# 1. Module 2 Overview

The goal of the second module is - now that we have a working, simple API - we'll lay the groundwork of a RESTful architecture.

We start the module with some theory, but we also put the theoretical aspects in practice very early on.

At the end of this module, we should have a very good understanding of how the API plays into HTTP and a working project shaped accordingly.

# 2. Goal of this Lesson

Understand the error semantics in the API and learn how to implement these semantics with Spring.

# 3. Lesson Notes

## 400 Bad Request

*“The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications."*

Quick notes on 400 Bad Request:

- the generic, catch-all for client side errors
- usually happens for PUT or POST requests

- usually means that the Representation of the Resource is in the right format but still doesn't make sense

**Example 1 - invalid JSON input - unknown fields**

*POST http://localhost:8082/api/privileges*

```
{
    "nameInvalid" : "PRIV10",
    "description" : "descr10"
}
```

Notice how, in the implementation - we are delegating to the *handleExceptionInternal* method in the base class. This is a simple and very useful helper implementation we can use to customize the body of our responses, when we handle exceptions.

**Example 2 - valid JSON input - but validation error**

*POST http://localhost:8082/api/privileges*

```
{
    "name" : null,
    "description" : "descr10"
}
```

Notes:

- it's going to result in a *MethodArgumentNotValidException*
- handled by *handleMethodArgumentNotValid*in the global exception handler

## Global Exception Handler - High Level

Finally, let's talk about this global exception handler.

The right way to think about this class is - independent of the application code.

At a high level - we have application code on the one hand - throwing exceptions, and on the other hand we have this exception handler - handling them.

As long as we have have a match - exceptions thrown by application code are handled here - it doesn't matter how the exceptions are thrown or what type they are.

You'll of course have to map out all of the exceptions of your system in order to know what to handle - and a great way to do that is with a large suite of live tests, triggering these exceptional scenarios in the API.

# 4. Resources

- Error Handling for REST with Spring

- The Module 2 Branch of the project on Github