Number Generator Design Specification

CONTENT

1. Requirement Analysis
2. Assumption Points
3. Design Considerations
4. Technologies Used
5. Additional Features
6. Usability/Execution
7. Requirement Analysis

A spring boot application that generates number:

**Ask1**:

The API should write into a file /tmp/{TASK\_ID} \_output.txt in the descending order, a sequence of numbers in the decreasing order till 0, with step and start number given as input; The First API returns a task;

**API\_1**

            POST /api/generate

            {

               "Goal":"10",

               "Step":"2"

            }

**Returns:**

202 ACCEPTED

            {

                        "task":"UUID of the task",

            }

**Ask2:**

The API, has to return appropriate status SUCCESS if done or IN\_PROGRESS if task is still running;

GET /api/tasks/{UUID of the task}/status

**Returns:**

 {"result": "SUCCESS/IN\_PROGRESS/ERROR”}

**Ask 3:**

The API when called with a completed task should return the list of numbers from the file;

GET /api/tasks/{UUID of the task}?action=get\_numlist

            {

                        "result": "10,8,6,4,2,0"

            }

**NOTE**: For execution ease added one more API which returns few other information which is detailed in Section 5.

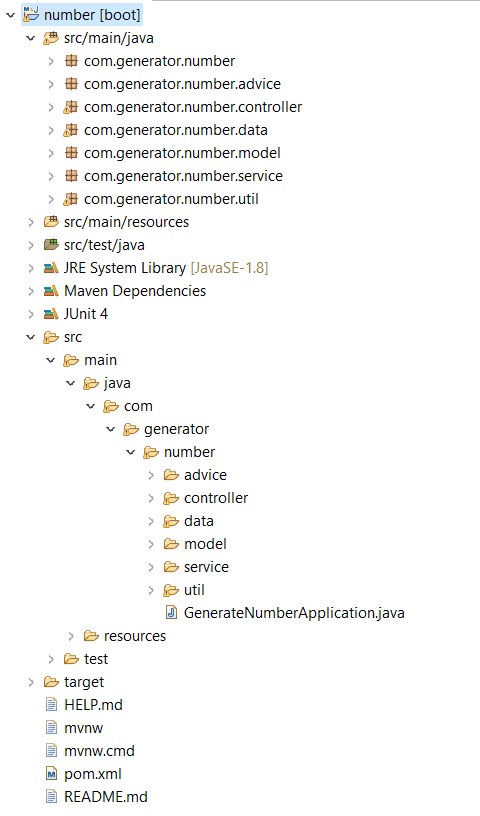
1. Assumption

UUID – Universally Unique Identifier.

File Path: From requirement, it looks like the application is expected to run on Unix, as I do not have Unix environment, I have added the output files in Java Temp directory in Windows.

1. Design Considerations

3.1 Structure



3.2 Modules Explained

**Base/start of application**: GenerateNumberApplication.java

**Model:** Holds the data model of the Application

**Controller:** is modularized for future extendibility and follow Single Responsibility principle.

**Service:** Service is called by Controller where the actual business logic falls. It calls Data layer.

**Data**: The data is written into the file. At this layer additional feature similar to caching is implemented, where the retrieval of data will not go and check output files every time. Rather the data is read from internal memory. This is to avoid read cost from output files. When task ID is not available internally it writes to or reads from the file.

**Advice:** Exceptions are handled.

**Test:** Junit for Controllers are written.

**Util:** Additional utility functions are added in util. Example Response Builder classes, Finding the operating system to write files.

1. Technologies Used

Springboot, REST APIs, Jersey, MockMvc, SpringJunit

Tools Used: STS, Postman

1. Additional Features:

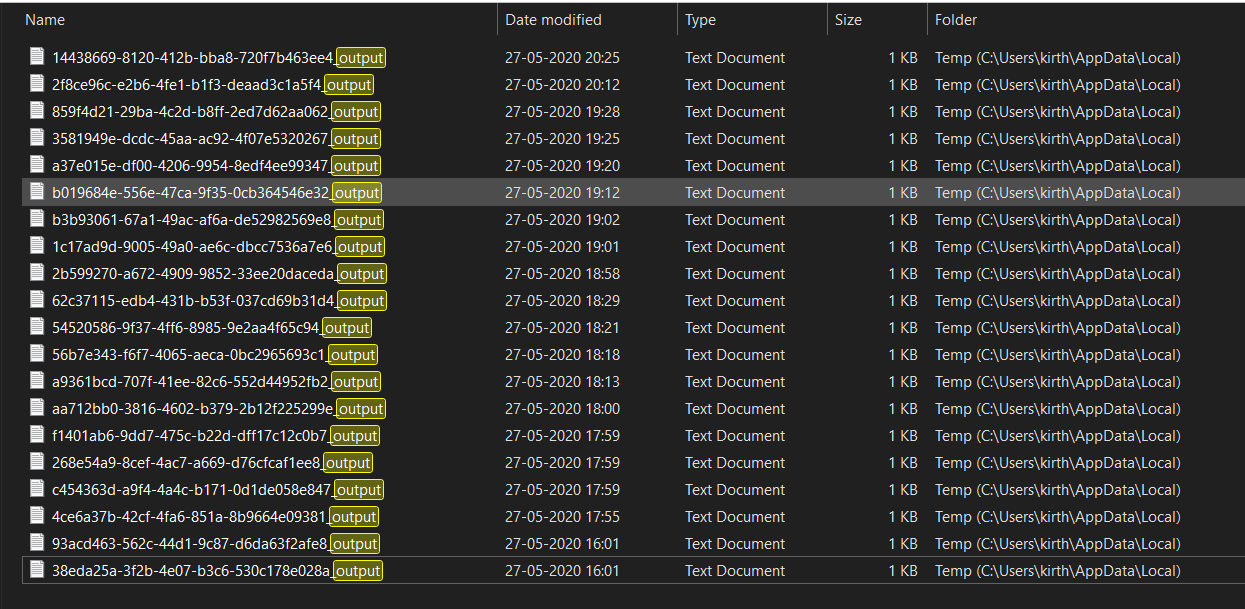
* Data caching, where application reads from internal storage of application before getting to the actual data storage, which is File system in this case.
* Since UUID is auto generated using UUID API in java, for the ease of getting UUID:

[**http://localhost:8080/api/tasks/getAllTasks**](http://localhost:8080/api/tasks/getAllTasks)This API returns all the tasks that are generated in the system.

* [**http://localhost:8080//api/tasks/getFilePath**](http://localhost:8080//api/tasks/getFilePath) This is for current implementation use, as I do not have unix environment. Where application has utility class to identify the OS, which can be extended to create right file paths.

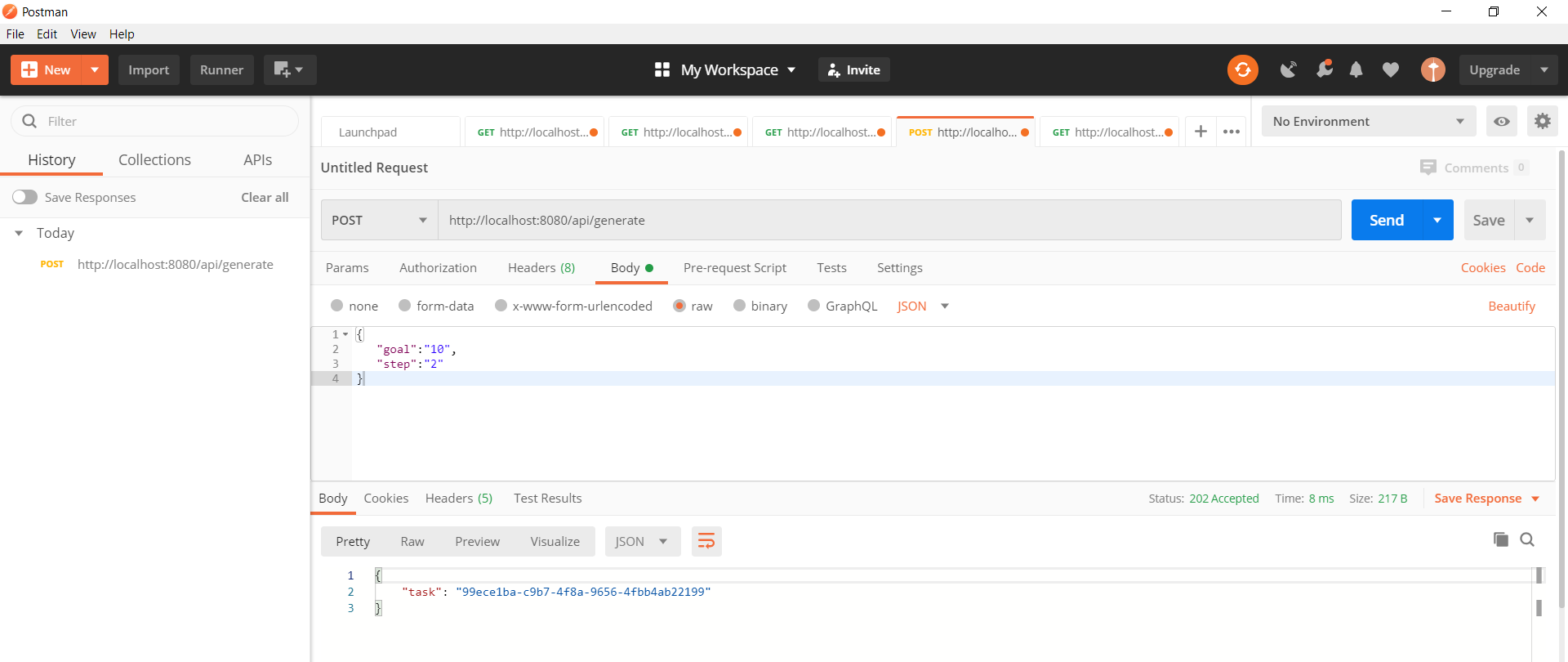
Example output:

C:\Users\kirth\AppData\Local\Temp\

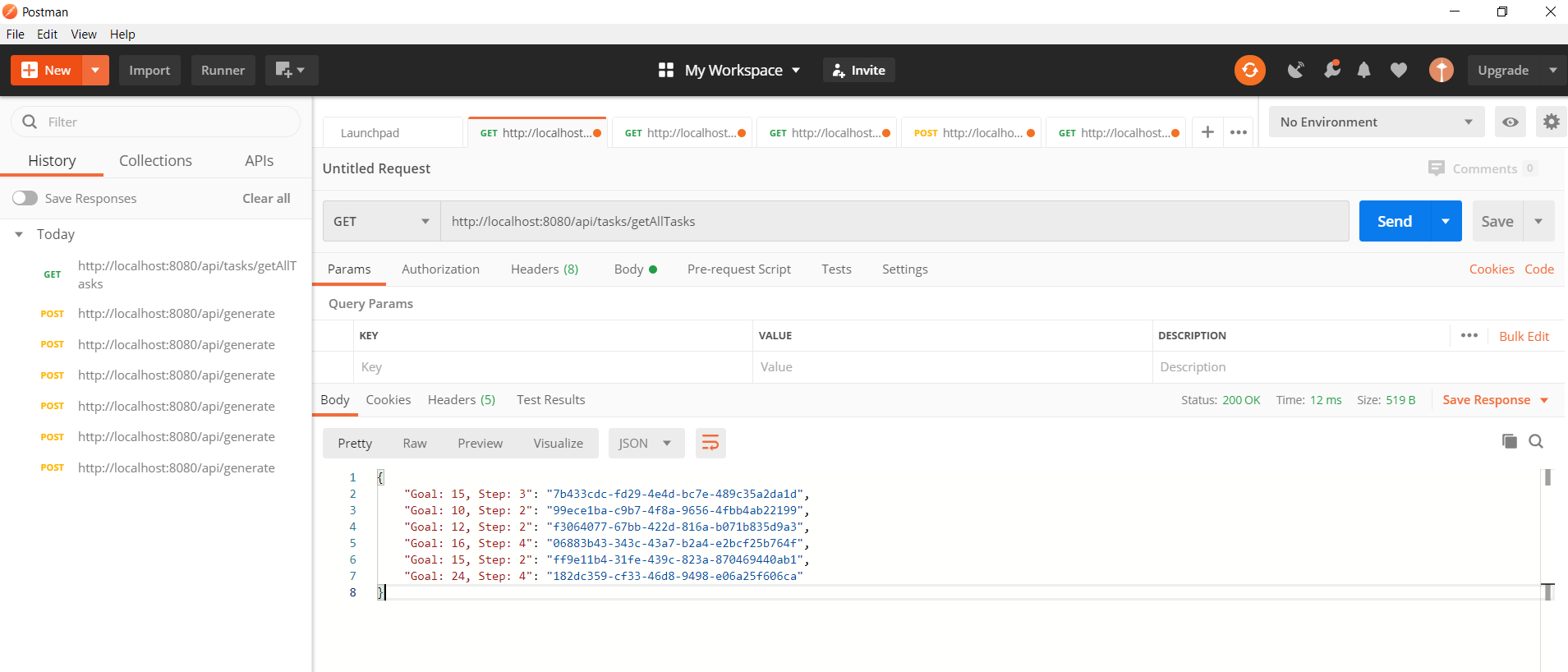


1. Execution Samples

API 1: http://localhost:8080/api/generate

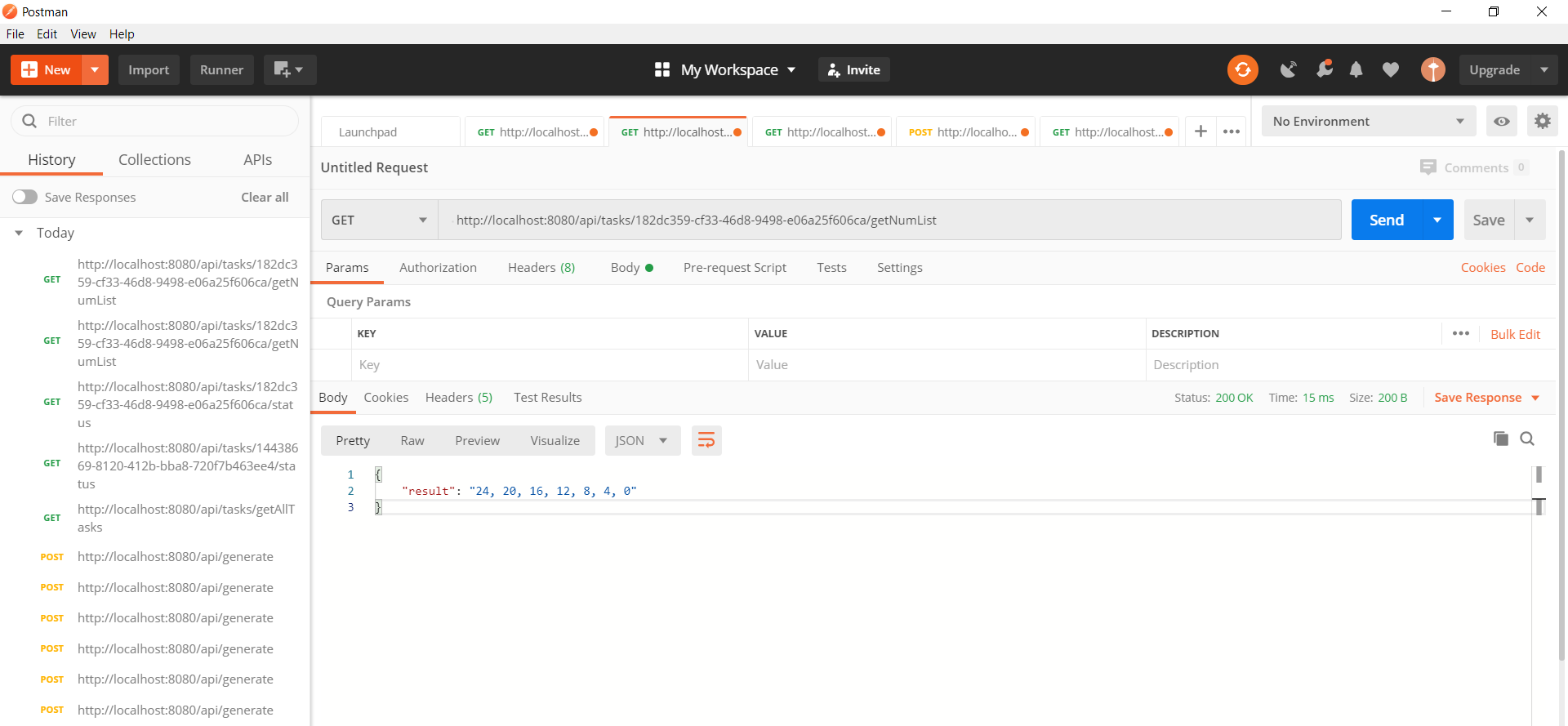


API 4: <http://localhost:8080/api/tasks/getAllTasks>

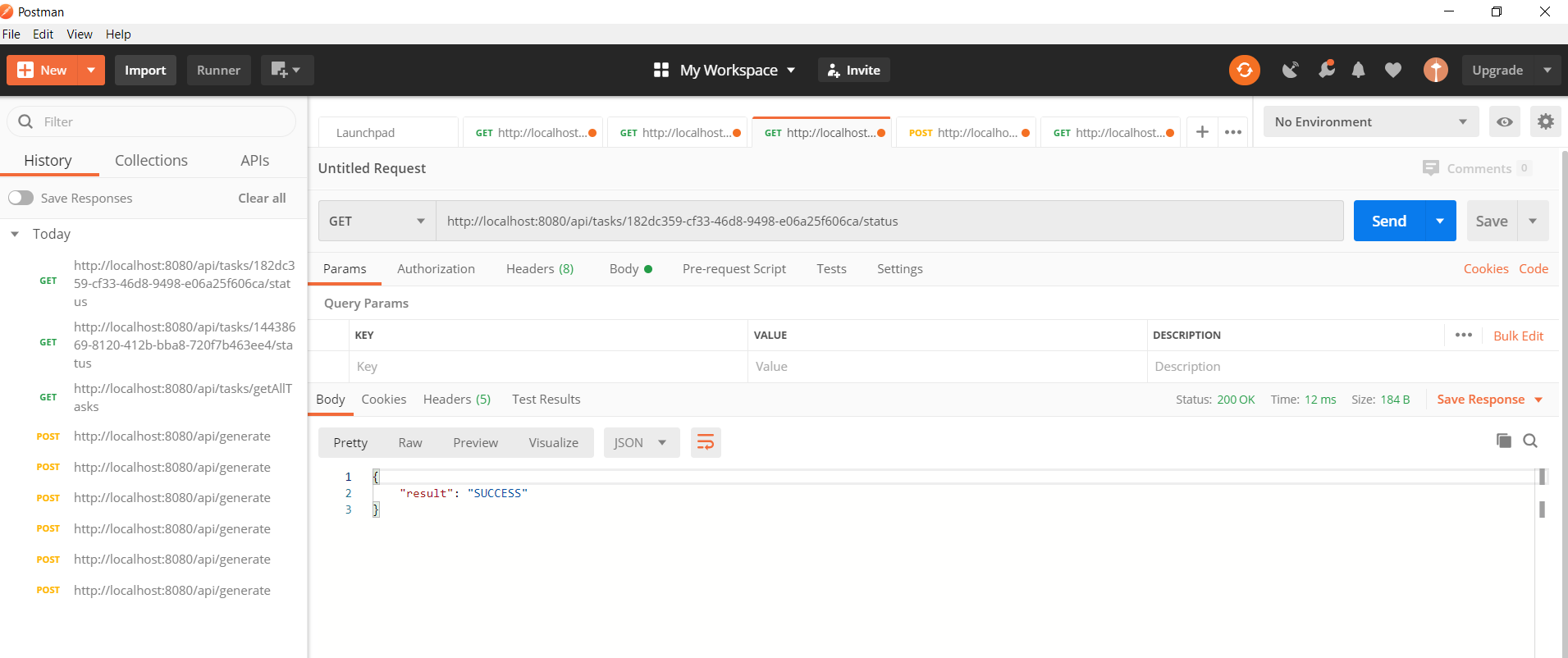


For a Goal and Step, it generates and sends back the same UUID.

API 2: http://localhost:8080/api/tasks/{UUID}/getNumList



API 3: http://localhost:8080/api/tasks/{UUID}/status



**API 5**: Get temp FilePath -> <http://localhost:8080/api/tasks/getFilePath>

