Guide to Server Stub generation using maven plugin

1. Create a Spring Boot starter project using any IDE(Eclipse/STS/others). I have used eclipse.
2. Place the **.yaml** files inside the **src/main/resources** folder in the maven project created by the previous step. Sample yaml file is provided at the end of this document.
3. Add the swagger code-gen plugin to the plugins section in **pom.xml**:

```
<plugin>
            <groupId>io.swagger.codegen.v3</groupId>
            <artifactId>swagger-codegen-maven-plugin</artifactId>
            <version>3.0.5</version>
            <executions>
             <execution>
               <id>First</id>
                <goals>
                  <goal>generate</goal>
                </goals>
             <configuration>
            <inputSpec>src/main/resources/userConfiguration.yaml</inputSpec>
                        <output>${project.basedir}</output>
                        <language>spring</language>
                        <generateModels>true</generateModels>
                        <modelsToGenerate>UserConfigurationRequest, UserConfiguration,
            UserConfigurations</modelsToGenerate>
            <configOptions>
                        <artifactDescription>User Configuration API</artifactDescription>
                        <title>User Configuration API</title>
                         <groupId>com.host.cbre.userconfiguration</groupId>
                        <artifactVersion>0.1.0-SNAPSHOT</artifactVersion>
                        <artifactId>userconfiguration</artifactId>
                        <sourceFolder>src/main/java</sourceFolder>
                        <library>spring-boot</library>
                        <basePackage>com.cbre.host.userconfiguration</basePackage>
                        <apiPackage>com.cbre.host.userconfiguration.controller</apiPackage>
                        <configPackage>com.cbre.host.userconfiguration.config</configPackage>
            </configOptions>
           </configuration>
</execution>
<execution>
               <id>Second</id>
                <goals>
                  <goal>generate</goal>
                </goals>
             <configuration>
            <inputSpec>src/main/resources/crud.yaml</inputSpec>
                        <output>${project.build.directory}/generated-sources</output>
                        <language>spring</language>
            <configOptions>
                        <artifactDescription>CRUD API</artifactDescription>
                        <title>CRUD  API</title>
```

```
                    <groupId>com.host.cbre.crud</groupId>
                    <artifactVersion>0.1.0-SNAPSHOT</artifactVersion>
                    <artifactId>crud</artifactId>
                    <sourceFolder>src/main/java</sourceFolder>
                    <library>spring-boot</library>
                    <basePackage>com.cbre.host.crud</basePackage>
                    <apiPackage>com.cbre.host.crud.controller</apiPackage>
                    <configPackage>com.cbre.host.crud.config</configPackage>
            </configOptions>
          </configuration>
    </execution>

    </executions>

  </plugin>
```

4. If we need to generate code for multiple services (with multiple .yaml files). Add a new execution tag in the plugin with its corresponding configurations.
5. Run the following while right clicking on the pom.xml
    a. Run As → Maven clean
    b. Run As → Maven generate-sources
6. ***Old pom.xml will be replaced with a new one if the <output> is set to ${project.basedir}.*** So, we set the **<output>** to generate the code in ***${project.build.directory}/generated-sources***. It will generate the code in target/generated-sources folder. Copy the packages to the src/main/java after the generation. Generate resources will generate the classes and place them in different packages as configured in configOptions of the plugin
    a. Application main class Swagger2SpringBoot.java is placed in the **<basePackage>**
    b. Controller interface and classes is placed in configured **<apiPackage>**
    c. Swagger doc is placed in configured**<configPackage>**
7. Then start providing the implementation in the controller class. Then provide the service layer and the persistence layers.

Note: *This version (3.0.5) of the plugin supports OpenAPI spec version 3. All the models are generated appropriately. One hurdle I see here is when we make changes to the existing services(.yaml). The code will be overwritten, and we lose the earlier code. For this the suggested approach is just to generate only the interface and we(developers) write the implementation class where we can accommodate the changes.*

userConfiguration.yaml

```
The http error codes are made compatible with the Kinvey
standards(https://devcenter.kinvey.com/rest/reference/flex/reference.html )
openapi: 3.0.0
info:
  version: 0.1.0
  title: User Configuration Service
  description: Java Micro Services API for CBRE 360.
  termsOfService: 'http://swagger.io/terms/'
  contact:
    name: PHOENIX Team
    email: CBRE360Phoenix@evry.com
    url:
'https://cbremb.atlassian.net/wiki/spaces/NDY/pages/784532442/Java+Microservices'
  license:
    name: Apache 2.0
    url: 'https://www.apache.org/licenses/LICENSE-2.0.html'
servers:
  - url: 'https://dev.cbrehost.com:8080'
    description: Development Server
    variables:
      scheme:
        description: 'The Java Micro Services API is accessible via https and http protocol'
        enum:
          - https
          - http
        default: http
      hostname:
        description: 'The host name on which the server is running'
        default: localhost
      port:
        description: 'The port number on which the server is running'
```

```
      enum:
        - '8080'
        - '80'
      default: '8080'
  - url: 'https://qa.cbrehost.com:8080'
    description: Staging Server
    variables:
      scheme:
        description: 'The Java Micro Services API is accessible via https and http protocol'
        enum:
          - https
          - http
        default: http
      hostname:
        description: 'The host name on which the server is running'
        default: qa.cbrehost.com
      port:
        description: 'The port number on which the server is running'
        enum:
          - '8080'
          - '80'
        default: '8080'
  - url: 'https://cbrehost.com:8080'
    description: Production Server
    variables:
      scheme:
        description: 'The Java Micro Services API is accessible via https and http protocol'
        enum:
          - https
          - http
        default: http
      hostname:
        description: 'The host name on which the server is running'
        default: cbrehost.com
      port:
        description: 'The port number on which the server is running'
        enum:
          - '8080'
          - '80'
        default: '8080'
paths:
  '/userConfiguration':
    post:
      description: User Configuration Creation.
```

```yaml
operationId: save
parameters:
  - in: header
    name: API-Version
    schema:
      type: string
      format: string
    required: true
requestBody:
  description: Request Body for User Configuration
  required: true
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/UserConfigurationRequest'
responses:
  '201':
    description: User Configuration Created
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/UserConfiguration'
  '400':
    description: Bad Request
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
  '401':
    description: Not authorized to create User Configuration
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
  '550':
    description: Custom runtime errors
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
  default:
    description: Unexpected Error
    content:
      application/json:
```

```
        schema:
          $ref: '#/components/schemas/Error'
get:
  description: >
    Fetch all the Configuration details available .The details will have
    the list of BuildingIds (SBS Space Ids of Type Venue configured for the
    User),

    VendorIds ( Vendor Ids from Vendors Collection)
  operationId: findAll
  parameters:
    - in: header
      name: API-Version
      schema:
        type: string
        format: string
      required: true
  responses:
    '200':
      description: All User Configuration Details
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: '#/components/schemas/UserConfigurations'
    '400':
      description: Bad Request
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    '401':
      description: Not authorized to fetch User Configuration details
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    '404':
      description: User Configuration Not Found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
```

```
    '550':
      description: Custom runtime errors
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    default:
      description: Unexpected Error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
'/userConfiguration/{customerId}/{userId}':
  get:
    description: Fetch User Configuration based on the Customer Id and User Id
    operationId: findByIds
    parameters:
      - name: customerId
        in: path
        description: ID of Customer to which User belongs
        required: true
        schema:
          type: string
      - name: userId
        in: path
        description: ID of the User for a particular Customer to Fetch
        required: true
        schema:
          type: string
      - in: header
        name: API-Version
        schema:
          type: string
          format: string
        required: true
    responses:
      '200':
        description: Fetch User Configuration
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserConfiguration'
      '400':
        description: Bad Request
```

```
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    '401':
      description: Not authorized to fetch User Configuration details
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    '404':
      description: User Configuration Not Found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    '550':
      description: Custom runtime errors
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
    default:
      description: unexpected error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
  delete:
    description: Delete User Configuration
    operationId: removeByIds
    parameters:
      - name: customerId
        in: path
        description: Customer Id of which the User belongs to delete
        required: true
        schema:
          type: string
      - name: userId
        in: path
        description: User Id of which the User belongs to delete
        required: true
        schema:
          type: string
```

```
      - in: header
        name: API-Version
        schema:
          type: string
          format: string
        required: true
    responses:
      '204':
        description: User Configuration Deleted
      '400':
        description: Bad Request
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '401':
        description: Not authorized to delete User Configuration
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '404':
        description: User Configuration Not Found
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '550':
        description: Custom runtime errors
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      default:
        description: Unexpected Error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
components:
  schemas:
    UserConfiguration:
      type: object
      properties:
```

```yaml
    _id:
      type: integer
      format: int64
    userId:
      type: string
    customerId:
      type: string
    buildingIds:
      type: array
      items:
        type: string
    vendorIds:
      type: array
      items:
        type: string
UserConfigurations:
  type: array
  items:
    $ref: '#/components/schemas/UserConfiguration'
UserConfigurationRequest:
  type: object
  properties:
    userId:
      type: string
    customerId:
      type: string
    buildingIds:
      type: array
      items:
        type: string
    vendorIds:
      type: array
      items:
        type: string
Error:
  required:
    - code
    - message
  properties:
    code:
      type: integer
      format: int32
    message:
      type: string
```