Backend test

TRIP SORTER

Guide time allowed: 2 hours

TASK

You are given a stack of boarding cards for various transportations that will take you from a point A to point B via several stops on the way. All of the boarding cards are out of order and you don't know where your journey starts, nor where it ends. Each boarding card contains information about seat assignment, and means of transportation (such as flight number, bus number etc).

Write an API that lets you sort this kind of list and present back a description of how to complete your journey.

For instance the API should be able to take an unordered set of boarding cards, provided in a format defined by you, and produce this list:

1. Take train 78A from Madrid to Barcelona. Sit in seat 45B.
2. Take the airport bus from Barcelona to Gerona Airport. No seat assignment.
3. From Gerona Airport, take flight SK455 to Stockholm. Gate 45B, seat 3A.
   Baggage drop at ticket counter 344.
4. From Stockholm, take flight SK22 to New York JFK. Gate 22, seat 7B.
   Baggage will we automatically transferred from your last leg.
5. You have arrived at your final destination.

The list should be defined in a format that's compatible with the input format.
The API is to be an internal PHP API so it will only communicate with other parts of a PHP application, not server to server, nor server to client.
Use PHP-doc to document the input and output your API accepts / returns.

REQUIREMENTS

No 3rd party frameworks are allowed apart from for testing. Start all code from scratch.

1. Provide a README file containing clear, simple instructions upon how to execute the code and tests.
2. If anything needs clarification which is not detailed here, make an assumption and note this in the README file.
3. Be prepared to suggest to us how we could extend the code towards new types of transportation, which might have different characteristics.
4. The implementation of your sorting algorithm should work with any set of boarding passes, as long as there is always an unbroken chain between all the legs of the trip. i.e. it's one continuous trip with no interruptions.
5. The algorithm doesn't need to consider that departure / arrival are in the correct order. In fact there is no information about any such times on the boarding passes. It is just assumed that your next connection is waiting for you when you arrive at a destination.
6. The algorithm should have the lowest possible order of complexity (Big O notation) you could think of.

WHAT WE LOOK AT

This task is designed to give us an idea of:

1. How you structure your code.
2. Your understanding of OOP.
3. Your understanding of TDD.
4. Your ability to deliver an appropriate, simple solution to a given problem
5. How you work when faced with limited time to solve a problem of significant complexity.
6. The efficiency of the sorting algorithm you implement.