Ingeniería Web

Informe sobre Tecnologías Web

Dorian Boleslaw Wozniak (817570@unizar.es)

Viernes, 20 de octubre de 2023

0. Índice

1. Resumen ejecutivo

Se describen una serie de protocolos, librerías, y frameworks para el desarollo de aplicaciones web, destacando sus características y poniendo ejemplos de escenarios donde se podría recomendar su uso o no.

- **gRPC** permite implementar RPC de alto rendimiento, asíncronas y bidireccinales, intercambiando datos en un formato binario mediante Protocol Buffers. Aunque pesado y a veces difícil de escalar, es extremadamente competente para gestionar grandes cargas de tráfico.
- **Express.js** esta extendido en el mundo de Node.js especialmente para la implementaciónde API REST. Sirviendo de base para otros *frameworks*, es muy flexible y aprovecha elementos de asincronía, pero también delega otras cuestiones del desarrollo y no ofrece buen soporte para WebSockets.
- **Ruby on Rails** es una navaja suiza para el desarrollo rápido de aplicaciones web, con una filosofía dirigida y centrada en la sencillez. A pesar de ofrecer bastante funcionalidad para la implementación de comunicación mediante REST y WebSockets, también peca de cierta lentitud y puede on escalar para proyectos mas grandes.
- **GraphQL** es una potente alternativa arquitectónica a REST, permitiendo implementar API con un lenguaje de consultas mas flexible. Aunque complejo, facilita el desarrollo en entornos con múltiples tipos de bases de datos, gran cantidad de tráfico y frecuentes modificaciones al modelo, aprovechando un sistema de tipos y errores, así como procedimientos para juntar y transformar diferentes definiciones de datos.
- **Flask** es un *micro-framework* para Python que se centra en la simpleza, tratando principalmente la gestión de peticiones HTTP. Aunque tiene algunos extras como un módulo de pruebas, depende mucho de otras librerías para algunas de las necesidades de un proyecto mas serio. Permite implementar fácilmente API REST y cuenta con soporte para WSGI.

2. gRCP

gRPC es un *framework* de código abierto desarrollado por Google para crear API en torno a RPC de alto rendimiento. Fue lanzado en 2016 como reemplazo a Stubby, también de Google [1].

gRPC centra su comunicación en **Protocol Buffers** [2], un lenguaje de definición de interfaces que permite serializar fácilmente datos para su transmisión. Aun así, gRPC también permite el envió de datos con otros formatos de representación como JSON o XML.

gRPC es actualmente utilizado por companías como Netflix o el procesador de pagos Square, herramientas como Docker, bases de datos modernos como CockroachDB, ... [3].

Sus principales características son su uso por defecto de HTTP/2 y TLS, su disponibilidad en múltiples lenguajes de programación [3], su eficiencia en cuanto a latencia gracias al proceso de compilación y serialización de los datos a enviar, y capacidad de escalar [3], así como la capacidad para comunicación continua bidireccional y poder realizar llamadas síncronas o asíncronas [4].

Esto lo hace ideal en escenarios donde se espera una gran carga de tráfico, especialmente en microservicios [5].

Algunos puntos negativos son el acoplamiento necesario entre cliente y servidor al ser necesario conocer a priori la definición de un *Protocol Buffer* [6]. Esto también convierte a gRPC en un sistema que necesita muchos mas recursos computacionales al tener que compilar binarios para su envío. El hecho de utilizar HTTP/2 como su base provoca que no sea muy compatible con algunos navegadores, relegándolo a la comunicación con *backends* [7].

gRPC no es ideal en escenarios donde se opera sobre recursos computacionales limitados [7]. También se recomienda evitar en casos donde la API a crear es bastante simple (p.e. para servir a una página web) [7].

3. Express.js

Express.js es un *framework* para el desarrollo de *backends* de aplicaciones web para Node.js. Lanzado en 2010 y apadrinado por StrongLoop, propiedad de IBM, se ha vuelto en un *middleware* muy utilizado en el entorno de Node, convirtiendose en una especie de estandar *de facto* , siendo utilizado como base por multitud de *frameworks* para el desarrollo *full-stack* de aplicaciones web [8].

Un aspecto a destacar es que Express se declara como un "*framework no opinionado y minimalista*". Es decir, que la integración de Express en un proyecto queda a cargo del programador, en vez de adaptarse al funcionamiento del framework. Esto permite que Express sea una herramienta muy simple y flexible, pero simultáneamente con cierta curva de aprendizaje y la necesidad de utilizar otras soluciones para cuestiones como la autenticación, seguridad o el *testing*. Por suerte, su amplia adopción supone también una gran cantidad de recursos y documentación [9].

En cuanto a lo técnico, el *framework* es muy popular para la creación de API REST, con un sistema de encaminamiento robusto y un sistema de gestión de solicitudes HTTP que permiten tratarlas de forma asíncrona [10]. Pero Express no se centra cuestiones de comunicación bidireccional. A pesar de tener cierto soporte de comunicación bidireccional mediante WebSockets [11], es recomendable el uso de otros *frameworks* como Socket.io [9].

Express destaca en su utilidad para proyectos de tamaño mediano, como páginas de una sola página [10] y con *renderizado* desde el servidor [9], donde su capacidad de tratar peticiones HTTP de forma asíncrona permite un escalado fácil de la aplicación.

Sin embargo, hay mejores alternativas en casos de que la comunicación en tiempo real sea el núcleo de la aplicación o para proyectos muy grandes y con gran tráfico [9].

4. Ruby on Rails

Nacido en 2005, **Ruby on Rails** es un *framework* de desarrollo de aplicaciones web para Ruby con una alta popularidad. Rails, a diferencia de Express, se proclama como un "*software* opinionado", teniendo una visión mas estricta de cómo desarrollar un proyecto, siguiendo las filosofías de "No Repetirse", donde cada elemento del código tiene un propósito muy concreto y no ambiguo; y de "Convención sobre configuración", con una configuración por defecto que asume ciertas decisiones para evitar tener que configurar todo un proyecto desde cero [12].

Gracias a esto, Rails destaca por la facilidad y rapidez en la que se puede implementar nuevas funcionalidades al abstraer muchas decisiones de diseño [13]. Esto, por el otro lado, significa que hay ciertos sacrificios en cuanto a flexibilidad: uno se tiene que adaptar a Rails, y no viceversa [14].

Otros contras son su rendimiento, siendo Rails un *framework* pesado sobre un lenguaje interpretado [14]. También destaca su tendencia a que los errores son costosos, tanto en el tiempo para solucionarlos como en el rendimiento. Por suerte, al tener una gran adopción en la industria tiene una gran cantidad de recursos y soporte [13].

En cuanto a lo técnico, Rails es buena opción para implementar API REST utilizando routers [15], o comunicación bidireccional mediante ActionCables [16].

Si se necesita una aplicación web de tamaño mediano en un escenario de desarrollo rápido, donde se pueden implementar funcionalidades complejas evitando cuestiones del lenguaje [17].

Por otro lado, los problemas de rendimiento que puede presentar son capaces de perjudicar a la larga un proyecto.

5. GraphQL

GraphQL es tanto un lenguaje de consultas para API como un entorno de ejecución para servidores que habilita realizar estas consultas. Fue desarrollado por Facebook y lanzado en 2015.

GraphQL se postula como una alternativa a arquitecturas como REST, asegurando que es mas rápida que REST y también que soluciona algunos de los problemas existentes en REST.

Algunas cuestiones positivas es la capacidad de crear esquemas y poder realizar transacciones de datos de forma tipada, con tipos de datos básicos como booleanos, cadenas, enteros, reales..., por lo que se puede conocer fácilmente como son los datos a enviar o recibir [19].

Otros problemas que soluciona son los casos de *overfetching*: usando REST, es mas difícil restringir solicitudes de consulta, mientras que GraphQL lo implementa de forma nativa. Por ello, puede reducir la cantidad de datos a enviar a la estrictamente necesaria [19].

Otras características son la facilidad de gestionar errores, mayor facilidad a la hora de implementar modificaciones a la API, evitando múltiples versiones, etc... [20]

Algunas cuestiones negativas son su nula utilización de códigos HTTP para indicar errores: incluso si la consulta falla, el servidor devolverá 200 [21]. Además, GraphQL es notoriamente compleja, permitiendo combinar esquemas de microservicios, transformaciones de tipos de datos, [19] y diferentes arquitecturas de integración para añadirlo a un proyecto existente con una o varias bases de datos .

En caso de tener un proyecto grande, con un gran intercambio de datos distribuido en múltiples servicios que pueden cambiar frecuentemente con las necesidades de la aplicación es cuando destaca, al aprovechar al abstracción que presentan las consultas y la optimización de obtener solo los datos que son necesarios [19].

Sin embargo, GraphQL puede ser demasiado compleja para proyectos muy sencillos, al añadir una capa de abstracción extra que puede ser innecesaria [21].

6. Flask

Lanzado en 2010, **Flask** es un *micro framework* para Python. Fue creado por Armin Ronacher, que inició su desarrollo en 2004 incialemente como una broma de *April Fools*.

En comparación a otros *frameworks* mas completos como Django, Flask se centra en ofrecer una alternativa liviana, flexible y extensible para crear aplicaciones web [22]. Flask está basado en los principios de **WSGI**, una especificación que describe como un servidor web debe comunicarse con aplicaciones cliente usando Python, de forma algo similar a CGI [23].

Una de las ventajas, ademas de las ya mencionadas, es la facilidad en la que se puede crear un proyecto, con un servidor básico ocupando solo 5 líneas [24], siguiendo la línea de Python en cuanto a sencillez. También es muy escalable.

Aunque trae algunas herramientas adicionales como un módulo de pruebas y su propio motor de plantillas HTML, su simpleza implica que muchas de las necesidades adicionales de un servidor web requieren librerías de terceros 25.

Flask ademas se centra en la creación de API REST, y puede aprovechar las capacidades de Python para implementar asincronía 26. Sin embargo, no ofrece soporte nativo para WebSockets, aunque existen librerías como Flask-SocketIO que se integran facilmente, destacando su dependencia de *software* de terceros 27.

En cuanto a sus desventajas serias, su uso no está tan extendido como Django, por lo que no tiene tantos recursos adicionales a su documentación 28.

Para proyectos simples o en los cuales se necesitan librerías alternativas a las de otros *frameworks* como Django, Flask puede ser una buena opción. Tambien puede ser una buena opción para prototipado. Pero su reducida adopción y la necesidad de integrar dichas librerías puede volverlo desaconsejaba para servicios de gran escala.

7. Bibliografía

[1] “Overview,” Protocol Buffers Documentation. https://protobuf.dev/overview/ V. Talwar, “gRPC: a true internet-scale RPC framework is now 1.0 and ready for production deployments,” Google Cloud Blog, Aug. 23, 2016. https://cloud.google.com/blog/products/gcp/grpc-a-true-internet-scale-rpc-framework-is-now-1-and-ready-for-production-deployments

[2] “Introduction to GRPC,” gRPC, Feb. 15, 2023. https://grpc.io/docs/what-is-grpc/introduction/

[3] “FAQ,” gRPC, Sep. 13, 2023. https://grpc.io/docs/what-is-grpc/faq/

[4] “Core concepts, architecture and lifecycle,” gRPC, Dec. 22, 2022. https://grpc.io/docs/what-is-grpc/core-concepts/

[5] Editor, “What is gRPC: Main Concepts, Pros and Cons, Use Cases,” AltexSoft, Mar. 25, 2021. https://www.altexsoft.com/blog/what-is-grpc/

[6] “gRPC vs REST - Difference Between Application Designs - AWS,” Amazon Web Services, Inc. https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/

[7] A. Joseph, “When to avoid using GRPC: Considerations for Scalable Microservices Architecture,” www.linkedin.com, [Online]. Available: https://www.linkedin.com/pulse/when-avoid-using-grpc-considerations-scalable-albin-joseph

[8] “Frameworks built on Express.” https://expressjs.com/en/resources/frameworks.html

[9] A. Taha, “Express.js: The good, the bad, and the ugly,” www.linkedin.com, [Online]. Available: https://www.linkedin.com/pulse/expressjs-good-bad-ugly-aziz-taha

[10] S. Eseme, “What is Express.js? Everything you should know,” Kinsta®, Sep. 2023, [Online]. Available: https://kinsta.com/knowledgebase/what-is-express-js/

[11] AlishaS, “What is Express in Node.js - Geek Culture - Medium,” Medium, Jun. 23, 2023. [Online]. Available: https://medium.com/geekculture/what-is-express-in-node-js-f51d178056b9

[12] “Getting Started with Rails — Ruby on Rails Guides,” Ruby on Rails Guides. https://guides.rubyonrails.org/getting_started.html

[13] V. Rak, “Pros & Cons of Ruby on Rails you should know before choosing the technology for your startup,” Sloboda Studio, May 2023, [Online]. Available: https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/

[14] Full Scale, “Ruby on Rails: Pros and Cons,” Full Scale, Apr. 04, 2023. https://fullscale.io/blog/ruby-on-rails-pros-and-cons/

[15] “Using rails for API-only applications — Ruby on Rails guides,” Ruby on Rails Guides. https://guides.rubyonrails.org/api_app.html

[16] “Action Cable Overview — Ruby on Rails Guides,” Ruby on Rails Guides. https://guides.rubyonrails.org/action_cable_overview.html

[17] S. Miller, “What is Ruby on Rails?,” Codecademy Blog, Sep. 15, 2021. https://www.codecademy.com/resources/blog/what-is-ruby-on-rails/

[18] S. Scholl, “What is GraphQL? | 8base.” https://www.8base.com/graphql

[19] H. Baldaniya, “Why and When to Use GraphQL,” dzone.com, Dec. 2022, [Online]. Available: https://dzone.com/articles/why-and-when-to-use-graphql-1

[20] “Why use GraphQL?,” Apollo GraphQL Blog. https://www.apollographql.com/blog/graphql/basics/why-use-graphql/

[21] “S. E. Team, Advantages and Disadvantages of GraphQL | Stable Kernel,” Stable Kernel, Oct. 2023, [Online]. Available: https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/

[22] Great Learning, “Flask vs Django: Which Python Framework to choose? - Great learning,” Great Learning Blog: Free Resources What Matters to Shape Your Career!, Sep. 06, 2022. https://www.mygreatlearning.com/blog/flask-vs-django/

[23] “PEP 3333 – Python Web Server Gateway Interface V1.0.1 | peps.python.org.” https://peps.python.org/pep-3333/

[24] “Quickstart — Flask Documentation (3.0.X).” https://flask.palletsprojects.com/en/latest/quickstart/

[25] Detimo, “Python Flask: pros and cons,” DEV Community, Nov. 2019, [Online]. Available: https://dev.to/detimo/python-flask-pros-and-cons-1mlo

[26] “Using async and await — Flask Documentation (3.0.x).” https://flask.palletsprojects.com/en/3.0.x/async-await/

[27] “Flask-SocketIO — Flask-SocketIO documentation.” https://flask-socketio.readthedocs.io/en/latest/

[28] A. Raju, “Disadvantages of Flask - Allwin Raju - Medium,” Medium, Jan. 06, 2022. [Online]. Available: https://allwin-raju-12.medium.com/disadvantages-of-flask-95f1ff8f83bf