



Karelia University of Applied Sciences  
Bachelor of Information and Communication Technology

# Popular API Technologies

REST, GraphQL and gRPC

Omer Ali

Thesis, June 2024

[www.karelia.fi](http://www.karelia.fi)



**THESIS**  
**June 2024**  
**Degree Programme in Information and Communications Technology**

Tikkarinne 9  
FI 80200  
JOENSUU, FINLAND  
Tel. +350 13 260 600

Author  
Omer Ali

Title  
Popular API Technologies: REST, GraphQL, and gRPC

In the rapidly evolving landscape of software development, APIs (Application Programming Interfaces) are crucial for assembly efficient, scalable, and high-performance applications. This thesis aims to present a comparative analysis of three prominent API technologies: REST (Representational State Transfer), GraphQL (Graph Query Language), and gRPC (Google Remote Procedure Call).

By examining their design principles, use cases, and emerging trends, this study aims to guide IT developers and IT professionals in making well-informed technology choices. The analysis covers the simplicity and widespread adoption of REST, the efficient and flexible data retrieval capabilities of GraphQL, and the high-performance communication facilitated by gRPC.

Case studies on platforms such as Amazon Web Services, Microsoft Azure, Google Cloud, GitHub, Facebook, Salesforce, Shopify, and Netflix illustrated the practical benefits and implementations of these technologies. The findings underscored the importance of selecting the appropriate API technology to drive digital transformation and integration across various industries.

Language  
English

Pages  
42

Keywords  
application programming interfaces, representational state transfer, graph query language, google remote procedure call

# Contents

Abbreviations .....	4
1 Introduction .....	6
1.1 API.....	6
2 API Technologies .....	8
2.1 REST.....	9
2.2 GraphQL .....	12
2.3 gRPC.....	17
2.4 Comparison.....	19
3 Case Studies .....	21
3.1 Amazon Web Services .....	22
3.2 Microsoft Azure .....	23
3.3 Google Cloud Platform .....	24
3.4 Salesforce .....	26
3.5 Shopify .....	27
3.6 Facebook .....	28
3.7 GitHub .....	28
3.8 Netflix .....	30
3.9 Analysis.....	32
4 Discussion.....	35
5 Conclusion .....	37
References.....	39

## Abbreviations

API	Application Programming Interface.
AWS	Amazon Web Services.
AKS	Azure Kubernetes Service.
CRM	Customer Relationship Management.
CRUD	Create, Read, Update, Delete.
FQL	Falcon Query Language.
GCP	Google Cloud Platform.
GraphQL	Graph Query Language.
gRPC	Google Remote Procedure Call.
HTML	HyperText Markup Language.
HATEOAS	Hypermedia As The Engine Of Application State.
HTTP	Hypertext Transfer Protocol.
I/O	Input/Output.
IT	Information Technology.
IaaS	Infrastructure as a Service.
JSON	JavaScript Object Notation.
PII	Personally Identifiable Information.
PaaS	Platform as a Service.
RPC	Remote Procedure Call.
REST	Representational State Transfer.

SOAP	Simple Object Access Protocols.
SaaS	Software as a Service.
TCP/IP	Transmission Control Protocol/Internet Protocol.
TLS	Transport Layer Security.
URI	Uniform Resource Identifier.
UI	User Interface.
URI	Uniform Resource Identifier.
UDP	User Datagram Protocol.
XML	eXtensible Markup Language.

# 1 Introduction

In the rapidly evolving software development landscape, the significance of APIs (Application Programming Interfaces) cannot be overstated. Popular API technologies like REST (Representational State Transfer), GraphQL (Graph Query Language), and gRPC (Google Remote Procedure Call) are leading the way in architectural design, providing diverse strategies for establishing connectivity and functionality of different applications. This thesis focuses on conducting a comparative analysis of these technologies to elucidate their roles, effectiveness, and potential in shaping the future of software development.

By examining the design principles, utilization in different domains, and emerging trends associated with each API technology, this study seeks to provide developers, IT engineers, and software architects with the insights needed to make informed decisions regarding software development. This analysis not only fosters enhanced collaboration and innovation but also paves the way for a new era of integration and efficiency in software creation.

## 1.1 API

An API is a set of rules and protocols that enable software applications to communicate and exchange data, features, and functions. APIs streamline and speed up application development by enabling developers to integrate data, services, and capabilities from other applications instead of building it from scratch. They offer a straightforward and secure method for application owners to share their data and functions within their organization and with business partners or third parties. APIs also enhance security by sharing only necessary information and keeping other internal system details hidden, enabling the sharing of small, and relevant data packets for specific requests (Goodwin 2024).

According to Biehl (2015) elaborated in his book *API Architecture*, APIs are a clean, straightforward way for any software system to connect, integrate, and extend, especially when developing distributed systems with components very loosely coupled from each other. Simplicity, clarity, and ease of working with APIs is their sovereignty: they offer a reusable interface to which different applications can easily connect, but an end user does not interact with an API directly. Instead, APIs work behind the scenes and are called directly by other applications. They make it possible to have machine-to-machine communication and link various software together.

Granli et al. (2015) point out that APIs have three main parts: the public interface, a functional execution, and an overlapping layer with extras like error handling and third-party tools. The interface usually describes what functions and data structures are available, while the execution makes those descriptions a reality.

Madden (2020) says that on behalf of users, an API responds to requests from clients, web browsers, smartphone applications, and internet of things devices. After processing requests by its internal logic, the API eventually provides the client with a response. It can be necessary to communicate with additional "backend" APIs offered by processing or database systems as shown in Figure 1.

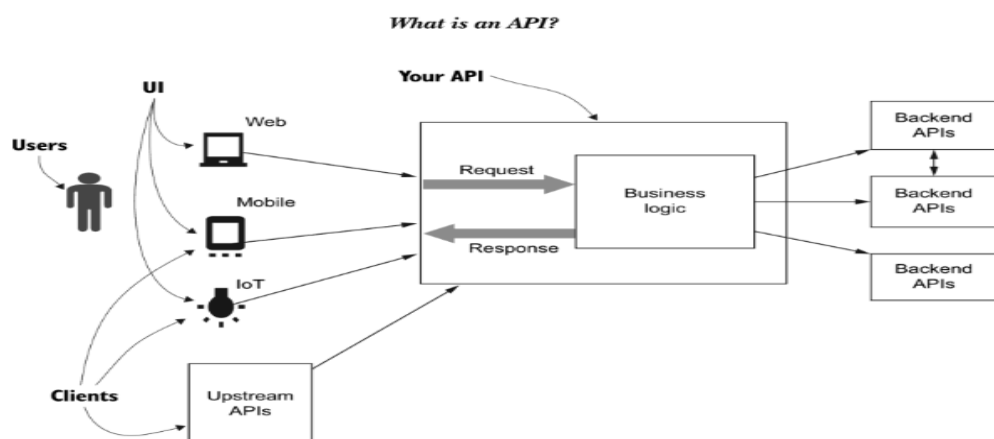


Figure 1. Schematic representation of API interactions within a digital ecosystem (Madden 2020).

## 2 API Technologies

Choosing the right API technology is crucial for building efficient, scalable, high-performance applications. In this thesis I will focus on REST, GraphQL, and gRPC because of their popularity and upward trend (Weir 2019). Despite originating in the 2000s and being regarded as old technology, 86% of developers still use it (Postman 2023). REST is lightweight, independent, scalable, and flexible. REST APIs, relying on the HTTP standard, are format-agnostic, enabling XML, JSON, HTML, and more, making them fast and lightweight for mobile apps and Internet of Things devices. They also allow for independent client-server operations, letting developers work on different areas separately and use various environments. Additionally, REST APIs offer crucial scalability and flexibility, allowing for quick scaling and easy integration (MuleSoft 2024).

GraphQL is used by 29% of developers to be efficient in data retrieval and flexibility (Postman 2023). This ability allows clients to ask for what they want, which puts off over-fetching and under-fetching of data. This minimizes bandwidth usage and improves performance, especially in mobile and complex environments with variable network conditions (Moronfolu 2024).

gRPC is used by 11% of the developers (Postman 2023). It utilizes HTTP/2 and protocol buffers for streamlined low-latency communication between servers and clients. This makes gRPC particularly suitable for microservices architectures that require quick, real-time interactions across distributed services. The use of a Protocol buffer for binary serialization enhances the speed and efficiency of data transmission, making gRPC ideal for environments where performance and resource optimization are critical (gRPC 2023; Protocol Buffers 2024).

I conducted a quick research on Google Trends (see Figure 2). Based on the latest data, REST API continues to show the highest relative search interest, highlighting its ongoing importance in the tech industry. Meanwhile, GraphQL and gRPC also display stable trends, which suggests their increasing relevance



in specific areas. This trend analysis confirms that these technologies are not fading but remain essential in modern software development. This supports previous findings from Weir (2019) that these technologies are crucial for technological advancement.

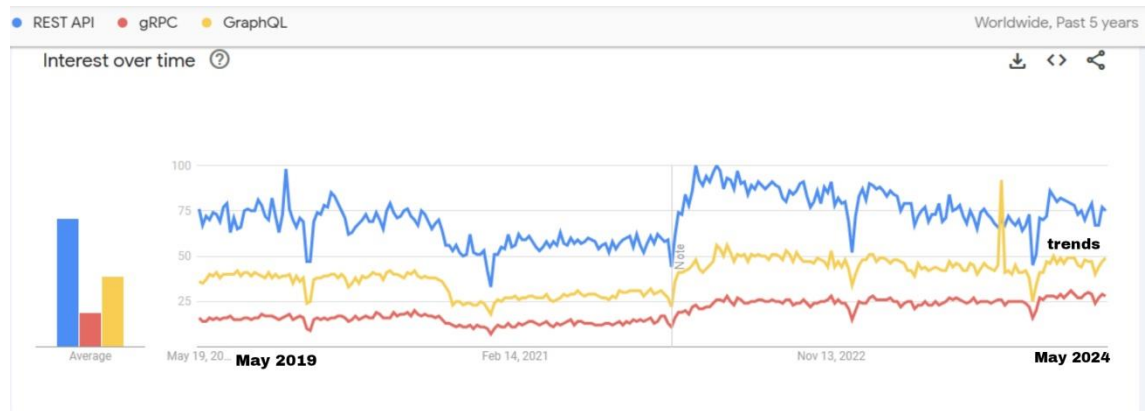


Figure 2. Google trends showing the search interest between May 2019 and May 2024 for REST API, gRPC and GraphQL.

In summary, REST provides simplicity and broad compatibility, GraphQL offers flexibility and efficient data fetching, and gRPC delivers high performance and scalability for demanding applications. These technologies collectively offer a comprehensive solution for various API needs, ensuring that developers can select the most appropriate technology for their specific requirements.

## 2.1 REST

REST is an architectural style for distributed hypermedia systems. It specifies software engineering principles and interaction constraints devised to enforce these principles. REST is also a hybrid style, which took some of the guidance from several network-based architectural styles and, on top of it, added further constraints for characterizing a uniform connector interface. This framework describes the architectural elements of REST, detailing sample processes, connectors, and data views typical of prototypical architectures; it has become one of the most popular methods for designing web-based APIs, promoting lightweight, scalable, and efficient communication between applications by

using standard data formats and already existing web technologies like HTTP. (Fielding 2000).

The essence of REST APIs lies in leveraging HTTP requests to execute a range of CRUD operations (Create, Read, Update, and Delete) on resources, each endpoint uniquely identified. A request comprises four key components: the endpoint, representing the URL structure (root-endpoint/?); the method, offering a suite of actions (GET, POST, PUT, PATCH, and DELETE); headers, serving various purposes like authentication and content information (-H or --header option); and data, the payload dispatched to a server, facilitated by option with PUT, PATCH, POST, or DELETE requests as shown in Figure 3. Together, these elements form the foundation for seamless communication and interaction with REST APIs, embodying simplicity, versatility, and efficiency (Husar 2022).

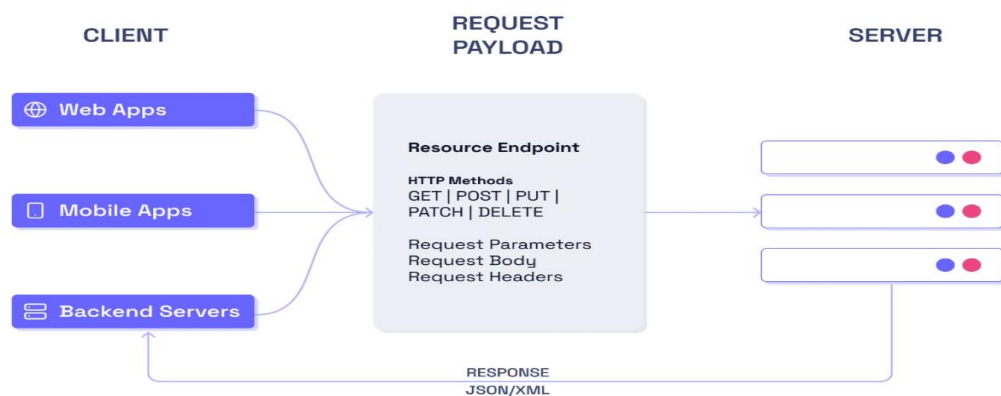


Figure 3. REST API with CRUD Operations (Salom 2023).

Gupta (2022) says that REST is an architecture approach used in networked hypermedia systems. REST has become a popular and 86% used by API developers (Postman 2023), and has been shown to work by the fact that it powers many well-known websites like Salesforce, Shopify, and Microsoft Azure, etc. REST follows rules that say how resources should be viewed and given to clients. This makes sure that the development of web services is consistent and standardized. REST makes web development easier by giving developers a way to work that is uniform and predictable.

REST architecture is based on the fundamental principle of separating the client and server components, as described by Fielding (2000). This separation enables the independent development and deployment of client-side user interfaces and server-side data storage. REST simplifies server implementation and enhances scalability. Additionally, it allows for greater portability of user interfaces across different platforms.

Another fundamental principle of REST architecture is stateless. Each client request should carry all the information; it should be independent and self-sufficient to be understood and processed by the server. State data pertinent to requests by a client should be maintained at the client and passed in every request. This statelessness helps make RESTful APIs entirely scalable and reliable. REST is an architectural style that provides a set of guiding principles; RESTful APIs are the practical implementation of the guiding principles (Makau 2023).

Caching is a mechanism employed by RESTful APIs to enhance network performance, as discussed by Fielding (2000). Through caching, RESTful APIs optimize performance and efficiency. Efficient caching reduces the need for repeated client-server interactions, thereby improving application performance. This caching mechanism is crucial for reducing latency and amplifying the overall user experience.

The uniform interface is a core principle of REST architecture. It ensures that RESTful applications consistently utilize standard HTTP methods to interact with resources. This conformity clarifies the design and implementation of RESTful systems and allows for the independent evolution of each component. The four fundamental principles of REST's uniform interface include identifying resources in requests. It manipulates resources through representations and self-descriptive messages and HATEOAS (Hypermedia As The Engine Of Application State), which is a vital part of the REST architectural style. It enables clients to navigate a web API by using hypermedia links provided in the API's response. This means that along with the requested data, the API

response also includes links to related resources and actions that the client can perform next (Gupta 2023).

## 2.2 GraphQL

GraphQL is a query language for APIs and a time that fulfills requests with existing data. It is an API description that allows the client to provide requests precisely at the point where it is needed without anymore and without any less. GraphQL makes APIs evolve over time more straightforward to manage and comes with robust developer tooling (GraphQL 2024a).

Because GraphQL queries can touch not only the properties of one resource but also follow references between them, they are more potent by design than RESTful endpoints. Apps using GraphQL request only the data needed by the view at that time, which yields much fewer network requests, making content delivery fast and efficient even under slow mobile network conditions, where typical REST APIs result in many round trips to load data from different URLs (GraphQL 2024b).

GraphQL was started internally by Facebook in 2012 to address the growing need for mobile applications. At the start of the smartphone era, devices had limited connectivity. Applications needed to make as few requests as possible to be fast and efficient. However, companies like Facebook, with rich applications and news feeds, struggled to meet these requirements because they needed multiple queries to gather all information related to a post. Facebook decided to create a new query standard that would allow them to gather all the necessary data in a single query. The need for a more efficient and flexible data-fetching API, especially for complex, high-performance mobile applications, led to the creation of GraphQL. GraphQL solves these problems by allowing customers to precisely specify the data they want (Byron & Schrock, 2015).

According to Byron (2015), Facebook needed an API that was both powerful enough to explain all of Facebook's data fetching and simple enough for their

product developers to learn and use. Facebook's mobile apps were getting harder to use and would sometimes crash because they were too complicated. They sent their news feed as HTML, and it was thought that an API data version would be useful. Facebook attempted to fix its problems with RESTful server resources and FQL (Falcon Query Language) tables, but they were unhappy with the fact that the data used in apps and server searches were not the same. Instead of resource URLs, extra keys, and join tables, they wanted an object graph with used models like JSON. A lot of code was also there for the client to understand and for the server to use when getting the data ready.

After rigorous internal use and refinement, Facebook made the technology open source which allowed developers outside Facebook to utilize and contribute to this technology. GraphQL's release marked a significant shift in the way APIs are designed and focused on giving clients the power to dictate the structure of the responses they receive (GraphQL 2024c).

Spasev et al. (2020) talk about how GraphQL could change the way applications are built. They show how GraphQL is different from the popular REST architecture by sometimes cutting the size of JSON by over 90%, which is a huge edge in today's API Technologies. In REST, clients get all the data that goes with an endpoint. GraphQL, on the other hand, only gives the fields that the client asked for, which saves time and data that would have been wasted. GraphQL also solves the issue of over- and under-fetching by getting all the data it needs in a single call, made possible by the ability to nest fields in the query. The writers do warn, though, that before adopting GraphQL, one should carefully think about whether it fits the goals and architecture of your application.

According to Bell (2023a), GraphQL is a way to ask for data from APIs that was created by Facebook engineers in 2015. It has become popular surrounded by developers, especially those working on big web applications. GraphQL is seen as a better option compared to traditional RESTful APIs. It is based on Graph Theory, which is about how networks of objects (nodes) work together. GraphQL makes asking for data easier by allowing specific and clear requests.

Just like telling a friend exactly what plans you have, developers can tell the API exactly what data they need using variables and filters, getting just the right response.

The API's capabilities are defined by the SDL (Schema Definition Language), which is a key component of the GraphQL design. For defining the types, fields, queries, mutations, and subscriptions that make up the API, it offers a simple syntax. With the help of SDL, developers may create a coherent schema that precisely represents the data model and operations that the API supports (Cocca 2023; Bell 2023b).

To preserve data integrity and enable efficient communication between clients and servers, key Points from the Document on GraphQL Schema Types, query, and mutation types are essential within a GraphQL schema. Scalar types are basic data types representing leaves of the query, such as Int, Float, String, Boolean, and ID, and custom scalar types like Date can be defined. Enumeration types, also called Enums, are restricted to a set of allowed values, ensuring type validation and communication of finite values. Types can be modified to be list arrays or non-nullable to ensure data validation, and these modifiers can be combined for complex validation requirements. Interfaces are abstract types that include a set of fields to be implemented by other types, useful for querying fields common to multiple types. Union types, like interfaces but without shared fields, are useful for returning one of several diverse types in a query, requiring inline fragments for querying specific fields. Input types allow passing complex objects as arguments, which is particularly useful for mutations where entire objects need to be created or modified (GraphQL 2024a).

Queries are the primary method to retrieve data from a GraphQL API. It gives clients the ability to indicate exactly which fields and their associations they wish to retrieve. Clients can request layered data structures with GraphQL queries, reducing the amount of data that is over- and under-fetched. To retrieve data from underlying data sources, the server uses resolver functions to resolve each field to its matching value while executing queries. With the help of this

method, consumers can obtain the exact data they require in an efficient and adaptable manner (Cocca 2023; Bell 2023).



```
query GetBookDetails {
  book(id: "1") {
    title
    author {
      name
    }
  }
}
```

Figure 4. Fetching book details with GraphQL query.

Let us consider a simpler example using a GraphQL query to retrieve information about a book from a library API (see Figure 4). This query requests the title and author of a specific book, and it has the following components:

- Query name: GetBookDetails – This name helps to identify it during debugging or in logs.
- Book field: book (id: "1") - This part of the query specifies the ID of the requested book.
- Information retrieved:
  - title - title of the book.
  - author {name} - author (only the name is requested).

This example demonstrates how GraphQL enables clients to fetch precisely what data they require, in this case, just the book's title and the name of its author, avoiding unnecessary data retrieval.

Mutations allow clients to make server-side changes to data, like adding, removing, or altering resources. Mutations, in contrast to queries, can change the status of the server's data and have unintended consequences. Sequential execution of GraphQL mutations guarantees atomicity and consistency while implementing modifications. Clients can communicate with the API to carry out

CRUD (Create, Read, Update, and Delete) operations and modify data according to their needs by using mutations (Cocca 2023; Bell 2023).

Clients can receive updates when events happen thanks to subscriptions, which allow real-time connections between clients and servers. Clients can subscribe to events or data changes of interest via GraphQL subscriptions, and they will get asynchronous notifications when pertinent updates take place. This makes it possible to create real-time applications where rapid updates are essential for user engagement and experience, such as chat apps, live dashboards, and collaborative editing tools. Web sockets or other real-time protocols are commonly used in subscription implementation to enable bidirectional communication between clients and servers (Cocca 2023; Bell 2023).

Resolver functions obtain and modify data for every field in the GraphQL schema. They carry out the logic necessary to resolve field values, serving as a link between the schema and the underlying data sources. Asynchronous resolver functions enable data retrieval from several sources, such as databases, REST APIs, or additional services. Developers can enable complicated data fetching logic and integration with a variety of data sources by customizing the data fetching behavior of each field through the definition of resolver functions.

The runtime element in charge of handling and carrying out GraphQL requests is the GraphQL execution engine. Incoming queries are parsed and checked against the schema, and the execution of resolver functions is coordinated to resolve every field. By grouping queries, storing results in a cache, and reducing round trips between the client and server, the execution engine maximizes query execution. It guarantees that requests are carried out effectively and consistently, offering scalable and responsive API performance. Furthermore, to safeguard against abusive or malicious requests, the execution engine implements rate-limiting restrictions and security safeguards, which improve the GraphQL API's overall dependability and security (Cocca 2023; Bell 2023; GraphQL 2024b).



GraphQL can be complex due to its flexibility and ability to specify exact data needs, reducing over-fetching and under-fetching. It requires defining a comprehensive schema and understanding query, mutation, and subscription mechanisms. The learning curve for GraphQL is steeper than REST. Developers need to learn its syntax, schema definitions, and resolver functions. However, its powerful querying capabilities and efficiency make the investment worthwhile for many applications.

### **2.3 gRPC**

gRPC is a modern, open-source RPC framework that can run on all platforms. This allows client and server applications to communicate seamlessly and facilitates the creation of connected systems (gRPC 2023).

RPC is a framework that enables high-level communication in operating systems. It uses lower-level transport protocols such as Transmission Control Protocol (TCP) or Internet Protocol (IP) or User Datagram Protocol (UDP) to transmit message data between applications. RPC builds a client-server logical communication framework specifically designed for network application support (IBM 2023).

Google's variant of RPC is known as gRPC. Google made gRPC, an open-source RPC technology that makes it easier to make distributed systems that work well and are safe. It uses HTTP/2 for transport and Protocol Buffers for serializing messages. It also has features like authentication, load balancing, bidirectional streaming, and more. Google's internal system needed a safe and quick way for services to talk to each other over the network, which is where gRPC got its start. In the past, Google did this with a custom method called Stubby, which was the basis for gRPC (gRPC 2023).

The study by Hoang (2021) explains that gRPC works like many other RPC programs; by allowing clients to execute procedures on remote servers, abstracting the complexities of network communication. It is all about defining a service, figuring out what methods it has, what inputs it needs, and what it gives

back when you call it from far away. On the server side, gRPC sets up the service and makes sure to know how to handle calls from clients. The client, on the other hand, has something called a "stub" that knows how to talk to the server using the same methods. One good thing about gRPC is that it works in lots of different settings, from big servers at Google to your computer. This flexibility makes it easy for developers to mix and match languages depending on what they are comfortable with. Another good feature is that many of Google's tools and services now use gRPC. This means developers can quickly add Google's features to their own apps without a lot of extra work.

gRPC utilizes Protocol Buffers, commonly known as its IDL (Interface Definition Language) protocol buffer is a versatile method of serializing structured data that can be used in various applications, such as communications protocols and data storage. Developers can define the structure of their data once and then utilize generated code to effortlessly read and write structured data from different data streams and programming languages (Protocol Buffers 2024; Robvet 2023).

gRPC is enhanced on top of HTTP/2, which was designed to overcome many of the shortcomings in HTTP/1.1. HTTP/2 introduces significant changes such as multiplexing (multiple requests in a single connection), server push, header compression, and more. These features allow gRPC to build a more powerful and efficient transport layer, which is ideal for the needs of modern applications demanding high throughput and low latency (Mohan 2021).

In gRPC, services are defined in a .proto file, where you specify named functions that could be remotely known with their parameter and return type. This strict schema specification helps in generating client and server code in various programming languages, ensuring that APIs are robust and type-safe (gRPC 2022).

gRPC automatically generates client and server stubs for you from .proto files. These stubs abstract the details of remote communication. The client stub

provides the same APIs as the server, which it internally translates into gRPC calls. This simplifies the development of client-server applications (gRPC 2022).

One of the distinguishing features of gRPC is its built-in support for streaming data. gRPC supports four kinds of streaming: server streaming, client streaming, bidirectional streaming, and no streaming (simple RPC call). This flexibility allows for continuous data transmission, fitting scenarios like real-time data feeds or other dynamic interactions allying the client and server. gRPC supports several kinds of streaming:

- Unary RPCs: Single request and response.
- Server streaming RPCs: Single request followed by a stream of responses.
- Client streaming RPCs: Stream of requests followed by a single response.
- Bidirectional streaming RPCs: Streams of requests and responses where both client and server can write and read in any order (gRPC 2022)

Interceptors are a powerful feature in gRPC that allows you to run your code before and after a request is processed. This is useful for tasks like logging, authentication, and monitoring. Middleware can manipulate, redirect, or block calls based on business logic or other rules (gRPC 2022).

GRPC supports strong authentication and encrypted data transmission. Security mechanisms like SSL/TLS encryption ensure that gRPC messages are transmitted securely across networks, safeguarding data integrity and privacy in client-server interactions (gRPC 2022).

## 2.4 Comparison

To compare, I will refer to a recent summary by Loganathan (2024) (see Table 1). REST is a mature and widely adopted standard studied for its simplicity and flexibility, making it great for public APIs and simple CRUD operations. It supports various media types and scales well, but it can struggle with data inefficiency and versioning challenges. GraphQL offers client-driven data fetching to reduce over fetching and efficiently manages complex data

relationships with real-time updates. While it adds server complexity and has a steeper learning curve, it is optimal for dynamic single-page applications and complex data structures. Furthermore, gRPC delivers high performance through HTTP/2 and Protocol Buffers, excels in streaming and real-time data scenarios, and maintains strong data integrity. However, its learning curve and less flexible nature make it best suited for high-performance microservices and data-intensive operations where efficiency and reliability are paramount.

<b>Feature</b>	<b>REST</b>	<b>GraphQL</b>	<b>gRPC</b>
Maturity	High	Medium	High
Learning Curve	Low	Medium	High
Flexibility	High	High	Low
Performance	Medium	High	High
Real-time Updates	Limited	Yes	Yes
Data Fetching	Multiple requests	Single request	Defined streams
Complexity	Low	Medium	High
Ideal Use Cases	Public APIs, simple operations	SPAs, complex data, real-time	Microservices, streaming, data-intensive

Table 1. Comparison of REST, GraphQL and gRPC (Loganathan 2024).

Being a mature and straightforward technology, REST is suitable in scenarios where the essential requirement is for a broad applicability. In contrast, GraphQL excels in highly dynamic cases, like data retrieval in real time and achieving efficiency in user interaction to the maximum extent possible on the client side. For this reason, gRPC provides superior performance, efficient communication, and strong type safety for microservices architectures, as well as real-time streaming applications. (Loganathan 2024).

### 3 Case Studies

This thesis explores the use of REST, GraphQL, and gRPC in real-world applications and provides reasons for their choices. This study will help to understand the dynamics of API technologies in modern digital environments, providing insights into how they support business operations.

In my research, I decided to include world-leading products like Salesforce, GitHub, Shopify, Facebook, and Netflix, etc. I chose products of several types of CRM (Customer Relationship Management), development platforms, E-commerce, social networks, streaming services, to see how the API technologies vary for each of them. I also decided to focus on the top 3 cloud service providers, to see if the API technologies vary there too, for some other reasons. Despite the existence of over 100 cloud service providers globally, AWS (Amazon Web Services) has consistently maintained its market dominance since its inception in 2004, commanding a 31% market share as reported by Synergy Research Group in (2024), with Microsoft Azure and Google Cloud holding 24% and 11%, respectively. These platforms have significantly shaped the digital landscape, with AWS leading since its early days, favored by a broad spectrum of users from startups to large enterprises. Azure and Google Cloud have also cemented their positions with increasing adoption over the years. Salesforce continues to lead as a global CRM provider, offering advanced, customizable tools that enhance user engagement and improve operational efficiencies (Andrei et al. 2024). Shopify dominates the e-commerce sector with a 26% market share, illustrating its extensive reach and influence (Haywood 2024). Facebook is the world's most popular social media platform with 3.1 billion users worldwide (Bernhardt 2024). Furthermore, GitHub serves as the premier platform for version control and collaborative development, essential for projects of any scale with features like pull requests and code review (GeeksforGeeks 2024). In the streaming sector, Netflix is celebrated for its superior user interface and experience, commanding the loyalty of 36% of streaming service users and boasting 269.6 million subscribers worldwide a significant growth from previous years (Durrani 2024).

These selections provide a comprehensive view of current technological impacts and future trends in API usage across multiple sectors.

### **3.1 Amazon Web Services**

Amazon Web Services (AWS) is globally recognized as the most extensive and widely utilized cloud platform, hosting over 200 comprehensive services from numerous data centers worldwide. A vast range of customers, from rapidly growing startups to large enterprises and government bodies, turn to AWS for its ability to reduce costs, increase agility, and speed up innovation. AWS stands out on the market due to the variety of services offered, with a higher number of features for each one of these services. This includes fundamental infrastructure technologies, such as computing, storage, and databases, from forefront domains spanning machine learning, artificial intelligence, data lakes, and analytics to the internet of things. These offerings simplify, speed up, and result in a much lower cost of transitioning existing applications to the cloud and developing any innovative application (Amazon 2024).

Moreover, AWS is much more functionally broad than others. For example, it contains the broadest choice of purpose-built databases ever created for the best performance in making applications, which allows choosing the most suitable tools to save on expenses and optimize cost and performance (Amazon 2024).

AWS (Amazon Web Services) utilizes various API technologies, including REST and GraphQL, each selected to optimize performance, flexibility, and developer productivity across different applications and workloads. REST APIs form the backbone of many AWS services like Amazon S3 and EC2, chosen for their statelessness which ensures scalability within dynamic cloud environments, and their uniform interface simplifies client-server interactions. This makes REST ideal for public-facing services where broad compatibility and scalability are essential. Conversely, AWS employs GraphQL through AWS AppSync, which offers efficient data loading in a single request, essential for mobile apps where minimizing data transfer is critical. GraphQL's real-time update capability

through subscriptions enhances dynamic user experiences, while its strong typing reduces bugs, boosting developer productivity (AWS 2024a; b).

### **3.2 Microsoft Azure**

Microsoft Azure is a cloud computing platform focused on the development, testing, launching, and management of its applications and services through the Microsoft data center. The platform is based on integrating SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service). Compatibility with different development languages, tools, and frameworks developed by Microsoft and those created by third-party companies has also been considered. The efficiency of running Microsoft Azure lies in using virtualization technology. In its simplest form, virtualization is creating a separation of hardware from software by emulating hardware function into software, thereby creating a virtual machine. This cloud environment involves a vast array of servers and hardware such that they support the deployment and management of these virtual services (Ekuan, 2023).

Microsoft Azure utilizes REST to manage cloud resources like computing instances, storage units, and networking elements. The APIs are critical for automating tasks such as deploying and scaling applications, monitoring resources, and managing security settings. This automation capability is essential for enterprises that leverage cloud computing to enhance their scalability and flexibility in IT operations (Lamos et al. 2024).

Azure's REST is designed to handle complex, scalable operations that are typical in cloud environments. They support the seamless integration of numerous services, regardless of the underlying technology, making it easier for developers to connect services and orchestrate operations across different platforms. The use of standard HTTP methods also simplifies the development and maintenance of applications that interact with the cloud, ensuring that secure, reliable, and efficient communication is maintained (Lamos et al. 2024).

Microsoft incorporates gRPC in its AKS (Azure Kubernetes Service) to enhance pod-to-pod communications, which is essential in the microservices architecture of AKS. gRPC facilitates efficient, language-agnostic communication among microservices, streamlining deployment, management, and operation within AKS.

Microsoft Azure utilizes gRPC in AKS to improve service efficiency. gRPC's adoption of HTTP/2 features such as header compression and multiplexing multiple requests over single TCP connections optimizes network use and reduces latency. This is critical in a microservices environment where reliable, frequent inter-service communication is necessary. The gRPC's support for Protocol Buffers ensures consistent and reliable API interactions, bolstering the robustness of Microsoft's cloud applications (Newton-King et al. 2022).

For a user-centric architecture, Microsoft Azure continues leveraging REST for its versatility and wide adoption, which is crucial for user-driven interactions across numerous services. The gRPC was added to improve real-time communication capabilities in user-facing applications, ensuring fast and efficient data exchanges that are critical for user satisfaction in cloud-based services. This combination allows Azure to offer responsive, scalable, and secure cloud services that meet the needs of diverse user bases, from developers deploying applications to businesses managing vast data across global infrastructures (Newton-King et al. 2022; Lamos et al. 2024; Microsoft 2024).

### **3.3 Google Cloud Platform**

GCP (Google Cloud Platform) is a collection of cloud computing services by Google. It enables scalable, reliable, high-performance infrastructure and platform solutions specifically designed for businesses and developers. These can make it feasible for one to build, scale, and manage applications and services on a cloud. GCP offers myriad services, including computing, storage, databases, networking, big data, and machine learning, among others. These services make it easier for organizations to innovate and accelerate their digital



transformation goals. Services like infrastructure as a service, platform as a service, and serverless computing are offered on GCP, whereby users harness the advanced technologies and infrastructures of Google to scale applications and services efficiently (Google 2024a).

Google Cloud extensively uses gRPC across its Google cloud services like Bigtable, Spanner, and Pub/Sub, to manage large-scale, distributed computing efficiently. This choice is driven by gRPC's ability to offer low latency and high throughput, essential for handling vast amounts of data across Google's service infrastructure.

Google Cloud uses REST APIs to facilitate interactions between clients and servers by following the REST architectural style. REST APIs use standard HTTP methods like GET, POST, PUT, and DELETE to enable performance on resources identified by URLs. This design provides simplicity and flexibility, making it easier for developers to integrate Google Cloud services into their applications, ensuring compatibility and ease of use across various platforms and devices (Google 2024b).

Google's preference for gRPC is based on its performance-enhancing features from HTTP/2, such as header compression and multiplexing, which significantly reduce latency. Additionally, gRPC's use of Protocol Buffers enhances data transmission speeds and resource efficiency. The framework's support for multiple programming languages and its ability to handle millions of concurrent calls ensure seamless integration and scalability across Google's diverse and extensive operations (Nally 2020).

Google's use of gRPC in its services is ideal for creating a user-centric architecture that requires fast, efficient, and reliable communication across numerous services. To further enhance this, Google integrates user-focused features like caching and smarter data syncing across devices, ensuring that users have quick and seamless interactions (Google 2024a; Google 2023).

### 3.4 Salesforce

Salesforce is a suite of cloud-based solutions primarily centered on CRM (Customer Relationship Management). It integrates various functionalities across sales, service, marketing, and IT departments into a unified platform, enabling businesses to enhance their customer engagement strategies.

Salesforce's platform utilizes AI to automate and optimize processes, thereby improving team collaboration and productivity across business functions. This dynamic approach helps organizations streamline their operations, increase efficiencies, and foster closer connections with customers by providing a 360-degree view of customer interactions (Salesforce 2024a).

Salesforce integrates REST to allow external systems to connect with its CRM (Customer Relationship Management) functionalities. This includes accessing Salesforce data like customer information, sales records, and custom reports, as well as manipulating these data (creating, updating, and deleting records) directly from third-party applications. This integration capability is vital for organizations looking to synchronize their customer relationship activities across multiple platforms without manual intervention. REST is primarily due to its ease of use and ability to seamlessly integrate disparate systems. These APIs support various data formats and are known for their straightforward, resource-oriented approach, which aligns well with Salesforce's need for an interactive, flexible CRM solution that can be tailored to specific user needs (Salesforce 2024c).

Salesforce's architecture benefits from the simplicity and effectiveness of REST to seamlessly integrate various CRM functionalities, enhancing user interactions by providing a cohesive experience. Tailoring the CRM system to be more responsive to user actions through real-time data updates and seamless third-party integrations can significantly improve the user experience, making the platform more intuitive and adaptive to individual business needs (Salesforce 2024b).

### 3.5 Shopify

Shopify is an e-commerce platform that allows anyone to start, manage, and grow a business. The platform allows users to create online stores, manage sales across various channels, market to customers, and accept payments both online and in actual locations. Shopify is designed to support businesses of all sizes, from solo entrepreneurs to global enterprises, offering a range of tools and features to streamline the selling process and enhance business management. This includes customizable templates, integrated payment processing, and multi-channel sales capabilities. Shopify's cloud-based infrastructure ensures that business owners can operate their stores from anywhere while maintaining high security and reliability (Shopify 2024).

Shopify uses REST to empower developers and merchants to extend the functionalities of the Shopify platform or their online stores. These APIs handle tasks like inventory management, order processing, and customer engagement through third-party apps. The APIs are utilized to generate personalized shopping experiences through the utilization of data analytics and customer insights. Shopify values REST for its straightforward integration capabilities and scalability, which are essential in the e-commerce sector where customer demands and data volumes can fluctuate significantly. The APIs allow for efficient data handling and provide the flexibility needed to customize and expand e-commerce operations and it supports a vast ecosystem of developers and merchants who rely on these APIs to manage their stores efficiently and adapt their offerings to meet the evolving needs of their customers (Shopify 2024b).

Shopify is focused on enhancing its REST usage to support the dynamic requirements of e-commerce platforms. Enhancing API responses and streamlining processes like inventory checks and order updates can significantly improve the user experience. Incorporating real-time capabilities to instantly reflect changes in product availability and order status to provide immediate feedback to users, is an essential feature in e-commerce operations (Shopify 2024b).

### **3.6 Facebook**

Facebook is a social networking website known to everyone, where people share information and connect with other family and friends over the internet. All these were the thoughts of Mark Zuckerberg when studying at Harvard University in the year 2004. Designed first for individuals aged 13 and over, the email address, users soon became addicted to the networking site, which has now resulted in it becoming the world's most extensive network, with over 1 billion users (Facebook 2024).

Facebook, the creator of GraphQL, initially developed this technology to manage the complexities of its vast data needs across multiple platforms efficiently. It allows their developers to request exactly what is needed from the backend, reducing unnecessary data transfer, and improving loading times, particularly on mobile devices with limited bandwidth. The key reason for Facebook's adoption of GraphQL was to enhance the performance of their applications by eliminating redundant data fetch operations, thus optimizing the user experience across diverse network conditions and devices. Facebook, optimizing GraphQL to manage complex data efficiently across platforms enhances user experience by minimizing response times and data over-fetching. This approach is particularly effective in a social network environment where speed and efficiency are crucial for user engagement. Facebook is improving its architecture by continuously updating GraphQL to handle new kinds of data and interactions, ensuring the platform remains responsive and tailored to user needs (GraphQL 2024b; GraphQL 2015).

### **3.7 GitHub**

GitHub is a platform tailored for developers, providing tools that facilitate coding, collaboration, and software deployment. The software provides various features such as version control using Git, issue tracking, continuous integration, and more, allowing teams to manage and enhance their software development

projects effectively. GitHub supports both private and open-source projects, serving a wide community from individual developers to large enterprises (Carpenter 2020).

GitHub employs REST to automate and enhance workflows related to code management and collaborative software development. These APIs allow developers to programmatically create, merge, and close pull requests; manage issues; and conduct code reviews. Such automation is particularly valuable in environments that require continuous integration and deployment processes, where manual oversight can introduce delays and become a significant bottleneck. The choice of REST by GitHub stems from its adaptability and developer-friendly nature, which are ideal for a platform serving a large community of developers. REST facilitates quick integrations and real-time data exchange, which are essential in the dynamic, collaborative environment of software development. These APIs support the automation of GitHub's core functionalities, enhancing developer productivity and operational efficiency (GitHub 2022).

In addition to REST, GitHub also leverages GraphQL to increase the flexibility and efficiency of its APIs. This adoption allows developers to specify exactly what data users need, significantly reducing the amount of data transmitted over the network, for example 2. This capability is crucial for improving the performance of integrations and services that depend on GitHub's data, particularly in reducing bandwidth consumption and enhancing responsiveness. GitHub adopted GraphQL to address the inefficiencies inherent in their previous REST API implementations, which often required multiple round trips to fetch complete data sets for Example 2. GraphQL has enabled GitHub to streamline client-server interactions significantly, allowing for a more efficient data-fetching process that tailors requests to the precise needs of the user. This optimization helps to minimize latency and improve the overall user experience by ensuring that only necessary data is retrieved and processed (GitHub 2024).

GitHub optimizes its user-centric architecture by combining REST for general API interactions and GraphQL for complex queries that enhance the user

experience. By using GraphQL, GitHub allows developers to precisely fetch what they need, reducing overhead and improving the speed of the interface. This is especially beneficial in reducing the load times and improving the responsiveness of GitHub's web and mobile interfaces, directly impacting user satisfaction (GitHub 2022; GitHub 2024).

GitHub initially adopted the REST due to its alignment with familiar web standards and its use of standard HTTP methods, which simplified common tasks such as creating, retrieving, updating, or deleting data linked to GitHub functionalities like pull requests and issue tracking. However, as GitHub grew, the limitations of REST in handling large data sets became apparent, often requiring multiple requests that led to inefficiencies. To overcome these issues, GitHub introduced GraphQL in 2016, Enabling developers to specify exactly what data users want in a single request, reducing bandwidth usage and enhancing performance in complex scenarios. This shift marked a significant move towards optimizing data retrieval processes at GitHub (2016).

### **3.8 Netflix**

Netflix is a subscription-based streaming service that gives users access to a diverse range of documentaries, movies, and TV shows, which can be streamed on various internet-connected devices. The service offers different subscription plans, each determining the number of devices that can access Netflix simultaneously and the video quality, which spans from standard definition to ultra-high definition. Members enjoy ad-free viewing and have the flexibility to download titles for offline viewing on select devices (Netflix 2024).

Netflix utilizes GraphQL to provide personalized content to millions of customers worldwide. This API technology allows the streaming service to adapt queries based on user preferences and viewing history, optimizing data delivery with minimal overhead. This capability is crucial for providing a seamless streaming experience, as it ensures that users receive content tailored to their tastes without unnecessary data transfer. The primary reason for adopting GraphQL at Netflix is its ability to handle scalable solutions required for vast data requests

efficiently. This feature reduces bandwidth usage and server load, critical for maintaining performance during peak viewing times (Netflix TechBlog 2020a, b; Shtatnov 2018).

In addition to GraphQL, Netflix also uses gRPC for robust internal microservice communication within its content distribution network. This technology is pivotal in managing complex data flows and streaming high-quality video to a global audience. Netflix's use of gRPC is driven by the need for a high-performance framework capable of handling intense loads, which gRPC provides through efficient binary serialization and support for bidirectional streaming. These capabilities optimize both speed and resource usage, crucial for the streaming giant's operations. The framework's ability to support multiple programming languages and handle millions of concurrent calls enhances its integration and scalability within Netflix's architecture (Borysov & Gardiner 2021a; b).

Netflix's combination of GraphQL for front-end operations and gRPC for backend services provides a robust architecture for streaming services. Enhancing GraphQL implementations to better predict user preferences and tailor content recommendations, alongside optimizing gRPC for smoother video delivery, can significantly improve user experience. Ensuring minimal buffering and quick access to content are critical for user satisfaction in media streaming platforms (Netflix TechBlog 2024).

Netflix adopted GraphQL to improve the performance of its digital interfaces, particularly its user interfaces on mobile devices where network conditions can vary significantly. Introduced around 2017, GraphQL allowed Netflix to efficiently manage data transfers between its clients and servers. With GraphQL, Netflix could tailor requests to exact client needs, significantly reducing the unnecessary load and enhancing user experience by speeding up response times and reducing latency (Shtatnov 2018).

The transition to gRPC As Netflix continued to evolve, the company began adopting gRPC to further optimize its backend services, especially for new microservices architectures where high-performance bidirectional streaming is

crucial. gRPC, which uses HTTP/2 and protocol buffers, offers significant improvements over traditional REST-based interfaces by reducing latency and enhancing the speed of internal service communications. This was particularly beneficial for Netflix's complex workflows and vast data requirements across its global content delivery networks (Borysov & Gardiner 2021a; b).

### 3.9 Analysis

I summarize these findings in Table 2.

Product	API Technology		
	REST	GraphQL	gRPC
Amazon Web Services	✓	✓	
Microsoft Azure	✓		✓
Google Cloud	✓		✓
Salesforce	✓		
Shopify	✓		
Facebook		✓	
GitHub	✓	✓	
Netflix		✓	✓

Table 2. Which API technologies REST, GraphQL, and gRPC are utilized by various prominent companies or products to enhance their digital platforms.

Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) demonstrate the critical role of REST APIs in managing scalable and public-facing services, while GraphQL and gRPC enhance performance by enabling efficient, real-time data loading and low-latency communication. These technologies are vital for handling diverse and extensive functionalities, from computing and storage to machine learning and big data.



Salesforce's use of REST APIs underscores the importance of seamless integration and real-time data updates in customer relationship management (CRM), ensuring enhanced productivity and system responsiveness. Shopify's reliance on REST APIs for inventory management and customer engagement highlights the necessity of reliable and secure e-commerce operations.

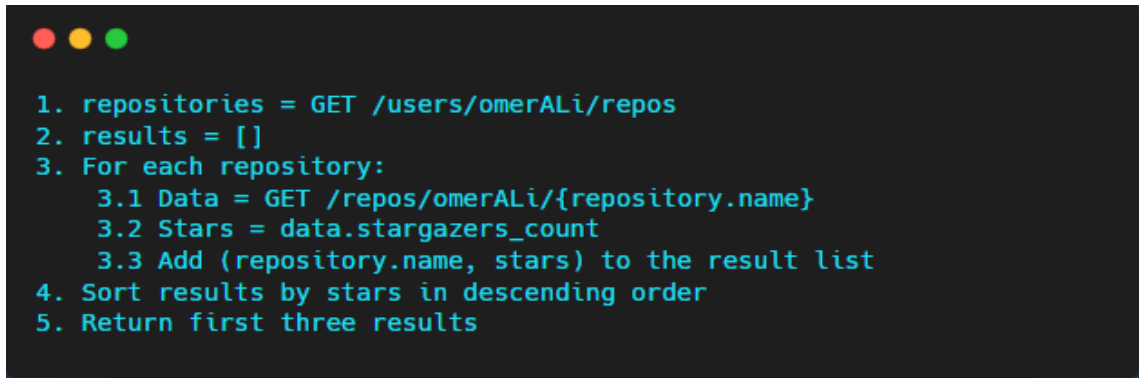
Facebook uses GraphQL to efficiently manage its vast data needs across multiple platforms. This technology enables developers to request specific data from the backend, reducing unnecessary data transfer and improving loading times, especially on mobile devices with limited bandwidth. By using GraphQL, Facebook enhances application performance, minimizes response times, and optimizes user experience across various network conditions and devices.

GitHub uses REST APIs to automate workflows, manage pull requests, issues, and code reviews, enhancing productivity in continuous integration and deployment environments. To address inefficiencies with REST, GitHub adopted GraphQL, which allows developers to request precise data, reducing data transfer and improving performance. This combination of REST for general interactions and GraphQL for complex queries optimizes GitHub's operations, enhancing the user experience and efficiency in data handling.

To understand the benefits GraphQL brings when querying data from GitHub let us look at the following examples (see Figure 5 and Figure 6).

```
1. query {
2.   user(username: "omerAli") {
3.     name
4.     repositories(first: 3, orderBy: {field: STAR, direction: DESC}) {
5.       nodes {
6.         name
7.         starCount
8.       }
9.     }
10.  }
11. }
```

Figure 5. Using GraphQL API to get a user's details and top 3 starred repositories.



```
1. repositories = GET /users/omerALi/repos
2. results = []
3. For each repository:
  3.1 Data = GET /repos/omerALi/{repository.name}
  3.2 Stars = data.stargazers_count
  3.3 Add (repository.name, stars) to the result list
4. Sort results by stars in descending order
5. Return first three results
```

Figure 6. Pseudocode using GitHub's REST API to get a user's details and top 3 starred repositories.

We observe how in Figure 5, the GraphQL query defines the username (line 2) and the data to retrieve: the name of the user (line 3), the repository names and star counts (lines 6 and 7), limiting to the top 3 (line 4). This is done in a single request to the server.

Using the REST API (see Figure 6) we first have to GET all repositories of the user (line 1), and then loop (line 3) and do separate GET requests for each repository (line 3.1). This demanding step is followed by sorting the results on the client (line 4) and returning the top 3 results (line 5).

Netflix's implementation of both GraphQL and gRPC illustrates the need for a dual approach to optimize user experience and internal communication. This combination ensures personalized content delivery with minimal latency and robust support for concurrent user calls, demonstrating how tailored data requests and efficient microservice communication can enhance streaming services.

Overall, the integration of REST APIs, GraphQL, and gRPC across these platforms highlights a strategic focus on scalability, efficiency, and user experience, highlighting best practices in cloud computing, CRM, e-commerce, social networking, software development, and streaming services. These learnings emphasize the importance of choosing the right API technologies to meet specific operational needs and improve service delivery.

## 4 Discussion

In this research, we have undertaken a comparative analysis of three pivotal API technologies: REST, GraphQL, and gRPC, each significantly influencing modern software architecture and connectivity. The choice of API technology profoundly impacts the efficiency, performance, and scalability of software applications. Therefore, understanding the strengths and limitations of each technology is crucial for developers, IT engineers, and software architects.

REST remains a cornerstone of web services due to its simplicity, scalability, and wide adoption. It utilizes standard HTTP request methods (GET, POST, PUT, and DELETE) to perform CRUD operations on resources identified by endpoint. REST's stateless nature contributes to its scalability and reliability. Additionally, REST APIs support multiple data formats, making them appropriate for a broad range of applications, including mobile and IoT devices. However, REST can become cumbersome with complex queries and relationships, heading to over-fetching or under-fetching of data, affecting performance, especially in mobile applications with limited bandwidth.

GraphQL, developed by Facebook, addresses certain limitations of REST by allowing users to specify exactly what data they require. This optimizes bandwidth usage and improves performance, particularly in mobile and complex environments. GraphQL's schema-based approach facilitates better data validation, documentation, and introspection. Despite its advantages, GraphQL can introduce complexity in implementation and might require more effort to set up compared to REST. Its flexibility can sometimes lead to performance issues if not carefully managed, as clients can inadvertently create overly complex and resource-intensive queries.

gRPC, an open-source RPC framework developed by Google, is designed for high-performance communication using HTTP/2 and protocol buffers. It best works in situations involving low-latency and high-throughput communication,

ideal for microservices architectures and real-time applications. gRPC's support for bidirectional streaming allows for continuous data transmission between client and server, useful for applications like real-time messaging and video streaming. However, gRPC has a steeper learning curve and can be more complex to implement than REST or GraphQL. Its reliance on HTTP/2 and protocol buffers means it might not be as broadly compatible with existing infrastructure and tools designed primarily for REST and JSON.

Case studies of leading technology companies like Amazon Web Services, Microsoft Azure, Google Cloud, Salesforce, GitHub, Shopify, Facebook, and Netflix demonstrate the practical applications of REST, GraphQL, and gRPC in various digital environments. These companies use multiple technologies to leverage their respective strengths: Amazon Web Services uses both REST and GraphQL, REST for simplicity and stateless interactions, and GraphQL for efficient data loading in mobile applications. Microsoft Azure utilizes both REST and gRPC, with REST managing cloud resources and gRPC enhancing pod-to-pod communications in Azure Kubernetes Service. Google Cloud employs both REST and gRPC extensively, using REST for client-server interactions and gRPC for low latency, high throughput tasks in services like Bigtable and Spanner. GitHub uses both REST and GraphQL to enhance workflow automation related to code management and collaborative software development. Shopify uses REST to manage inventory, orders, and customer interactions. Facebook utilizes GraphQL to handle vast amounts of data efficiently across multiple platforms. Netflix leverages GraphQL for personalized content delivery and gRPC for high-performance internal microservice communication.

Supporting multiple API technologies allows companies to ensure flexibility, performance, and efficiency across different application scenarios. This study, however, focused on selecting popular products of big companies. This might not be possible (cost-wise) for small companies; therefore, more research could be done in this direction.

## 5 Conclusion

This thesis provided a comprehensive comparative analysis of three popular API technologies: REST, GraphQL, and gRPC. The purpose was to guide developers and IT professionals in determining the appropriate API technology to use based on their specific needs.

The study examined the design principles, use cases, and emerging trends associated with each technology. REST was highlighted for its simplicity and widespread adoption, making it suitable for many applications due to its statelessness and reliance on standard HTTP methods. However, it can become inefficient when dealing with complex queries and enormous amounts of data.

GraphQL was praised for its ability to provide customers with precise data, reducing over-fetching and under-fetching issues. This makes GraphQL highly efficient in mobile and complex environments but introduces complexity in its implementation and potential performance management challenges.

gRPC was recognized for its high-performance communication capabilities, particularly suited for microservices architectures and real-time applications. Its use of HTTP/2 and protocol buffers allows for effective, low-latency communication, although it has a steeper learning curve and may not be as compatible with existing REST-based infrastructures.

Case studies from industry leaders such as Amazon Web Services, Microsoft Azure, Google Cloud Platform, Salesforce, Shopify, Facebook, GitHub, and Netflix showed the practical applications of these technologies. AWS employs both REST and GraphQL, Azure uses REST and gRPC, Google Cloud leverages REST and gRPC extensively, Salesforce relies on REST, Shopify uses REST for e-commerce operations, Facebook utilizes GraphQL, GitHub combines REST and GraphQL, and Netflix integrates GraphQL and gRPC.

The analysis stated that API technology application should be based on the project requirements. REST is recommended for its simplicity and broad compatibility, GraphQL for its efficiency in data retrieval and handling complex data structures, and gRPC for high-performance and real-time applications.

Future research could include broader case studies, performance benchmarks, security considerations, and a deeper analysis of developer experiences. Additionally, recent breakthroughs in artificial intelligence may play a significant role in the future.

## References

- Amazon. 2024. What are AWS? Amazon Web Services, Inc. <https://aws.amazon.com/what-is-aws>. 06.06.2024.
- Andrei, I., Ballard, B., Choudhary, U., & Williams, O. 2024. Best CRM software of 2024. TechRadar. <https://www.techradar.com/best/the-best-crm-software>. 23.05.2024.
- AWS. 2024a. What is GraphQL?. Amazon Web Services, Inc. <https://aws.amazon.com/graphql>. 06.06.2024.
- AWS. 2024b. Amazon API Gateway - AWS. Amazon Web Services, Inc. <https://aws.amazon.com/api-gateway>. 06.06.2024.
- Bernhardt, G. 2024. Top 10 most popular social media platforms. Shopify. <https://www.shopify.com/blog/most-popular-social-media-platforms>. 30.05.2024.
- Bell, H. 2023a. What is GraphQL: Definition & Uses | Noname Security. <https://nonamesecurity.com/learn/what-is-graphql>. 02.05.2024.
- Bell, H. 2023b. GraphQL Tutorials. <https://www.apollographql.com/tutorials/intro-strawberry/02-graphql-basics>. 02.05.2024.
- Biehl, M. 2015. API architecture (Vol. 2). API-University Press. 01.05.2024.
- Borysov, A., Gardiner, R. 2021a. Practical API design at Netflix, Part 1: Using ProtobufFieldMask. <https://netflixtechblog.com/practical-api-design-at-netflix-part-1-using-protobuf-fieldmask-35cfdc606518>. 20.05.2024.
- Borysov, A., Gardiner, R. 2021b. Practical API Design at Netflix, Part 2: ProtobufFieldMask for mutation operations. <https://netflixtechblog.com/practical-api-design-at-netflix-part-2-protobuf-fieldmask-for-mutation-operations-2e75e1d230e4>. 20.05.2024.
- Byron, L. 2015. GraphQL: A data query language. Facebook Engineering, Core Data, Developer Tools. 04.05.2024.
- Byron, L. & Schrock, N. 2015. GraphQL: A data query language, GraphQL.org, GraphQL Introduction. 04.05.2024.
- Carpenter, M. 2020. An introduction to GitHub. United States government <https://digital.gov/resources/an-introduction-github>. 03.06.2024.
- Cocca, G. 2023. The GraphQL API Handbook – How to build, test, consume and document GraphQL APIs. freeCodeCamp.org. <https://www.freecodecamp.org/news/building-consuming-and-documenting-a-graphql-api>. 26.05.2024.
- Durrani, A. 2024. Top streaming statistics in 2024. Forbes Home. <https://www.forbes.com/home-improvement/internet/streaming-stats>. 30.05.2024.
- Ekuan, M. 2023. How does Azure work? Cloud Adoption Framework. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure>. 21.05.2024.
- Facebook. 2024. What is Facebook? GCFGlobal.org. <https://edu.gcfglobal.org/en/facebook101/what-is-facebook/1/#>. 15.05.2024.
- Fielding, R.T. 2000. Architectural styles and the design of network-based software architectures. University of California, Irvine. 28.04.2024.

- GeeksforGeeks. 2024. 8 Best collaboration tools for software development. GeeksforGeeks. <https://www.geeksforgeeks.org/best-collaboration-tools-for-software-development>. 31.05.2024.
- GitHub. 2016. The GitHub GraphQL API. The GitHub Blog. <https://github.blog/2016-09-14-the-github-graphql-api>. 17.05.2024.
- GitHub. 2022. GitHub REST API documentation <https://docs.github.com/en/rest?apiVersion=2022-11-28>. 26.05.2024.
- GitHub. 2024. GitHub GraphQL API documentation. <https://docs.github.com/en/graphql>. 26.05.2024.
- Google Cloud. 2024a. Google Cloud overview. <https://cloud.google.com/docs/overview>. 23.05.2024.
- Google Cloud. 2024b. Cloud APIs HTTP. <https://cloud.google.com/apis/docs/http>. 23.05.2024.
- Google Cloud. 2024c. Cloud Architecture Center. <https://cloud.google.com/architecture>. 23.05.2024.
- Google Cloud. 2023. Google Cloud Architecture. <https://cloud.google.com/architecture/framework>. 23.05.2024.
- Goodwin, M. 2024. What is an API? IBM Newsletter. <https://www.ibm.com/topics/api>. 26.04.2024.
- Granli, W., Burchell, J., Hammouda, I. & Knauss, E. 2015. The driving forces of API evolution. In Proceedings of the 14th International Workshop on Principles of Software Evolution (p. 28-37). 23.04.2024.
- GraphQL. 2024a. Schemas and Types. <https://graphql.org/learn/schema>. 19.05.2024.
- GraphQL. 2024b. Getting Started. <https://graphql.org/faq/getting-started>. 19.05.2024.
- GraphQL. 2024c. GraphQL Specification. <https://spec.graphql.org>. 19.05.2024.
- gRPC. 2022. Core concepts, architecture, and lifecycle. <https://grpc.io/docs/what-is-grpc/core-concepts>. 23.05.2024.
- gRPC. 2023. Introduction to gRPC. Online. <https://grpc.io/docs/what-is-grpc/introduction>. 23.05.2024.
- gRPC. 2024 A high-performance, open-source universal RPC framework. Retrieved from <https://grpc.io>. 23.05.2024.
- Gupta, L. 2023. HATEOAS driven REST APIs. REST API Tutorial. <https://restfulapi.net/hateoas>. 27.04.2024.
- Gupta, L. 2022. REST API Tutorial. Retrieved from <https://restfulapi.net>. 27.04.2024.
- Haywood, P. 2024. Most Popular Ecommerce Platforms (2023 Stats) - EcommerceGold. EcommerceGold. <https://www.ecommerce-gold.com/most-popular-ecommerce-platforms>. 30.05.2024.
- Hoang, V. 2021. Applying microservice architecture with modern gRPC API to scale up large and complex applications, Metropolia University of Applied Sciences, Engineering Information Technology Bachelor's Thesis <https://urn.fi/URN:NBN:fi:amk-2021060314024>. 12.05.2024.
- Husar, A. 2022, How to Use REST APIs – A Complete Beginner's Guide. Retrieved from [www.freecodecamp.org](http://www.freecodecamp.org): <https://www.freecodecamp.org/news/how-to-use-rest-api>. 03.05.2024.
- IBM. 2023. Remote Procedure Call. <https://www.ibm.com/docs/en/aix/7.3?topic=concepts-remote-procedure-call>. 24.06.2024.



- Lamos, B., Addie, S., Klaas, & Dietzel, D. 2024. Azure REST API reference documentation. Microsoft Learn. <https://learn.microsoft.com/en-us/rest/api/azure>. 22.05.2024.
- Loganathan, P. 2024. API architecture showdown - Rest vs graphql vs gRPC. Pradeep Loganathan's Blog. <https://pradeepl.com/blog/api/rest-vs-graphql-vs-grpc/#graphql---the-dynamic-orchestrator>. 31.05.2024.
- Madden, N. 2020. API security in action. Simon & Schuster Book. 22.04.2024
- Makau, L. 2023. Understanding the Distinction: REST vs. RESTful APIs. <https://www.linkedin.com/pulse/understanding-distinction-rest-vs-restful-apis-lucky-makau>. 23.05.2024.
- Microsoft. 2024. Azure documentation. Microsoft Learn. <https://learn.microsoft.com/en-us/azure/?product=popular>. 23.05.2024.
- MuleSoft, 2024. Top 3 benefits of REST APIs 2024 | MuleSoft. <https://www.mulesoft.com/resources/api/top-3-benefits-of-rest-apis>. 28.05.2024.
- Mohan, N. 2021. Think gRPC, when you are architecting modern microservices. <https://www.cncf.io/blog/2021/07/19/think-grpc-when-you-are-architecting-modern-microservices>. 20.05.2024.
- Moronfolu, M. 2024. Top advantages and disadvantages of GraphQL. Hygraph. <https://hygraph.com/blog/graphql-advantages>. 29.05.2024.
- Nally, M. 2020. Google Cloud Blog. <https://cloud.google.com/blog/products/api-management/understanding-grpc-openapi-and-rest-and-when-to-use-them%20>. 24.05.2024.
- Netflix. 2024. What is Netflix? Help Center. <https://help.netflix.com/en/node/412>. 26.05.2024.
- Netflix TechBlog. 2020a. How Netflix scales its API with GraphQL Federation. <https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-1-ae3557c187e2>. 26.05.2024.
- Netflix TechBlog. 2020b. Scaling Netflix's API via GraphQL Federation (#2). <https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-2-bbe71aaec44a>. 26.05.2024.
- Netflix TechBlog. 2024. The Netflix TechBlog. <https://netflixtechblog.com>. 26.05.2024.
- Newton-King, J. 2022. Overview for GRPC on .NET. Microsoft Learn. <https://learn.microsoft.com/en-us/aspnet/core/grpc/?view=aspnetcore-6.0>. 11.05.2024.
- Postman. 2023. State of the API Report, 2023 API Technologies. Postman API Platform. <https://www.postman.com/state-of-api/api-technologies>. 30.05.2024.
- Protocol Buffers. 2024. <https://protobuf.dev/overview>. 29.05.2024.
- Robvet. 2023. GRPC - .NET. Microsoft Learn. <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/grpc#protocol-buffers>. 13.05.2024.
- Salesforce. 2024a. What does Salesforce do? <https://www.salesforce.com/products/what-is-salesforce>. 25.05.2024.
- Salesforce. 2024b. Salesforce Developers. [https://developer.salesforce.com/docs/atlas.en-us.api\\_rest.meta/api\\_rest/intro\\_rest\\_architecture.htm](https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_rest_architecture.htm). 25.05.2024.

- Salesforce. 2024c. Introduction to REST API. Salesforce Developers. [https://developer.salesforce.com/docs/atlas.en-us.api\\_rest.meta/api\\_rest/intro\\_rest.htm](https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/intro_rest.htm). 25.05.2024.
- Salom, E. 2023. Designing REST APIs with CRUD Operations. <https://medium.com/@eliassalom/designing-apis-with-crud-operations-29d4a51fcfde>. 02.06.2024.
- Shopify. 2024a. Shopify Help Center. <https://help.shopify.com/en/manual/intro-to-shopify/overview>. 20.05.2024.
- Shopify. 2024b. REST Admin API reference. <https://shopify.dev/docs/api/admin-rest>. 20.05.2024.
- Shtatnov, A 2018. Our learnings from adopting GraphQL | Medium. <https://netflixtechblog.com/our-learnings-from-adopting-graphql-f099de39ae5f>. 19.04.2024.
- Spasev, V. Dimitrovski, I. & Kitanovski, I. 2020. An Overview of GraphQL: Core Features and Architecture. 10.04.2024.
- Synergy Research Group. 2024. Cloud Market Gets its Mojo Back; AI Helps Push Q4 Increase in Cloud Spending to New Highs. <https://www.srgresearch.com/articles/cloud-market-gets-its-mojo-back-q4-increase-in-cloud-spending-reaches-new-highs>. 01.06.2024.
- Weir, L. A. 2019. A brief look at the evolution of interface protocols leading to modern APIs. A brief look at the evolution of interface protocols leading to modern APIs <https://www.soa4u.co.uk/2019/02/a-brief-look-at-evolution-of-interface.html>. 29.05.2024.