Elaway Developer Exercise

Train Ticket Machine

You are asked to write an API to support the user interface of a train ticket machine.

You will not be creating any actual User Interface but instead you should model the problem space and implement a search feature to help the user find train stations by name in order to buy a ticket.

These machines have a direct but slow and unreliable connection to the central system and use a touchscreen display which works as follows.

As the user types each character of the station's name on the touchscreen, the display should:

1. Update to show all valid choices for the next character
2. List of possible matching stations.

The illustration below shows what is needed when 'D A R T' has been entered.

User input: D A R T __

A	B	C	D	E		**DARTFORD**
F	G	H	I	J		**DARTMOUTH**
K	L	**M**	N	O		
P	Q	R	S	T		
U	V	W	X	Y		
Z						

Requirements

1. Typing a search string will return:
    a. All stations that start with the search string.
    b. All valid next characters for each matched station.
2. Runtime speed is very important.
3. A space is a valid character when returning a list of next characters.

Expected Scenarios

A. **Given** a list of stations ‘DARTFORD’, ‘DARTMOUTH’, ‘TOWER HILL’, ‘DERBY’

- **When** input ‘DART’
- **Then** should return:
    1. The characters of ‘F’, ‘M’
    2. The stations ‘DARTFORD’, ‘DARTMOUTH’

B. **Given** a list of stations ‘LIVERPOOL’, ‘LIVERPOOL LIME STREET’, ‘PADDINGTON’

- **When** input ‘LIVERPOOL’
- **Then** should return:
    1. The characters of ‘ ‘
    2. The stations ‘LIVERPOOL’, ‘LIVERPOOL LIME STREET’

C. **Given** a list of stations ‘EUSTON’, ‘LONDON BRIDGE’, ‘VICTORIA’

- **When** input ‘KINGS CROSS’
- **Then** should return:
    1. No next characters
    2. No stations

Evaluation Guidelines

1. **Understanding and interpretation of the domain**
    - Concepts
    - Boundaries
    - Ubiquitous Language
2. **Delivery quality**
    - Complete solution meeting all requirements
    - No typographical errors
    - No unnecessary files
3. **Code readability**
    - Naming of classes, records, functions, methods, properties and fields
    - Consistent code formatting
    - Adequate documentation
4. **Code quality**
    - Coding against tests
    - Code coverage & complexity
    - Correct usage of data structures and techniques
    - Solution dependencies and their correct usage
5. **Solution quality**
    - Structure and organisation
    - Separation of concerns
6. **Bonus Points**

- Patterns & Practises
- Production readiness
- Choice of communications protocol
- Docker