# Technical Specifications: Inventory Management System API

**Project:** Inventory Management System

**Document Version:** 1.0.0

**Status:** Baseline

**Base URL:** https://localhost:{PORT}/api

**Description:** This document outlines the requirements for the RESTful API backend, facilitating a decoupled architecture between the ASP.NET Core Web API and the jQuery/Bootstrap frontend.

## 1. Architectural Overview

The system utilizes an **N-Tier Architecture** with a decoupled **Data Access Layer (DAL)**. This ensures high maintainability, testability, and clear separation of concerns.

### 1.1 Logical Layers

1. **Presentation Layer (Controllers):** Handles HTTP routing, request parsing, and response status codes.
2. **Business Logic Layer (Services):** Orchestrates domain logic, data transformation, and validation.
3. **Data Access Layer (DAL):** Manages data persistence using the Repository Pattern.
4. **Entities/Models:** Defines the structure of the data across all layers.

## 2. Domain Models & Data Contracts

### 2.1 Product Entity

This model defines the core structure of a product in the system.

| **Property** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| Id | int | Primary Key, Auto-increment | Unique identifier for the product. |
| Name | string | Required, Max Length 100 | Name of the item. |
| Price | decimal | Required, Value > 0 | Price in decimal format. |
| Category | string | Required | Product category grouping. |

### 2.2 Serialization Standards

* **Media Type:** application/json
* **Naming Convention:** All JSON outputs **MUST** follow camelCase (e.g., productId, productName).
* **DateTime Format:** ISO 8601 (YYYY-MM-DDTHH:mm:ssZ).

## 3. Data Access Layer (DAL)

The Repository Pattern is required to abstract the persistence mechanism.

### 3.1 IProductRepository Interface

public interface IProductRepository {  
 IEnumerable<Product> GetAll();  
 Product GetById(int id);  
 void Add(Product entity);  
 void Update(Product entity);  
 void Delete(int id);  
}

## 4. API Endpoints (REST Specification)

### 4.1 Product Controller (/api/products)

| **Method** | **Endpoint** | **Description** | **Request Body** | **Success** | **Error Codes** |
| --- | --- | --- | --- | --- | --- |
| GET | /products | Fetch all products. | None | 200 OK | 500 |
| GET | /products/{id} | Fetch a single product. | None | 200 OK | 404 |
| POST | /products | Create a new product. | Product Object | 201 | 400 |
| PUT | /products/{id} | Replace/Update product. | Product Object | 204 | 400, 404 |
| DELETE | /products/{id} | Remove product from database. | None | 200 OK | 404 |

### 4.2 Authentication Controller (/api/auth)

| **Method** | **Endpoint** | **Description** | **Request Body** | **Success** |
| --- | --- | --- | --- | --- |
| POST | /auth/login | Authenticate user via JWT. | LoginDTO | 200 OK |

## 5. Security & Middleware Configuration

### 5.1 Cross-Origin Resource Sharing (CORS)

The API must be configured to allow traffic from the UI host to prevent "Same-Origin Policy" blocks.

* **Allowed Origins:** http://localhost:{UI\_PORT}
* **Allowed Methods:** GET, POST, PUT, DELETE, OPTIONS.
* **Allowed Headers:** Content-Type, Authorization.

### 5.2 Dependency Injection (DI)

Registration in Program.cs is mandatory for the architecture to function:

* IProductRepository: **Singleton** (for in-memory development) or **Scoped** (for Database).
* ProductService: **Scoped**.

### 5.3 Authentication

Endpoints within the Product Controller must be secured using the [Authorize] attribute. Clients are expected to send an **Authorization Header** as follows:

Authorization: Bearer <JWT\_TOKEN>

## 6. Error Handling Standards

All API errors must return a consistent status code and, where possible, a JSON error object:

* **400 Bad Request:** Validation errors (e.g., negative price).
* **401 Unauthorized:** Missing or expired token.
* **404 Not Found:** Resource not found in the repository.
* **500 Internal Server Error:** Server-side logic failure.

## 7. Implementation Roadmap

1. **Model Definition:** Implement Product POCO class.
2. **DAL Implementation:** Create IProductRepository and ProductRepository with a static List<Product>.
3. **Service Layer:** Implement ProductService to handle business rules.
4. **Controller Setup:** Implement ProductsController injecting the service layer.
5. **Global Config:** Configure CORS, DI, and JSON camelCase options in Program.cs.
6. **Integration Testing:** Verify endpoints via Swagger before linking to the jQuery UI.









