UNIVERSIDAD TÉCNICA DEL NORTE

Facultad De Ingeniería En Ciencias Aplicadas.

Carrera de Software.

TEMA:

REALIZACIÓN DE UN SURVEY A REPOSITORIOS GITHUB PARA IDENTIFICAR LA TENDENCIA DE GRAPHQL Y SUS COMPONENTES EN PROYECTOS DE SOFTWARE

Trabajo de grado previo la obtención del título de Ingeniero en Software

Autor:

Stalin Andre Maigua Gordillo

Director:

PhD. Cathy Pamela Guevara Vega MSc.

Ibarra – Ecuador 2025

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TECNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004612824		
APELLIDOS Y NOMBRES:	MAIGUA GORDILLO STALIN ANDRE		
DIRECCIÓN:	ATUNTAQUI, Calle Junín y América		
EMAIL:	samaiguag@utn.edu.ec		
TELÉFONO FIJO	0992292919	**TELÉFONO MÓVIL:**	0992292919

DATOS DE LA OBRA	
TÍTULO:	REALIZACIÓN DE UN SURVEY A REPOSITORIOS GITHUB PARA IDENTIFICAR LA TENDENCIA DE GRAPHQL Y SUS COMPONENTES EN PROYECTOS DE SOFTWARE
AUTOR:	MAIGUA GORDILLO STALIN ANDRE
FECHA:	03/09/2025
PROGRAMA:	■ PREGRADO ☐ POSGRADO
TITULO POR EL QUE OPTA:	INGENIERO EN SOFTWARE
DIRECTOR:	PHD. CATHY GUEVARA, MSC.
ASESOR 1:	PHD. Antonio Quiña, MSC.
ASESOR 2:	PHD. Irvin Reascos, MSC.

CONSTANCIA

2.CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de esta y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 3 días del mes de septiembre de 2025

EL AUTOR:

ESTUDIANTE
Maigua Gordillo Stalin Andre
C.I: 1004612824

CERTIFICADO DEL DIRECTOR DE TRABAJO DE GRADO

CERTIFICACIÓN DIRECTOR

Ibarra 03 de septiembre del 2025

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE TITULACIÓN

Por medio del presente yo PhD. Cathy Guevara, MSC, certifico que el Sr. **Stalin Andre Maigua Gordillo** portador de la cédula de ciudadanía número **1004612824**, ha trabajado en el desarrollo del proyecto de grado **"REALIZACIÓN DE UN SURVEY A REPOSITORIOS GITHUB PARA IDENTIFICAR LA TENDENCIA DE GRAPHQL Y SUS COMPONENTES EN PROYECTOS DE SOFTWARE",** previo a la obtención del Título de **Ingeniero de Software** realizado con interés profesional y responsabilidad que certifico con honor de verdad.

Es todo en cuanto puedo certificar en honor a la verdad.

Atentamente

PhD. Cathy Guevara
DIRECTORA DE TESIS

Dedicatoria

Dedico este trabajo.

A ti, papá y a ti mami por tu paciencia y tus palabras siempre oportunas, que supieron levantarme en los momentos difíciles. Y a ti, mamá, por tu amor incondicional, tu fe en mí y por demostrarme que cada día de sacrificio y la entrega no conocen límites cuando se trata de un hijo.

Con infinita gratitud Stalin Andre Maigua Gordillo orgulloso siempre de ser su hijo.

Agradecimiento

Al llegar al cierre de esta etapa académica tan significativa en mi vida, siento la necesidad de expresar mi más sincero agradecimiento a todas las personas que, de una u otra manera, formaron parte de este recorrido y dejaron huellas imborrables en mi formación personal y profesional.

A los docentes de la carrera de Ingeniería de Software, mi profundo reconocimiento. Su entrega, dedicación y sabiduría fueron guía constante durante estos años. Cada clase impartida y cada consejo brindado contribuyeron de manera invaluable a mi crecimiento como futuro profesional.

Quiero extender un agradecimiento especial a mi tutora de titulación, PhD. Cathy Guevara, MSc., por su acompañamiento cercano y comprometido durante la elaboración de este trabajo. Su orientación, paciencia y exigencia académica fueron claves para alcanzar este objetivo. También agradezco al PhD. Antonio Quiña y al PhD. Irvin Reascos, MSc., por compartir sus conocimientos, su experiencia y por haber sido pilares fundamentales a lo largo de mi formación universitaria.

A mis amigos, quienes me brindaron su compañía, apoyo y alegrías durante todo este camino, gracias por estar presentes en los momentos buenos y en los más difíciles. De manera especial a D.L.C.Q, por siempre estar apoyándome a cada momento y estar al pendiente de mi pese a las circunstancias y tropiezos que pude a ver cometido a lo largo de la carrera.

A cada una de las personas que formaron parte de este proceso, gracias. Hoy cierro este capítulo con el corazón lleno de gratitud y entusiasmo por lo que viene. Todo lo vivido junto a ustedes ha marcado positivamente mi historia, y me acompañará siempre en lo que está por venir.

Tabla de Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El objetivo de la presente investigación fue realizar un estudio empírico para definir la tendencia de uso del lenguaje de consultas GraphQL, abordando el desconocimiento existente sobre su adopción real en proyectos de software y contribuyendo con conocimiento actualizado al campo de la ingeniería de software.

El objetivo principal de este trabajo de titulación fue crear una aplicación capaz de consumir la API de GitHub para realizar un Survey. Esta integración facilitó la recolección y análisis sistemático de datos permitiendo identificar los proyectos que utilizan GraphQL y los componentes que implementan.

La investigación examinó las características de GraphQL y los componentes más utilizados. Para ello, se desarrolló una herramienta propia que, mediante consultas a la API de GitHub, extrajo características como el lenguaje principal, las métricas de popularidad (estrellas, bifurcaciones) y visualizaciones que tienen.

El resultado fue un análisis detallado que demuestra que GraphQL ha alcanzado un estado de madurez y consolidación, con una fuerte presencia en distintos lenguajes de programación. Al definir claramente la tendencia actual y el uso avanzado de sus componentes, esta tesis aporta información de gran valor para la comunidad de desarrolladores, promoviendo la toma de decisiones informadas en la arquitectura de APIs modernas.

Palabras Clave: Survey, GitHub-API, GraphQL.

ABSTRACT

The objective of this research was to conduct an empirical study to define usage trends of the GraphQL query language, addressing the lack of knowledge about its current adoption in software projects and providing up-to-date insights into the field of software engineering.

The main objective of this thesis was to create an application capable of consuming the GitHub API to conduct a survey. This integration facilitated systematic data collection and analysis, allowing us to identify projects that use GraphQL and the components they implement.

The research examined GraphQL characteristics and the most commonly used components. To this end, a proprietary tool was developed that, through queries to the GitHub API, extracted features such as the primary language, popularity metrics (stars, forks), and visualizations.

The result was a detailed analysis demonstrating that GraphQL has reached a state of maturity and consolidation, with a strong presence in various programming languages. By clearly defining current trends and advanced usage of its components, this thesis provides valuable insights for the developer community, promoting informed decision-making in modern API architecture.

Keywords: Survey, GitHub API, GraphQL.

INTRODUCCIÓN

Antecedentes:

Situación Actual:

Planteamiento del Problema:

Actualmente se desconoce la tendencia del uso de GraphQL y sus componentes de desarrollo de software lo que impide a nuevos investigadores generar nuevas aplicaciones orientadas a servicios, al encontrar información como artefactos, frameworks, métodos, modelos con respecto a GraphQL y sus componentes permitirá disminuir el vacío que existe en el cuerpo del conocimiento acerca del desarrollo de aplicaciones informáticas que consuman APIs.

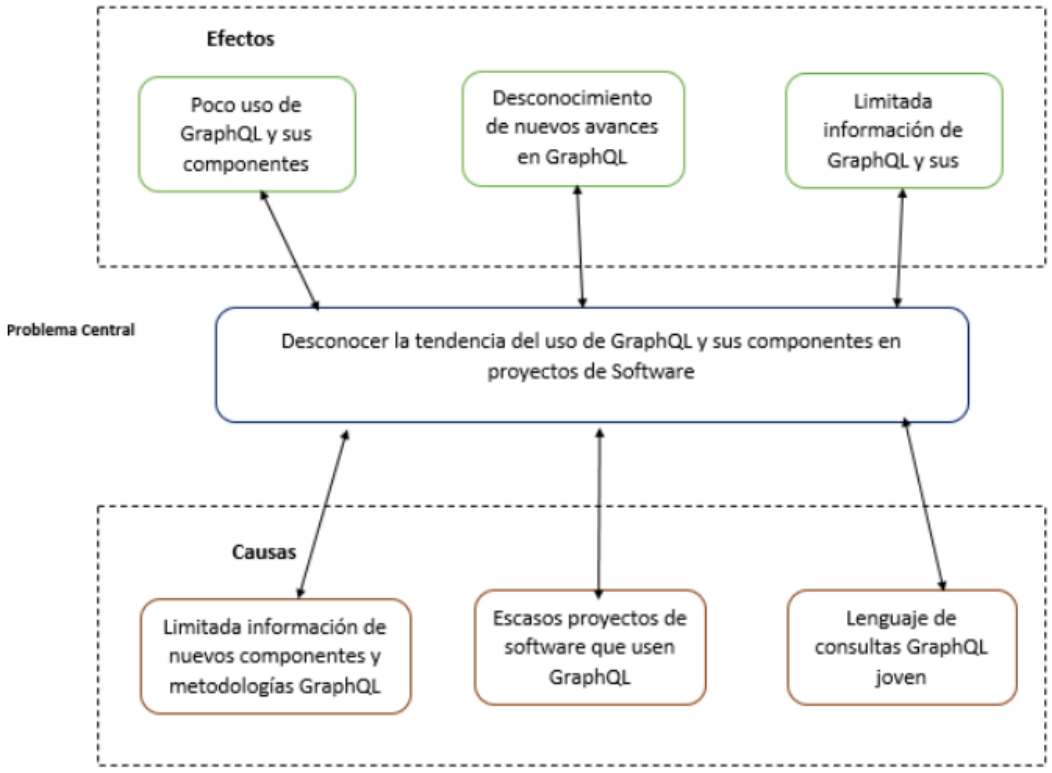


Figura 1. Árbol de Problemas
Fuente: Propia

Objetivos

Objetivo General

Realizar un Survey a repositorios GitHub para identificar la tendencia de GraphQL y sus componentes en proyectos de software.

Objetivos Específicos

- Conocer el estado actual del lenguaje de consultas GraphQL y repositorios GitHub.
- Desarrollar una aplicación orientada a servicios API GraphQL para consumir la API de GitHub.
- Identificar los proyectos de software que usan GraphQL mediante un Survey a los repositorios GitHub consumiendo la aplicación desarrollada.
- Definir la tendencia del uso de GraphQL y sus componentes con los resultados del Survey.

Alcance

El presente trabajo pretende realizar una aplicación orientada a servicios, que consuma el API de GitHub para identificar la tendencia del uso de GraphQL y sus componentes en proyectos de software. Se realizará un Survey para identificar los proyectos que usan GraphQL. En la primera etapa se realizará un marco teórico, para conocer el estado actual del lenguaje de consultas GraphQL y de los repositorios GitHub con respecto al uso de GraphQL en proyectos de desarrollo de software. En la segunda etapa, se desarrollará una aplicación orientada a servicios usando GraphQL para consumir el API de GitHub que posteriormente nos ayudará en la siguiente etapa. Finalmente, en la tercera etapa, se identificará los proyectos de software que usen GraphQL mediante un Survey a los repositorios GitHub usando la aplicación desarrollada. Las tecnologías que se usarán para el desarrollo de la aplicación orientada a servicios son:

- IDE de programación Visual Studio Code
- Lenguaje de consultas GraphQL
- NodeJs
- Motor de base de datos PostgreSQL

Finalmente, con los resultados obtenidos del Survey se definirá la tendencia que tiene GraphQL y sus componentes en proyectos de software en el repositorio GitHub.

Metodología

El siguiente trabajo tiene un enfoque cuantitativo basado en encuestas hacia GitHub y el uso de una arquitectura basada en servicios para el consumo de API. Para poder verificar los resultados de las encuestas el uso de la herramienta GraphQL y API. El siguiente trabajo constara de:

- Diseño
- Codificación
- Obtención de resultados
- Interpretación de resultaos

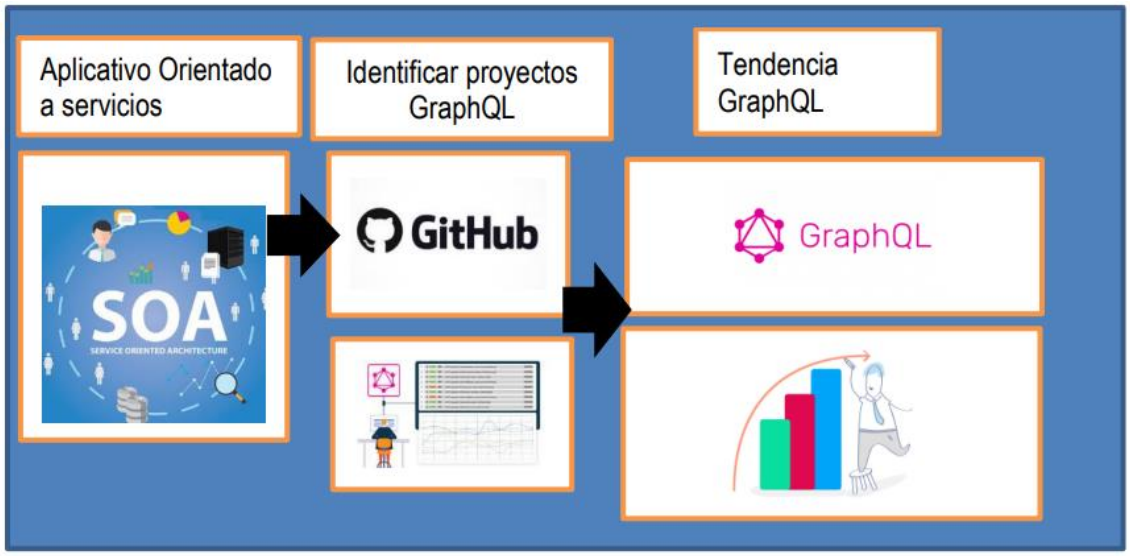


Figura 2. Cuadro metodológico
Fuente: Propia

Justificación

El enfoque de este trabajo va dirigido hacia los objetivos de desarrollo sostenible (ODS) Educación de calidad (Objetivo 4): Mediante la investigación realizada en este trabajo, se pretende contribuir al acceso a formación técnica y a obtener conocimiento acerca del contenido de este trabajo (Arenales, 2020). Trabajo decente y crecimiento económico (Objetivo 8): Estimular el crecimiento económico mediante la generación de trabajo por medio de la innovación tecnológica (ODS Territorio Ecuador, 2018). La contribución de herramientas tecnológicas accesibles a la sociedad se ha vuelto hoy en día una práctica que permite el bien a la sociedad, que cada día va avanzando y crece aún más la necesidad de saber y recurren con mayor afán a las herramientas tecnológicas que estén a su alcance como es el software libre (Sataloff et al., n.d.). GitHub es un repositorio para proyectos de software que permite a la comunidad gestionar proyectos de desarrollo con funcionalidades directas con diferentes IDS de programación. Con los resultados que se obtengan del crecimiento de GraphQL y con el uso del Api, el presente trabajo ayudara a las nuevas plazas de trabajo y aumentar la producción conocimiento con respecto a la joven herramienta GraphQL.

Riesgos

Métricas incorrectas en los modelos de Survey: Si las métricas no están correctamente escogidas los resultados no serán los que esperamos obtener.

Uso inadecuado de los datos: Para poder minimizar este riesgo es necesario basarse de técnicas de uso de datos.

Inexperiencia en el uso de herramientas: El desconocimiento de las herramientas a aplicarse, puede traer grandes conflictos en el momento del diseño.

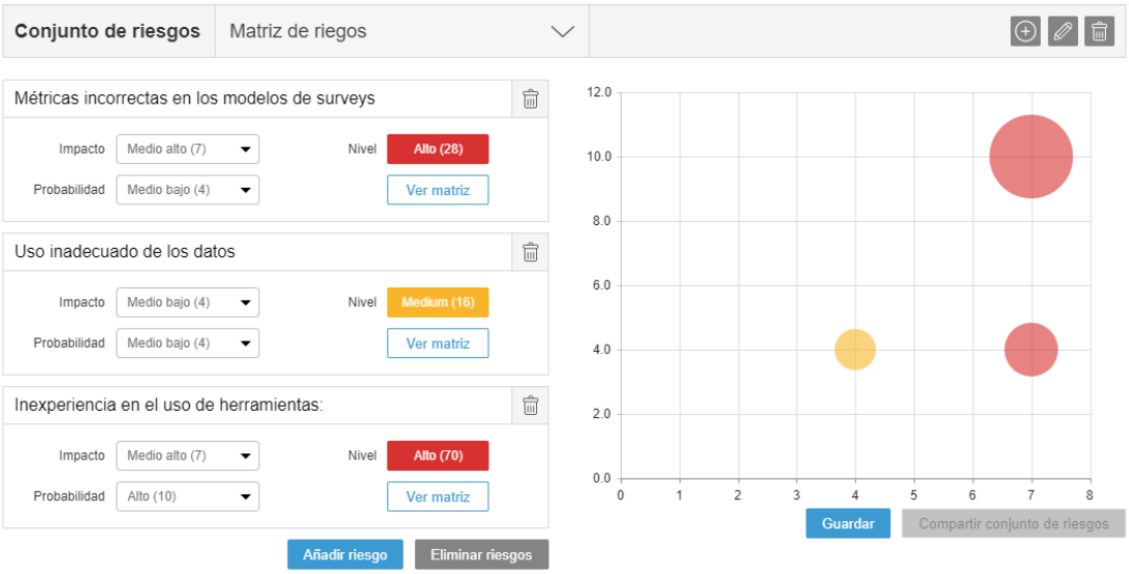


Figura 3. Matriz de Riesgos
Fuente: HTTPS://ITMPLATFORM.COM/ES/RECURSOS/ MATRIZ-DE-EVALUACION-DE-RIESGOS/

CAPÍTULO I

1. Marco teórico

1.1. Fundamentos de la investigación empírica en software

1.1.1. Definición y propósito

La base de la investigación empírica es la observación de fenómenos y la captura de experiencias, por lo que es fundamental planificar los pasos para realizar sus estudios; esto permitirá al investigador resolver problemas u obstáculos que puedan surgir durante el proceso experimental. Además, el investigador debe ser consciente de las amenazas potenciales a su estudio y justificar sus hallazgos(Guevara-Vega et al., 2021).

En este sentido, nos planteamos explicar tres elementos que el investigador debe conocer(Guevara-Vega et al., 2021):

1)La unidad de análisis, es el objeto crítico analizado en la investigación empírica. Puede ser un artefacto, un proyecto, un rol específico de una persona, un grupo de personas o una organización. La decisión de la unidad de análisis es crítica ya que dicta la pregunta de investigación que tiende a integrarse en los métodos de investigación.

2)Muestreo, es el proceso de seleccionar qué porción de la población analizar. Hay varias estrategias de muestreo que un investigador puede adoptar, como el muestreo aleatorio y la bola de nieve. Los investigadores deben comprender bien el proceso de muestreo antes de comenzar a recopilar sus datos.

3)Fiabilidad y Validez, se consideran en cada paso de los puntos de decisión del diseño de la investigación; esto puede incluir el diseño de un cuestionario o entrevistas, el diseño experimental, la estrategia de muestreo o cómo se analizan los datos. Realizar una investigación empírica sin considerar su confiabilidad y validez no tiene sentido porque el investigador no generalizará a partir de los resultados.

En la actualidad, toda empresa competitiva u organización, cualquiera que sea su naturaleza, basa su gestión y funcionamiento sobre un concepto que se considera fundamental. La estrategia es el objetivo de la actividad que realiza la dirección de la empresa, que debe perseguir que su organización funcione de manera eficiente, y la mejor manera de que esto ocurra es que no existan conflictos en la misma. Es por esto que la cúpula de la empresa deberá planificar

su estrategia en función de los objetivos que persiga, para lo que debe definir claramente lo que quiere conseguir, la forma de conseguir los objetivos fijados y un posterior sistema de control. Es lo que se denomina Formulación e Implantación de la estrategia, los cuales no se quedan sólo en el estudio previo, sino que en la práctica se desarrollan al mismo tiempo. Con la formulación de la estrategia, la dirección de la empresa define los objetivos que pretende alcanzar; Para ello parte de un análisis de su propia empresa, así como del entorno que la rodea(Espinosa & Cabrera, 2017).

Se pueden distinguir dos enfoques diferentes al realizar una investigación empírica según (Neill & Cortez Suárez, n.d.).

- El enfoque **cualitativo** se basa en estudiar la naturaleza del objeto y en interpretar un fenómeno a partir de la concepción que las personas tienen del mismo. Los datos que se obtienen de estas investigaciones están principalmente compuestos por texto, gráficas e imágenes, entre otros.
- El enfoque **cuantitativo** se corresponde con encontrar una relación numérica entre dos o más grupos. Se basa en cuantificar una relación o comparar variables de estudio.

Los datos que se obtienen en este tipo de estudios son siempre valores numéricos, lo que permite realizar comparaciones y análisis estadístico. Es posible utilizar los enfoques cualitativos y cuantitativos para investigar el mismo tema, pero cada enfoque responde a diferentes interrogantes (Apa et al., 2010a).

1.1.2. Comparación con otras estrategias empíricas

Se puedo definir 3 tipos principales de técnicas o estrategias para la investigación empírica.

- **Survey**

Los Surveys son utilizados con mayor amplitud como un proceso de investigación que permite la recopilación de datos de manera rápida y eficaz.

En la Ingeniería de Software Empírica las encuestas se utilizan de forma similar, se obtiene un conjunto de datos de un evento que ha ocurrido para determinar cómo reacciona la población frente a una técnica, herramienta o método particular, o para determinar relaciones o tendencias(Apa et al., 2010b).

Según Luis & Gonzáles (2017) en un estudio es fundamental seleccionar correctamente las variables a estudiar, pues de ellas dependen los resultados que se pueden obtener. Si los resultados no permiten concluir sobre los objetivos del estudio se han elegido mal las variables.

Una de la característica más relevante del Survey es que te proporciona distintas variables para realizar el estudio. Esto hace posible construir una variedad de modelos y luego seleccionar el que mejor se ajusta a los propósitos de la investigación.

El instrumento básico utilizado en la investigación por encuesta es el cuestionario.

Según Casas Anguita (2003) el objetivo que se persigue con el cuestionario es traducir variables empíricas, sobre las que se desea información, en preguntas concretas capaces de suscitar respuestas fiables, válidas y susceptibles de ser cuantificadas. Como ya se ha mencionado, el guion orientativo del que se debe partir para diseñar el cuestionario lo constituyen las variables previamente establecidas.

- **Casos de Estudio**

Concordando con lo que dice Stake (2017) los casos de estudio no manipulan las variables, sino que éstas son determinadas por la situación que se está investigando. Al igual que las encuestas, los casos de estudio pueden ser clasificados como cualitativos o cuantitativos dependiendo de lo que se quiera investigar del proyecto en curso.

- **Experimentos**

La investigación experimental está integrada por un conjunto de actividades metódicas y técnicas que se realizan para recabar la información y datos necesarios sobre el tema a tratar; La investigación experimental se presenta mediante la manipulación de una variable experimental no comprobada, en condiciones rigurosamente controladas, con el fin de describir de qué modo o por qué causa se produce una situación o acontecimiento particular (Investigación Experimental, 2019).

1.2. Arquitectura de software orientada a servicios y microservicios

1.2.1. Arquitectura orientada a Servicios

La arquitectura orientada a servicios (SOA) es un modelo organizativo de componentes de software en el que se encuentran sus recursos distribuidos e

independientes, se puede controlar de muchas maneras diferentes. Este intercambio de recursos, también conocido como servicios, es uno de estos objetivos SOA. El uso de esta estandarización garantiza que los recursos de su aplicación pueden ser utilizado por cualquier usuario que necesite este servicio(Rosado Gomez & Jaimes Fernández, 2018).

La gestión de este modelo arquitectónico involucra a proveedores y consumidores. servicio, la simplicidad que aporta esta arquitectura es tanto para aplicaciones el prestador, como cliente del servicio, podrá hablar diferentes idiomas o plataformas sin perder su eficacia y uso previsto, se logra la compatibilidad e integración entre diferentes sistemas(Milena Caicedo & ROJAS DIAZ Ing Electrónico Maestrante Ciencias Computacionales Profesor Auxiliar, 2008).

1.2.2. Arquitectura Orientada a Microservicios

El término microservicios no es relativamente nuevo, este estilo arquitectural fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo que los participantes estaban explorando. No existe una definición en concreto para microservicio, sin embargo, una aproximación que la realiza lo define como: “Pequeños servicios autónomos que trabajan juntos”. Una arquitectura de microservicios promueve el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares y auto-contenidas, lo cual difiere de la forma tradicional o monolítico(López & Maya, n.d.).

Características según (Farnham et al., 2020):

- Usa tecnología heterogénea.
- Servicios independientes.
- Gran escalabilidad.
- Despliegue independiente.

En la siguiente figura se muestra un ejemplo de arquitectura de microservicios en donde se puede notar 4 capas:

- Primera capa, en esta capa se encuentran los clientes donde se realiza el consumo de los diferentes servicios y posteriormente mostrar en una interfaz para usuarios y terminales.
- Segunda capa, en esta capa se encuentra el API en la cual se recibe las peticiones del servicio-cliente a través de un protocolo HTTTP.

- Tercera capa, en esta capa se encuentra la “orquesta de contenedores” aquí se colocará toda la lógica del emprendimiento, podemos decir que en esta capa tendremos varios microservicios en distintos lenguajes y frameworks.
- Cuarta capa, en esta capa tenemos todos los datos guardados independientemente de una base de datos para los microservicios.

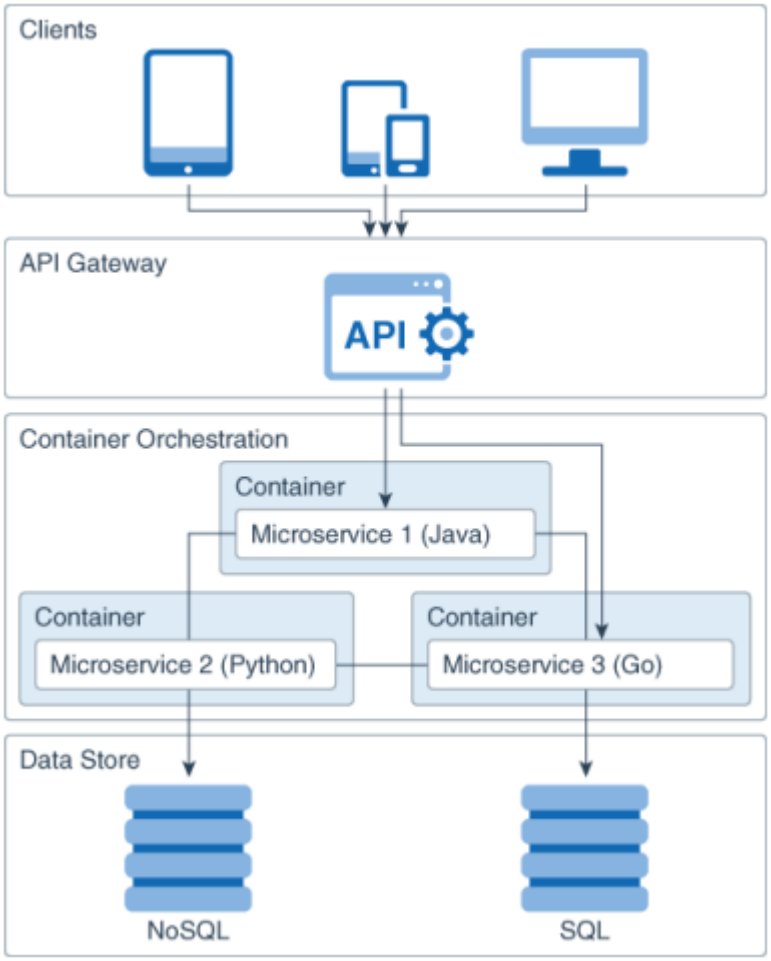


Figura 4. Arquitectura de microservicios
Fuente: https://docs.oracle.com/es/solutions/learn-architect-microservice/index.html

1.3. Lenguaje de consulta GraphQL

1.3.1. Introducción

GraphQL es un lenguaje de consultas desarrollado para las APIs que ejecuta consultas del lado del servidor, basándose en la definición de un sistema de tipos para los datos predefinido por el desarrollador. GraphQL hace flexible el consumo de la API lo cual da el control de la obtención de datos al cliente; por medio de consultas personalizadas, el cliente obtiene solo lo que necesita y por medio de

una única petición. GraphQL elimina el versiona miento utilizado por REST, ya que solo se necesita cambiar la estructura de la solicitud para obtener una nueva estructura de datos (Linux-Fundation, 2019).

Esta tecnología fue desarrollada por Facebook e implementado en el lado del servidor. aunque es una consulta lenguaje, el GraphQL no está conectado directamente con el base de datos. En otras palabras, GraphQL no se limita tanto a SQL y bases de datos NOSQL(*2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT),* 2018).

1.3.2. Operaciones

GraphQL tiene la capacidad de realizar las cuatro funciones fundamentales para gestionar información almacenada conocido como CRUD (crear, consultar, actualizar y eliminar) y especifica tres tipos de operaciones las cuales son independientes de la solicitud HTTP; estas operaciones son: consultas, mutaciones y suscripciones

Queries

Un query (o petición en español) es el mensaje que se le envía al servidor para lograr extraer información de un servicio GraphQL(Wittern et al., 2018).

Se observa en la Fig 4, hace una petición específica que es timeline, lo cual está pidiendo posts en relación a un usuario que es pasado por parámetros como **of** además utilización del atributo de Post, replies, lo que indica al servicio GraphQL que también se están requiriendo todas las respuestas de todos los Posts que se encuentren del usuario. Cuando se hace un query, GraphQL requiere que todos los atributos requeridos sean tipos primitivos para aumentar la predictibilidad(Quiña-Mera et al., 2023).

```
query {
  timeline(of: "MaxMustermann") {
    ...basicPostFields
    replies {
      ...basicPostFields
    }
  }
}

fragment basicPostFields on Post {
  content
  author {
    nickName
  }
}
```

Figura 5. Ejemplo de un Query
Fuente: (Alfredo Guillen-Drija & Quintero, 2019.)

Mutations

Las mutations proporcionan una manera de modificar los datos ya sea creando datos nuevos o actualizando los ya existentes, su sintaxis es parecida a la de la query, también pueden recibir argumentos de entrada, aunque esto es usual que estos argumentos sirvan para crear el dato o actualizar alguno de sus campos esto es útil para comprobar que el objeto se ha actualizado o creado correctamente (TFG_ALVARO_SAINZ_DEL_NOGAL, 2019).

En la fig. 5, se puede observar un ejemplo sencillo de una mutation en donde se crea un estudiante y mediante un argumento de entrada se indica el nombre que debe de tener. En la query se muestra que luego de la creación del dato y la actualización del objeto se quiere ver un campo name y como se indica la respuesta es el dato deseado.

Figura 6. Ejemplo de una Mutación
Fuente: (TFG_ALVARO_SAINZ_DEL_NOGAL, 2019.)

Subscriptions

Las Subcripciones es aquella informacion o datos que se quiere enviar a un cliente tomando en cuenta que tiene un servidor que tiene una lista de subscripciones que admite evidenciando los eventos a los que se desea suscribir.

En la fig.6 es un ejemplo de mutación básico basado en un storyLike, que se pude usar para que los clientes puedan dar me gusta a una publicación.

```
mutación StoryLikeMutation ( $ entrada : StoryLikeInput ) {
  storyLike ( entrada : $entrada ) {
    historia {
      Me gusta { contar }
      likeSentence { texto }
    }
  }
}
```

Figura 7. Ejemplo de Mutacion 2
Fuente: (Schafer & Kuenzel, 2015)

En la fig.7 se pude observar que tiene un campo llamado *storyLikeSubscribe* que le permite al cliente obtener los datos que se envían cuando le gusta o no le gusta una historia, esto se vería como la actualización que desearía el usuario y donde se emplearía una suscripción.

```
suscripción StoryLikeSubscription ( $ entrada : StoryLikeSubscribeInput ) {
  storyLikeSubscribe ( entrada : $entrada ) {
    historia {
      Me gusta { contar }
      likeSentence { texto }
    }
  }
}
```

Figura 8. Ejemplo de Mutación 3
Fuente: (Schafer & Kuenzel, 2015)

1.3.3. Schema

Una de las principales características de GraphQL es que utiliza un esquema base que representará toda la data que se encuentra en la API. Es incorrecto, entonces, referirse a estos tipos de data como recursos, pues no

responden a un URI específico. En el esquema, se definen diferentes Types, definiendo la forma en la que estos se relacionan y cómo están compuestos(Alfredo Guillen-Drija & Quintero, 2018.).

Como puede observarse en la fig.8 es un *Schema* simple y sencillo, se observa que *Student* posee los campos name y tutor, asignándose de tipo *String* y de tipo Tutor respectivamente.

```
type Student {
  name: String!
  tutor: Tutor!
}
```

Figura 9. Ejemplo de Schema
Fuente:(TFG_ALVARO_SAINZ_DEL_NOGAL, 2019.)

1.3.4. Validación

Para saber si una consulta es válida, existen un conjunto de reglas de validación que garantiza que el contenido de una consulta sea semánticamente correcto. Por medio del sistema de tipos se puede realizar la validación de la consulta, evitando acceder a comprobaciones en tiempo real. Algunos tipos de errores semánticos que se especifican en (Linux-Fundation, 2019).

- Fragmento repetitivo Si en el contenido de una consulta por medio de un fragmento se llama al mismo fragmento, se estaría creando un ciclo repetitivo infinito dando como resultado una sobrecarga en procesamiento e ineficiencia en general

- Campos inexistentes Los campos que se establecen en una consultan, deben existir en el tipo de la definición del esquema de la aplicación, si no existen el resultado mostrará un error de campo inválido.

- Sin campos para devolver Si la estructura de la consulta no tiene como último nivel un tipo escalar, el resultado mostrará un error; este error se produce cuando de un tipo objeto no se selecciona ningún campo de nivel inferior; es decir, no existe un anidamiento completo en la consulta.

- Campo escalar Los campos escalares son el último nivel en la estructura de un tipo; es decir no existen niveles más debajo de estos, por esta razón la respuesta ante una petición de campos tipos escalares generará un error.

1.3.5. Ejecución

En la ejecución de operaciones GraphQL, la validación de campos es fundamental para consultas o mutaciones, sin embargo, para mostrar resultados, un servidor debe ejecutar estas operaciones por medio de resolutores. Cada campo establecido es denominado como una función que devuelve un valor para el siguiente tipo, si el valor producido por estos campos es escalar, se finaliza la ejecución de dicho campo, caso contrario, continua al siguiente nivel (LinuxFundation, 2019).

- **Resolutores**

Los resolutores en GraphQL son funciones las cuales son usadas para obtener los datos de los campos definidos en el esquema. Cada campo del esquema definido tiene una función de correspondencia (Ghebremicael, 2017). Las instrucciones de los resolutores, permiten la conversión de las operaciones en datos; cada campo tiene una función de resolución y este conjunto de funciones son llamadas mapa de resolución. Los resolutores tienen por defecto cuatro argumentos, los cuales son: parent, args, context, info (ApolloGraphQL, 2019). En la Fig. 13 se muestra el resolutor del tipo course con los argumentos mencionados como parámetros, en la imagen se usa el argumento args para obtener el id enviado desde la estructura de la consulta en el esquema.

a) Parent (obj): Es el objeto anterior; este parámetro es inservible para el campo raíz; así se evidencia el anidamiento de las consultas en GraphQL.

b) Args: Contiene los parámetros del campo que fueron pasados en la consulta.

c) Context: Este argumento contiene el estado de cada solicitud, donde se incluye la autenticación, y toda la información de resolución de la consulta; este es un objeto compartido para los resolutores.

d) Info: La información correspondiente al estado de la consulta es contenida en este argumento, también contiene información del nombre y la ruta jerárquica del campo.

Existen resolutores asíncronos, estos resolutores se usan cuando existe una carga a una fuente de datos ya sea una base de datos, una API, etc. Estas operaciones al ser asíncronas necesitan las Promises, Futures, Tasks o Deferred dependiendo del lenguaje (Linux-Fundation, 2019).

1.4. GitHub

1.4.1. Concepto, funcionalidad y origen.

Se describe a GitHub como un repositorio de control de versiones basado en la web que permite a los desarrolladores alojar, colaborar y supervisar las versiones de su código fuente. Los equipos de desarrollo pueden colaborar en proyectos y compartir su trabajo con otros usuarios en una plataforma fácilmente accesible y colaborativa llamada GitHub. A continuación, se detallaron las funcionalidades clave de GitHub. Estas incluyen:

Repositorios: Los usuarios pueden establecer repositorios para almacenar y organizar su código fuente. En el mismo repositorio, pueden monitorear los cambios, revertir a versiones anteriores y colaborar con otros desarrolladores.

Ramas (branches): Las ramas permiten a los desarrolladores trabajar en paralelo en una variedad de características o soluciones sin afectar el proyecto principal. 28 Esto facilita el desarrollo y la colaboración simultáneas en varios aspectos del proyecto.

Solicitudes de extracción (Pull Requests): Las solicitudes de extracción son una forma de sugerir cambios en un repositorio. Los desarrolladores pueden enviar solicitudes de extracción para que otros revisen y aprueben sus cambios antes de fusionarlos con el departamento principal del proyecto.

Problemas (Issues): Para rastrear errores, solicitudes de funciones o tareas pendientes en un proyecto, se utilizan problemas. Dentro del repositorio, los desarrolladores y colaboradores pueden comentar, asignar y resolver problemas.

Wikis y documentación: GitHub permite la creación de wikis y documentación que estén relacionadas con un repositorio, lo que facilita el trabajo en equipo y el intercambio de conocimientos.

GitHub cuenta con una estructura de trabajo muy intuitiva, hecha para los desarrolladores que requieren realizar sus proyectos colaborativos de una manera rápida e intuitiva. En la figura 10 se puede apreciar la estructura básica de trabajo de GitHub (Michael Avila Armas Director & Pamela Guevara Vega, n.d.).

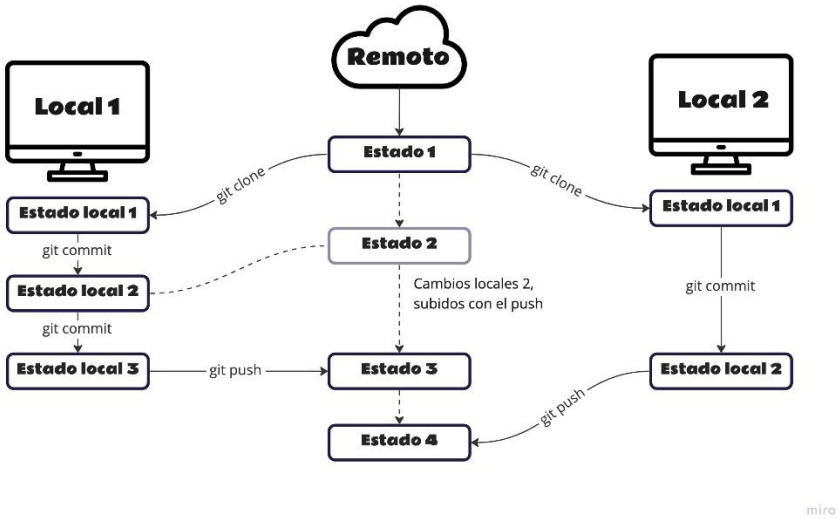


Figura 10. Estructura basica de GitHub
Fuente: https://leondesarrollo.es/git/primeros-pasos-con-git/

Finalmente, examinamos cuán importante es incorporar la API de GitHub en una aplicación orientada a servicios. La API de GitHub permite que las aplicaciones accedan de manera programática a los repositorios, ramas, solicitudes de extracción, problemas y otras funcionalidades de GitHub. Esto proporciona ventajas significativas, como una mayor automatización de tareas, una mejor sincronización entre la aplicación y GitHub y una mayor eficiencia en la gestión de proyectos de desarrollo de software. Por último, pero no menos importante, la integración de la API de GitHub en una aplicación orientada a servicios permite aprovechar las funcionalidades y la colaboración de GitHub para mejorar la gestión de proyectos de desarrollo de software (Michael Avila Armas Director & Pamela Guevara Vega, n.d.).

1.4.2. Ventajas y desventajas de GitHub.

Ventajas	**Desventajas**
Facilita el control de versiones del código fuente del sistema de inventario, permitiendo mantener un historial completo de cambios.	Requiere conocimientos técnicos y experiencia en el uso de control de versiones para aprovechar todas sus funcionalidades.
Permite la colaboración eficiente entre múltiples desarrolladores o equipos, facilitando la coordinación y la resolución de problemas	La gestión de archivos binarios o grandes volúmenes de datos puede ser menos eficiente en comparación con archivos de texto plano.
Proporciona un entorno seguro y confiable para almacenar y respaldar el código fuente del sistema de inventario.	Puede haber costos asociados con el uso de características y servicios avanzados de GitHub, como repositorios privados o integraciones adicionales.
Ofrece herramientas y funcionalidades adicionales, como problemas (issues) y solicitudes de extracción (pull requests), para facilitar el seguimiento y la resolución de problemas en el sistema de inventario.	Puede haber una curva de aprendizaje inicial para aquellos que no están familiarizados con el flujo de trabajo y la terminología de GitHub.
Proporciona integraciones con otras herramientas y servicios populares, lo que permite una mayor automatización y personalización del flujo de trabajo del sistema de inventario.	Dependiendo de los requisitos de seguridad y privacidad de la empresa, puede haber preocupaciones sobre el almacenamiento y la propiedad de los datos en un entorno externo.

Tabla 1.Ventajas y desventajas de GitHub

CAPÍTULO II

2. Desarrollo

Este capítulo describe métodos para inspeccionar los repositorios de GitHub para medir las tendencias de uso de GraphQL. Detallará los pasos necesarios para seleccionar un repositorio, los criterios de inclusión y exclusión, las herramientas de recopilación de datos y los métodos de análisis utilizados. Además, se explicará el proceso de desarrollo de las herramientas y scripts necesarios para realizar este proyecto de tesis.

2.1. Análisis y diseño de la aplicación

2.1.1. Estilo de desarrollo de software

Es un patrón de diseño de software que se utiliza para dividir la lógica de la aplicación en tres componentes interconectados.

Modelo: controla los datos de la aplicación y la lógica empresarial. Se encarga de acceder a la base de datos, realizar cálculos y gestionar el estado de la aplicación.

Pantalla: La interfaz del usuario permite la interacción entre lo visual con el cliente. Su función principal es representar los datos proporcionados por el modelo de forma adecuada.

Controlador: Actúa como el mediador entre el modelo y la vista. Recibe la informacion del usuario desde la parte visual, y luego procede a procesar la informacion y luego actualiza la vista para reflejar cualquier cambio.

Este proceso nos ayuda a organizar el código de una manera que se puede diferenciar claramente las responsabilidades, lo que hace que la aplicación sea más facil de comprender y la hace modular.

2.1.2. SCRUM

Scrum es un marco ágil para gestionar y ejecutar proyectos complejos, especialmente en el desarrollo de software. Se basa en iteraciones cortas y regulares llamadas "sprints" que normalmente duran de una a cuatro semanas.

Scrum tiene varios roles principales:

Product Owner: la persona responsable de definir las características y prioridades del producto y de mantener al equipo comprometido con las tareas más importantes.

Scrum Master: actúa como facilitador del equipo, ayuda a resolver obstáculos y garantiza el cumplimiento de las prácticas de Scrum.

Equipo de desarrollo: un grupo de profesionales que trabajan juntos para generar crecimiento del producto al final de cada sprint. Scrum se basa en ciertos eventos y artefactos:

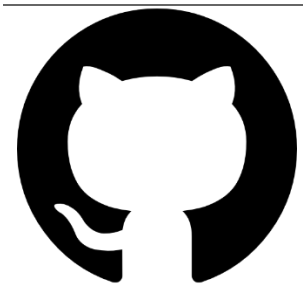
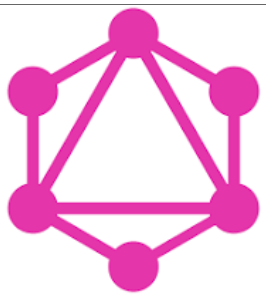
Planificación de Sprint: una reunión al comienzo de cada sprint para decidir qué trabajo se debe realizar.

Reunión Diaria: Una breve reunión diaria para sincronizar actividades y eliminar obstáculos.

Revisión de Sprint: al final de cada sprint, se revisa el progreso y se recopilan comentarios.

Retrospectiva del Sprint: Reflexione sobre el sprint completado para identificar mejoras continuas en los procesos.
Los artefactos incluyen el trabajo pendiente del producto (una lista priorizada de todo lo que se puede hacer en el producto), el trabajo pendiente del sprint (las tareas que se completarán durante el sprint) y la escalada (el resultado del trabajo del sprint que debe ser utilizable). Scrum incentivo a la total transparencia, la inspección y la adaptación, permitiendo a los equipos de desarrollo responder de una manera totalmente eficaz al cambio y mejorar continuamente la forma en que trabaja el team.

2.1.3. Herramientas de trabajo

LOGO	NOMBRE	DESCRIPCION
	Visual Studio Code	Es un editor de código fuente gratuito y de código abierto desarrollado por Microsoft. Está diseñado para ser ligero pero potente, ofreciendo una variedad de características que lo hacen ideal para desarrolladores de todos los niveles.
	**Postman**	Una herramienta popular para probar y documentar APIs es Postman. Los desarrolladores pueden usar una API para enviar solicitudes HTTP y HTTPS y recibir y analizar las respuestas. Con Postman, es rápido y sencillo realizar pruebas de ¿integración, depuración y validación de API.
	**GitHub**	GitHub es una plataforma de desarrollo colaborativo y de alojamiento de código fuente que utiliza el sistema de control de versiones Git. Fue creada para ayudar a los desarrolladores a gestionar, revisar y colaborar en proyectos de software.
	**GraphQL**	GraphQL es un lenguaje de consulta para APIs y un entorno de ejecución que permite a los clientes solicitar únicamente los datos que necesitan. Desarrollado por Facebook en 2012 y liberado como código abierto en 2015, GraphQL ofrece una forma más eficiente y flexible de interactuar con las APIs en comparación con REST.

	**React Vite**	Vite es una herramienta de construcción y desarrollo rápida para aplicaciones web modernas, creada por Evan You, el creador de Vue.js. Aunque Vite es agnóstico respecto al marco, se utiliza ampliamente con React debido a su eficiencia y facilidad de configuración.
	**Mui**	Es una biblioteca de componentes de interfaz de usuario (UI) para React que implementa las directrices de diseño de Material Design de Google. MUI facilita la creación de aplicaciones web con una apariencia moderna y coherente.
	**NodeJs**	Node.js es un entorno de ejecución de JavaScript en el lado del servidor, construido sobre el motor V8 de Chrome. Permite a los desarrolladores utilizar JavaScript para crear aplicaciones de red rápidas y escalables.

Tabla 2.Herramientas de trabajo

2.1.4. Configuración de GitHub

Para la obtención de la mayor cantidad de datos directos de GitHub se requiere una serie de caracteres a los que se los denomina TOKEN'S que son una clave de acceso que te permite como usuario realizar consulta más reforzadas mediante el consumo del API de GitHub.

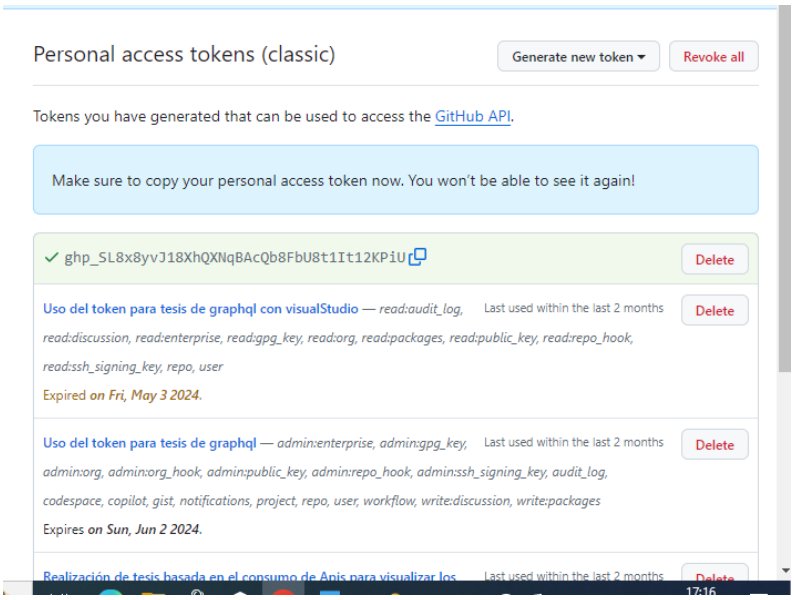


Figura 11. Creación de Tokens

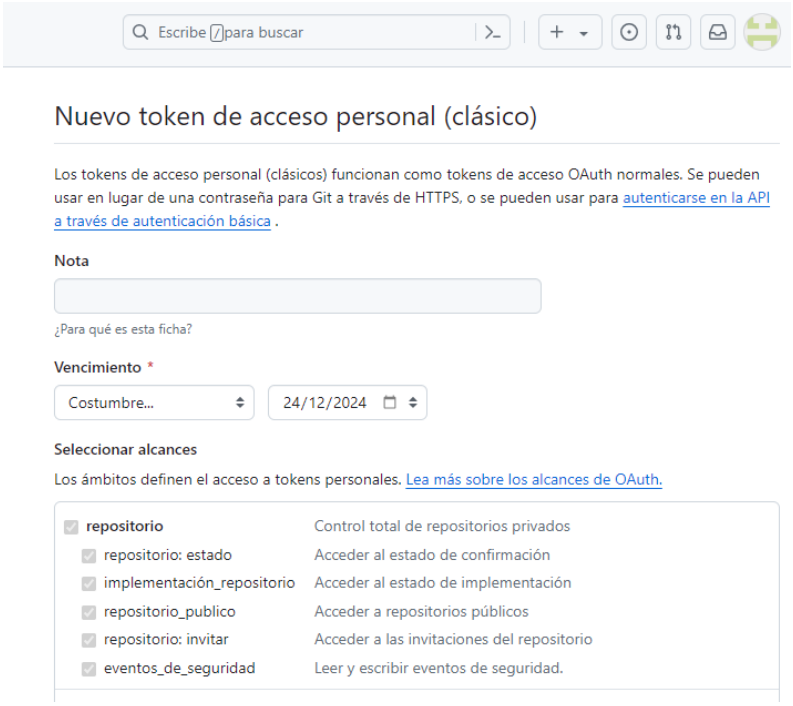


Figura 12. Creación de Tokens 2

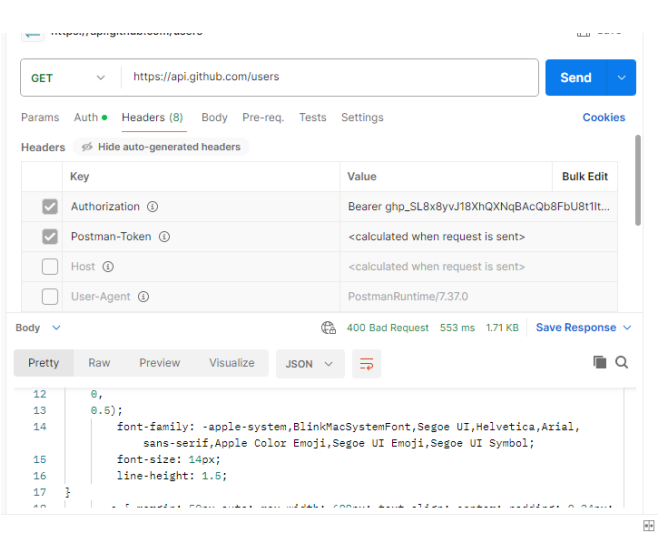


Figura 13. Petición en PostMan del API de GitHub

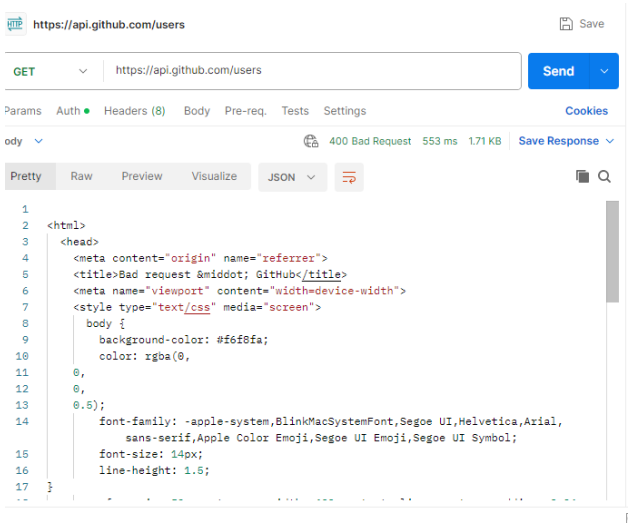


Figura 14. Peticion en PostMan del APU de GitHub 2

2.2. Desarrollo

2.2.1. Sprint 0

En el presente proyecto se utilizará el API de GitHub para consumir todos los proyectos de software que estén realizados con GraphQl para medir la tendencia y poder identificar sus componentes.

En este proceso se verán involucradas técnicas, tecnologías, lenguajes de programación que ayudarán al desarrollo al progreso del proyecto, serán limitados al comienzo por el desarrollo y serán usados en el proceso de implementación de la arquitectura orientada a servicios.

Se debe tomar en cuenta que las librerías, dependencias, lenguajes de programación y clases estarán ordenadas estructuralmente. Para garantizar que todas las tareas se lleven a cabo correctamente, se desarrollarán mediante la metodología scrum y el uso apropiado de sus herramientas que se describen en la siguiente tabla.

2.2.1.1. Roles del proyecto

Nombre	**ROL**	Descripción
PhD. Cathy Guevara	**Product Owner**	Dar seguimiento a que las funcionalidades sean correctamente implementadas en el aplicativo.
PhD. Cathy Guevara	**Scrum Master**	Persona que tiene la facultad de entender y dar las pautas a poner en práctica en las reglas, puntos e hitos del marco de trabajo Scrum, para que de tal forma el producto sea desarrollado de la mejor manera.
Stalin André Maigua Gordillo	**Deveplopment Team**	Un equipo de desarrollo, también conocido como equipo de desarrollo de software o equipo de ingeniería, es un grupo de personas que trabajan para crear, desarrollar y entregar soluciones de software.

Tabla 3.Roles del proyecto

2.2.1.2. Historias Épicas

Épicas nos ayudan a agrupar grandes conjuntos de historias de usuarios que no puedes ser realizadas o terminadas en un solo sprint debido a su tamaño y complejidad. A continuación, se muestran dos Historias épicas que deben ser cumplidas simultáneamente.

Código	**Titulo**	**Prioridad**
HE.01	Desarrollar aplicación para consumir el Api de GitHub, arquitectura MVC	Alta
HE.02	Obtener los proyectos de software para identificar la tendencia	Alta

Tabla 4.Historias épicas

2.2.1.3. Producto Backlog

Las historias épicas, que se dieron a conocer anteriormente, se dividieron en varias historias de usuario para que mediante estas se puedan cumplir con los requisitos funcionales y no funcionales, a la par se estableció prioridades y pesos, para construir los componentes que tendrá nuestra arquitectura.

Al ser una metodología ágil, Scrum permite reducir los tiempos con mayor facilidad, lo que permitirá realizar cada Sprint en menor tiempo, a continuación, se muestra el Product Backlog.

Historias de Usuario				
H.Épica	**Código**	**Titulo**	**Peso**	**Prioridad**
E-PROD-1	PB.01	Preparar el ambiente de trabajo GitHub.	1	Alta
E-PROD-1	PB.02	Desarrollar un modelo de base de consumo.	2	Alta
E-PROD-1	PB.03	Diseño de interfaces de usuario en Balsamiq.	2	Alta
E-PROD-1	PB.04	Diseño de arquitectura	2	Alta
E-PROD-1	PB.05	Levantar ambiente de desarrollo con Visual Studio code	1	Alta
E-PROD-1	PB.06	Integrar pluguis, herramientas para el desarrollo	2	Alta
E-PROD-2	PB.07	Implementar clase services para el consumo de Apis	2	Alta

E-PROD-2	PB.08	Implementar apartado de controllers para los repositorios de GitHub	2	Alta
E-PROD-2	PB.09	Implementar aparto de componentes que se usaran en el desarrollo	3	Alta
E-PROD-2	PB.10	Al usar el aplicativo el usuario debe se colocar un token de verificación	2	Alta
E-PROD-2	PB.11	Al usar el aplicativo el usuario debe colocar el nombre del lenguaje principal que desee indagar	2	Alta
E-PROD-2	PB.12	Al ser usuario puede elegir un proyecto para visualizar mejor las características que tienen	3	Alta
E-PROD-3	PB.13	Al ser usuario puede determinar cuántos proyectos desea que aparezcan	2	Alta
E-PROD-3	PB.14	Los datos que se consumen a través de GitHub se reflejaran automáticamente en el aplicativo	1	Alta
E-PROD-3	PB.15	Gestionar componentes de GraphQL	4	Alta

Tabla 5. Product Backlog

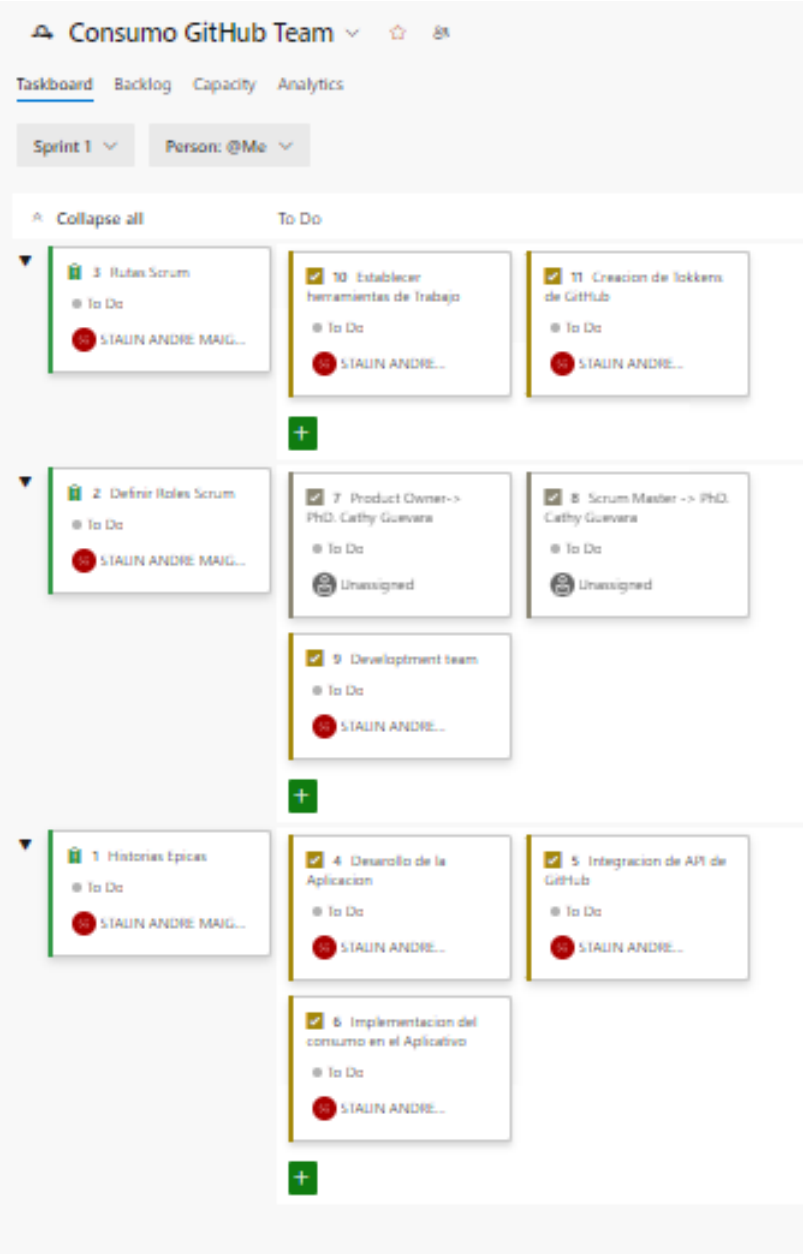


Figura 15. Azure DevOps - Sprint 0

2.2.2. Sprint 1

2.2.2.1. Sprint Planning

El objetivo del primer sprint fue realizar el diseño del aplicativo. Se concordaron los siguientes objetivos:

- Diseñar la arquitectura global, consumo de servicios.
- Diseñar las interfaces del aplicativos (Balsamiq)
- Levantar ambiente de trabajo
- Generar Tokkens de acceso.

2.2.2.2. Sprint Backlog

Para realizar el primer Sprint, se tomaron las historias de usuario en el Anexo número 1, donde se acogen los criterios de aceptación y las tareas a realizar.

2.2.2.3. Ejecución del Sprint.

Todos los servicios necesarios para la realización del consumo del API de GitHub fueron realizados primeramente en un ambiente de prueba tal como Visual Studio Code. Se priorizo la arquitectura del proyecto para tener un mejor orden en donde se pudiera evidenciar los módulos del proyecto, como se muestra en la siguiente Figura.

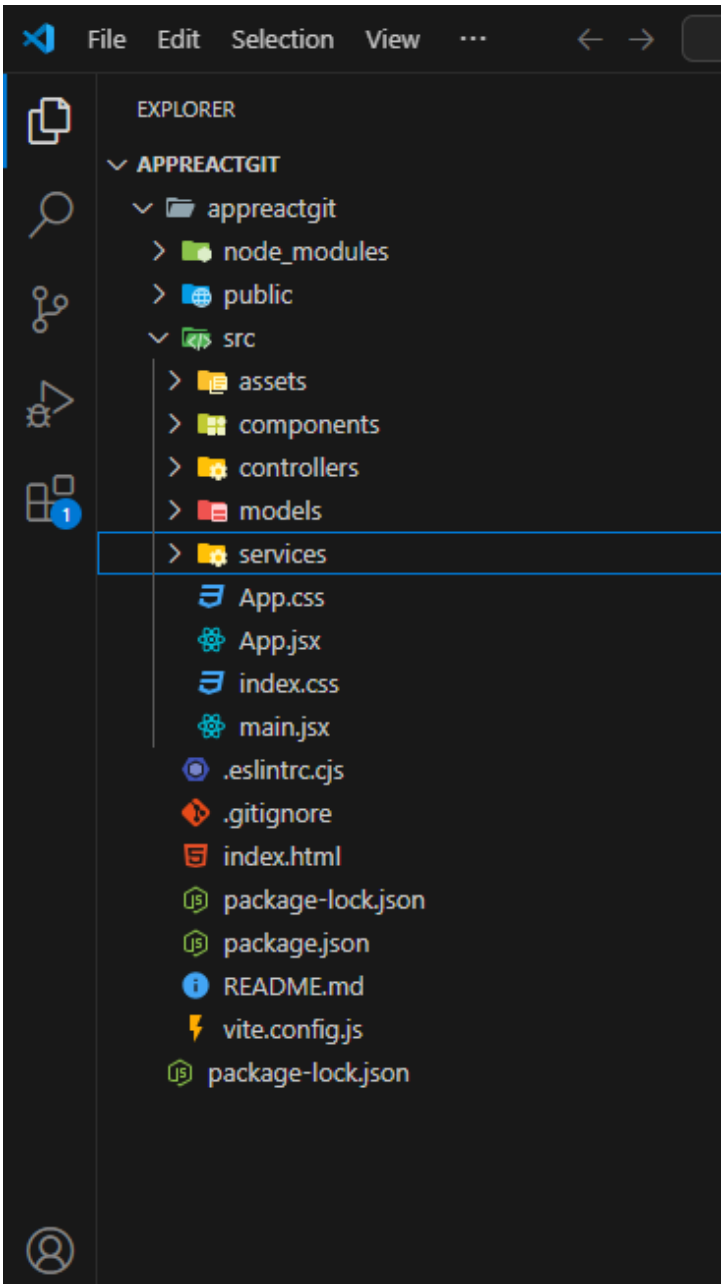


Figura 16. Arquitectura del Proyecto

Consumo del API de GitHub, aplicación

Cada una de las capas cumplen una función en específico lo que permite que la aplicación funcione de la mejor manera, en la capa de services se encuentra el consumo del API y los TOKKENS de acceso, en la capa de models se encuentran los datos necesarios que queremos obtener al consumir la API, en la capa de componentes se encuentre la lógica del proyecto.

Interfaces de usuario- Balsamiq

El objetivo de la maquetación actual era crear interfaces de usuario modernas y fáciles de usar. Para ello, se utilizaron Sass (Syntactically Awesome Style Sheets) y elementos responsivos. Estos componentes, al ser ejecutados directamente en scripts interpretados por el navegador, son rápidos y fáciles de manejar. Se diseñó un esquema lógico de navegación una vez definidas las tecnologías frontend, lo que permite desarrollar interfaces de manera estructurada y mantener una coherencia global en la aplicación.

Durante esta investigación, se utilizó Balsamiq, una herramienta eficaz para crear prototipos rápidos de interfaces de usuario, facilitando la comunicación y la colaboración en las fases iniciales del diseño. Esto permitió enfocarse en la funcionalidad y estructura de las interfaces, ahorrar tiempo y costos, y recibir valiosos comentarios antes de comenzar el desarrollo completo. Para mejorar la interacción del usuario con el software, se definieron varias capas, siendo las más destacadas:

- **Página principal:** En esta página es donde el usuario buscara los proyectos de acuerdo al lenguaje de programación que desee indagar.

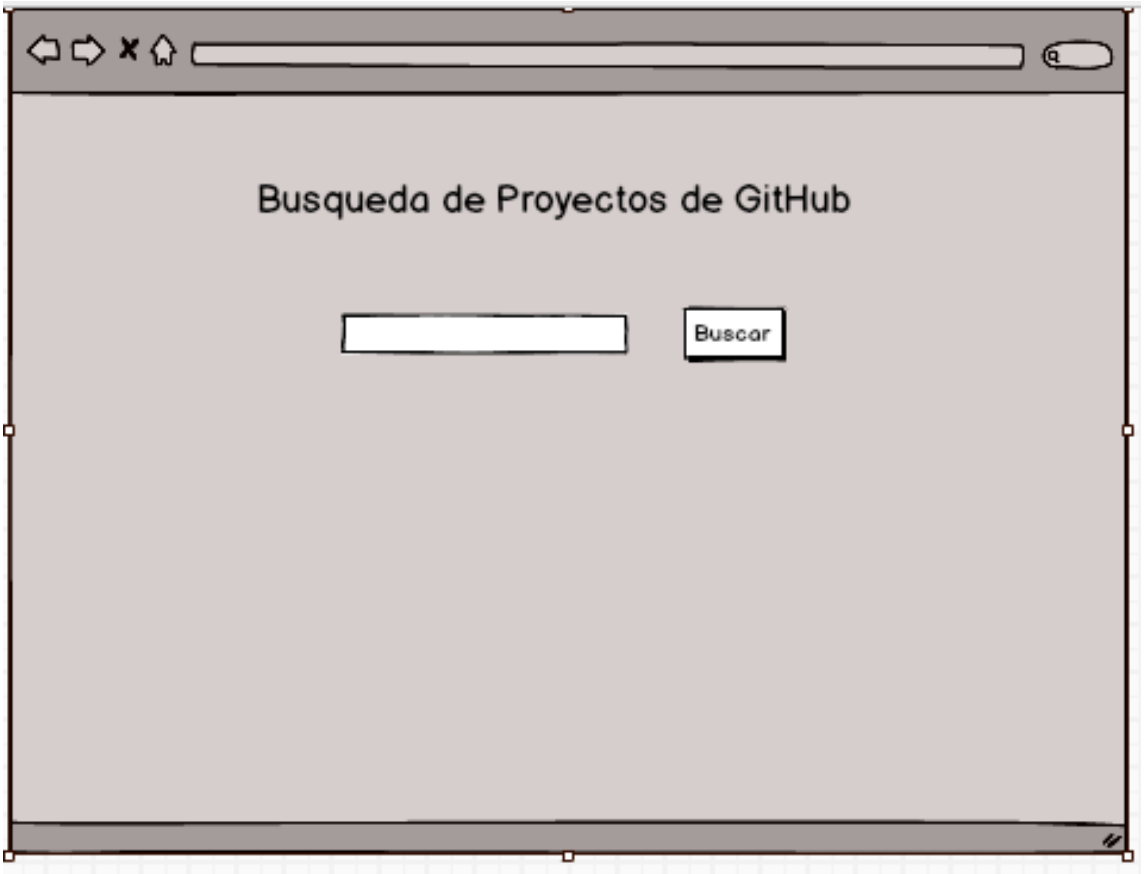


Figura 17. Diseño Web, Home

- **Zona de información:** En donde se va a mostrar la información obtenida del consumo del Api mediante el botón de búsqueda como se observa en la siguiente figura.

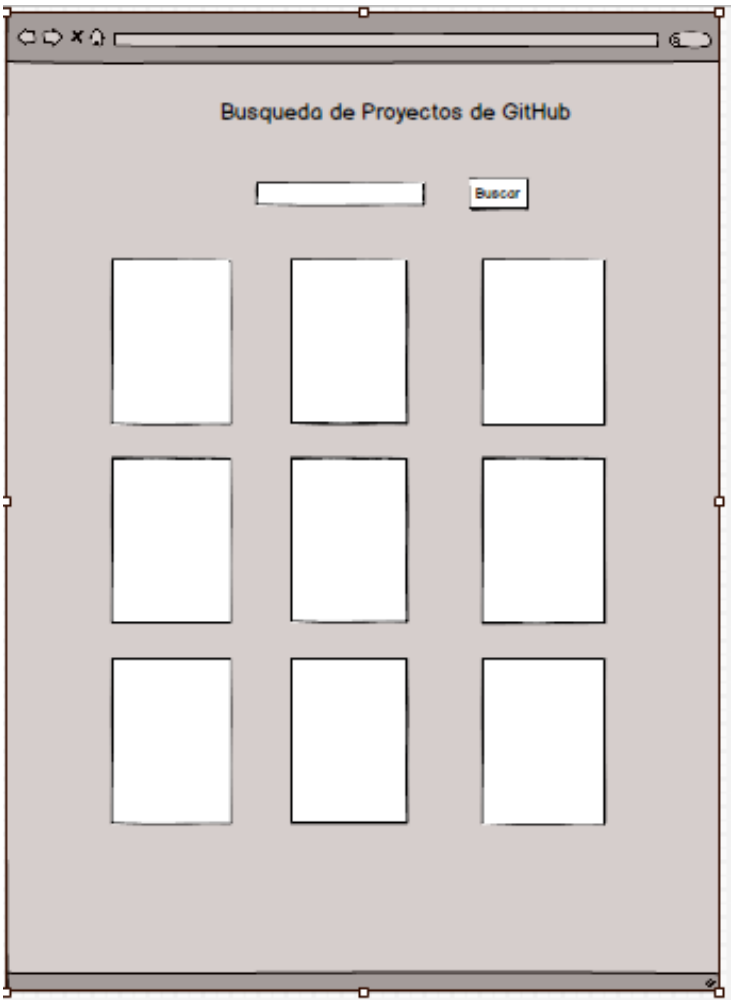


Figura 18. Diseño Web, zona de información

Como resultado, se crearon nuevas imágenes y diseños del producto final. Cabe destacar que una de las principales ventajas de utilizar Balsamiq es su área de pruebas en un entorno de simulación, que permite visualizar cómo se verán las interfaces una vez desarrolladas. Se llevaron a cabo las pruebas necesarias de la UI, las cuales incluyeron:

- El sitio web será responsivo.
- El sitio web contendrá contenido relevante para cada una de las áreas identificadas en el diagrama de navegación.
- El sitio será sometido a pruebas de rendimiento con frecuencia, lo que mejorará la velocidad y la capacidad de respuesta.

Arquitectura del aplicativo

En la siguiente figura se muestra la arquitectura del proyecto

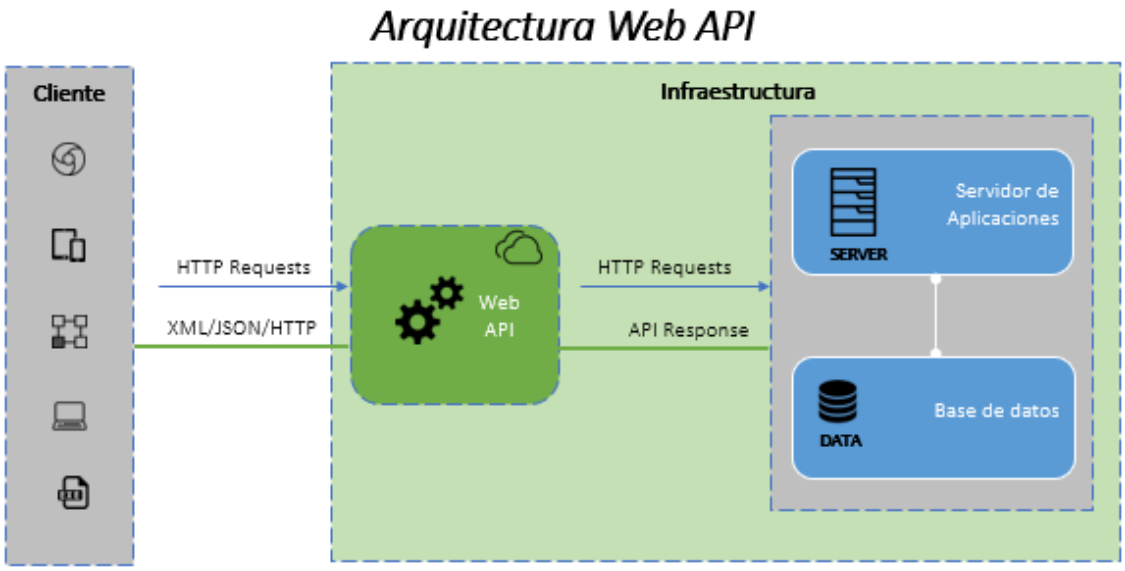


Figura 19. Arquitectura del Web API

2.2.2.4. Sprint Review

Las historias de usuario llevadas acabo se las pueden evidenciar en la siguiente tabla

Historias de Usuario				
H.Épica	**Código**	**Titulo**	**Peso**	**Prioridad**
E-PROD-1	PB.01	Preparar el ambiente de trabajo GitHub.	1	Alta
E-PROD-1	PB.02	Desarrollar un modelo de base de consumo.	2	Alta
E-PROD-1	PB.03	Diseño de interfaces de usuario en Balsamiq.	2	Alta
E-PROD-1	PB.04	Diseño de arquitectura	2	Alta

E-PROD-1	PB.05	Levantar ambiente de desarrollo con Visual Studio code	1	Alta
E-PROD-1	PB.06	Integrar pluguis, herramientas para el desarrollo	2	Alta

Tabla 6. Sprint Review

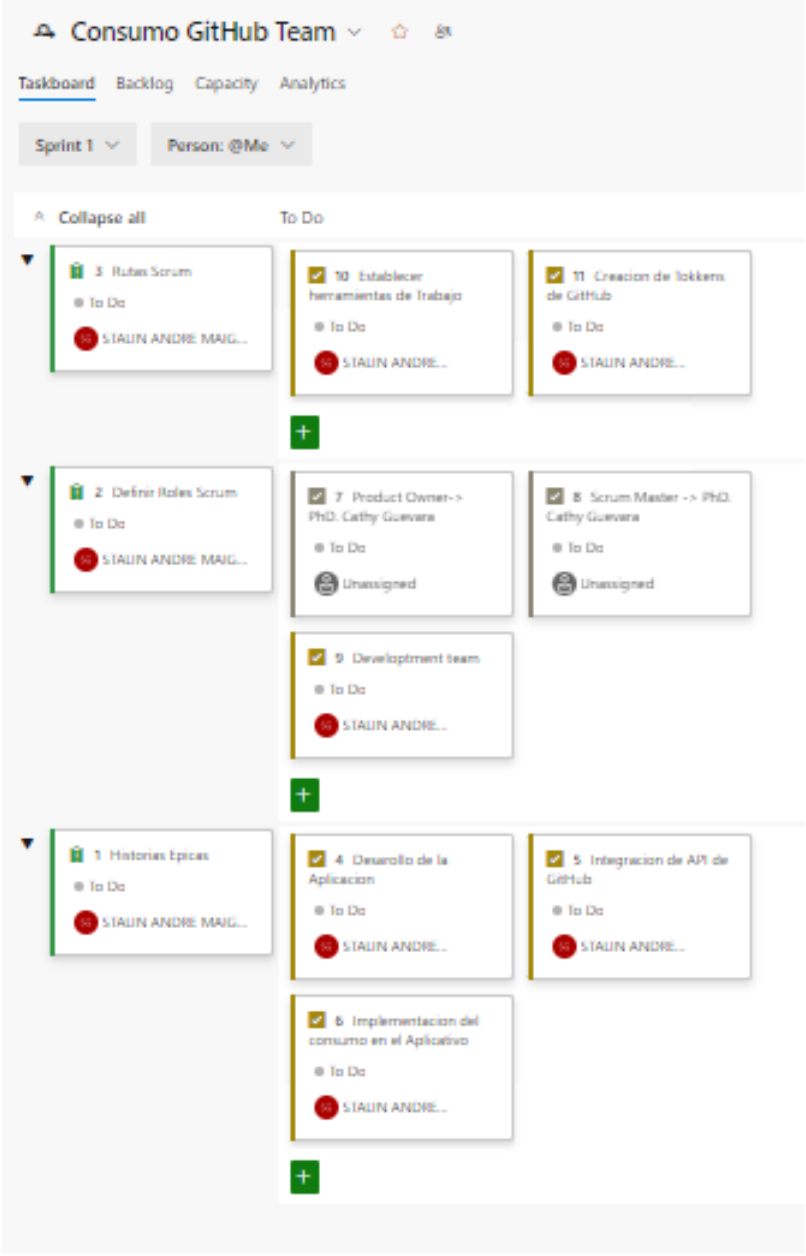


Figura 20. Sprint 1- Azure DevOps

2.2.2.5. Sprint Retrospective

¿Qué salió bien en el sprint?

Los entornos de trabajo, las instancias del API y otros servicios iniciales se instalaron correctamente, y todos los cambios en los componentes y paquetes se realizaron de la mejor manera. Los usuarios que acceden al front-end disfrutan de interfaces de usuario rápidas y responsivas, y se han utilizado herramientas de pruebas de rendimiento para solucionar los errores que podrían surgir al desplegar todos los componentes en producción.

¿Qué se aprendió del sprint para mejorar el proyecto?

Para mejorar el proyecto, será necesario implementar gradualmente los paquetes requeridos y controlar las versiones de cada una de las librerías. Las interfaces de usuario se desarrollarán paulatinamente con los componentes necesarios para que el sitio ofrezca una experiencia de usuario profesional y eficiente.

2.2.3. Sprint 2

2.2.3.1. Sprint Planning

Los propósitos para cumplir el segundo sprint son los siguientes:

- Establecer los controladores.
- Configurar los servicios interconectados con el API de GitHub.
- Desarrollar los modelos requeridos.
- Uso del Software Postman para probar el API

Mapa de procesos General

En el siguiente mapa se automatizaron algunos procesos desde sus primeras etapas de la ejecución de los sprints, de los cuales se obtuvo una mejor visión del alcance del proyecto. El mapa de procesos se muestra a continuación en la siguiente Fing.20.

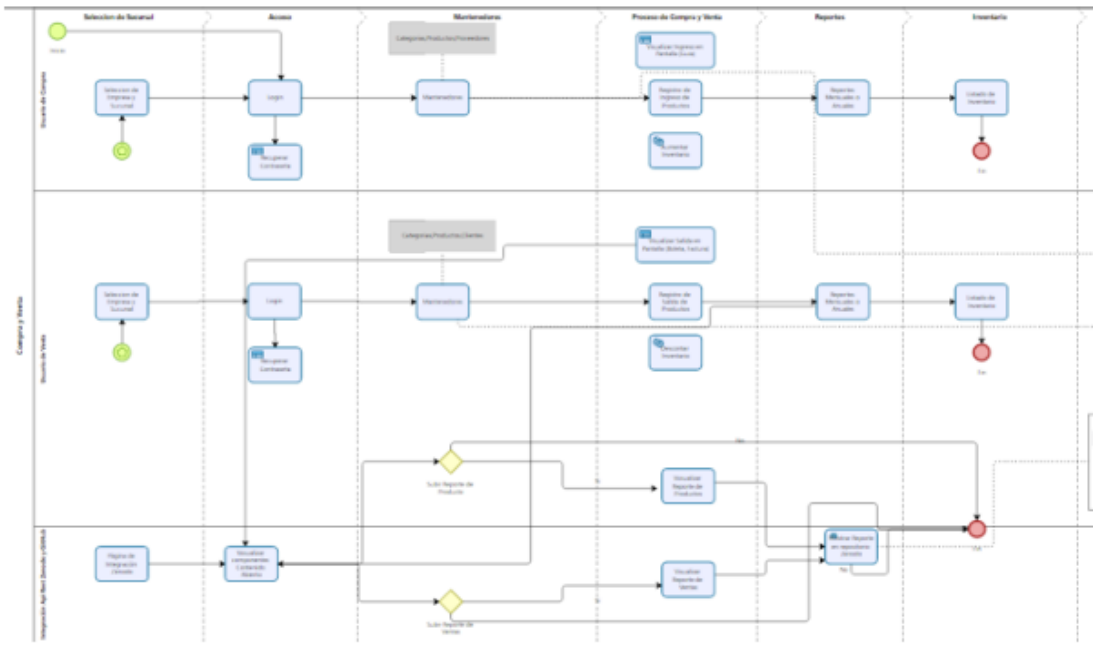


Figura 21. Diagrama Menú

Menú principal- Aplicativo

En la página principal se puedes observar en donde podremos usar el token de autenticación del API de GitHub en la Fig.22.

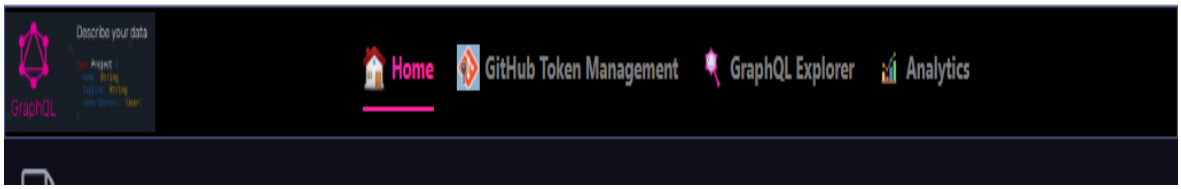


Figura 22. Menú

Pantalla principal

En la pantalla principal se puede evidenciar el cuadro de texto donde se puede indagar el lenguaje de programación que se requiera buscar. En la Fig.23

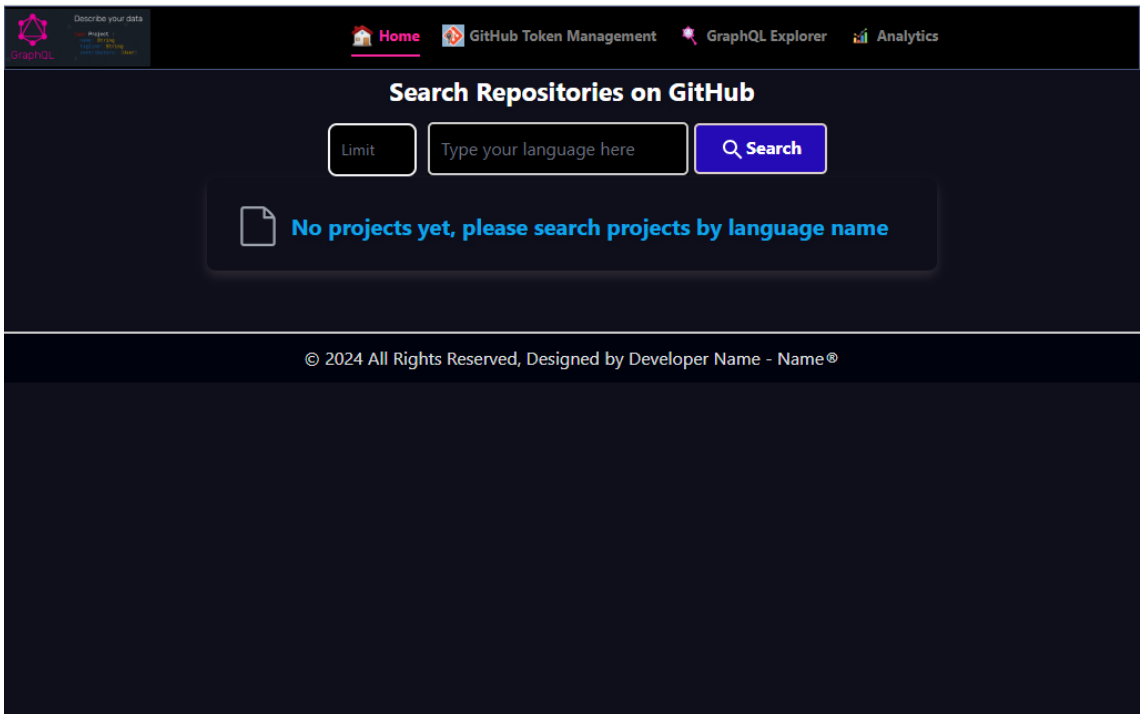


Figura 23. Pantalla principal

Página principal – búsqueda de datos

En el cuadro de texto donde se ingresa el lenguaje de programación que se desea buscar, se visualizara en forma de cartas los proyectos que se encuentran del repositorio. Ver en la Fig. 24.

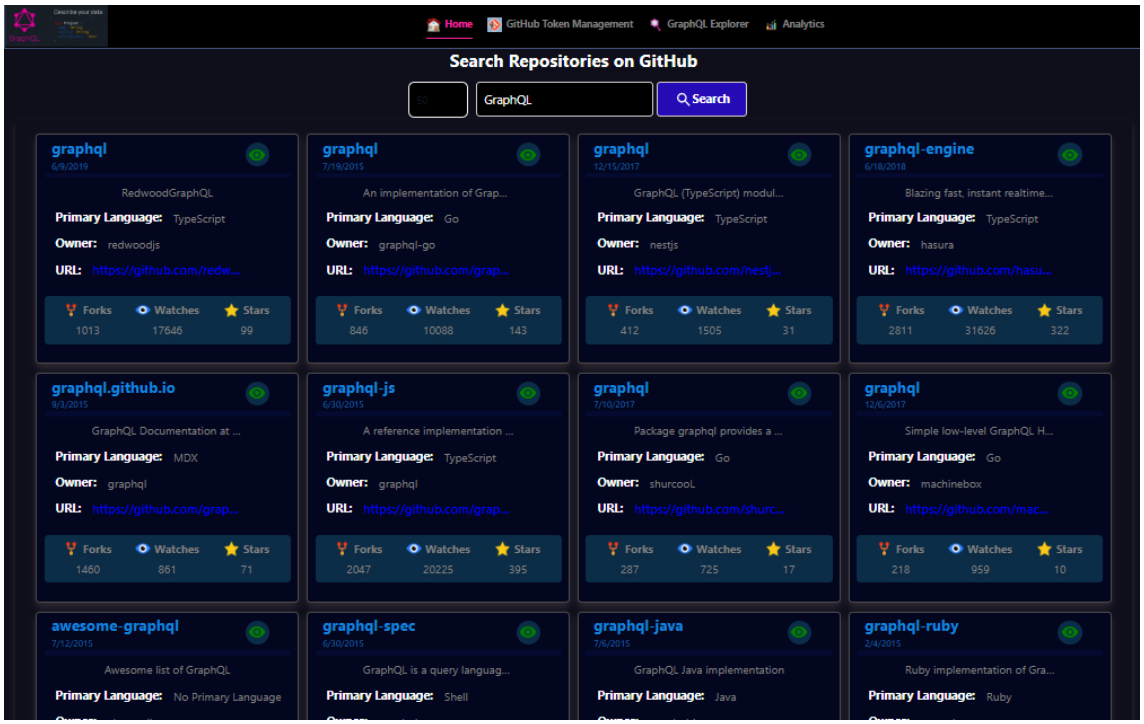


Figura 24. Pantalla Principal Búsqueda

Página Secundaria – GitHub Managment

En esta página se puede visualizar los Tokens que se tiene y se pueden volver a usar dependiendo del caso.

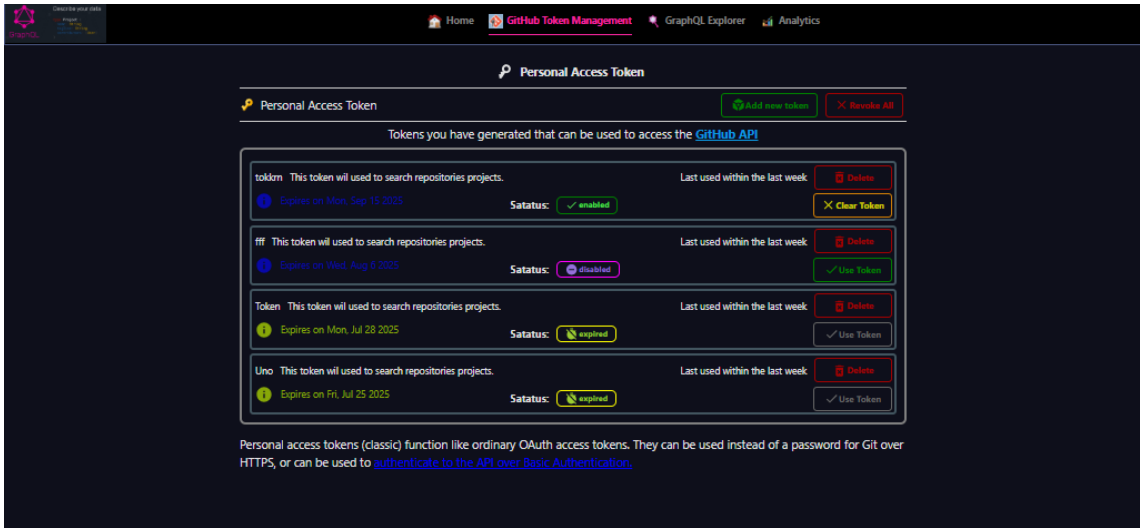


Figura 25. Pagina Secundaria - Tokens

Página Secundaria – GitHub Managment- Agregar nuevo token

En esta página se puede agregar un nuevo token en caso de requerirlo en los siguientes campos se coloca un nombre al token, el token extraído de GitHub y una pequeña descripción en caso de requerirlo.

Figura 26. Pagina Secundaria- Crear un Token

Página Terciaria – Analytics

En esta página podemos visualizar algunos gráficos que corresponden a las analíticas de los proyectos según los Watchers, Stars y Forks.

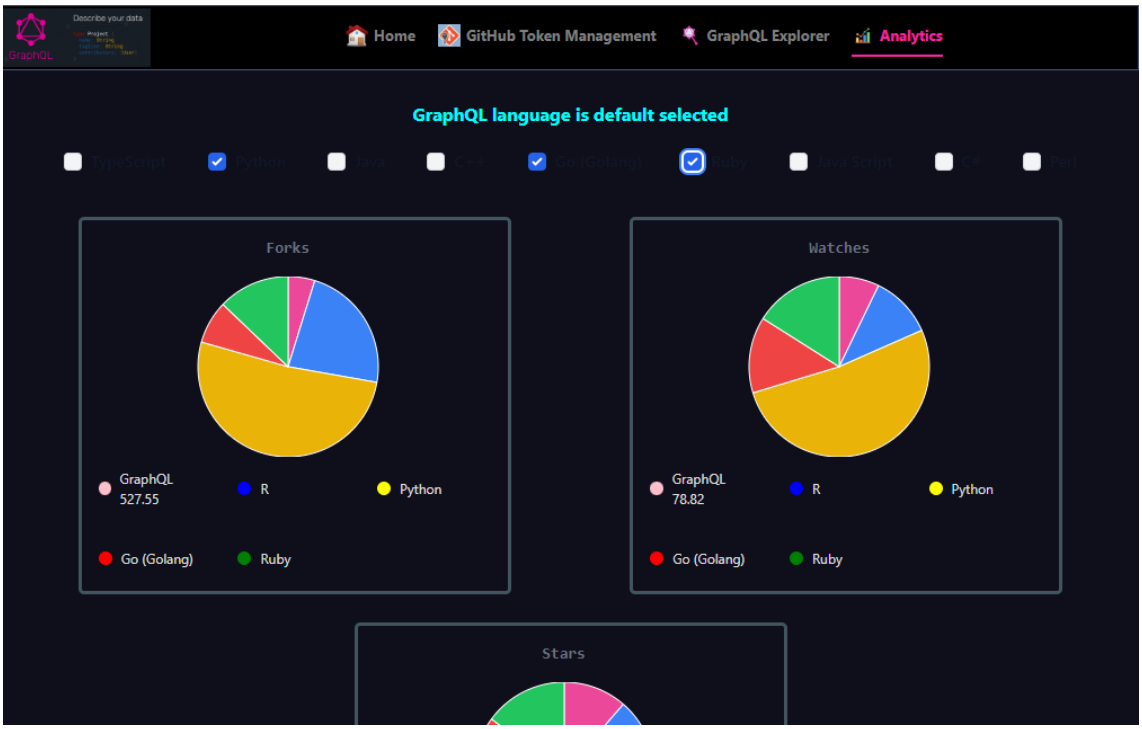


Figura 27. Pagina -Analytics

Figura 28. Página -Analytics 2

2.2.3.2. Sprint Review

Las historias de usuarios que se realizaron en este Spring se encuentran en la siguiente tabla.

E-PROD-2	**PB.07**	**Implementar clase services para el consumo de Apis**	**2**	**Alta**
E-PROD-2	PB.08	Implementar apartado de controllers para los repositorios de GitHub	2	Alta
E-PROD-2	PB.09	Implementar aparto de componentes que se usaran en el desarrollo	3	Alta
E-PROD-2	PB.10	Al usar el aplicativo el usuario debe se colocar un token de verificación	2	Alta
E-PROD-2	PB.11	Al usar el aplicativo el usuario debe colocar el nombre del lenguaje principal que desee indagar	2	Alta
E-PROD-2	PB.12	Al ser usuario puede elegir un proyecto para visualizar mejor las características que tienen	3	Alta

Tabla 7. Sprint Review 2

2.2.3.3. Spring Retrospective

¿Qué salió bien en el sprint?

Las interfaces del aplicativo se completaron exitosamente, Los usuarios que acceden al front-end disfrutan de interfaces rápidas y responsivas. Además, se utilizaron herramientas de pruebas de rendimiento para solucionar posibles errores al desplegar los componentes en producción.

¿Qué se aprendió del sprint para mejorar el proyecto?

Para mejorar el proyecto, es necesario implementar gradualmente los paquetes necesarios y asegurar las versiones de cada librería. Las interfaces de usuario se desarrollarán progresivamente con los componentes necesarios para ofrecer una experiencia profesional y eficiente.

2.2.4. Sprint 3

2.2.4.1. Spring Planning

Los objetivos para cumplir este Sprint son los siguientes:

- Mostrar una extensa cantidad de proyectos relacionados con el lenguaje de programación que se desea el usuario.

2.2.4.2. Spring Review

Las historias de usuario que se presentan a continuación fueron analizadas para este Sprint y que se lo realizara con estándares de aceptación y calidad.

2.2.4.3. Ejecución del Sprint

2.2.4.4. Spring Review

E-PROD-3	**PB.13**	**Al ser usuario puede determinar cuántos proyectos desea que aparezcan**	**2**	**Alta**
E-PROD-3	PB.14	Los datos que se consumen a través de GitHub se reflejaran automáticamente en el aplicativo	1	Alta
E-PROD-3	PB.15	Gestionar componentes de GraphQl	4	Alta

Tabla 8. Sprint Review 3

2.2.4.5. Sprint Retrospective

¿Qué salió bien en el sprint?

Se implementaron eficazmente todos los componentes necesarios en cada etapa del desarrollo del sprint actual, utilizando las herramientas adecuadas para el proyecto. La arquitectura se desplegó con CI/CD, y se instalaron componentes para validar el código en el repositorio de GitHub.

El uso de GitHub ha proporcionado muchos beneficios, como un entorno de desarrollo controlado con imágenes de versiona miento, facilitando la implementación de nuevas versiones en el hosting de Hostinger. Sin embargo, es necesario revisar la documentación de cada herramienta para asegurar la compatibilidad de todas las dependencias.

¿Qué se aprendió en el sprint que pueda optimizar el proyecto?

Con el objetivo de que cada rol fuera completamente, se incorporaron elementos a lo largo del desarrollo, acatando las buenas prácticas de la interfaz de usuario sugeridas por la documentación correspondiente a cada herramienta. A manera que la aplicación y su la arquitectura favorece, se destaca que el uso de tecnologías, Frameworks, servicios web y otro tipo de herramientas avanzadas tienen un repunte el valor del producto.

CAPÍTULO III

3. Aplicación del Survey a GitHub

3.1. Introducción

En este capitulo se presentará la ultima fase del proceso de investigación que se esta llevando acabo en la que se presenta se analizan los datos recolectados mediante el uso del aplicativo desarrollado en el capitulo anterior. Estos datos obtenidos nos van a permitir evaluar empíricamente el uso de GraphQL. La finalidad de este capitulo es validar el cumplimiento del tercer objetivo el cual es identificar los proyectos que

utilizan GraphQL para posteriormente determinar la tendencia del suso de este miso lenguaje.

3.2. Metodología de obtención de datos

La obtención de datos se realizo mediante el uso de una aplicación personalizada que principalmente realiza consultas hacia el API GraphQL de GitHub para identificar distintos proyectos de software de software que implementan GraphQL. La herramienta desarrollada fue configurada para reconocer y recolectar metadatos de los proyectos que se encuentran en el repositorio de GitHub principalmente los que estén asociados con el lenguaje de programación GraphQL.

Para respaldar la veracidad de los resultados obtenidos mediante el uso del aplicativo desarrollado, se establecieron filtros de inclusión y exclusión, así como criterios técnicos para validar el uso de GraphQL en los distintos proyectos. En el siguiente apartado se detallarán los aspectos aplicados en la recopilación de datos.

3.2.1. Configuración técnica y autenticación

La aplicación automatizada se introdujo en un entorno de aplicación continúa aprobado por el marcador personal GitHub para expandir las distintas restricciones que puede tener el acceso a la API. Era importante asegurarse de que las consultas no fueran rechazadas para exceder el límite de solicitud durante una hora. La lógica de reintentos con un rebote exponencial (retroceder exponencial) se incrustó para convertirse en un error de cara transitoria y evitar roturas durante el proceso de extracción.

3.2.2. Estrategia de búsqueda y extracción

Se empleó una búsqueda basada en topics y palabras clave como 'graphql', 'apollo', 'relay', 'graphql-java', entre otros. Además, se inspeccionaron archivos como package.json, go.mod y pom.xml para verificar la presencia de dependencias directas asociadas a GraphQL.

También se examinó la existencia de esquemas con extensiones '.graphql', '.gql' y '.graphqls' en carpetas comunes como /schema o /graphql.

3.2.3. Parámetros de búsqueda

Para asegurar la exactitud y la relevancia de los datos adquiridos, se plantea los siguientes criterios de busqueda

Criterios de inclusión:

- Repositorios que contengan archivos con extensiones relacionadas a GraphQL (.graphql, .gql)
- Proyectos que incluyan dependencias de GraphQL en sus archivos de configuración (package.json, pom.xml, requirements.txt, etc.)
- Repositorios con esquemas GraphQL definidos
- Proyectos que implementen servidores o clientes GraphQL
- Código que contenga importaciones o referencias explícitas a librerías GraphQL

Criterios de exclusión:

- Repositorios fork sin contribuciones significativas
- Proyectos con menos de 5 commits
- Repositorios sin actividad en los últimos 2 años
- Proyectos de prueba o tutoriales básicos (identificados por palabras clave como "test", "tutorial", "demo" en nombres o descripciones)
- Repositorios privados (por limitaciones de acceso del API)

3.2.4. Configuración técnica del Survey

En la aplicación se configuro para realizar el survey con el siguiente instrumento de recolección de datos.

Variable por recolectar	Descripción/ tipo de dato	Relevancia para la Tesis
ID del repositorio	Nombre completo (ej:	Identificador único para cada proyecto analizado.

	facebook/graphql-js). Dato Categórico.	
URL del repositorio	Enlace directo al repositorio. Texto.	Permite la verificación y el acceso directo a la fuente.
Lenguaje Principal	Lenguaje de programación principal detectado por GitHub. Dato Categórico.	Identifica las tecnologías más comunes usadas con GraphQL.
Fecha de creación	Fecha en que el repositorio fue creado. Fecha/Hora.	Clave para analizar la tendencia de adopción a lo largo del tiempo.
Numero de Estrellas (Stars)	Cantidad de estrellas del repositorio. Dato Numérico.	Métrica de popularidad e impacto en la comunidad.
Número de Bifurcaciones (Forks)	Cantidad de veces que el proyecto ha sido bifurcado. Dato Numérico.	Métrica de interés y reutilización por parte de otros desarrolladores.
Numero de ramas	Cantidad de ramas. Dato Numérico.	Métrica de interés por parte de desarrolladores para ver los involucrados del proyecto

Tabla 9. Configuración de Survey

3.3. Resultados Obtenidos

Luego de aplicar el Survey automatizado en el aplicativo se pudo definir una muestra de 715 proyectos que se encuentran activos en donde se usa el lenguaje de programación GraphQL. La informacion que se obtuvo se analizaron desde las perspectivas antes mencionadas como: Lenguaje principal, watchers, forks, componentes claves y la popularidad del proyecto.

Cada aspecto nos brinda informacion sobre la madurez que tiene GraphQL en la comunidad de desarolladores.

3.3.1. Distribución por lenguaje de programación

Uno de los pilares principales para realizar el análisis en el contexto tecnológico basado en GraphQL es el Lenguaje de programación en el que se basa cada proyecto que se ha identificado. Según los metadatos de GitHub, la manera en la que se encuentra distribuida por el lenguaje de los 715 proyectos fue la siguiente:

Lenguaje de programación	Porcentaje %
GraphQL	19.54%
R	27.02%
Java	100%
JavaScript	14.94%
Go	27.36%
Python	96.54%

Tabla 10. Distribución de Lenguajes de programación

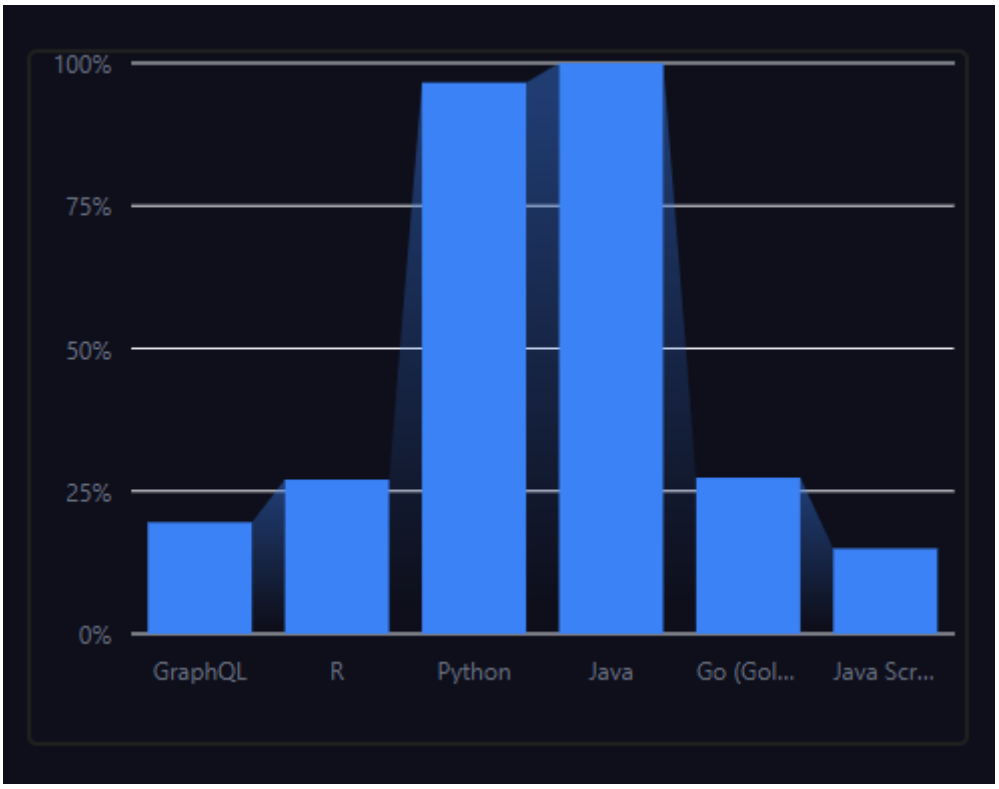


Figura 29. Gráfico de lenguajes de Programación

3.3.2. Distribución según Stars(Estrellas) en GitHub

La popularidad de un proyecto que se encuentra en un repositorio de GitHub se realiza su medición mediante el número de estrellas (Stars). Este indicador nos permite visualizar la importancia, el acoplamiento y la viabilidad de los proyectos dentro de los desarolladores.

Lenguaje de programación	Estrellas (Stars)
GraphQL	19.54%
R	27.02%
Java	35.44%
JavaScript	4.78%
Go	10.93%

Python	33.29%

Tabla 11. Distribución según Stars

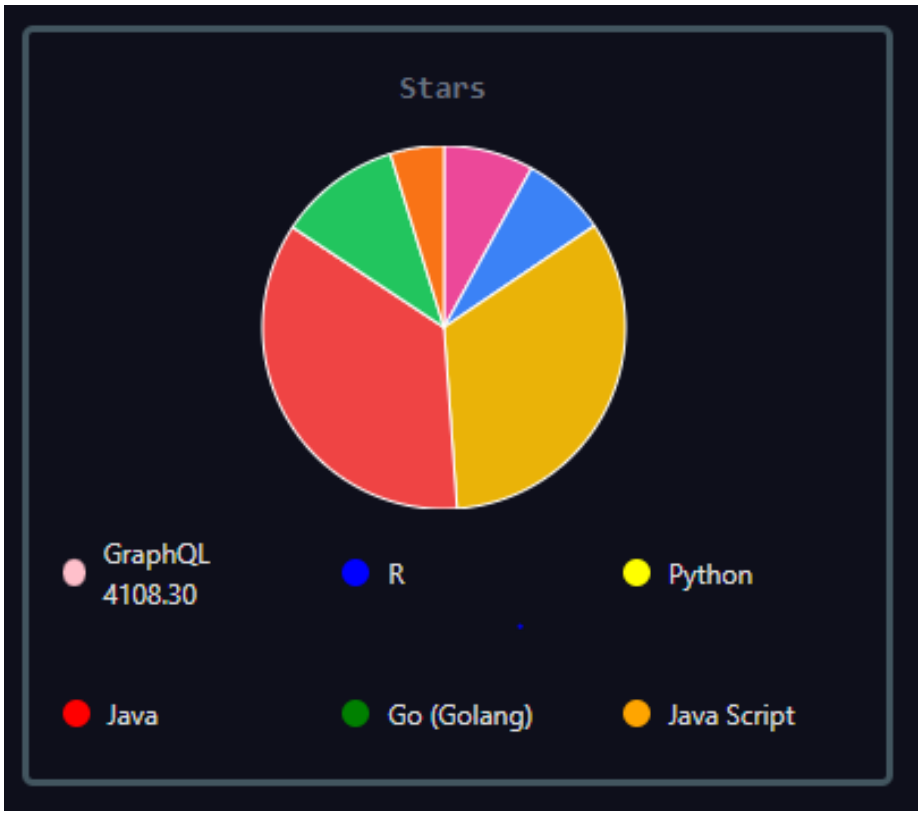


Figura 30. Diagrama de pastel según Stars

3.3.3. Distribución según Forks en GitHub

Uno de los aspectos que resalta un proyecto es el numero de copias que se realizan de un mismo proyecto el cual puede ser escalable esto quiere decir que en cada copia se agregan nuevas funcionalidades y mejoras al proyecto lo que genera un beneficio a la comunidad de desarolladores.

Lenguaje de programación	Forks
GraphQL	3.29%
R	15.76%
Java	33.59%

JavaScript	6.75%
Go	5.30%
Python	35.32%

Tabla 12.Distribución según forks

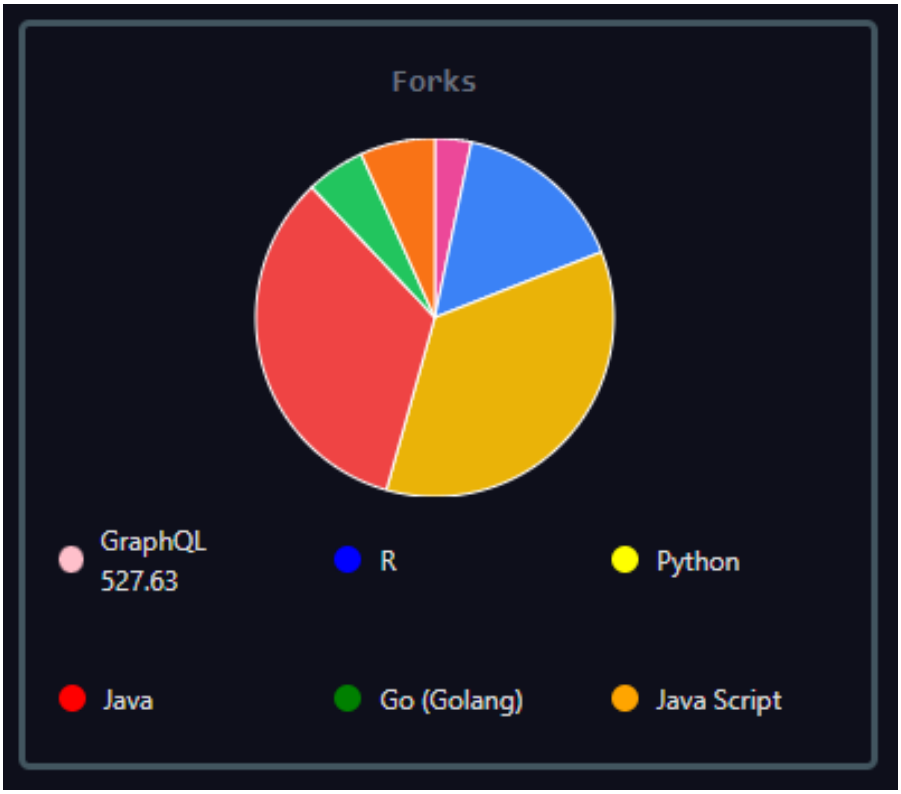


Figura 31. Diagrama de Pastel según Forks

3.3.4. Distribución según Watches de GitHub

Un proyecto mientras más visualizaciones más popularidad adquiere dentro del repositorio de GitHub que se convierte en una de las características principales para medir el nivel de visualizaciones que tiene un proyecto.

Lenguaje de programación	Watches

GraphQL	5.06%
R	7.89%
Java	36.37%
JavaScript	4.65%
Go	9.59%
Python	36.44%

Tabla 13.Distribución según Watches

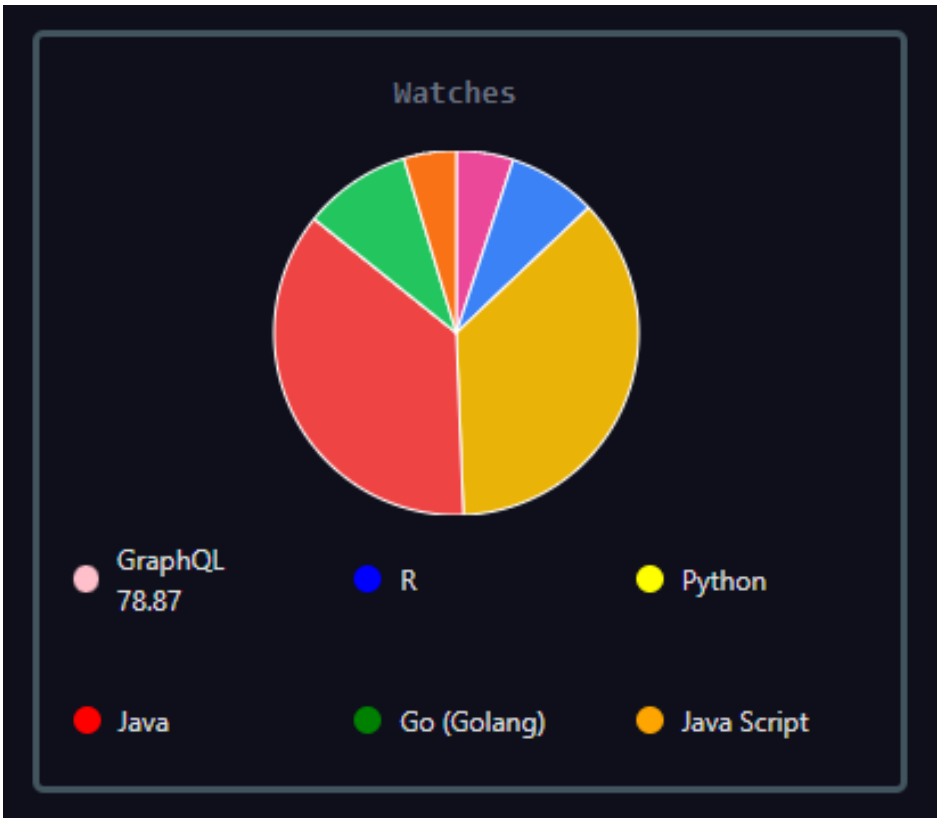


Figura 32. Diagrama de Patel según Watchers

3.3.5. Componentes de GraphQL utilizados

Se realizo una inspección en los esquemas de GraphQL para definir los componentes que predominan tales como: Queries, Mutations, Subscriptions, Interfaces, Unions y Directivas, los cuales proporcionaron los siguientes datos.

Componente	Porcentaje
Queries	100%
Mutations	97%
Subscriptions	45%
Interfaces	32%
Unions	25%
Directivas personalizadas	18%

Tabla 14. Componentes de GraphQL

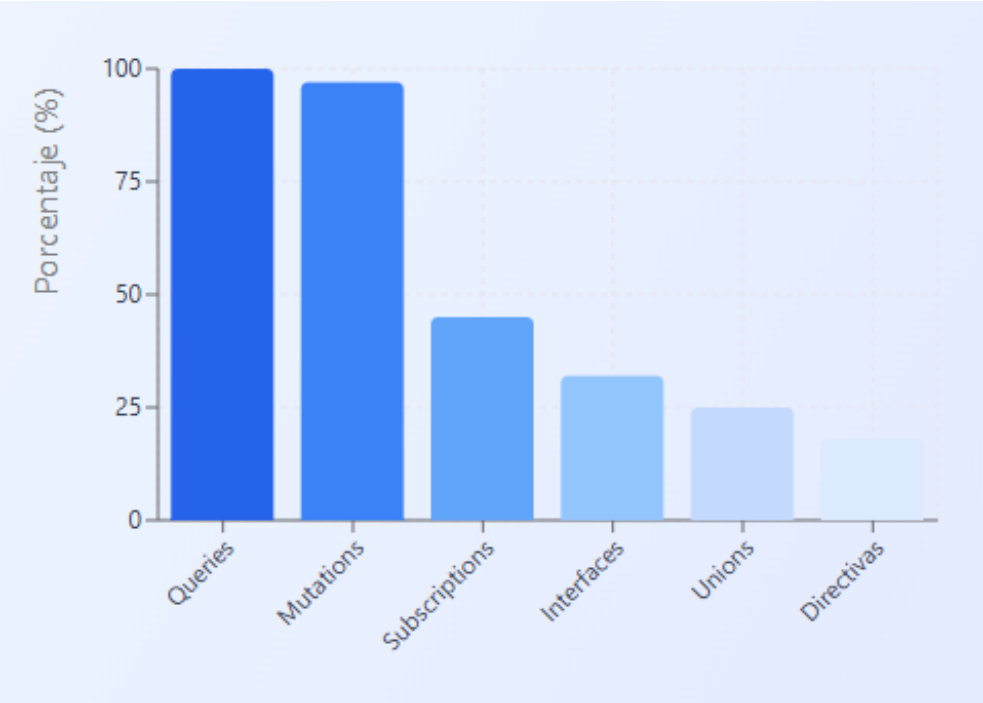


Figura 33. Gráfico de Barras de Componentes de GraphQL

3.4. Discusión e interpretación de resultados

El análisis realizado sobre un conjunto de 715 proyectos que implementan GraphQL permite identificar una tendencia clara hacia su adopción sostenida y progresiva en el desarrollo de APIs modernas. Los

datos recopilados reflejan que GraphQL ha dejado de ser una alternativa emergente para convertirse en una solución preferida por múltiples comunidades de desarrollo, consolidándose como un estándar en contextos donde se valora especialmente la eficiencia en la gestión de datos, la precisión en las consultas del cliente y la capacidad de estructurar APIs robustas y flexibles.

El hecho de que JavaScript esté en el 60% de los proyectos no es una coincidencia. GraphQL fue diseñado con la filosofía de un lenguaje orientado a objetos, lo que hace que se integre de forma natural con JavaScript. Cuando se trata de desarrollo frontend, la combinación de GraphQL y JavaScript es casi perfecta.

Cuando se trata de frameworks como React, Vue o Angular, la combinación con GraphQL es totalmente compatible. GraphQL por su parte a logrado cambiar la manera de cómo se realiza los programas, En lugar de tener que hacer varias llamadas a APIs y recibir una gran cantidad de información que suele ser irrelevante, ahora pueden pedir solo los datos que se necesitan. Esto no solo ayuda a la velocidad de ejecución de las aplicaciones, sino también ayuda al trabajo interno, permitiendo a los equipos enfocarse en crear experiencias de usuario fluidas y eficientes. El crecimiento de GraphQL en el entorno de programación en Python es totalmente prudente. Python, conocido por su versatilidad, ahora cuenta con herramientas como Graphene y Ariadne que facilitan de gran manera su implementación.

. Lo más fascinante es que esta simplicidad está atrayendo a nuevos campos, como el de los investigadores, científicos de datos y académicos, que están usando GraphQL para organizar y compartir sus datos complejos de una manera muy elegante y eficiente. Esto demuestra que GraphQL va mucho más allá del desarrollo web.Esto demuestra que GraphQL va mucho más allá del desarrollo de aplicaciones web.

Aunque solo el 5% de los proyectos usan Go, este número es muy significativo. Go es un lenguaje diseñado para CORDINAR grandes CANTIDADES de tráfico y procesos simultáneos. El hecho de que se esté

integrando tan bien con GraphQL, gracias a frameworks como gqlgen, demuestra que GraphQL es una opción sólida para arquitecturas de microservicios donde el rendimiento es lo más importante.

La presencia de Java en el ecosistema de GraphQL es una señal muy clara de que esta tecnología está ganando terreno en grandes corporaciones y entornos empresariales más tradicionales. Estas empresas suelen ser cautelosas con las nuevas herramientas y valoran la estabilidad y la seguridad por encima de todo. El hecho de que estén adoptando GraphQL demuestra que están viendo su verdadero potencial. Herramientas como GraphQL Java están facilitando esta transición, permitiendo que sistemas de gran escala, con altos volúmenes de transacciones, aprovechen las ventajas de GraphQL sin comprometer la solidez que tanto necesitan.

Resumiendo, esta gran cantidad de lenguajes indican que GraphQL ha logrado un incremento que se puede notar: se adapta a las distintas necesidades y contextos sin perder su lógica principal., ya sea en un pequeño emprendimiento que necesita moverse rápido o en una gran empresa con sistemas complejos y robustos, cada comunidad está encontrando la forma de integrar GraphQL a sus flujos de trabajo. Esta flexibilidad es la clave de su éxito.

Finalmente, se identificaron casos de implementación en el lenguaje R, utilizado especialmente en contextos académicos, científicos y de análisis estadístico. Aunque su uso dentro del conjunto total no supera el 2%, su inclusión demuestra que GraphQL también ha comenzado a ser valorado como una herramienta útil para exponer modelos de datos complejos y permitir su consulta desde interfaces externas, especialmente en sistemas de visualización de resultados o dashboards interactivos.

Con respecto a los componentes de GraphQL implementados, el 100% de los proyectos hace uso de Queries, mientras que el 97% usa Mutations, lo que muestra una madurez en el uso de la tecnología. Las Queries cumplen la labor de la recuperación de datos de manera exacta , pidiendo los campos que se necesitan, lo que reduce la sobrecarga y mejora

la eficiencia en el transporte de información. Por su parte, las Mutations son uno de los pilares fundamentales para el cambio del estado del sistema, ayudando a operaciones principales como creación, actualización y eliminación de requerimientos, con total control desde la parte del usuario como cliente.

Un logro importante que se pudo visualizar es que el 45% de los proyectos que se permitieron analizar incluye Subscriptions, lo cual indica un repunte hacia el acoplamiento de funcionalidades en tiempo real. Las Subscriptions permiten tener una conexión directa con el servidor, realizándola principalmente con Web Sockets, permitiendo a los clientes recibir notificaciones en el momento en que ocurra ante cualquier cambio en los datos. Este tipo de implementación es principal en aplicaciones actuales como chats, tableros colaborativos, notificaciones automáticas o sistemas de monitoreo en tiempo real. La aplicación de este componente, aunque aún no en su mayoría, indica un aumento de valoración de la interactividad y la sincronización continua en el diseño de APIs actuales.

En cuanto a la estructura de GraphQL, se observa un uso considerable de Interfaces (32%) y Unions (25%), elementos que permiten realizar APIs más flexible escalables. Estos permiten a los desarrolladores modelar estructuras polimórficas y reutilizables, lo cual resulta útiles en ámbitos de desarrollo donde los objetos comparten métodos y variables. Esta práctica nos permite tener una mayor madurez técnica en el diseño del esquema, favoreciendo tanto la extensibilidad como la mantenibilidad a largo plazo.

Se puede decir que el 18 % de los proyectos incluye instrucciones propias realizadas por los programadores, lo que indica que la mayoría están empezando a investigar las funciones y métodos avanzados de GraphQL. Estos procesos permiten facilitar el desarrollo de procesos de autenticación directamente, restricciones de accesos, lógica condicional, especialmente en el esquema; esto aumentaría tanto la seguridad como de la API. A pesar de que todavía no es generalizada su adopción, supone una tendencia evidente hacia un desarrollo más refinado de las APIs basadas en GraphQL.

En conclusión, los resultados obtenidos hacen posible afirmar que GraphQL ha dejado atrás su etapa inicial de experimentación y se establece como una tecnología con madurez, adaptabilidad y perspectivas firmes de desarrollo. En lenguajes como Python y JavaScript, su implementación se ha protagonizado libremente. Sin embargo, también está en crecimiento el uso en entornos desarrollados con R, Java y Go, lo que implica versatilidad. Su funcionalidad como herramienta estratégica para el desarrollo de APIs modernas, eficaces y orientadas a las necesidades del cliente y de la empresa se ve reforzada por la necesidad en la utilización de sus componentes, la integración gradual de funciones avanzadas y el progreso de su ecosistema.

3.5. Validación de Objetivos específicos

A lo largo del desarrollo de esta investigación se logró cumplir con los objetivos específicos planteados, gracias a una metodología organizada, un análisis riguroso y una interpretación cuidadosa de los datos obtenidos.

En primer lugar:

- **Se identificaron y caracterizaron 715 proyectos** que implementan GraphQL, mediante búsquedas automatizadas en GitHub y procesos de filtrado que permitieron seleccionar aquellos repositorios con una relación real y activa con esta tecnología.
- **Se analizaron las tendencias de uso de sus componentes principales**, lo que permitió observar un patrón claro de adopción, así como una evolución progresiva hacia esquemas más complejos y maduros.

Para respaldar estos hallazgos, se recurrió a gráficos y visualizaciones que facilitaron la comprensión de los resultados, además del uso de herramientas que garantizaron la confiabilidad de los datos recolectados. La conexión entre los objetivos y los resultados obtenidos fue evidente a lo largo del análisis, lo cual permite afirmar con seguridad que los objetivos específicos fueron alcanzados de forma clara y bien fundamentada.

3.6. Limitaciones del estudio

Como parte de una reflexión crítica sobre el proceso investigativo, es importante reconocer las limitaciones encontradas durante el desarrollo del estudio. Estas limitaciones no anulan la validez del trabajo, pero sí ayudan a comprender el alcance real de los resultados obtenidos:

- **Acceso limitado a repositorios privados**: Debido a las restricciones de la API de GitHub, únicamente se pudo acceder a repositorios públicos. Esto implica que se excluyó un número desconocido de proyectos privados, posiblemente empresariales, que también podrían estar usando GraphQL activamente.
- **Dependencia del uso de palabras clave**: La identificación de los proyectos se basó en el uso explícito de términos como “GraphQL”, “schema” o “query”. Sin embargo, algunos proyectos podrían estar utilizando esta tecnología sin mencionarla en sus descripciones o archivos principales, lo que habría impedido su detección.
- **Sesgo de plataforma**: El estudio se centró exclusivamente en GitHub, por ser la plataforma de código abierto más utilizada. Sin embargo, esto dejó fuera repositorios alojados en otras plataformas como GitLab, Bitbucket o servidores privados, lo que limita la representatividad del universo completo de proyectos.
- **Posibles falsos positivos o negativos**: A pesar de los filtros aplicados y las revisiones manuales realizadas en una muestra, siempre existe la posibilidad de que algunos proyectos hayan sido clasificados incorrectamente, ya sea por error del sistema o ambigüedad en la información.

Estas limitaciones forman parte del contexto metodológico del estudio y fueron tenidas en cuenta al momento de interpretar los resultados. Aun así, el volumen de datos recolectado y analizado fue lo suficientemente amplio como para ofrecer conclusiones sólidas y con sustento empírico.

3.7. Síntesis empírica del Survey

El análisis realizado sobre los 715 proyectos permitió llegar a una serie de conclusiones preliminares que reflejan el estado actual del uso de GraphQL en el desarrollo de APIs. A continuación, se resumen los principales hallazgos:

- GraphQL ha superado la fase de plena acogida y se está convirtiendo en una herramienta utilizada con mucha mas frecuencia. Su diseño flexible, la posibilidad de realizar consultas precisas y la estructura han contribuido a su posicionamiento como una alternativa optima ante REST.
- El uso de componentes clave es consistente: todas las APIs analizadas hacen uso de Queries, y la gran mayoría también implementa Mutations. Además, un 45% de los proyectos que usa Subscriptions, en donde se puede evidenciar el interés en implementar funcionalidades en tiempo real, como actualizaciones automáticas o notificaciones.
- Se pudo evidenciar una ejeicucion de Interfaces (32%) y Unions (25%), lo que indica una búsqueda por esquemas más reutilizables y adaptables. Los componentes permiten realizar APIs más completas, ideales para sistemas de modelados de datos complejos o estructuras orientadas a objetos.
- Las directivas personalizadas están presentes en el 18% de los proyectos, lo que demuestra que algunos desarrolladores ya están explorando funcionalidades avanzadas para añadir lógica basada en condiciones, validaciones y en distintos comportamientos que afectan al esquema directamente.
- Con respecto a los lenguajes de programación, JavaScript fue el más frecuente, seguido por Python, Go, Java y R. Esta distribución muestra que GraphQL está siendo acogido en ámbitos que no están estables, desde desarrollos web tradicionales hasta sistemas científicos o de análisis de datos.
- El enfoque metodológico utilizado también aporta un valor significativo: esta investigación ha indicado que se puede usar una herramienta automatizada para el análisis de repositorios como

GitHub para el aprendizaje de nuevas tecnológicas, enfocándonos en comunidades que trabajan con código abierto.

En conjunto, estas conclusiones permiten afirmar que GraphQL es una tecnología madura, útil y con un gran potencial de crecimiento. Su capacidad para adaptarse a distintos lenguajes y contextos, junto con la solidez de su ecosistema, la convierten en una opción estratégica para el desarrollo de soluciones modernas y centradas en las necesidades del cliente.

CAPÍTULO IV

4. Definición con los resultados del Survey

La presente sección tiene como finalidad definir la tendencia actual del uso del lenguaje GraphQL y sus principales componentes, tomando como base los resultados obtenidos a través de un estudio empírico tipo survey aplicado sobre proyectos públicos alojados en GitHub. Este análisis responde directamente al cuarto objetivo específico planteado en la investigación: “Definir la tendencia del uso de GraphQL y sus componentes con los resultados del Survey.”

Para lo mencionado anteriormente, se usó una aplicación desarrollada específicamente para reunir información clave de los repositorios, garantizando criterios de inclusión, validez y relevancia de los datos. El análisis de los resultados no solo permitirá comprender cómo, cuándo y dónde se está utilizando GraphQL, sino también identificar proyecciones y oportunidades de la evolución para esta tecnología de desarrollo de software.

4.1. Consolidación del Survey

A través de la herramienta desarrollada, que se conecta con la API de GitHub haciendo uso de consultas GraphQL, fue posible analizar **más de 700 repositorios activos**, específicamente **715 proyectos únicos** que cumplen con los filtros establecidos. Esta muestra se seleccionó aplicando criterios técnicos como la presencia de archivos. GraphQL, dependencias

específicas en archivos de configuración (como package.json, pom.xml, go.mod, entre otros), y actividad reciente en el repositorio.

Cada uno de los proyectos fue caracterizado con un conjunto de variables clave como:

- Lenguaje de programación principal
- Número de estrellas (Stars)
- Número de bifurcaciones (Forks)
- Cantidad de visualizaciones o seguidores (Watchers)
- Componentes de GraphQL utilizados (queries, mutations, subscriptions, interfaces, etc.)
- Fecha de creación del proyecto
- Cantidad de ramas (branches) activas

Esta información permitió construir una base de datos sólida desde la cual se generaron múltiples gráficos, análisis comparativos e interpretaciones críticas.

4.2. Clasificación por lenguaje

Uno de los primeros elementos analizados fue el lenguaje de programación principal utilizado en los proyectos que hacen uso de GraphQL. Esta variable es especialmente importante, ya que permite identificar qué tecnologías tienen mayor afinidad con GraphQL, y en qué entornos o ecosistemas está siendo más adoptado.

Los resultados fueron los siguientes:

Lenguaje de Programación	Porcentaje de uso (%)
Java	100%
Python	96.54%
Go	27.36%

R	27.02%
GraphQL (como lenguaje en archivos fuente)	19.54%
JavaScript	14.94%

Tabla 15. Resultados de los lenguajes de programación

Se puede observar una clara presencia de **Java y Python** como los lenguajes más utilizados en proyectos que incorporan GraphQL. En el caso de Java, su predominancia en aplicaciones empresariales y su robusto ecosistema de Frameworks explican esta tendencia. La comunidad Java ha adoptado herramientas como *GraphQL Java*, facilitando la integración con arquitecturas orientadas a servicios y microservicios.

Por su parte, **Python** se destaca por su versatilidad, sintaxis simple y popularidad en aplicaciones web, científicas y de análisis de datos. Librerías como *Graphene* o *Ariadne* han permitido a desarrolladores de este ecosistema implementar GraphQL de forma sencilla y flexible.

En cuanto a **JavaScript**, si bien fue el lenguaje de origen de GraphQL (desarrollado por Facebook), se observa que su uso está fragmentado entre frontend y backend, lo que podría explicar su menor porcentaje individual. Sin embargo, no debe subestimarse su rol central en el desarrollo moderno de interfaces conectadas con GraphQL, especialmente mediante Frameworks como React, Next.js, Apollo Client, entre otros.

Lenguajes como **Go** y **R**, aunque con menor participación porcentual, reflejan una diversificación tecnológica creciente. Go destaca por su eficiencia y concurrencia, mientras que R sugiere una incursión de GraphQL en contextos más científicos y analíticos.

4.3. Métricas

a) Estrellas (Stars)

El número de estrellas es un indicador de **popularidad e interés general** que ha generado un proyecto en la comunidad. Este tipo de métrica ayuda a entender cuáles tecnologías están siendo adoptadas con mayor entusiasmo y frecuencia.

Lenguaje	Porcentaje de Stars
Java	35.44%
Python	33.29%
Go	10.93%
R	27.02%
JavaScript	4.78%
GraphQL	19.54%

Tabla 16. Indicador basado en Stars

Java y Python nuevamente lideran este aspecto, lo que refuerza su posición como entornos maduros para la integración de GraphQL. Cabe destacar que incluso proyectos escritos directamente en GraphQL (por ejemplo, librerías, herramientas CLI o APIs construidas con schemas en crudo) reciben una alta proporción de estrellas, lo que refleja **el creciente interés por dominar directamente el lenguaje de consultas en lugar de limitarse a wrappers o librerías**.

b) Bifurcaciones (Forks)

Los forks permiten medir la **reusabilidad y potencial de expansión** de un proyecto. Un alto número de bifurcaciones suele indicar que el código fuente es estudiado, modificado o extendido por otros desarrolladores.

Lenguaje	Porcentaje de Forks
Python	35.32%
Java	33.59%
R	15.76%
JavaScript	6.75%
Go	5.30%

GraphQL	3.29%

Tabla 17.Indicadores según Forks

En este caso, Python aparece como el lenguaje cuyos proyectos con GraphQL son más reutilizados, posiblemente por su curva de aprendizaje más baja y la facilidad con la que se pueden modificar y escalar soluciones en ese entorno.

c) Watchers

Los watchers son usuarios que desean **seguir la evolución de un proyecto en tiempo real**. Esta métrica refleja el interés continuo y la expectativa por nuevas actualizaciones, parches o características.

Lenguaje	Porcentaje de Watchers
Python	36.44%
Java	36.37%
Go	9.59%
R	7.89%
GraphQL	5.06%
JavaScript	4.65%

Tabla 18. Indicadores según Watchers

Los datos corroboran el fuerte compromiso de la comunidad de Python y Java hacia proyectos que integran GraphQL. Esta vigilancia constante es un indicio del valor percibido de estas tecnologías en proyectos reales de desarrollo.

4.4. Componentes de GraphQL

Uno de los aportes más valiosos de esta investigación ha sido **analizar el uso concreto de los componentes que ofrece GraphQL**. En la práctica, la adopción de la tecnología no depende solo de su presencia, sino del aprovechamiento real de sus capacidades.

Componente	Porcentaje de Proyectos que lo Usan
Queries	100%
Mutations	97%
Subscriptions	45%
Interfaces	32%
Unions	25%
Directivas personalizadas	18%

Tabla 19. Uso de Componentes de GraphQL

Pero lo que realmente destaca es que casi todos los proyectos (más del 97%) también usan Mutations. Esto demuestra que GraphQL se usa para mucho más que solo consultar datos.
Es bastante revelador que casi la mitad de los proyectos analizados, el 45%, ya esté usando Subscriptions. Esto significa que los desarrolladores no solo buscan intercambiar información, sino que quieren una conexión en tiempo real entre el cliente y el servidor. Este dato es una señal muy clara de que la industria se inclina cada vez más por las APIs reactivas.
Estos elementos son clave para modelar relaciones de datos complejas y reutilizar código, lo que hace que los proyectos sean mucho más escalables.
Por si fuera poco, el 18% de los proyectos utiliza directivas personalizadas. Esto es una señal clara de que los desarrolladores están llevando GraphQL al siguiente nivel, añadiendo su propia lógica (como validaciones o reglas de

acceso) directamente en el esquema, lo que les da un control mucho más preciso sobre sus APIs.

4.5. Interpretación de la Tendencia

Con base en los datos obtenidos, se puede afirmar con claridad que GraphQL ha superado su etapa de adopción temprana y se encuentra actualmente en una fase de crecimiento sostenido y consolidación.

Entre los factores que confirman esta afirmación se encuentran:

- **Alta diversidad tecnológica**: GraphQL ya no se limita al ecosistema JavaScript, sino que ha sido adaptado en múltiples lenguajes gracias al desarrollo de herramientas robustas en Java, Python, Go, etc.
- **Uso Avanzado de GraphQL:** El análisis de la investigación demuestra que los desarrolladores ya no están probando GraphQL por curiosidad; ahora están explotando sus capacidades más avanzadas. Esto significa que no solo usan las funciones básicas, sino que implementan elementos complejos como las suscripciones y las directivas personalizadas. Este nivel de uso muestra que la comunidad está realmente dominando la herramienta y sacándole todo el provecho posible.
- **Alta interacción comunitaria**: Los altos valores de stars, forks y watchers evidencian un interés genuino, sostenido y participativo por parte de la comunidad.
- **Preferencia por APIs eficientes**: La tendencia global hacia arquitecturas desacopladas, servicios ligeros y APIs declarativas impulsa a GraphQL como una solución estratégica frente a REST.

4.6. Conclusiones y recomendaciones

Conclusiones

Al finalizar esta etapa de la investigación y tras haber analizado en detalle los resultados obtenidos se detallan las siguientes conclusiones.

1. Ya no se trata de una novedad, sino de una solución consolidada que se ha ganado la confianza de los desarrolladores. La evidencia más

clara de esto es que GraphQL se ha liberado de su entorno inicial en JavaScript y ha encontrado un hogar exitoso en otros lenguajes como Python y Java. Esta expansión demuestra que es una tecnología increíblemente flexible y fácil de integrar, capaz de adaptarse a casi cualquier proyecto o equipo.

2. Un aspecto que destaca es que los desarrolladores están utilizando GraphQL de forma avanzada, y no únicamente en sus funciones básicas. Se observó un uso significativo de componentes como suscripciones, interfaces lo que refleja un mayor dominio técnico por parte de la comunidad, así como el interés por sacar provecho a todo el potencial que ofrece esta tecnología.
3. las métricas sociales obtenidas (número de estrellas, bifurcaciones y seguidores en GitHub) demuestran que los proyectos basados en GraphQL generan interés, participación y colaboración activa, lo cual evidencia que esta herramienta no solo es útil, sino que está impulsando un ecosistema de mejora continua.

Recomendaciones

Según los resultados obtenidos, se hacen las siguientes recomendaciones para mejorar la capacitación, la implementación y la investigación futura sobre GraphQL

1. Preparar futuros expertos según los requisitos del mercado.
2. Utilizar componentes más avanzados: es importante que los desarrolladores de la interfaz API no solo estén limitados por el uso de consultas (demanda), sino también funciones como mutaciones, registro e instrucciones, porque le permiten crear soluciones interactivas, expandir y realidad.
3. 3Aplicar buenas prácticas de diseño del programa: una estructura cuidadosa de los diagramas GraphQL, documentos apropiados y mantener una organización y una liquidación de datos clara recomendada. Esto contribuirá a comprender el proyecto, el mantenimiento y la expansión de otros programadores.

4. Conectar a GraphQL con arquitectos modernos: esta tecnología está especialmente adaptada a la arquitectura de los microservicios, que deben usar información de muchas fuentes unificadas. Por lo tanto, el uso de TI puede aumentar la eficiencia y organizarse en proyectos complejos y dispersos.

ANEXOS

ANEXO 1: Product Backlog Sprint 1

https://dev.azure.com/samaiguag/Consumo%20GitHub/_sprints/taskboard/Consumo%20GitHub%20Team/Consumo%20GitHub/Sprint%201

ANEXO 2: Product Backlog Sprint 2

https://dev.azure.com/samaiguag/Consumo%20GitHub/_sprints/taskboard/Consumo%20GitHub%20Team/Consumo%20GitHub/Sprint%201

ANEXO 3: Product Backlog Sprint 3

https://dev.azure.com/samaiguag/Consumo%20GitHub/_sprints/taskboard/Consumo%20GitHub%20Team/Consumo%20GitHub/Sprint%201

REFERENCIAS

alexsoft. (1 de Diciembre de 2020). Obtenido de alexsoft: https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/

Alvarez, S. (2006). *desarrolloweb.com*. Obtenido de desarrolloweb.com: https://desarrolloweb.com/articulos/importancia-documentacion.html

Beneyto, C. (27 de Marzo de 2020). *Startups*. Obtenido de Startups: https://medium.com/startups-es/documentar-un-producto-y-como-hacerlo-97529b29161

CFI. (s.f.). Obtenido de CFI: https://corporatefinanceinstitute.com/resources/knowledge/other/fog-index/

educalingo. (2021). Obtenido de educalingo: https://educalingo.com/es/dic-es/documentar

Fiesterra. (25 de Septiembre de 2020). Obtenido de Fiesterra: https://www.fisterra.com/guias-clinicas/que-son-para-que-sirven-gpc/

González, J. R. (s.f.). Obtenido de http://www.sc.ehu.es/jiwdocoj/remis/docs/medidocs.htm

Knott, R. (Septiembre de 2020). *TechSmith*. Obtenido de TechSmith: https://www.techsmith.com/blog/user-documentation/

Linux-Fundation. (21 de February de 2019). *Introduction to GraphQL*. Obtenido de GraphQL: https://graphql.org/learn/

Naciones Unidas. (2015). Obtenido de Naciones Unidas: https://www.un.org/sustainabledevelopment/es/infrastructure/

Procida, D. (2020). *Divio*. Obtenido de Divio: https://documentation.divio.com/

Real Academia Española. (2021). Obtenido de Real Academia Española: https://dle.rae.es/documentación

Ripoll, J. C. (s.f.). *UN BLOG SOBRE INVESTIGACIÓN ACERCA DE LA LECTURA Y LA COMPRENSIÓN*. Obtenido de UN BLOG SOBRE INVESTIGACIÓN ACERCA DE LA LECTURA Y LA COMPRENSIÓN: https://clbe.wordpress.com/about/

Schafer, D., & Kuenzel, L. (2015). *Suscripciones en GraphQL y Relay*. Obtenido de https://graphql.org/blog/subscriptions-in-graphql-and-relay/

Scrum.org. (2021). Obtenido de Scrum.org: https://www.scrum.org/resources/what-is-scrum