# Product Management System

Relevant source files

This document details the product CRUD operations, routing, and business logic implementation for Project 1 (Proy1). The system provides a complete product inventory management interface with role-based access control and form validation. For database configuration and schema details, see Database & Configuration. For development setup information, see Development Environment.

## Route Structure and CRUD Operations

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 `products.js` router module.

### CRUD Route Mapping

| Operation | Method | Route | Purpose | Admin Required |
|---|---|---|---|---|
| Read All | GET | `/products/` | List all products | No |
| 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


Sources: Proy1/routers/products.js 1-198

## Authorization and Security Model

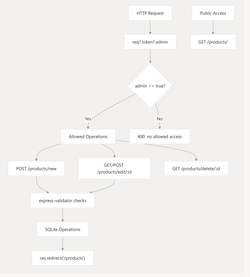
The system implements a token-based authorization model where administrative operations require valid admin tokens. This security layer protects all modification operations while allowing public read access.

### Admin Authorization Check

All destructive operations (Create, Update, Delete) implement the same authorization pattern:

```
if(req?.token?.admin != true){
    res.status(400).json("no allowed access, need to be admin")
    return
}
```

### Security Implementation Diagram




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

The system implements comprehensive input validation using `express-validator` middleware and handles price normalization for storage.

### Validation Rules

| Field | Validation Rules | Implementation |
|---|---|---|
| `name` | Not empty, trimmed, ASCII only | `query('name').notEmpty().trim().isAscii()` |
| `description` | Not empty, ASCII only | `query('description').notEmpty().isAscii()` |
| `price` | Not empty, integer, minimum 1 | `query('price').notEmpty().isInt({min:1})` |
| `stock` | Not empty, integer, minimum 1 | `query('stock').notEmpty().isInt({min:1})` |

### Price Handling

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 57-68 Proy1/routers/products.js 147-158 Proy1/routers/products.js 75-76 Proy1/routers/products.js 164

## Database Operations

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

### SQL Operations by Function

| Operation | SQL Statement | Error Handling |
|---|---|---|
| List Products | `SELECT * FROM products` | Promise rejection with 500 status |
| Create Product | `INSERT INTO products (name, description, price, stock) VALUES (?, ?, ?, ?)` | Try-catch with 500 status |
| Read Single | `SELECT * FROM products WHERE id = ?` | Promise rejection with 500 status |
| Update Product | `UPDATE products SET name = ?, description = ?, price = ?, stock = ? WHERE id = ?` | Promise rejection with 500 status |
| Delete Product | `DELETE FROM products WHERE id = ?` | Try-catch with 500 status |

### Database Interaction Pattern


Sources: Proy1/routers/products.js 18-28 Proy1/routers/products.js 77-78 Proy1/routers/products.js 121-132 Proy1/routers/products.js 161-173 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.

### Template Structure

| Template | Purpose | Key Features |
|---|---|---|
| `products.ejs` | Product listing | Card-based layout, conditional admin buttons |
| `products_new.ejs` | Creation form | Full product form, admin-only access |
| `products_edit.ejs` | Edit form | Pre-populated form, admin-only access |

### Dynamic UI Authorization

The templates implement conditional rendering based on session admin status:

```
<%- local?.session?.admin == true ? '' : 'disabled' %>
```

This pattern is used for:

- Add Product button enablement
- Edit button enablement
- Delete button enablement

### UI Component Structure


Sources: Proy1/views/products.ejs 44-62 Proy1/views/products_new.ejs 38-54 Proy1/views/products_edit.ejs 38-54 Proy1/views/products.ejs 34-38 Proy1/views/products.ejs 58-59

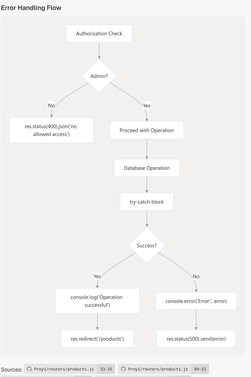
## Error Handling and Response Patterns

The system implements consistent error handling across all operations with appropriate HTTP status codes and user feedback.

### Error Response Patterns

| Scenario | Status Code | Response Type | Redirect Behavior |
|---|---|---|---|
| Database Error | 500 | `res.status(500).send(error)` | None |
| Authorization Failure | 400 | `res.status(400).json("no allowed access, need to be admin")` | None |
| Successful Operation | 302 | `res.redirect('/products')` | Returns to product list |
| Form Display | 200 | `res.render(template, data)` | None |

### Error Handling Flow




Sources: Proy1/routers/products.js 33-35 Proy1/routers/products.js 49-51 Proy1/routers/products.js 78-80 Proy1/routers/products.js 100-102 Proy1/routers/products.js 138-141 Proy1/routers/products.js 177-181