

VERGELIJKING VAN API ARCHITECTUREN VOOR DE API SERVER VAN HET PROJECT ROBOTICA

Wessel Tip <contact@wessel.gg> (Student nummer 696770, <https://wessel.gg/>)

Technische Informatica — Project Robotica

Hoogeschool Inholland Alkmaar

Jaar 2, Semester 2 (Jan. 2024 - Jun. 2024)

Samenvatting—In dit verslag zal besproken worden waarom er is gekozen om een RESTful API te maken voor het project Robotica.

De toegankelijkheid en schaalbaarheid van de API-architecturen REST, SOAP, GraphQL en gRPC zullen worden vergeleken om uiteindelijk op de conclusie te komen waarom REST gebruikt wordt.

In het Robotica-project, waarbij voor mijn persoonlijke bijdrage de communicatie tussen de gebruiker en de robot centraal staat, is de keuze van een geschikte API-architectuur van groot belang.

De API moet eenvoudig te implementeren en te onderhouden zijn. Dit verslag zal vier verschillende API-architecturen bespreken en vergelijken om uiteindelijk tot de conclusie te komen voor welk architectuur het beste is voor het project.

INHOUDSOPGAVE

I	Inleiding	1
II	Probleemstelling	1
II-A	Toegankelijkheid van de diverse functies	1
II-B	Schaalbaarheid en Prestaties	1
II-C	Onderhoudbaarheid en Uitbreidbaarheid	1
III	theoretisch Kader	2
III-A	API-Architecturen	2
III-B	RESTful (Representational State Transfer)	2
III-B1	Voordelen van REST	2
III-B2	Nadelen van REST	2
III-C	SOAP (Simple Object Access Protocol)	2
III-C1	Voordelen van SOAP	2
III-C2	Nadelen van SOAP	2
III-D	GraphQL	2
III-D1	Voordelen van GraphQL	2
III-D2	Nadelen van GraphQL	2
III-E	gRPC (Google Remote Procedure Call)	2
III-E1	Voordelen van gRPC	2
III-E2	Nadelen van gRPC	2
IV	Conclusie	3
IV-A	Overwegingen	3
IV-B	Keuze voor REST	3
IV-C	Afweging tegen alternatieven	3
IV-D	UML model voor de API Server	3
	Referenties	3

I. INLEIDING

In de moderne softwareontwikkeling spelen API's (Application Programming Interfaces) een belangrijke rol. Ze vormen de brug tussen verschillende softwarecomponenten en zorgen voor een gestructureerde manier om gegevens uit te wisselen.[1]

II. PROBLEEMSTELLING

In het project is het belangrijk dat de gebruiker op afstand kan communiceren met de mapping robot. De robot moet de commando's van de gebruiker kunnen ontvangen en uitvoeren.

De gebruiker moet ook kunnen zien wat de robot allemaal gemapt heeft, om eventueel in te grijpen als dit niet klopt.

De robot vereist een structurele manier om deze gegevens uit te wisselen, en de optie om meerdere robots tegelijkertijd te besturen. De aandachtspunten hiervoor zijn:

A. Toegankelijkheid van de diverse functies

Het interface dat de gebruiker ziet moet alle verzamelde data laten zien. Hij zal de mapping en de geplande route moeten presenteren in een overzichtelijke manier. Ook moet de robot diverse commando's vanuit de gebruiker kunnen ontvangen om zo de gewenste taken uit te kunnen voeren.

B. Schaalbaarheid en Prestaties

Naarmate het project groeit, moeten de communicatie kanalen tussen de verschillende componenten schaalbaar blijven zonder dat dit ten koste gaat van de prestaties. Dit betekent dat de gekozen API-oplossing in staat moet zijn om een toenemend aantal verzoeken af te handelen zonder vertragingen of systeemuitval.

C. Onderhoudbaarheid en Uitbreidbaarheid

De gekozen API-architectuur moet eenvoudig te onderhouden zijn, met een heldere en overzichtelijke structuur. De architectuur moet ook flexibel genoeg zijn om toevoegingen en aanpassingen te ondersteunen zonder dat het codebase herschreven moet worden.

III. THEORETISCH KADER

A. API-Architecturen

API's (Application Programming Interfaces) zijn onmisbare componenten in moderne software. Deze API's maken de communicatie tussen verschillende softwarecomponenten mogelijk.[2]

Er zijn over de jaren vele verschillende manieren bedacht voor het ontwerpen en implementeren van API's, elk met zijn eigen voordelen en nadelen. [3]

Dit hoofdstuk zal vier van de meest gebruikte API architecturen bespreken.

B. RESTful (Representational State Transfer)

REST is een architectuur dat wordt gebruikt voor het communiceren van gegevens om netwerkanvragen te ontwerpen.

Het maakt gebruik van standaard HTTP-methoden (zoals GET, POST, PUT, DELETE) om gegevens te communiceren tussen clients en servers.[1]

Door de simpliciteit en het al gebruik maken van de bestaande HTTP protocollen is REST een populaire keuze voor API's. Dit maakt het ook erg geschikt voor web gebaseerde applicaties.

1) Voordelen van REST:

- 1) **Eenvoudige structuur** — gebaseerd op standaard HTTP-methoden en JSON
- 2) **Makkelijke implementatie** — meeste talen ondersteunen de basis van webrequests en JSON serialisatie al
- 3) **Schaalbaarheid** — RESTful API's zijn stateless en kunnen eenvoudig horizontaal geschaald worden, echter zal de prestatie wel minder zijn dan andere methodes zoals gRPC of GraphQL.[3]

2) **Nadelen van REST:** Echter heeft REST ook zijn nadelen, sommige hiervan zijn:

- 1) **Overbodig veel data** — Minder geschikt voor complexere query's en datastructuren (over-fetching en under-fetching)
- 2) **Documentatie** — Geen directe ondersteuning voor documentatie, schemas en typecontrole
- 3) **Real-time** — Doordat de connectie niet levend wordt gehouden nadat de aanvraag is afgehandeld is het minder geschikt voor real-time communicatie.

C. SOAP (Simple Object Access Protocol)

SOAP is een protocol voor uitwisseling van informatie netzoals REST vermeld in [deelparagraaf III-B](#). Echter is SOAP meer complex en minder populair dan REST.

SOAP maakt gebruik van XML voor het sturen van berichten en ondersteunt in tegenstelling tot REST verschillende transportprotocollen zoals HTTP en SMTP. SOAP biedt ook robuuste beveiligings- en transactiebeheerfuncties wat REST niet zo maar heeft.[3], [4]

1) Voordelen van SOAP:

- 1) **Beveiliging** — Sterke beveiligingsfuncties met behulp van WS-Security en ondersteuning voor ACID transacties.

2) **Nadelen van SOAP:** Net zoals REST heeft SOAP ook zijn nadelen:

- 1) **Ingewikkeld formaat** — Complexiteit en overhead door XML-berichten
- 2) **Prestatieproblemen** — Langzamere prestaties vergeleken met REST

D. GraphQL

GraphQL is een querytaal voor API's die is ontwikkeld door Facebook in 2012[5]. In tegenstelling tot REST en SOAP, waarbij de server bepaalt welke gegevens worden geretourneerd, stelt GraphQL clients in staat om een specifieke set data op te vragen. Ook biedt GraphQL een sterke typecontrole en introspectie aan door middel van schemas.[6]

1) Voordelen van GraphQL: Voordelen van GraphQL:

- 1) **Efficiëntie** — Door de query aard van GraphQL is het makkelijk om snel en flexibele bepaalde stukken data aan te vragen.
- 2) **Gegevensbesparing** — Omdat de gebruiker alleen maar de gegevens ontvangt die zij nodig hebben, vermindert dit over-fetching en under-fetching waardoor er meer bandbreedte en rekenkracht wordt bespaard.[6]
- 3) **Documentatie** — Sterke typecontrole en introspectie door het gebruik van schemas.
- 4) **compatibiliteit** — Ook al is het niet gewenst, GraphQL is compatibel met RESTful clients, hierdoor kan de implementatie vrij simpel zijn als de kracht van GraphQL niet nodig is.

2) Nadelen van GraphQL: Nadelen van GraphQL:

- 1) **Intergratie** — Door de aard van GraphQL is de serverimplementatie veel complexer, en daarom ook aangeraden om een library te gebruiken.
- 2) **Complexiteit** — Mogelijkheid van te complexe queries die de serverbelasting verhogen

E. gRPC (Google Remote Procedure Call)

gRPC is een modern RPC-framework dat in 2016 door Google is ontwikkeld. Het maakt gebruik van HTTP/2 voor transport, Protocol Buffers voor berichtserialisatie en biedt functies zoals load balancing en monitoring. Door zijn load balancing functies is gRPC zeer geschikt voor high-performance en real-time communicatie.[7], [3]

Echter is het wel de moeilijkste methoden om te implementeren en te onderhouden.

1) Voordelen van gRPC:

- 1) **Prestatie** — Hoge prestaties door het gebruik van protocol buffers, berichten zijn tot 30% kleiner dan JSON.
- 2) **Real-time** — In tegenstelling tot REST, GraphQL en SOAP is gRPC wel geschikt voor real-time communicatie

2) Nadelen van gRPC:

- 1) **Complexiteit** — Complexer dan alle andere methodes vermeld om op te zetten en te debuggen
- 2) **Limitaties** — Door het vele gebruik van HTTP/2.0 is het niet compatibel met oudere web browsers. Ook is er

veel minder bekend over gRPC dan de andere methodes door zijn relatief nieuwe staat.

IV. CONCLUSIE

Na een evaluatie van de verschillende API architectuur vermeld in [paragraaf III](#) en de eisen van het Robotica-project, is er tot de conclusie gekomen dat REST de meest geschikte oplossing is.

De uiteindelijke UML diagram voor de API server is te zien in [Figuur 1](#).

A. Overwegingen

De belangrijkste punten die in overwegingen zijn genomen bij het kiezen van de API architectuur waren:

- 1) **Te ontwikkelen zonder libraries** — Een van de vereisten van het project is dat de API server met zo min mogelijk libraries gemaakt moet worden.
- 2) **Toegankelijkheid van de API** — Het is essentieel dat de API eenvoudig te begrijpen en te gebruiken is. De API moest alle verzamelde data op een overzichtelijke manier kunnen presenteren.
- 3) **Onderhoudbaarheid en Uitbreidbaarheid:** De API moet eenvoudig te onderhouden zijn, met een heldere en overzichtelijke structuur. Daarnaast moest de architectuur flexibel genoeg zijn om toekomstige uitbreidingen en aanpassingen te ondersteunen zonder dat de codebase herschreven moet worden.

B. Keuze voor REST

Na het overwegen van de bovenstaande eisen en de doelen van het Robotica project, is er uiteindelijk voor gekozen om REST te gebruiken door de volgende punten:

- 1) **Eenvoudige implementatie** — REST maakt gebruik van standaard HTTP/1.1-methoden, waardoor het gemakkelijk te implementeren is zonder de noodzaak van complexe frameworks of libraries. Hierdoor kan de API ontwikkeld worden met alleen een ‘TcpListener’.
- 2) **Breed ondersteund en compatibel** — Door zijn leeftijd ondersteunt Vrijwel iedere webbrowser de infrastructuur waar REST op gebaseerd is, dit maakt het een ideale keuze voor toegankelijke webgebaseerde applicaties.
- 3) **Flexibiliteit en uitbreidbaarheid** — RESTful APIs kunnen eenvoudig worden uitgebreid met nieuwe functionaliteiten zonder bestaande endpoints te verstoren. Dit maakt het mogelijk om de API aan te passen aan eventuele nieuwe functies van het project.

C. Afweging tegen alternatieven

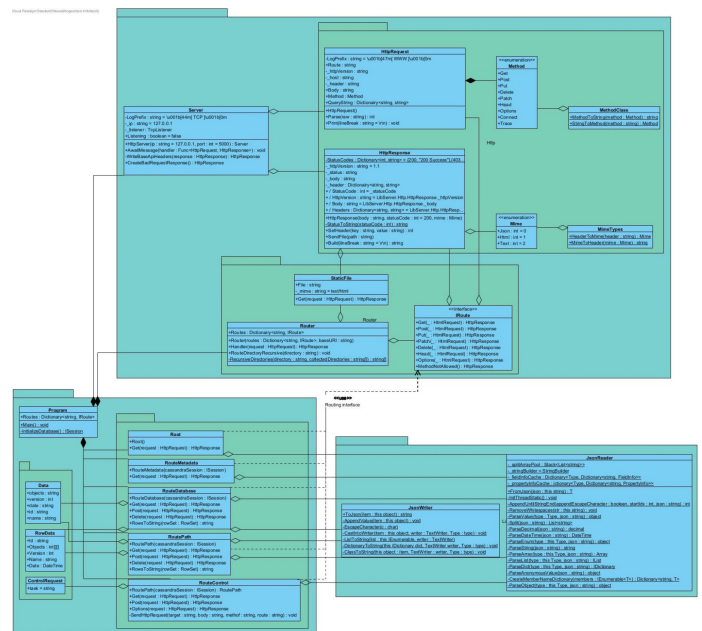
Hoewel de alternatieve architecturen ook voordelen bieden over REST, waren er enkele beperkingen die hen minder geschikt maakten voor dit project:

- 1) **SOAP** — Hoewel SOAP sterke beveiligingsfuncties biedt, werd het als te complex beschouwd voor de behoeften van dit project. De XML-gebaseerde berichten

van SOAP introduceren heel wat overhead waardoor er meer ruimte voor fouten is.

- 2) **GraphQL** — Hoewel GraphQL veel flexibeler en efficiënter data kan opvragen dan REST, wordt er bij dit project geen gebruik gemaakt van grote set key-value paren, waardoor GraphQL overbodig is. Ook maakt de complexiteit het moeilijk om GraphQL te implementeren zonder bestaande library.
- 3) **gRPC** — gRPC biedt hoge prestaties en real-time communicatie, maar de complexiteit net zoals GraphQL en de beperkingen tot het gebruik van nieuwere browsers maakten het minder geschikt voor dit project.

D. UML model voor de API Server



Figuur 1: Resulterende UML voor de API server.

ERKENNINGEN

De auteur wilt graag de volgende mensen bedanken voor hun contributie bij het schrijven van dit verslag:

Buurman, W. J.	Groepsgeenoot
Slikker, T.	Groepsgeenoot
Ottens, N.	Begeleiding
Tilman, K.	Begeleiding

REFERENCES

- [1] M. Massee, *REST API design rulebook designing consistent restful web service interfaces* mark massee. ed.: Simon St. Laurent. OReilly, 2011, geraadpleegd op Juni 8, 2024. [Online]. Available: <https://books.google.nl/books?id=eABpyTcJNIC&lpg=PR3&ots=vBPD3-lDKC&dq=restful%20api&lr&hl=nl&pg=PP1#v=onepage&q=restful%20api&f=false>
- [2] L. M. Afonso, R. F. de G. Cerqueira, and C. S. de Souza, "Evaluating application programming interfaces as communication artefacts," 2012, geraadpleegd op Juni 6, 2024. [Online]. Available: <http://www3.serg.inf.puc-rio.br/docs/MarquesPPIG2012.pdf>

- [3] M. Śliwa and B. Pańczyk, "Performance comparison of programming interfaces on the example of rest api, graphql and grpc," *Journal of Computer Sciences Institute*, vol. 21, Dec. 2021, geraadpleegd op Juni 10, 2024. [Online]. Available: <https://ph.pollub.pl/index.php/jcsi/article/view/2744>
- [4] W. W. W. C. W3C, "Soap version 1.2 part 1: Messaging framework second edition," geraadpleegd op Juni 15, 2024. [Online]. Available: <https://www.w3.org/TR/soap12-part1/>
- [5] Facebook, "GraphQL: A query language for your api," geraadpleegd op Juni 14, 2024. [Online]. Available: <https://graphql.org>
- [6] O. Hartig and J. Perez, "Semantics and complexity of graphql," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 1155–1164, geraadpleegd op Juni 12, 2024. [Online]. Available: <https://doi.org/10.1145/3178876.3186014>
- [7] Google, "grpc - a high-performance, open source universal rpc framework," geraadpleegd op Juni 16, 2024. [Online]. Available: <https://grpc.io/>