**Steps for Creating API’s**

Create the following packages first

1. org.knowsterplatfrm.organisation.office.api
   1. this package holds all ApiResource classes this is the class where request comes from client/browser
   2. Naming convention to be followed while creating API class is
      1. Make sure to append “ApiResource” after every class name
         1. Example **OfficesApiResource .java**
2. org.knowsterplatfrm.organisation.office.data
3. org.knowsterplatfrm.organisation.office.domain
4. org.knowsterplatfrm.organisation.office.service
5. org.knowsterplatfrm.organisation.office.handler
6. org.knowsterplatfrm.organisation.office.serialization
7. org.knowsterplatfrm.organisation.office.exception

**OfficesApiResource.java:**

**private** **final** Set<String> RESPONSE\_DATA\_PARAMETERS = **new** HashSet<String>(Arrays.*asList*("id", "name", "nameDecorated", "externalId",

"openingDate", "hierarchy", "parentId", "parentName", "allowedParents"));

The above response parameter is used to validate the the response parameters being passed as response to the client.

**private** **final** String resourceNameForPermissions = "OFFICE";

the above variable resourceNameForPermissions holds is used as a authentication parameter,

internally OFFICE will get converted into OFFICE\_READ variable to validate weather the user is Authorized to access this resource or not

**private** **final** PlatformSecurityContext context;

**private** **final** OfficeReadPlatformService readPlatformService;

**private** **final** DefaultToApiJsonSerializer<OfficeData> toApiJsonSerializer;

**private** **final** ApiRequestParameterHelper apiRequestParameterHelper;

**private** **final** PortfolioCommandSourceWritePlatformService commandsSourceWritePlatformService;

@Autowired

**public** OfficesApiResource(**final** PlatformSecurityContext context,

**final** OfficeReadPlatformService readPlatformService,

**final** DefaultToApiJsonSerializer<OfficeData> toApiJsonSerializer,

**final** ApiRequestParameterHelper apiRequestParameterHelper,

**final** PortfolioCommandSourceWritePlatformService commandsSourceWritePlatformService) {

**this**.context = context;

**this**.readPlatformService = readPlatformService;

**this**.toApiJsonSerializer = toApiJsonSerializer;

**this**.apiRequestParameterHelper = apiRequestParameterHelper;

**this**.commandsSourceWritePlatformService = commandsSourceWritePlatformService;

}

Above we are creating and initializing service and components which are going to be used for validating user authentication and authorization(**context**),

Reading offices/user details from database(**readPlatformService**,

converting Java objects into Json using Gson before sending back the response(**toApiJsonSerializer**).

For validating the user parameters which are being returned to the user in response(**apiRequestParameterHelper**).

And for creating Beans after validating the user permission for performing write operations(**PortfolioCommandSourceWritePlatformService**)

@GET

@Consumes({ MediaType.APPLICATION\_JSON })

@Produces({ MediaType.APPLICATION\_JSON })

**public** String retrieveOffices(@Context **final** UriInfo uriInfo) {

context.authenticatedUser().validateHasReadPermission(resourceNameForPermissions);

**final** Collection<OfficeData> offices = **this**.readPlatformService.retrieveAllOffices();

**final** ApiRequestJsonSerializationSettings settings = apiRequestParameterHelper.process(uriInfo.getQueryParameters());

**return** **this**.toApiJsonSerializer.serialize(settings, offices, RESPONSE\_DATA\_PARAMETERS);

}

Above code implements GET method for OfficeApiResource which returns all offices details,

1. first it check for the user authentication
2. it retrives offices details from offices tables using readPlatformService class methiod retrieveAllOffices();
3. third it gets the query parameters from the URL string and check weather users had requested for specific return parameters.(not mandatory in all cases)
4. fourth the office data is being converted into a JSON object/string for transferring it back to user.

That was GET call now lets check POST method call

@POST

@Consumes({ MediaType.APPLICATION\_JSON })

@Produces({ MediaType.APPLICATION\_JSON })

**public** String createOffice(**final** String apiRequestBodyAsJson) {

**final** CommandWrapper commandRequest = **new** CommandWrapperBuilder() //

.createOffice() //

.withJson(apiRequestBodyAsJson) //

.build();

**final** CommandProcessingResult result = **this**.commandsSourceWritePlatformService.logCommandSource(commandRequest);

**return** **this**.toApiJsonSerializer.serialize(result);

}

The above code is a POST call to OfficeApiResource webservice it performs following operations:

1. it creates a Command Wrapper using CommanWrapperdBuilder class which creates “CREATE\_OFFICE” key string for user authentication check at the time of writing data to database.
2. Second line of code will log commandRequest object. This method will convert the json String into Java object and performs necessary validation check then then persists the object into table.
   1. There are many steps performed while doing this operation, which we are going to see later.
3. As usual the response is returned back to the user/client.

The above steps where for creating a API Resource class.

Let’s have a look at Service Classes, where main business logic is written.

Following Naming Convention has to be followed when creating a Service class for performing Read/Write operation.

If the service is for reading then it has to be like.. OfficeReadPlatformService

And for write it has to be OfficeWritePlatfromService.

And there implementation classes has to be like. OfficeReadPlatfromServiceImpl and OfficeWritePlatformServiceImpl