EA Technology – Technical Test

Design documentation

2024

# Solution overview

The solution consists of the following parts:

A .NET Core web API which receives the initial request.

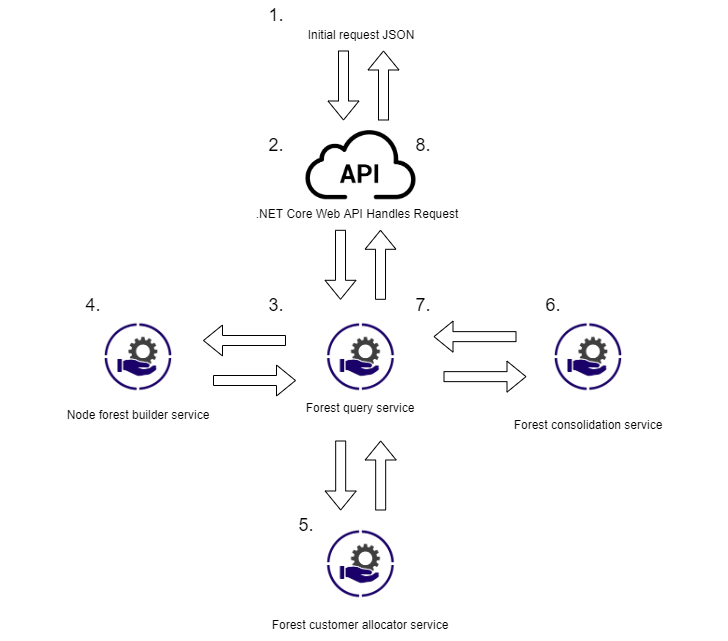
A ‘forest query service’ which handles the request from the API.

A ‘node forest builder service’ component which is responsible for converting the branch data into trees of nodes. Depending on the order of the branch data, it can produce split trees.

A ‘forest customer allocator service’ which is responsible for allocating the number of customers to each node in the node forest.

Finally, a ‘forest consolidation service’ which is responsible for combining and consolidating trees. The ‘forest consolidation service’ isn’t necessary but I’ve included it as an additional challenge to myself and hopefully to showcase my talent and interest in solving problems. It overcomes the issue of split trees when the branch data is provided out of order. In a hypothetical real-world scenario whereby the tree data is requested from multiple IoT devices, (e.g. each device is aware of its neighbours and customer allocation) the data returned from those devices would likely be received in a random order as each device responds. These responses would then need to be interpreted and consolidated to faithfully represent how they’ve been networked together.

# Order of operation



Incoming request from caller.

1. Initial incoming request from the web.
2. .NET Core Web API handles the HTTP request.
3. The forest query service handles the request in the controller method.
4. A forest builder service builds the node forests.
5. A forest customer allocator service allocates the customers to each node.
6. A forest consolidation service combines split trees.
7. The query service produces the return model data from the allocated node forest.
8. The data is returned to the API to be relayed back to the caller.

# Output data model

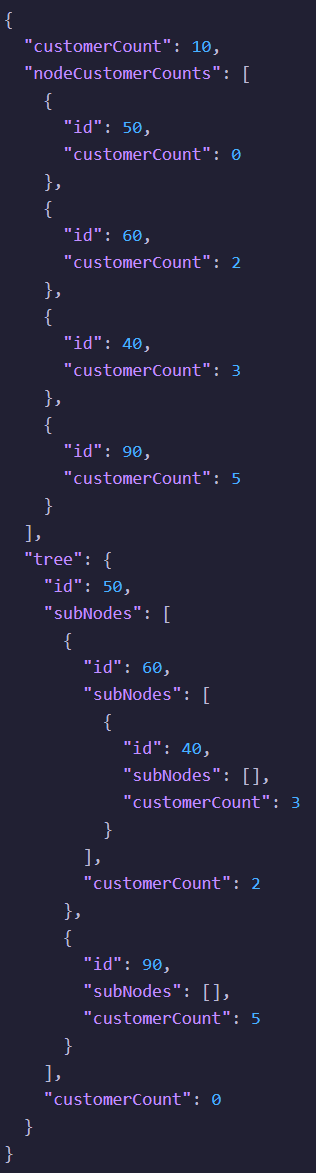
Consists of 3 parts:

The customer count based on the selected node, traversing all paths down the node tree to aggregate the customer allocation to each node.

A flat list of nodes involved and their customer allocations. Useful for simple checking and may also be useful information in this simple to interpret format by the caller.

The section of the node tree based on the selected node and all sub nodes and their customer allocations. Again, useful for checking but as allows the caller to see example how the subsection of the node tree has been interpreted and represented.

Sample output:



# Considered further improvements

Both the flat list and the tree section could be conditionally included based on additional request parameters. Efficiency saving in terms of I/O and processing, especially if the tree network was very large.

The option to traverse the tree in the opposite direction to request the same data. More flexible requests and maybe more efficient if many of the downstream nodes in a large tree aren’t relevant.

Refactoring the forest builder service to remove the need for the forest consolidation service. It may be possible to implement a more efficient builder service that tries the sort the branch data prior to process in an attempt to avoid split node trees. Efficiency saving by avoiding a secondary consolidation operation, simplification of the solution.