1. /products endpoint
   1. GET /products
      1. Check that GET/products API return a status code 200 with a response body contains all the products existing
      2. Check that GET/products API that total number of products as integer value
      3. Check that GET/products API apply the limit parameter by validating the length of data array is equal to the limit parameter
      4. Check that GET/products API returns the skip parameter
      5. Check that GET/products returns a status code 500 when limit parameter is a string
      6. Check that GET/products returns a status code 500 when skip parameter is a string
   2. POST/products
      1. Check that POST/products API return a status code 201 with a response body with the created product when the request body contains a name, type, upc, price, description and model
      2. Check that POST/products API return a status code 400 with an error response when the request body does not include the product name
      3. Check that POST/products API return a status code 400 with an error response when the request body does not include the product type
      4. Check that POST/products API return a status code 400 with an error response when the request body does not include the product upc
      5. Check that POST/products API return a status code 400 with an error response when the request body does not include the product description
      6. Check that POST/products API return a status code 400 with an error response when the request body does not include the product model
      7. Check that POST/products API return a status code 400 with an error response when the request body is empty
   3. DELETE/products/{id}
      1. Check that DELETE/products/{id} API return a status code of 200 with a response body of product deleted when the id is existing
      2. Check that DELETE/products/{id} API returns a status code of 404 when the id is not found for a product
   4. GET /products/{id}
      1. Check that GET/products/{id} API return a status code of 200 with a response body of the product when the id is existing
      2. Check that GET/products/{id} API returns a status code of 404 when the id is not found for a product
   5. PATCH/products/{id}
      1. Check that PATCH/products{id} API return a status code of 200 with a response body of the updated product when the id is existing
      2. Check that PATCH/products{id} API can update the product name successfully when the id is found
      3. Check that PATCH/products{id} API can update the product type successfully when the id is found
      4. Check that PATCH/products{id} API can update the product upc successfully when the id is found
      5. Check that PATCH/products{id} API can update the product price successfully when the id is found
      6. Check that PATCH/products{id} API can update the product shipping successfully when the id is found
      7. Check that PATCH/products{id} API can update the product description successfully when the id is found
      8. Check that PATCH/products{id} API can update the product manufacturer successfully when the id is found
      9. Check that PATCH/products{id} API can update the product model successfully when the id is found
      10. Check that PATCH/products{id} API can update the product url successfully when the id is found
      11. Check that PATCH/products{id} API can update the product image successfully when the id is found
      12. Check that PATCH/products{id} API returns a status code 404 and an error response when the request is sent with a not found id
2. /stores endpoint
   1. GET /stores
      1. Check that GET/stores API return a status code 200 with a response body contains all the stores existing
      2. Check that GET/stores API that total number of stores as integer value
      3. Check that GET/stores API apply the limit parameter by validating the length of data array is equal to the limit parameter
      4. Check that GET/stores API returns the skip parameter value
      5. Check that GET/stores returns a status code 500 when limit parameter is a string
      6. Check that GET/stores returns a status code 500 when skip parameter is a string
   2. POST/stores
      1. Check that POST/stores API return a status code 201 with a response body with the created store when the request body contains a name, type, zip, address, address2, city, state, lat, lng and hours
      2. Check that POST/stores API return a status code 400 with an error response when the request body does not include the store name
      3. Check that POST/stores API return a status code 400 with an error response when the request body does not include the store zip
      4. Check that POST/stores API return a status code 400 with an error response when the request body does not include the store address
      5. Check that POST/stores API return a status code 400 with an error response when the request body does not include the store’s city
      6. Check that POST/stores API return a status code 400 with an error response when the request body does not include the store’s state
      7. Check that POST/stores API returns a status code 400 with an error response when the request body include an object for an existing store
   3. DELETE/stores/{id}
      1. Check that DELETE/stores/{id} API return a status code of 200 with a response body of store deleted when the id is existing
      2. Check that DELETE/stores/{id} API returns a status code of 404 when the id is not found for a store
   4. GET /stores/{id}
      1. Check that GET/stores/{id} API return a status code of 200 with a response body of the store when the id is existing
      2. Check that GET/stores/{id} API returns a status code of 404 when the id is not found for a store
   5. PATCH/stores/{id}
      1. Check that PATCH/stores{id} API return a status code of 200 with a response body of the updated store when the id is existing
      2. Check that PATCH/stores{id} API can update the store name successfully when the id is found
      3. Check that PATCH/stores{id} API can update the store type successfully when the id is found
      4. Check that PATCH/stores{id} API can update the store address successfully when the id is found
      5. Check that PATCH/stores{id} API can update the store address2 successfully when the id is found
      6. Check that PATCH/stores{id} API can update the store city successfully when the id is found
      7. Check that PATCH/stores{id} API can update the store state successfully when the id is found
      8. Check that PATCH/stores{id} API can update the store zip successfully when the id is found
      9. Check that PATCH/stores{id} API can update the store lat successfully when the id is found
      10. Check that PATCH/stores{id} API can update the store lng successfully when the id is found
      11. Check that PATCH/stores{id} API can update the store hours successfully when the id is found
      12. Check that PATCH/stores{id} API returns a status code 404 and an error response when the request is sent with a not found id
3. /services endpoint
   1. GET /services
      1. Check that GET/services API return a status code 200 with a response body contains all the services existing
      2. Check that GET/services API that total number of services as integer value
      3. Check that GET/services API apply the limit parameter by validating the length of data array is equal to the limit parameter
      4. Check that GET/services API returns the skip parameter value
      5. Check that GET/services returns a status code 500 when limit parameter is a string
      6. Check that GET/services returns a status code 500 when skip parameter is a string
   2. POST/services
      1. Check that POST/services API return a status code 201 with a response body with the created service when the request body contains the name
      2. Check that POST/services API return a status code 400 with an error response when the request body does not include the service name
      3. Check that POST/stores API returns a status code 400 with an error response when the request body include an object for an existing store
   3. DELETE/services/{id}
      1. Check that DELETE/services/{id} API return a status code of 200 with a response body of service deleted when the id is existing
      2. Check that DELETE/services/{id} API returns a status code of 404 when the id is not found for a service
   4. GET /services/{id}
      1. Check that GET/services/{id} API return a status code of 200 with a response body of the service when the id is existing
      2. Check that GET/services/{id} API returns a status code of 404 when the id is not found for the service
   5. PATCH/services/{id}
      1. Check that PATCH/services{id} API return a status code of 200 with a response body of the updated service name when the id is existing
      2. Check that PATCH/services{id} API returns a status code 404 and an error response when the request is sent with a not found id
4. /categories endpoint
   1. GET /categories
      1. Check that GET/categories API return a status code 200 with a response body contains all the categories existing
      2. Check that GET/categories API that total number of categories as integer value
      3. Check that GET/categories API apply the limit parameter by validating the length of data array is equal to the limit parameter
      4. Check that GET/categories API returns the skip parameter value
      5. Check that GET/categories returns a status code 500 when limit parameter is a string
      6. Check that GET/categories returns a status code 500 when skip parameter is a string
   2. POST/categories
      1. Check that POST/categories API return a status code 201 with a response body with the created category when the request body contains the name and id
      2. Check that POST/categories API return a status code 400 with an error response when the request body does not include the category name
      3. Check that POST/categories API return a status code 400 with an error response when the request body does not include the category id
      4. Check that POST/categories API returns a status code 400 with an error response when the request body include an object for an existing category id
   3. DELETE/categories/{id}
      1. Check that DELETE/categories/{id} API return a status code of 200 with a response body of category deleted when the id is existing
      2. Check that DELETE/categories/{id} API returns a status code of 404 when the id is not found for a category
   4. GET /categories/{id}
      1. Check that GET/categories/{id} API return a status code of 200 with a response body of the category when the id is existing
      2. Check that GET/categories/{id} API returns a status code of 404 when the id is not found for the category
   5. PATCH/categories/{id}
      1. Check that PATCH/categories{id} API return a status code of 200 with a response body of the updated category name when the id is existing
      2. Check that PATCH/categories{id} API returns a status code 404 and an error response when the request is sent with a not found id
5. /utilities endpoint
   1. GET/version
      1. Check that GET/version API return a status code 200 with a response body that includes the current version for the API
   2. GET/healthcheck
      1. Check that GET/healthcheck API returns a status code 200 with a response body that includes the total number of products, stores and categories