Practical Examination 1: Building a CRUD Application with Node.js and Database Integration

Objective:

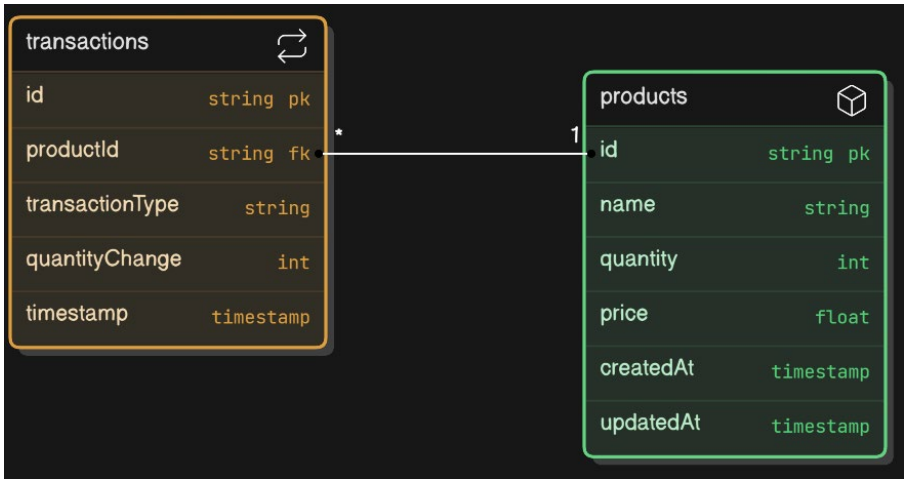
Develop a backend application that serves as an inventory management system, incorporating a database integration and asynchronous handling of operations. This examination focuses on content from lectures 1 to 7.

Requirements for Pass:

1. **Backend Development:** Create a backend application using Node.js that handles basic inventory items (e.g., products). The system should allow CRUD (Create) operations for the inventory.

2. **HTTP Server:** Develop an HTTP server using Node.js to manage inventory items.

3. **Database Integration:** Connect the application to a SQL-based database (e.g., SQLite or MySQL). Develop a simple database schema to store product information, such as name, quantity, and price.



4. **Asynchronous Operations:** Use async/await to handle database interactions asynchronously to ensure smooth performance.

5. **Basic Security Considerations:** Implement input validation to prevent SQL injections and sanitize user inputs.

Requirements for Pass with Distinction:

1. **Error Handling:** Include comprehensive error handling for all endpoints, returning appropriate HTTP status codes and descriptive error messages.

2. **Advanced Endpoint:** Develop an endpoint to retrieve products filtered by supplier (e.g., all products supplied by the “Samsung” supplier or products with quantity below a certain threshold).

API Endpoints for Practical Examination 1

The following are example API endpoints that can be implemented to retrieve product details, all products, a product's supplier, and a specific product's inventory for the inventory management system.

1. Retrieve a Product

Endpoint: GET /api/products/{product_id}

Description: Retrieves information about a specific product, identified by its product_id.

Example Response:

```
{
  'product_id': 1,
  'product_name': 'Sample Product',
  'description': 'This is a sample product description',
  'price': 100.00
}
```

2. Retrieve All Products

Endpoint: GET /api/products

Description: Retrieves a list of all products available in the inventory.

Example Response:

```
[
  {
    'product_id': 1,
    'product_name': 'Sample Product',
    'description': 'This is a sample product description',
    'price': 100.00
  },
  {
    'product_id': 2,
    'product_name': 'Another Product',
    'description': 'Another description',
    'price': 150.00
  }
]
```

3. Retrieve a Product's Supplier

Endpoint: GET /api/products/{product_id}/supplier

Description: Retrieves the supplier details for a specific product.

Example Response:

```
{
  'supplier_id': 1,
  'supplier_name': 'Supplier Name',
  'contact_info': 'contact@example.com'
}
```

4. Retrieve a Product's Inventory

Endpoint: GET /api/products/{product_id}/inventory

Description: Retrieves the inventory details for a specific product, including quantity and last updated timestamp.

Example Response:

```
{
  'product_id': 1,
  'supplier_id': 1,
  'quantity': 50,
  'last_updated': '2024-01-01 12:00:00'
}
```

Advanced API Endpoints for Pass with Distinction

1. Retrieve Products by Supplier

Endpoint: GET /api/products?supplier_id={supplier_id}

Description: Retrieves all products supplied by a specific supplier, identified by supplier_id.

Example Response:

```
[
  {
    'product_id': 1,
    'product_name': 'Product A',
    'description': 'Product A description',
    'price': 100.00
  },
  {
    'product_id': 3,
    'product_name': 'Product C',
    'description': 'Product C description',
    'price': 200.00
  }
]
```

2. Retrieve Products by Inventory Quantity

Endpoint: GET /api/products?min_quantity={min_quantity}

Description: Retrieves all products where the inventory quantity is below a specified minimum quantity (min_quantity).

Example Response:

```
[
  {
    'product_id': 2,
    'product_name': 'Product B',
    'description': 'Product B description',
    'price': 150.00,
    'quantity': 10
  }
]
```