Kaiburr Task1 documentation

In today's world, REST APIs have become an essential part of the software development process. By providing a simple and universal interface, REST APIs allow developers to create applications that can communicate with each other seamlessly. In this essay, we will discuss how to implement an application in Java that provides a REST API with endpoints for searching, creating, and deleting "server" objects.

To begin with, let's define what a REST API is. REST (Representational State Transfer) is an architectural style that defines a set of constraints to be used for creating web services. REST is based on the HTTP protocol, and it uses HTTP verbs (GET, POST, PUT, DELETE) to interact with resources. A REST API provides a uniform interface, which means that the implementation details of the server are hidden from the client.

Now that we understand what a REST API is, let's move on to the implementation details. In this scenario, we need to create endpoints for searching, creating, and deleting "server" objects. First, let's consider the GET request. When no parameters are passed, the endpoint should return all the servers. If a server ID is passed, the endpoint should return a single server or a 404 error if there's no such a server. To implement this endpoint, we need to use the GET verb and define two different routes: one for retrieving all the servers and one for retrieving a single server by ID.

The PUT request is used to create a new server object. The server object is passed as a JSON-encoded message body, and it contains the name, ID, language, and framework of the server. To implement this endpoint, we need to use the PUT verb and define a route that accepts a JSON-encoded message body.

The DELETE request is used to delete a server object by its ID. To implement this endpoint, we need to use the DELETE verb and define a route that accepts a server ID as a parameter.

Finally, we need to implement an endpoint to find servers by name. The endpoint should accept a string parameter and return one or more servers found. If nothing is found, the endpoint should return a 404 error. To implement this endpoint, we need to use the GET verb and define a route that accepts a string parameter.

All "server" objects should be stored in a MongoDB database. MongoDB is a popular NoSQL database that stores data in JSON-like documents. Using a NoSQL database like MongoDB allows us to store data in a flexible and scalable way.

In conclusion, implementing a REST API with endpoints for searching, creating, and deleting "server" objects requires careful planning and attention to detail. By using HTTP verbs and defining routes, we can create a simple and universal interface that allows different applications to communicate with each other seamlessly. MongoDB provides a flexible and scalable way to store "server" objects. With the right implementation, our REST API can become an essential part of the software development process.