Sending Data and RequestBody

Now let's see how to create an API that accepts data from the client. We already have addJob() methods in our repository and service layers, so we just need to add an endpoint in our controller.

👉 @RequestBody

The @RequestBody annotation is crucial for receiving data in REST APIs:

- It converts JSON data from the request body into a Java object.
- Spring automatically maps JSON fields to matching properties in your Java class.
- No need to manually parse JSON - Spring does it for you.
- Data comes in as the exact object type you need.
- Implementing the Add Job API

```
@PostMapping("/jobPost")
public JobPost addJob(@RequestBody JobPost jobPost){
    service.addJob(jobPost);
    return service.getJob(jobPost.getPostId());
}
```

- @PostMapping("/jobPost") creates an endpoint that accepts POST requests
- @RequestBody JobPost jobPost converts the incoming JSON to a JobPost object
- The method adds the job and then returns the newly added job

Example:

```java
package com.telusko.spring_boot_rest.controller;
import java.util.List;
import com.telusko.spring_boot_rest.model.JobPost;
import com.telusko.spring_boot_rest.service.JobService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "http://localhost:3000")
public class JobRestController {
    @Autowired
    private JobService service;

    @GetMapping("jobPosts")
    public List<JobPost> getAllJobs() {
        return service.getAllJobs();
    }

    @GetMapping("jobPost/{postId}")
    public JobPost getJob(@PathVariable("postId") int postId){
         return service.getJob(postId);
    }

    @PostMapping("/jobPost")
    public JobPost addJob(@RequestBody JobPost jobPost){
        service.addJob(jobPost);
        return service.getJob(jobPost.getPostId());
    }
}
```

Output: