

Cloud Web Services - 2213112

by Andrew Davies

Submission date: 05-Dec-2025 04:02PM (UTC+0000)

Submission ID: 269771420

File name: 40869_Andrew_Davies_Cloud_Web_Services_-_2213112_708807_137479730.docx (111.92K)

Word count: 1251

Character count: 7175



CLOUD BASED WEB SERVICES

Assignment 1

Andrew Davies
2213112@uwtsd.ac.uk

Contents

Introduction..... 2

REST APIs 2

GraphQL..... 3

WebSockets 3

Bibliography 4

Introduction

Web applications are now a core part of everyday life, from social media and online banking to streaming and shopping. Behind the scenes, these applications rely on smooth communication between different systems to function properly. This communication is made possible through Application Programming Interfaces (APIs), which allow software applications to exchange data over the internet. A web service is any API accessible over HTTP, and while this concept has existed for many years, earlier methods such as SOAP and WSDL were often seen as complex and difficult to work with [1].

In recent years, developers have turned to more modern and flexible methods that simplify how data is shared between clients and servers. This essay will first evaluate REST (Representational State Transfer), which is considered the architectural foundation for most web services, before discussing two modern alternatives, GraphQL and WebSockets. We will explore how each approach works, discuss their strengths and limitations, and highlight the scenarios in which each is best suited.

REST APIs

Before looking at modern variations of web service design, it's important to first understand the foundation that many of them are built upon. In simple terms, REST is a way for different computer systems to communicate with each other over the internet using standard web tools. Roy Fielding's dissertation, now considered to be one of the cornerstones of web architecture, introduces the concept of REST [1] [2]. REST builds on existing web standards such as HTTP and URIs, using the same protocols and methods already employed by the web to transfer and identify information [1]. According to Richards and Amundsen [3], RESTful design treats data as resources identified by unique web addresses (URIs) and accessed through standard HTTP methods like GET, POST, PUT and DELETE. These resources can be represented in formats such as JSON or XML, making them simple, scalable and easy to use across platforms [3].

REST's simplicity and scalability make it a popular choice for building reliable web services. Software development stacks such as MERN (MongoDB, Express, React, Node.js) often rely on REST APIs to handle communication between the front-end and the backend. The familiar web protocols used by REST allows it to easily integrate with a wide variety of tools and programming languages, making it accessible to developers and suitable for most general web applications needs.

However, as web technologies advance and applications become more complex, some limitations of REST have become more noticeable. As new protocols emerged, the term "REST is dead" appeared as a provocative headline in online discussions about modern API design [4]. This statement is somewhat misleading, as REST is still widely used and

remains the backbone of many public APIs. The discussion around this phrase highlights some of the challenges REST can face in certain contexts: over-fetching and under-fetching of data, limited support for real-time updates, and the reliance on polling or webhooks to compensate [4]. These issues become more prevalent in applications that require frequent data updates, such as data dashboards and messaging platforms.

GraphQL

GraphQL is a query-based language for APIs, which was initially developed by Facebook (now Meta) and made available to the public in 2015 [5]. One of the main selling points of GraphQL is that an end-user can request exactly the data it needs in a single query, rather than receiving extra information or needing to send multiple requests, which is often the case with REST. Furthermore, GraphQL allows multiple related pieces of data to be retrieved together, even if they come from different parts of the system [4]. For example, platforms such as Monday.com use GraphQL to gather complex and interconnected data, improving performance in data-heavy dashboards and user interfaces [6].

However, one recognised limitation of GraphQL is the additional processing required on the server-side to resolve client queries. Because the server must interpret and assemble the response each time, the performance can degrade when handling large and complex queries. A recent study comparing REST and GraphQL found that REST performed faster overall, with a 51% improvement in response time and 37% higher throughput, while GraphQL placed more processing load on the server [7]. The same study also noted that GraphQL requires developers to define queries, mutations and types before use [7]. This additional setup can introduce more development overhead in comparison to REST, especially in systems where data structures are already stable and well defined. This suggests that GraphQL is most beneficial in applications where data requirements change frequently, while REST is preferred in simpler and more predictable workflows [7].

WebSockets

WebSockets provide a different approach to data exchange by establishing a single, continuous, bidirectional connection between the client and the server [8]. This differs from REST as it allows data to be sent and received at any time without the client needing to repeatedly request updates. Once the connection has been established and is open, the server can push information updates to the client as soon as it's available. This reduces the need for techniques such as polling, where the client must continuously request new data [8]. As a result, WebSockets are commonly used for live messaging applications, notifications, and interactive dashboards. Libraries such as Socket.IO build on top of WebSockets to provide a reliable real-time messaging with automatic reconnection and fallbacks if a WebSocket connection isn't available, which helps simplify development and makes real-time features easier to implement [9].

Despite WebSockets being the ideal protocol for applications that require real-time communication, they also introduce some challenges. Because the connection remains open, the server must manage multiple active clients, which can increase resource usage and make it more difficult to scale effectively [10]. Debugging also becomes increasingly complex, due to the volume of traffic in comparison to REST's simpler request-response pattern. Therefore, while WebSockets are highly effective for continuous, live data exchange, REST or GraphQL remain more appropriate for applications and systems where data does not change frequently.

REST, GraphQL, and WebSockets each support communication between clients and servers but suit different needs. REST is widely used due to its simplicity and strong compatibility with existing web standards. GraphQL offers flexibility by allowing clients to request specific data, reducing over-fetching and under-fetching. WebSockets provide real-time, continuous data exchange, useful for live updates and interactive applications. Rather than replacing one another, these technologies are used in different contexts. The choice depends on how frequently data changes, how much control the client needs, and whether real-time responses are required.

Bibliography

- [1] E. Brown, Web Development with Node and Express : Leveraging the JavaScript Stack, O'Reilly Media, Inc, 2019.
- [2] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Dissertation, 2000.
- [3] L. R. a. M. Amundsen, RESTful Web APIs, O'Reilly, 2013.
- [4] E. P. & A. Banks, Learning GraphQL - Declarative Data Fetching for Modern Web APIs, O'Reilly Media, 2018.
- [5] The GraphQL Foundation , "The query language for modern APIs," The GraphQL Foundation, [Online]. Available: <https://graphql.org/>. [Accessed 6th November 2025].
- [6] Monday.com, "GraphQL overview," Monday.com, [Online]. Available: <https://developer.monday.com/api-reference/docs/introduction-to-graphql>. [Accessed 6th November 2025].

- [7] B. L. E. P. Armin Lawi, "Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System," *Computers MDPI*, vol. 10, no. 11, p. 138, 2021.
- [8] F. S. P. M. Vanessa Wang, *The Deinitive Guide to HTML5 WebSocket*, APress, 2013.
- [9] "Sockets.IO - How it Works," Sockets.IO, [Online]. Available: <https://socket.io/docs/v4/how-it-works/>. [Accessed 11 November 2025].
- [10] V. Chopra, *WebSocket Essentials - Building Apps with HTML5 WebSockets*, O'Reilly, 2015.

FINAL GRADE

GENERAL COMMENTS

82/100

Generic Assessment Criteria	Marks available	Marks awarded
1. Engagement with Literature Skills	Assignment 1: 5	
Good range and number of sources, mostly credible and academic. References correctly used through the text.		4.5
2. Knowledge and Understanding Skills	Assignment 1: 10	
Investigation, use-cases and advantages / disadvantages. Some good understanding from the sources demonstrated of REST, GraphQL, and websockets.		8

Some good discussion of comparisons and how best used. A little more detailed discussion of use-cases would have been beneficial.		
3. Cognitive and Intellectual Skills	Assignment 1: 30	
Design and documentation Test documentation and testing Some discussion of the implementation of the API would have been desirable. Good documentation within the readme file demonstrates good cognitive understanding. Evidence of testing should have been shown within the report.		18
4. Practical Application Skills	Assignment 1: 50	
AJAX implementation REST API		49

A nice example of a functional REST API with testing. Good example of REST endpoints, some good evidence of testing. Good use of Express for memory / server. Really good and informative readme.md PATCH request should have included the ability to use ID to change.		
5. Transferable Skills for Life and Professional Practice	Assignment 1: 5	
Presentation Reasonable presentation. Better sectioning and a conclusion would have been desirable.		2.5

