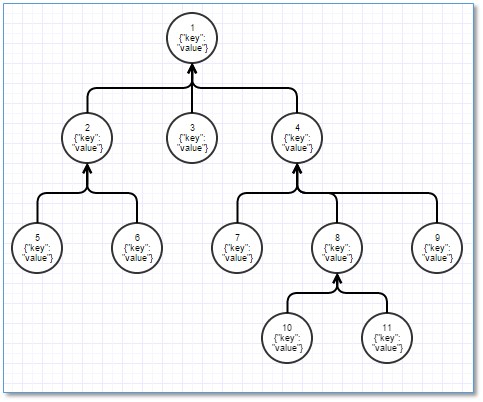
Please see below the abstract of the individual project and the evaluation criteria and procedure. Comments and feedback is encouraged.

**Context**

B2B2C systems often require the ability to store configuration cascaded based on the internal organization structure of the businesses that use the system. The runtime system behavior depends on the resulting composed (merged) configuration.

**Description**

Build a service that stores arbitrary configuration in a tree structure. Configuration is represented by JSON documents containing key value pairs and uniquely identified. The following diagram shows a representation of a tree of configuration:



The service needs to present the following functions for operating on the configuration structure:

1. Create a new configuration node under a specific parent node
2. Update the configuration node under the same parent node
3. Update the parent of a configuration node
4. Remove a single configuration node (implies also removing all of its children)
5. Fetch the configuration of a single specific node
6. Fetch the configuration of a node by computing the composed (merged) configuration using a predefined strategy (top-down / bottom-up)
7. Identify the root node
8. Identify the children of a specific node
9. Identify the parent of a node

The configuration needs to be persistent (the data needs to survive across service restarts).

The service needs to present its functions over a RESTful interface and also present a web based interface that consumes the RESTful interface for managing the configuration stored.

The service usage pattern is 99.9% reads / 0.1% writes. It needs to be performant on reads, while performance on writes (create / update) is a secondary concern.

**Additional constraints:**

* Languages: Java / Javascript
* Technology stack: Spring - Spring MVC / Java EE
* Persistent store: Any familiar persistent store, preferably JPA compliant, but not mandatory

**Example configuration node:**

{

  “max\_daily\_spend\_per\_card”: 1000.0,

  “country\_whitelist”: [

    “RO”, “GB”, “US”

  ]

}

**Merging configuration:**

Whenever the service is required to return a composed (merged) configuration it needs to read the configuration of all the nodes that form the path between the specified node and the root node.

Depending on the strategy specified (top-down / bottom-up) it will compose the resulting configuration:

* Top-down: for duplicate configuration entries in parent / child -> the parent value wins
* Bottom-up: for duplicate configuration entries in parent / child -> the child value wins

**Additional notes:**

The usage pattern allows the use of eager caching for high throughput reads, but requires an external caching system for scaling out.

There is sufficient material on the internet on the options available to persist tree based structures of arbitrary data (there’s no need to re-invent the wheel).

The code needs to be available on github.

**Evaluation Criteria**

* The system meets the requirements
  + It publishes all the required functions correctly
  + It publishes its functions over a RESTful interface
  + Data is persistent
  + It publishes a web interface that allows access to all its functions
* UI/UX – ease of use of the web interface
* Coding standards (including reasonable code coverage for both web interface and server)
* Performance – sustained (10 minutes) throughput of the usage scenario on a system with 10,000 nodes

**Evaluation Procedure**

Each person will have 15 minutes to present his solution (10 minutes for presentation + 5 minutes for Q & A):

* the running system and the interaction with the system
* a high level design of a system (that includes a description of the solution, chosen technology stack and persistent store and their use)
* achieved throughput for the usage pattern

The final score will be determined by result of the presentation exercise and the codebase on the referenced github repository.