

FACULTAD DE INGENIERÍA

Carrera de Ingeniería Empresarial y de Sistemas

DESARROLLO DE LA IMPLEMENTACIÓN DEL PROTOCOLO ESTÁNDAR GRAPHQL EN EL LENGUAJE DE PROGRAMACIÓN GO

Trabajo de Suficiencia Profesional para optar el Título Profesional de Ingeniero Empresarial y de Sistemas

ROGER CHRISTIAN RAMÓN RODRIGUEZ

TODO: código-proyecto

ASESOR:

TODO: nombre-del-asesor

código-del-asesor

Lima – Perú

2024

ÍNDICE

[**Capítulo 1: Introducción**](#_775smjeqxpsa) **3**

[**Capítulo 2: Planteamiento Del Problema**](#_3m4ttctefy4h) **4**

[2.1 Situación Problemática.](#_elf2bvone7xx) 4

[2.2 Formulación del Problema.](#_454ge9ht3kqy) 5

[2.3 Justificación de la Investigación.](#_d1me7wx6fg4h) 6

[2.4 Objetivos.](#_1g8fawmz541q) 7

[2.4.1 Objetivo General.](#_qolbtt504n1o) 7

[2.4.2 Objetivo Específicos.](#_mp2x29c44som) 7

[**Capítulo 3: Marco Teórico**](#_gzcbtqci9l55) **8**

[3.1 Antecedentes Internacionales.](#_9573u6ppmgpc) 8

[3.2 Antecedentes Nacionales.](#_tkyslh2r3s4i) 9

[3.3 Bases Teóricas.](#_quf0flrwy4ry) 9

[3.4 Definición De Términos.](#_ff08vfe0zj3t) 9

[**Capítulo 4: Desarrollo Del Proyecto**](#_735x30yh2olh) **11**

[4.1. Metodología Utilizada.](#_vwnqhtsfhbk7) 11

[4.2 Fases.](#_yki2msn9g6es) 12

[4.2.1 Análisis.](#_i45o8l36u3eh) 12

[4.2.1.1 Especificación de Requerimientos.](#_8f59lyjebyqg) 12

[4.2.2 Diseño.](#_72qvhvnhqcue) 13

[4.2.3 Desarollo.](#_342kkpfqc3q9) 13

[4.2.4 Testeo.](#_bv8jawejm890) 13

[4.2.5 Despliegue.](#_cjhs8gxen5ud) 13

[4.3 Tipos de Metodologías Ágiles.](#_y3zpcse102r1) 13

[4.4 Artefactos.](#_dhgycpqsza5) 13

[4.5 Arquitectura de la Implementación.](#_id4p5agleqh6) 13

[**Capítulo 5: Análisis Y Resultados**](#_939qh4plo1xc) **14**

[**Conclusiones**](#_ig57yxit44lf) **14**

[**Recomendaciones**](#_llrikgevodof) **14**

[**Referencias**](#_ou9oa94xna0e) **14**

## 

## Capítulo 1: Introducción

La presente tesis plantea el desarrollo de la implementación de la biblioteca open-source graphql-go, que sigue el estándar referencial, GraphQL es un protocolo para poder intercambiar datos entre el cliente y el servidor.

Es innovador porque soluciona múltiples problemas de tradicionales protocolos de comunicación entre cliente y servidor, los cuales tienen como principal forma de comunicación centralizar la responsabilidad del lado del servidor.

La implementación final es resultado de cubrir todo el estándar hasta el año 2016, la forma como se verifica que la implementación es funcional es a través de cubrir todos las pruebas de la implementación referencial graphql-js.

El resultado de la tesis produjo la biblioteca open-source graphql-go, que es usada en muchas empresas a nivel mundial, y también varios proyectos open-source que tiene mucho impacto positivo para poder mejorar la efectividad y eficiencia en la comunicación de clientes y servidores.

## Capítulo 2: Planteamiento Del Problema

### 2.1 Situación Problemática.

La forma tradicional de construir software es usando herramientas que permiten comunicar servidores y clientes a través de protocolos tradicionales cómo: SOAP, REST y RPC.

Estas herramientas tienen cómo principal funcionamiento delegar la responsabilidad al servidor, que es dónde se definen que data puede ser obtenida, modificada y eliminada, esta estrategia de comunicación genera varios problemas que son resueltos a demanda.

Los protocolos tradicionales requieren más recursos para poder enviar los datos desde el servidor al cliente.

Los límites de qué datos enviar del cliente al servidor son definidos en el servidor causando más sobrecarga al lado del cliente, la forma de definir que datos intercambiar son de múltiples formas causando la necesidad de crear un estándar.

La escalabilidad es limitada del lado del frontend porque múltiples niveles de abstracción son necesarios para coordinar la comunicación creando complejidad que hace el frontend poco mantenible por lo tanto creando necesidad de integrar muchas soluciones para aliviar los problemas.

La separación de responsabilidad es limitada porque los servidores definen que data se obtiene.

Los protocolos actuales causan múltiples formas de manejar el versionamiento de las APIs que tienen como resultado complejidad en el servidor que causan limitada escalabilidad y crear múltiples formas de versionamiento y agregan complejidad al frontend.

El intercambio de datos entre cliente y servidor usando herramientas tradicionales causan múltiples idas y vueltas de intercambio de datos que incrementan el uso de recursos y agregan complejidad del lado del frontend para consolidar el estado del software.

El desarrollo del software es lento porque requiere coordinación con múltiples endpoints con las herramientas tradicionales, por lo tanto el flujo de trabajo requiere mayores recursos porque hacer cambios impacta el servidor y el cliente.

### 2.2 Formulación del Problema.

**Problema general**

¿Por qué desarrollar la implementación del estándar GraphQL en el lenguaje de programación Go, y confirmar su compatibilidad con la implementación referencial, y cómo validar su utilidad en empresas y proyectos open-source ?

**Problema específicos**

¿Qué otras bibliotecas open-source similares existen?

¿Cómo fue la coordinación de la construcción de la biblioteca open-source graphql-go?

¿Qué herramientas se usaron para la construcción de la biblioteca open-source graphql-go?

¿Por qué se decidió que la biblioteca graphql-go sea open-source?

¿Qué fue implementado de la biblioteca open-source graphql-go?

¿Cómo la biblioteca open-source graphql-go fue implementado?

¿Cuánto tiempo tomó construir la biblioteca open-source graphql-go?

¿Qué estrategia se usó para construir la biblioteca open-source graphql-go?

¿Dónde fue presentado y cómo fue aceptado?

¿Cuándo alcanzó la estabilidad y qué implica?

¿Cómo medimos la compatibilidad con la implementación referencial?

¿Qué empresas y proyectos open-source usan la biblioteca open-source graphql-go?

¿Cuál es el impacto positivo en las empresas y proyectos open-source?

¿Cuál es el estado actual del proyecto open-source graphql-go?

¿Cuántas personas colaboraron en la construcción del proyecto open-source graphql-go?

¿Cómo es el mantenimiento del proyecto open-source graphql-go?

### 2.3 Justificación de la Investigación.

Hernández y Mendoza (2018) la justificación de los argumentos son necesarios y responden a las razones del desarrollo del estudio, la importancia del problema y a quién afecta.

Seguidamente se desarrollan las justificaciones del estudio.

**Justificación teórica**

Hernández y Mendoza (2018) argumentan que se crea valor cuando se justifica una teoría, el resultado puede ser usado por nuevos investigadores y crear nuevos conceptos.

La investigación justifica la teoría porque implementa la biblioteca inicial del estándar GraphQL en el lenguaje de programación Go, crea valor porque funda las bases para siguientes implementaciones.

Además la investigación es justificada porque la biblioteca es necesaria para resolver problemas modernos en el intercambio de datos entre clientes y servidores.

**Justificación práctica**

Hernández y Mendoza (2018) argumenta que para justificar la práctica se debe crear valor al resolver uno más problemas reales a través de innovaciones y tecnologías para la calidad de los procesos.

La investigación justifica la práctica porque se crea valor mediante la implementación de la biblioteca GraphQL en el lenguaje de programación Go, a través de pequeñas iteraciones que agregan valor, se usó herramientas de desarrollo modernas.

**Justificación social**

Hernández y Mendoza (2018) argumenta que la justificación social impacta la sociedad y para los usuarios finales de la investigación

La investigación tiene impacto social porque tiene beneficio en las empresas y proyectos porque el estándar GraphQL es open-source lo cual tiene impacto positivo en términos de usar la biblioteca de manera que no incurra en costos, y se pueda obtener el beneficio de múltiples contribuidores que mejoran la implementación.

Así cómo también permite mejor mantenimiento a largo plazo que podrá ser a prueba de futuro, apalancamiento infinitas mejoras cómo errores y correcciones de seguridad.

### 2.4 Objetivos.

### 2.4.1 Objetivo General.

Implementar el estándar GraphQL en el lenguaje de programación Go.

### 2.4.2 Objetivo Específicos.

Describir los detalles de la implementación del estándar GraphQL en el lenguaje de programación Go, especialmente relacionados a seguir la implementación de referencia: graphql-js.

Describir que la implementación cubra los tests fundamentales de la implementación referencial(graphql-js) hasta la versión v0.6.0.

Describir la utilidad de los usuarios finales y su impacto positivo en empresas y proyectos open-source.

## Capítulo 3: Marco Teórico

### 3.1 Antecedentes Internacionales.

* 99designs/gqlgen (2016/10/12): Implements a schema based graphql which generates all the components on the server side at build time, the key value is that all the parts are described on the graphql file which contains all the definitions of a type system and from there via using the command line interface commands it generates all the Go code which are: An http/s server, GraphQL endpoint and unit tests, and related development tools such as GraphQL Playground.

The drawbacks being that the Go code is auto-generated, which limits the extensibility, since the code is locked-in and making changes are difficult because it requires core dependency from the upstream project.

* graph-gophers/graphql-go (2016/10/12): Implements the GraphQL standard by having as an entry-points the schema as a file, and the types resolvers are defined via code, it is used a library, main advantage being the adoption, also the core is shared across other implementations therefore the benefits are increased by the contributions from the open-source community.

The drawbacks being that the schema is defined at text level which have their limitations in terms of extensibility because it requires extra tooling to maintain the schema source of true.

* dosco/graphjin (2019/03/24): Implements the GraphQL standard, being main features the ability to generate the API from a schema that generates the SQL queries, so it is an end-to-end solution similar than a object-relational-mapping, it does support multiple databases, allows end users to just focus on the GraphQL schema side so the Go code is generated, using the API enables to perform queries, mutations and subscriptions.

The drawbacks being that the end-to-end solution locks-in the implementation which is hard to extend, therefore the dependency on having internal hooks exposed and requires strong dependency on the maintainers.

### 3.2 Antecedentes Nacionales.

* (Karla Cecilia Reyes Burgos, 2023) Realizó un estudio titulado: Servicios web con GraphQL: una revisión sistemática de la literatura, el cual detalla la recopilación de fuentes que investigan el uso de GraphQL, y tienen cómo principal objetivo responder las siguientes preguntas:
  + ¿Qué áreas de la ciencia demuestran mayor interés en el desarrollo de servicios web usando GraphQL?
    - Concluye que la medicina es la mayor área de ciencia interesada y que aplica una implementación de GraphQL.
  + ¿Qué países han demostrado mayor interés en el desarrollo de servicios web
  + usando GraphQL?
    - Concluye que el país con mayor interés en aplicar una implementación de GraphQL es Estados Unidos.
  + ¿En qué año se encuentran la mayor cantidad de investigaciones con respecto al desarrollo de servicios web usando GraphQL?
    - Concluye que el año dónde se realizaron más soluciones informáticas es el 2019.

### 3.3 Bases Teóricas.

#### 3.4 Definición De Términos.

**Rapid Application Development (RAD):** A methodology that emphasizes and prioritizes rapid iterations of software development.

Some of the key aspects being:

* Prototyping: Leverages testing ideas and collects feedback before creating the final software version.
* User Feedback: Focuses on user feedback and continuous iteration of small working parts instead of a strict plan.
* Fast turnaround: Benefits the turnaround of the development of the software to pivot making it appealing for developers that are into highly fast paced working environments.

**Open-source Development Methodology:** A methodology that focuses on collaboration across different numbers of contributors, it is decentralized and in openness development.

Some of the key aspects being:

* Peer Review: Continue development of parts that are improved by the community so the risk of errors are reduced as much as possible.
* Decentralized Contributors: Development as a decentralized iterations via different tools that the community can leverage for synchronizing all the development workflows without single point of control.
* Public Codebase: The source of the project is at a public repository so can be accessed by any contributor to leverage all the benefits that are tied to the license, and the key feature is that security fixes can be addressed.
* Rapid Prototyping: Creates the right environment for rapid prototyping of new features, which enables the participation of all the community via the tooling of the source of the code.
* Collaboration: Encourages all types of participation which can be from different experiences so the community of contributors are able to participate and learn in different ways.

Planteamiento del problema.

Resulta que el problema principal es que lo estándares evolucionaron desde múltiples protocolos para intercambiar datos de cliente y servidor, eg. soap, rest, estas soluciones tienen como principal funcionamiento enviar datos del servidor, quiere decir que el servidor decide que datos enviar, por lo tanto los recursos que se usan son más amplios, la optimización es a nivel ad-hoc y no muy escalable, por lo tanto el mantenimiento resulta complejo y por eso nace la necesidad de crear nuevas herramientas.

La definición de que datos obtener son el servidor, por lo tanto la forma de construir los sistemas depende de la documentación de los APIs del lado del servidor.

Pero una de las formas más eficientes de construir apps es empezar desde la UI, entonces cuando se desarrolla la app, el equipo UI sabe que datos necesita, por lo tanto en lado del software de la UI se decide qué datos obtener, y la forma de obtener los datos con las herramientas existentes hace complejo el trabajo porque los datos que obtener se definen en el servidor y una mejor estrategia es definir qué datos usar basado en los UI components del lado del cliente.

Also we can mention that the need of an implementation on the go programming language was needed because the language is fully adopted for modern software therefore the need to have a version of the graphql-js.

## Capítulo 4: Desarrollo Del Proyecto

#### 4.1. Metodología Utilizada.

The methodologies that were used are:

* Rapid Application Development (RAD): We used this methodology in order to implement rapidly iteratively the GraphQL Go implementation following the graphql-js reference:
  + Prototyping: The GraphQL Go implementation was created from very basic initial interfaces that started from the text source, parser, lexer, AST, collectors and resolver.
  + User Feedback: As we made progress we received constant feedback that is documented mainly in GitHub, we did not follow a strict plan.
  + Fast turnaround: We did quick iterations in the implementation because it enabled us to constantly release multiple versions so in order to do so the contributors worked on a fast paced working environment, and also to cover the need of a working version for different companies that leverage the initial versions.
* Open-source development methodology: The development of the project was by leveraging the different characteristics of the open-source methodology to have multiple contributors adding value to the project, it is decentralized so the project can be keep forever due to the dependency on GitHub which guarantee durability and also the development was continuously iterated in the openness so any person could benefit from the project from learning, up-to proposing changes.
  + Peer Review: The project was constantly peer-review by different persons across the planet, and the key part of those besides proposing improvements is that they reported and sent security fixes to keep the project trustable and safe from vulnerabilities.
  + Decentralized Contributors: The project was iterated and leveraged GitHub as a central control for the development; it enabled multiple contributors from different parts of the planet.
* Public Codebase: The project was publicly available since the early stages of development, it allows other contributors to propose security fixes as well as reported bug plus security issues, the license is MIT which matches the reference graphql-js and enables creation of multiple branches of similar projects plus forks with improvements as well as internal forks that require special characteristics that cover inner needs of private projects.
* Rapid Prototyping: Since we leveraged GitHub as a central part of the workflow of the project, it enabled rapid iterations of including sets of changes from the graphql-js reference implementation versions.
* Collaboration: it enabled different contributors from different parts that learned from the project and they were to add value.

#### 4.2 Fases.

#### 4.2.1 Análisis.

Why was decided to do the implementation

4.2.1.1 Introducción

The implementation was decided to be built because at the time when the GraphQL Standard was released in 2015 there was no implementation in the programming language Go.

The reference implementation that was released together was graphq-js.

So there was the need to create the library in the Go programming language which matches the reference implementation in order to have compatibility at certain version.

4.2.1.2 Identificar Tareas

Which tasks were created in order to detail the implementation

The strategy we did in order to find the tasks to create the Go implementation was to port changes matching the version from graphql-js.

Those tasks were delivered using mainly GitHub via pull-requests, besides the porting changes there were small incremental pull-requests that improved certain parts of the implementation.

The following tasks were identified:

Porting Changes Matching Versions:

* Porting changes from graphql-js version 0.4.18.
  + <https://github.com/graphql-go/graphql/pull/117>
* Porting changes from graphql-js version 0.5.0.
  + <https://github.com/graphql-go/graphql/pull/123>
* Porting changes from graphql-js version 0.6.0.
  + <https://github.com/graphql-go/graphql/pull/192>

Main Components:

* Executor
  + <https://github.com/graphql-go/graphql/pull/8>
* Source:
  + <https://github.com/graphql-go/graphql/pull/5>
* Visitor
  + <https://github.com/graphql-go/graphql/pull/10>
* Printer
  + <https://github.com/graphql-go/graphql/pull/10>
* Parser:
  + <https://github.com/graphql-go/graphql/pull/2>
* Lexer:
* AST:
  + <https://github.com/graphql-go/graphql/pull/3>
* Collector:
* Resolver:
  + <https://github.com/graphql-go/graphql/pull/288>

Other Components:

* Types
  + <https://github.com/graphql-go/graphql/pull/12>
* Errors
  + <https://github.com/graphql-go/graphql/pull/423>

Build System:

* CircleCI:
  + <https://github.com/graphql-go/graphql/pull/361>

4.2.1.3 Priorizar Tareas

4.2.1.4 Mapear Tareas

4.2.1.5 Medir Alcance

#### 4.2.1.1 Especificación de Requerimientos.

Related to how the system behaves at an internal level.

How all different apis of gql behave

Requerimientos funcionales:

* Source: Source-in.
* Parser: Source to AST and Lexer.
* Lexer: Tokenizes the source-in.
* AST: The source as a tree.
* Collector: Matches AST to resolvers.
* Resolver: Produces end-result, meaning the source-out.

Related to the usability, ux of the end-user, at external level.

Requerimientos no funcionales:

* Performance:
* Usability:
* Scalability:
* Security:
* Maintainability:
* Compatibility:
* Portability:

Related to the domain category of the type of software, specific to the domain/industry.

Requerimientos de dominio:

* GraphQL Standard Spec:
* Open-source Standards:
* Go Standards:

#### 4.2.2 Diseño.

#### 4.2.3 Desarollo.

#### 4.2.4 Testeo.

#### 4.2.5 Despliegue.

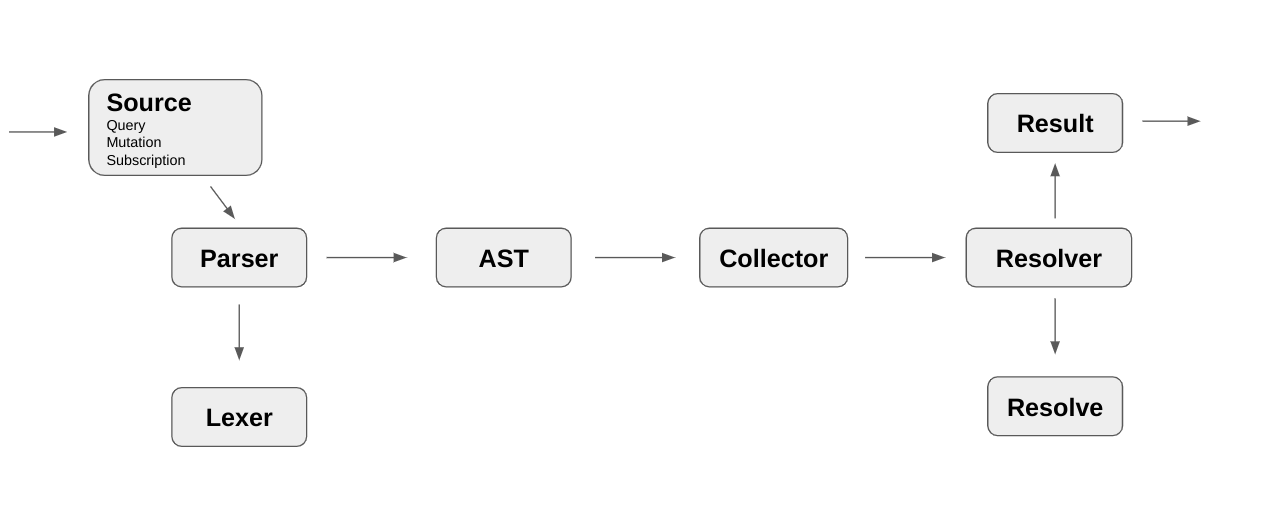
What are the process of the open-source project ?

4.2.1 Diagrama de flujo del proceso AS IS

#### 4.3 Tipos de Metodologías Ágiles.

#### 4.4 Artefactos.

#### 4.5 Arquitectura de la Implementación.



query/mutation/subscription as source.

Source: Source-in.

Parser: Source to AST and Lexer.

Lexer: Tokenizes the source-in.

AST: The source as a tree.

Collector: Matches AST to resolvers.

Resolver: Produces end-result, meaning the source-out.

State machine is a representation of a workflow of nodes that produce as an end-result a given state, the nodes that produce the final state can be partial, meaning the paths of the end-result are defined per set.

GraphQL have multiple components that can be represented as a state machine, for example the most important component that are abstracted as:

* AST state machine, up-down direction.
* Source partial state machine, because one way flow.

## Capítulo 5: Análisis Y Resultados

## Conclusiones

## Recomendaciones

## Referencias

Marco Teórico

Antecedentes internacionales

All other go implementations

Antecedentes nacionales

Check go local implementations

Bases teoricas

Sistemas de informacion q es

Estandard basado en graphql-js, check the implemtation strategy

Metodologia Scrum: Yes because new way build software so we can conclude i followed this.

Definicion de terminos

All the new terms that the new standard generates based on the solution

Desarrollo del Proyecto

Scrum strategy in details

With end result of the project

Análisis y Resultados

We can include here that the tools is well created and useful we can show the test passing the percentages based on the graphlq js implementation

And also listed of the dependant projects, and perhaps we add some emails asking how does it work for them, we can do some analysis on how their apis work well.

Add more number of the tests and adoption.

Conclusiones

Here we can conclude that the the the end result of the problem was fully covered and that the library is completed well stablished full adopted has their niche.

**Referencias Bibliográficas**

Facebook, Inc. (2019). GraphQL referencia estándar.

<https://spec.graphql.org/October2021/>

GraphQL Spec.

<https://github.com/graphql/graphql-spec>

Official Website.

<https://graphql-go.github.io/graphql-go.org/>

GraphQL Queries as state machine.

<https://rmosolgo.github.io/ruby/graphql/2016/11/12/graphql-query-as-a-state-machine.html>

GraphQL Spec License.

<https://jointdevelopment.org/>