Modern API Delivery

Kevin Chisholm

Andrew Rohn

CMSC 495 6381

University of Maryland Global Campus

## Table of Contents

Table of Contents 2

Problem 3

Solution 4

Project 5

Roles 5

Course Dictated Milestones 6

Project Goals 6

Technology Stack 7

Stretch Goals 7

Entity Relationship Diagram 8

Site Map 9

References 10

## Problem

Traditional REST API’s have been the de facto method to share serialized data for the use of front-end applications.  Due to its popularity, there have been many issues discussed around the use and implementation of REST APIs.

The first issue with this default method is the reliance on multiple endpoints to retrieve the information one may require. As the consumer of the API, imagine we wanted to find out what job specialization someone held, we would first have to find the specific person by hitting the user endpoint then take the relevant information from that endpoint to hit the job endpoint and search by the ID. In figures 1 and 2 we demonstrate what the API calls, and the returned data would look like.

The second issue we run into is handling the data that we receive from the API. There are best practices, but it is not sure that all APIs will follow these practices. In a front end application, we may be expecting certain types of data. In the examples provided in figures 1 and 2, the job\_id field returns a string in the first API call and a number in the second API call. In JavaScript, this may be a problem that can be worked around, but there is always the need to avoid unnecessary side-effects. In implementations such as TypeScript, this would blow up completely due to the expected data types, not corresponding to the data received.

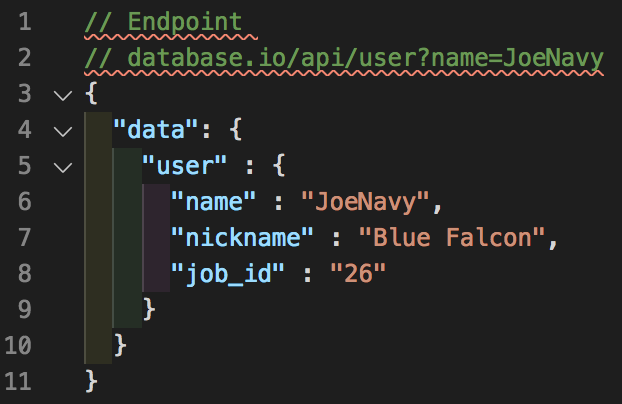


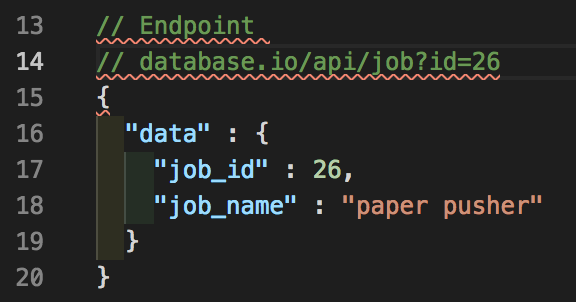
Figure 1

Figure 2

## Solution

In 2015, Facebook announced a new technology that was lauded as the replacement for traditional REST APIs, a technology known as GraphQL. GraphQL is a query language for APIs that aim to solve both of the problems listed in the section above. A GraphQL query reaches out to a single endpoint to retrieve only the requested data, solving the first problem discussed. GraphQL is also a typed system, requiring the data types of the data queried and returned to be specified, solving problem number two.

Figure 3 [1]

## Project

We will implement a REST API and GraphQL API that serves trivia questions to demonstrate the benefits of GraphQL over a traditional REST API.  The REST API portion of the application will only be used for demonstration purposes while the GraphQL API will be utilized to demonstrate our trivia question delivery service.

## Roles

Kevin Chisholm:  Project Manager, Back-end Designer, Dev ops, Programmer

Andrew Rohn:  Front-end Designer, Documentation, Testing, Programmer

## Course Dictated Milestones

Week 1: Project Goal

Week 2: Requirements Analysis

Week 3: User's Guide and Test Plan

Week 4: Design

Week 5: Coding Plan: Phase I

Week 6: Coding Plan: Phase II

Week 7: Coding Plan: Phase III

Week 8: Final Evaluation

## Project Goals

Week 1:  Project idea, group formations and role designation.

Week 2:  Database schema design, tech stack choice.

Week 3:  REST API implementation and deployment

Week 4:  GraphQL implementation and deployment

Week 5:  Front-end implementation and deployment

Week 6:  Stretch goal work

Week 7:  Final testing, stretch goal work

Week 8:  Completion of project and documentation

## Technology Stack

Front-end:  Vue (Javascript library), Blade (PHP Templating Engine)

Back-end:  Laravel (PHP framework)

Database: PostgresQL

Version control:  Git (Github)

Deployment:  Heroku

Environment:  Mac

## Stretch Goals

Create quiz game

Make game multiplayer utilizing web sockets

## Entity Relationship Diagram

Figure 4

## Site Map

* [quizmate.com](http://test.com)

- /api

- /users

- /:id

- /:name

- /questions

- /:id

- /:value

- /categories

- /:id

- /:name

- /graphql

- /game

- /leaderboard

## References

[1]. *Queries and Mutations*. graphql.org/learn/queries/.