1. **Custom Error Schema & Base Exception Design with Builder Pattern& Implement (Create own schema)**

* **Create own Attribute class, Extend & Override Spring default error Attributes**

@Component

public class ApplicationErrorAttributes extends DefaultErrorAttributes {

@Override

public Map<String, Object> getErrorAttributes(WebRequest webRequest,

ErrorAttributeOptions options) {

Map<String, Object> errorAttributes = super.getErrorAttributes(webRequest, options);

errorAttributes.put(ApiConstants.TYPE, ApiConstants.TYPE\_DEFAULT\_VALUE);

errorAttributes.put(ApiConstants.TITTLE, errorAttributes.get(ApiConstants.ERROR));

errorAttributes.put(ApiConstants.INSTANCE, errorAttributes.get(ApiConstants.PATH));

errorAttributes.put(ApiConstants.TIMESTAMP, OffsetDateTime.now());

errorAttributes.remove(ApiConstants.PATH);

errorAttributes.remove(ApiConstants.ERROR);

errorAttributes.remove(ApiConstants.MESSAGE);

return errorAttributes;

}

}

* **Create Constant Class**

public static final String TYPE = "type";

public static final String TITTLE = "tittle";

public static final String INSTANCE = "instance";

public static final String TIMESTAMP = "timestamp";

public static final String ERROR = "error";

public static final String PATH = "path";

public static final String MESSAGE = "message";

public static final String TYPE\_DEFAULT\_VALUE = "about:blank";

**Done…. Now default error schema has been changed.**

* **BASE EXCEPTION CLASS DESIGN**
  + **Create BaseException Class here You can change fields as per the requirements**

@Getter

public class BaseException extends Exception {

private URI type;

private String detail;

private URI instance;

private final String code;

private Map<String, Object> params;

@Builder(builderMethodName = "baseBuilder")

public BaseException(URI type, String detail, URI instance, String code,

List<ErrorResponseSubDetails> subDetails,

Map<String, Object> params) {

super(detail);

this.type = type;

this.detail = detail;

this.instance = instance;

this.code = code;

this.params = params;

}

public static BaseExceptionBuilder baseBuilder(String code) {

return new BaseExceptionBuilder().code(code);

}

}

1. **IMPELEMENT THIS ERROR IN A SERVICE (EXAMPLE)**

* **Create a Exception Class**

@Getter

public class ContentNotFoundException extends BaseException {

private String newField;

@Builder(builderMethodName = "contentNotFoundBuilder")

public ContentNotFoundException(URI type, String detail, URI instance, String code,

List<ErrorResponseSubDetails> subDetails,

Map<String, Object> params) {

super(type, detail, instance, code, subDetails, params);

this.newField = newField;

}

public static ContentNotFoundExceptionBuilder contentNotFoundBuilder(String code) {

return new ContentNotFoundExceptionBuilder().code(code);

}

}

* **Create Handler**

@RequiredArgsConstructor

@ControllerAdvice

@JsonInclude(JsonInclude.Include.NON\_NULL)

public class ContentStackExceptionHandler extends BaseExceptionHandler {

private final HttpServletRequest httpServletRequest;

private final ResourceBundleMessageSource messageSource;

@ExceptionHandler(ContentNotFoundException.class)

public ResponseEntity<ErrorResponse> handleContentNotFoundException(ContentNotFoundException e) {

String details = null;

HttpStatus status = HttpStatus.NOT\_FOUND;

if (e.getCode() != null) {

details = messageSource.getMessage(e.getCode(), new Object[] {e.getDetail()}/\*null\*/,

new Locale(httpServletRequest.getHeader(ApiConstants.ACCEPT\_LANGUAGE)));

}

ErrorResponse errorResponse = new ErrorResponse();

errorResponse.setTimestamp(OffsetDateTime.now());

errorResponse.setInstance(URI.create(httpServletRequest.getRequestURI()));

errorResponse.setType(URI.create(ApiConstants.TYPE\_DEFAULT\_VALUE));

errorResponse.setTitle(status.name());

errorResponse.setDetail(details);

errorResponse.setCode(e.getCode());

return ResponseEntity.status(status).body(errorResponse);

}

}

* **Throw Exception from Service**

if (!"header".equals(name) && !"footer".equals(name)) {

throw ContentNotFoundException.contentNotFoundBuilder("comcartds-invalid-content")

.detail(name).build();

}

1. GENERIC METHOD & GET REQUEST PARAMETERS
2. public static <T> ResponseEntity<T> exchangeProxy(Class<T> responseType, Object body){  
    Enumeration<String> incomingHeaders = null;  
    HttpServletRequest incomingRequest = null;  
    String requestURI = null;  
    HttpHeaders httpHeaders = new HttpHeaders();  
    ServletRequestAttributes requestAttributes =  
    (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
    if (requestAttributes != null) {  
    incomingRequest = requestAttributes.getRequest();  
    requestURI = incomingRequest.getRequestURI();  
    if(!StringUtils.*isEmpty*(incomingRequest.getQueryString())){  
    requestURI = requestURI+"?"+incomingRequest.getQueryString();  
    }  
    incomingHeaders = incomingRequest.getHeaderNames();  
    }  
    if (incomingHeaders != null) {  
    while (incomingHeaders.hasMoreElements()) {  
    String header = incomingHeaders.nextElement();  
    String value = incomingRequest.getHeader(header);  
    httpHeaders.add(header, value);  
    }  
    }  
    return *restTemplate*.exchange(*basePath*+requestURI,HttpMethod.*resolve*(incomingRequest.getMethod()),new HttpEntity<>(httpHeaders),responseType);  
    }

* Get application properties without @Value



* Rest template unit tests



* Microcks client using rest template



* Common config



* String split example using stream
* String api = Arrays.stream(apiPath.split("/"))  
   .map(part -> {  
   try {  
   return URLEncoder.encode(part, StandardCharsets.UTF\_8.toString());  
   } catch (UnsupportedEncodingException e) {  
   throw new RuntimeException(e);  
   }  
   })  
   .collect(Collectors.joining("/"));  
  return ApiConstants.MICROCKS\_BASE\_URL + api;
* Controller Layer junit
* 



* Service layer for webClient





* Service layer test for Rest Template

