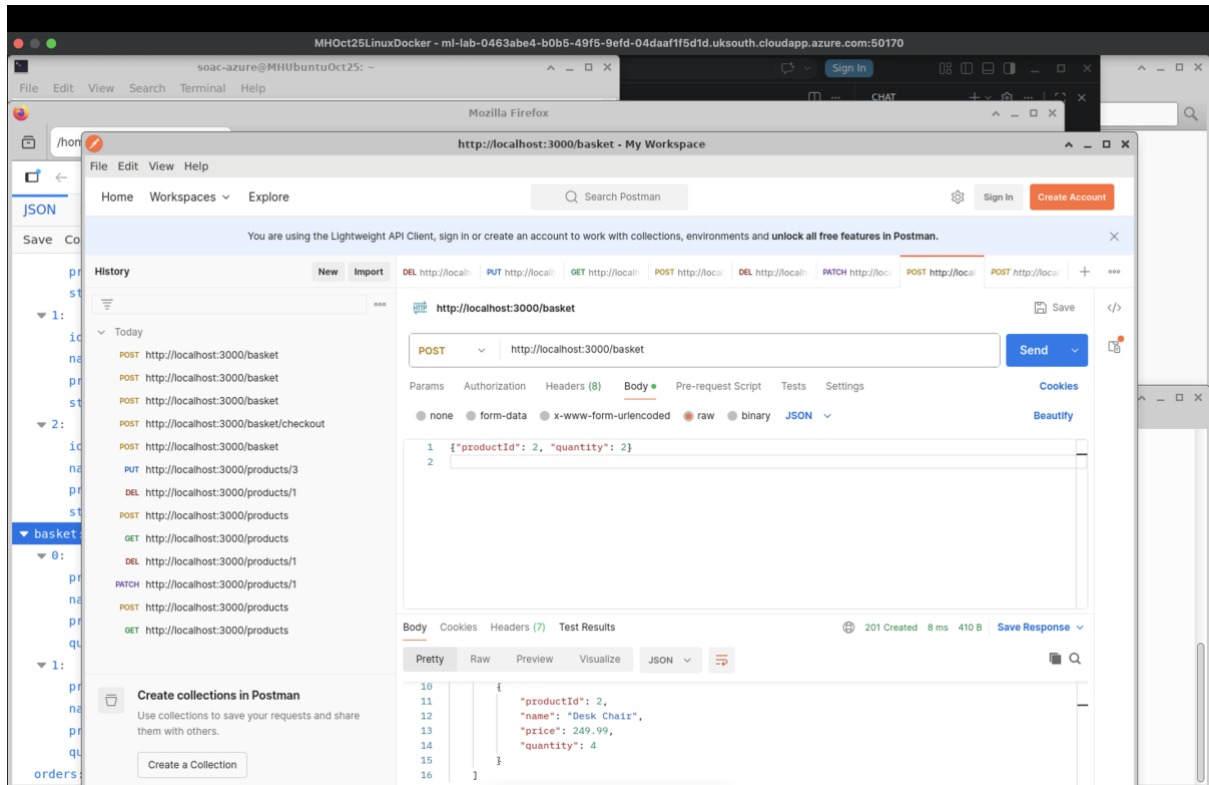


Figure 10 Inserting into the basket

2.4.3 Basket checkout

This section demonstrates the second and final stage of the order processing workflow, showing how the user confirms their order through the checkout process, resulting in the basket being cleared and a confirmed order record being created.

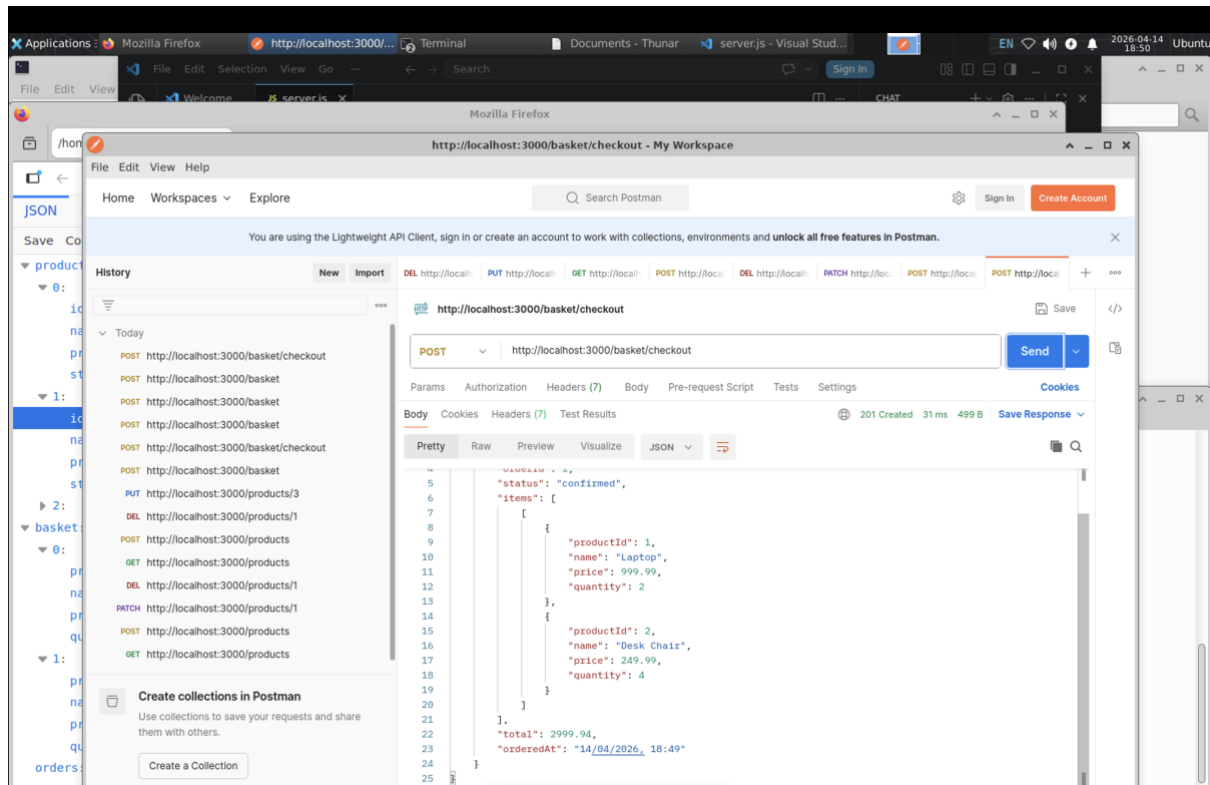


Figure 11 basket checkout

3. Software testing

3.1 Testing approach

This section outlines the approach taken to test the Order processing API, covering the methodology selected and how each test was carried out.

3.1.1 Testing Methodology

For this API a manual testing approach was chosen using postman, a widely used tool that allows for http requests to be sent to an API and gives visual responses, this approach was selected due to the simplicity and visual nature, as it ensures that all endpoints are returning correct values. Manual testing was considered most appropriate for this API due to the scale of this, as it allows for each request and response to have a documented response or screenshot.

3.1.2 Test Execution

Each endpoint was tested individually by sending requests through Postman with the appropriate HTTP method, URL, and request body where required. Tests were designed to cover both successful

scenarios, such as successfully adding a product to the basket, and error scenarios, such as attempting to checkout with an empty basket. The results of each test were recorded in the test table in section 3.2, with the expected outcome defined before testing took place and the actual outcome filled in afterwards.

3.2 Test cases

Test ID	Test Name	Method	URL	Test Data	Expected Outcome	Actual Outcome
T-01	GET all products	GET	/products	None	200 OK returns a list of all 3 products	200 OK returns a list of all 3 products
T-02	GET one product (found)	GET	/products/1	None	200 OK returns Laptop	200 OK returns Laptop
T-03	GET one product (not found)	GET	/products/999	None	404 Not Found { "error": "Product not found" }	404 product not found
T-04	POST create product	POST	/products	{ "name": "Monitor", "price": 349.99, "stock": 10 }	201 Created returns new product with id: 4	201 new product with id as 4
T-05	PUT replace product	PUT	/products/1	{ "name": "Gaming Laptop", "price": 1299.99, "stock": 5 }	200 OK, product fully replaced with new values	200 product fully replaces with new values
T-06	PATCH update stock only	PATCH	/products/1	{ "stock": 3 }	200 OK, only stock updated, name and price unchanged	200, only stock updated
T-07	PATCH update price only	PATCH	/products/2	{ "price": 199.99 }	200 OK, only price updated, name and stock unchanged	200 only price changed
T-08	DELETE product (found)	DELETE	/products/1	None	200 OK, { "message": "Product deleted" }	200 product deleted
T-09	DELETE product (not found)	DELETE	/products/999	None	404 Not Found { "error": "Product not found" }	404 not found
T-10	Add item to basket (Stage 1)	POST	/basket	{ "productid": 1, "quantity": 2 }	201 Created, item added, stock	201 reduced available stock

Test ID	Test Name	Method	URL	Test Data	Expected Outcome	Actual Outcome
					reduced from 10 to 8	

3.3 Test analysis

The results demonstrate that the API behaves correctly across a range of scenarios including successful and unsuccessful requests. Tests T-01 and T-02 confirmed that the GET endpoints retrieve data accurately, whilst T-03 verified that the API handles requests for non-existent resources correctly by returning a 404 status code rather than crashing or returning incorrect data.

Tests T-06 and T-07 are particularly significant as they confirm that the PATCH implementation is working correctly with only the specified field being updated whilst all other fields remain unchanged. This demonstrates a clear distinction between PATCH and PUT, with T-05 confirming that PUT replaces the entire resource as expected. Tests T-08 and T-09 further confirmed that the DELETE endpoint handles both found and not found scenarios appropriately.

Finally T-10 confirmed that the first stage of the order processing workflow functions correctly, with the item being added to the basket and the stock level reducing accordingly from 10 to 8.

4. Conclusion

This report has covered the development of an order processing API Built with Node.JS and express, which allowed users to add products into their basket and complete the checkout process. There are some limitations to the API built, the main concern is the data is stored in the servers memory, which means everything is wiped when the server turns off as there is no database.

5. Bibliography

- [1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 16 April 2026. [Online]. Available: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf.
- [2] A. Ehsan, A. A. M. E. Mohammed, C. Catal and D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," *Applied Sciences*, vol. 12, no. 9, p. 4369, 2022.
- [3] W. C. Li Li, "Design and Describe REST API without Violating REST: A Petri Net Based Approach," *IEEE Xplore*, vol. 70, no. 2, p. 508–515, 2011.

- [4] GraphQL Foundation, "GraphQL: A query language for APIs.," GraphQL Foundation, 01 Jan 2026. [Online]. Available: <https://graphql.org/learn/>. [Accessed 20 Apr 2026].
- [5] GitHub, "GitHub GraphQL API," GitHub, 01 Jan 2026. [Online]. Available: <https://docs.github.com/en/graphql>. [Accessed 20 April 2026].
- [6] gRPC Authors, "gRPC," gRPC, 01 Jan 2026. [Online]. Available: <https://grpc.io>. [Accessed 20 Apr 2026].
- [7] R. A. S. A. B. Muhammad Niswar, "Performance evaluation of microservices communication with REST, GraphQL, and gRPC," *International Journal of Electronics and Telecommunications*, vol. 70, no. 2, pp. 429-436, 2024.
- [8] Stripe, "Stripe API Reference," Stripe, [Online]. Available: <https://docs.stripe.com/api>. [Accessed 16 April 2026].
- [9] X, "X Developer Platform," X, 01 Jan 2026. [Online]. Available: <https://docs.x.com/overview>. [Accessed 19 April 2026].