Transforming API Response

- Extend your class with `@ResponseBodyAdvice<Object>` to define the custom return type
- Use `@RestControllerAdvice` for global API Response transformation
- Return appropriate HTTP status codes and error messages
- You can also return the timestamp of the API response

Goal with this activity is to pass only three segments of the API response in all possible times to contain `timestamp`, `data`, `error`. Start by declaring a class the holds this info in a structure

```
@Data
public class ApiResponse<T> {
    @JsonFormat(pattern = "hh:mm:ss dd-MM-yyyy")
    private LocalDateTime timestamp;

    private T data;

    private ApiError error;

    public ApiResponse() {
        this.timestamp = LocalDateTime.now();
    }

    public ApiResponse(T data) {
        this();
        this.data = data;
    }

    public ApiResponse(ApiError error) {
        this();
```

```
        this.error = error;
    }
}
```

Similar to our global exception handler, to the entire app responses that are sent we would want to send it via only `GlobalResponseHandler`.

```
@RestControllerAdvice
public class GlobalResponseHandler implements ResponseBodyAdvice<Ob
ject> {

    @Override
    public boolean supports(MethodParameter returnType, Class<? extends
HttpMessageConverter<?>> converterType) {
        return true;
    }

    @Override
    public @Nullable Object beforeBodyWrite(
            @Nullable Object body,
            MethodParameter returnType,
            MediaType selectedContentType,
            Class<? extends HttpMessageConverter<?>> selectedConverterTyp
e,
            ServerHttpRequest request, ServerHttpResponse response
          ) {
        if (body instanceof ApiResponse<?>) {
            return body;
        }

        return new ApiResponse<>(body);
    }
}
```

Once done we can have the global exception handler also to return the same data type now,

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiResponse<?>> handleResourceNotFound(Res
ourceNotFoundException exception) {
        ApiError apiError = ApiError
                            .builder()
                            .status(HttpStatus.NOT_FOUND)
                            .message(exception.getMessage())
                            .build();

        return buildErrorResponseEntity(apiError);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ApiResponse<?>> handleInputValidationErrors(M
ethodArgumentNotValidException exception) {
        List<String> errors = exception
                .getBindingResult()
                .getAllErrors()
                .stream()
                .map(err -> err.getDefaultMessage())
                .collect(Collectors.toList());

        ApiError apiError = ApiError
                .builder()
                .status(HttpStatus.BAD_REQUEST)
                .message("Input validation failed")
                .subErrors(errors)
                .build();

        return buildErrorResponseEntity(apiError);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiResponse<?>> handleException(Exception ex
ception) {
```

```
        ApiError apiError = ApiError
                            .builder()
                            .status(HttpStatus.INTERNAL_SERVER_ERROR)
                            .message(exception.getMessage())
                            .build();

        return buildErrorResponseEntity(apiError);
    }

    private ResponseEntity<ApiResponse<?>> buildErrorResponseEntity(Api
Error apiError) {
        return new ResponseEntity<>(new ApiResponse<>(apiError), apiError.
getStatus());
    }
}
```