



CLOUD BASED WEB SERVICES

Assignment 2 Ryan Michael Hopkins

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.]

When you're ready to add your content, just click here and start typing.]

Ryan Hopkins (2221956)

[Email address]

Table of Contents

1.Introduction.....	2
2. Frontend Integration.....	2
2.1 Overview and Design Rationale	2
2.1.1 Package.Json	2
2.2 Index.html.....	2
2.3 Style.css	3
2.4 AJAX.....	3
2.5 Data storage	4
2.6 Data.JSON	4
3. Testing	4
3.1 Testing Approach	4
3.1.1 Testing Methodology	4
3.1.2 Test Execution.....	4
3.2 Test Cases	4
3.3 Test Analysis	6
3.3.1 Test Results	6
3.3.2 Analysis	6

1.Introduction

This report aims to document the frontend integration for the Order Processing API, discussing how it was set up, how each file works and testing to ensure that the application functions correctly. The report covers the backend built using Node.js and Express, the frontend built using HTML, CSS and JavaScript, and the use of AJAX to allow the two to communicate without a page reload.

2. Frontend Integration

This section covers the frontend of the application, built using HTML, CSS and JavaScript working together to provide a simple interface for the Order Processing API

2.1 Overview and Design Rationale

The Order Processing API was built using Node.js and Express, two widely used technologies for building REST APIs. Node.js was chosen due to its lightweight nature and ability to handle multiple requests efficiently, whilst Express was selected as it provides a simple and minimal framework for defining API endpoints quickly.

2.1.1 Package.Json

The package.json file is a configuration file that every Node.js project requires. It stores important information about the project such as the name and version. It also stores the main file that's being hosted, for this instance its server.js.

2.2 Index.html

The index.html file provides the structure of the web page. It contains sections for displaying the product list, viewing the basket, adding a new product, updating a product using PATCH, and viewing the order confirmation. Each section is linked to the JavaScript file which handles all the communication with the API. See figure 1 for the HTML output

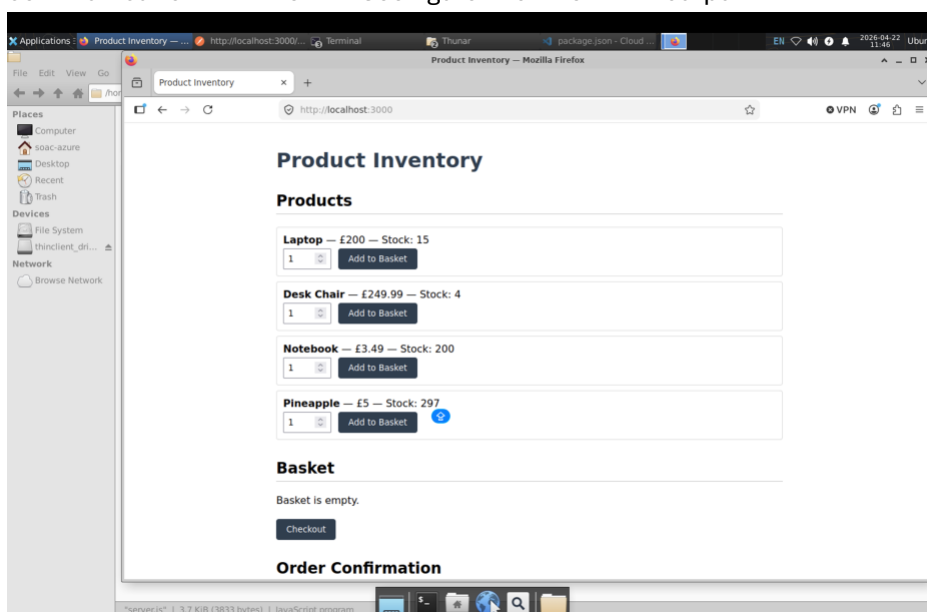


Figure 1: HTML output with CSS integration.

2.3 Style.css

The style.css file handles the visual presentation of the page. It keeps the layout clean and easy to read by styling the product listings, input fields and buttons. No complex design was needed as the application only needs to be functional rather than visually appealing. See figure 1 for the HTML and CSS output. See figure 2 for the style sheet.

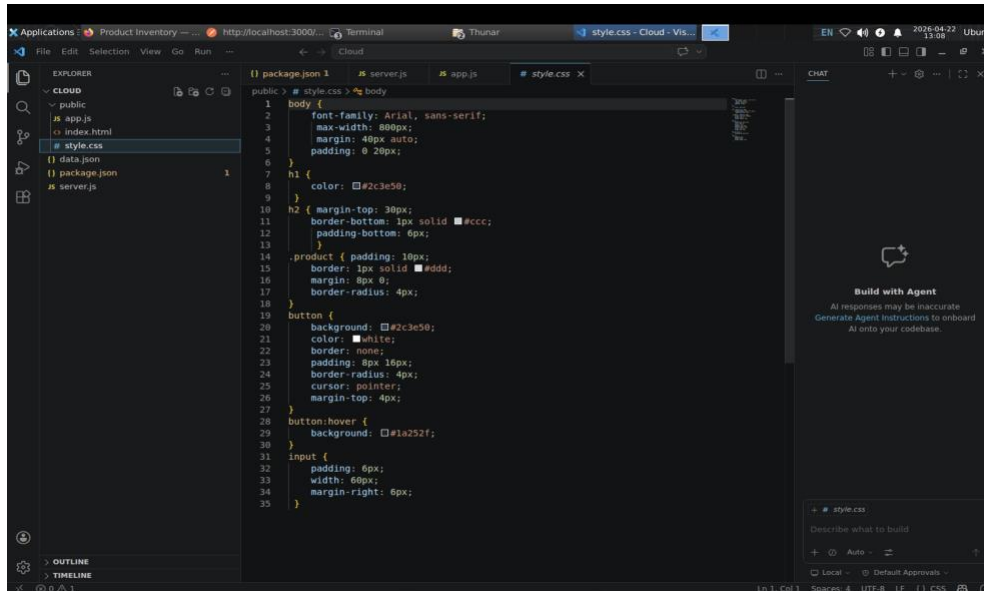


Figure 2: CSS stylesheet.

2.4 AJAX

AJAX (Asynchronous JavaScript and XML) is used throughout the application via the JavaScript fetch function. This allows the page to communicate with the API in the background without needing to fully reload each time an action is performed. See figure 3 for the network tab showing the interaction between frontend and backend through the AJAX.

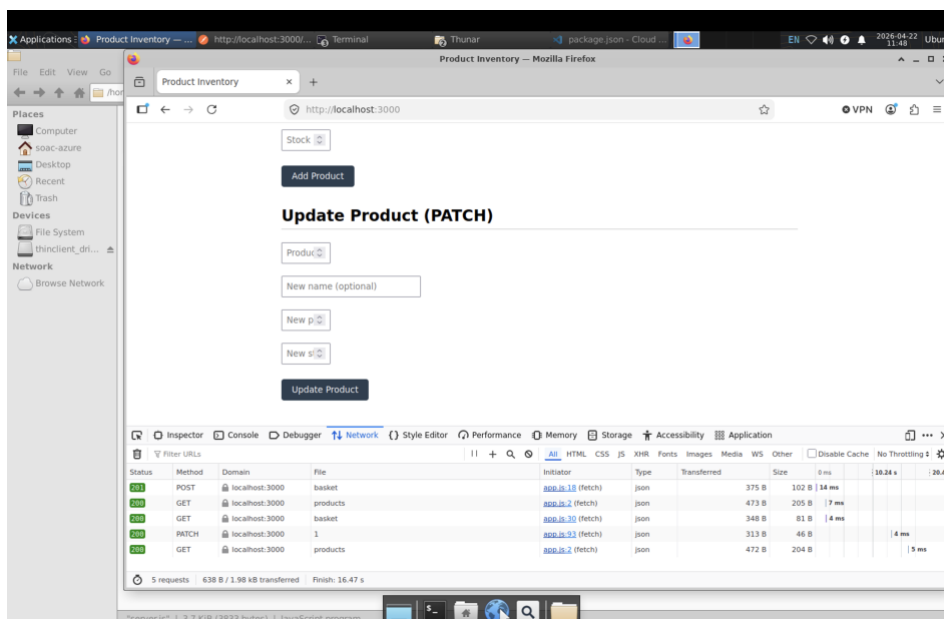


Figure 3: Index.HTML with the network tab opened.

2.5 Data storage

this application stores all data in memory using JavaScript arrays. This means the products, basket and orders are held in the server whilst it is running but are lost when the server is switched of. This approach was chosen to keep the application simple and easy to run locally without requiring any additional software

2.6 Data.JSON

When developing the API difficulties arose where the developer couldn't tell if the basket was full or empty, this was the debug fix so every time the server is started the Data.JSON file will be reset back to the basic entry.

3. Testing

3.1 Testing Approach

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

3.1.1 Testing Methodology

For this application, the browser was used to test the frontend functionality including the product list, basket and checkout process. This approach was selected as it allows both the API and the frontend to be tested together, providing a complete picture of how the application behaves from the user's perspective

3.1.2 Test Execution

Each feature was tested individually by interacting directly with the frontend in the browser. Tests were designed to cover both successful scenarios and error scenarios such as adding too many items to the basket or attempting to checkout with an empty basket. The browser developer tools Network tab was also used to verify that AJAX requests were being sent in the background without a page reload. Results were recorded in the test table in section 3.2.

3.2 Test Cases

Test ID	Test Name	Location	Test Data	Expected Outcome	Actual Outcome
T-01	Page loads and products display	Browser product list	None	Page loads showing all 3 products with name, price and stock	

Test ID	Test Name	Location	Test Data	Expected Outcome	Actual Outcome
T-02	Add new product via form	Frontend add product form	{ "name": "Monitor", "price": 349.99, "stock": 10 }	New product appears in the list instantly without page reload	
T-03	Update product via PATCH form	Frontend PATCH form	{ "stock": 3 }	Only stock updates, name and price remain unchanged	
T-04	Add item to basket	Frontend basket section	Product ID 1, quantity 2	Item added to basket, stock reduces from 15 to 13	
T-05	Remove item from basket	Frontend basket section	None	Item removed, stock restored, basket updates instantly	
T-06	Not enough stock error	Frontend basket section	Product ID 2, quantity 999	Alert message — not enough stock	
T-07	Checkout basket	Frontend checkout button	None	Order confirmed, basket cleared, order details shown on screen	

3.3 Test Analysis

3.3.1 Test Results

Test ID	Test Name	Location	Test Data	Expected Outcome	Actual Outcome
T-01	Page loads and products display	Browser product list	None	Page loads showing all 3 products with name, price and stock	Page loads without issues see figure 1
T-02	Add new product via form	Frontend add product form	{ "name": "Monitor", "price": 349.99, "stock": 10 }	New product appears in the list instantly without page reload	Creating a product shows lower on the list see figure 4
T-03	Update product via PATCH form	Frontend PATCH form	{ "stock": 3 }	Only stock updates, name and price remain unchanged	Modifies a product depending on what is selected see figure 5
T-04	Add item to basket	Frontend basket section	Product ID 1, quantity 2	Item added to basket, stock reduces from 15 to 13	Stock gets reduced in the product overview but added to basket see figure 6
T-05	Remove item from basket	Frontend basket section	None	Item removed, stock restored, basket updates instantly	Item gets removed and stock gets increased see figure 7
T-06	Not enough stock error	Frontend basket section	Product ID 2, quantity 999	Alert message not enough stock	Popup error message not enough stock see figure 8
T-07	Checkout basket	Frontend checkout button	None	Order confirmed, basket cleared, order details shown on screen	Order confirmed and basket gets cleared figure 9

3.3.2 Analysis

The results demonstrate that the application behaves correctly across all tested scenarios. T-01 confirmed that the page loads successfully and all products are retrieved and displayed through an AJAX GET request on page load. T-02 and T-03 confirmed that the add product form and PATCH form both communicate with the API correctly, with T-03 being particularly important as it proves that only the specified field was updated whilst all others remained unchanged. T-04 and T-05 confirmed that the basket workflow functions correctly, with items being added and removed instantly and stock levels updating accordingly each time. T-06 confirmed that the application handles errors correctly by displaying an appropriate message when the requested quantity exceeds the available stock. T-07 confirmed that the full order processing workflow operates as expected, with the order being confirmed and the basket clearing upon checkout.

3.5 Testing Screenshots

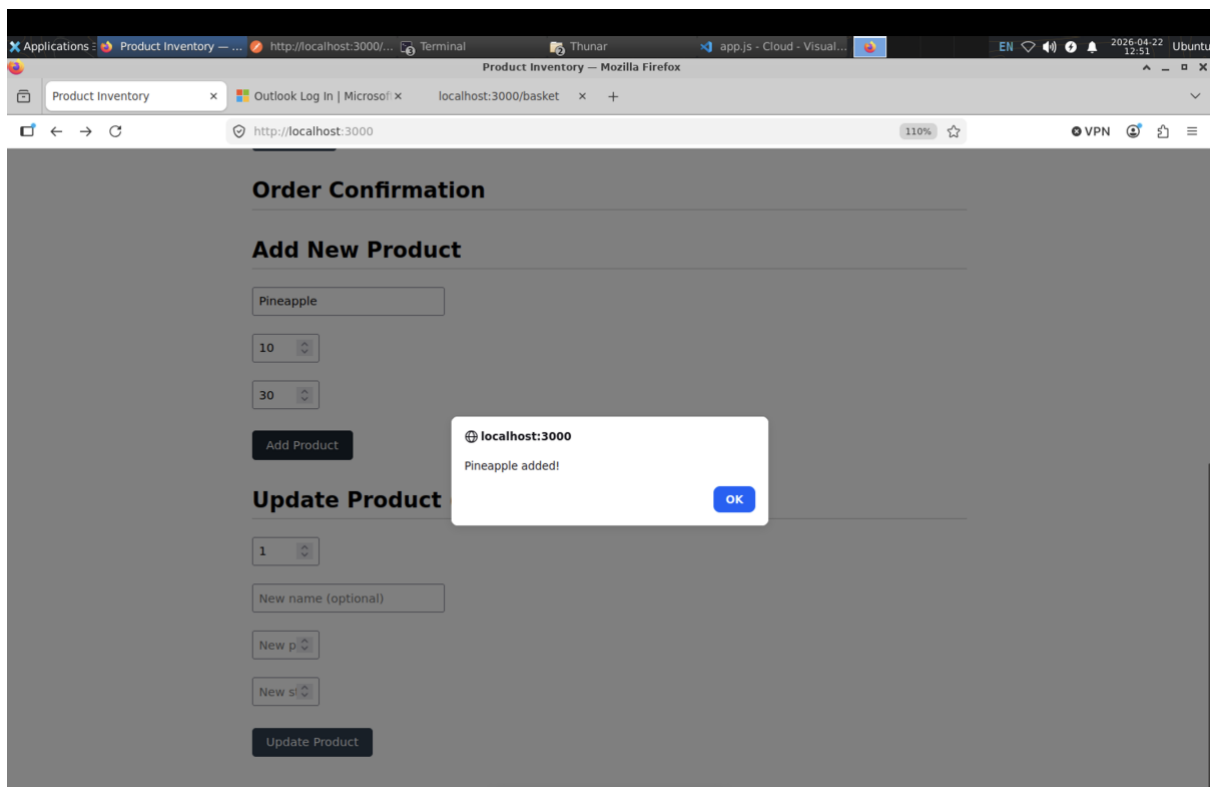


Figure 4: creating a new product.

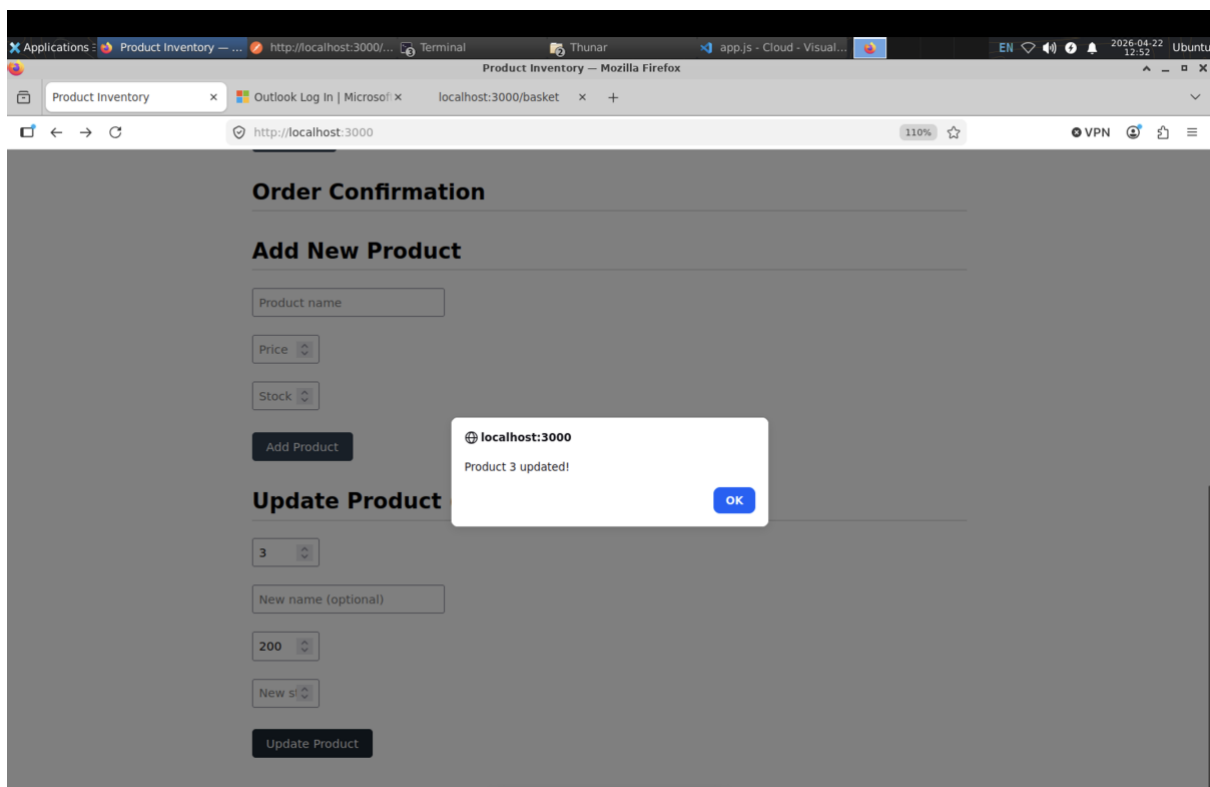


Figure 5: modifying product number 3's price to 200.

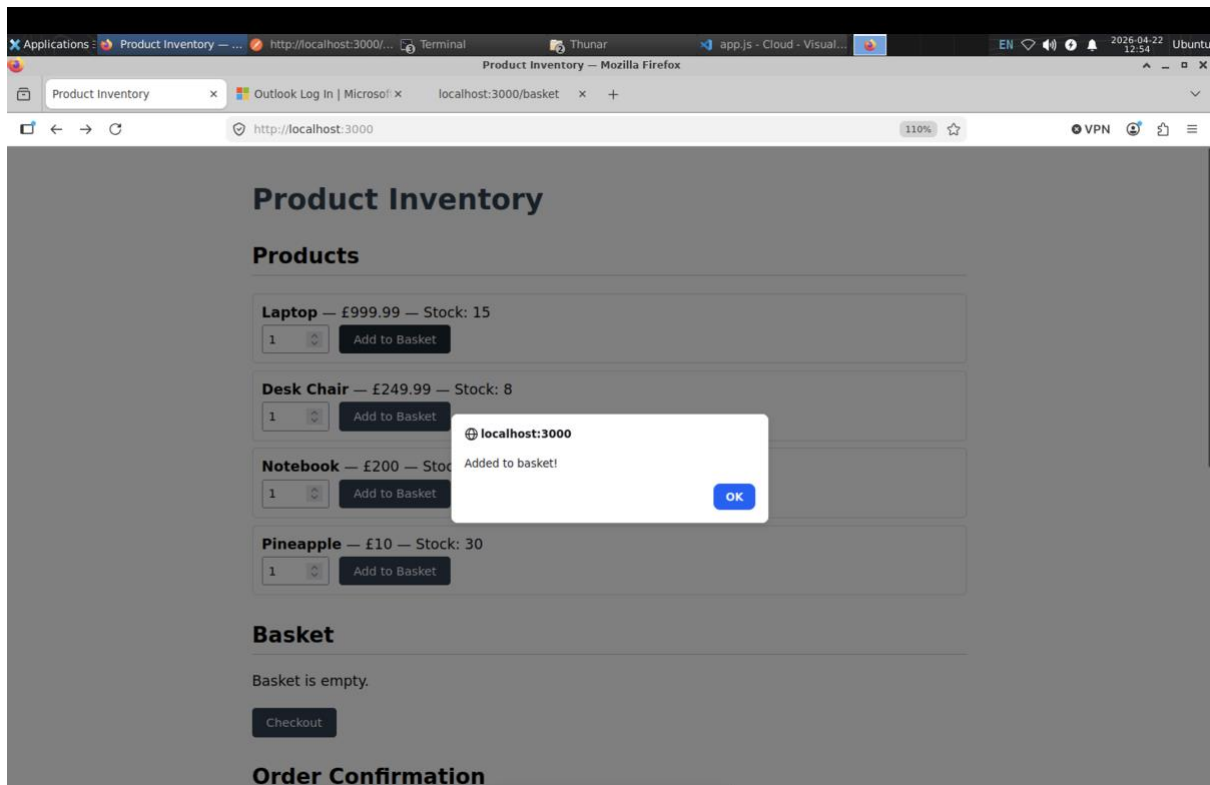


Figure 6: adding a product into the basket

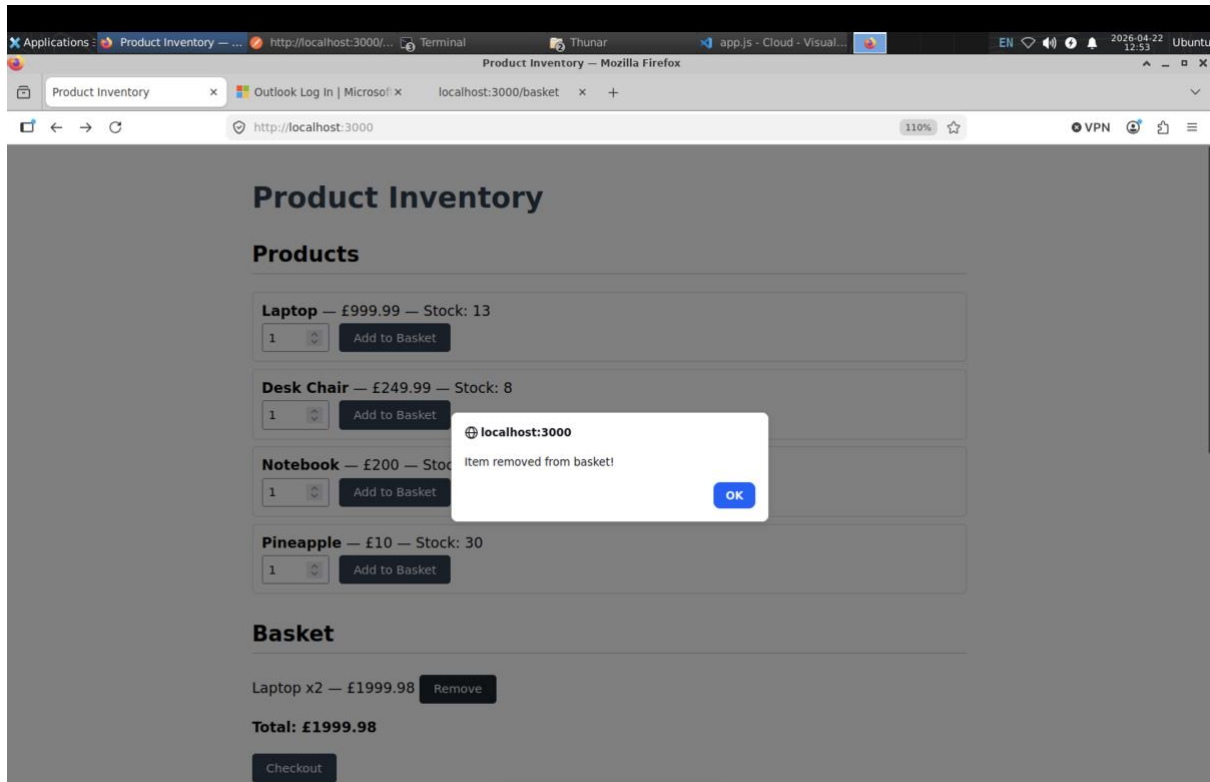


Figure 7: removing item from the basket

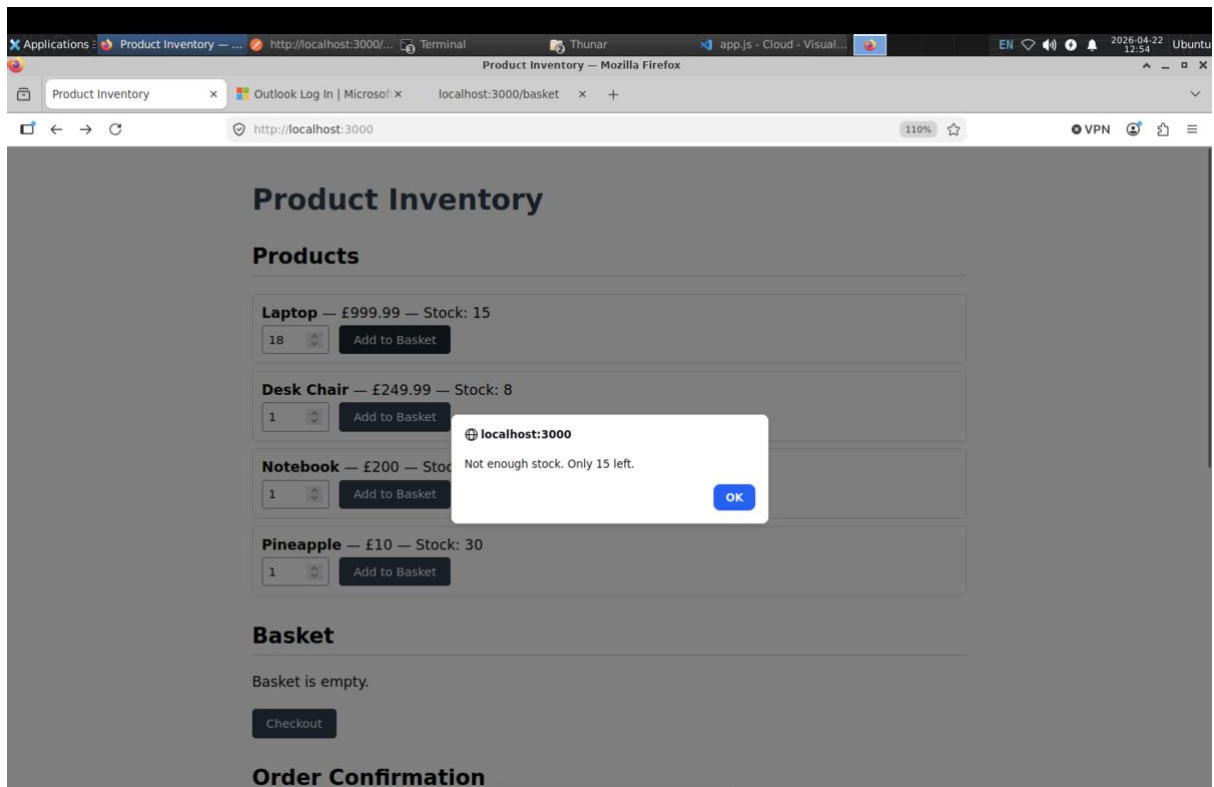


Figure 8: trying to reserve too much stock

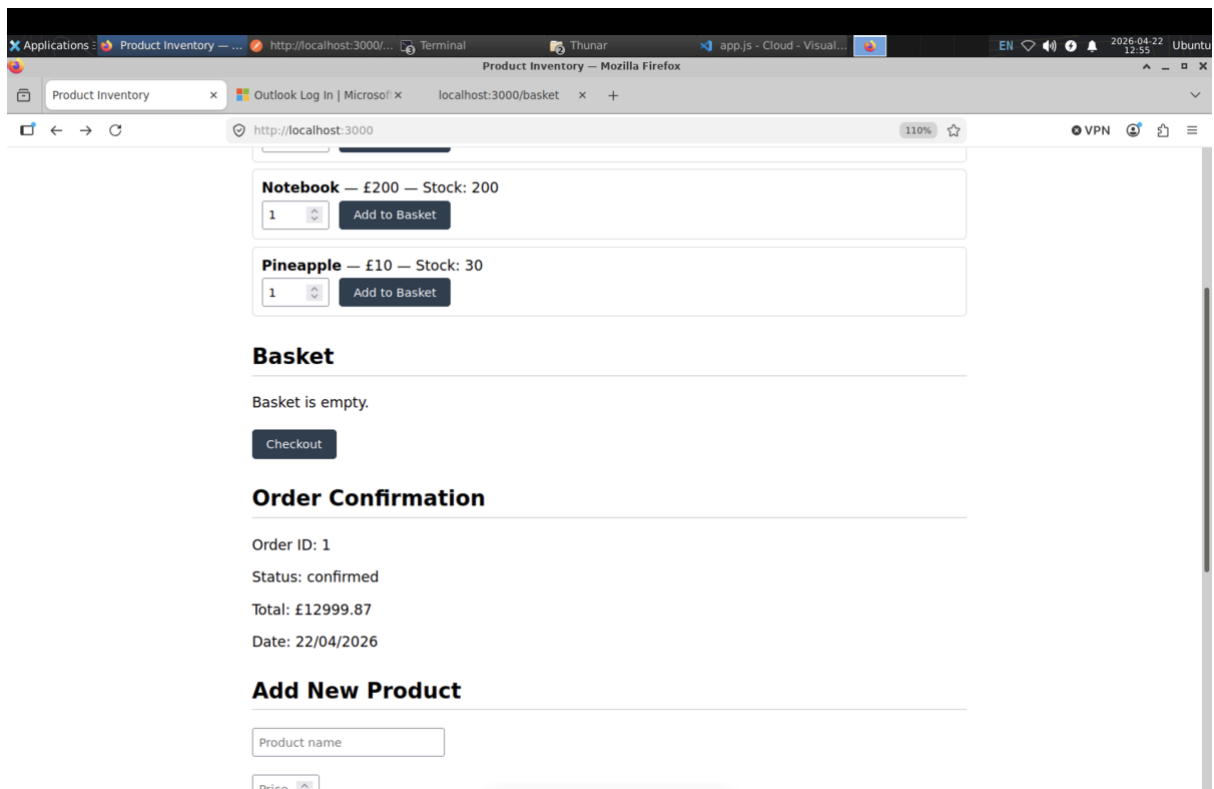


Figure 9: order confirmation.