**≪** Share

**Admin Required** 

No

Last indexed: 11 August 2025 (172577)

Repository Structure **Getting Started** 

Overview

(Proy1)

Project 1: Product Management

Database & Configuration

Product Management System **Development Environment** 

Project 2: Course Platform

(Proy2\_Cursos)

Authentication & Middleware

Data Models & Database

Development & Testing

Project 3: Food Delivery Platform (Proy3\_Pedidos)

Architecture & Dependencies Database Models & Schema

**Environment & Configuration** 

Development Environment

**Dependency Management** 

VS Code & Debugging

Create Form	GET	/products/new Show creation form		Yes	
Create	POST	/products/new	Process new product	Yes	
Update Form	GET	/products/edit/:id	Show edit form	Yes	
Update	POST	/products/edit/:id	Process product update	Yes	
Delete	GET	/products/delete/:id	Delete product	Yes	
Product CRUD Flow Diagram					

Purpose

List all products

This document details the product CRUD operations, routing, and business logic implementation for

role-based access control and form validation. For database configuration and schema details, see

Project 1 (Proy1). The system provides a complete product inventory management interface with

<u>Database & Configuration</u>. For development setup information, see <u>Development Environment</u>.

The product management system implements a complete CRUD (Create, Read, Update, Delete)

interface through a RESTful routing pattern. All product-related routes are handled by the

Update	POST	/produc	ts/edit/:id	Pro	Process product update		e Yes	
Delete	GET	/produc	/products/delete/:id Delete product		Yes			
Product CF	RUD Flow	Diagram						
/products/	/products/new GET	/products/edit/id GET	Client Browser /products/new POST		/products/ed	it/id POST		
db.all(SELECT * FROM products')	req?.token?.admin == true	req?.token?.admin == true	req?.token?.admin == true exp	press-validator r	req?.token?.admin == true	express-validator	/products/delet	e/id GET
products.ejs	products_new.ejs	products_edit.ejs	INSER	T INTO products		UPDATE products SET	req?.token?.admin == true	DELETE FROM products WHERE Id

The system implements a token-based authorization model where administrative operations

**Product Management System** 

**Route Structure and CRUD Operations** 

Route

/products/

> Relevant source files

products.js router module.

**CRUD Route Mapping** 

Method

**GET** 

Operation

Read All

# **Admin Authorization Check**

All destructive operations (Create, Update, Delete) implement the same authorization pattern: if(req?.token?.admin != true){

# return

res.status(400).json("no allowed access, need to be admin")

**HTTP Request** Public Access req?.token?.admin GET /products/

Not empty, ASCII only description query('description').notEmpty().isAscii()

**Implementation** 

query('name').notEmpty().trim().isAscii()

query('price').notEmpty().isInt({min:1})

query('stock').notEmpty().isInt({min:1})

**Error Handling** 

500 status

Promise rejection with

Try-catch with 500 status

Promise rejection with

Yes

The system implements comprehensive input validation using express-validator middleware and

Not empty, trimmed, ASCII only name

**Validation Rules** 

handles price normalization for storage.

### Not empty, integer, minimum 1

i noc nanating					
The system implements a currency normalization pattern where prices are:					
Stored in the database as integers (cents)					
Displayed to users as decimal values (dollars)					
• Converted during processing: req.body.price * 100					
Sources: Proy1/routers/products.js 57-60 Proy1/routers/products.js 147-150					

Proy1/routers/products.js 164

List

**Products** 

Create

Product

Read Single

The system uses raw SQLite queries with Promise-based error handling for all database interactions.

**SQL Operations by Function** Operation **SQL Statement** 

SELECT \* FROM products

SELECT \* FROM products WHERE id = ?

(?, ?, ?, ?)

		500 status		
Update Product	<pre>UPDATE products SET name = ?, description = ?, price = ?, stock = ? WHERE id = ?</pre>	Promise rejection with 500 status		
Delete Product	DELETE FROM products WHERE id = ?	Try-catch with 500 status		
Database Interaction Pattern				

Router Method

new Promise((resolve, reject))

db.all/get/run()

err?

INSERT INTO products (name, description, price, stock) VALUES

resolve(rows/row) console.error('Error:', err) res.status(500).send(err) reject(err) Process Query Result throw new Error(err) res.render() or res.redirect() Sources: Proy1/routers/products.js 18-28 Proy1/routers/products.js 77-78 Proy1/routers/products.js 161-173 Proy1/routers/products.js 121-132 Proy1/routers/products.js 99-100 **View Templates and User Interface** The system uses EJS templates with Bootstrap for responsive UI components. The interface adapts based on user authorization levels.

**Key Features** 

Card-based layout, conditional admin buttons

Full product form, admin-only access

Pre-populated form, admin-only access

products\_new.ejs

Name, Description, Price,

Stock

POST /products/new

Proy1/views/products\_new.ejs 30-54

Proy1/views/products.ejs 34-38

products\_edit.ejs

Pre-populated Form Fields

POST /products/edit/:id

**Redirect Behavior** 

Returns to product

None

None

list

## This pattern is used for: Add Product button enablement

Conditional Admin Buttons

/products/new/ link

/products/delete/:id

Response Type

be admin")

res.status(500).send(error)

res.redirect('/products')

res.status(400).json("no allowed access, need to

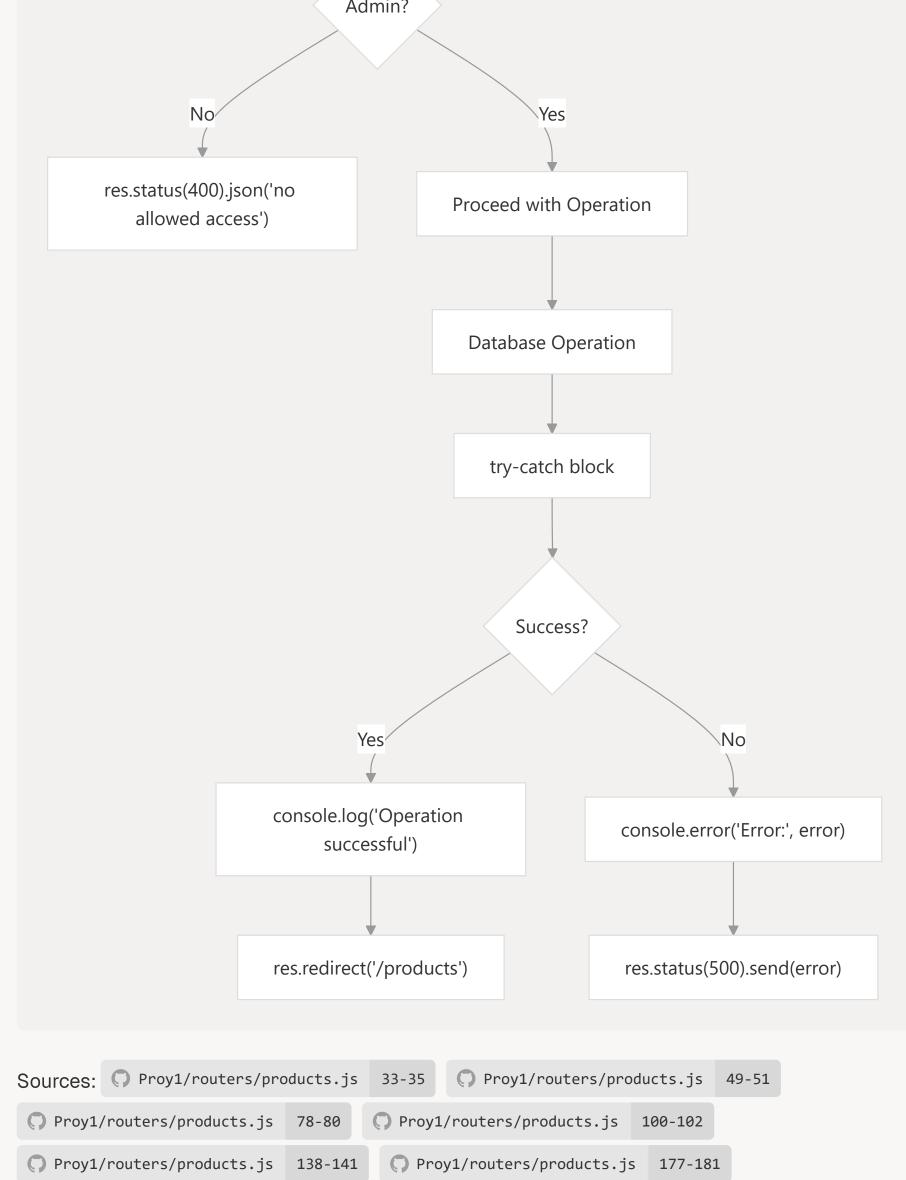
The templates implement conditional rendering based on session admin status:

# Sources: Proy1/views/products.ejs 44-62 Proy1/views/products\_edit.ejs 30-54

Name, Price, Stock Display

- **Error Handling and Response Patterns** The system implements consistent error handling across all operations with appropriate HTTP status codes and user feedback.
- Authorization 400 Failure Successful 302

Form Display	200	res.render(t	template, data)	None			
Error Handlir	ng Flow						
		Authorization (	Check				
		Admin?					
Admini							
	No		Yes				
roc cto	tus(400) iso	o('no	4				



Sources: Proy1/routers/products.js 1-190 **Authorization and Security Model** require valid admin tokens. This security layer protects all modification operations while allowing public read access.

**Security Implementation Diagram** admin == true? Yes No **Allowed Operations** 400: no allowed access **GET/POST** POST /products/new GET /products/delete/:id /products/edit/:id express-validator checks **SQLite Operations** res.redirect('/products') Sources: Proy1/routers/products.js 44-47 Proy1/routers/products.js 69-72 Proy1/routers/products.js 94-97 Proy1/routers/products.js 116-119 Proy1/routers/products.js 156-159 **Data Processing and Validation** 

Not empty, integer, minimum 1 price stock **Price Handling** 

**Validation Rules** 

Field

Proy1/routers/products.js 75-76 **Database Operations** 

No

**Template Structure Template Purpose** Product listing products.ejs

**Dynamic UI Authorization** 

**UI Component Structure** 

partials/header.ejs

products\_new.ejs

products\_edit.ejs

<%- locals?.session?.admin == true ? '' : 'disabled' -%> Edit button enablement Delete button enablement

products.ejs

Product Card Grid

Individual Product Cards

/products/edit/:id

Edit/Delete Buttons

Creation form

Edit form

Proy1/views/products.ejs 50-59

Status Scenario Code Database Error 500

**Error Response Patterns** 

Operation