NAME : CHANDAN H S

CLASS : C SECTION

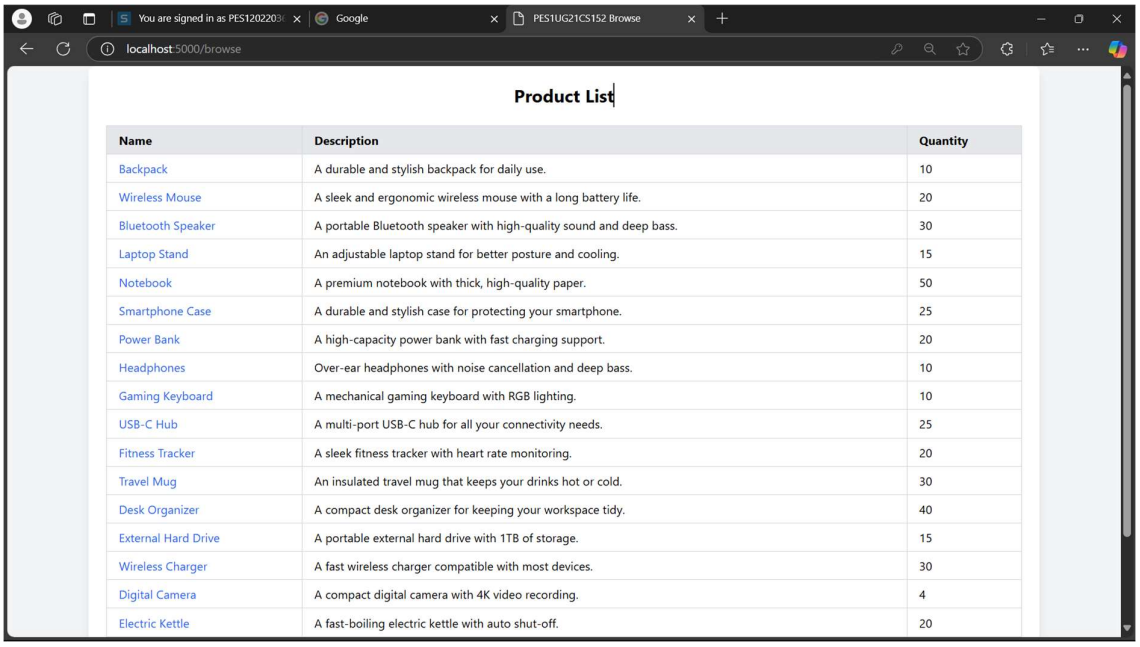
SRN : PES1UG22CS152

Lab : CC_expt3

Github repository link : https://github.com/Chandanhssuresh/CC_LAB_expt3

1)

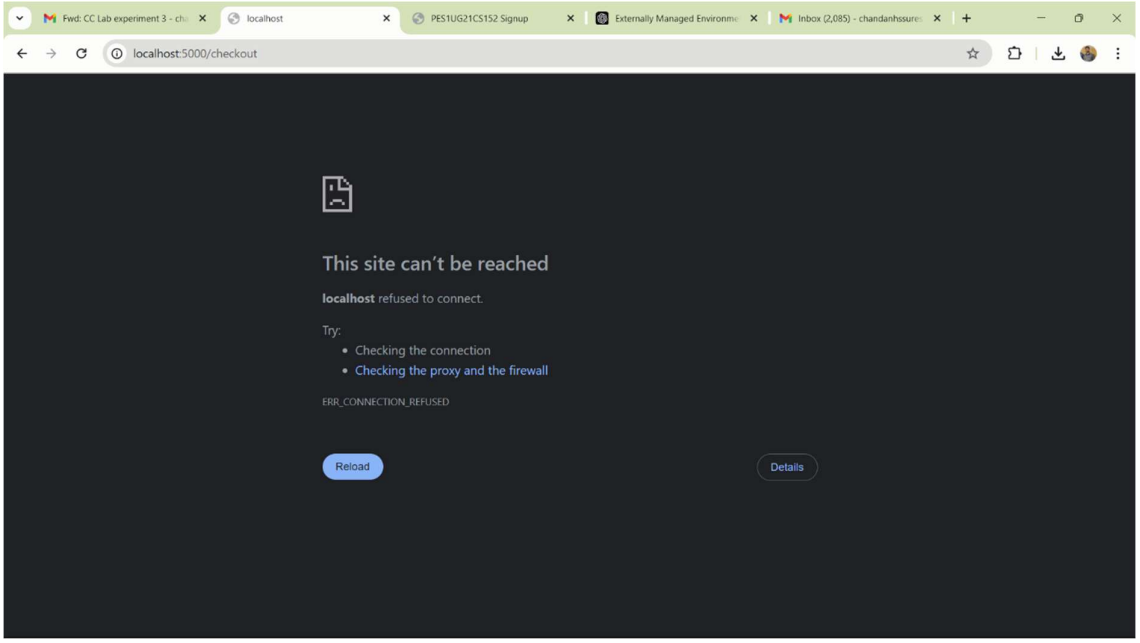
SS1 :



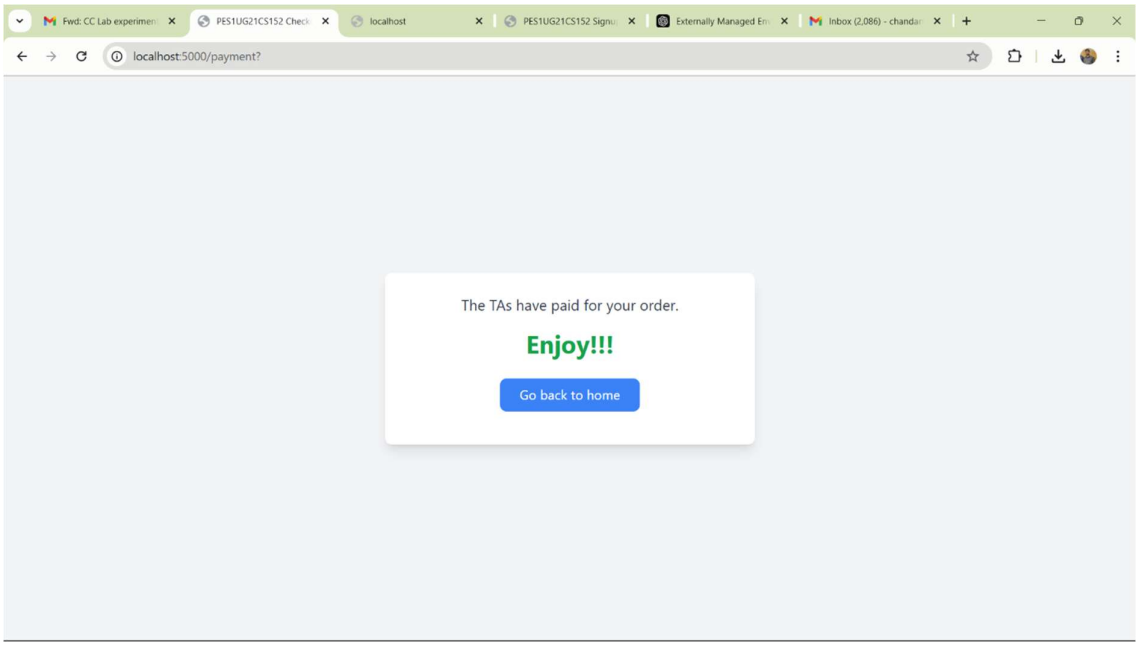
Product List

Name	Description	Quantity
Backpack	A durable and stylish backpack for daily use.	10
Wireless Mouse	A sleek and ergonomic wireless mouse with a long battery life.	20
Bluetooth Speaker	A portable Bluetooth speaker with high-quality sound and deep bass.	30
Laptop Stand	An adjustable laptop stand for better posture and cooling.	15
Notebook	A premium notebook with thick, high-quality paper.	50
Smartphone Case	A durable and stylish case for protecting your smartphone.	25
Power Bank	A high-capacity power bank with fast charging support.	20
Headphones	Over-ear headphones with noise cancellation and deep bass.	10
Gaming Keyboard	A mechanical gaming keyboard with RGB lighting.	10
USB-C Hub	A multi-port USB-C hub for all your connectivity needs.	25
Fitness Tracker	A sleek fitness tracker with heart rate monitoring.	20
Travel Mug	An insulated travel mug that keeps your drinks hot or cold.	30
Desk Organizer	A compact desk organizer for keeping your workspace tidy.	40
External Hard Drive	A portable external hard drive with 1TB of storage.	15
Wireless Charger	A fast wireless charger compatible with most devices.	30
Digital Camera	A compact digital camera with 4K video recording.	4
Electric Kettle	A fast-boiling electric kettle with auto shut-off.	20

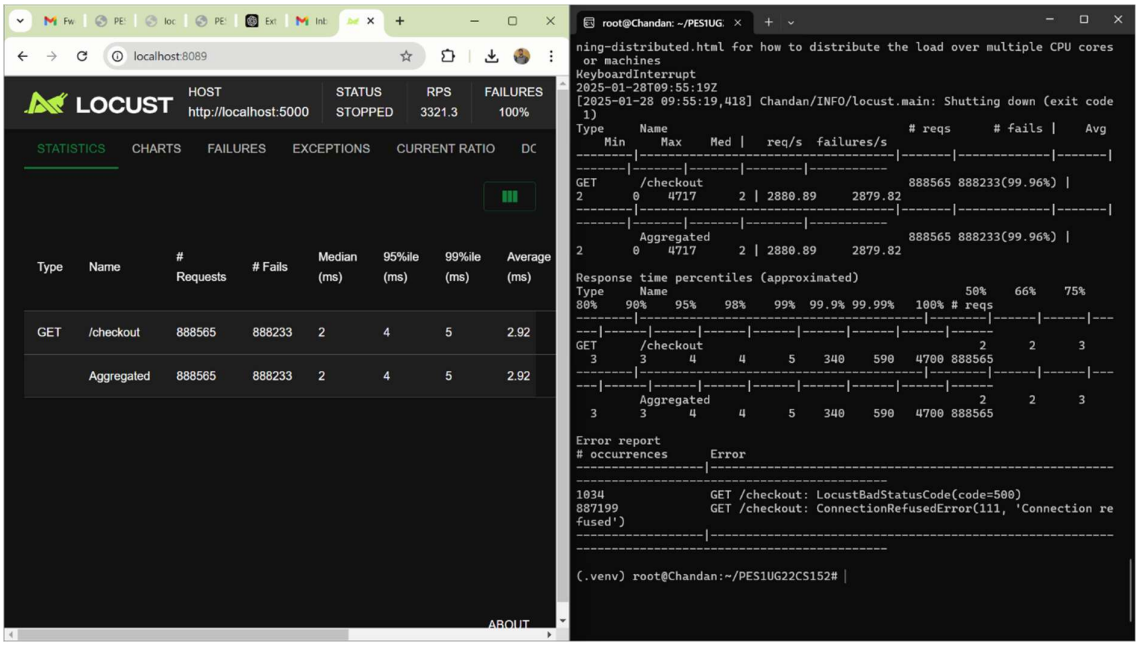
SS2:


localhost:5000/checkout
This site can't be reached
localhost refused to connect.
Try:
Checking the connection
Checking the proxy and the firewall
ERR_CONNECTION_REFUSED
Reload
Details

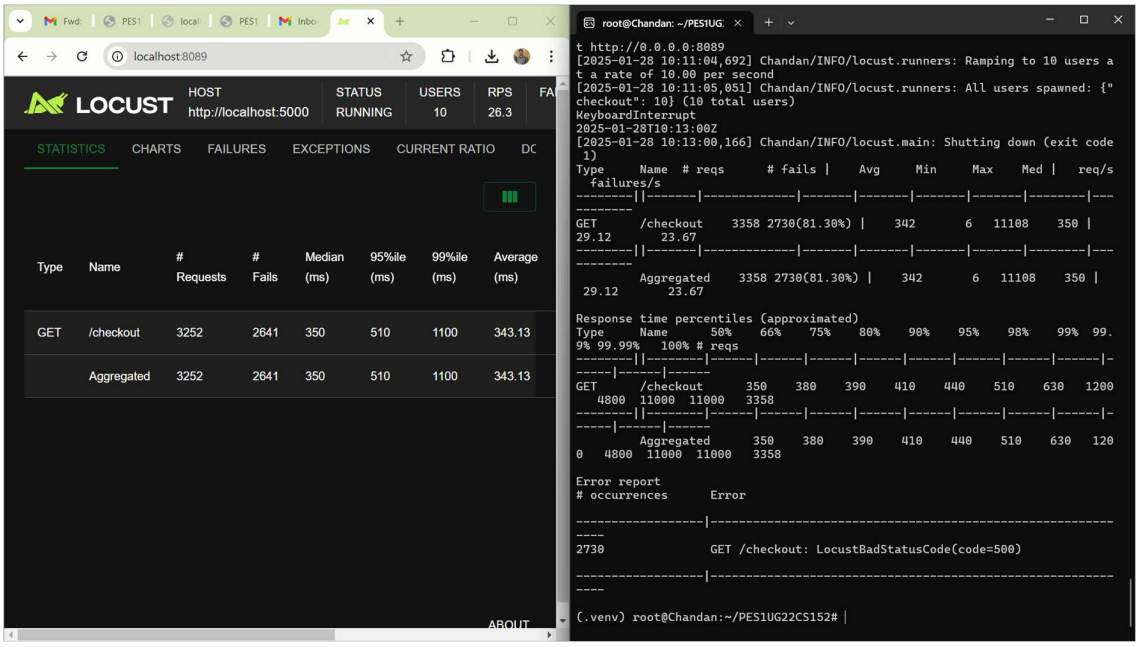
SS3 :


localhost:5000/payment?
The TAs have paid for your order.
Enjoy!!!
Go back to home

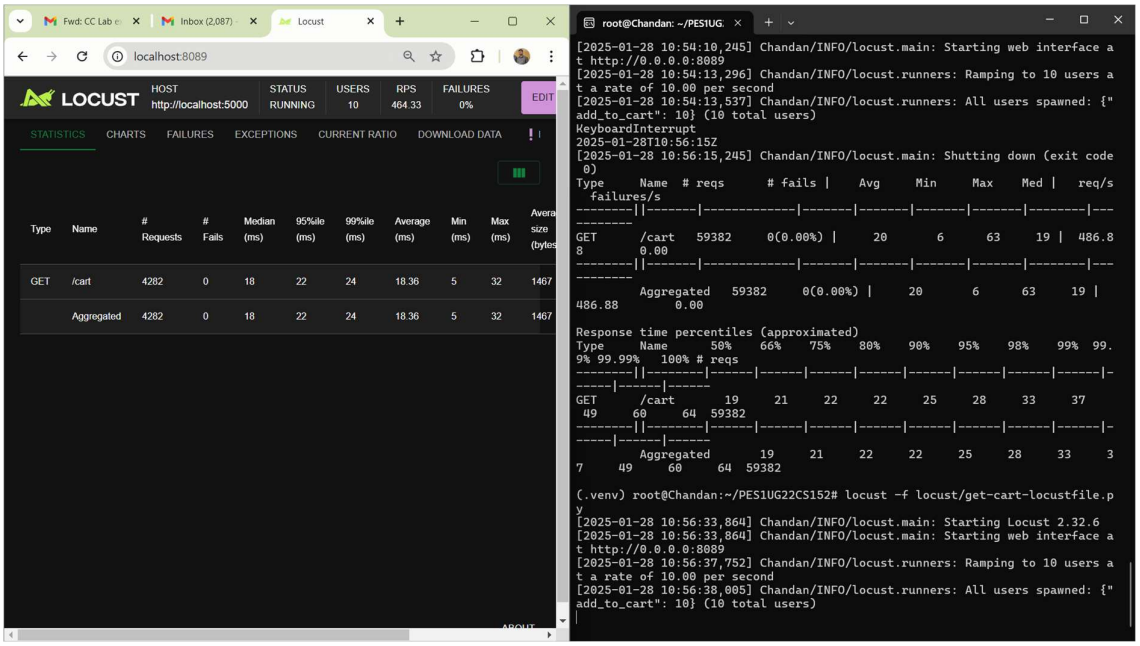
SS4 :(Before Optimization)



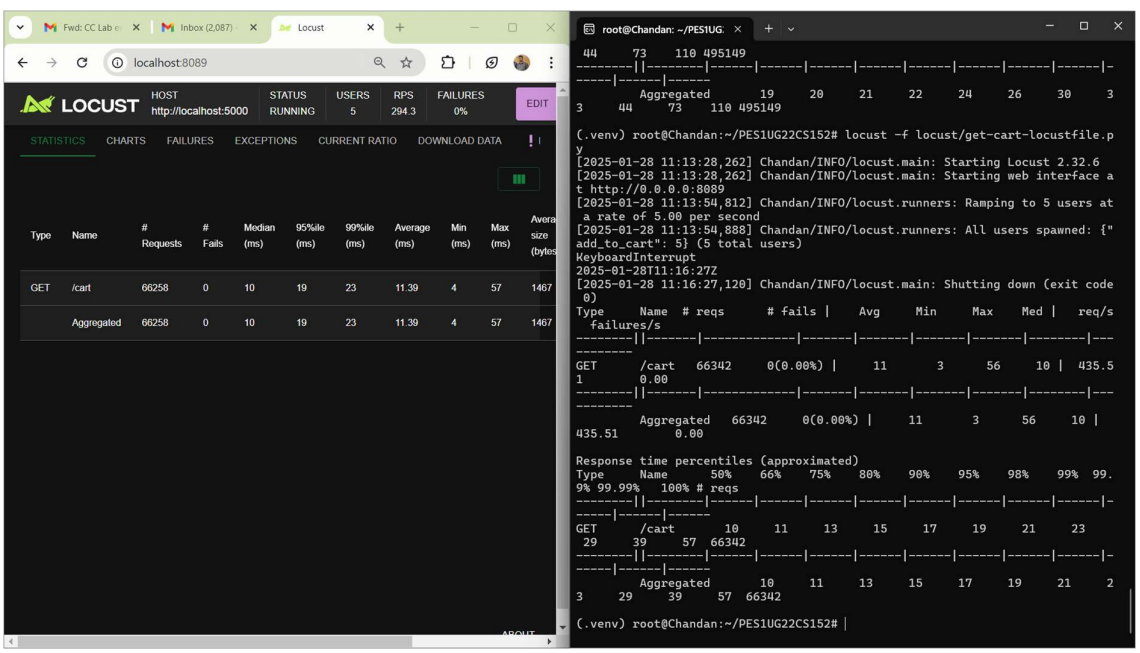
SS5 : (After optimization)



SS6 : (Before optimization)



SS7 : (After optimization)

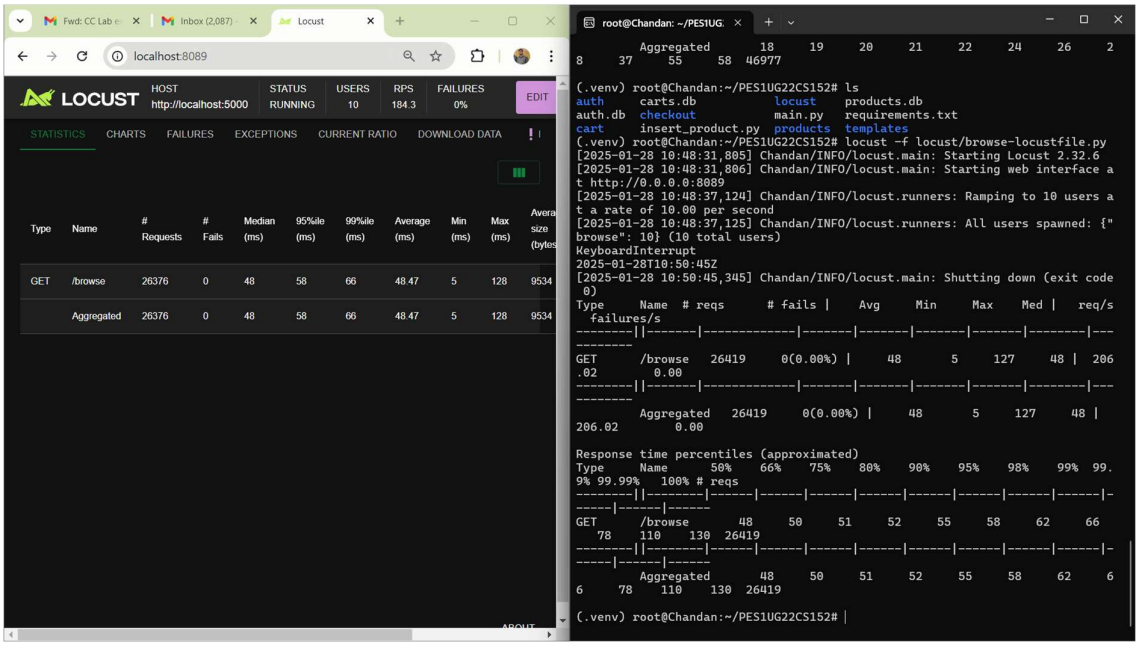


Optimized Code :

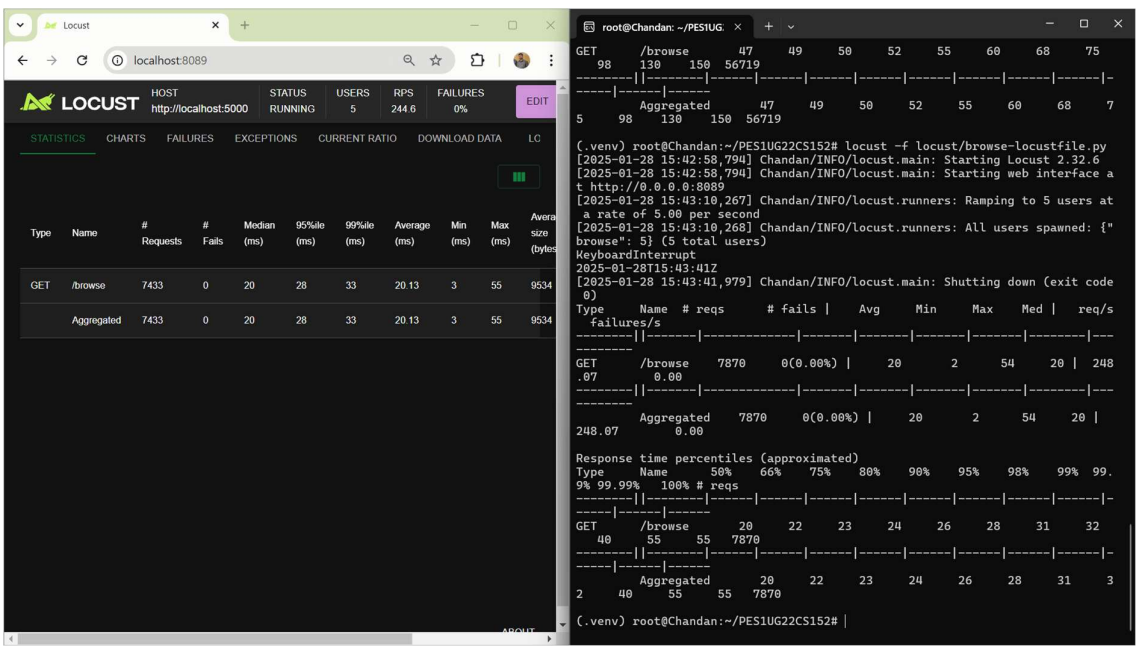
```python
import json

import json
import products
from cart import dao
from products import Product


class Cart:
    def __init__(self, id: int, username: str, contents: list[Product], cost: float):
        self.id = id
        self.username = username
        self.contents = contents
        self.cost = cost

    @staticmethod
    def load(data):
        # Parse contents safely
        contents = json.loads(data['contents']) if isinstance(data['contents'], str) else data['contents']
        return Cart(data['id'], data['username'], contents, data['cost'])


def get_cart(username: str) -> Cart:
    """
    Retrieve the cart for a given username.

    :param username: The username of the cart owner.
    :return: A Cart object containing products and total cost.
    """
    cart_details = dao.get_cart(username)
    if not cart_details:
        return Cart(id=0, username=username, contents=[], cost=0.0)

    # Extract all product IDs from the cart
    product_ids = []
    for cart_detail in cart_details:
        try:
            # Safely parse contents using json.loads
            contents = json.loads(cart_detail['contents'])
            product_ids.extend(contents)
```

The optimized code enhances **performance** by replacing redundant loops with efficient list comprehensions for product retrieval and using batch operations to minimize database calls. It improves **robustness** with added error handling for invalid inputs and missing products, ensuring the application fails gracefully. Input validation ensures data integrity, making the code **secure** and less prone to errors. Finally, **readability** is improved with streamlined logic and proper documentation, making the code more maintainable.

SS8 : (Before optimization)



SS9 : (After optimization)



Optimized Code :

```python
from products import dao


class Product:
    def __init__(self, id: int, name: str, description: str, cost: float, q
        """
        Represents a product with an ID, name, description, cost, and quant
        """
        self.id = id
        self.name = name
        self.description = description
        self.cost = cost
        self.qty = qty

    @staticmethod
    def load(data: dict) -> "Product":
        """
        Create a Product instance from a dictionary.
        """
        return Product(
            id=data['id'],
            name=data['name'],
            description=data['description'],
            cost=data['cost'],
            qty=data['qty']
        )


def list_products() -> list[Product]:
    """
    Fetch all products from the database and return them as a list of Produ
    """
    # Use list comprehension for concise and efficient mapping
    return [Product.load(product) for product in dao.list_products()]


def get_product(product_id: int) -> Product:
    """
    Fetch a single product by ID and return it as a Product object.
```

```python
def add_product(product: dict):
    """
    Add a new product to the database.
    :param product: A dictionary representing the product data.
    """
    # Validate required keys before adding the product
    required_keys = {'id', 'name', 'description', 'cost', 'qty'}
    if not required_keys.issubset(product.keys()):
        raise ValueError(f"Product data is missing required fields: {requir

    dao.add_product(product)


def update_qty(product_id: int, qty: int):
    """
    Update the quantity of a product in the database.
    :param product_id: The ID of the product to update.
    :param qty: The new quantity value (must be non-negative).
    """
    if qty < 0:
        raise ValueError('Quantity cannot be negative')

    # Ensure the product exists before updating
    product_data = dao.get_product(product_id)
    if not product_data:
        raise ValueError(f"Product with ID {product_id} not found.")

    dao.update_qty(product_id, qty)
```

The optimized code enhances **performance** by replacing redundant loops with efficient list comprehensions for product retrieval and using batch operations to minimize database calls. It improves **robustness** with added error handling for invalid inputs and missing products, ensuring the application fails gracefully. Input validation ensures data integrity, making the code **secure** and less prone to errors. Finally, **readability** is improved with streamlined logic and proper documentation, making the code more maintainable.