4. Migrate Java API to Kotlin API

Let’s start by converting the *PrivilegeController* to Kotlin and moving a single method. We'll create a new package *com.baeldung.um.web.controller* under the Kotlin source directory

Then, we'll create the following *PrivilegeController* Kotlin class:

```
JAVA
@RestController
@RequestMapping(UmMappings.PRIVILEGES)
class PrivilegeController(private val service: IPrivilegeService) : AbstractController<Privilege>()

    @GetMapping("/{id}")
    fun findOne(@PathVariable("id") id: Long): Privilege = findOneInternal(id);

    fun findOneInternal(id: Long): Privilege = RestPreconditions.checkNotNull(getService().findOne(:

    override fun getService() = service

}
```

Let’s run the code and test this endpoint. Note that, for now, we have to delete the *PrivilegeController.java* class to avoid conflicts on startup.

Similarly we can convert all the *PrivilegeController* operations to Kotlin.

The final module of this lesson has the fully converted controller and the service layer for the *privilege* resource.

Next, we're going to use the following interesting features available in Kotin:

- Kotlin data classes
- Kotlin Extension functions

We'll use Kotlin Extension function for *Entity* to *Dto* conversion.

Let’s create a *com.baeldung.um.web.dto* package in the Kotlin source directory.

Then, we'll create *PrivilegeDto.kt* with the following code in *com.baeldung.um.web.dto* package:

```
JAVA
data class PrivilegeDto(val id: Long, val name: String, val description: String?)

fun Privilege.toDto() = PrivilegeDto(id, name, description)
```

Let’s now implement the *findOne()* method in *PrivilegeController* using the *PrivilegeDto*:

```
JAVA
@GetMapping("/{id}")
    fun findOne(
        @PathVariable("id") id: Long): PrivilegeDto = findOneInternal(id).toDto()
```

Note that we are using Kotlin Extension function for *Entity* to *Dto* conversion.

We can now test the endpoint for retrieving a privilege again.

5. Resources

- Kotlin Reference - Using Maven