Summary

This RFC intends to address three areas:

[1] a number of gaps in code generation in the light-codegen, where config files are not generated for all the modules which are automatically added to the default handler chain

[2] generation, on demand, and configurable, of environment variables for each configuration file generated by the light-codegen, scoped uniquely across the configuration of that API.
Work on environment variable/values support is in place, based on the RFC: "light-4j/0006-config-injection-guidance indicates how environment variables/values"

[3] ensure that the code, with additional support in the light-4j/config module, returns the same data for directly set default values or environment variable-based default values

Motivation

[1] APIs move from Development to multiple test phase, the PTE and Production, with a set number of variables changing in the config files from environment to environment.

This allows a team to use the values.yml file as a one-stop shop for changes across environments

[2] Development teams would see all configuration files generated for their APIs, according to the handlers supported in the handler.yml file.
At this time, some are deeply hidden in the respective modules and they are ignored by teams, who might not be aware of their ability to use them.

Guide-level explanation

These are the proposed changes for this RFC:

[1] handler.yml injects the following handlers in the chain by default and will have to add the corresponding config file for each handler, giving a developer a clear understanding of all config files available for the generated code.

```
handlers:
  # Light-framework cross-cutting concerns implemented in the microservice
  - com.networknt.exception.ExceptionHandler@exception
```

```
    - com.networknt.metrics.MetricsHandler@metrics
    - com.networknt.traceability.TraceabilityHandler@traceability
    - com.networknt.correlation.CorrelationHandler@correlation
    - com.networknt.openapi.OpenApiHandler@specification
    - com.networknt.openapi.JwtVerifyHandler@security
    - com.networknt.body.BodyHandler@body
    - com.networknt.audit.AuditHandler@audit
    # DumpHandler is to dump detail request/response info to log, useful for troubleshoo
    # - com.networknt.dump.DumpHandler@dump
    - com.networknt.sanitizer.SanitizerHandler@sanitizer
    - com.networknt.openapi.ValidatorHandler@validator
    # Customer business domain specific cross-cutting concerns handlers
    # - com.example.validator.CustomizedValidator@custvalidator
    # Framework endpoint handlers
    - com.networknt.health.HealthGetHandler@health
    - com.networknt.info.ServerInfoGetHandler@info
    - com.networknt.specification.SpecDisplayHandler@spec
    - com.networknt.specification.SpecSwaggerUIHandler@swaggerui
    # - com.networknt.metrics.prometheus.PrometheusGetHandler@getprometheus
```

There are a number of handlers which teams MUST change in PTE/PROD, to assist in their functionality.

For example, the Correlation module defines in its own config:

```
# Correlation Id Handler Configuration
---
# If enabled is true, the handler will be injected into the request and response chain
enabled: true

# If set to true, it will auto-generate the correlationID if it is not provided in the
autogenCorrelationID: true
```

If the DevOps teams need to reset the `autogenCorrelationID` element for example, they need to create themselves a correlation.yml file in their API, then add an environment variable, followed by setting it in the values.yml

With this change, a correlation.yml will be added to the generated /sr/main/resources/config folder.
Part [3] will explain hw the correlatio.yml file will be generated on demand with/witout an environment variable, as follows:

```
# Correlation Id Handler Configuration
---
# If enabled is true, the handler will be injected into the request and response chain
enabled: true

# If set to true, it will auto-generate the correlationID if it is not provided in the
autogenCorrelationID: true
```

or

```
# Correlation Id Handler Configuration
---
# If enabled is true, the handler will be injected into the request and response chain
enabled: ${correlation.enabled:true}

# If set to true, it will auto-generate the correlationID if it is not provided in the
autogenCorrelationID: ${correlation.autogenCorrelationID:true}
```

Therefore if the handler.yml generates the above mentioned handlers, it should also generate the following config files:

```
- metrics.yml
- correlation.yml
- traceability.yml
- audit.yml
- body.yml
- sanitizer.yml
- validator.yml
- health.yml
- info.yml
```

[2] The list of handlers to be generated is to be added to the config.json file as a list
It can be added either as a list of `handlers`, using the format `handlerName : configModule`.

default:
If it is not specified, a default list as currently available in the handler.yml should be used, leaving the current config.json unchanged from this perspective

[3] Generating environment variables on demand, in a configurable manner

The config.json file is to be augmented with a

`generate environment variables on demand element`, called `generateEnvVars`.

If disabled, each element in each file will be defaulted, as in:

```
# Correlation Id Handler Configuration
---
# If enabled is true, the handler will be injected into the request and response chain
enabled: true

# If set to true, it will auto-generate the correlationID if it is not provided in the
autogenCorrelationID: true
```

If it is enabled, each element in reach file will be augmented with an environment variable, ensuring uniqueness across the API, in the format: **${file_name.key:default_value}**

```
# Correlation Id Handler Configuration
---
# If enabled is true, the handler will be injected into the request and response chain
enabled: ${correlation.enabled:true}

# If set to true, it will auto-generate the correlationID if it is not provided in the
autogenCorrelationID: ${correlation.autogenCorrelationID:true}
```

The light-codegen can automatically generate the environment variables by adding code in or around the transfer() function call:

```
Ex.:
changing the config map passed into the transfer() call

transfer(targetPath, ("src.main.resources.config").replace(".", separator), "service.y
```

[4] light-4j/config module needs to ensure that the data returned to an API is consistent, primitive

or object, regardless if the config value was set directly as `enabled: true` or `enabled: ${correlation.enabled:true}`

Reference-level explanation

Drawbacks

Rationale and Alternatives

Unresolved questions