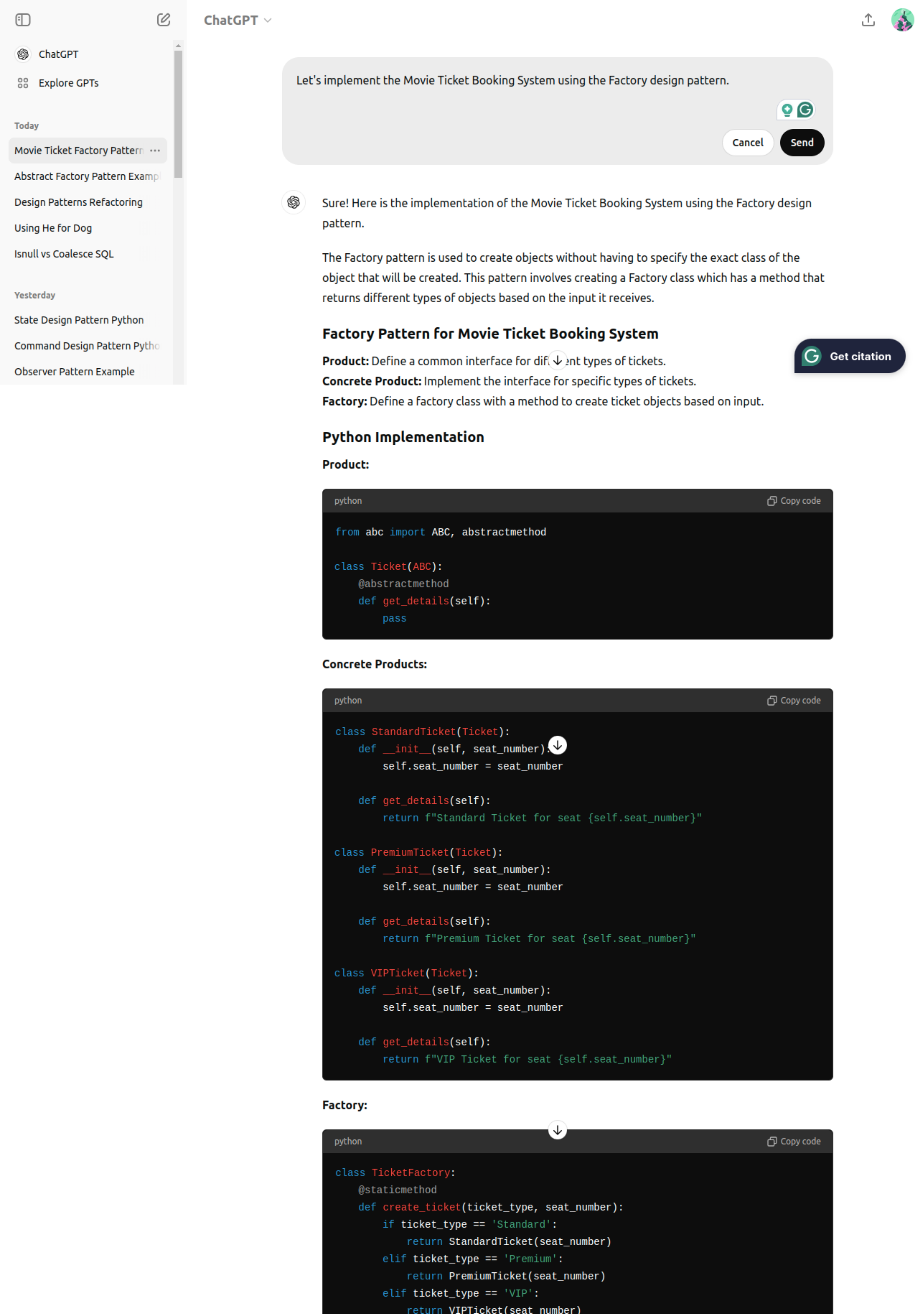
ChatGPT
Explore GPTs
Today
Movie Ticket Factory Pattern
Abstract Factory Pattern Examp
Design Patterns Refactoring
Using He for Dog
Isnull vs Coalesce SQL
Yesterday
State Design Pattern Python
Command Design Pattern Pytho
Observer Pattern Example
ChatGPT
Cancel
Send
Get citation
python
Copy code

Let's implement the Movie Ticket Booking System using the Factory design pattern.

Sure! Here is the implementation of the Movie Ticket Booking System using the Factory design pattern.

The Factory pattern is used to create objects without having to specify the exact class of the object that will be created. This pattern involves creating a Factory class which has a method that returns different types of objects based on the input it receives.

Factory Pattern for Movie Ticket Booking System

Product: Define a common interface for different types of tickets.
Concrete Product: Implement the interface for specific types of tickets.
Factory: Define a factory class with a method to create ticket objects based on input.

Python Implementation

Product:

```python
from abc import ABC, abstractmethod

class Ticket(ABC):
    @abstractmethod
    def get_details(self):
        pass
```

Concrete Products:

```python
class StandardTicket(Ticket):
    def __init__(self, seat_number):
        self.seat_number = seat_number

    def get_details(self):
        return f"Standard Ticket for seat {self.seat_number}"

class PremiumTicket(Ticket):
    def __init__(self, seat_number):
        self.seat_number = seat_number

    def get_details(self):
        return f"Premium Ticket for seat {self.seat_number}"

class VIPTicket(Ticket):
    def __init__(self, seat_number):
        self.seat_number = seat_number

    def get_details(self):
        return f"VIP Ticket for seat {self.seat_number}"
```

Factory:

```python
class TicketFactory:
    @staticmethod
    def create_ticket(ticket_type, seat_number):
        if ticket_type == 'Standard':
            return StandardTicket(seat_number)
        elif ticket_type == 'Premium':
            return PremiumTicket(seat_number)
        elif ticket_type == 'VIP':
            return VIPTicket(seat_number)
    else:
        raise ValueError(f"Unknown ticket type: {ticket_type}")
```

Client Code:

python Copy code

```python
def book_ticket(ticket_type, seat_number):
    ticket = TicketFactory.create_ticket(ticket_type, seat_number)
    print(ticket.get_details())

# Usage
book_ticket('Standard', 'A1')
book_ticket('Premium', 'B2')
book_ticket('VIP', 'C3')
```

Explanation

Product:

`Ticket` is an abstract class that declares the `get_details` method.

Concrete Product:

`StandardTicket`, `PremiumTicket`, and `VIPTicket` are concrete implementations of the `Ticket` abstract class, each with a specific seat number.

Factory:

`TicketFactory` is a factory class with a static method `create_ticket` that takes the ticket type and seat number as parameters and returns an instance of the appropriate ticket class.

Client:

The `book_ticket` function uses the `TicketFactory` to create a ticket of the specified type and seat number, and then prints the details of the ticket.

When you run this code, it will create and display details of different types of tickets based on the input provided. This demonstrates the use of the Factory pattern to create objects of different types without needing to specify the exact class of the object.

Upgrade plan
Get GPT-4, DALL·E, and more

Message ChatGPT

ChatGPT can make mistakes. Check important info.