Memoria final Proyecto API Evangelist – Restaurants API

Convocatoria de 2021

*Gema Martín Martínez –* [*gema.m2@gmail.com*](mailto:gema.m2@gmail.com)

*https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021*

# Resumen / Abstract

# Diseño del API

## Definición – buenas prácticas

# Arquitectura

## Definición técnica

## Modelo de datos

## API Manager

## API developer portal

# Seguridad

## Autenticación / Autorización del API

## OWASP Top6

## Pruebas de seguridad

## Automatización de pruebas

# Testing

## Tests de definición

## Tests unitarios y cobertura de código

## Tests de integración

RESUMEN

Se ha definido un API para la gestión de restaurantes. La idea es permitir añadir tu restaurante a un listado de restaurantes disponibles dando su información relevante y ofreciendo la posibilidad de realizar reservas y ver o añadir reseñas.

Existen diferentes roles que acotan las funcionalidades disponibles de la aplicación.

El rol administrador tendrá acceso a todos los endpoints sin restricciones.

El rol de owner de restaurantes será aquel que represente al dueño de un restaurante que quiere darse de alta en la aplicación. Podrá añadir su restaurante o darlo de baja. Además, puede consultar el listado completo de reservas, crearlas o eliminarlas. Tener acceso a las reviews para visualizarlas y a los menús disponibles.

El rol de customer representa a un usuario del restaurante al que se le permite crear o cancelar una reserva (la suya) y crear o actualizar una reseña (la suya). Además de poder visualizar los restaurantes disponibles, las reviews disponibles, los menús, etc.

Por simplificar se han omitido los endpoints de creación/borrado de menús y platos. De manera que estos se han creado mediante inserción directa en base de datos.

DISEÑO DEL API

Para definir el API se ha realizado con el estándar openAPI 3.0 y la metodología API First. Siendo el primer paso la definición del fichero [apiDefinition.yaml](https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021/-/blob/master/docs/apiDefinition.yaml) a partir del cuál se han generado las clases pertenecientes al modelo y a los controladores del código.

Buenas prácticas

* ***Nombrado de recursos***

Los recursos identificados se han nombrado con sustantivos y en plural. Estableciendo la jerarquía que les relaciona entre sí ya que las reseñas, las reservas y los menús pertenecen a un restaurante determinado.

/restaurants

/reviews

/bookings

/menus

* ***Verbos HTTP***

Los métodos HTTP definidos corresponden con la acción a realizar:

GET para obtener un listado de recursos o un recurso concreto a partir de su identificador

POST para crear nuevos recursos

PUT para la actualización completa del recurso

DELETE para el borrado del recurso a partir de su identificador

* ***Códigos HTTP en las respuestas***

Se han definido las respuestas de error que corresponden a cada método.

201 – Created

200 - OK

400 – Bad request

404 – Not found

500 – Internal server Error

401 – Unauthorized

403 – Forbidden

* ***Estandarización de respuestas***

Se ha definido una respuesta estándar para los casos de error utilizando un objeto ErrorResponse. En el caso de las respuestas de éxito se ha optado por devolver el recurso o el listado de recursos. En este último caso, se podría haber generado una respuesta estándar que permitiera separar los datos de la información de ordenación y paginación. Pero por simplicidad no se ha implementado ninguna de estas funcionalidades.

* ***Cabeceras de contenido***

Se ha definido como tipo de request/response “application/json” y se ha indicado en los métodos que corresponda.

* ***Parámetros de entrada***

En esta primera aproximación de API únicamente hemos definido parámetros en el path para identificar los recursos. No se está realizando ningún filtrado ni paginación u ordenación con lo que no se definen query parameters.

ARQUITECTURA

La implementación del API se ha realizado como una primera aproximación productiva o MVP de manera que nos ofrece una serie de funcionalidades básicas o v1, y nos permiten realizar iteraciones incrementales con nueva funcionalidad, pero estando ya disponible para su consumo.

Definición técnica

El backend del api se implementa en lenguaje de programación java 8 con el framework spring boot. Se añade el uso de spring data para gestión de entidades de base de datos.

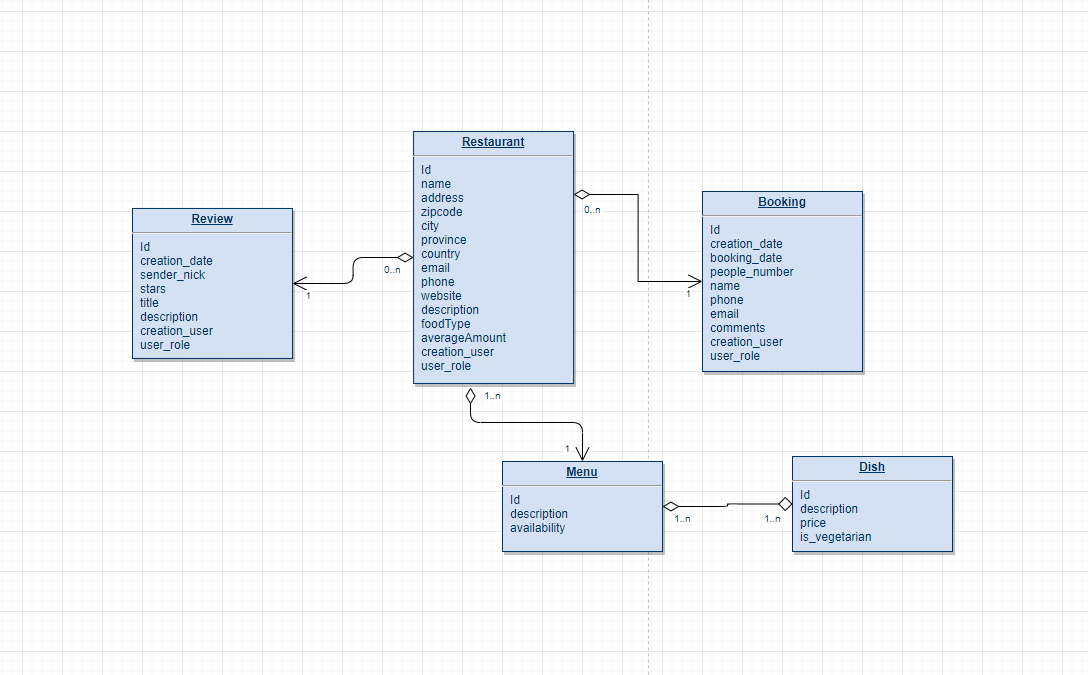
Se ha desplegado en Azure cloud como servicio y se ha configurado la base de datos con azure sql, una base de datos tipo SQL server.

*NOTA: el servicio en producción se encuentra no disponible debido a temas de pricing. Si se quiere realizar alguna prueba previamente a la fecha de la defensa se me debe comunicar para levantarlo.*

El API se ha expuesto en el API Manager de Azure y se ha dado de alta en su developer portal, en secciones posteriores entraremos más en detalle sobre la configuración de estos puntos.

Modelo de datos

El modelo de datos relacional que representa las entidades implicadas en el API se muestra a continuación:



El script de creación de la bbdd se puede ver en [db.sql](https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021/-/blob/master/src/main/resources/db/db.sql)

API Manager

Se ha desplegado el API en el API Manager de azure:

API developer portal

Se ha dado de alta el api en el azure developer portal: <https://my-api-mngr.developer.azure-api.net/>

SEGURIDAD

La implementación del API se ha realizado como una primera aproximación productiva o MVP de manera que nos ofrece una serie de funcionalidades básicas o v1, y nos permiten realizar iteraciones incrementales con nueva funcionalidad, pero estando ya disponible para su consumo.

Autenticación / Autorización del API

La autenticación de los usuarios se ha configurado utilizando el Azure active directory…

En el api manager se ha configurado oauth2…

Al backend se le envían dos cabeceras custom generadas a partir del token jwt que indican el usuario y el rol de la petición. En el backend se utilizan estas cabeceras para hacer las validaciones de acceso a los recursos de manera que se restringen actividades como borrado o actualización de recursos que no pertenecen al usuario.

Se ha configurado TLS para encriptar el canal de comunicación.

OWASP Top 6

* ***A1: Object Level Access Control***
* El Api permite obtener los recursos si y solo si previamente nos hemos autenticado
* Existen recursos que solo permiten ser accedidos por usuarios autorizados como por ejemplo el borrado de una reserva, solo puede ejecutarse por un usuario con rol administrador, owner o por el customer que creo la reserva.
* Los identificadores de los recursos son UUIDs
* ***A2: broken authentication***
* Los keys y secrets son gestionados por azure.
* Las aplicaciones deben estar autenticadas.
* Gestión de autorización a nivel de endpoint por cabeceras
* ***A3: Excessive Data exposure***
* No existen apenas recursos con información confidencial, únicamente las reservas.
* Solo los usuarios con rol admin, owner o los propios customer creadores de las reservas pueden recuperar la información de esta con los datos correspondientes.
* ***A4: Lack of resources and rate limiting***
* Se ha limitado el rating mediante el azure portal basado en clientId.
* Se ha configurado el autoescalado del servicio para no sobrepasar los limites establecidos.
* ***A5: Resource Level access Control***
* Se valida si el usuario tiene acceso a los endpoints tal y como se ha definido con anterioridad.
* El propio framework seleccionado no acepta peticiones a endpoints no implementados.
* ***A6: Mass assignment***
* La información descrita en el api es exactamente la que se mapea al objeto de entrada. El serializador ignora cualquier campo no especificado.
* Utilizando javax.validation (JSR-303) se permite hacer reject de las peticiones que no tienen el formato de parámetros de entrada esperado. (también evita A8)

Pruebas de seguridad

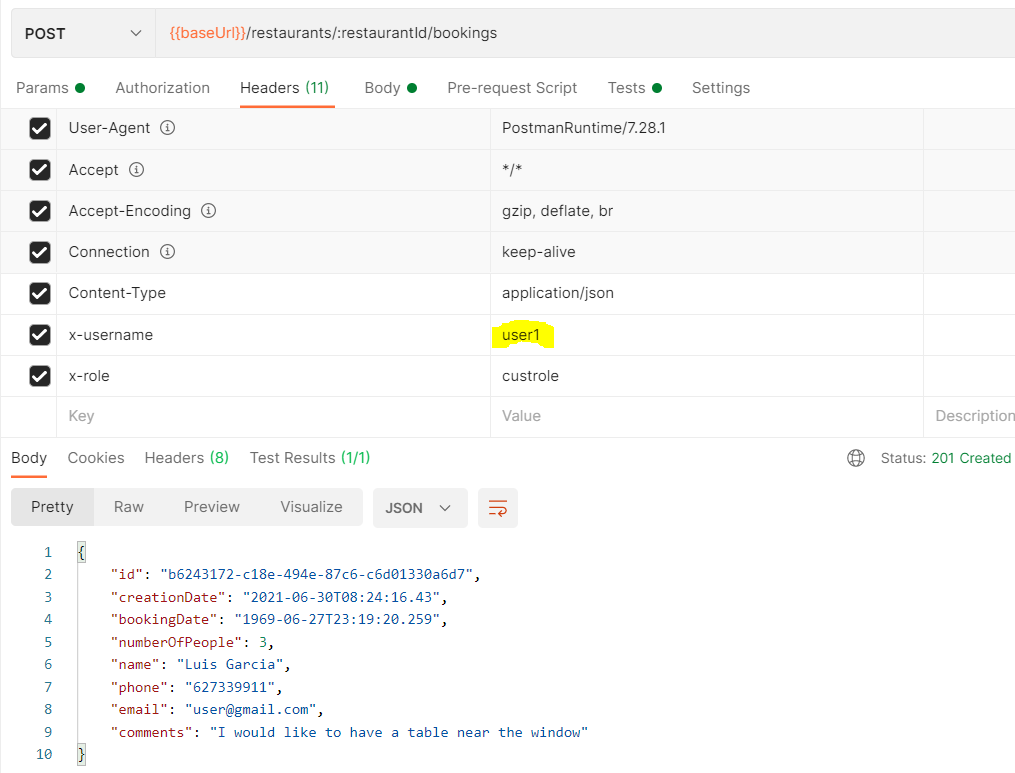
* ***Input validation***

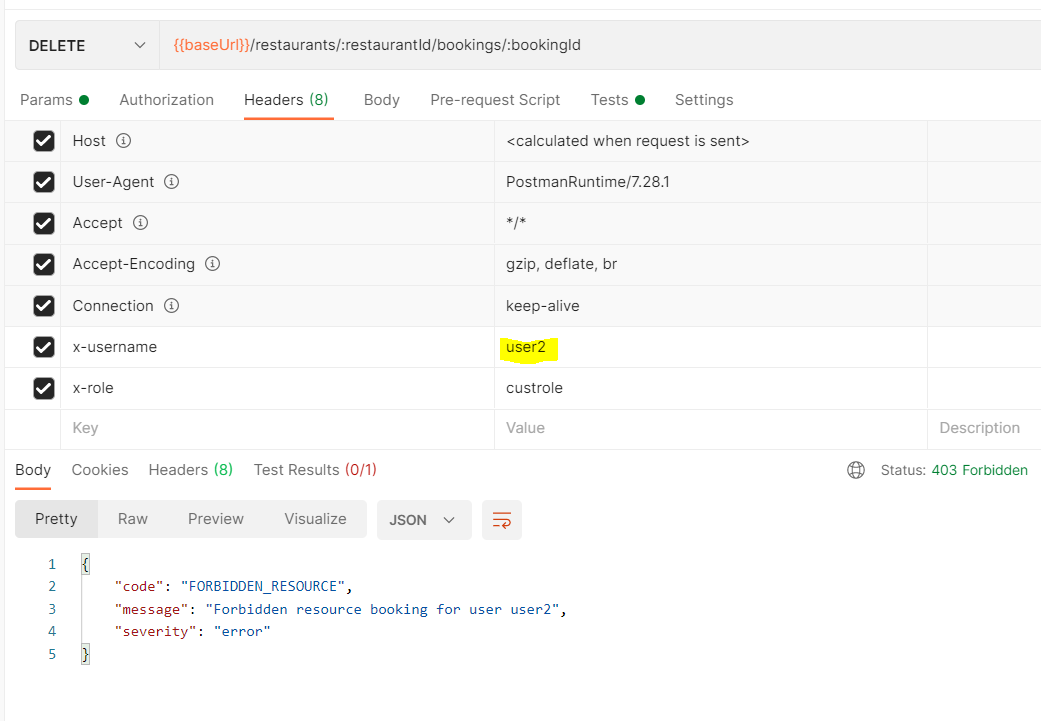
Dentro de los propios test de integración se han definido pruebas para validar cada uno de los parámetros de entrada con sus campos.

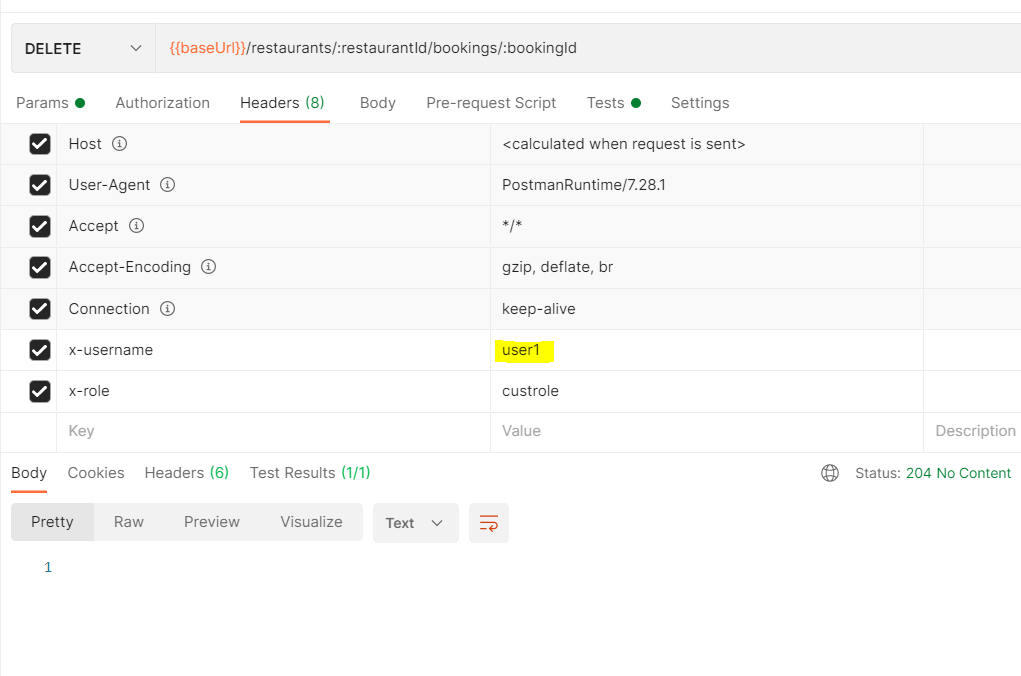
Se puede cargar la colección de postman en [/tests/it](https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021/-/tree/master/tests/it)

* ***BOLA***

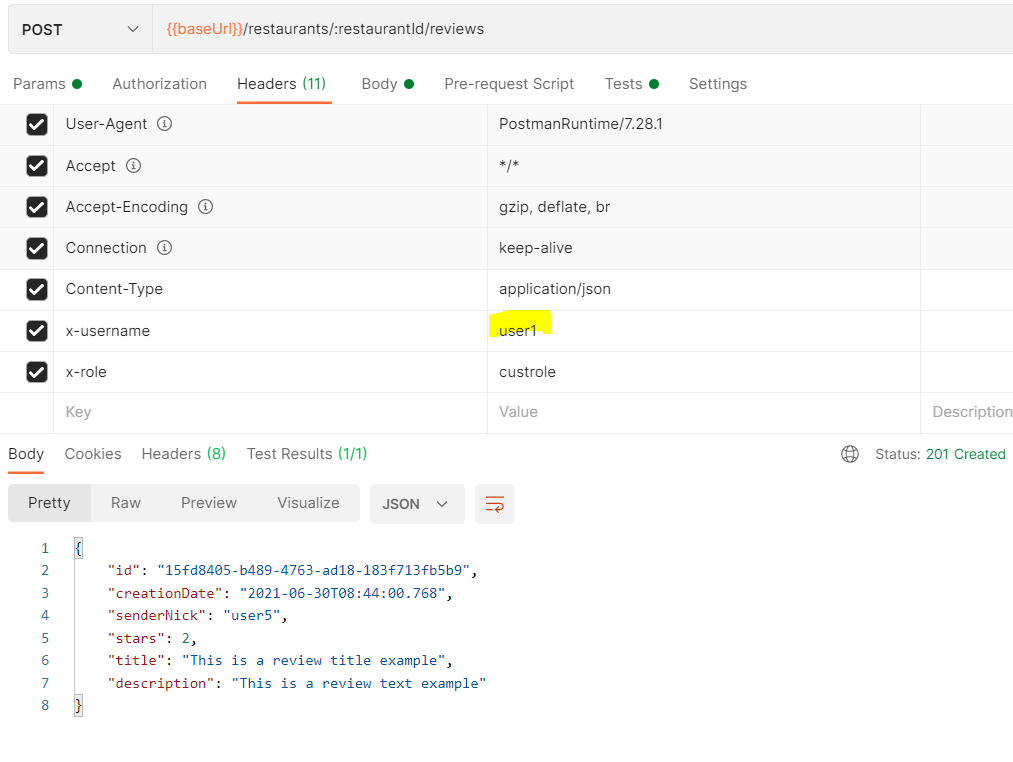
crear un recurso de tipo booking con un usuario de tipo customer. Intentar borrarlo con un usuario distinto de tipo customer -> No está permitido

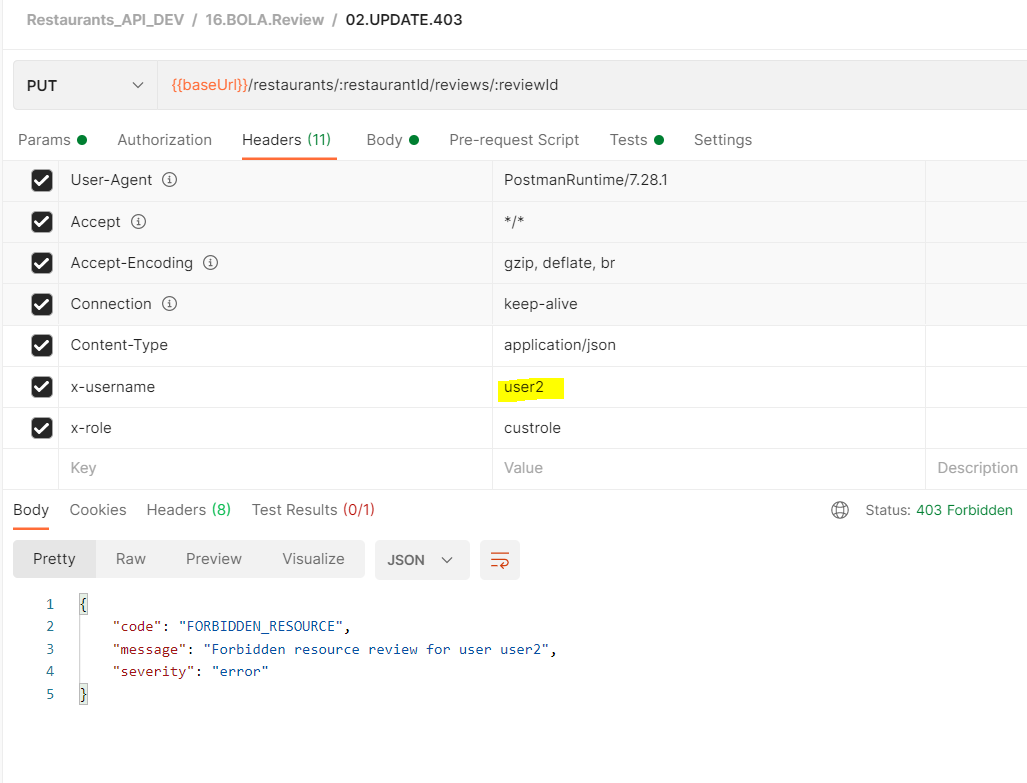


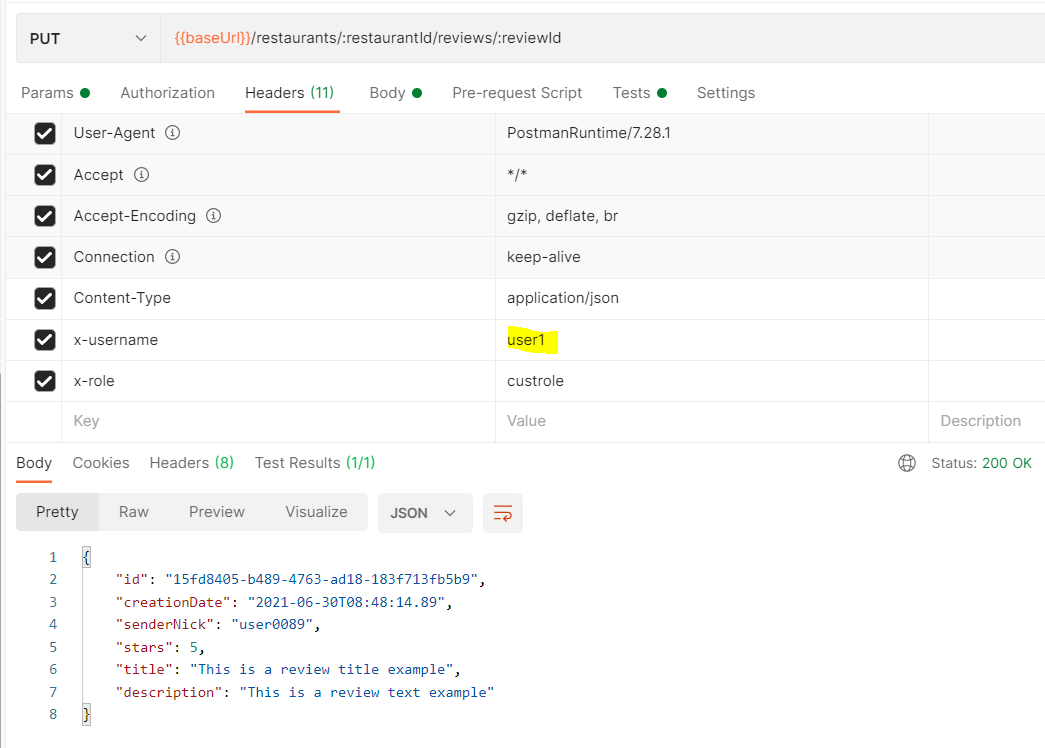




crear un recurso de tipo review con un usuario de tipo customer. Intentar actualizarlo con un usuario distinto de tipo customer -> No está permitido







Automatización de pruebas

Existen distintas maneras de automatizar las pruebas dentro del ciclo de vida de la aplicación. Personalmente he trabajado añadiendo los tests como un step dentro del flujo de CI/CD.

En el caso concreto, era un proyecto de microservicios donde los test de integración estaban diseñados con gherking/cucumber e integrados dentro del propio jar del proyecto configurados con Maven de manera que podían ejecutarse en local como parte de la build o específicamente contra un endpoint definido en configuración. Los test hacían tanto validación de los métodos definidos en el api, parámetros, etc como validación de la funcionalidad del backend (acceso a bbdd, eventos, etc).

Dentro de la configuración del pipeline de Jenkins se ejecutaba el build y el deploy del microservicio, posteriormente se lanzaban los test de integración contra el servicio ya desplegado. Si el resultado es correcto se da el pipeline por finalizado correctamente.

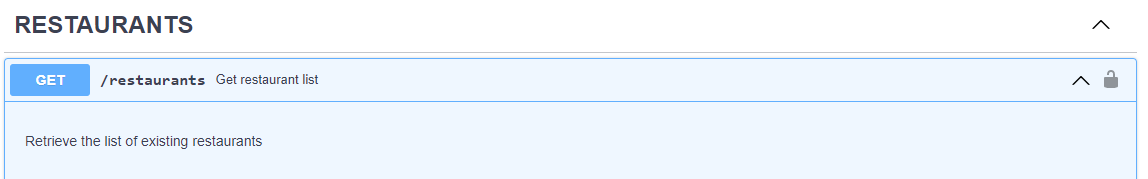
De la misma manera que en este caso se realizaban los test de integración con cucumber, se pueden realizar con Newman o muchos otros frameworks de tests.

TESTING

La implementación del API se ha realizado como una primera aproximación productiva o MVP de manera que nos ofrece una serie de funcionalidades básicas o v1, y nos permiten realizar iteraciones incrementales con nueva funcionalidad, pero estando ya disponible para su consumo.

Test de definición

Usaremos esta sección para validar la definición de cada uno de los endpoints del api.



/restaurants:

    get:

      summary: Get restaurant list

      description: >

        Retrieve the list of existing restaurants

      tags:

        - RESTAURANTS

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      responses:

        '200':

          description: Partial Content

          content:

            application/json:

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/RestaurantModel'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

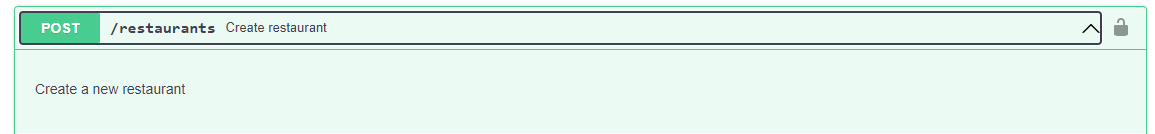
**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 401, 403, 500

**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.



post:

      tags:

        - RESTAURANTS

      summary: Create restaurant

      description: Create a new restaurant

      security:

        - OAuth2:

            - admin

            - owner

      requestBody:

        description: The information of the restaurant.

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/RestaurantRequestModel'

              description: The information of the restaurant.

      responses:

        '201':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/RestaurantModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

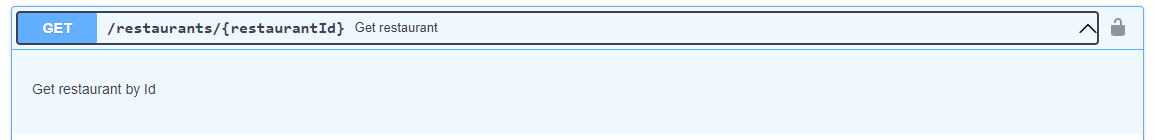
**√** Response 201, 401, 403, 500

**√** Consumes application/json

**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.



/restaurants/{restaurantId}:

    get:

      tags:

        - RESTAURANTS

      summary: Get restaurant

      description: Get restaurant by Id

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/RestaurantModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

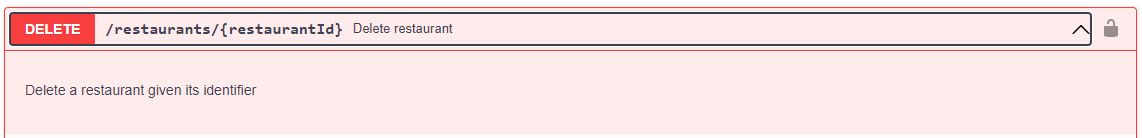
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



delete:

      summary: Delete restaurant

      description: >

        Delete a restaurant given its identifier

      security:

        - OAuth2:

            - admin

            - owner

      tags:

        - RESTAURANTS

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      responses:

        '204':

          description: Success

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

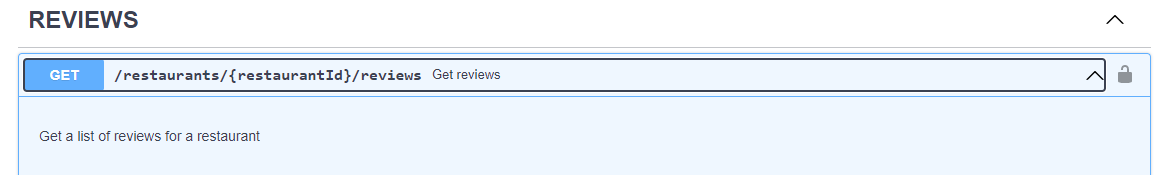
**√** Response 204, 400, 401, 403, 404, 500

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



  /restaurants/{restaurantId}/reviews:

    get:

      tags:

        - REVIEWS

      summary: Get reviews

      description: Get a list of reviews for a restaurant

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/ReviewModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

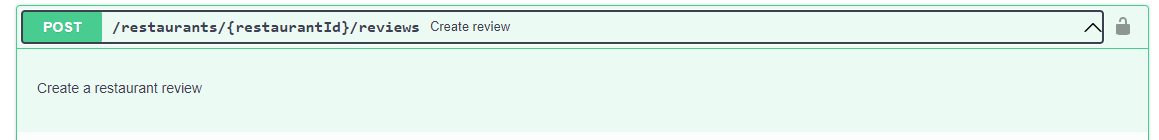
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



   post:

      tags:

        - REVIEWS

      summary: Create review

      description: Create a restaurant review

      security:

        - OAuth2:

            - admin

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      requestBody:

        description: The information of the restaurant.

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/ReviewRequestModel'

              description: The information of the review.

      responses:

        '201':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/ReviewModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 201, 400, 401, 403, 404, 500

**√** Consumes application/json

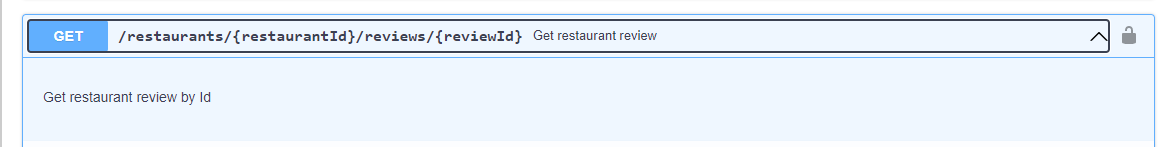
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



 /restaurants/{restaurantId}/reviews/{reviewId}:

    get:

      tags:

        - REVIEWS

      summary: Get restaurant review

      description: Get restaurant review by Id

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

        - name: reviewId

          in: path

          description: The review identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The review identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/ReviewModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



    put:

      tags:

        - REVIEWS

      summary: Update restaurant review

      description: Update restaurant review by Id

      security:

        - OAuth2:

            - admin

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

        - name: reviewId

          in: path

          description: The review identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The review identifier.

      requestBody:

        description: The information of the review.

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/ReviewRequestModel'

              description: The information of the review.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/ReviewModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

**√** Consumes application/json

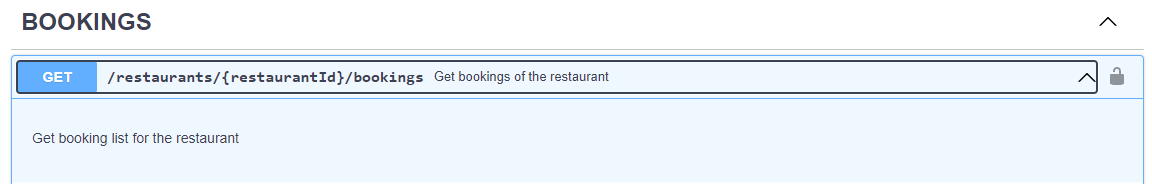
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



  /restaurants/{restaurantId}/bookings:

    get:

      tags:

        - BOOKINGS

      summary: Get bookings of the restaurant

      description: Get booking list for the restaurant

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/BookingModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

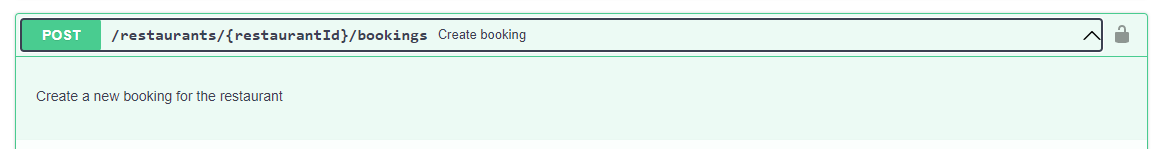
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



    post:

      tags:

        - BOOKINGS

      summary: Create booking

      description: Create a new booking for the restaurant

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      requestBody:

        description: The information of the booking.

        content:

          application/json:

            schema:

              $ref: '#/components/schemas/BookingRequestModel'

              description: The information of the booking.

      responses:

        '201':

          description: Success

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/BookingModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 201, 400, 401, 403, 404, 500

**√** Consumes application/json

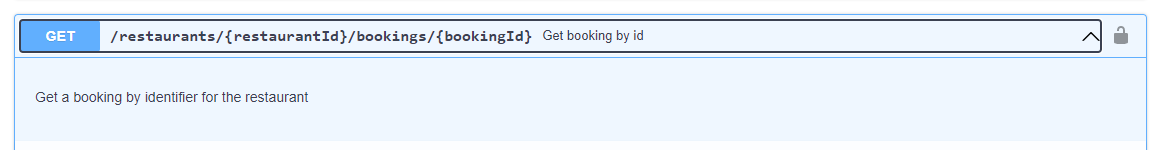
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



  /restaurants/{restaurantId}/bookings/{bookingId}:

    get:

      tags:

        - BOOKINGS

      summary: Get booking by id

      description: Get a booking by identifier for the restaurant

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

        - name: bookingId

          in: path

          description: The booking identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The booking identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                  $ref: '#/components/schemas/BookingModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

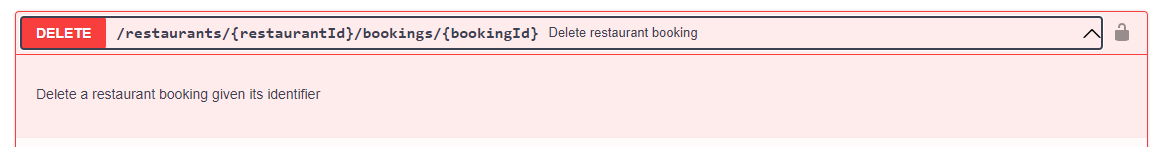
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



    delete:

      summary: Delete restaurant booking

      description: >

        Delete a restaurant booking given its identifier

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      tags:

        - BOOKINGS

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

        - name: bookingId

          in: path

          description: The booking identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The booking identifier.

      responses:

        '204':

          description: Success

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

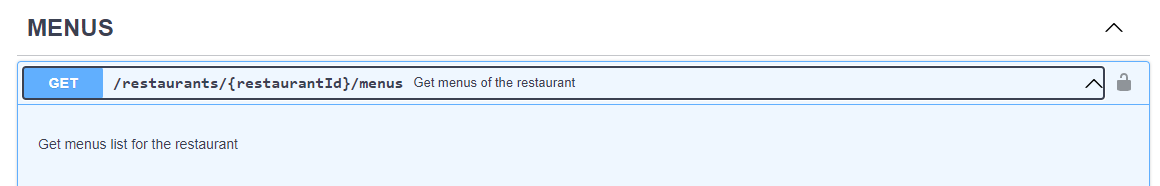
**√** Response 204, 400, 401, 403, 404, 500

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



  /restaurants/{restaurantId}/menus:

    get:

      tags:

        - MENUS

      summary: Get menus of the restaurant

      description: Get menus list for the restaurant

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/SummaryMenuModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

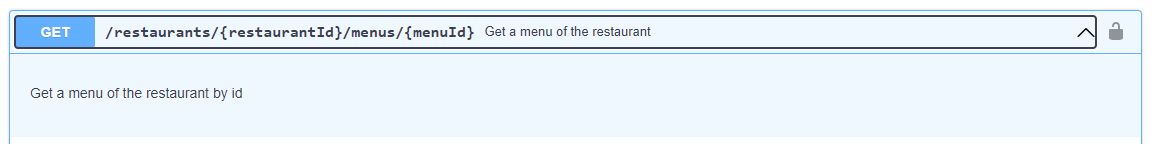
**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)



  /restaurants/{restaurantId}/menus/{menuId}:

    get:

      tags:

        - MENUS

      summary: Get a menu of the restaurant

      description: Get a menu of the restaurant by id

      security:

        - OAuth2:

            - admin

            - owner

            - customer

      parameters:

        - name: restaurantId

          in: path

          description: The restaurant identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The restaurant identifier.

        - name: menuId

          in: path

          description: The menu identifier.

          required: true

          schema:

            type: string

            format: uuid

            description: The menu identifier.

      responses:

        '200':

          description: Success

          content:

            application/json:

              schema:

                  $ref: '#/components/schemas/MenuModel'

        '400':

          $ref: '#/components/responses/bad\_request\_response'

        '401':

          $ref: '#/components/responses/unauthorized\_error\_response'

        '403':

          $ref: '#/components/responses/forbidden\_error\_response'

        '404':

          $ref: '#/components/responses/not\_found\_response'

        default:

          $ref: '#/components/responses/server\_error\_response'

**√** Ruta en plural, kebab-case, no verbos.

**√** Response 200, 400, 401, 403, 404, 500

**√** Produces application/json

**√** Security definition, should be contain (401) response.

**√** Security definition with scopes, should be contain (403) response.

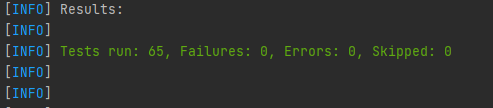
**√** Parameters required in PATH should be contain not found (404) response.

**√** Defined path parameter format UUID (validation pattern returns 400)

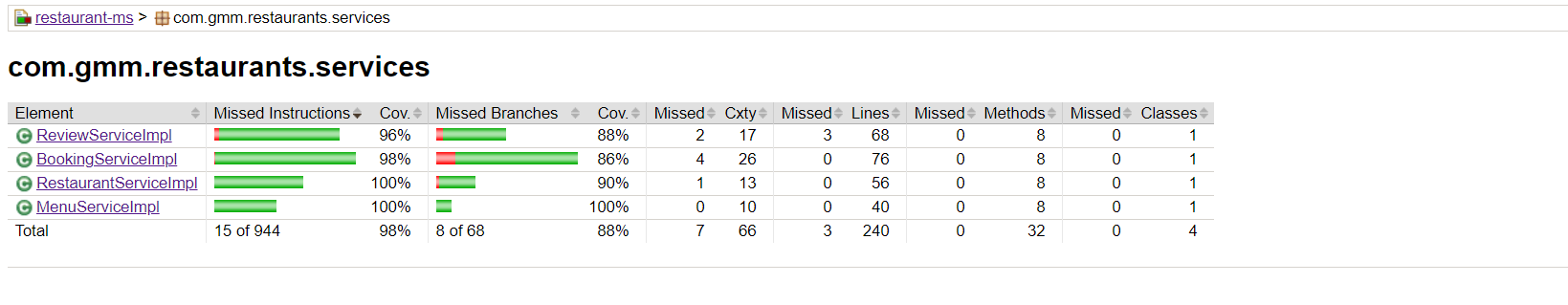
Test unitarios y cobertura de código

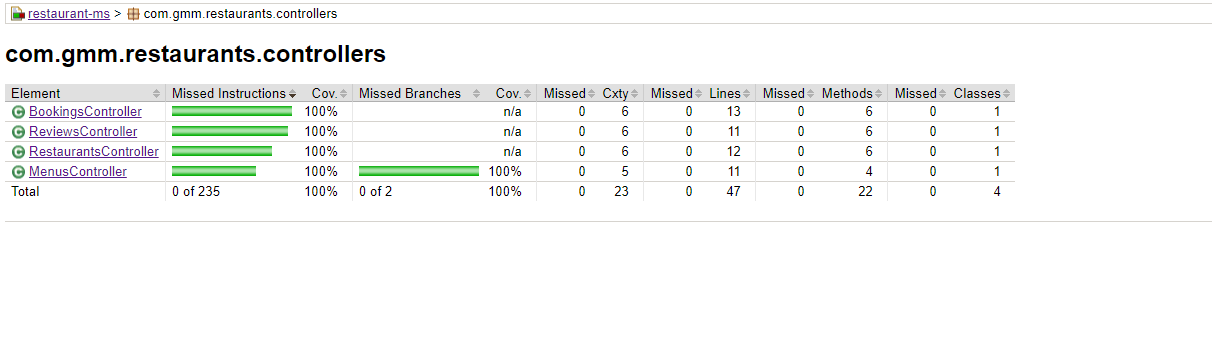
Los test unitarios se han realizado con junit y mockito y se encuentran en la carpeta [/src/test](https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021/-/tree/master/src/test) . Están asociados al plugin de Maven correspondiente de manera que se ejecutan con cada build del código.

*mvn clean install*



Además, se ha añadido la configuración del plugin de jacoco para analizar la cobertura de código dentro del servicio. Estableciendo un umbral de 80% y excluyendo las clases de configuración y modelo. El report se genera también al hacer el build de la aplicación en /target/site/jacoco/index.html.





Test de integración

Los tests de integración se han definido mediante una colección de postman que prueba cada uno de los endpoints disponibles. Validando los códigos de respuesta y los parámetros de entrada.

Se puede acceder a las colecciones de postman y a las ejecuciones de las test suite en [/tests/it](https://gitlab.com/apiaddicts1/apievangelist/projectos/propuesta-api-gema/api-evangelist-2021/-/tree/master/tests/it)

Adjunto puede verse el resumen de una ejecución contra el servicio.



