Mock booking system

The task is to create a simple/mock booking system for the travel industry.

The intention of this kind of booking system is to allow the users to **Search** holiday options (example: hotel and flight) for a certain period of time (example: 10 to 15 of June) for a given destination (example: Barcelona). As a result of the search, the user receives multiple options from the system to choose from. When the user decides on a given option, make a request to **Book** it. Because the booking process usually takes longer to complete, the user should be able to know when the booking is completed and **Check the Status** of the booking (example: Pending, Success or Fail).

The task will consist of an API that will provide the necessary endpoints to make a booking as explained above.

The booking flow will contain 3 methods:

1. Search
2. Book
3. CheckStatus

Request and Response classes are provided at the end of this document.

Requirements:

- Visual Studio
- .NET Core
- Internet access
- No need for external storage, you can store objects in memory

When developing the solution please have in mind that this task is a small part of a large solution and you must think in advance for future extension and optimization.

Evaluation criteria:

- OOP
- SOLID principles
- Design patterns implementation
- Asynchronous Programming
- Clean code and Best practices
- Performance and code optimization

Create an API

Create 3 different controllers for each method: **search**, **book** and **checkStatus**.

Suggested interface:

interface IManager

- SearchRes Search(SearchReq)
- BookRes Book(BookReq)
- CheckStatusRes CheckStatus(CheckStatusReq)

Based on the input/request parameters you should decide which type of manager to create. All the responses should be stored in local memory because they will be needed later on in the other endpoints. Ex: the result from **Search** will be needed in **Book** and the result from the book in **CheckStatus** method/endpoint.

To get Hotels data use this endpoint:
https://tripx-test-functions.azurewebsites.net/api/SearchHotels?destinationCode={destinationCode}

- Destination code is a short string for the destination. Ex: SKP, BCN, DXB

To get Flights data use this endpoint:

https://tripx-test-functions.azurewebsites.net/api/SearchFlights?departureAirport={departureAirport}&arrivalAirport={arrivalAirport}

- DepartureAirport is an IATA code for the airport. Ex: CPH, OSL, STO
- ArrivalAirport is a destination. Ex. SKP,BCN,DXB

Search endpoint:

The API needs to support 3 types of searches:

- **HotelOnly -** If "DepartureAirport" is not provided you should search for hotels (**HotelOnly**). In the response, FlightCode should be empty
- **HotelAndFlight -** If the search request contains "DepartureAirport" then you should do a **HotelAndFlight** search. The response should be a combination of flights and hotels.
- **LastMinuteHotels** - If "FromDate" is in the next 45 days make a last-minute search **LastMinuteHotels** (similar to hotel only).

Validate the required fields for **SearchReq**. Keep in mind that there will be more search types in the future, and the solution should be easily scalable/extendable.

Book endpoint:

Based on request data, generate the following object and store it in memory.

- **BookingCode** - random code (6 chars [0-9a-zA-Z])
- **SleepTime** - random number between 30-60
- **BookingTime** - (DateTime.Now)

CheckStatus endpoint:

The whole booking process can't be processed immediately, it might take some time, so we don't want the customer to wait. So, the booking will be completed after **sleepTime** has elapsed (in seconds). In the meantime, you should return status with the message "*Pending*" from this endpoint.

Ex: for a booking, if we have **sleepTime** = 45 and BookingTime = '8/16/2022 1:45:43', the booking is completed after '8/16/2022 1:46:28'.

For **HotelOnly** and **HotelAndFlight** return *Success*, and for **LastMinuteHotels** return *Failed* after the sleep time elapsed.

In a text file, provide an example request for each endpoint for testing purposes.

Search Request:
class SearchReq
- string Destination (required) - Ex. **SKP**, **BCN**, **CPH**
- string DepartureAirport (optional)
- DateTime FromDate (required)
- DateTime ToDate (required)

Search Response:
class SearchRes
- Option[] Options

class Option
- string OptionCode
- string HotelCode
- string FlightCode
- string ArrivalAirport
- double Price

Book Request:
class BookReq
- string OptionCode (required)
- SearchReq SearchReq (required)

Book Response:
class BookRes
- string BookingCode
- DateTime BookingTime

Check status request:
class CheckStatusReq
- string BookingCode (required)

Check status response:
class CheckStatusRes
- BookingStatusEnum Status

enum BookingStatusEnum
- Success
- Failed
- Pending

Bonus tasks (optimization):

1. Implement global exception handling with a generic error message. We don't want our clients to see our error messages
2. Implement header authorization