**User Guide & Test Plan**

**Kevin Chisholm**

**Andrew Rohn**

**CMSC 495 6381**

**University of Maryland Global Campus**

## Table of Contents

Table of Contents……………………………………………………..2

Implementation……………………………………………………… 3

REST API User Guide……………………………………………… 3

GraphQL API User Guide…………………………………………... 4

Test Plan…………………………………………………………….. 5

## Implementation

## The goal of our project is to illustrate the benefits of using the GraphQL API query language over that of a traditional Representational State Transfer (REST) API. In order to do this, we will be creating a database, with which both query languages will communicate. The database will be populated with values that we ourselves will enter. Both the GraphQL and REST APIs will be used to access the same or similar data so the user can see how the two work and which returns more efficient results.

## Since the GraphQL API and the REST API differ both in front-end and back-end design, a unique front and a unique back-end will need to be created for both. However, they will still access the same data from the same database. The results of both types of requests will be JavaScript Object Notation (JSON) files displayed in their respective output area. Since the user interfaces will be different for the two, they will need their own user guides.

## REST API User Guide

## The REST API defines how applications communicate with each other over the Hypertext Transfer Protocol (HTTP) and thus uses HTTP request verbs for queries. Therefore, our REST API user interface will allow the user to select one of four HTTP request verbs (GET, POST, PUT, and DELETE) via a drop-down menu in order to manipulate data in the database; GET for retrieving data, POST for adding data, PUT for replacing data, and DELETE for removing data. There will be a text field where the user can input the HTTP request query. Once an HTTP request verb has been selected and the request query has been filled, the user will be able to send the request by pressing a “Send Request” button. Afterwards, the results of that request will be displayed in a text area. An example of such a user interface can be seen below in Figure 1.

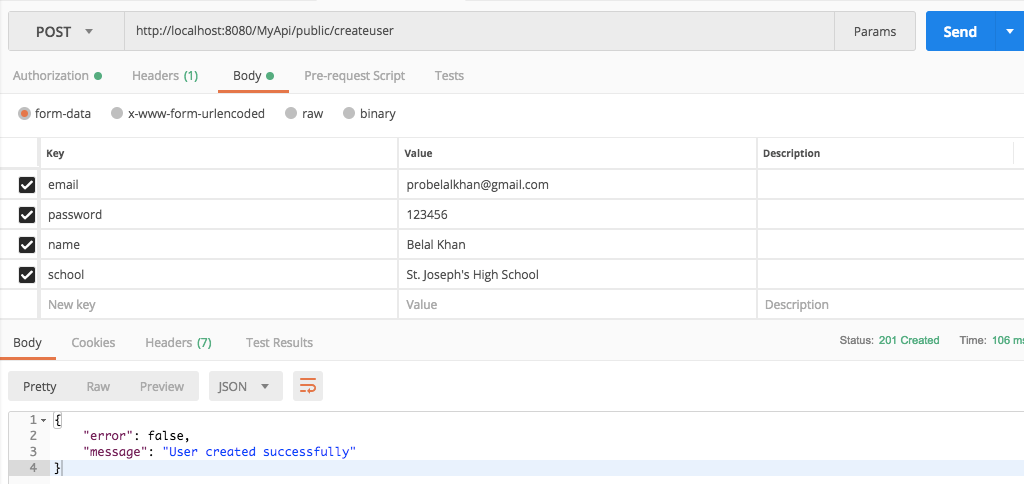


Figure 1 - REST API User Interface

**GraphQL API User Guide**

## The GraphQL API doesn’t utilize HTTP request verbs like REST API and instead uses three different operations: query, mutation, and subscription. A query operation is used for accessing data, a mutation for adding/removing/replacing data, and a subscription is used for opening a web socket to allow for dynamic request updates should values change in the database. We haven’t decided whether or not to add any functionality beyond a simple data query. Should we do so, a drop-down menu could be implemented in a similar fashion to the REST API’s HTTP request verb selection. Similarly a “Send Request” button will be pressed to process the query.

## A GraphQL request query is formatted like typical JavaScript code with curly brackets, indentations, and multiple lines. As such, a large text area is needed for the input as well as the output. An example of a GraphQL user interface can be seen below in Figure 2.

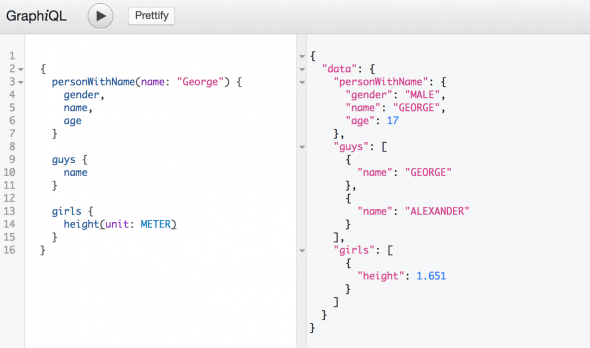


Figure 2 - GraphQL User Interface

**Test Plan**

## After the database has been populated and both the front-end and back-end designs have been completed, we will begin testing. Testing will reveal the quality of data returned from both APIs, so that the user can see the difference in the two API query languages side-by-side. We will be doing numerous test cases so that the difference between the two is evident. The user input, expected output, and actual outputs will be entered into a data table similar to the one below (see Figure 3). Screenshots will also be provided of each test case with a corresponding test case number so the difference in query results of the two API query languages can be seen side-by-side.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **User Input** | **Expected Output** | **Actual Output** |
| **#1** | **API: REST**  **HTTP Verb:**  **Query:** | (Expected Data Values) | (Actual Data Values) |
| **API: GraphQL**  **Query:** | (Expected Data Values) | (Actual Data Values) |
| **#2** | **API: REST**  **HTTP Verb:**  **Query:** | (Expected Data Values) | (Actual Data Values) |
| **API: GraphQL**  **Query:** | (Expected Data Values) | (Actual Data Values) |
| **#3…** | **API: REST**  **HTTP Verb:**  **Query:** | (Expected Data Values) | (Actual Data Values) |
| **API: GraphQL**  **Query:** | (Expected Data Values) | (Actual Data Values) |

Figure 3 - Test Case Table