Backend Developer Exercise

Train ticket machine

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

You don’t have to write any actual user interface code, but you should develop a search algorithm to help the user entering the name of a station.

The machine has 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;
4. You don’t need to go overboard with your station list in your tests. A small enough list of stations to adequately test each condition will suffice

Not Required:

- A fast loading time is not required at start-up, runtime performance takes priority;
- This will be run on a dedicated machine designed for the purpose;
- The application will be used by a single user at a time. There’s no need to code for concurrency;
- No code is required for reading the stations from a data store;
- You may stub the station list or mock a station reader in your tests, whichever you feel represents the best real world solution;

Expected Scenarios:

- **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’.

- **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’

- **Given** a list of stations ‘EUSTON’, ‘LONDON BRIDGE’, ‘VICTORIA’
  - **When** input ‘KINGS CROSS’
  - **Then** the application will return:
    1. no next characters
    2. no stations

Evaluation Guidelines:

1. **Delivery quality**
   a. Complete solution meeting all requirements;
   b. Project structure;
   c. Packaging and installation;
2. **Code readability**
   a. Assemblies organization;
   b. Class organization;
   c. Class and method and fields naming;
   d. Effective documentation;
3. **Code quality**
   a. Coding against tests;
   b. Code coverage;
   c. Code complexity;
4. **Solution quality**
   a. Use of O.O.P approach;
   b. Use of S.O.L.I.D. principles;
   c. Code against interfaces;
   d. K.I.S.S;
   e. D.R.Y;
   f. Correct use of proven Patterns;
5. **Use of framework features;**
   a. Correct use of frameworks features;
   b. Data structures fit for purpose;
   c. Use of the fast features;

Submission

The delivery should be done either by email or a publicly hosted GIT repository. Email submissions should contain a single zip file as attachment, containing what would exist in the GIT repository (ideally including .git, .gitignore and the like).

The root of the project should contain a readme file with instructions on how to build and run or any other relevant information for a fellow developer.