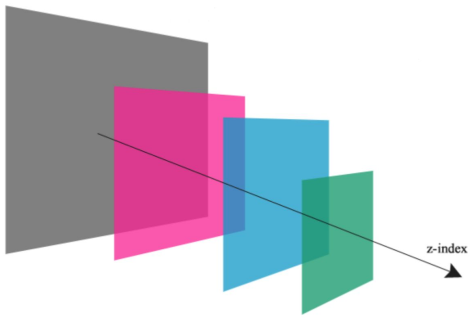
Test task for a backend developer

Summary

A web service (HTTP REST API) to work with widgets.



Glossary

A Widget is an object on a plane in a Cartesian coordinate system that has coordinates (X, Y), Z-index, width, height, last modification date, and a unique identifier. X, Y, and Z-index are integers (may be negative).

A Z-index is a unique sequence common to all widgets that determines the order of widgets (regardless of their coordinates). Gaps are allowed. The higher the value, the higher the widget lies on the plane.

Details

Operations to be provided by the web service:

- Creating a widget. Having set coordinates, z-index, width, and height, we get a complete widget description in the response.
    - The server generates the identifier.
    - If a z-index is not specified, the widget moves to the foreground. If the existing z-index is specified, then the new widget shifts all widgets with the same and greater index upwards.
- Getting a widget by its identifier. In the response we get a complete description of the widget.
- Change widget data by its identifier. In the response we get an updated full description of the widget.
    - You cannot change widget id.
    - You cannot delete widget attributes.
    - All changes to widgets must occur atomically. That is, if we change the *XY* coordinates of the widget, then we should not get an intermediate state during concurrent reading.
- Deleting a widget. We can delete the widget by its identifier.

- Getting a list of widgets. In the response we get a list of all widgets sorted by z-index, from smallest to largest.

Requirements

- The API must conform to the REST architecture.
- Do only the server side, you don't need to do visualization.
- It should be a Spring Boot application.
- Maven should be used as a build tool.
- Data should only be stored in memory. You can use any classes of the standard Java library to organize storage. It is not allowed to use any external repositories and databases.
- All changes to widgets must occur atomically. Z-index must be unique.
- At least 30% of the code should be covered by tests (preferably the presence of both unit and integration tests).
- Submit sources via a public git repository.

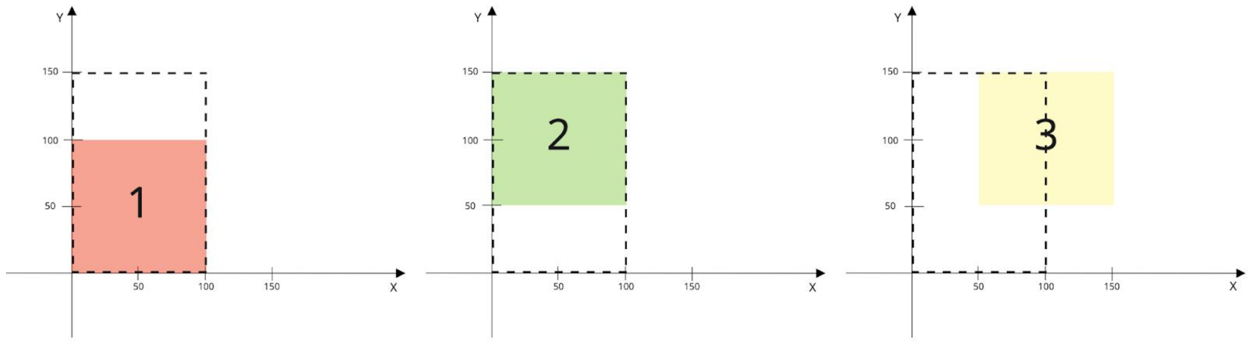
Complication

All complications are optional — you may or may not do them. You can pick any of them.

1. Pagination
Implement a pagination mechanism for a list of widgets request, which by default will produce 10 elements, but with the ability to specify a different number (up to 500).

2. Filtering
While getting the list of widgets, we can specify the area in which widgets are located. Only the widgets that fall entirely into the region fall into the result.

For example, we have 3 widgets that have a width and a height of 100. The centers of these widgets are at points 50:50, 50:100 and 100:100. We want to get only the widgets that are inside the rectangle, the lower-left point of which is at 0:0, and the upper right is at 100:150.



In this case, we should get the first and second widgets.

An important condition is to find an implementation method in which the asymptotic complexity of the algorithm is on average less than O (n).

3. Rate limiting
It is necessary to implement the possibility to limit the number of requests sent to the server in a time interval, and if the specified limit is exceeded, the server returns an appropriate response. The restriction should be global - for all web service clients. By default, there is a rate limit for all requests, but it should be possible to set a value for a specific endpoint. All values (both default and overridden) can be changed without restarting the application. In the response headers, the server should return the rate limit parameters for the called endpoint: rate limit itself, the number of available requests, the date of the next reset (window border).

Settings example: default limit is 1000 requests per minute, rate limit for receiving all widgets on the board is 200 requests per minute.

4. SQL database
You need to create another storage implementation that works with SQL databases. You can use any database in-memory implementation (for example, H2).

The choice of storage implementation (native or in-memory SQL) should be determined in the server configuration and does not require changes in the code.