Validate and operate product details

This document outlines the process of taking the user details, validating the details subsequently inserting , updating and deleting the product details

CONSTANTS.PY

Added a variable to store the product details

```
#----------------PRODUCT OPERATION's---------
PRODUCT_DETAILS = {}
```

VALIDATE_PRODUCT_UTILS.PY

Logging module is imported for storing logs and PRODUCT_DETAILS is imported for checking the product details

```
from logger import *
from constants import PRODUCT_DETAILS
```

is_valid_product_id function takes product_id as argument and check's if it is valid or not and after checking the validity, it check's if that product_id already present's in the existing details and then return's TRUE if all the mentioned conditions are satisfied, else ValueError is raised

```
def is_valid_product_id(product_id):
    '''
    This function validates if the product ID is valid
    param : product_id : int
    return : bool
    '''
    if isinstance(product_id, int) and product_id > 0:
        if product_id not in PRODUCT_DETAILS['product_id']:
            logging.debug(f'Product ID verified successfully: {product_id}')
            return True
        else:
            logging.debug(f"product id already exists {PRODUCT_DETAILS['product_id']}'
    else:
        logging.error(f'Invalid Product ID: {product_id}')
        raise ValueError(f'Invalid Product ID: {product_id}')
```

is_valid_product_name function takes name as an argument and check's the name is string or not and the length is between 3 to 50 characters, returns TRUE if all the conditions satisfied else raise ValueError

```
def is_valid_product_name(name):
```

```
    '''
    This function validates if the product name is valid
    param : name : str
    return : bool
    '''
    if isinstance(name, str) and 3 <= len(name.strip()) <= 50:
        logging.info(f'Product name verified successfully: {name}')
        return True
    else:
        logging.error(f'Invalid Product name: {name}')
        raise ValueError(f'Invalid Product name: {name}')
```

is_valid_product_price function takes price as an argument and checks if the price if the it's type is in int or float and check's if it is greater than zero and if the conditions are met then returns TRUE. else it will raise ValueError

```
def is_valid_product_price(price):
    '''
    This function validates if the product price is valid
    param : price : int or float
    return : bool
    '''
    if isinstance(price, (int, float)) and price >= 0:
        logging.info(f'Product price verified successfully: {price}')
        return True
    else:
        logging.error(f'Invalid Product price: {price}')
        raise ValueError(f'Invalid Product price: {price}')
```

is_valid_product_description function takes description as an argument and check's if the argument type is string or not and also check if it is not empty string and returns TRUE, else it will return ValueError

```
def is_valid_product_description(description):
    '''
    This function validates if the product description is valid
    param : description : str
    return : bool
    '''
    if isinstance(description, str) and len(description.strip()) > 0:
        logging.info(f'Product description verified successfully: {description}')
        return True
    else:
        logging.error(f'Invalid Product description: {description}')
        raise ValueError(f'Invalid Product description: {description}')
```

is_valid_product_specifications function takes specifications argument as a dictionary and check's both the key and value is in string format or not and also checks the specifications is empty or not and returns TRUE if the condition's meet else it will return ValueError

```
def is_valid_product_specifications(specifications):
    '''
    This function validates if the product specifications are valid
    param : specifications : str
    return : bool
    '''
    if isinstance(specifications, dict) and len(specifications) > 0:
        for key, value in specifications.items():
            if not isinstance(key, str) or not isinstance(value, str):
                logging.error(f'Invalid specification key or value: {key}: {value}')
                raise ValueError(f'Invalid specification key or value: {key}: {value}'
        logging.info(f'Product specifications verified successfully: {specifications}'
        return True
    else:
        logging.error(f'Invalid Product specifications: {specifications}')
        raise ValueError(f'Invalid Product specifications: {specifications}')
```

PRODUCT_DETAILS.PY

Required modules are imported

```
from logger import *
from validate_product_utils import *
from constants import PRODUCT_DETAILS
```

class **product_operations** is defined and a constructor is initiated for storing the constants

```
class product_operations():
    '''
    This class insert's, update's and delete's the product information based on user r
    '''
    def __init__(self):
        self.product_details = PRODUCT_DETAILS
```

insert_product method is defined which takes product_details as an argument which is dict type and separate variables are defined for storing the product details which are sent to the functions based on the category and validated. After successful validation, A dict with the key as product_id and value which consists remaining details as another dict if formed and returns TRUE. if any of the above validation fails then it return's ValueError and error is captured through the exception handling

```
def insert_product(self, product_details):
        '''
        This function inserts a new product into the product database
        param : product_details : dict
        return : bool
        '''
        logging.info("\nProduct Insertion started...\n")
        try:
```

```
            product_id = product_details.get('product_id')
            name = product_details.get('name')
            price = product_details.get('price')
            description = product_details.get('description')
            specifications = product_details.get('specifications')

            if (is_valid_product_id(product_id) and is_valid_product_name(name) and is
                and is_valid_product_description(description) and is_valid_product_spe
                if product_id not in self.product_details:
                    self.product_details[product_id] = {
                        'name': name,
                        'price': price,
                        'description': description,
                        'specifications': specifications
                    }
                    logging.debug(f'Product inserted successfully: {self.product_detai
                    return True
                else:
                    logging.error(f'Product ID already exists: {self.product_details}'
                    raise ValueError(f'Product ID already exists: {self.product_detail
        except Exception as err:
            logging.error(f'Error while insert :- {err} ')
```

update_product method defined for updating the details of the existing product. In which it takes the product_id and dict as argument and then separate's each and every key and check's for individual key's and updates the existing product if the key has value.
if the product_id doe's not exist in the product_details then ValueError is raised and error is captured through the log file.

```
    def update_product(product_id, update_details):
        '''
        This function updates an existing product in the product database
        param1 : product_id : int
        param2 : update_details : dict
        return : bool
        '''
        logging.info("\nProduct details updating...\n")
        try:
            if is_valid_product_id(product_id):
                if product_id in self.product_details:
                    name = update_details.get('name')
                    price = update_details.get('price')
                    description = update_details.get('description')
                    specifications = update_details.get('specifications')

                    if name is not None and is_valid_product_name(name):
                        self.product_details[product_id]['name'] = name
                    if price is not None and is_valid_product_price(price):
                        self.product_details[product_id]['price'] = price
                    if description is not None and is_valid_product_description(descri
                        self.product_details[product_id]['description'] = description
                    if specifications is not None and is_valid_product_specifications(
                        self.product_details[product_id]['specifications'] = specifica
```

```
                    logging.info(f'Product updated successfully: {self.product_details
                    return True
                else:
                    logging.error(f'Product ID does not exist: {product_id}')
                    raise ValueError(f'Product ID does not exist: {product_id}')
        except Exception as err:
            logging.error(f'Error while update :- {err} ')
```

delete_product method defined for deleting the existing record's which takes only the product_id as argument which will be in int format if the product_id is already present in the existing record's then the record will be deleted

```
    def delete_product(product_id):
        '''
        This function deletes an existing product from the product database
        param : product_id : int
        return : bool
        '''
        try:
            if is_valid_product_id(product_id):
                if product_id in self.product_details:
                    del self.product_details[product_id]
                    logging.info(f'Product deleted successfully: {self.product_details
                    return True
                else:
                    logging.error(f'Product ID does not exist: {self.product_details}'
                    raise ValueError(f'Product ID does not exist: {self.product_detail
        except Exception as err:
            logging.error(f'Error while deletion :- {err} ')
```

argument's are stored in a variable to pass to the method's and we can send the dictionary directly while calling the method with an object.

```
new_product = {
    'product_id': 1,
    'name': "T-Shirt",
    'price': '99.99',
    'description': "This is a good product",
    'specifications': {"color": "red", "size": "M"}
    }

updated_details = {
    'price': 109.99,
    'description': "It is a fantastic one"
    }

obj = product_operations()
obj.insert_product(new_product)
```

LOGGER.PY

Logger file stores the log's based on the systime and path is specified in the file for storing the file and it is set to append mode, so that the new log's will be appended to the old log's

```
import logging

logging.basicConfig(
    filename = "c:/Users/w125682/Downloads/practice.log",
    level = logging.DEBUG,
    format = '%(asctime)s - %(levelname)s - %(message)s',
    filemode = 'a'
    )
```

LOGS

2024-06-26 20:31:59,958 - INFO -
Product Insertion started...

2024-06-26 20:31:59,958 - DEBUG - Product ID verified successfully: 1
2024-06-26 20:31:59,958 - DEBUG - Product name verified successfully: T-Shirt
2024-06-26 20:31:59,958 - DEBUG - Product price verified successfully: 99.99
2024-06-26 20:31:59,958 - DEBUG - Product description verified successfully: This is a good product
2024-06-26 20:31:59,958 - DEBUG - Product specifications verified successfully: {'color': 'red', 'size': 'M'}
2024-06-26 20:31:59,958 - DEBUG - Product inserted successfully: {'name': 'T-Shirt', 'price': 99.99, 'description': 'This is a good product', 'specifications': {'color': 'red', 'size': 'M'}}
2024-06-26 20:31:59,958 - INFO -
Product details updating...

2024-06-26 20:31:59,958 - DEBUG - Product ID verified successfully: 1
2024-06-26 20:31:59,959 - DEBUG - Product price verified successfully: 109.99
2024-06-26 20:31:59,959 - DEBUG - Product description verified successfully: It is a fantastic one
2024-06-26 20:31:59,959 - INFO - Product updated successfully: {'name': 'T-Shirt', 'price': 109.99, 'description': 'It is a fantastic one', 'specifications': {'color': 'red', 'size': 'M'}}